MATH 5330: Computational Methods of Linear Algebra

Lecture Note 6: Conjugate Gradient Method for General Systems

Xianyi Zeng
Department of Mathematical Sciences, UTEP

1 The Lanczos Method

We consider applying the Arnoldi's method to symmetric systems. In particular, we learned from previous lecture that:

$$V_m^t A V_m = H_m\,,$$

hence when A is symmetric, H_m is also symmetric or even better, tridiagonal. This gives rise to a simplified version of the Arnoldi's algorithm for symmetric systems:

```
$\boldsymbol{v}_1 = \boldsymbol{r}_0/||\boldsymbol{r}_0||\,,\ \beta_1 = 0\,,\ \boldsymbol{v}_0 = 0\,;$        (1.1)
for $j = 1,\cdots,m$ :
    $\boldsymbol{w}_j = A\boldsymbol{v}_j - \beta_j\boldsymbol{v}_{j-1}\,;$
    $\alpha_j = \boldsymbol{w}_j\cdot\boldsymbol{v}_j\,;$
    $\boldsymbol{w}_j = \boldsymbol{w}_j - \alpha_j\boldsymbol{v}_j\,;$
    $\beta_{j+1} = ||\boldsymbol{w}_j||\,;$
    if $\beta_{j+1} = 0$, stop;
    $\boldsymbol{v}_{j+1} = \boldsymbol{w}_j/h_{j+1,j}\,;$
end
```

or in matrix form in place of the Hessenberg matrix H_m, we have the tridiagonal matrix T_m:

$$T_m = \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_m \\ & & \beta_m & \alpha_m \end{bmatrix} \tag{1.2}$$

The tridiagonal structure of T_m is very much appreciated but it is only obtained when A is symmetric. For non-symmetric matrices, however, Lanczos proposed to build a pair of biorthogonal bases for two subspaces:

$$\mathcal{K}_m(A,\boldsymbol{v}_1) = \mathrm{span}(\boldsymbol{v}_1, A\boldsymbol{v}_1,\cdots,A^{m-1}\boldsymbol{v}_1)\,, \tag{1.3}$$

and

$$\mathcal{K}_m(A^t,\boldsymbol{v}_1) = \mathrm{span}(\boldsymbol{w}_1, A^t\boldsymbol{w}_1,\cdots,(A^t)^{m-1}\boldsymbol{w}_1)\,. \tag{1.4}$$

The target is to build two orthogonal bases $\{\boldsymbol{v}_1,\cdots,\boldsymbol{v}_m\}$ and $\{\boldsymbol{w}_1,\cdots,\boldsymbol{w}_m\}$ for the two subspaces, respectively, such that they satisfy the following biorthogonal condition:

$$\boldsymbol{v}_i\cdot\boldsymbol{w}_j = \delta_{ij}\,, \tag{1.5}$$

the Kronecker delta symbol. Note that we drop the requirement that $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$ are unit vectors. The corresponding modified algorithm is known as the *Lanczos biorthogonalization procedure*:

```
Choose $\boldsymbol{v}_1,\boldsymbol{w}_1 : \boldsymbol{v}_1\cdot\boldsymbol{w}_1 = 1$;                    (1.6)
$\beta_1 = \delta_1 = 0$, $\boldsymbol{v}_0 = \boldsymbol{w}_0$;
for $j = 1,\cdots,m$ :
    $\alpha_j = (A\boldsymbol{v}_j)\cdot\boldsymbol{w}_j$;
    $\hat{\boldsymbol{v}}_{j+1} = A\boldsymbol{v}_j - \alpha_j\boldsymbol{v}_j - \beta_j\boldsymbol{v}_{j-1}$;
    $\hat{\boldsymbol{w}}_{j+1} = A^t\boldsymbol{w}_j - \alpha_j\boldsymbol{w}_j - \delta_j\boldsymbol{w}_{j-1}$;
    $\delta_{j+1} = \sqrt{|\hat{\boldsymbol{v}}_{j+1}\cdot\hat{\boldsymbol{w}}_{j+1}|}$;
    if $\delta_{j+1} = 0$, stop;
    $\beta_{j+1} = \hat{\boldsymbol{v}}_{j+1}\cdot\hat{\boldsymbol{w}}_{j+1}/\delta_{j+1}$;
    $\boldsymbol{w}_{j+1} = \hat{\boldsymbol{w}}_{j+1}/\beta_{j+1}$;
    $\boldsymbol{v}_{j+1} = \hat{\boldsymbol{v}}_{j+1}/\delta_{j+1}$;
end
```

In this case, we define a tridiagonal matrix:

$$T_m = \begin{bmatrix} \alpha_1 & \beta_2 & & \\ \delta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_m \\ & & \delta_m & \alpha_m \end{bmatrix}, \tag{1.7}$$

and deduce the following matrix-relations of the preceding algorithm:

$$\boldsymbol{v}_j\cdot\boldsymbol{w}_i = \delta_{ij}, \tag{1.8a}$$

$$AV_m = V_mT_m + \delta_{m+1}\boldsymbol{v}_{m+1}\boldsymbol{e}_m^t, \tag{1.8b}$$

$$A^tW_m = W_mT_m^t + \beta_{m+1}\boldsymbol{w}_{m+1}\boldsymbol{e}_m^t, \tag{1.8c}$$

$$W_m^tAV_m = T_m. \tag{1.8d}$$

Despite the fact that the algorithm (1.6) leads to a smaller matrix T_m comparing to the full Hessenberg one H_m, it in effect solves two linear systems: one associated with A and the other one associated with A^t. Furthermore, we will need to store two sets of basis instead of one set in the original Arnoldi's algorithm. A more serious issue of the Lanczos algorithm for non-symmetric matrices is the potential risk of "serious breakdown".

To see this point, in the practical world the Arnoldi's method the algorithm stops only if $h_{j+1,j}\approx 0$, or when $\mathcal{K}_m$ is close to be invariant under the operation of A, which indicates that the affine space almost contains the true solution. For the Lanczos method (1.6), however, there exists "lucky breakdown" when $\hat{\boldsymbol{v}}_{j+1}\approx 0$ meaning that $\mathcal{K}_m$ is almost invariant under A; but there is also "serious breakdown" when $\hat{\boldsymbol{v}}_{j+1}\cdot\hat{\boldsymbol{w}}_{j+1}\approx 0$, in which case the algorithm may stop without finding a good affine space that is close to contain the true solution.

Despite these drawbacks, the Lanczos' method lays out a framework for designing other more robust algorithms, including a generalized conjugate gradient method for both symmetric and non-symmetric systems that is widely used in practice.

2 The Quasi-Minimal Residual Method

The first method we'll look at is called the quasi-minimal residual (QMR) method. It is very similar to the GMRES, except that we replace the Arnoldi's process by the Lanczos' method (1.6). Recall that H_m is only tridiagonal when A is symmetric, thus for general system $A\boldsymbol{x}=\boldsymbol{b}$, QMR in general does not give rise to an orthonormal basis. Let us denote:

$$\overline{T}_m = \begin{bmatrix} T_m \\ \delta_{m+1}\boldsymbol{e}_m^t \end{bmatrix} \tag{2.1}$$

similar as before (c.f. $\overline{H}_m$ for GMRES); then we have $AV_m = V_{m+1}\overline{T}_m$. For $\boldsymbol{x}\in\boldsymbol{x}_0+\mathcal{K}_m$ or equivalently $\boldsymbol{x}=\boldsymbol{x}_0+V_m\boldsymbol{y}$, $\boldsymbol{y}\in\mathbb{R}^m$:

$$\boldsymbol{b}-A\boldsymbol{x}=\boldsymbol{r}_0-AV_m\boldsymbol{y}=\beta\boldsymbol{v}_1-V_{m+1}\overline{T}_m\boldsymbol{y}=V_{m+1}(\beta\boldsymbol{e}_1-\overline{T}_m\boldsymbol{y})\,.$$

Following the philosophy of minimizing the residual as in GMRES, we'd like to search for $\boldsymbol{y}_m\in\mathbb{R}^m$ so that $||\boldsymbol{r}_0-AV_m\boldsymbol{y}||$ is minimized. In GMRES, this is equivalent to minimize $\left|\left|\beta\boldsymbol{e}_1-\overline{H}_m\boldsymbol{y}\right|\right|$; in QMR, we still choose $\boldsymbol{y}_m$ as:

$$\boldsymbol{y}_m=\arg\min_{\boldsymbol{y}\in\mathbb{R}^m}\left|\left|\beta\boldsymbol{e}_1-\overline{T}_m\boldsymbol{y}\right|\right|\,. \tag{2.2}$$

This least-squares problem is easier to solve than that of GMRES because $\overline{T}_m$ is tridiagonal; but this $\boldsymbol{y}_m$ in general does not minimize the residual since $V_{m+1}^tV_{m+1}$ usually is not the identity matrix.

The major advantage of QMR over GMRES is that the former is less storage demanding and the associated least-squares problem is much easier to solve. The disadvantage includes the possible stop of the Lanczos process due to serious breakdowns. However, if we know that the Lanczos algorithm does not break at the m-th iteration (i.e., we construct V_{m+1} successfully), the following estimate can be obtained:

$$\left|\left|\boldsymbol{r}_m^Q\right|\right|\le\kappa(V_{m+1})\left|\left|\boldsymbol{r}_m^G\right|\right|\,, \tag{2.3}$$

where $\boldsymbol{r}_m^Q$ and $\boldsymbol{r}_m^G$ are the residual vectors obtained from the QMR and GMRES, respectively; and $\kappa(V_{m+1})$ is the condition number w.r.t. L^2-norm of the matrix V_{m+1}. Hence providing that the Lanczos process does not break and that V_{m+1} is well conditioned, we expect QMR to converge. In practice, people modify the original Lanczos process to avoid "serious breakdown", such as restart the process whenever $||\hat{\boldsymbol{v}}_{j+1}||$ is not small but $\hat{\boldsymbol{v}}_{j+1}\cdot\hat{\boldsymbol{w}}_{j+1}$ is so.

3 The Conjugate Gradient Method for Non-Symmetric Systems

In order to relate the Lanczos algorithm to the conjugate gradient method, we first look at algorithm (1.1) and consider the LU decomposition $T_m=L_mU_m$, where L_m has unity diagonal elements. Recalling that in showing the convergence of the GMRES method (the exercise from last lecture), we constructed a sequence $\tilde{\boldsymbol{y}}_m$ that can be obtained by solving a linear system with Hessenberg matrix $\boldsymbol{H}_m$. This method is known as FOM (Full Orthogonalization Method). The counterpart of using the Lanczos algorithm for symmetric systems gives:

$$\tilde{\boldsymbol{x}}_m=\boldsymbol{x}_0+V_mT_m^{-1}(\beta\boldsymbol{e}_1)=\boldsymbol{x}_0+V_mU_m^{-1}V_m^{-1}(\beta\boldsymbol{e}_1)=\boldsymbol{x}_0+P_m\boldsymbol{z}_m\,, \tag{3.1}$$

where $P_m=V_mU_m^{-1}$ and $\boldsymbol{z}_m=V_m^{-1}(\beta\boldsymbol{e}_1)$.

The $n \times m$ matrix P_m has a very interesting property:

$$P_m^t A P_m = U_m^{-t} V_m^t A V_m U_m^{-1} = U_m^{-t} T_m U_m^{-1} = U_m^{-t} L_m \,,$$

which is lower tridiagonal as well as symmetric; hence $P_m^t A P_m$ is diagonal. That is, the column vectors of P_m are A-conjugate to each other.

Furthermore, U_{m+1} contains U_m as its upper-left block, the same relation holds for U_{m+1}^{-1} and U_m^{-1}. That is:

$$U_{m+1}^{-1} = \begin{bmatrix} U_m^{-1} & \boldsymbol{q} \\ 0 & \gamma \end{bmatrix}$$

and we have:

$$P_{m+1} = V_{m+1} U_{m+1}^{-1} = \begin{bmatrix} V_m & \boldsymbol{v}_{m+1} \end{bmatrix} \begin{bmatrix} U_m^{-1} & \boldsymbol{q} \\ 0 & \gamma \end{bmatrix} = \begin{bmatrix} P_m & \boldsymbol{p}_{m+1} \end{bmatrix} .$$

Thus the method is equivalent to gradually building the subspace $\mathcal{K}_m$ by finding the search directions $\boldsymbol{p}_m$ that are A-conjugate to each other. This is exactly the conjugate gradient method we've studied before!

Next we extend this idea to non-symmetric systems starting with the Lanczos algorithm (1.6). In this case, we still write $T_m = L_m U_m$ and compute $\boldsymbol{x}_m$ according to (3.1); however, the column vectors of P_m are no longer A-conjugate to each other. To this end, let $P'_m = W_m L_m^{-t}$:

$$(P'_m)^t A P_m = L_m^{-1} W_m^t A V_m U_m^{-1} = L_m^{-1} T_m U_m^{-1} = I \,, \tag{3.2}$$

thus the column vectors of P'_m and those of P_m are A-conjugate to each other:

$$(\boldsymbol{p}'_i)^t A \boldsymbol{p}_j = \delta_{ij} \,. \tag{3.3}$$

Similar as before, $\boldsymbol{p}'_i$ and $\boldsymbol{p}_i$ form the bases of $\mathcal{K}_m(A^t, \boldsymbol{w}_1)$ and $\mathcal{K}_m(A, \boldsymbol{v}_1)$ and they are built gradually. This leads to a CG-like algorithm for general systems:

```
$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ ,  Choose $\boldsymbol{r}'_0$ s.t. $\boldsymbol{r}_0 \cdot \boldsymbol{r}'_0 \neq 0$ ;        (3.4)
$\boldsymbol{p}_0 = \boldsymbol{r}_0$ , $\boldsymbol{p}'_0 = \boldsymbol{r}'_0$ ;
for $j = 0, 1, \cdots,$until converge :
    $\alpha_j = (\boldsymbol{r}_j \cdot \boldsymbol{r}'_j)/((A\boldsymbol{p}_j) \cdot \boldsymbol{p}'_j)$ ;
    $\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j \boldsymbol{p}_j$ ;
    $\boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j A \boldsymbol{p}_j$ ;
    $\boldsymbol{r}'_{j+1} = \boldsymbol{r}'_j - \alpha_j A^t \boldsymbol{p}'_j$ ;
    $\beta_j = (\boldsymbol{r}_{j+1} \cdot \boldsymbol{r}'_{j+1})/(\boldsymbol{r}_j \cdot \boldsymbol{r}'_j)$ ;
    $\boldsymbol{p}_{j+1} = \boldsymbol{r}_{j+1} + \beta_j \boldsymbol{p}_j$ ;
    $\boldsymbol{p}'_{j+1} = \boldsymbol{r}'_{j+1} + \beta_j \boldsymbol{p}'_j$ ;
end
```

Properties of the biconjugate gradient method include:

$$\boldsymbol{r}_j \cdot \boldsymbol{r}'_i = 0\,, \quad (A\boldsymbol{p}_j)\cdot \boldsymbol{p}'_i = 0\,, \quad \forall i \neq j\,; \tag{3.5a}$$

$$(A\boldsymbol{p}_j)\cdot \boldsymbol{r}'_i = (A\boldsymbol{p}_j)\cdot(\boldsymbol{p}'_i - \beta_{i-1}\boldsymbol{p}'_{i-1}) = 0\,, \quad \forall i \neq j, j+1\,; \tag{3.5b}$$

$$(A\boldsymbol{r}_j)\cdot \boldsymbol{p}'_i = \frac{1}{\alpha_i}\boldsymbol{r}_j \cdot (\boldsymbol{r}'_i - \boldsymbol{r}'_{i+1}) = 0\,, \quad \forall i \neq j, j-1\,, \tag{3.5c}$$

providing that the algorithm does not stop, i.e., $\alpha_j \neq 0$.

The algorithm (3.4) suffers from the same issue as the Lanczos biorthogonalization process before. First we deal with the need for the transpose of A, i.e., in the update $\boldsymbol{r}'_{j+1} = \boldsymbol{r}'_j - \alpha_j A^t \boldsymbol{p}'_j$. This is problematic because in Newton-based nonlinear solves, A usually represents the Jacobian matrix. In this case, a widely used methodology, so called the Jacobian-free method, approximates $A\boldsymbol{d}$ by:

$$A\boldsymbol{d} \approx \frac{1}{\epsilon}(\boldsymbol{f}(\boldsymbol{x}+\epsilon\boldsymbol{d}) - \boldsymbol{f}(\boldsymbol{x}))\,,$$

where ϵ is a small number and $A = \partial\boldsymbol{f}/\partial\boldsymbol{x}$. Within such a context, $A^t\boldsymbol{d}$ is not easy to evaluate.

Now we consider constructing a transpose-free version of the BCG algorithm. Particularly, we notice there exist polynomials of degree j, namely ϕ_j and π_j such that $\phi_j(0) = 1$ and:

$$\boldsymbol{r}_j = \phi_j(A)\boldsymbol{r}_0\,, \quad \boldsymbol{p}_j = \pi_j(A)\boldsymbol{r}_0\,;$$
$$\boldsymbol{r}'_j = \phi_j(A^t)\boldsymbol{r}'_0\,, \quad \boldsymbol{p}'_j = \pi_j(A^t)\boldsymbol{r}'_0\,.$$

To eliminate the usage of A^t, we would like to avoid computing $\boldsymbol{r}'_j$ explicitly. For example:

$$\alpha_j = \frac{\boldsymbol{r}_j\cdot\boldsymbol{r}'_j}{(A\boldsymbol{p}_j)\cdot\boldsymbol{p}'_j} = \frac{(\phi_j^2(A)\boldsymbol{r}_0)\cdot\boldsymbol{r}'_0}{(A\pi_j^2(A)\boldsymbol{r}_0)\cdot\boldsymbol{r}'_0}\,, \quad \beta_j = \frac{\boldsymbol{r}_{j+1}\cdot\boldsymbol{r}'_{j+1}}{\boldsymbol{r}_j\cdot\boldsymbol{r}'_j} = \frac{(\phi_{j+1}^2(A)\boldsymbol{r}_0)\cdot\boldsymbol{r}'_0}{(\phi_j^2(A)\boldsymbol{r}_0)\cdot\boldsymbol{r}'_0}\,,$$

where we used the fact that $(\phi_j(A))^t = \phi_j(A^t)$ and $(\pi_j(A))^t = \pi_j(A^t)$.

The foundation of the conjugate gradient squared (CGS) method is the recurrences for:

$$\begin{aligned}\tilde{\boldsymbol{r}}_j &= \phi_j^2(A)\boldsymbol{r}_0\,,\\ \tilde{\boldsymbol{p}}_j &= \pi_j^2(A)\boldsymbol{r}_0\,,\\ \tilde{\boldsymbol{q}}_j &= \phi_{j+1}(A)\pi_j(A)\boldsymbol{r}_0\,,\end{aligned}$$

instead of using $\boldsymbol{r}_j$, $\boldsymbol{p}_j$, $\boldsymbol{r}'_j$, and $\boldsymbol{p}'_j$. To derive a recursive formula for updating these vectors, we first note that:

$$\boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j A\boldsymbol{p}_j \quad \Rightarrow \quad \phi_{j+1}(A) = \phi_j(A) - \alpha_j A\pi_j(A)\,,$$

and:

$$\boldsymbol{p}_{j+1} = \boldsymbol{r}_{j+1} + \beta_j\boldsymbol{p}_j \quad \Rightarrow \quad \pi_{j+1}(A) = \phi_{j+1}(A) + \beta_j\pi_j(A)\,.$$

Hence:

$$\begin{aligned}\tilde{\boldsymbol{r}}_{j+1} &= \phi_{j+1}^2(A)\boldsymbol{r}_0 = \left[\phi_j^2(A) - 2\alpha_j A\phi_j(A)\pi_j(A) + \alpha_j^2A^2\pi_j^2(A)\right]^2\boldsymbol{r}_0\\ &= \tilde{\boldsymbol{r}}_j - \alpha_j A(2\tilde{\boldsymbol{r}}_j + 2\beta_{j-1}\tilde{\boldsymbol{q}}_{j-1} - \alpha_j A\tilde{\boldsymbol{p}}_j)\,,\\ \tilde{\boldsymbol{p}}_{j+1} &= \pi_{j+1}^2(A)\boldsymbol{r}_0 = [\phi_{j+1}(A) + \beta_j\pi_j(A)]^2\boldsymbol{r}_0 = \tilde{\boldsymbol{r}}_{j+1} + 2\beta_j\tilde{\boldsymbol{q}}_j + \beta_j^2\tilde{\boldsymbol{p}}_j\,,\\ \tilde{\boldsymbol{q}}_j &= \phi_{j+1}(A)\pi_j(A)\boldsymbol{r}_0 = [\phi_j(A) - \alpha_j A\pi_j(A)]\pi_j(A)\boldsymbol{r}_0\\ &= \tilde{\boldsymbol{r}}_j + \beta_{j-1}\tilde{\boldsymbol{q}}_{j-1} - \alpha_j A\tilde{\boldsymbol{p}}_j\,,\end{aligned}$$

which completes the loop.

The resulted algorithm reads:

```
$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$ ,  Choose $\boldsymbol{r}'_0$ s.t. $\boldsymbol{r}_0 \cdot \boldsymbol{r}'_0 \neq 0$ ;        (3.6)
$\boldsymbol{p}_0 = \boldsymbol{u}_0 = \boldsymbol{r}_0$ ;
for $j = 0, 1, \cdots$, until converge :
    $\alpha_j = (\boldsymbol{r}_j \cdot \boldsymbol{r}'_0)/((A\boldsymbol{p}_j) \cdot \boldsymbol{r}'_0)$ ;
    $\boldsymbol{q}_j = \boldsymbol{u}_j - \alpha_j A\boldsymbol{p}_j$ ;
    $\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j(\boldsymbol{u}_j + \boldsymbol{q}_j)$ ;
    $\boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j A(\boldsymbol{u}_j + \boldsymbol{q}_j)$ ;
    $\beta_j = (\boldsymbol{r}_{j+1} \cdot \boldsymbol{r}'_0)/(\boldsymbol{r}_j \cdot \boldsymbol{r}'_0)$ ;
    $\boldsymbol{u}_{j+1} = \boldsymbol{r}_{j+1} + \beta_j \boldsymbol{q}_j$ ;
    $\boldsymbol{p}_{j+1} = \boldsymbol{u}_{j+1} + \beta_j(\boldsymbol{q}_j + \beta_j \boldsymbol{p}_j)$ ;
end
```

Here we omitted the tilde for clarity – the vectors in (3.6) should not be confused with the same symbols in (3.4). Note that the iterate $\boldsymbol{x}_j$ is computed in a way such that it leads to the residual $\boldsymbol{r}_j = \phi_j^2(A)\boldsymbol{r}_0$, which in general does not coincide with an iterate of the original BCG method. However, we know that if BCG converges $\phi_m(A)\boldsymbol{r}_0 = 0$ for some $m > 0$, hence we also expect $\phi_m^2(A)\boldsymbol{r}_0 = 0$, i.e., the CGS also converges in m iterations. Furthermore, because $\left|\left|\phi_j^2(A)\boldsymbol{r}_0\right|\right| \leq ||\phi_j(A)||||\phi_j(A)\boldsymbol{r}_0||$, the CGS is expected to converge faster (usually twice as fast) than the BCG method. The important contribution of the CGS method is that it does not require any matrix-vector multiplication involving A^t; but two such operations with A are involved instead of one comparing to BCG.

One issue with the CGS method is that since the polynomials are squared comparing to the BCG method, it is more vulnerable to round-off errors. In the next lecture, we describe the last algorithm for linear systems in this class, namely the biconjugate gradient stabilized or the BICGSTAB algorithm. It is designed based on an idea similar to CGS that avoids matrix-vector multiplication with A^t as well as a procedure similar to the SOR method where an optimal relaxation parameter is computed for each iteration.