Large-Scale Matrix Computations

Roland W. Freund
Department of Mathematics
University of California, Davis
One Shields Avenue
Davis, CA 95616
freund@math.ucdavis.edu

Computational problems, especially in science and engineering, often involve large matrices. Examples of such problems include large sparse systems of linear equations [FGN92, Saa03, vdV03], e.g., arising from discretizations of partial differential equations, eigenvalue problems for large matrices [BDD00, LM05], linear time-invariant dynamical systems with large state-space dimensions [FF94, FF95, Fre03], and large-scale linear and nonlinear optimization problems [KR91, Wri97, NW99, GMS05]. The large matrices in these problems exhibit special structures, such as sparsity, that can be exploited in computational procedures for their solution. Roughly speaking, computational problems involving matrices are called 'large-scale' if they can be solved only by methods that exploit these special matrix structures.

1 Basic Concepts

Many of the most efficient algorithms for large-scale matrix computations are based on approximations of the given large matrix by small matrices obtained via Petrov-Galerkin projections onto suitably chosen small-dimensional subspaces. In this section, we present some basic concepts of such projections.

Definitions:

Let $C \in \mathbb{C}^{n \times n}$, and let $V_j = [\,\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_j\,] \in \mathbb{C}^{n \times j}$ be a matrix with orthonormal columns, i.e.,

$$\mathbf{v}_i^* \mathbf{v}_k = \begin{cases} 0 & \text{if } i \neq k, \\ 1 & \text{if } i = k, \end{cases} \qquad \text{for all} \quad i, k = 1, 2, \ldots, j.$$

The matrix

$$C_j := V_j^* C V_j \in \mathbb{C}^{j \times j}$$

is called the **orthogonal Petrov-Galerkin projection** of C onto the subspace

$$S = \operatorname{span}\{\,\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j\,\}$$

of $\mathbb{C}^n$ spanned by the columns of V_j.

Let $C \in \mathbb{C}^{n \times n}$, and let $V_j = [\,\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_j\,] \in \mathbb{C}^{n \times j}$ and $W_j = [\,\mathbf{w}_1 \quad \mathbf{w}_2 \quad \cdots \quad \mathbf{w}_j\,] \in \mathbb{C}^{n \times j}$ be two matrices such that $W_j^T V_j$ is nonsingular. The matrix

$$C_j := (W_j^T V_j)^{-1} W_j^T C V_j \in \mathbb{C}^{j \times j}$$

is called the **oblique Petrov-Galerkin projection** of C onto the subspace

$$S = \operatorname{span}\{\,\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j\,\}$$

of $\mathbb{C}^n$ spanned by the columns of V_j and orthogonally to the subspace

$$T = \operatorname{span}\{\,\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_j\,\}$$

of $\mathbb{C}^n$ spanned by the columns of W_j.

A **flop** is the work associated with carrying out any one of the elementary operations $a + b$, $a - b$, ab, or a/b, where a, $b \in \mathbb{C}$, in floating-point arithmetic.

Let $A = [\,a_{ik}\,] \in \mathbb{C}^{m \times n}$ be a given matrix. Matrix-vector multiplications with A are said to be **fast** if for any $\mathbf{x} \in \mathbb{C}^n$, the computation of $\mathbf{y} = A\mathbf{x}$ requires significantly fewer than $2mn$ flops.

A matrix $A = [\,a_{ik}\,] \in \mathbb{C}^{m \times n}$ is said to be **sparse** if only a small fraction of its entries a_{ik} are nonzero.

For a sparse matrix $A = [\,a_{ik}\,] \in \mathbb{C}^{m \times n}$, $nnz(A)$ denotes the number of nonzero entries of A.

A matrix $A = [\,a_{ik}\,] \in \mathbb{C}^{m \times n}$ is said to be **dense** if most of its entries a_{ik} are nonzero.

Facts:

The following facts on sparse matrices can be found in [Saa03, Chapter 3] and the facts on computing Petrov-Galerkin projections of matrices in [Saa03, Chapter 6].

1. For a sparse matrix $A = [\, a_{ik} \,] \in \mathbb{C}^{m \times n}$, only its nonzero or potentially nonzero entries a_{ik}, together with their row and column indices i and k, need to be stored.
2. Matrix-vector multiplications with a sparse matrix $A = [\, a_{ik} \,] \in \mathbb{C}^{m \times n}$ are fast. More precisely, for any $\mathbf{x} \in \mathbb{C}^n$, $\mathbf{y} = A\mathbf{x}$ can be computed with at most $2nnz(A)$ flops.
3. If $C \in \mathbb{C}^{n \times n}$ and $j \ll n$, the computational cost for computing the orthogonal Petrov-Galerkin projection of C onto the j-dimensional subspace $S = \operatorname{span}\{\, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j \,\}$ of $\mathbb{C}^n$ is dominated by the j matrix-vector products $\mathbf{y}_i = C\mathbf{v}_i$, $i = 1, 2, \ldots, j$.
4. If $C \in \mathbb{C}^{n \times n}$ and $j \ll n$, the computational cost for computing the oblique Petrov-Galerkin projection of C onto the j-dimensional subspace $S = \operatorname{span}\{\, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j \,\}$ of $\mathbb{C}^n$ and orthogonally to the j-dimensional subspace $T = \operatorname{span}\{\, \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_j \,\}$ of $\mathbb{C}^n$ is dominated by the j matrix-vector products $\mathbf{y}_i = C\mathbf{v}_i$, $i = 1, 2, \ldots, j$.
5. If matrix-vector products with a large matrix $C \in \mathbb{C}^{n \times n}$ are fast, then orthogonal and oblique Petrov-Galerkin projections C_j of C can be generated with low computational cost.

2 Sparse Matrix Factorizations

In this section, we present some basic concepts of sparse matrix factorizations. A more detailed description can be found in [DER89].

Definitions:

Let $A \in \mathbb{C}^{n \times n}$ be a sparse nonsingular matrix. A **sparse LU factorization** of A is a factorization of the form

$$A = PLUQ,$$

where $P,\ Q \in \mathbb{R}^{n\times n}$ are permutation matrices, $L \in \mathbb{C}^{n\times n}$ is a sparse unit lower triangular matrix, and $U \in \mathbb{C}^{n\times n}$ is a sparse nonsingular upper triangular matrix.

Fill-in of a sparse LU factorization $A = PLUQ$ is the set of nonzero entries of L and U that appear in positions (i,k) where $a_{ik} = 0$.

Let $A = A^* \in \mathbb{C}^{n\times n}$, $A \succ 0$, be a sparse Hermitian positive definite matrix. A **sparse Cholesky factorization** of A is a factorization of the form

$$A = PLL^*P^T,$$

where $P \in \mathbb{R}^{n\times n}$ is a permutation matrix and $L \in \mathbb{C}^{n\times n}$ is a sparse lower triangular matrix.

Fill-in of a sparse Cholesky factorization $A = PLL^*P^T$ is the set of nonzero entries of L that appear in positions (i,k) where $a_{ik} = 0$.

Let $T \in \mathbb{C}^{n\times n}$ be a sparse nonsingular (upper or lower) triangular matrix, and let $\mathbf{b} \in \mathbb{C}^n$. A **sparse triangular solve** is the solution of a linear system

$$T\mathbf{x} = \mathbf{b}$$

with a sparse triangular coefficient matrix T.

Facts:

The following facts can be found in [DER89].

1. The permutation matrices P and Q in a sparse LU factorization of A allow for reorderings of the rows and columns of A. These reorderings serve two purposes. First, they allow for pivoting for numerical stability in order to avoid division by the number 0 or by numbers close to 0, which would result in breakdowns or numerical instabilities in the procedure used for the computation of the factorization. Second, the reorderings allow for pivoting for sparsity, the goal of which is to minimize the amount of fill-in.

2. For Cholesky factorizations of matrices $A = A^* \succ 0$, the positive definiteness of A implies that pivoting for numerical stability is not needed. Therefore, the permutation matrix P in a sparse Cholesky factorization serves the single purpose of pivoting for sparsity.

3. For both sparse LU and sparse Cholesky factorizations, the problem of 'optimal' pivoting for sparsity, i.e., finding reorderings that minimize the amount of fill-in, is NP-complete. This means that for practical purposes, minimizing the amount of fill-in of factorizations of large sparse matrices is impossible in general. However, there is a large number of pivoting strategies that—while not minimizing fill-in—efficiently limit the amount of fill-in for many important classes of large sparse matrices; see, e.g., [DER89].

4. A sparse triangular solve with the matrix T requires at most $2nnz(T)$ flops.

5. Not every large sparse matrix A has a a sparse LU factorization with limited amounts of fill-in. For example, LU or Cholesky factorizations of sparse matrices A arising from discretization of partial differential equations for three-dimensional problems are often prohibitive due to the large amount of fill-in.

Examples:

1. Given a sparse LU factorization $A = PLUQ$ of a sparse nonsingular matrix $A \in \mathbb{C}^{n\times n}$, the solution $\mathbf{x}$ of the linear system $A\mathbf{x} = \mathbf{b}$ with any right-hand side $\mathbf{b} \in \mathbb{C}^n$ can be computed as follows:

$$\begin{aligned} \text{Set} \quad & \mathbf{c} = P^T\mathbf{b}; \\ \text{Solve} \quad & L\mathbf{z} = \mathbf{c} \quad \text{for} \quad \mathbf{z}; \\ \text{Solve} \quad & U\mathbf{y} = \mathbf{z} \quad \text{for} \quad \mathbf{y}; \\ \text{Set} \quad & \mathbf{x} = Q^T\mathbf{y}. \end{aligned}$$

   Since P and Q are permutation matrices, the first and the last step are just reorderings of the entries of the vectors $\mathbf{b}$ and $\mathbf{y}$, respectively. Therefore, the main computational cost is the two triangular solves with L and U, which requires at most $2(nnz(L) + nnz(U))$ flops.

2. Given a sparse Cholesky factorization $A = PLL^*P^T$ of a sparse Hermitian positive definite matrix $A \in \mathbb{C}^{n\times n}$, the solution $\mathbf{x}$ of the linear system $A\mathbf{x} = \mathbf{b}$ with any right-

hand side $\mathbf{b} \in \mathbb{C}^n$ can be computed as follows:

$$\begin{aligned}
\text{Set} \quad & \mathbf{c} = P^T \mathbf{b}; \\
\text{Solve} \quad & L\mathbf{z} = \mathbf{c} \quad \text{for} \quad \mathbf{z}; \\
\text{Solve} \quad & L^* \mathbf{y} = \mathbf{z} \quad \text{for} \quad \mathbf{y}; \\
\text{Set} \quad & \mathbf{x} = P^T \mathbf{y}.
\end{aligned}$$

Since P is a permutation matrix, the first and the last step are just reorderings of the entries of the vectors $\mathbf{b}$ and $\mathbf{y}$, respectively. Therefore, the main computational cost is the two triangular solves with L and L^*, which requires at most $4nnz(L)$ flops.

3. In large-scale matrix computations, sparse factorizations are often not applied to a given sparse matrix $A \in \mathbb{C}^{n\times n}$, but to a suitable 'approximation' $A_0 \in \mathbb{C}^{n\times n}$ of A. For example, if sparse factorizations of A itself are prohibitive due to excessive fill-in, such approximations A_0 can often be obtained by computing an 'incomplete' factorization of A that simply discards unwanted fill-in entries. Given a sparse LU factorization

$$A_0 = PLUQ$$

of a sparse nonsingular matrix $A_0 \in \mathbb{C}^{n\times n}$, which in some sense approximates the original matrix $A \in \mathbb{C}^{n\times n}$, one then uses iterative procedures that only involve matrix-vector products with the matrix

$$C := A_0^{-1} A = Q^T U^{-1} L^{-1} P^T A,$$

or possibly its transpose C^T. In the context of solving linear systems $A\mathbf{x} = \mathbf{b}$, the matrix A_0 is called a **preconditioner**, and the matrix C is called the **preconditioned** coefficient matrix.

In general, the matrix $C = A_0^{-1} A$ is full. However, if C is only used in the form of matrix-vector products, then there is no need to explicitly form C. Instead, for any

$\mathbf{v} \in \mathbb{C}^n$, the result of the matrix-vector product $\mathbf{y} = C\mathbf{v}$ can be computed as follows:

$$\begin{aligned} \text{Set} \quad & \mathbf{c} = A\mathbf{v}; \\ \text{Set} \quad & \mathbf{d} = P^T\mathbf{c}; \\ \text{Solve} \quad & L\mathbf{f} = \mathbf{d} \quad \text{for} \quad \mathbf{f}; \\ \text{Solve} \quad & U\mathbf{z} = \mathbf{f} \quad \text{for} \quad \mathbf{z}; \\ \text{Set} \quad & \mathbf{y} = Q^T\mathbf{z}. \end{aligned}$$

Since P and Q are permutation matrices, the second and the last step are just reorderings of the entries of the vectors $\mathbf{c}$ and $\mathbf{z}$, respectively. Therefore, the main computational cost is the matrix-vector product with the sparse matrix A in the first step, the triangular solve with L in the third step, and the triangular solve with U in the fourth step, which requires a total of at most $2(nnz(A) + nnz(L) + nnz(U))$ flops. Similarly, each matrix-product with C^T can be computed with at most $2(nnz(A) + nnz(L) + nnz(U))$ flops. In particular, matrix-vector products with both C and C^T are fast.

4. For sparse Hermitian matrices $A = A^* \in \mathbb{C}^{n\times n}$, preconditioning is often applied in a symmetric manner. Suppose

$$A_0 = PLL^*P^T$$

is a sparse Cholesky factorization of a sparse matrix $A_0 = A_0^* \in \mathbb{C}^{n\times n}$, $A_0 \succ 0$, which in some sense approximates the original matrix A. Then the symmetrically preconditioned matrix C is defined as

$$C := (PL)^{-1}A(L^*P^T)^{-1} = L^{-1}P^TAP(L^*)^{-1}.$$

Note that $C = C^*$ is a Hermitian matrix. For any $\mathbf{v} \in \mathbb{C}^n$, the result of the matrix-vector product $\mathbf{y} = C\mathbf{v}$ can be computed as follows:

$$\begin{aligned} \text{Solve} \quad & L^*\mathbf{c} = \mathbf{v} \quad \text{for} \quad \mathbf{c}; \\ \text{Set} \quad & \mathbf{d} = P\mathbf{c}; \\ \text{Set} \quad & \mathbf{f} = A\mathbf{d}; \\ \text{Set} \quad & \mathbf{z} = P^T\mathbf{f}; \\ \text{Solve} \quad & L\mathbf{y} = \mathbf{z} \quad \text{for} \quad \mathbf{y}. \end{aligned}$$

The main computational cost is the triangular solve with L^* in the first step, the matrix-vector product with the sparse matrix A in the third step, and the triangular solve with L in the last step, which requires a total of at most $2(nnz(A) + 2nnz(L))$ flops. In particular, matrix-vector products with C are fast.

3 Krylov Subspaces

Petrov-Galerkin projections are often used in conjunction with Krylov subspaces. In this section, we present the basic concepts of Krylov subspaces. In the following, it is assumed that $C \in \mathbb{C}^{n \times n}$ and $\mathbf{r} \in \mathbb{C}^n$, $\mathbf{r} \neq \mathbf{0}$.

Definitions:

The sequence

$$\mathbf{r}, C\mathbf{r}, C^2\mathbf{r}, \ldots, C^{j-1}\mathbf{r}, \ldots$$

is called the **Krylov sequence** induced by C and $\mathbf{r}$.

Let $j \geq 1$. The subspace

$$K_j(C, \mathbf{r}) := \operatorname{span}\{\, \mathbf{r}, C\mathbf{r}, C^2\mathbf{r}, \ldots, C^{j-1}\mathbf{r} \,\}$$

of $\mathbb{C}^n$ spanned by the first j vectors of the Krylov sequence is called the **jth Krylov subspace** induced by C and $\mathbf{r}$.

A sequence of linearly independent vectors

$$\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j \in \mathbb{C}^n$$

is said to be a **nested basis** for the jth Krylov subspace $K_j(C, \mathbf{r})$ if

$$\operatorname{span}\{\, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_i \,\} = K_i(C, \mathbf{r}) \quad \text{for all} \quad i = 1, 2, \ldots, j.$$

Let $p(\lambda) = c_0 + c_1\lambda + c_2\lambda^2 + \cdots + c_{d-1}\lambda^{d-1} + \lambda^d$ be a monic polynomial of degree d with coefficients in $\mathbb{C}$. The **minimal polynomial of C with respect to $\mathbf{r}$** is the unique monic polynomial of smallest possible degree for which $p(C)\mathbf{r} = \mathbf{0}$.

The **grade of C with respect to $\mathbf{r}$**, $d(C, \mathbf{r})$, is the degree of the minimal polynomial of C and $\mathbf{r}$.

Facts:

The following facts can be found in [Hou75, Section 1.5], [SB02, Section 6.3], or [Saa03, Section 6.2].

1. The vectors
$$\mathbf{r}, C\mathbf{r}, C^2\mathbf{r}, \dots, C^{j-1}\mathbf{r}$$
are linearly independent if, and only if, $j \le d(C, \mathbf{r})$.

2. Let $d = d(C, \mathbf{r})$. The vectors
$$\mathbf{r}, C\mathbf{r}, C^2\mathbf{r}, \dots, C^{d-2}\mathbf{r}, C^{d-1}\mathbf{r}, C^j\mathbf{r}$$
are linearly dependent for all $j > d$.

3. The dimension of the jth Krylov subspace $K_j(C, \mathbf{r})$ is given by
$$\dim K_j(C, \mathbf{r}) = \begin{cases} j & \text{if } j \le d(C, \mathbf{r}), \\ d(C, \mathbf{r}) & \text{if } j > d(C, \mathbf{r}). \end{cases}$$

4. $d(C, \mathbf{r}) = \operatorname{rank} [\, \mathbf{r} \quad C\mathbf{r} \quad C^2\mathbf{r} \quad \cdots \quad C^{n-1}\mathbf{r} \,]$.

4 The Symmetric Lanczos Process

In this section, we assume that $C = C^* \in \mathbb{C}^{n\times n}$ is a Hermitian matrix, and that $\mathbf{r} \in \mathbb{C}^n$, $\mathbf{r} \ne \mathbf{0}$, is a nonzero starting vector. We discuss the **symmetric Lanczos process** [Lan50] for constructing a nested basis for the Krylov subspace $K_j(C, \mathbf{r})$ induced by C and $\mathbf{r}$.

```
**Algorithm** (Symmetric Lanczos process)
Compute $\beta_1 = \|\mathbf{r}\|_2$, and set $\mathbf{v}_1 = \mathbf{r}/\beta_1$, and $\mathbf{v}_0 = \mathbf{0}$.
For $j = 1, 2, \ldots,$ do:
    1) Compute $\mathbf{v} = C\mathbf{v}_j$, and set $\mathbf{v} = \mathbf{v} - \mathbf{v}_{j-1}\beta_j$.
    2) Compute $\alpha_j = \mathbf{v}_j^*\mathbf{v}$, and set $\mathbf{v} = \mathbf{v} - \mathbf{v}_j\alpha_j$.
    3) Compute $\beta_{j+1} = \|\mathbf{v}\|_2$.
       If $\beta_{j+1} = 0$, stop.
       Otherwise, set $\mathbf{v}_{j+1} = \mathbf{v}/\beta_{j+1}$.
end for
```

Facts:

The following facts can be found in [CW85], [SB02, Section 6.5.3], or [Saa03, Section 6.6].

1. In exact arithmetic, the algorithm stops after a finite number of iterations. More precisely, it stops when $j = d(C, \mathbf{r})$ is reached.

2. The **Lanczos vectors**

$$\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j$$

generated during the first j iterations of the algorithm form a nested basis for the jth Krylov subspace $K_j(C, \mathbf{r})$.

3. The Lanczos vectors satisfy the three-term recurrence relations

$$\mathbf{v}_{i+1}\beta_{i+1} = C\mathbf{v}_i - \mathbf{v}_i\alpha_i - \mathbf{v}_{i-1}\beta_i, \quad i = 1, 2, \ldots, j.$$

4. These three-term recurrence relations can be written in compact matrix form as follows:

$$CV_j = V_jT_j + \beta_{j+1}\mathbf{v}_{j+1}\mathbf{e}_j^T = V_{j+1}T_j^{(e)}.$$

Here, we set

$$V_j = [\,\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_j\,], \quad \mathbf{e}_j^T = [\,0 \quad 0 \quad \cdots \quad 0 \quad 1\,] \in \mathbb{R}^{1\times j},$$

$$T_j = \begin{bmatrix} \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ \beta_2 & \alpha_2 & \beta_3 & \ddots & \vdots \\ 0 & \beta_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_j \\ 0 & \cdots & 0 & \beta_j & \alpha_j \end{bmatrix}, \quad T_j^{(e)} = \begin{bmatrix} T_j \\ \beta_{j+1}\mathbf{e}_j^T \end{bmatrix}, \quad \text{and} \quad V_{j+1} = [\,V_j \quad \mathbf{v}_{j+1}\,].$$

Note that $T_j \in \mathbb{C}^{j\times j}$ and $T_j^{(e)} \in \mathbb{C}^{(j+1)\times j}$ are tridiagonal matrices.

5. In exact arithmetic, the Lanczos vectors are orthonormal. Since the Lanczos vectors are the columns of V_j, this orthonormality can be stated compactly as follows:

$$V_j^* V_j = I_j \quad \text{and} \quad V_j^* \mathbf{v}_{j+1} = \mathbf{0}.$$

6. These orthogonality relations, together with the above compact form of the three-term recurrence relations, imply that

$$T_j = V_j^* C V_j.$$

Thus the jth **Lanczos matrix** T_j is the orthogonal Petrov-Galerkin projection of C onto the jth Krylov subspace $K_j(C, \mathbf{r})$.

7. The computational cost of each jth iteration of the symmetric Lanczos process is fixed, and it is dominated by the matrix-vector product $\mathbf{v} = C\mathbf{v}_j$. In particular, the computational cost for generating the orthogonal Petrov-Galerkin projection T_j of C is dominated by the j matrix-vector products with C.

8. If C is a sparse matrix or a preconditioned matrix with a sparse preconditioner, then the matrix-vector products with C are fast. In this case, the symmetric Lanczos process is a very efficient procedure for computing orthogonal Petrov-Galerkin projections T_j of C onto Krylov subspaces $K_j(C, \mathbf{r})$.

9. The three-term recurrence relations used to generate the Lanczos vectors explicitly enforce orthogonality only among each set of three consecutive vectors, $\mathbf{v}_{j-1}$, $\mathbf{v}_j$, and $\mathbf{v}_{j+1}$.

As a consequence, in finite-precision arithmetic, round-off error will usually cause loss of orthogonality among all Lanczos vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{j+1}$.

10. For applications of the Lanczos process in large-scale matrix computations, this loss of orthogonality is often benign, and only delays convergence. More precisely, in such applications, the Lanczos matrix $T_j \in \mathbb{C}^{j \times j}$ for some $j \ll n$ is used to obtain an approximate solution of a matrix problem involving the large matrix $C \in \mathbb{C}^{n \times n}$. Due to round-off error and the resulting loss of orthogonality, the number j of iterations that is needed to obtain a satisfactory approximate solution is larger than the number of iterations that would be needed in exact arithmetic.

5 The Nonsymmetric Lanczos Process

In this section, we assume that $C \in \mathbb{C}^{n \times n}$ is a general square matrix, and that $\mathbf{r} \in \mathbb{C}^n$, $\mathbf{r} \neq \mathbf{0}$, and $\mathbf{l} \in \mathbb{C}^n$, $\mathbf{l} \neq \mathbf{0}$, is a pair of right and left nonzero starting vectors. The **nonsymmetric Lanczos process** [Lan50] is an extension of the symmetric Lanczos process that simultaneously constructs a nested basis for the Krylov subspace $K_j(C, \mathbf{r})$ induced by C and $\mathbf{r}$, and a nested basis for the Krylov subspace $K_j(C^T, \mathbf{l})$ induced by C^T and $\mathbf{l}$. In the context of the nonsymmetric Lanczos process, $K_j(C, \mathbf{r})$ is called the jth **right** Krylov subspace, and $K_j(C^T, \mathbf{l})$ is called the jth **left** Krylov subspace.

```
**Algorithm** (Nonsymmetric Lanczos process)
Compute $\rho_1 = \|\mathbf{r}\|_2$, $\eta_1 = \|\mathbf{l}\|_2$, and set $\mathbf{v}_1 = \mathbf{r}/\beta_1$, $\mathbf{w}_1 = \mathbf{l}/\eta_1$, $\mathbf{v}_0 = \mathbf{w}_0 = \mathbf{0}$, and $\delta_0 = 1$.
For $j = 1, 2, \ldots,$ do:
    1) Compute $\delta_j = \mathbf{w}_j^T \mathbf{v}_j$.
       If $\delta_j = 0$, stop.
    2) Compute $\mathbf{v} = C\mathbf{v}_j$, and set $\beta_j = \eta_j \delta_j / \delta_{j-1}$ and $\mathbf{v} = \mathbf{v} - \mathbf{v}_{j-1}\beta_j$.
    3) Compute $\alpha_j = \mathbf{w}_j^T \mathbf{v}$, and set $\mathbf{v} = \mathbf{v} - \mathbf{v}_j \alpha_j$.
    4) Compute $\mathbf{w} = C^T \mathbf{w}_j$, and set $\gamma_j = \rho_j \delta_j / \delta_{j-1}$ and $\mathbf{w} = \mathbf{w} - \mathbf{w}_j \alpha_j - \mathbf{w}_{j-1}\gamma_j$.
    5) Compute $\rho_{j+1} = \|\mathbf{v}\|_2$ and $\eta_{j+1} = \|\mathbf{w}\|_2$.
       If $\rho_{j+1} = 0$ or $\eta_{j+1} = 0$, stop.
       Otherwise, set $\mathbf{v}_{j+1} = \mathbf{v}/\rho_{j+1}$ and $\mathbf{w}_{j+1} = \mathbf{w}/\eta_{j+1}$.
end for
```

Facts:

The following facts can be found in [SB02, Section 8.7.3] or [Saa03, Section 7.1].

1. The occurrence of $\delta_j = 0$ in Step 1) of the nonsymmetric Lanczos process is called an **exact breakdown**. In finite-precision arithmetic, one also needs to check for $\delta_j \approx 0$, which is called a **near-breakdown**. It is possible to continue the nonsymmetric Lanczos process even if an exact breakdown or a near-breakdown has occurred, by using so-called 'look-ahead' techniques; see, e.g., [FGN93] and the references given there. However, in practice, exact breakdowns and even near-breakdowns are fairly rare, and therefore, here we consider only the basic form of the nonsymmetric Lanczos process without look-ahead.

2. In exact arithmetic and if no exact breakdowns occur, the algorithm stops after a finite number of iterations. More precisely, it stops when $j = \min\{\, d(C, \mathbf{r}), d(C^T, \mathbf{l})\,\}$ is reached.

3. The **right Lanczos vectors** and the **left Lanczos vectors**

$$\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j \quad \text{and} \quad \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_j$$

generated during the first j iterations of the algorithm form a nested basis for the jth right Krylov subspace $K_j(C, \mathbf{r})$ and the jth left Krylov subspace $K_j(C^T, \mathbf{l})$, respectively.

4. The right and left Lanczos vectors satisfy the three-term recurrence relations

$$\mathbf{v}_{i+1}\rho_{i+1} = C\mathbf{v}_i - \mathbf{v}_i\alpha_i - \mathbf{v}_{i-1}\beta_i, \quad i = 1, 2, \ldots, j,$$

   and

$$\mathbf{w}_{i+1}\eta_{i+1} = C^T\mathbf{w}_i - \mathbf{w}_i\alpha_i - \mathbf{w}_{i-1}\gamma_i, \quad i = 1, 2, \ldots, j,$$

   respectively.

5. These three-term recurrence relations can be written in compact matrix form as follows:

$$\begin{aligned} CV_j &= V_jT_j + \rho_{j+1}\mathbf{v}_{j+1}\mathbf{e}_j^T = V_{j+1}T_j^{(e)}, \\ C^TW_j &= W_j\tilde{T}_j + \eta_{j+1}\mathbf{w}_{j+1}\mathbf{e}_j^T. \end{aligned}$$

   Here, we set

$$\begin{aligned} V_j &= [\,\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_j\,], \quad W_j = [\,\mathbf{w}_1 \quad \mathbf{w}_2 \quad \cdots \quad \mathbf{w}_j\,], \\ T_j &= \begin{bmatrix} \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ \rho_2 & \alpha_2 & \beta_3 & \ddots & \vdots \\ 0 & \rho_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_j \\ 0 & \cdots & 0 & \rho_j & \alpha_j \end{bmatrix}, \quad \tilde{T}_j = \begin{bmatrix} \alpha_1 & \gamma_2 & 0 & \cdots & 0 \\ \eta_2 & \alpha_2 & \gamma_3 & \ddots & \vdots \\ 0 & \eta_3 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \gamma_j \\ 0 & \cdots & 0 & \eta_j & \alpha_j \end{bmatrix}, \\ \mathbf{e}_j^T &= [\,0 \quad 0 \quad \cdots \quad 0 \quad 1\,] \in \mathbb{R}^{1\times j}, \quad \text{and} \quad T_j^{(e)} = \begin{bmatrix} T_j \\ \rho_{j+1}\mathbf{e}_j^T \end{bmatrix}. \end{aligned}$$

   Note that T_j, $\tilde{T}_j \in \mathbb{C}^{j\times j}$, and $T_j^{(e)} \in \mathbb{C}^{(j+1)\times j}$ are tridiagonal matrices.

6. The matrix $T_j^{(e)}$ has full rank, i.e., $\operatorname{rank} T_j^{(e)} = j$.

7. In exact arithmetic, the right and left Lanczos vectors are **biorthogonal** to each other, i.e.,

$$\mathbf{w}_i^T\mathbf{v}_k = \begin{cases} 0 & \text{if } i \neq k, \\ \delta_i & \text{if } i = k, \end{cases} \qquad \text{for all} \quad i, k = 1, 2, \ldots, j.$$

Since the right and left Lanczos vectors are the columns of V_j and W_j, respectively, the biorthogonality can be stated compactly as follows:

$$W_j^T V_j = D_j, \quad W_j^T \mathbf{v}_{j+1} = \mathbf{0}, \quad \text{and} \quad V_j^T \mathbf{w}_{j+1} = \mathbf{0}.$$

Here, D_j is the diagonal matrix

$$D_j = \operatorname{diag}(\delta_1, \delta_2, \ldots, \delta_j).$$

Note that D_j is nonsingular, as long as no exact breakdowns occur.

8. These biorthogonality relations, together with the above compact form of the three-term recurrence relations, imply that

$$T_j = D_j^{-1} V_j^* C V_j = (W_j^T V_j)^{-1} W_j^T C V_j.$$

Thus, the jth **Lanczos matrix** T_j is the oblique Petrov-Galerkin projection of C onto the jth right Krylov subspace $K_j(C, \mathbf{r})$, and orthogonally to the jth left Krylov subspace $K_j(C^T, \mathbf{l})$.

9. The matrices T_j and $\tilde{T}_j^T$ are diagonally similar:

$$\tilde{T}_j^T = D_j T_j D_j^{-1}.$$

10. The computational cost of each jth iteration of the nonsymmetric Lanczos process is fixed, and it is dominated by the matrix-vector product $\mathbf{v} = C\mathbf{v}_j$ with C and by the matrix-vector product $\mathbf{w} = C^T \mathbf{w}_j$ with C^T. In particular, the computational cost for generating the oblique Petrov-Galerkin projection T_j of C is dominated by the j matrix-vector products with C and the j matrix-vector products with C^T.

11. If C is a sparse matrix or a preconditioned matrix with a sparse preconditioner, then the matrix-vector products with C and C^T are fast. In this case, the nonsymmetric Lanczos process is a very efficient procedure for computing oblique Petrov-Galerkin projections T_j of C onto right Krylov subspaces $K_j(C, \mathbf{r})$ and orthogonally to left Krylov subspaces $K_j(C^T, \mathbf{l})$.

12. The three-term recurrence relations, which are used to generate the right and left Lanczos vectors, explicitly enforce biorthogonality only between three consecutive right vectors, $\mathbf{v}_{j-1}$, $\mathbf{v}_j$, $\mathbf{v}_{j+1}$, and three consecutive left vectors, $\mathbf{w}_{j-1}$, $\mathbf{w}_j$, $\mathbf{w}_{j+1}$. As a consequence, in finite-precision arithmetic, round-off error will usually cause loss of biorthogonality between all right vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{j+1}$ and all left vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{j+1}$.

13. For applications of the Lanczos process in large-scale matrix computations, this loss of orthogonality is often benign, and only delays convergence. More precisely, in such applications, the Lanczos matrix $T_j \in \mathbb{C}^{j \times j}$ for some $j \ll n$ is used to obtain an approximate solution of a matrix problem involving the large matrix $C \in \mathbb{C}^{n \times n}$. Due to round-off error and the resulting loss of biorthogonality, the number j of iterations that is needed to obtain a satisfactory approximate solution is larger than the number of iterations that would be needed in exact arithmetic; see, e.g., [CW86].

14. If $C = C^*$ is a Hermitian matrix and $\mathbf{l} = \overline{\mathbf{r}}$, i.e., the left starting vector $\mathbf{l}$ is the complex conjugate of the right starting vector $\mathbf{r}$, then the right and left Lanczos vectors satisfy

$$\mathbf{w}_i = \overline{\mathbf{v}_i} \quad \text{for all} \quad i = 1, 2, \ldots, j+1,$$

and the nonsymmetric Lanczos process reduces to the symmetric Lanczos process.

6 The Arnoldi Process

The **Arnoldi process** [Arn51] is another extension of the symmetric Lanczos process for Hermitian matrices to general square matrices. Unlike the nonsymmetric Lanczos process, which produces bases for both right and left Krylov subspaces, the Arnoldi process generates basis vectors only for the right Krylov subspaces. However, these basis vectors are constructed to be orthonormal, resulting in a numerical procedure that is much more robust than the nonsymmetric Lanczos process.

In this section, we assume that $C \in \mathbb{C}^{n \times n}$ is a general square matrix, and that $\mathbf{r} \in \mathbb{C}^n$, $\mathbf{r} \neq \mathbf{0}$, is a nonzero starting vector.

```
**Algorithm** (Arnoldi process)
Compute $\rho_1 = \|\mathbf{r}\|_2$, and set $\mathbf{v}_1 = \mathbf{r}/\rho_1$.
For $j = 1, 2, \dots,$ do:
  1) Compute $\mathbf{v} = C\mathbf{v}_j$.
  2) For $i = 1, 2, \dots, j$, do:
        Compute $h_{ij} = \mathbf{v}^* \mathbf{v}_i$, and set $\mathbf{v} = \mathbf{v} - \mathbf{v}_j h_{ij}$.
     end for
  3) Compute $h_{j+1,j} = \|\mathbf{v}\|_2$.
     If $h_{j+1,j} = 0$, stop.
     Otherwise, set $\mathbf{v}_{j+1} = \mathbf{v}/h_{j+1,j}$.
end for
```

Facts:

The following facts can be found in [Saa03, Section 6.3].

1. In exact arithmetic, the algorithm stops after a finite number of iterations. More precisely, it stops when $j = d(C, \mathbf{r})$ is reached.

2. The **Arnoldi vectors**

$$\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_j$$

   generated during the first j iterations of the algorithm form a nested basis for the jth Krylov subspace $K_j(C, \mathbf{r})$.

3. The Arnoldi vectors satisfy the $(i+1)$-term recurrence relations

$$\mathbf{v}_{i+1} h_{i+1,i} = C\mathbf{v}_i - \mathbf{v}_i h_{ii} - \mathbf{v}_{i-1} h_{i-1,i} - \cdots - \mathbf{v}_2 h_{2i} - \mathbf{v}_1 h_{1i}, \quad i = 1, 2, \dots, j.$$

   These $(i+1)$-term recurrence relations can be written in compact matrix form as follows:

$$CV_j = V_j H_j + h_{j+1,j} \mathbf{v}_{j+1} \mathbf{e}_j^T = V_{j+1} H_j^{(e)}.$$

Here, we set

$$\begin{aligned}
V_j &= [\,\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_j\,], \quad \mathbf{e}_j^T = [0 \quad 0 \quad \cdots \quad 0 \quad 1] \in \mathbb{R}^{1\times j},\\
H_j &= \begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdots & h_{1j} \\ h_{21} & h_{22} & h_{23} & \ddots & \vdots \\ 0 & h_{32} & \ddots & \ddots & h_{j-2,j} \\ \vdots & \ddots & \ddots & \ddots & h_{j-1,j} \\ 0 & \cdots & 0 & h_{j,j-1} & h_{jj} \end{bmatrix},\\
H_j^{(e)} &= \begin{bmatrix} H_j \\ h_{j+1,j}\mathbf{e}_j^T \end{bmatrix}, \quad \text{and} \quad V_{j+1} = [\,V_j \quad \mathbf{v}_{j+1}\,].
\end{aligned}$$

Note that $H_j \in \mathbb{C}^{j\times j}$ and $H_j^{(e)} \in \mathbb{C}^{(j+1)\times j}$ are upper Hessenberg matrices.

4. The matrix $H_j^{(e)}$ has full rank, i.e., $\operatorname{rank} H_j^{(e)} = j$.

5. Since the Arnoldi vectors are the columns of V_j, this orthonormality can be stated compactly as follows:

$$V_j^* V_j = I_j \quad \text{and} \quad V_j^* \mathbf{v}_{j+1} = \mathbf{0}.$$

6. These orthogonality relations, together with the above compact form of the recurrence relations, imply that

$$H_j = V_j^* C V_j.$$

Thus, the jth **Arnoldi matrix** H_j is the orthogonal Petrov-Galerkin projection of C onto the jth Krylov subspace $K_j(C, \mathbf{r})$.

7. As in the case of the symmetric Lanczos process, each jth iteration of the Arnoldi process requires only a single matrix-vector product $\mathbf{v} = C\mathbf{v}_j$. If C is a sparse matrix or a preconditioned matrix with a sparse preconditioner, then the matrix-vector products with C are fast.

8. However, unlike the Lanczos process, the additional computations in each jth iteration do increase with j. In particular, each jth iteration requires the computation of j inner products of vectors of length n, and the computation of j SAXPY-type updates of the form $\mathbf{v} = \mathbf{v} - \mathbf{v}_j h_{ij}$ with vectors of length n.

9. For most large-scale matrix computations, the increasing work per iteration limits the number of iterations that the Arnoldi process can be run. Therefore, in practice, the Arnoldi process is usually combined with restarting, i.e., after a number of iterations (with the matrix C and starting vector $\mathbf{r}$), the algorithm is started again with the same matrix C, but a different starting vector, say $\mathbf{r}_1$.

10. On the other hand, the $(i+1)$-term recurrence relations used to generate the Arnoldi vectors explicitly enforce orthogonality among the first $i+1$ vectors, $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{i+1}$. As a result, the Arnoldi process is much less susceptible to round-off error in finite-precision arithmetic than the Lanczos process.

7 Eigenvalue Computations

In this section, we consider the problem of computing a few eigenvalues, and possibly eigenvectors, of a large matrix $C \in \mathbb{C}^{n \times n}$. We assume that matrix-vector products with C are fast. In this case, orthogonal and, in the non-Hermitian case, oblique Petrov-Galerkin projections of C onto Krylov subspaces $K_j(C, \mathbf{r})$ can be computed efficiently, as long as $j \ll n$.

Facts:

The following facts can be found in [CW85], [CW86], and [BDD00].

1. Assume that $C = C^* \in \mathbb{C}^{n \times n}$ is a Hermitian matrix. We choose any nonzero starting vector $\mathbf{r} \in \mathbb{C}^n$, $\mathbf{r} \neq \mathbf{0}$, e.g., a vector with random entries, and run the symmetric Lanczos process. After j iterations of the algorithm, we have computed the jth Lanczos matrix T_j, which—in exact arithmetic—is the orthogonal Petrov-Galerkin projection of C onto the jth Krylov subspace $K_j(C, \mathbf{r})$. Neglecting the last term in the compact form of the three-term recurrence relations used in the first j iterations of the symmetric Lanczos process, we obtain the approximation

$$CV_j \approx V_j T_j.$$

This approximation suggests to use the j eigenvalues $\lambda_i^{(j)}$, $i = 1, 2, \ldots, j$, of the jth

Lanczos matrix $T_j \in \mathbb{C}^{j\times j}$ as approximate eigenvalues of the original matrix C. Furthermore, if one is also interested in approximate eigenvectors, then the above approximation suggests to use

$$\mathbf{x}_i^{(j)} = V_j \mathbf{z}_i^{(j)} \in \mathbb{C}^n, \quad \text{where} \quad T_j \mathbf{z}_i^{(j)} = \mathbf{z}_i^{(j)} \lambda_i^{(j)}, \quad \mathbf{z}_i^{(j)} \neq \mathbf{0},$$

as an approximate eigenvector of C corresponding to the approximate eigenvalue $\lambda_i^{(j)}$ of C.

2. Assume that $C \in \mathbb{C}^{n\times n}$ is a general square matrix. Here one can use either the nonsymmetric Lanczos process or the Arnoldi process to obtain approximate eigenvalues.

3. In the case of the nonsymmetric Lanczos process, one chooses any nonzero starting vectors $\mathbf{r} \in \mathbb{C}^n$, $\mathbf{r} \neq \mathbf{0}$, and $\mathbf{l} \in \mathbb{C}^n$, $\mathbf{l} \neq \mathbf{0}$, $\mathbf{r} \in \mathbb{C}^n$, $\mathbf{r} \neq \mathbf{0}$. In analogy to the symmetric case, the eigenvalues of Lanczos matrix $T_j \in \mathbb{C}^{j\times j}$ computed by j iterations of the nonsymmetric Lanczos process are used as approximate eigenvalues of the original matrix C. Corresponding approximate right eigenvectors are given by the same formula as above. Furthermore, one can also obtain approximate left eigenvectors from the left eigenvectors of T_j and the first j left Lanczos vectors. A discussion of many practical aspects of using the nonsymmetric Lanczos process for eigenvalue computations can be found in [CW86].

4. In the case of the Arnoldi process, one only needs to choose a single nonzero starting vector $\mathbf{r} \in \mathbb{C}^n$, $\mathbf{r} \neq \mathbf{0}$. Here, one has the approximation

$$CV_j \approx V_j H_j,$$

where V_j is the matrix containing the first j Arnoldi vectors as columns and H_j is the jth Arnoldi matrix. The eigenvalues $\lambda_i^{(j)}$, $i = 1, 2, \ldots, j$, of $H_j \in \mathbb{C}^{j\times j}$ are used as approximate eigenvalues of C. Furthermore, for each i,

$$\mathbf{x}_i^{(j)} = V_j \mathbf{z}_i^{(j)} \in \mathbb{C}^n, \quad \text{where} \quad H_j \mathbf{z}_i^{(j)} = \mathbf{z}_i^{(j)} \lambda_i^{(j)}, \quad \mathbf{z}_i^{(j)} \neq \mathbf{0},$$

is an approximate eigenvector of C corresponding to the approximate eigenvalue $\lambda_i^{(j)}$ of C.

8 Linear Systems of Equations

In this section, we consider the problem of solving large systems of linear equations,

$$C\mathbf{x} = \mathbf{b},$$

where $C \in \mathbb{C}^{n\times n}$ is a nonsingular matrix and $\mathbf{b} \in \mathbb{C}^n$. We assume that any possible preconditioning was already applied, and so in general, C is a preconditioned version of the original coefficient matrix. In particular, the matrix C may actually be dense. However, we assume that matrix-vector products with C and possibly C^T are fast. This is the case when C is a preconditioned version of a sparse matrix A and a preconditioner A_0 that allows a sparse LU or Cholesky factorization.

Facts:

The following facts can be found in [FGN92] or [Saa03].

1. Let $\mathbf{x}_0 \in \mathbb{C}^n$ be an arbitrary initial guess for the solution of the linear system, and denote by

   $$\mathbf{r}_0 = \mathbf{b} - C\mathbf{x}_0$$

   the corresponding residual vector. A Krylov subspace-based iterative method for the solution of the above linear system constructs a sequence of approximate solutions of the form

   $$\mathbf{x}_j \in \mathbf{x}_0 + K_j(C, \mathbf{r}_0), \quad j = 1, 2, \ldots,$$

   i.e., the jth iterate is an additive correction of the initial guess, where the correction is chosen from the jth Krylov subspace $K_j(C, \mathbf{r}_0)$ induced by the coefficient matrix C and the initial residual $\mathbf{r}_0$. Now let $V_j \in \mathbb{C}^{n\times j}$ be a matrix the columns of which form a nested basis for $K_j(C, \mathbf{r}_0)$. Then, any possible jth iterate can be parametrized in the form

   $$\mathbf{x}_j = \mathbf{x}_0 + V_j\mathbf{z}_j, \quad \text{where} \quad \mathbf{z}_j \in \mathbb{C}^j.$$

   Moreover, the corresponding residual vector is given by

   $$\mathbf{r}_j = \mathbf{b} - C\mathbf{x}_j = \mathbf{r}_0 - CV_j\mathbf{z}_j.$$

Different Krylov subspace-based iterative methods are then obtained by specifying the choice of the basis matrix V_j and the choice of the parameter vector $\mathbf{z}_j$.

2. The **biconjugate gradient algorithm** (BCG) [Lan52] employs the nonsymmetric Lanczos process to generate nested bases for the right Krylov subspaces $K_j(C, \mathbf{r}_0)$ and the left Krylov subspaces $K_j(C^T, \mathbf{l})$. Here, $\mathbf{l} \in \mathbb{C}^n$, $\mathbf{l} \neq \mathbf{0}$, is an arbitrary nonzero starting vector. The biorthogonality of the right and left Lanczos vectors is exploited to construct the jth iterate $\mathbf{x}_j$ such that the corresponding residual vector $\mathbf{r}_j$ is orthogonal to the left Lanczos vectors, i.e., $W_j^T \mathbf{r}_j = \mathbf{0}$. Using the recurrence relations of the Lanczos process and the above relation for $\mathbf{r}_j$, one can show that the defining condition $W_j^T \mathbf{r}_j = \mathbf{0}$ is equivalent to $\mathbf{z}_j$ being the solution of the linear system

$$T_j \mathbf{z}_j = \mathbf{e}_1^{(j)} \rho_1,$$

where $\mathbf{e}_1^{(j)}$ denotes the first unit vector of length j. Moreover, the corresponding iterates $\mathbf{x}_j$ can be obtained via a simple update from the previous iterate $\mathbf{x}_{j-1}$, resulting in an elegant overall computational procedure. Unfortunately, in general, it cannot be guaranteed that all Lanczos matrices T_j are nonsingular. As a result, BCG iterates $\mathbf{x}_j$ may not exist for every j. More precisely, BCG breaks down if T_j is singular, and it exhibits erratic convergence behavior when T_j is nearly singular.

3. The possible breakdowns and the erratic convergence behavior can be avoided by replacing the $j \times j$ linear system $T_j \mathbf{z}_j = \mathbf{e}_1^{(j)} \rho_1$ by the $(j+1) \times j$ least-squares problem

$$\min_{\mathbf{z} \in \mathbb{C}^j} \| \mathbf{e}_1^{(j+1)} \rho_1 - T_j^{(e)} \mathbf{z} \|_2.$$

Since $T_j^{(e)} \in \mathbb{C}^{(j+1) \times j}$ always has full rank j, the above least-squares problem has a unique solution $\mathbf{z}_j$. The resulting iterative procedure is the **quasi-minimal residual method** (QMR) [FN91].

4. The **generalized minimal residual algorithm** (GMRES) [SS86] uses the Arnoldi process to generate orthonormal basis vectors for the Krylov subspaces $K_j(C, \mathbf{r}_0)$. The orthonormality of the columns of the Arnoldi basis matrix V_j allows to choose $\mathbf{z}_j$ such

that the residual vector $\mathbf{r}_j$ has the smallest possible norm, i.e.,

$$\|\mathbf{r}_j\|_2 = \|\mathbf{r}_0 - CV_j\mathbf{z}_j\|_2 = \min_{\mathbf{z}\in\mathbb{C}^j} \|\mathbf{r}_0 - CV_j\mathbf{z}\|_2.$$

Using the compact form of the recurrence relations used to generate the Arnoldi vectors, one readily verifies that the above minimal residual property is equivalent to $\mathbf{z}_j$ being the solution of the least-squares problem

$$\min_{\mathbf{z}\in\mathbb{C}^j} \|\mathbf{e}_1^{(j+1)}\rho_1 - H_j^{(e)}\mathbf{z}\|_2,$$

where $H_j^{(e)} \in \mathbb{C}^{(j+1)\times j}$ is an upper Hessenberg matrix.

5. The idea of quasi-minimization of the residual vector can also be applied to Lanczos-type iterations that, in each jth step, perform two matrix-vector products with C, instead of one product with C and one product with C^T. The resulting algorithm is called the **transpose-free quasi-minimal residual method** (TFQMR) [Fre93]. We stress that QMR and TFQMR produce different sequences of iterates, and thus QMR and TFQMR are not mathematically equivalent algorithms.

9 Dimension Reduction of Linear Dynamical Systems

In this section, we discuss the application of the nonsymmetric Lanczos process to a large-scale matrix problem that arises in dimension reduction of time-invariant linear dynamical systems. A more detailed description can be found in [Fre03].

Definitions:

Let A, $E \in \mathbb{C}^{n\times n}$. The matrix pencil $A - sE$, $s \in \mathbb{C}$, is said to be **regular** if the matrix $A - sE$ is singular only for finitely many values $s \in \mathbb{C}$.

A **single-input single-output time-invariant linear dynamical system** is a system of differential-algebraic equations (DAEs) of the form

$$\begin{aligned} E\frac{d}{dt}\mathbf{x} &= A\mathbf{x} + \mathbf{b}u(t), \\ y(t) &= \mathbf{l}^T\mathbf{x}(t), \end{aligned}$$

together with suitable initial conditions. Here, $A,\ E \in \mathbb{C}^{n\times n}$ are given matrices such that $A - sE$ is a regular matrix pencil, $\mathbf{b} \in \mathbb{C}^n$, $\mathbf{b} \neq \mathbf{0}$, and $\mathbf{l} \in \mathbb{C}^n$, $\mathbf{l} \neq \mathbf{0}$, are given nonzero vectors, $\mathbf{x}(t) \in \mathbb{C}^n$ is the vector of **state variables**, $u(t) \in \mathbb{C}$ is the given input function, $y(t) \in \mathbb{C}$ is the output function, and n is the **state-space dimension**.

The rational function

$$H\ :\ \mathbb{C} \mapsto \mathbb{C} \cup \infty, \quad H(s) := \mathbf{l}^T \left(sE - A\right)^{-1} \mathbf{b},$$

is called the **transfer function** of the above time-invariant linear dynamical system.

A **reduced-order model** of state-space dimension j $(< n)$ of the above system is a single-input single-output time-invariant linear dynamical system of the form

$$\begin{aligned} E_j \frac{d}{dt}\mathbf{z} \ &= A_j \mathbf{z} + \mathbf{b}_j u(t), \\ y(t) \ &= \mathbf{l}_j^T \mathbf{z}(t), \end{aligned}$$

where $A_j,\ E_j \in \mathbb{C}^{j\times j}$ and $\mathbf{b}_j,\ \mathbf{l}_j \in \mathbb{C}^j$, together with suitable initial conditions.

Let $s_0 \in \mathbb{C}$ be such that the matrix $A - s_0 E$ is nonsingular. A reduced-order model of state-space dimension j of the above system is said to be a **Padé model** about the expansion point s_0 if the matrices A_j, E_j and the vectors $\mathbf{b}_j$, $\mathbf{l}_j$ are chosen such that the Taylor expansions about s_0 of the transfer function H of the original system and of the reduced-order transfer function

$$H_j\ :\ \mathbb{C} \mapsto \mathbb{C} \cup \infty, \quad H_j(s) := \mathbf{l}_j^T \left(sE_j - A_j\right)^{-1} \mathbf{b}_j,$$

agree in as many leading Taylor coefficients as possible, i.e.,

$$H_j(s) = H(s) + \mathcal{O}\Big((s - s_0)^{q(j)}\Big),$$

where $q(j)$ is as large as possible.

Facts:

The following facts can be found in [FF94], [FF95], or [Fre03].

1. In the 'generic' case, $q(j) = 2j$.

2. In the general case, $q(j) \geq 2j$; the case $q(j) > 2j$ occurs only in certain degenerate situations.

3. The transfer function H can be rewritten in terms of a single square matrix $C \in \mathbb{C}^{n \times n}$ as follows:

$$H(s) = \mathbf{l}^T \Big(I_n + (s - s_0)C\Big)^{-1} \mathbf{r}, \quad \text{where} \quad C := (s_0 E - A)^{-1} E, \quad \mathbf{r} := (s_0 E - A)^{-1} \mathbf{b}.$$

   Note that the matrix C can be viewed as a preconditioned version of the matrix E using the 'shift-and-invert' preconditioner $s_0 E - A$.

4. In many cases, the state-space dimension n of the original time-invariant linear dynamical system is very large, but the large square matrices A, $E \in \mathbb{C}^{n \times n}$ are sparse. Furthermore, these matrices are usually such that sparse LU factorizations of the shift-and-invert preconditioner $s_0 E - A$ can be computed with limited amounts of fill-in. In this case, matrix-vector products with the preconditioned matrix C and its transpose C^T are fast.

5. The above definition of Padé models suggests the computation of these reduced-order models by first explicitly generating the leading $q(j)$ Taylor coefficients of H about the expansion point s_0, and then constructing the Padé model from these. However, this process is extremely ill-conditioned and numerically unstable; see the discussion in [FF94, FF95].

6. A much more stable way to compute Padé models without explicitly generating the Taylor coefficients is based on the nonsymmetric Lanczos process. The procedure is simply as follows. One uses the vectors $\mathbf{r}$ and $\mathbf{l}$ from the above representation of the transfer function H as right and left starting vectors, and applies the nonsymmetric Lanczos process to the preconditioned matrix C. After j iterations, the algorithm has produced the $j \times j$ tridiagonal Lanczos matrix T_j. The reduced-order model defined by

$$A_j := s_0 T_j - I_j, \quad E_j := T_j, \quad \mathbf{b}_j := (\mathbf{l}^T \mathbf{r})\, \mathbf{e}_1^{(j)}, \quad \mathbf{l}_j := \mathbf{e}_1^{(j)}$$

   is a Padé model of state-space dimension j about the expansion point s_0. Here $\mathbf{e}_1^{(j)}$ denotes the first unit vector of length j.

7. In the large-scale case, Padé models of state-space dimension $j \ll n$ often provide very accurate approximations of the original system of state-space dimension n. In particular, this is the case for applications in VLSI circuit simulation; see [FF95, Fre03] and the references given there.

8. Multiple-input multiple-output time-invariant linear dynamical systems are extensions of the above single-input single-output case with the vectors $\mathbf{b}$ and $\mathbf{l}$ replaced by matrices $B \in \mathbb{C}^{n\times m}$ and $L \in \mathbb{C}^{n\times p}$, respectively, where m is the number of inputs and p is the number of outputs. The approach outlined in this section can be extended to the general multiple-input multiple-output case. A suitable extension of the nonsymmetric Lanczos process that can handle multiple right and left starting vectors is needed in this case. For a discussion of such a Lanczos-type algorithm and its application in dimension reduction of general multiple-input multiple-output time-invariant linear dynamical systems, we refer the reader to [Fre03] and the references given there.

References

[Arn51] W. E. Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quart. Appl. Math.*, 9:17–29, 1951.

[BDD00] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide.* SIAM Publications, Philadelphia, Pennsylvania, 2000.

[CW85] J. K. Cullum and R. A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Volume 1, Theory.* Birkhäuser, Basel, Switzerland, 1985.

[CW86] J. K. Cullum and R. A. Willoughby. A practical procedure for computing eigenvalues of large sparse nonsymmetric matrices. In J. K. Cullum and R. A. Willoughby, editors, *Large Scale Eigenvalue Problems*, pages 193–240. North-Holland, Amsterdam, The Netherlands, 1986.

[DER89] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, Oxford, United Kingdom, 1989.

[FF94] P. Feldmann and R. W. Freund. Efficient linear circuit analysis by Padé approximation via the Lanczos process. In *Proceedings of EURO-DAC '94 with EURO-VHDL '94*, pages 170–175, Los Alamitos, California, 1994. IEEE Computer Society Press.

[FF95] P. Feldmann and R. W. Freund. Efficient linear circuit analysis by Padé approximation via the Lanczos process. *IEEE Trans. Computer-Aided Design*, 14:639–649, 1995.

[Fre93] R. W. Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14:470–482, 1993.

[Fre03] R. W. Freund. Model reduction methods based on Krylov subspaces. *Acta Numerica*, 12:267–319, 2003.

[FGN92] R. W. Freund, G. H. Golub, and N. M. Nachtigal. Iterative solution of linear systems. *Acta Numerica*, 1:57–100, 1992.

[FGN93] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. *SIAM J. Sci. Comput.*, 14:137–158, 1993.

[FN91] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.

[GMS05] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Rev.*, 47:99–131, 2005.

[Hou75] A. S. Householder. *The Theory of Matrices in Numerical Analysis.* Dover Publications, New York, 1975.

[KR91] N. K. Karmarkar and K. G. Ramakrishnan. Computational results of an interior point algorithm for large scale linear programming. *Math. Programming*, 52:555–586, 1991.

[Lan50] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Standards*, 45:255–282, 1950.

[Lan52] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bur. Standards*, 49:33–53, 1952.

[LM05] A. N. Langville and C. D. Meyer. A survey of eigenvector methods for web information retrieval. *SIAM Rev.*, 47(1):135–161, 2005.

[NW99] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, New York, New York, 1999.

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM Publications, Philadelphia, Pennsylvania, second edition, 2003.

[SS86] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7(3):856–869, 1986.

[SB02] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, New York, third edition, 2002.

[vdV03] H. A. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, Cambridge, 2003.

[Wri97] S. J. Wright. *Primal-dual interior-point methods*. SIAM Publications, Philadelphia, Pennsylvania, 1997.