FEM and sparse linear system solving

Lecture 12, Dec 8, 2017: Nonsymmetric Lanczos methods

http://people.inf.ethz.ch/arbenz/FEM17

Peter Arbenz
Computer Science Department, ETH Zürich
E-mail: arbenz@inf.ethz.ch

Survey on lecture

- The finite element method
- Direct solvers for sparse systems
- Iterative solvers for sparse systems
  - Stationary iterative methods, preconditioning
  - Preconditioned conjugate gradient method (PCG)
  - Krylov space methods for nonsymmetric systems
    GMRES, MINRES
  - Preconditioning
  - Multigrid (preconditioning)
  - Nonsymmetric Lanczos iteration based methods
    Bi-CG, QMR, CGS, BiCGstab

Outline of this lecture

1. The non-symmetric Lanczos procedure and Bi-CG
2. The Quasi-Minimal Residual (QMR) algorithm
3. The Conjugate Gradient Squared (CGS) algorithm
4. The BiCGstab algorithm

Literature

- Y. Saad: *Iterative Methods for Sparse Linear Systems*, SIAM, 2nd edition, 2003.
- H. A. van der Vorst: *Iterative Krylov Methods for Large Linear Systems*, Cambridge University Press, 2003.

Krylov spaces

Definition

For given A, the m-th Krylov space generated by the vector $\boldsymbol{r}$ is given by

$$\mathcal{K}_m = \mathcal{K}_m(A, \boldsymbol{r}) := \text{span}\left\{\boldsymbol{r}, A\boldsymbol{r}, A^2\boldsymbol{r}, \ldots, A^{m-1}\boldsymbol{r}\right\}.$$

We can also write

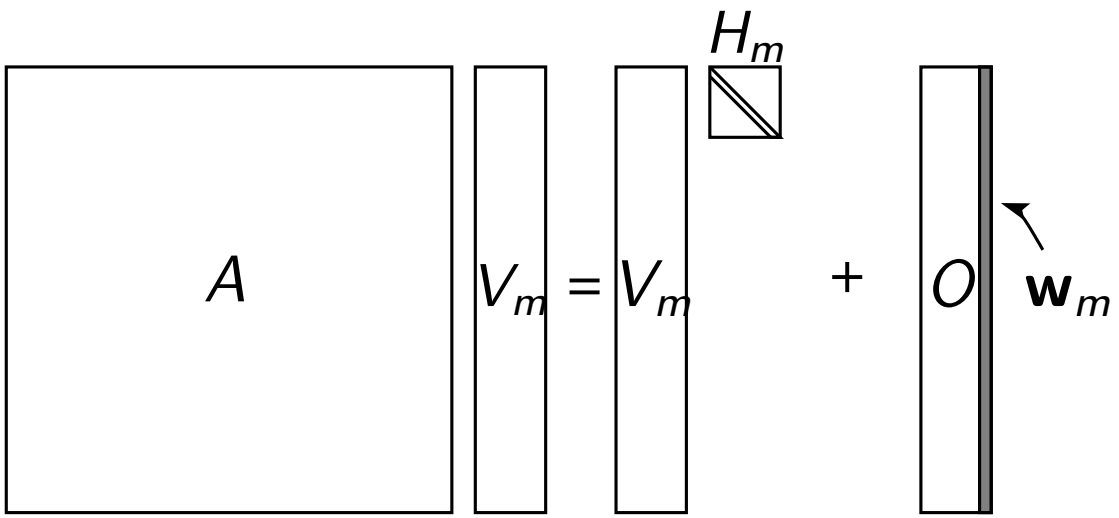
$$\mathcal{K}_m(A, \boldsymbol{r}) = \left\{p(A)\boldsymbol{r} \mid p \in \mathbb{P}_{m-1}\right\},$$

where $\mathbb{P}_d$ denotes the set of polynomials of degree at most d.

Arnoldi relation

Define the orthonormal basis $V_m := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m]$ of $\mathcal{K}_m(A, \boldsymbol{r}_0)$.
Then we get the Arnoldi relation

$$AV_m = V_m H_m + \boldsymbol{w}_m \boldsymbol{e}_m^T = V_{m+1} \bar{H}_m.$$



Arnoldi relation (cont.)

Here,

$$\bar{H}_m = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1,m} \\ h_{21} & h_{22} & \cdots & h_{2,m} \\ & h_{3,2} & \cdots & h_{3,m} \\ & & \ddots & \vdots \\ & & & h_{m+1,m} \end{bmatrix}$$

The square matrix $H_m \in \mathbb{R}^{m \times m}$ is obtained from $\bar{H}_m \in \mathbb{R}^{(m+1) \times m}$ by deleting the last row.
Notice that

$$H_m = V_m^T A V_m.$$

Therefore, if A is symmetric $\Rightarrow H_m \equiv T_m$ is tridiagonal!

Galerkin approach for nonsymmetric systems

- Most popular solver for nonsymmetric systems is GMRES.
- GMRES has big disadvantages: work for orthogonalization $\mathcal{O}(m^2 n)$ flops and memory consumption $\mathcal{O}(mn)$ bytes.
- This is due to the long recurrences. In order to compute a new Arnoldi vector $\boldsymbol{v}_{m+1}$ the vector $A\boldsymbol{v}_m$ must be *orthogonalized against all previous Arnoldi vectors* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$.
- $\Longrightarrow$ Restarting is necessary.
- In the symmetric case we have short recurrences as

$$(A\boldsymbol{v}_k)^T \boldsymbol{v}_j = \boldsymbol{v}_k^T A^T \boldsymbol{v}_j \overset{A=A^T}{=} \boldsymbol{v}_k^T \underbrace{(A\boldsymbol{v}_j)}_{\in \mathcal{K}_{j+1}} = 0 \qquad \text{if } j+1 < k.$$

- New idea: *generate test vectors from a different Krylov space!*

Petrov-Galerkin approach for nonsymmetric systems

Let (as earlier) $\boldsymbol{x}_m \in \boldsymbol{x}_0 + \mathcal{K}_m(A, \boldsymbol{r}_0)$. We can compute an *arbitrary* (in particular *nonorthogonal*) basis $\{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m\}$ for $\mathcal{K}_m(A, \boldsymbol{r}_0)$

$$AV_m = V_{m+1}\bar{H}_m, \qquad V_m = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m].$$

Let $W_m = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m]$ with $W_m^T V_m = \Delta_m$ diagonal. Then, we can determine $\boldsymbol{x}_m$ by the Petrov-Galerkin condition

$$\begin{aligned} \mathbf{0} &= W_m^T(\boldsymbol{b} - A\boldsymbol{x}_m) = W_m^T(\boldsymbol{r}_0 - AV_m\boldsymbol{y}_m) \\ &= W_m^T\boldsymbol{r}_0 - W_m^T V_m H_m \boldsymbol{y}_m \\ &= \beta\,\boldsymbol{e}_1 - \Delta_m H_m\,\boldsymbol{y}_m, \qquad\qquad \beta := (\boldsymbol{w}_1^T\boldsymbol{v}_1)\|\boldsymbol{r}_0\|_2. \end{aligned}$$

We try to arrange the $\boldsymbol{w}_m \in \mathcal{K}_m(A^T, \tilde{\boldsymbol{r}}_0)$ s.t. $\Delta_m H_m$ is tridiagonal.

Nonsymmetric Lanczos algorithm

Let $\boldsymbol{v}_1$, $\boldsymbol{w}_1$ be given with $\boldsymbol{w}_1^T \boldsymbol{v}_1 = d_1 \neq 0$. Typically, $\boldsymbol{v}_1 = \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$. Otherwise $\boldsymbol{w}_1$ is arbitrary. Sometimes, one has to solve a complementary system $A^T \tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$. Then, $\boldsymbol{w}_1$ should be set to $\tilde{\boldsymbol{r}}_0/\|\tilde{\boldsymbol{r}}_0\|$ where $\tilde{\boldsymbol{r}}_0 = \tilde{\boldsymbol{b}} - A^T \tilde{\boldsymbol{x}}_0$.

Now set

$$h_{21} \boldsymbol{v}_2 := A\boldsymbol{v}_1 - h_{11} \boldsymbol{v}_1.$$

We can choose h_{21} arbitrarily but non-zero, e.g., such that $\|\boldsymbol{v}_2\| = 1$, or such that $\boldsymbol{v}_2 = \boldsymbol{r}_1$.

To determine h_{11} we require that $\boldsymbol{w}_1^T \boldsymbol{v}_2 = 0$:

$$\boldsymbol{w}_1^T A\boldsymbol{v}_1 - h_{11} \boldsymbol{w}_1^T \boldsymbol{v}_1 = 0 \quad \underset{\boldsymbol{w}_1^T \boldsymbol{v}_1 \neq 0}{\Longrightarrow} \quad h_{11} = \frac{\boldsymbol{w}_1^T A\boldsymbol{v}_1}{\boldsymbol{w}_1^T \boldsymbol{v}_1}.$$

Nonsymmetric Lanczos algorithm (cont.)

Let us define

$$h_{21}\boldsymbol{w}_2 := A^T\boldsymbol{w}_1 - h_{11}\boldsymbol{w}_1.$$

Note that this is the 'same' formula as with the definition of $\boldsymbol{v}_2$!

The construction was such that $\boldsymbol{w}_1^T\boldsymbol{v}_2 = 0$. Now we *show* that $\boldsymbol{v}_1^T\boldsymbol{w}_2 = 0$. Indeed,

$$\begin{aligned} h_{21}\boldsymbol{v}_1^T\boldsymbol{w}_2 &= \boldsymbol{v}_1^T A^T\boldsymbol{w}_1 - h_{11}\boldsymbol{v}_1^T\boldsymbol{w}_1 \\ &= (A\boldsymbol{v}_1 - h_{11}\boldsymbol{v}_1)^T\boldsymbol{w}_1 = h_{21}\boldsymbol{v}_2^T\boldsymbol{w}_1 = 0 \end{aligned}$$

as $\boldsymbol{v}_2^T\boldsymbol{w}_1 = \boldsymbol{w}_1^T\boldsymbol{v}_2$.

(Note that we stick with real arithmetic. Complex arithmetic would be slightly different.)

Nonsymmetric Lanczos algorithm (cont.)

In general, let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ be a basis of $\mathcal{K}_k(A, \boldsymbol{r}_0)$ and $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k$ be a basis of $\mathcal{K}_k(A^T, \tilde{\boldsymbol{r}}_0)$ satisfying the biorthogonality relations

$$\boldsymbol{v}_j^T \boldsymbol{w}_i = \begin{cases} d_j \neq 0, & i = j, \\ 0, & i \neq j. \end{cases}$$

Let us set

$$h_{k+1,k} \boldsymbol{v}_{k+1} = A\boldsymbol{v}_k - \sum_{i=1}^{k} h_{ik} \boldsymbol{v}_i$$

($h_{k+1,k}$ arbitrary but non-zero) and determine the coefficients h_{ik} such that $\boldsymbol{w}_j^T \boldsymbol{v}_{k+1} = 0$, $j = 1, \ldots, k$. Then,

$$0 = \boldsymbol{w}_j^T A\boldsymbol{v}_k - \sum_{i=1}^{k} h_{ik} \boldsymbol{w}_j^T \boldsymbol{v}_i = \boldsymbol{w}_j^T A\boldsymbol{v}_k - h_{jk} \boldsymbol{w}_j^T \boldsymbol{v}_j \implies h_{jk} = \frac{\boldsymbol{w}_j^T A\boldsymbol{v}_k}{\boldsymbol{w}_j^T \boldsymbol{v}_j}.$$

Nonsymmetric Lanczos algorithm (cont.)

We now define

$$h_{k+1,k}\,\boldsymbol{w}_{k+1} := A^T\boldsymbol{w}_k - \sum_{i=1}^{k} h_{ik}\,\boldsymbol{w}_i$$

and show that $\boldsymbol{v}_j^T\boldsymbol{w}_{k+1} = 0$, $j = 1,\ldots,k$. In fact, for $j < k$,

$$\begin{aligned}
\boldsymbol{v}_j^T A^T\boldsymbol{w}_k - \sum_{i=1}^{k} h_{ik}\,\boldsymbol{v}_j^T\boldsymbol{w}_i &= \boldsymbol{v}_j^T A^T\boldsymbol{w}_k - h_{jk}\,\boldsymbol{v}_j^T\boldsymbol{w}_j = \boldsymbol{w}_k^T(A\boldsymbol{v}_j) - h_{jk}\,\boldsymbol{w}_j^T\boldsymbol{v}_j \\
&= \boldsymbol{w}_k^T \sum_{i=1}^{j+1} h_{ij}\,\boldsymbol{v}_i - h_{jk}\,\boldsymbol{w}_j^T\boldsymbol{v}_j = h_{kj}\,\boldsymbol{w}_k^T\boldsymbol{v}_k - h_{jk}\,\boldsymbol{w}_j^T\boldsymbol{v}_j \\
&= \sum_{i=1}^{j+1} h_{ij}\,\boldsymbol{w}_i^T\boldsymbol{v}_k - h_{jk}\,\boldsymbol{w}_j^T\boldsymbol{v}_j \\
&= (A^T\boldsymbol{w}_j)^T\boldsymbol{v}_k - h_{jk}\,\boldsymbol{w}_j^T\boldsymbol{v}_j = \boldsymbol{w}_j^T A\boldsymbol{v}_k - h_{jk}\,\boldsymbol{w}_j^T\boldsymbol{v}_j = 0
\end{aligned}$$

For $j = k$ the statement is 'trivial'.

Nonsymmetric Lanczos algorithm (cont.)

Notice that $A^T \boldsymbol{w}_j \in \mathcal{K}_{j+1}(A^T, \boldsymbol{w}_1)$. Thus, for $j+1 < k$,

$$(A^T \boldsymbol{w}_j)^T \boldsymbol{v}_k = \boldsymbol{w}_j^T A \boldsymbol{v}_k = 0 \quad \Longrightarrow \quad h_{jk} = 0.$$

Therefore,

$$A\boldsymbol{v}_k = \sum_{i=1}^{k+1} h_{ik} \boldsymbol{v}_i = h_{k+1,k} \boldsymbol{v}_{k+1} + h_{k,k} \boldsymbol{v}_k + h_{k-1,k} \boldsymbol{v}_{k-1}$$

$$\equiv t_{k+1,k} \boldsymbol{v}_{k+1} + t_{k,k} \boldsymbol{v}_k + t_{k-1,k} \boldsymbol{v}_{k-1}.$$

So, there are only three nontrivial terms in the expression.

Nonsymmetric Lanczos algorithm (cont.)

Finally, we arrive at

$$AV_m = V_m T_m + t_{m+1,m} \boldsymbol{v}_{m+1} \boldsymbol{e}_m^T,$$
$$A^T W_m = W_m T_m + t_{m+1,m} \boldsymbol{w}_{m+1} \boldsymbol{e}_m^T,$$

with

$$T_m = \begin{bmatrix} t_{11} & t_{12} & & & & \\ t_{21} & t_{22} & t_{23} & & & \\ & t_{32} & t_{33} & \ddots & & \\ & & \ddots & \ddots & t_{m-1,m} & \\ & & & t_{m,m-1} & t_{mm} \end{bmatrix}.$$

Nonsymmetric Lanczos algorithm (cont.)

$$W_m^T A V_m = W_m^T V_m T_m \equiv \Delta_m T_m, \qquad \text{with } \Delta_m = W_m^T V_m = V_m^T W_m$$
$$V_m^T A^T W_m = V_m^T W_m T_m = \Delta_m T_m,$$

(From this we see that $\Delta_m T_m$ is *symmetric tridiagonal*.)

Notice that this algorithm may break down!!

1. We may have $\boldsymbol{w}_j^T \boldsymbol{v}_j = 0$. This is called a serious breakdown. This quantity appears in the denominator of the formula that define α_j and β_{j+1}.
2. The matrix T_m is nonsingular but there is no LU factorization.
3. One of the vectors $\boldsymbol{v}_{j+1}$ or $\boldsymbol{w}_{j+1}$ vanish. This means that we have found an invariant subspace for A or A^T. We also have found a solution for $A\boldsymbol{x} = \boldsymbol{b}$ or for $A^T \tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}$.

The Biconjugate Gradient (BiCG) algorithm

We proceed similar as with CG. Let

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m \boldsymbol{y}_m, \qquad \boldsymbol{r}_m = \boldsymbol{b} - A\boldsymbol{x}_m.$$

We determine $\boldsymbol{y}_m$ by

$$\begin{aligned} \mathbf{0} &= W_m^T \boldsymbol{r}_m = W_m^T(\boldsymbol{r}_0 - AV_m\boldsymbol{y}_m) \\ &= W_m^T \boldsymbol{r}_0 - W_m^T V_m T_m \boldsymbol{y}_m \\ &= \beta\, \boldsymbol{e}_1 - \Delta_m T_m\, \boldsymbol{y}_m, \qquad\qquad \beta := \boldsymbol{w}_1^T \boldsymbol{r}_0. \end{aligned}$$

We can decompose $\Delta_m T_m = L_m D_m L_m^T$ where D_m is diagonal. We do not allow pivoting here and hope for the best!

The Biconjugate Gradient (BiCG) algorithm (cont.)

The equation $\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m \boldsymbol{y}_m$ implies

$$\boldsymbol{r}_m = \boldsymbol{b} - A\boldsymbol{x}_m = \boldsymbol{r}_0 - AV_m \boldsymbol{y}_m \in \mathcal{K}_{m+1}(A, \boldsymbol{r}_0).$$

Together with the Galerkin condition we have that

$$\boldsymbol{r}_m \in \mathcal{K}_{m+1}(A, \boldsymbol{r}_0) \ominus \mathcal{K}_m(A^T, \tilde{\boldsymbol{r}}_0).$$

Therefore, the $\boldsymbol{r}_j$ are aligned with the 'right' Lanczos vectors $\boldsymbol{v}_{j+1}$ (and $\tilde{\boldsymbol{r}}_j$ is aligned with $\boldsymbol{w}_{j+1}$). So, we can choose the Lanczos vectors to be the residuals:

$$\boldsymbol{v}_k = \boldsymbol{r}_{k-1},\ \boldsymbol{w}_k = \tilde{\boldsymbol{r}}_{k-1} \quad \Longrightarrow \quad (\Delta_m)_{kk} = \tilde{\boldsymbol{r}}_{k-1}^T \boldsymbol{r}_{k-1},$$

$$(T_m)_{k,j} = \frac{\tilde{\boldsymbol{r}}_{k-1}^T A\, \boldsymbol{r}_{j-1}}{\tilde{\boldsymbol{r}}_{k-1}^T \boldsymbol{r}_{k-1}}$$

The Biconjugate Gradient (BiCG) algorithm (cont.)

Since T_m is nonsingular we can write

$$\begin{aligned} \boldsymbol{x}_m &= \boldsymbol{x}_0 + V_m \boldsymbol{y}_m = \boldsymbol{x}_0 + \beta V_m T_m^{-1} \boldsymbol{e}_1 \\ &= \boldsymbol{x}_0 + \underbrace{(V_m L_m^{-T})}_{P_m} \underbrace{(D_m^{-1} L_m^{-1} \boldsymbol{e}_1 \beta)}_{\boldsymbol{a}_m} = \boldsymbol{x}_0 + P_m \boldsymbol{a}_m. \end{aligned}$$

Let $\tilde{P}_m = W_m L_m^{-T}$. Then,

$$\tilde{P}_m^T A P_m = L_m^{-1} W_m^T A V_m L_m^{-T} = L_m^{-1} \Delta_m T_m L_m^{-T} = D_m.$$

We now investigate the two equations

$$V_m = P_m L_m^T, \qquad L_m D_m \boldsymbol{a}_m = \beta \boldsymbol{e}_1,$$

to establish the short recurrences of BiCG.

The Biconjugate Gradient (BiCG) algorithm (cont.)

From

$$[\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m] = [\boldsymbol{p}_0, \ldots, \boldsymbol{p}_{m-1}] \begin{bmatrix} 1 & -\beta_1 & & & \\ & 1 & -\beta_2 & & \\ & & 1 & \ddots & \\ & & & \ddots & -\beta_{m-1} \\ & & & & 1 \end{bmatrix}$$

we get

$$\begin{cases} \boldsymbol{p}_0 = \boldsymbol{v}_1, \\ \boldsymbol{p}_m = \boldsymbol{v}_{m+1} + \beta_m \boldsymbol{p}_{m-1}, \quad m > 0. \end{cases}$$

The $\boldsymbol{p}$ vectors correspond to the search directions in CG.
Note that the $\tilde{\boldsymbol{p}}$ vectors satisfy similar recurrence relations,

$$\tilde{\boldsymbol{p}}_m = \boldsymbol{w}_{m+1} + \beta_m \tilde{\boldsymbol{p}}_{m-1}.$$

The Biconjugate Gradient (BiCG) algorithm (cont.)

We now set $\boldsymbol{v}_1 = \boldsymbol{r}_0$ and $\boldsymbol{v}_{m+1} = \boldsymbol{r}_m$.

Then the A-orthogonality of $\boldsymbol{p}_m$ and $\tilde{\boldsymbol{p}}_{m-1}$ yields

$$0 = \tilde{\boldsymbol{p}}_{m-1}^T A \boldsymbol{p}_m = \tilde{\boldsymbol{p}}_{m-1}^T A \boldsymbol{r}_m + \beta_m \tilde{\boldsymbol{p}}_{m-1}^T A \boldsymbol{p}_{m-1}.$$

Thus,

$$\beta_m = -\frac{\tilde{\boldsymbol{p}}_{m-1}^T A \boldsymbol{r}_m}{\tilde{\boldsymbol{p}}_{m-1}^T A \boldsymbol{p}_{m-1}}. \tag{1}$$

We will later need that

$$\tilde{\boldsymbol{r}}_m^T \boldsymbol{p}_m = \tilde{\boldsymbol{r}}_m^T \boldsymbol{r}_m + \beta_m \tilde{\boldsymbol{r}}_m^T \boldsymbol{p}_{m-1} = \tilde{\boldsymbol{r}}_m^T \boldsymbol{r}_m, \tag{2}$$

which is true because $\boldsymbol{p}_{m-1} \in \mathcal{K}_m(A, \boldsymbol{r}_0)$.

The Biconjugate Gradient (BiCG) algorithm (cont.)

From

$$(L_m D_m)\boldsymbol{a}_m = \begin{bmatrix} \delta_1 & & & & \\ -\beta_1\delta_1 & \delta_2 & & & \\ & -\beta_2\delta_2 & \delta_3 & & \\ & & \ddots & \ddots & \\ & & & -\beta_m\delta_m & \delta_{m+1} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \vdots \\ \alpha_m \end{bmatrix} = \begin{bmatrix} \beta \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

we get

$$\begin{cases} \alpha_0 = \beta/\delta_1, \\ \alpha_k = \dfrac{\beta_k\delta_k}{\delta_{k-1}}\,\alpha_{k-1}, \quad k > 1. \end{cases}$$

So, the elements of $\boldsymbol{a}_m$ can be computed in a recursive manner, i.e., $\boldsymbol{a}_m = [\boldsymbol{a}_{m-1}^T, \alpha_m]^T$.

The Biconjugate Gradient (BiCG) algorithm (cont.)

From this we have

$$\begin{aligned} \boldsymbol{x}_m &= \boldsymbol{x}_0 + P_m\, \boldsymbol{a}_m = \boldsymbol{x}_0 + P_{m-1}\, \boldsymbol{a}_{m-1} + \boldsymbol{p}_{m-1}\alpha_m \\ &= \boldsymbol{x}_{m-1} + \alpha_{m-1}\, \boldsymbol{p}_{m-1}, \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{r}_m &= \boldsymbol{r}_{m-1} - \alpha_{m-1}\, A\, \boldsymbol{p}_{m-1}, \\ \tilde{\boldsymbol{r}}_m &= \tilde{\boldsymbol{r}}_{m-1} - \alpha_{m-1}\, A^T\, \tilde{\boldsymbol{p}}_{m-1}. \end{aligned} \tag{3}$$

Multiplying the first equation in (3) by $\tilde{\boldsymbol{p}}_{m-1}^T$ we get

$$\alpha_m = \frac{\tilde{\boldsymbol{p}}_m^T \boldsymbol{r}_m}{\tilde{\boldsymbol{p}}_m^T A\, \boldsymbol{p}_m} \overset{(2)}{=} \frac{\tilde{\boldsymbol{r}}_m^T \boldsymbol{r}_m}{\tilde{\boldsymbol{p}}_m^T A\, \boldsymbol{p}_m}, \qquad m = 0, 1, \ldots \tag{4}$$

(We have incremented the index $m-1$ by 1.)

The Biconjugate Gradient (BiCG) algorithm (cont.)

Multiplying the second equation in (3) by $\boldsymbol{r}_m^T$ we get

$$\boldsymbol{r}_m^T \tilde{\boldsymbol{r}}_m = -\alpha_{m-1}\, \boldsymbol{r}_m^T A^T\, \tilde{\boldsymbol{p}}_{m-1}.$$

Using this and (4) we can beautify formula (1) in

$$\beta_m = -\frac{\tilde{\boldsymbol{p}}_{m-1}^T A \boldsymbol{r}_m}{\tilde{\boldsymbol{p}}_{m-1}^T A \boldsymbol{p}_{m-1}} = \frac{1}{\alpha_{m-1}} \frac{\tilde{\boldsymbol{r}}_m^T \boldsymbol{r}_m}{\tilde{\boldsymbol{p}}_{m-1}^T A \boldsymbol{p}_{m-1}} \overset{(4)}{=} \frac{\tilde{\boldsymbol{r}}_m^T \boldsymbol{r}_m}{\tilde{\boldsymbol{r}}_{m-1}^T \boldsymbol{r}_{m-1}} \equiv \frac{\rho_m}{\rho_{m-1}}.$$

Notice that the derivations of these formulae are very similar to those of the conjugate gradient (CG) algorithm.

The complete BiCG algorithm is given on the next slide.

The BiCG algorithm

1: Choose $\boldsymbol{x}_0$. Set $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$. Choose $\tilde{\boldsymbol{r}}_0$ s.t. $\rho_0 = \tilde{\boldsymbol{r}}_0^T \boldsymbol{r}_0 \neq 0$.
2: Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$, $\tilde{\boldsymbol{p}}_0 = \tilde{\boldsymbol{r}}_0$, $\boldsymbol{q}_0 = A\boldsymbol{p}_0$, $\tilde{\boldsymbol{q}}_0 = A^T \tilde{\boldsymbol{p}}_0$.
3: **for** $k = 0, 1, \ldots$ **do**
4: $\quad \alpha_k = (\boldsymbol{r}_k^T \tilde{\boldsymbol{r}}_k)/(\boldsymbol{q}_k^T \tilde{\boldsymbol{p}}_k)$.
5: $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$.
6: $\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{q}_k, \quad \tilde{\boldsymbol{r}}_{k+1} = \tilde{\boldsymbol{r}}_k - \alpha_k \tilde{\boldsymbol{q}}_k$.
7: $\quad$ Test for convergence.
8: $\quad \rho_{k+1} = \boldsymbol{r}_{k+1}^T \tilde{\boldsymbol{r}}_{k+1}$.
9: $\quad$ If $\rho_{k+1} = 0$ method fails.
10: $\quad \beta_{k+1} = \rho_{k+1}/\rho_k$.
11: $\quad \boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1} \boldsymbol{p}_k, \quad \boldsymbol{q}_{k+1} = A\boldsymbol{p}_{k+1}$.
12: $\quad \tilde{\boldsymbol{p}}_{k+1} = \tilde{\boldsymbol{r}}_{k+1} + \beta_{k+1} \tilde{\boldsymbol{p}}_k, \quad \tilde{\boldsymbol{r}}_{k+1} = A^T \tilde{\boldsymbol{p}}_{k+1}$.
13: **end for**

The BiCG algorithm (cont.)

- The big advantage of the nonsymmetric Lanczos/BiCG algorithm over Arnoldi/GMRES algorithm is its small memory requirements. Only 7 vectors need to be stored.

  This is particularly important in the presence of slow Arnoldi/GMRES convergence.
- The big disadvantage of the nonsymmetric Lanczos/BiCG algorithm is the possibility of breakdowns.
- Nonsymmetric Lanczos/BiCG also requires multiplications with A^T.

The QMR algorithm

The BiCG algorithm can lead to very erratic behavior, in particular, if T_m is ill-conditioned. The QMR algorithm leads to much smoother residual norms $\|\boldsymbol{r}_j\|$. Starting from the extended Lanczos relation we write

$$\begin{aligned}\boldsymbol{r}_m &= \boldsymbol{r}_0 - AV_m\boldsymbol{y}_m \\ &= \beta\,\boldsymbol{v}_1 - V_{m+1}\overline{T}_m\boldsymbol{y}_m \\ &= V_{m+1}(\beta\,\boldsymbol{e}_1 - \overline{T}_m\boldsymbol{y}_m).\end{aligned}$$

Clearly, $\|\boldsymbol{r}_m\| = \|V_{m+1}(\beta\,\boldsymbol{e}_1 - \overline{T}_m\boldsymbol{y}_m)\|$.

Minimizing this w.r.t. $\boldsymbol{y}_m$ yields the GMRES solution.
But, here, V_{m+1} is *not* orthogonal. So, minimizing $\|\beta\,\boldsymbol{e}_1 - \overline{T}_m\boldsymbol{y}_m\|$ does not yield the GMRES solution.

The QMR algorithm (cont.)

It turns out that minimizing

$$J(\boldsymbol{y}) \equiv \|\beta\, \boldsymbol{e}_1 - \overline{T}_m \boldsymbol{y}_m\|$$

is a reasonable idea. The resulting approximate solution

$$\boldsymbol{x}_m = \boldsymbol{x}_0 + V_m \boldsymbol{y}_m = \boldsymbol{x}_0 + \beta V_m (\overline{T}_m)^+ \boldsymbol{e}_1.$$

is called the Quasi-Minimal Residual (QMR) approximation.

By construction,

$$\|\boldsymbol{r}_m^{\mathrm{GMRES}}\| \leq \|\boldsymbol{r}_m^{\mathrm{QMR}}\|.$$

The QMR algorithm (cont.)

Let $\|\boldsymbol{r}_m^Q\| = \|\beta\,\boldsymbol{e}_1 - \overline{T}_m \boldsymbol{y}_m\|$ and let $\|\boldsymbol{v}_i\| = 1$ for all i. Then,

$$\|\boldsymbol{r}_m^{\mathrm{QMR}}\| = \|V_{m+1}\underbrace{(\beta\,\boldsymbol{e}_1 - \overline{T}_m \boldsymbol{y}_m)}_{\boldsymbol{z}}\|$$

$$= \left\|\sum_{i=1}^{m+1} \boldsymbol{v}_i z_i\right\| \leq \left(\sum_{i=1}^{m+1} \|\boldsymbol{v}_i\|^2\right)^{1/2} \left(\sum_{i=1}^{m+1} |z_i|^2\right)^{1/2} = \sqrt{m+1}\|\boldsymbol{r}_m^Q\|.$$

One furthermore can observe that after steps of strong decrease of $\|\boldsymbol{r}_k^Q\|$ the norms of the BiCG and the QMR residuals are close.

The Conjugate Gradient Squared algorithm

The *conjugate gradient squared* (CGS) algorithm was developed mainly to avoid using the transpose of A in BiCG. We write the residual and conjugate direction in BiCG in polynomial form

$$\boldsymbol{r}_j = \phi_j(A)\boldsymbol{r}_0, \qquad \boldsymbol{p}_j = \pi_j(A)\boldsymbol{r}_0.$$

Then, by construction,

$$\tilde{\boldsymbol{r}}_j = \phi_j(A^T)\tilde{\boldsymbol{r}}_0, \qquad \tilde{\boldsymbol{p}}_j = \pi_j(A^T)\tilde{\boldsymbol{r}}_0,$$

such that

$$\alpha_j = \frac{\boldsymbol{r}_j^T \tilde{\boldsymbol{r}}_j}{\boldsymbol{q}_j^T \tilde{\boldsymbol{p}}_j} = \frac{(\phi_j(A)\boldsymbol{r}_0)^T(\phi_j(A^T)\tilde{\boldsymbol{r}}_0)}{(A\pi_j(A)\boldsymbol{r}_0)^T(\pi_j(A^T)\tilde{\boldsymbol{r}}_0)} = \frac{(\phi_j^2(A)\boldsymbol{r}_0)^T \boldsymbol{r}_0}{(A\pi_j^2(A)\boldsymbol{r}_0)^T \tilde{\boldsymbol{r}}_0}.$$

If we can get recursions for the vectors $\phi_j^2(A)\boldsymbol{r}_0$ and $\pi_j^2(A)\boldsymbol{r}_0$ then we can compute α_j and, similarly, β_j, using these new recursions.

The Conjugate Gradient Squared algorithm (cont.)

We have from the recurrences of the $\boldsymbol{r}_j$ and $\boldsymbol{p}_j$

$$\phi_{j+1}(t) = \phi_j(t) - \alpha_j t \pi_j(t), \qquad \pi_{j+1}(t) = \phi_{j+1}(t) + \beta_j \pi_{j-1}(t).$$

After squaring

$$\phi^2_{j+1}(t) = \cdots \qquad \pi^2_{j+1}(t) = \cdots$$

and some cumbersome algebraic manipulation one finds that vectors

$$\boldsymbol{r}_j = \phi^2_j(A)\boldsymbol{r}_0, \;\; \boldsymbol{p}_j = \pi^2_j(A)\boldsymbol{r}_0, \;\; \boldsymbol{u}_j = \phi_{j-1}(A)\pi_j(A)\boldsymbol{r}_0, \;\; \boldsymbol{q}_j = \phi_j(A)\pi_j(A)\boldsymbol{r}_0,$$

are needed in a potential algorithm. They satisfy the recurrences

$$\boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j A(\boldsymbol{u}_j + \boldsymbol{q}_j), \qquad \boldsymbol{p}_{j+1} = \boldsymbol{u}_{j+1} + \beta_j(\boldsymbol{q}_j + \beta_j \boldsymbol{p}_j)$$
$$\boldsymbol{u}_{j+1} = \boldsymbol{r}_j + \beta_j \boldsymbol{q}_j, \qquad \boldsymbol{q}_{j+1} = \boldsymbol{u}_{j+1} - \alpha_{j+1} A \boldsymbol{p}_{j+1}$$

The CGS algorithm

1: Choose $\boldsymbol{x}_0$. Set $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$. Choose $\tilde{\boldsymbol{r}}_0$ s.t. $\rho_0 = \tilde{\boldsymbol{r}}_0^T \boldsymbol{r}_0 \neq 0$.
2: Set $\boldsymbol{p}_0 := \boldsymbol{u}_0 := \boldsymbol{r}_0$.
3: **for** $k = 0, 1, \ldots$ **do**
4: $\quad \alpha_k = (\boldsymbol{r}_k^T \tilde{\boldsymbol{r}}_0)/((A\boldsymbol{p}_k)^T \tilde{\boldsymbol{p}}_0)$.
5: $\quad \boldsymbol{q}_k = \boldsymbol{u}_k - \alpha_k A\boldsymbol{p}_k$.
6: $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k(\boldsymbol{u}_k + \boldsymbol{q}_k), \quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A(\boldsymbol{u}_k + \boldsymbol{q}_k)$.
7: $\quad$ Test for convergence.
8: $\quad \rho_{k+1} = \boldsymbol{r}_{k+1}^T \tilde{\boldsymbol{r}}_0$.
9: $\quad$ If $\rho_{k+1} = 0$ method fails.
10: $\quad \beta_{k+1} = \rho_{k+1}/\rho_k$.
11: $\quad \boldsymbol{u}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{q}_k, \quad \boldsymbol{p}_{k+1} = \boldsymbol{u}_{k+1} + \beta_k(\boldsymbol{q}_k + \beta_k \boldsymbol{p}_k)$.
12: **end for**

The BiCGstab algorithm

- The CGS algorithm squares the residual polynomials and replaces multiplications with A^T by multiplications with A. So, each iteration step requires two multiplications with A.
- The algorithm still leads to the same irregular convergence as BiCG.
- To improve this situation the stabilized biconjugate gradient (BiCGstab) algorithm was designed. Notice that the left Krylov space does not show up in the previous algorithms. We chose some initial vector $\tilde{\boldsymbol{r}}_0$. But $\mathcal{K}_j(A^T, \tilde{\boldsymbol{r}}_0)$ is present only implicitly. The tridiagonal matrix T_m reflects the way the left basis vectors $\boldsymbol{w}_j$ are constructed.
- In BiCGstab the basis of the left Krylov space is chosen differently.

The BiCGstab algorithm (cont.)

- In BiCGstab the left residuals are not generated by the same polynomials as the right residuals but by the very simple rule

$$\tilde{\boldsymbol{r}}_{j+1} = (I - \omega_j A)\,\tilde{\boldsymbol{r}}_j = \psi_{j+1}(A)\tilde{\boldsymbol{r}}_0$$

which leads to

$$\psi_{j+1}(t) = (1 - \omega_j t)\,\psi_j(t).$$

As with CGS, vectors have to be found that make possible the computation of

$$\boldsymbol{r}_j = \psi_j(A)\phi_j(A)\boldsymbol{r}_0.$$

This can be done similarly as with CGS, see the books of Saad or van der Vorst.

The BiCGstab algorithm (cont.)

- How is the parameter ω_j chosen?
  We have

$$\boldsymbol{r}_j = \psi_j(A)\phi_j(A)\boldsymbol{r}_0 = (I - \omega_j A)\underbrace{\psi_{j-1}(A)\phi_j(A)\boldsymbol{r}_0}_{\boldsymbol{s}_j} \equiv (I - \omega_j A)\boldsymbol{s}_j.$$

  ω_j is chosen such that $\|\boldsymbol{r}_j\| = \|(I - \omega_j A)\boldsymbol{s}_j\|$ is minimized.

  This is formally steepest descent and leads to

$$\omega_j = \frac{(A\boldsymbol{s}_j)^T \boldsymbol{s}_j}{(A\boldsymbol{s}_j)^T A\boldsymbol{s}_j}$$

The BiCGstab algorithm

Choose $\boldsymbol{x}_0$. Set $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$. Choose $\tilde{\boldsymbol{r}}_0$.
Set $\boldsymbol{p}_0 := \boldsymbol{r}_0$.
for $k = 0, 1, \ldots$ **do**
  $\alpha_k = (\boldsymbol{r}_k^T \tilde{\boldsymbol{r}}_0)/((A\boldsymbol{p}_k)^T \tilde{\boldsymbol{p}}_0)$.
  $\boldsymbol{s}_k = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k$.
  $\omega_k := ((A\boldsymbol{s}_k)^T \boldsymbol{s}_k)/((A\boldsymbol{s}_k)^T A\boldsymbol{s}_k)$.
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k + \omega_k \boldsymbol{s}_k$.
  $\boldsymbol{r}_{k+1} = \boldsymbol{s}_k - \omega_k A\boldsymbol{s}_k$.
  Test for convergence.
  $\rho_{k+1} = \boldsymbol{r}_{k+1}^T \tilde{\boldsymbol{r}}_0$.
  $\beta_k = \frac{\rho_{k+1}}{\rho_k} \cdot \frac{\alpha_{k+1}}{\omega_k}$.
  $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k(\boldsymbol{p}_k - \omega_k A\boldsymbol{p}_k)$.
end for

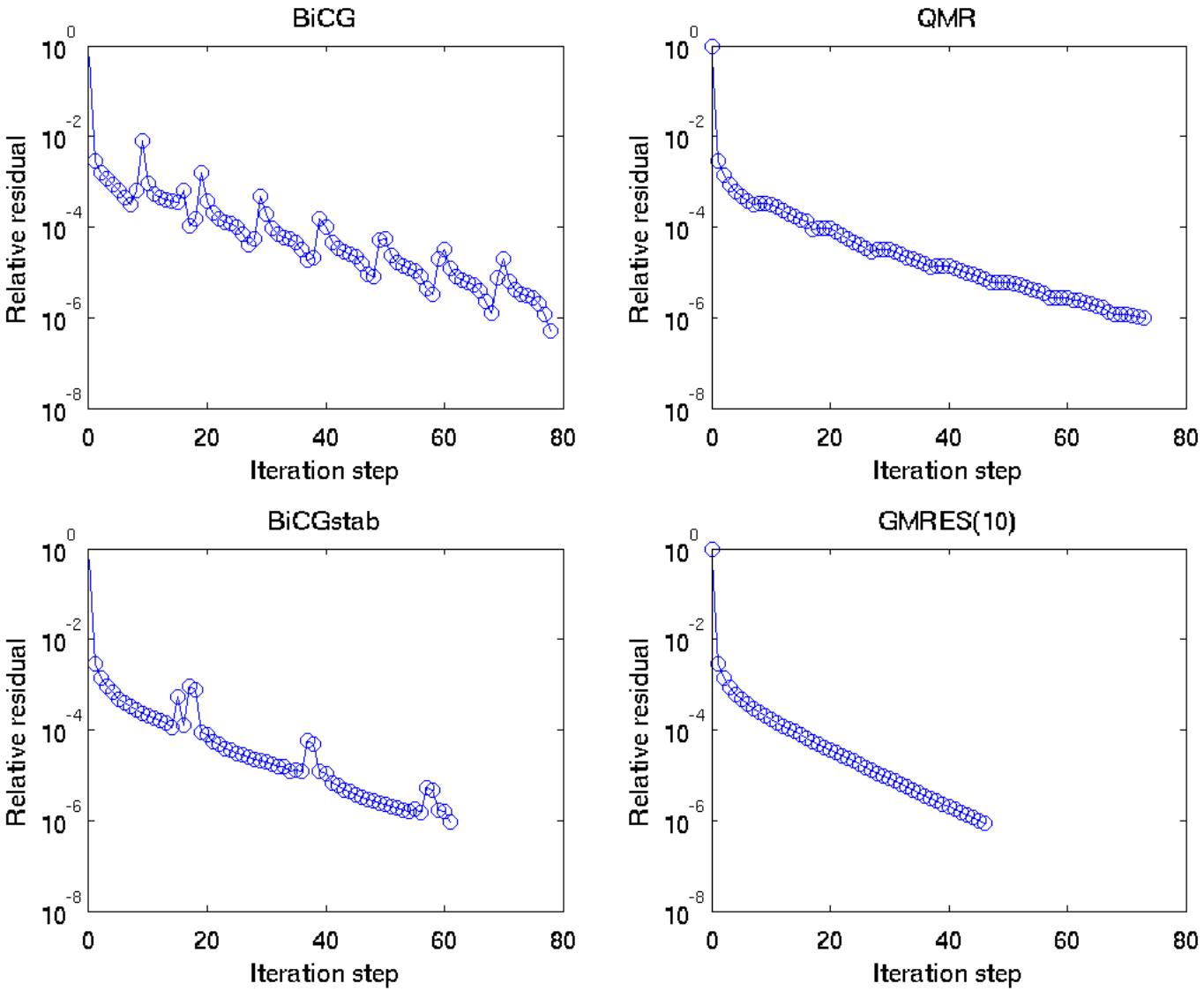
Comparison

Generate the 30'000×30'000 example

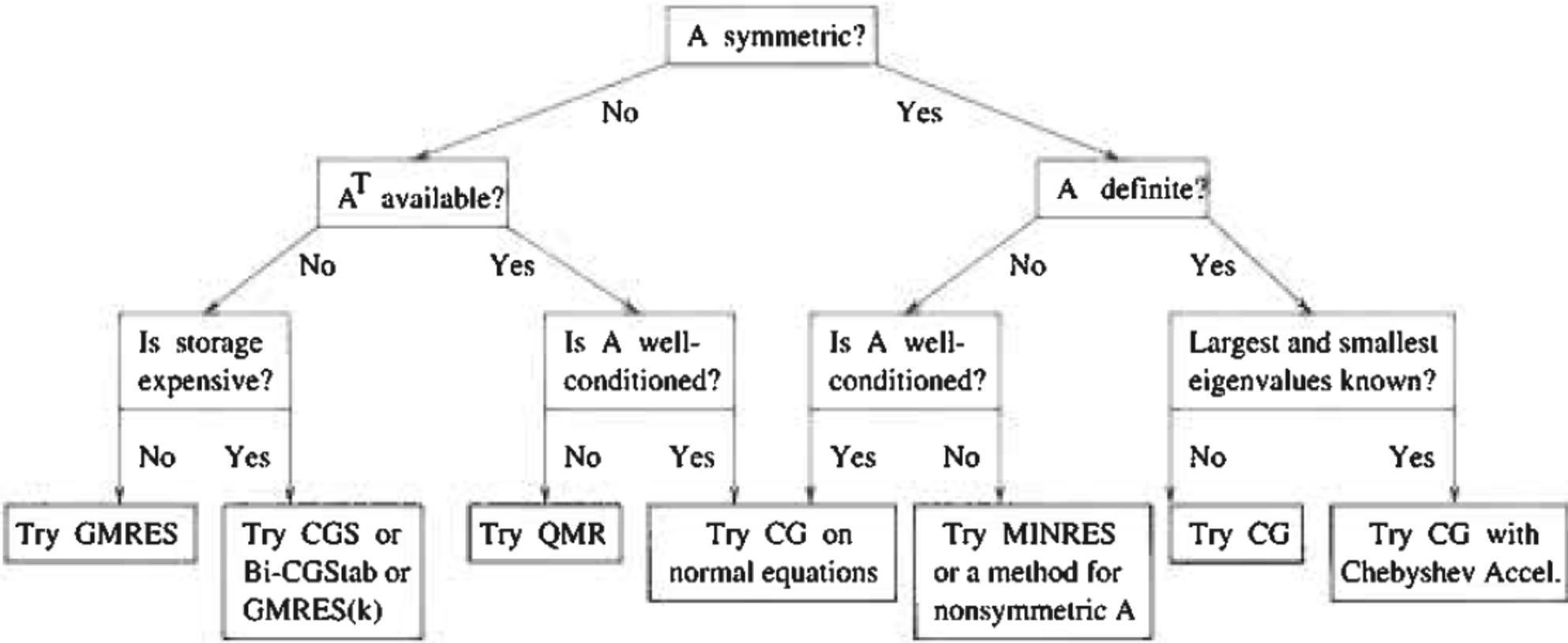
```
A = gallery('toeppen', 30000, 2 ,3, 8, 3.5, 4.5);
```

The matrix A is a nonsymmetric banded matrix with band width 5.

The condition of A is small $\kappa(A) \approx 32$.


BiCG
QMR
BiCGstab
GMRES(10)
Relative residual
Iteration step
10^0
10^-2
10^-4
10^-6
10^-8
0
20
40
60
80

Decision tree



Flow chart from J. Demmel *Applied Numerical Linear Algebra*

Iterative linear solvers in MATLAB

Function	Matrix	Method
`bicg`	General	BiConjugate Gradient
`bicgstab`	General	BiConjugate Gradient Stabilized
`cgs`	General	Conjugate Gradient Squared
`gmres`	General	Generalized Minimum Residual
`lsqr`	General	Conjugate Gradient (Normal Equations)
`minres`	Hermitian	Minimum Residual
`pcg`	Hermitian p.d.	Preconditioned Conjugate Gradient
`qmr`	General	Quasi-Minimal Residual
`symmlq`	Hermitian	Symmetric LQ

Tabelle 1: Iterative linear equation solvers in MATLAB