Randomized methods for matrix computations and analysis of high dimensional data

Per-Gunnar Martinsson

July 5, 2016.

Contents

CHAPTER 1

Matrix factorizations and low rank approximation

The first chapter provides a quick review of basic concepts from linear algebra that we will use frequently. Note that the pace is fast here, and assumes that you have seen these concepts in prior course-work. If not, then additional reading on the side is strongly recommended!

1.1. Notation, etc

1.1.1. Norms. Let $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ denote a vector in $\mathbb{R}^n$ or $\mathbb{C}^n$. Our default norm for vectors is the Euclidean norm

$$\|\mathbf{x}\| = \left(\sum_{j=1}^{n} |x_j|^2\right)^{1/2}.$$

We will at times also use ℓ^p norms

$$\|\mathbf{x}\|_p = \left(\sum_{j=1}^{n} |x_j|^p\right)^{1/p}.$$

Let $\mathbf{A}$ denote an $m \times n$ matrix. For the most part, we allow $\mathbf{A}$ to have complex entries. We define the *spectral norm* of $\mathbf{A}$ via

$$\|\mathbf{A}\| = \sup_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\| = \sup_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|}.$$

We define the *Frobenius norm* of $\mathbf{A}$ via

$$\|\mathbf{A}\|_{\mathrm{F}} = \left(\sum_{i=1}^{m}\sum_{j=1}^{n} |\mathbf{A}(i,j)|^2\right)^{1/2}.$$

Observe that

$$\|\mathbf{A}\| \leq \|\mathbf{A}\|_{\mathrm{F}} \leq \sqrt{\min(m,n)}\,\|\mathbf{A}\|.$$

1.1.2. Transpose and adjoint. Given an $m \times n$ matrix $\mathbf{A}$, the *transpose* $\mathbf{A}^{\mathrm{t}}$ is the $n \times m$ matrix $\mathbf{B}$ with entries

$$\mathbf{B}(i,j) = \mathbf{A}(j,i).$$

The transpose is most commonly used for *real* matrices. It can also be used for a *complex* matrix, but more typically, we then use the *adjoint*, which is the complex conjugate of the transpose

$$\mathbf{A}^* = \overline{\mathbf{A}^{\mathrm{t}}}.$$

1.1.3. Subspaces. Let $\mathbf{A}$ be an $m \times n$ matrix.

- The *row space* of $\mathbf{A}$ is denoted $\mathrm{row}(\mathbf{A})$ and is defined as the subspace of $\mathbb{R}^n$ spanned by the rows of $\mathbf{A}$.
- The *column space* of $\mathbf{A}$ is denoted $\mathrm{col}(\mathbf{A})$ and is defined as the subspace of $\mathbb{R}^m$ spanned by the columns of $\mathbf{A}$. The column space is the same as the *range* or $\mathbf{A}$, so $\mathrm{col}(\mathbf{A}) = \mathrm{ran}(\mathbf{A})$.
- The *nullspace* or *kernel* of $\mathbf{A}$ is the subspace $\ker(\mathbf{A}) = \mathrm{null}(\mathbf{A}) = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{A}\mathbf{x} = \mathbf{0}\}$.

1.1.4. Special classes of matrices. We use the following terminology to classify matrices:

- An $m \times n$ matrix $\mathbf{A}$ is *orthonormal* if its columns form an orthonormal basis, i.e. $\mathbf{A}^*\mathbf{A} = \mathbf{I}$.
- An $n \times n$ matrix $\mathbf{A}$ is *normal* if $\mathbf{A}\mathbf{A}^* = \mathbf{A}^*\mathbf{A}$.
- An $n \times n$ real matrix $\mathbf{A}$ is *symmetric* if $\mathbf{A}^{\mathrm{t}} = \mathbf{A}$.
- An $n \times n$ matrix $\mathbf{A}$ is *self-adjoint* if $\mathbf{A}^* = \mathbf{A}$.
- An $n \times n$ matrix $\mathbf{A}$ is *skew-adjoint* if $\mathbf{A}^* = -\mathbf{A}$.
- An $n \times n$ matrix $\mathbf{A}$ is *unitary* if it is invertible and $\mathbf{A}^* = \mathbf{A}^{-1}$.

Suppose that $\mathbf{A}$ is an $n \times n$ self-adjoint matrix. Then for any $x \in \mathbb{C}^n$, one can easily verify that $\mathbf{x}^*\mathbf{A}\mathbf{x}$ is a *real* scalar number. We then say that $\mathbf{A}$ is *non-negative* if $\mathbf{x}^*\mathbf{A}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{C}^n$. The properties *positive*, *non-positive*, and *negative* are defined analogously.

1.2. Low rank approximation

1.2.1. Exact rank deficiency. Let $\mathbf{A}$ be an $m \times n$ matrix. Let k denote an integer between 1 and $\min(m, n)$. Then the following conditions are equivalent:

- The columns of $\mathbf{A}$ span a subspace of $\mathbb{R}^m$ of dimension k.
- The rows of $\mathbf{A}$ span a subspace of $\mathbb{R}^n$ of dimension k.
- The nullspace of $\mathbf{A}$ has dimension $n - k$.
- The nullspace of $\mathbf{A}^*$ has dimension $m - k$.

If $\mathbf{A}$ satisfies any of these criteria, then we say that $\mathbf{A}$ *has rank* k. When $\mathbf{A}$ has rank k, it is possible to find matrices $\mathbf{E}$ and $\mathbf{F}$ such that

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{E}} \underset{k \times n}{\mathbf{F}}.$$

The columns of $\mathbf{E}$ span the column space of $\mathbf{A}$, and the rows of $\mathbf{F}$ span the row space of $\mathbf{A}$. Having access to such factors $\mathbf{E}$ and $\mathbf{F}$ can be very helpful:

- Storing $\mathbf{A}$ requires mn words of storage.
  Storing $\mathbf{E}$ and $\mathbf{F}$ requires $km + kn$ words of storage.
- Given a vector $\mathbf{x}$, computing $\mathbf{A}\mathbf{x}$ requires mn flops.
  Given a vector $\mathbf{x}$, computing $\mathbf{A}\mathbf{x} = \mathbf{E}(\mathbf{F}\mathbf{x})$ requires $km + kn$ flops.
- The factors $\mathbf{E}$ and $\mathbf{F}$ are often useful for *data interpretation.*

In practice, we often impose conditions on the factors. For instance, in the well known QR decomposition, the columns of $\mathbf{E}$ are orthonormal, and $\mathbf{F}$ is upper triangular (up to permutations of the columns).

1.2.2. Approximate rank deficiency. The condition that $\mathbf{A}$ has *precisely* rank k is of high theoretical interest, but is not realistic in practical computations. Frequently, the numbers we use have been measured by some device with finite precision, or they may have been computed via a simulation with some approximation errors (e.g. by solving a PDE numerically). In any case, we almost always work with data that is stored in some finite precision format (typically about 10^{-15}). For all these reasons, it is very useful to define the concept of *approximate rank.* In this course, we will typically use the following definition:

DEFINITION 1. Let $\mathbf{A}$ be an $m \times n$ matrix, and let ε be a positive real number. We then define the *ε-rank of* $\mathbf{A}$ as the unique integer k such that both the following two conditions hold:

(a) There exists a matrix $\mathbf{B}$ of precise rank k such that $\|\mathbf{A} - \mathbf{B}\| \leq \varepsilon$.
(b) There does not exist any matrix $\mathbf{B}$ of rank less than k such that (a) holds

The term ε-rank is sometimes used without enforcing condition (b): We sometimes say that $\mathbf{A}$ has ε-rank k if

$$\inf\{\|\mathbf{A} - \mathbf{B}\| : \mathbf{B} \text{ has rank } k\} \leq \varepsilon,$$

without worrying about whether the rank could actually be smaller. In other words, we sometimes say "$\mathbf{A}$ has ε-rank k" when we really mean "$\mathbf{A}$ has ε-rank at most k."

1.3. The eigenvalue decomposition

Let $\mathbf{A}$ be an $n \times n$ matrix (it must be *square* for eigenvalues and eigenvectors to exist). We then say that λ is an eigenvalue and $\mathbf{v}$ is an eigenvector of $\mathbf{A}$ if $\mathbf{v} \neq \mathbf{0}$ and

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}.$$

THEOREM 1. *Let $\mathbf{A}$ be an $n \times n$ matrix. Then $\mathbf{A}$ is normal (meaning that $\mathbf{A}\mathbf{A}^* = \mathbf{A}^*\mathbf{A}$) if and only if $\mathbf{A}$ admits a factorization of the form*

$$\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^* \tag{1.1}$$

where $\mathbf{V}$ is unitary and $\mathbf{D}$ is diagonal.

The equation (1.1) can alternatively be written

$$\mathbf{A} = \sum_{j=1}^{n} \lambda_j \, \mathbf{v}_j \, \mathbf{v}_j^*,$$

where $\{\lambda_j, \mathbf{v}_j\}$ are the *eigenpairs* of $\mathbf{A}$, and

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{bmatrix},$$
$$\mathbf{D} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}.$$

In other words, the columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{A}$. These eigenvectors form an orthonormal basis for $\mathbb{C}^n$. In the basis $\{\mathbf{v}_j\}_{j=1}^n$, the matrix $\mathbf{A}$ is diagonal.

Recall that *self-adjoint*, *skew-adjoint*, and *unitary* matrices are special cases of *normal* matrices, so these classes all allow spectral decompositions. It is easy to verify that:

$\mathbf{A}$ is self-adjoint	$\Leftrightarrow$	$\mathbf{A}^* = \mathbf{A}$	$\Leftrightarrow$	Every eigenvalue is real.
$\mathbf{A}$ is skew-adjoint	$\Leftrightarrow$	$\mathbf{A}^* = -\mathbf{A}$	$\Leftrightarrow$	Every eigenvalue is imaginary.
$\mathbf{A}$ is unitary	$\Leftrightarrow$	$\mathbf{A}^* = \mathbf{A}^{-1}$	$\Leftrightarrow$	Every eigenvalue satisfies $\lvert\lambda_j\rvert = 1$.

Note that even a matrix whose entries are all real may have complex eigenvalues and eigenvectors.

What about non-normal matrices? Every square matrix has at least one (possibly complex) eigenvalue and one eigenvector. But if $\mathbf{A}$ is not normal, then there is no orthonormal basis consisting of eigenvectors. While eigenvalue decompositions of non-normal matrices are still very useful for certain applications, the lack of an ON-basis consisting of eigenvectors typically makes the *singular value decomposition* a much better tool for low-rank approximation in this case.

1.4. The Singular Value Decomposition

1.4.1. Definition of full SVD. Let $\mathbf{A}$ be an $m \times n$ matrix (*any* matrix, it can be rectangular, complex or real valued, etc). Set $p = \min(m, n)$. Then $\mathbf{A}$ admits a factorization

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times p}{\mathbf{U}} \; \underset{p \times p}{\mathbf{D}} \; \underset{p \times n}{\mathbf{V}^*}, \tag{1.2}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal, and where $\mathbf{D}$ is diagonal. We write these out as

$$\mathbf{U} = \begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_p\end{bmatrix},$$
$$\mathbf{V} = \begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_p\end{bmatrix},$$
$$\mathbf{D} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \sigma_p \end{bmatrix}.$$

The vectors $\{\mathbf{u}_j\}_{j=1}^p$ are the *left singular vectors* and the vectors $\{\mathbf{v}_j\}_{j=1}^p$ are the *right singular vectors*. These form orthonormal bases of the ranges of $\mathbf{A}$ and $\mathbf{A}^*$, respectively. The values $\{\sigma_j\}_{j=1}^p$ are the *singular values* of $\mathbf{A}$. These are customarily ordered so that

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_p \geq 0.$$

The SVD (1.2) can alternatively be written as a decomposition of $\mathbf{A}$ as a sum of p "outer products" of vectors

$$\mathbf{A} = \sum_{j=1}^{p} \sigma_j \, \mathbf{u}_j \, \mathbf{v}_j^*.$$

1.4.2. Low rank approximation via SVD. For purposes of approximating a given matrix by a matrix of low rank, the SVD is in a certain sense *optimal*. To be precise, suppose that we are given a matrix $\mathbf{A}$, and have computed its SVD (1.2). Then for an integer $k \in \{1, 2, \ldots, p\}$, we define

$$\mathbf{A}_k = \sum_{j=1}^{k} \sigma_j \, \mathbf{u}_j \, \mathbf{v}_j^*.$$

Clearly $\mathbf{A}_k$ is a matrix of rank k. It is in fact the particular rank-k matrix that best approximates $\mathbf{A}$:

$$\|\mathbf{A} - \mathbf{A}_k\| = \inf\{\|\mathbf{A} - \mathbf{B}\| : \mathbf{B} \text{ has rank } k\},$$
$$\|\mathbf{A} - \mathbf{A}_k\|_{\mathrm{F}} = \inf\{\|\mathbf{A} - \mathbf{B}\|_{\mathrm{F}} : \mathbf{B} \text{ has rank } k\}.$$

If $k < p$, then it is easily verified that the minimal residuals evaluate to

$$\|\mathbf{A} - \mathbf{A}_k\| = \sigma_{k+1},$$
$$\|\mathbf{A} - \mathbf{A}_k\|_{\mathrm{F}} = \left(\sum_{j=k+1}^{p} \sigma_j^2 \right)^{1/2}.$$

REMARK 1.1. For *normal* matrices, a truncated eigenvalue decomposition attains exactly the same approximation error as a truncated SVD, so for this particular class of matrices, either decomposition can be used. (But note that the EVD could be complex even for a real matrix.)

1.4.3. Connection between the SVD and the eigenvalue decomposition. Let $\mathbf{A}$ be a general matrix of size $m \times n$. Then form the $m \times m$ matrix

$$\mathbf{B} = \mathbf{A}\mathbf{A}^*.$$

Observe that $\mathbf{B}$ is self-adjoint and non-negative, so there exist m orthonormal eigenvector $\{\mathbf{u}_j\}_{j=1}^m$ with associated *real* eigenvalues $\{\lambda_j\}_{j=1}^m$. Then $\{\mathbf{u}_j\}_{j=1}^m$ are left singular vectors of $\mathbf{A}$ associated with singular values $\sigma_j = \sqrt{\lambda_j}$. Analogously, if we form the $n \times n$ matrix

$$\mathbf{C} = \mathbf{A}^*\mathbf{A},$$

then the eigenvectors of $\mathbf{C}$ are right singular vectors of $\mathbf{A}$.

1.4.4. Computing the SVD. One can prove that in general computing the singular values (or eigenvalues) of an $n \times n$ matrix *exactly* is an equivalent problem to solving an n'th order polynomial (the characteristic equation). Since this problem is known to not, in general, have an algebraic solution for $n \geq 5$, it is not surprising that algorithms for computing singular values are iterative in nature. However, they typically converge very fast, so for practical purposes, algorithms for computing a full SVD perform quite similarly to algorithms for computing full QR decompositions. For details, we refer to standard textbooks [**7, 19**], but some facts about these algorithms are relevant to what follows:

- Standard algorithms for computing the SVD of a dense $m \times n$ matrix (as found in Matlab, LAPACK, etc), have practical asymptotic speed of $O(m\,n\,\min(m,n))$.
- While the asymptotic complexity of algorithms for computing full factorizations tend to be $O(m\,n\,\min(m,n))$ regardless of the choice of factorization (LU, QR, SVD, etc), the scaling constants are different. In particular pivoted QR is slower than non-pivoted QR, and the SVD is even slower.
- Standard library functions for computing the SVD almost always produce results that are accurate to full double precision accuracy. They can fail to converge for certain matrices, but in high-quality software, this happens very rarely.
- Standard algorithms are challenging to parallelize well. For a small number of cores on a modern CPU (as of 2016) they work well, but performance deteriorates as the number of cores increase, or if the computation is to be carried out on a GPU or a distributed memory machine.

1.5. The QR factorization

Let $\mathbf{A}$ be an $m \times n$ matrix. Set $p = \min(m,n)$. Let $\{\mathbf{a}_j\}_{j=1}^n$ denote the columns of $\mathbf{A}$,

$$\mathbf{A} = \big[\mathbf{a}_1\ \mathbf{a}_2\ \cdots\ \mathbf{a}_n\big].$$

Our objective is now to find an orthonormal set of vectors $\{\mathbf{q}_j\}_{j=1}^p$ that form a "good" set of basis vectors for expressing the columns of $\mathbf{A}$. In other words, if we set

$$\mathbf{Q}_k = \big[\mathbf{q}_1\ \mathbf{q}_2\ \cdots\ \mathbf{q}_k\big],$$

then we want

$$\|\mathbf{A} - \mathbf{Q}_k\mathbf{Q}_k^*\mathbf{A}\| \approx \inf\{\|\mathbf{A} - \mathbf{B}\| :\ \mathbf{B} \text{ has rank } k\} = \sigma_{k+1}.$$

We recall that the *optimal* basis (in this sense) is the left singular vectors of $\mathbf{A}$. However, these are tricky to compute, and we seek a simpler and faster algorithm that leads to *close to* optimal basis vectors.

1.5.1. The Gram-Schmidt process. The Gram-Schmidt process is a simple "greedy" algorithm that can be coded efficiently. (Or at least reasonably efficiently, see Section 1.5.5.) Informally speaking, the idea is to take the collection of vectors $\{\mathbf{a}_j\}_{j=1}^n$, grab the largest one, normalize it to make its length one, and then use the resulting vector as the first basis vector. Then project the remaining $n-1$ vectors away from the one that was first chosen. Then take the largest vector of the remaining ones, normalize it to form the second basis vector, project the remaining $n-2$ vectors away from the new basis vector, etc. The resulting algorithm is shown in Figure 1.1.

REMARK 1.2 (Break-down for rank-deficient matrices). The process shown in Figure 1.1 will break down if the matrix has exact rank that is less than $\min(m,n)$. In this case, the residual matrix $\mathbf{A}$ will at some point be exactly zero. For purposes of formulating a mathematically correct algorithm, all one needs to do is to introduce a stopping criterion that breaks the loop if this happens. In practice, since all computations are carried out in finite precision, the residual is very unlikely to ever be *exactly* zero. However, when the residual gets small, round-off errors will create serious loss of accuracy. Techniques that overcome this problem are described in Section 1.5.2.

(1) $\mathbf{Q}_0 = [\,]$; $\mathbf{R}_0 = [\,]$; $\mathbf{A}_0 = \mathbf{A}$; $p = \min(m, n)$;
(2) **for** $j = 1 : p$
(3) $\quad i_j = \text{argmin}\{\|\mathbf{A}(:, \ell)\| : \ell = 1, 2, \ldots, n\}$
(4) $\quad \mathbf{q} = \mathbf{A}(:, i)/\|\mathbf{A}(:, i)\|$.
(5) $\quad \mathbf{r} = \mathbf{q}^*\mathbf{A}_{j-1}$
(6) $\quad \mathbf{Q}_j = [\mathbf{Q}_{j-1}\ \mathbf{q}]$
(7) $\quad \mathbf{R}_j = \begin{bmatrix} \mathbf{R}_{j-1} \\ \mathbf{r} \end{bmatrix}$
(8) $\quad \mathbf{A}_j = \mathbf{A}_{j-1} - \mathbf{q}\mathbf{r}$
(9) **end for**
(10) $\mathbf{Q} = \mathbf{Q}_p$; $\mathbf{R} = \mathbf{R}_p$;

FIGURE 1.1. The basic Gram Schmidt process. Given an input matrix $\mathbf{A}$, the algorithm computes an ON matrix $\mathbf{Q}$ and a "morally" upper triangular matrix $\mathbf{R}$ such that $\mathbf{A} = \mathbf{Q}\mathbf{R}$. At the intermediate steps, we have $\mathbf{A} = \mathbf{A}_j + \mathbf{Q}_j\mathbf{R}_j$. Moreover at any step k, the columns of $\mathbf{Q}(:, 1 : k)$ form an ON basis for the space spanned by the pivot vectors $\mathbf{A}(:, [i_1, i_2, \ldots, i_k])$.

1.5.2. The QR factorization. Starting with the simplistic process described in Section (1.5.1), we will make two sets of improvements:

(1) *Improving numerical accuracy:* The method described in Figure 1.1 works perfectly when executed in exact arithmetic. However, for large matrices, the basis vectors generated tend to lose orthonormality as the computation proceeds. To avoid this, we perform an additional re-orthonormalization step. Specifically, after line (4), one should insert two new lines:

$$\text{(4')} \qquad \mathbf{q} = \mathbf{q} - \mathbf{Q}_{j-1}\,(\mathbf{Q}_{j-1}^*\mathbf{q}).$$
$$\text{(4")} \qquad \mathbf{q} = \mathbf{q}/\|\mathbf{q}\|.$$

These additional steps would make no difference in exact arithmetic, but they are important in practical computations. Without them, you often lose orthonormality in the basis vectors, which greatly degrades the utility of the computed factorization.

(2) *Better pivoting:* In practice, it is convenient to explicitly swap out the pivot vector you choose at step k to the k'th column in the matrix $\mathbf{Q}$ that you build. One must also do the analogous swap in the $\mathbf{R}$ matrix being built. (Moreover, it is possible to improve computational speed by accelerating the pivot selection step on line (3). The idea is to maintain a vector of length $1 \times n$ that holds the sum of the squares of all remaining residuals. This vector can be cheaply downdated at the end of the loop, which saves us the need to compute the norms of the columns of $\mathbf{Q}$. See [**7**, Sec. 5.4.1].)

(3) *Improved book-keeping:* The algorithm as written is wasteful of memory. One reasonably storage efficient way of implementing the method is to define at the outset a new matrix $\mathbf{Q}$ of size $m \times n$ and simply copying $\mathbf{A}$ over to $\mathbf{Q}$. This matrix $\mathbf{Q}$ is used to hold both the new basis vectors that are stored in $\mathbf{Q}_k$ in Figure 1.1 and the residual columns that are stored in $\mathbf{A}_k$ in Figure 1.1. Observe that in each step, we zero out one residual vector, and create one new basis vector, so there is a perfect match. To be precise, at the end of the k'th step, the matrix $\mathbf{Q}$ holds $\mathbf{Q} = \big[\mathbf{Q}_k\ \tilde{\mathbf{A}}_k\big]$, where $\mathbf{Q}_k$ holds the computed k columns of $\mathbf{Q}$, and $\tilde{\mathbf{A}}_k$ holds the remaining $n - k$ residual vectors.

The algorithm resulting from incorporating these improvements is given in Figure 1.2.

REMARK 1.3 (Householder QR). The QR factorization algorithm described in Figure 1.2 can be optimized further. In professional software packages, standard practice is to form the matrix $\mathbf{Q}$ as a product of so called *Householder reflectors.* These provide optimal stability and maintain very high orthonormality among the columns of $\mathbf{Q}$. Further, when Householder reflectors are used, all the information needed to form $\mathbf{R}$ and $\mathbf{Q}$ can be stored in a single array of size $m \times n$, as opposed to the formulation we give where two copies of the array are used. The trick is to use the

Create and initialize the output matrices.
(1) $\mathbf{Q} = \mathbf{A}$; $\mathbf{R} = \texttt{zeros}(\min(m,n), n)$; $J = 1 : n$;
(2) **for** $j = 1 : \min(m, n)$
Find the pivot column i.
(3) $i = \text{argmin}\{\|\mathbf{Q}(:,\ell)\| : \ell = j, j+1, j+2, \ldots, n\}$
Move the chosen pivot column to the j'th slot.
(4) $J([j,i]) = J([i,j])$; $\mathbf{Q}(:,[j,i]) = \mathbf{Q}(:,[i,j])$; $\mathbf{R}(:,[j,i]) = \mathbf{R}(:,[i,j])$;
(5) $\rho = \|\mathbf{Q}(:,j)\|$
(6) $\mathbf{R}(j,j) = \rho$
Perform the "paranoid" reorthonormalization.
(7) $\mathbf{q} = (1/\rho)\,\mathbf{Q}(:,j)$
(8) $\mathbf{q} = \mathbf{q} - \mathbf{Q}(:,1:(j-1))\,(\mathbf{Q}(:,1:(j-1))^*\,\mathbf{q})$
(9) $\mathbf{q} = (1/\|\mathbf{q}\|)\,\mathbf{q}$
(10) $\mathbf{Q}(:,j) = \mathbf{q}$
Compute the expansion coefficients and update $\mathbf{Q}$ and $\mathbf{R}$.
(11) $\mathbf{r} = \mathbf{q}^*\,\mathbf{Q}(:,(j+1):n)$
(12) $\mathbf{R}(j,(j+1):n) = \mathbf{r}$
(13) $\mathbf{Q}(:,(j+1):n) = \mathbf{Q}(:,(j+1):n) - \mathbf{q}\mathbf{r}$
(14) **end for**
If $n > m$, then we need to delete the last columns of $\mathbf{Q}$.
(15) $\mathbf{Q} = \mathbf{Q}(:,1:\min(m,n))$

FIGURE 1.2. QR factorization via Gram Schmidt. The algorithm takes as input an $m \times n$ matrix $\mathbf{A}$. The output is, with $p = \min(m,n)$, an index vector J, an $m \times p$ ON matrix $\mathbf{Q}$, and a $p \times n$ upper triangular matrix $\mathbf{R}$ such that $\mathbf{A}(:,J) = \mathbf{QR}$.

zero elements formed "under the diagonal" in $\mathbf{R}$ to store enough information to uniquely define $\mathbf{Q}$. See [**7**, Sec. 5.4] or [**19**, Lecture 10].

1.5.3. Low rank approximation via the QR factorization. The algorithm for computing the QR factorization can trivially be modified to compute a low rank approximation to a matrix. Consider the algorithm given in Figure 1.2. After k steps of the algorithm, we find that the matrices $\mathbf{Q}$ and $\mathbf{R}$ hold the following entries:

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_k \\ \mathbf{0} \end{bmatrix} \qquad \text{and} \qquad \mathbf{Q} = \begin{bmatrix} \mathbf{Q}_k & \tilde{\mathbf{A}}_k \end{bmatrix}.$$

The matrix $\mathbf{R}_k$ is the matrix of size $k \times n$ holding the top k rows of $\mathbf{R}$, the matrix $\mathbf{Q}_k$ is of size $m \times k$ and holds the first k columns of $\mathbf{Q}$, and the matrix $\tilde{\mathbf{A}}_k$ is of size $m \times (n-k)$ and holds the "remainder" of the columns that have not yet been chosen as pivot columns (in other words, it contains the non-zero columns of the matrix $\mathbf{A}_k$ in Figure 1.1). Then the partial factorization we have computed after k steps reads

$$\mathbf{A}(:,J) = \mathbf{Q}_k\mathbf{R}_k + \begin{bmatrix} \mathbf{0} & \tilde{\mathbf{A}}_k \end{bmatrix}. \tag{1.3}$$

Let $\mathbf{P}_k$ denote the permutation matrix defined by the index vector J so that

$$\mathbf{A}(:,J) = \mathbf{A}\mathbf{P}_k.$$

Then we can rewrite (1.3) as (note that $\mathbf{P}_k^{-1} = \mathbf{P}_k^*$)

$$\mathbf{A} = \mathbf{Q}_k\mathbf{R}_k\mathbf{P}_k^* + \begin{bmatrix} \mathbf{0} & \tilde{\mathbf{A}}_k \end{bmatrix}\mathbf{P}_k^*.$$

The first term has rank k and the second term is the "remainder."

Create and initialize the output matrices.
(*) $\mathbf{Q} = \mathbf{A}$; $\mathbf{R} = \texttt{zeros}(\min(m,n), n)$; $J = 1:n$;
(*) **for** $j = 1 : \min(m, n)$
Find the pivot column i.
(*) $i = \text{argmin}\{\|\mathbf{Q}(:,\ell)\| : \ell = j, j+1, j+2, \ldots, n\}$
(*) $J([j,i]) = J([i,j])$; $\mathbf{Q}(:,[j,i]) = \mathbf{Q}(:,[i,j])$; $\mathbf{R}(:,[j,i]) = \mathbf{R}(:,[i,j])$;
Move the chosen pivot column to the j'th slot.
(*) $\rho = \|\mathbf{Q}(:,j)\|$
(*) $\mathbf{R}(j,j) = \rho$
Perform the "paranoid" reorthonormalization.
(*) $\mathbf{q} = (1/\rho)\,\mathbf{Q}(:,j)$
(*) $\mathbf{q} = \mathbf{q} - \mathbf{Q}(:,1:(j-1))\,(\mathbf{Q}(:,1:(j-1))^*\,\mathbf{q})$
(*) $\mathbf{q} = (1/\|\mathbf{q}\|)\,\mathbf{q}$
(*) $\mathbf{Q}(:,j) = \mathbf{q}$
Compute the expansion coefficients and update $\mathbf{Q}$ and $\mathbf{R}$.
(*) $\mathbf{r} = \mathbf{q}^*\,\mathbf{Q}(:,(j+1):n)$
(*) $\mathbf{R}(j,(j+1):n) = \mathbf{r}$
(*) $\mathbf{Q}(:,(j+1):n) = \mathbf{Q}(:,(j+1):n) - \mathbf{q}\mathbf{r}$
Check the accuracy of the partial factorization.
if $\texttt{sum}(\texttt{sum}(\mathbf{Q}(:,(j+1):n) .* \mathbf{Q}(:,(j+1):n))) \leq \varepsilon^2$ **then break**
(*) **end for**
(*) $k = j$; $\mathbf{Q} = \mathbf{Q}(:,1:k)$; $\mathbf{R} = \mathbf{R}(1:k,:)$;

FIGURE 1.3. Partial QR factorization via Gram Schmidt. The algorithm takes as input an $m \times n$ matrix $\mathbf{A}$ and a tolerance ε. The output is, an index vector J, an $m \times k$ ON matrix $\mathbf{Q}$, and a $k \times n$ upper triangular matrix $\mathbf{R}$ such that $\|\mathbf{A}(:,J) - \mathbf{QR}\|_{\text{F}} \leq \varepsilon$. The integer k is the computed rank, and is an output parameter. (Observe that the computation of the Frobenius norm of the remainder matrix, and the determination of the pivot vectors can be optimized.)

Suppose that we are interested in computing a low rank factorization of $\mathbf{A}$ that is accurate to some precision ε, using the Frobenius norm. Then after the k'th step, we can simply evaluate $\|\tilde{\mathbf{A}}_k\|_{\text{F}}$ and stop when this quantity drops below ε. The resulting algorithm is given in Figure 1.3.

1.5.4. Getting the SVD from the QR factorization. The technique described in Section 1.5.3 results in a factorization of the form

$$\underset{m\times n}{\mathbf{A}} \approx \underset{m\times k}{\mathbf{Q}}\ \underset{k\times n}{\mathbf{R}\mathbf{P}^*} + \underset{m\times n}{\mathbf{E}} \tag{1.4}$$

where $\mathbf{Q}$ is orthonormal, $\mathbf{R}$ is upper triangular, and the "error" or "remainder" matrix $\mathbf{E}$ satisfies

$$\|\mathbf{E}\|_{\text{F}} \leq \varepsilon.$$

(We dropped the subscripts k here.) Suppose now that we seek a partial SVD. It turns out that this can be accomplished through two simple steps:

(1) Compute a full SVD of the matrix $\mathbf{R}\mathbf{P}^*$, which is cheap since $\mathbf{R}$ is small (it has only k rows)

$$\mathbf{R}\mathbf{P}^* = \hat{\mathbf{U}}\,\mathbf{D}\,\mathbf{V}^*.$$

(2) Multiply $\mathbf{Q}$ and $\hat{\mathbf{U}}$ together

$$\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}.$$

Observe that now $\mathbf{U}$ and $\mathbf{V}$ are both orthonormal, $\mathbf{D}$ is diagonal, and

$$\mathbf{A} = \mathbf{Q}\underbrace{\mathbf{R}\mathbf{P}^*}_{=\hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*} + \mathbf{E} = \underbrace{\mathbf{Q}\hat{\mathbf{U}}}_{=\mathbf{U}}\mathbf{D}\mathbf{V}^* + \mathbf{E} = \mathbf{U}\mathbf{D}\mathbf{V}^* + \mathbf{E}. \tag{1.5}$$

We have obtained a partial SVD. Observe in particular that the error term $\mathbf{E}$ is exactly the same in both (1.4) and (1.5).

1.5.5. Blocking of algorithms and execution speed. The various QR factorization algorithms described in this section are extremely powerful and useful. They have been developed over decades, and current implementations are highly accurate, entirely robust, and fairly fast. They suffer from one serious short coming, however, which is that they inherently are formed as a sequence of n low-rank updates to a matrix. The reason this is bad is that on modern computers, the cost of moving data (from RAM to cache, between levels of cache, etc) often exceeds the time to execute flops. As an illustration of this phenomenon, suppose that we are given an $m \times n$ matrix $\mathbf{A}$ and a set of vectors $\{\mathbf{x}_i\}_{i=1}^p$, and that we seek to evaluate the vectors

$$\mathbf{y}_i = \mathbf{A}\,\mathbf{x}_i, \qquad i = 1, 2, \ldots, p.$$

One could either do this via a simple loop:

for $i = 1 : p$
  $\mathbf{y}_i = \mathbf{A}\mathbf{x}_i$
end for

Or, one could put all the vectors in a matrix and simply evaluate a matrix-matrix product:

$$\big[\mathbf{y}_1\ \mathbf{y}_2\ \cdots\ \mathbf{y}_p\big] = \mathbf{A}\big[\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_p\big].$$

The two options are mathematically equivalent, and they both require precisely mnp flops. But, executing the computation as a matrix-matrix multiplication is *much* faster. To simplify slightly, the reason is that when you execute the loop, the matrix $\mathbf{A}$ has to be read from memory p times. (Real life is more complicated since the compiler might be smart and optimize the loop, etc.) In general, any linear algebraic operation that can be coded using matrix-matrix operations tends to be much faster than a corresponding operation coded as a sequence of matrix-vector operators. Technically, we sometimes refer to "BLAS3" operations (matrix-matrix) versus "BLAS2" operations (martix-vector) [**3, 1**].

The problem with the column pivoted QR factorization is that it inherently consists of a sequence of BLAS2 operations. This makes it hard to get good performance on multicore CPUs and GPUs (and in fact, even on singlecore CPUs, due to the multiple levels of cache on modern processors). This leaves us in an uncomfortable spot when it comes to low rank approximation. To explain, suppose that $\mathbf{A}$ is an $n \times n$ matrix, and let us consider three different matrix factorization algorithms:

(1) QR factorization without pivoting ("QR").
(2) QR factorization with column pivoting ("CPQR").
(3) Singular value decomposition ("SVD").

All three algorithms have asymptotic complexity of $O(n^3)$, meaning that there are constants such that

$$T_{\text{QR}} \sim C_{\text{QR}}\, n^3, \qquad T_{\text{CPQR}} \sim C_{\text{CPQR}}\, n^3, \qquad T_{\text{SVD}} \sim C_{\text{SVD}}\, n^3.$$

On most computer architectures we have $C_{\text{QR}} < C_{\text{CPQR}} < C_{\text{SVD}}$, and the differences typically are not small. (See Exercise ??). Comparing these three algorithms, we find that:

Algorithm:	QR	CPQR	SVD
Speed:	Fast.	Slow.	Slowest.
Ease of parallelization:	Fairly easy.	Very hard.	Very hard.
Useful for low-rank approximation:	No.	Yes.	Excellent.
Partial factorization possible?	Yes, but not useful.	Yes.	Not easily.

What we would want is an algorithm for low rank approximation that can be *blocked* so that it can be implemented using BLAS3 operations rather than BLAS2 operations. It turns out that randomized sampling provides an excellent path for this, as we will see in Section 2.

1.6. Subspaces associated with low rank approximation

This section briefly describes the geometric interpretation of low-rank approximation. Throughout the section, suppose that $\mathbf{A}$ is an $m \times n$ matrix of rank k. (Typically, this would be an *approximate* rank, but for simplicity, let us ignore the error for now.) Our starting point here is that we have computed rank k factorizations, either a QR factorization

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{Q}}\ \underset{k\times n}{\mathbf{R}}\ \underset{n\times n}{\mathbf{P}^*} \tag{1.6}$$

or an SVD

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{U}}\ \underset{k\times k}{\mathbf{D}}\ \underset{k\times n}{\mathbf{V}^*}. \tag{1.7}$$

Set $X = \mathbb{R}^n$ and $Y = \mathbb{R}^m$ so that

$$\mathbf{A} : X \to Y.$$

Now observe that

$$X = X_1 \oplus X_2, \qquad \text{and} \qquad Y = Y_1 \oplus Y_2, \tag{1.8}$$

where

- $X_1 = \text{row}(\mathbf{A})$ is a subspace of rank k.
- $X_2 = \text{null}(\mathbf{A})$ is a subspace of rank $n - k$.
- $Y_1 = \text{col}(\mathbf{A})$ is a subspace of rank k.
- $Y_2 = \text{col}(\mathbf{A})^{\perp}$ is a subspace of rank $m - k$.

The decomposition (1.8) clarifies the action of $\mathbf{A}$. Given a vector $\mathbf{x} \in \mathbb{R}^n$, we write

$$\mathbf{x} = \underbrace{\mathbf{x}_1}_{\in X_1} + \underbrace{\mathbf{x}_2}_{\in X_2}.$$

Then $\mathbf{A}\mathbf{x}_2 = \mathbf{0}$, and so $\mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{x}_1 + \mathbf{A}\mathbf{x}_2 = \mathbf{A}\mathbf{x}_1 \in Y_1$. So A maps X_2 to zero, while the restriction $A:\ X_1 \to Y_1$ is one-to-one.

Next, let us discuss how the partial factorizations (1.6) and (1.7) provide us with bases for the fundamental spaces associated with $\mathbf{A}$:

The column space. This is easy, the columns of $\mathbf{Q}$ and $\mathbf{U}$ directly form ON-bases for $\text{col}(\mathbf{A})$. The two bases are typically different.

The row space. If you have the SVD (1.7), then the columns of $\mathbf{V}$ form an ON-basis for $\text{row}(\mathbf{A})$. If you have the QR factorization (1.7), then the rows of $\mathbf{R}$ in principle form a basis for the row space of $\mathbf{A}$, but it is not an *orthonormal* basis. Observe however that since k is small, it is easy to obtain an ON-basis by simply performing Gram-Schmidt on the rows of $\mathbf{R}\mathbf{P}^*$ (pivoting is typically not required). For instance, if we execute

$$\big[\mathbf{S}, \sim\big] = \texttt{qr}(\mathbf{P}\mathbf{R}^*, 0)$$

then the columns of $\mathbf{S}$ form an ON basis for $\text{row}(\mathbf{A})$.

The nullspace of a matrix. The *partial* factorizations we computed do not directly provide a basis for the nullspace of a matrix. If the matrix is small, then you could compute the full factorizations, and get the bases that way. For big matrices, this tends to not be feasible, though. However, observe that the information provided is enough to *characterize* the null-space. Suppose that we have access to an $n \times k$ matrix $\mathbf{V}$ holding an ON basis for the row space of $\mathbf{A}$ (e.g. the matrix of right singular vectors). Observe that we can then split the identity operator as

$$\mathbf{I} = \mathbf{V}\mathbf{V}^* + \big(\mathbf{I} - \mathbf{V}\mathbf{V}^*\big).$$

The first term is the orthogonal projection onto $\text{row}(\mathbf{A})$, and the second term is the orthogonal projection onto $\text{row}(\mathbf{A})^\perp = \text{null}(\mathbf{A})$. In other words, given a vector $\mathbf{x} \in \mathbb{R}^n$, we can write

$$\mathbf{x} = \mathbf{y} + \mathbf{z},$$

where $\mathbf{y}$ and $\mathbf{z}$ are the orthogonal projections of $\mathbf{x}$ onto $\text{row}(\mathbf{A})$ and $\text{null}(\mathbf{A})$, respectively:

$$\mathbf{y} = \mathbf{V}\mathbf{V}^*\mathbf{x} \in \text{row}(\mathbf{A}), \qquad \text{and} \qquad \mathbf{z} = \mathbf{x} - \mathbf{y} = \big(\mathbf{I} - \mathbf{V}\mathbf{V}^*\big)\mathbf{x} \in \text{null}(\mathbf{A}).$$

CHAPTER 2

Randomized methods for low rank approximation

2.1. Introduction

This chapter describes randomized techniques for computing approximate low-rank factorizations to matrices. To quickly introduce the key ideas, let us describe a simple prototypical randomized algorithm: Let $\mathbf{A}$ be a matrix of size $m \times n$ that is approximately of low rank. In other words, we assume that for some integer $k < \min(m, n)$, and some tolerance $\varepsilon > 0$, there exists a matrix $\mathbf{A}_k$ of rank k such that

$$\|\mathbf{A} - \mathbf{A}_k\| \leq \varepsilon.$$

Then a natural question is how do you in a computationally efficient manner construct such a rank-k approximating matrix $\mathbf{A}_k$? In 2006, it was observed in [**13**] (which was inspired by [**6**], and later led to [**12, 14, 11**]) that random matrix theory provides a simple and elegant solution: Draw a *Gaussian random matrix* $\mathbf{G}$ of size $n \times k$ and form a *sampling matrix* $\mathbf{Y} = \mathbf{AG}$. Then in many important situations, the matrix

$$\underset{m \times n}{\mathbf{A}_k} := \underset{m \times k}{\mathbf{Y}} \underset{k \times n}{\left(\mathbf{Y}^{\dagger}\mathbf{A}\right)}, \tag{2.1}$$

where $\mathbf{Y}^{\dagger}$ is the Moore-Penrose pseudo-inverse of $\mathbf{Y}$, is close to optimal. With this observation as a starting point, one can construct highly efficient algorithms for computing approximate spectral decompositions of $\mathbf{A}$, for solving certain least-squares problems, for doing principal component analysis of large data sets, etc.

Many of the randomized sampling techniques we will describe are supported by rigorous mathematical analysis. For instance, it has been proved that the matrix $\mathbf{A}_k$ defined by (2.1) provides almost as good of an approximation to $\mathbf{A}$ as the best possible approximant of rank, say, $k - 5$, with probability almost 1. (The number "5" in "$k - 5$" results from a particular choice of a tuning parameter.) See [**11**, Sec. 10] and Section 2.6.

The algorithms that result from using randomized sampling techniques are computationally efficient, and are simple to implement as they rely on standard building blocks such as matrix-matrix multiplication, unpivoted QR factorization, etc, that are available for most computing environments (multicore CPU, GPU, distributed memory machines, etc). As an illustration, we invite the reader to peek ahead at Figure 2.1, which provides a complete Matlab code for a randomized algorithm that computes an approximate singular value decomposition of a matrix. Examples of improvements enabled by these randomized algorithms include:

- Given an $m \times n$ matrix $\mathbf{A}$, the cost of computing a rank-k approximant using classical methods is $O(mnk)$. Randomized algorithms attain complexity $O(mn \log k + k^2(m + n))$ [**11**, Sec. 6.1], and Section 2.5.
- Techniques for performing principal component analysis (PCA) of large data sets have been greatly accelerated, in particular when the data is stored out-of-core [**10**].
- Randomized methods tend to require less communication than traditional methods, and can be efficiently implemented on severely communication constrained environments such as GPUs [**15**] and even distributed computing platforms such as the Amazon EC2 Cloud computer [**9**, Ch. 4].
- Randomized algorithms enable *single-pass* matrix factorization in which the matrix is streamed and never stored, cf. [**11**, Sec. 6.3] and Section 2.4.

2.2. A two-stage approach

The problem of computing an approximate low-rank factorization to a given matrix can conveniently be split into two distinct stages. For concreteness, we describe the split for the specific task of computing an approximate

singular value decomposition. To be precise, given an $m \times n$ matrix $\mathbf{A}$ and a target rank k, we seek to compute factors $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ such that

$$\underset{m\times n}{\mathbf{A}} \approx \underset{m\times k}{\mathbf{U}} \underset{k\times k}{\mathbf{D}} \underset{k\times n}{\mathbf{V}^*}.$$

The factors $\mathbf{U}$ and $\mathbf{V}$ should be orthonormal, and $\mathbf{D}$ should be diagonal. (For now, we assume that the rank k is known in advance, techniques for relaxing the assumption are described in Section 2.9.) Following [**11**], we split this task into two computational stages:

Stage A — find an approximate range: Construct an $m \times k$ matrix $\mathbf{Q}$ with orthonormal columns such that $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$. This step will be executed via a randomized process described in Section 2.3.

Stage B — form a specific factorization: Given the matrix $\mathbf{Q}$ computed in Stage A, form factors $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$, via classical deterministic techniques. For instance, this stage can be executed via the following steps:
(1) Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.
(2) Decompose the matrix $\mathbf{B}$ in a singular value decomposition $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.
(3) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

The point here is that in a situation where $k \ll \min(m, n)$, the difficult part of the computation is concentrated to Stage A. Once that is finished, the post-processing in Stage B is easy.

REMARK 2.1. Stage B is exact up to floating point arithmetic so all errors in the factorization process are incurred at Stage A. In other words, if the factor $\mathbf{Q}$ satisfies

$$||\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}|| \leq \varepsilon,$$

then the full factorization satisfies

$$||\mathbf{A} - \mathbf{U}\mathbf{D}\mathbf{V}^*|| \leq \varepsilon \tag{2.2}$$

unless ε is close to the machine precision. Note that (2.2) does not in general guarantee that the computed singular vectors in the matrices $\mathbf{U}$ and $\mathbf{V}$ are within distance ε of the exact leading k singular vectors. In many (but not all) contexts, this is not a problem since only the error in the product $\mathbf{U}\mathbf{D}\mathbf{V}^*$ matters. (The computed singular values in $\mathbf{D}$ *are* within distance ε of the exact singular values, but note that for small singular values the relative precision may be poor.)

2.3. A randomized algorithm for "Stage A" — the range finding problem

This section describes a randomized technique for solving the range finding problem introduced as "Stage A" in Section 2.2. As a preparation for this discussion, let us recall that an "ideal" basis matrix $\mathbf{Q}$ for the range of a given matrix $\mathbf{A}$ is the matrix $\mathbf{U}_k$ formed by the k leading left singular vectors of $\mathbf{A}$. Letting $\sigma_j(\mathbf{A})$ denote the j'th singular value of $\mathbf{B}$, the Eckard-Young theorem [**18**] states that

$$\inf\{||\mathbf{A} - \mathbf{C}|| : \mathbf{C} \text{ has rank } k\} = ||\mathbf{A} - \mathbf{U}_k\mathbf{V}_k^*\mathbf{A}|| = \sigma_{k+1}(\mathbf{A}).$$

Now consider a simplistic randomized method for constructing a spanning set with k vectors for the range of a matrix $\mathbf{A}$: Draw k random vectors $\{\mathbf{g}_j\}_{j=1}^k$ from a Gaussian distribution, map these to vectors $\mathbf{y}_j = \mathbf{A}\mathbf{g}_j$ in the range of $\mathbf{A}$, and then use the resulting set $\{\mathbf{y}_j\}_{j=1}^k$ as a basis. Upon orthonormalization via, e.g., Gram-Schmidt, an orthonormal basis $\{\mathbf{q}_j\}_{j=1}^k$ would be obtained. If the matrix $\mathbf{A}$ has *exact* rank k, then the vectors $\{\mathbf{A}\mathbf{g}_j\}_{j=1}^k$ would with probability 1 be linearly independent, and the resulting ON-basis $\{\mathbf{q}_j\}_{j=1}^k$ would exactly span the range of $\mathbf{A}$. This would in a sense be an ideal algorithm. The problem is that in practice, there are almost always many non-zero singular values beyond the first k ones. These modes will shift the sample vectors $\mathbf{A}\mathbf{g}_j$ out of the space spanned by the k leading singular vectors of $\mathbf{A}$ and the process described can (and frequently does) produce a poor basis. Luckily, there is a fix: Simple take a few extra samples. It turns out that if we take, say, $k + 10$ samples instead of k, then the process will with probability almost 1 produce a basis that is comparable to the best possible basis.

To summarize the discussion in the previous paragraph, the randomized sampling algorithm for constructing an approximate rank k basis for the range of a given $m \times n$ matrix $\mathbf{A}$ proceeds as follows: First pick a small integer

ALGORITHM: RSVD — BASIC RANDOMIZED SVD

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p (say $p = 10$).

Outputs: Matrices $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate rank-$(k+p)$ SVD of $\mathbf{A}$. (I.e. $\mathbf{U}$ and $\mathbf{V}$ are orthonormal and $\mathbf{D}$ is diagonal.)

Stage A:

(1) Form an $n \times (k+p)$ Gaussian random matrix $\mathbf{G}$. `G = randn(n,k+p)`

(2) Form the sample matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{G}$. `Y = A*G`

(3) Orthonormalize the columns of the sample matrix $\mathbf{Q} = \mathtt{orth}(\mathbf{Y})$. `[Q,~] = qr(Y,0)`

Stage B:

(4) Form the $(k+p) \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$. `B = Q'*A`

(5) Form the SVD of the small matrix $\mathbf{B}$: $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$. `[Uhat,D,V] = svd(B,'econ')`

(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$. `U = Q*Uhat`

FIGURE 2.1. A basic randomized algorithm. If a factorization of precisely rank k is desired, the factorization in Step 5 can be truncated to the k leading terms.

p representing how much "over-sampling" we do. (The choice $p = 10$ is often good.) Then execute the following steps:

(1) Form a set of $k+p$ random Gaussian vectors $\{\mathbf{g}_j\}_{j=1}^{k+p}$.

(2) Form a set $\{\mathbf{y}_j\}_{j=1}^{k+p}$ of samples from the range where $\mathbf{y}_j = \mathbf{A}\mathbf{g}_j$.

(3) Perform Gram-Schmidt on the set $\{\mathbf{y}_j\}_{j=1}^{k+p}$ to form the ON-set $\{\mathbf{q}_j\}_{j=1}^{k+p}$.

Now observe that the $k+p$ matrix-vector products are independent and can advantageously be executed in parallel. A full algorithm for computing an approximate SVD using this simplistic sampling technique for executing "Stage A" is summarized in Figure 2.1.

The error incurred by the randomized range finding method described in this section is a random variable. There exist rigorous bounds for both the expectation of this error, and for the likelihood of a large deviation from the expectation. These bounds demonstrate that when the singular values of $\mathbf{A}$ decay "reasonably fast," the error incurred is close to the theoretically optimal one. We provide more details in Section 2.6.

2.4. Single pass algorithms

The randomized algorithm described in Figure 2.1 accesses the matrix $\mathbf{A}$ twice, first in "Stage A" where we build an ON-basis for the column space, and then in "Stage B" where we project $\mathbf{A}$ on to the space spanned by the computed basis vectors. It turns out to be possible to modify the algorithm in such a way that each entry of $\mathbf{A}$ is accessed only *once*. This is important because it allows us to compute the factorization of a matrix that is too large to be stored even out-of-core.

For *Hermitian* matrices, the modification to Algorithm 2.1 is very minor and we describe it in Section 2.4.1. Section 2.4.2 then handles the case of a general matrix.

REMARK 2.2 (Loss of accuracy). The single-pass algorithms described in this section tend to produce a factorization of lower accuracy than what Algorithm 2.1 would yield. In situations where a two-pass algorithm is feasible, it is therefore often preferable to the single-pass algorithm.

REMARK 2.3 (Streaming Algorithms). We say that an algorithm for processing a matrix is a *streaming algorithm* if each entry of the matrix is accessed only once, and if, in addition, it can be fed the entries in any order. (In other words, the algorithm is not allowed to dictate the order in which the elements are viewed.) The algorithms described in this section satisfy both of these conditions.

2.4.1. Hermitian matrices. Suppose that $\mathbf{A} = \mathbf{A}^*$, and that our objective is to compute an approximate eigenvalue decomposition

$$\underset{n\times n}{\mathbf{A}} \approx \underset{n\times k}{\mathbf{U}} \quad \underset{k\times k}{\mathbf{D}} \quad \underset{k\times n}{\mathbf{U}^*} \tag{2.3}$$

with $\mathbf{U}$ an ON matrix and $\mathbf{D}$ diagonal. (Note that for a Hermitian matrix, the EVD and the SVD are essentially equivalent, and that the EVD is the more natural factorization.) Then execute Stage A with an over-sampling parameter p to compute an ON matrix $\mathbf{Q}$ whose columns form an approximate basis for the column space of $\mathbf{A}$:

(1) Draw a Gaussian random matrix $\mathbf{G}$ of size $n \times (k+p)$.
(2) Form the sampling matrix $\mathbf{Y} = \mathbf{AG}$.
(3) Orthonormalize the columns of $\mathbf{Y}$ to form $\mathbf{Q}$, in other words $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$.

Then

$$\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}. \tag{2.4}$$

Observe that since in this case the column space and the row space are equivalent, we also have

$$\mathbf{A} \approx \mathbf{AQQ}^*. \tag{2.5}$$

Combine (2.4) and (2.5) to obtain

$$\mathbf{A} \approx \mathbf{QQ}^*\mathbf{AQQ}^*. \tag{2.6}$$

We define

$$\mathbf{C} = \mathbf{Q}^*\mathbf{AQ}. \tag{2.7}$$

If $\mathbf{C}$ is known, then the post-processing is straight-forward: Simply compute the EVD of $\mathbf{C}$ to obtain $\mathbf{C} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^*$, then define $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$, to find that

$$\mathbf{A} \approx \mathbf{QCQ}^* = \mathbf{Q}\hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{U}}^*\mathbf{Q}^* = \mathbf{UDU}^*.$$

The problem now is that since we are seeking a single-pass algorithm, we are not in position to evaluate $\mathbf{C}$ directly from formula (2.7). Instead, we will derive a formula for $\mathbf{C}$ that can be evaluated without revisiting $\mathbf{A}$. To this end, multiply (2.7) by $\mathbf{Q}^*\mathbf{G}$ to obtain

$$\mathbf{C}(\mathbf{Q}^*\mathbf{G}) = \mathbf{Q}^*\mathbf{AQQ}^*\mathbf{G} \approx \{\text{Use (2.5)}\} \approx \mathbf{Q}^*\mathbf{AG} = \mathbf{Q}^*\mathbf{Y} \tag{2.8}$$

From (2.5), we know that $\mathbf{AQQ}^* \approx \mathbf{A}$ so we can approximate right hand side in (2.8) via $\mathbf{Q}^*\mathbf{AQQ}^*\mathbf{G} \approx \mathbf{Q}^*\mathbf{AG} = \mathbf{Q}^*\mathbf{Y}$. Ignoring the approximation error, we define $\mathbf{C}$ as the solution of the linear system (recall $\ell = k + p$)

$$\underset{\ell\times\ell}{\mathbf{C}} \quad \underset{\ell\times\ell}{\big(\mathbf{Q}^*\mathbf{G}\big)} \quad = \quad \underset{\ell\times\ell}{\big(\mathbf{Q}^*\mathbf{Y}\big)}. \tag{2.9}$$

At first, it may appears that (2.9) is perfectly balanced in that there are ℓ^2 equations for ℓ^2 unknowns. However, we need to enforce that $\mathbf{C}$ is Hermitian, so the system is actually over-determined by roughly a factor of two.

The procedure described in this section is less accurate than the procedure described in Figure 2.1 for two reasons:

(1) The approximation error in formula (2.6) tends to be larger than the error in (2.4). In fact, with $\varepsilon = \|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\|$, we have

$$\|\mathbf{A} - \mathbf{QQ}^*\mathbf{AQQ}^*\| = \|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\| + \|\mathbf{QQ}^*\mathbf{A} - \mathbf{QQ}^*\mathbf{AQQ}^*\| = \\ \varepsilon + \|\mathbf{QQ}^*(\mathbf{A} - \mathbf{AQQ}^*)\| \leq \varepsilon + \|\mathbf{A} - \mathbf{AQQ}^*\| = 2\varepsilon,$$

where we used that $\|\mathbf{QQ}^*\| \leq 1$ (since $\mathbf{Q}$ is ON, and $\mathbf{QQ}^*$ therefore is an ON projection) and that $\|\mathbf{A} - \mathbf{AQQ}^*\| = \|(\mathbf{A} - \mathbf{AQQ}^*)^*\| = \|\mathbf{QQ}^*\mathbf{A} - \mathbf{A}\|$. In other words, we could in a worst case scenario double the error.
(2) While the matrix $\mathbf{Q}^*\mathbf{G}$ is invertible, it tends to be very ill-conditioned.

ALGORITHM: SINGLE-PASS RANDOMIZED EVD FOR A HERMITIAN MATRIX

Inputs: An $n \times n$ Hermitian matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p (say $p = 10$).

Outputs: Matrices $\mathbf{U}$ and $\mathbf{D}$ in an approximate rank-k EVD of $\mathbf{A}$. (I.e. $\mathbf{U}$ is orthonormal and $\mathbf{D}$ is diagonal.)

Stage A:

(1) Form an $n \times (k+p)$ Gaussian random matrix $\mathbf{G}$.
(2) Form the sample matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{G}$.
(3) Let $\mathbf{Q}$ denote the ON-matrix formed by the k dominant left singular vectors of $\mathbf{Y}$.

Stage B:

(4) Let $\mathbf{C}$ denote the $k \times k$ least squares solution of $\mathbf{C}\left(\mathbf{Q}^*\mathbf{G}\right) = \left(\mathbf{Q}^*\mathbf{Y}\right)$ obtained by enforcing that $\mathbf{C}$ should be Hermitian.
(5) Decompose the matrix $\mathbf{C}$ in an eigenvalue decomposition $[\hat{\mathbf{U}},\, \mathbf{D}] = \texttt{eig}(\mathbf{C})$.
(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

FIGURE 2.2. A basic randomized algorithm single-pass algorithm suitable for a Hermitian matrix.

REMARK 2.4 (Extra over-sampling). The combat the problem that $\mathbf{Q}^*\mathbf{G}$ tends to be ill-conditioned, it is helpful to over-sample more aggressively when using a single pass algorithm. Specifically, let us form $\mathbf{Q}$ as the leading k left singular vectors of $\mathbf{Y}$ (compute these by forming the full SVD of $\mathbf{Y}$, and then discard the last p components). Then $\mathbf{C}$ will be of size $k \times k$, and the equation that specifies $\mathbf{C}$ reads

$$\underset{k\times k}{\mathbf{C}} \quad \underset{k\times \ell}{\left(\mathbf{Q}\mathbf{G}\right)} \;=\; \underset{k\times \ell}{\mathbf{Q}^*\mathbf{Y}}. \tag{2.10}$$

Since (2.10) is over-determined, we solve it using a least-squares technique. Observe that we are now looking for less information (a $k \times k$ matrix rather than an $\ell \times \ell$ matrix), and have more information in order to determine it.

2.4.2. General matrices. Now consider a general $m \times n$ matrix $\mathbf{A}$. We now need to apply randomized sampling to both its row space and its column space simultaneously. We proceed as follows:

(1) Draw two Gaussian random matrices $\mathbf{G}_{\rm c}$ of size $n \times (k+p)$ and $\mathbf{G}_{\rm r}$ of size $m \times (k+p)$.
(2) Form two sampling matrices $\mathbf{Y}_{\rm c} = \mathbf{A}\mathbf{G}_{\rm c}$ and $\mathbf{Y}_{\rm r} = \mathbf{A}^*\mathbf{G}_{\rm r}$.
(3) Compute two basis matrices $\mathbf{Q}_{\rm c} = \texttt{orth}(\mathbf{Y}_{\rm c})$ and $\mathbf{Q}_{\rm r} = \texttt{orth}(\mathbf{Y}_{\rm r})$.

Now define the small projected matrix via

$$\mathbf{C} = \mathbf{Q}_{\rm c}^*\mathbf{A}\mathbf{Q}_r. \tag{2.11}$$

We will derive two relationships that together will determine $\mathbf{C}$ is a manner that is analogous to (2.8). First left multiply (2.11) by $\mathbf{G}_{\rm r}^*\mathbf{Q}_{\rm c}$ to obtain

$$\mathbf{G}_{\rm r}^*\mathbf{Q}_{\rm c}\mathbf{C} = \mathbf{G}_{\rm r}^*\mathbf{Q}_{\rm c}\mathbf{Q}_{\rm c}^*\mathbf{A}\mathbf{Q}_{\rm r} \approx \mathbf{G}_{\rm r}^*\mathbf{A}\mathbf{Q}_{\rm r} = \mathbf{Y}_{\rm r}^*\mathbf{Q}_{\rm r}. \tag{2.12}$$

Next we right multiply (2.11) by $\mathbf{Q}_{\rm r}^*\mathbf{G}_{\rm c}$ to obtain

$$\mathbf{C}\mathbf{Q}_{\rm r}^*\mathbf{G}_{\rm c} = \mathbf{Q}_{\rm c}^*\mathbf{A}\mathbf{Q}_r\mathbf{Q}_{\rm r}^*\mathbf{G}_{\rm c} \approx \mathbf{Q}_{\rm c}^*\mathbf{A}\mathbf{G}_{\rm c} = \mathbf{Q}_{\rm c}^*\mathbf{Y}_{\rm c}. \tag{2.13}$$

We now define $\mathbf{C}$ as the least-square solution of the two equations

$$\left(\mathbf{G}_{\rm r}^*\mathbf{Q}_{\rm c}\right)\mathbf{C} = \mathbf{Y}_{\rm r}^*\mathbf{Q}_{\rm r} \qquad \text{and} \qquad \mathbf{C}\left(\mathbf{Q}_{\rm r}^*\mathbf{G}_{\rm c}\right) = \mathbf{Q}_{\rm c}^*\mathbf{Y}_{\rm c}.$$

Again, the system is over-determined by about a factor of 2, and it is advantageous to make it further over-determined by more aggressive over-sampling, cf. Remark 2.4.

ALGORITHM: SINGLE-PASS RANDOMIZED SVD FOR A GENERAL MATRIX

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p (say $p = 10$).

Outputs: Matrices $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{D}$ in an approximate rank-k SVD of $\mathbf{A}$. (I.e. $\mathbf{U}$ and $\mathbf{V}$ are ON and $\mathbf{D}$ is diagonal.)

Stage A:

(1) Form two Gaussian random matrices $\mathbf{G}_{\rm c} = \texttt{randn}(n, k+p)$ and $\mathbf{G}_{\rm r} = \texttt{randn}(m, k+p)$.
(2) Form the sample matrices $\mathbf{Y}_{\rm c} = \mathbf{A}\,\mathbf{G}_{\rm c}$ and $\mathbf{Y}_{\rm r} = \mathbf{A}^*\,\mathbf{G}_{\rm r}$.
(3) Form ON matrices $\mathbf{Q}_{\rm c}$ and $\mathbf{Q}_{\rm r}$ consisting of the k dominant left singular vectors of $\mathbf{Y}_{\rm c}$ and $\mathbf{Y}_{\rm r}$.

Stage B:

(4) Let $\mathbf{C}$ denote the $k \times k$ least squares solution of the joint system of equations formed by the equations $\big(\mathbf{G}_{\rm r}^*\mathbf{Q}_{\rm c}\big)\,\mathbf{C} = \mathbf{Y}_{\rm r}^*\mathbf{Q}_{\rm r}$ and $\mathbf{C}\,\big(\mathbf{Q}_{\rm r}^*\mathbf{G}_{\rm c}\big) = \mathbf{Q}_{\rm c}^*\mathbf{Y}_{\rm c}$.
(5) Decompose the matrix $\mathbf{C}$ in a singular value decomposition $[\hat{\mathbf{U}},\, \mathbf{D},\, \hat{\mathbf{V}}] = \texttt{svd}(\mathbf{C})$.
(6) Form $\mathbf{U} = \mathbf{Q}_{\rm c}\hat{\mathbf{U}}$ and $\mathbf{V} = \mathbf{Q}_{\rm r}\hat{\mathbf{V}}$.

FIGURE 2.3. A basic randomized algorithm single-pass algorithm suitable for a general matrix.

2.5. A method with complexity $O(mn \log k)$ for dense matrices that fit in RAM

The RSVD algorithm described in Figure 2.1 for constructing an approximate basis for the range of a given matrix $\mathbf{A}$ is highly efficient when we have access to fast algorithms for evaluating matrix-vector products $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$. For the case where $\mathbf{A}$ is a general $m \times n$ matrix given simply as an array or real numbers, the cost of evaluating the sample matrix $\mathbf{Y} = \mathbf{A}\mathbf{G}$ (in Step (2) of the algorithm in Figure 2.1) is $O(mnk)$. The algorithm is still often faster than classical methods since the matrix-matrix multiply can be highly optimized, but it does not have an edge in terms of asymptotic complexity. However, it turns out to be possible to modify the algorithm by replacing the Gaussian random matrix $\mathbf{G}$ with a different random matrix $\mathbf{\Omega}$ that has two seemingly contradictory properties:

(1) $\mathbf{\Omega}$ is sufficiently *structured* that the product $\mathbf{A\Omega}$ can be evaluated in $O(mn\log(k))$ flops.
(2) $\mathbf{\Omega}$ is sufficiently *random* that the columns of $\mathbf{A\Omega}$ accurately span the range of $\mathbf{A}$.

For instance, a good choice $\mathbf{\Omega}$ is

$$\underset{n\times\ell}{\mathbf{\Omega}} \;=\; \underset{n\times n}{\mathbf{D}}\;\; \underset{n\times n}{\mathbf{F}}\;\; \underset{n\times\ell}{\mathbf{S}}, \tag{2.14}$$

where $\mathbf{D}$ is a diagonal matrix whose diagonal entries are complex numbers of modulus one drawn from a uniform distribution on the unit circle in the complex plane, where $\mathbf{F}$ is the discrete Fourier transform,

$$\mathbf{F}(p,q) = n^{-1/2}\, e^{-2\pi {\rm i}(p-1)(q-1)/n}, \qquad p, q \in \{1,\, 2,\, 3,\, \ldots,\, n\},$$

and where $\mathbf{S}$ is matrix consisting of a random subset of ℓ columns from the $n \times n$ unit matrix (drawn without replacement). In other words, given an arbitrary matrix $\mathbf{X}$ of size $m \times n$, the matrix $\mathbf{XS}$ consists of a randomly drawn subset of ℓ columns of $\mathbf{X}$. For the matrix $\mathbf{\Omega}$ specified by (2.14), the product $\mathbf{X\Omega}$ can be evaluated via a subsampled FFT in $O(mn\log(\ell))$ operations. The parameter ℓ should be chosen slightly larger than the target rank k; the choice $\ell = 2k$ is often good.

By using the structured random matrix described in this section, we can reduce the complexity of "Stage A" in the RSVD from $O(mnk)$ to $O(mn\log k)$. We next need to modify "Stage B" to eliminate the need to compute $\mathbf{Q}^*\mathbf{A}$. One option is to use the single pass algorithm described in 2.3, using the structured random matrix to approximate both the row and the column spaces of $\mathbf{A}$. A second, and typically better, option is to use a so called *row-extraction* technique for Stage B, we describe the details in Section 4.4.

The current error analysis for the accelerated range finder is less satisfactory than the one for Gaussian random matrices. In the general case, only very weak results can be proven. In practice, the accelerated scheme is often as accurate as the Gaussian one, but we do not currently have good theory to predict precisely when this happens, see [**11**, Sec. 11].

2.6. Theoretical performance bounds

In this section, we will briefly summarize some proven results concerning the error in the output of the basic RSVD algorithm in Figure 2.1. Observe that the factors $\mathbf{U}$, $\mathbf{D}$, $\mathbf{V}$ depend not only on $\mathbf{A}$, but also on the draw of the random matrix $\mathbf{G}$. This means that the error that we try to bound is a *random variable.* It is therefore natural to seek bounds on first the "expectation" or "mean" of the error, and then on the likelihood of large deviations from the mean.

Before we start, let us recall that all the error incurred by the RSVD algorithm in Figure 2.1 is incurred in Stage A. The reason is that the "post-processing" in Stage B is exact (up to floating point arithmetic):

$$\mathbf{A} - \mathbf{Q}\underbrace{\mathbf{Q}^*\mathbf{A}}_{=\mathbf{B}} = \mathbf{A} - \mathbf{Q}\underbrace{\mathbf{B}}_{=\hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*} = \mathbf{A} - \underbrace{\mathbf{Q}\hat{\mathbf{U}}}_{=\mathbf{U}}\mathbf{D}\mathbf{V}^* = \mathbf{A} - \mathbf{U}\mathbf{D}\mathbf{V}^*.$$

Consequently, we can (and will) restrict ourselves to providing bounds on $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$.

2.6.1. Bounds on the expectation of the error. For instance, Theorem 10.6 of [**11**] states:

THEOREM 2. *Let $\mathbf{A}$ be an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$. Let k be a target rank, and let p be an over-sampling parameter such that $p \geq 2$ and $k + p \leq \min(m, n)$. Let $\mathbf{G}$ be a Gaussian random matrix of size $n \times (k + p)$ and set $\mathbf{Q} = \mathtt{orth}(\mathbf{A}\mathbf{G})$. Then the average error, as measured in the Frobenius norm, satisfies*

$$\mathbb{E}\big[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\mathrm{Fro}}\big] \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}. \tag{2.15}$$

The corresponding result for the spectral norm reads

$$\mathbb{R}\big[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|\big] \leq \left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1} + \frac{e\sqrt{k+p}}{p}\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}. \tag{2.16}$$

When errors are measured in the *Frobenius norm,* Theorem 2 is very gratifying. For our standard recommendation of $p = 10$, we are basically within a factor of $\sqrt{k/10}$ of the theoretically minimal error. (Recall that the Eckart-Young theorem states that $\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}$ is a lower bound on the residual for any rank-k approximant.) If you over-sample a little more aggressively and set $p = k$, then we are within a distance of $\sqrt{2}$ of the theoretically minimal error.

When errors are measured in the *spectral norm,* the situation is much less rosy. The first term in the bound in (2.16) is perfectly acceptable, but the second term is unfortunate in that it involves the minimal error in the Frobenius norm, which can be a much bigger factor and potentially renders this bound highly sub-optimal. The theorem is quite sharp, as it turns out, so this disparity reflects a true problem for the basic randomized scheme.

The extent to which the suboptimality in (2.16) is problematic depends on how rapidly the "tail" singular values $\{\sigma_j\}_{j>k}$ decay. If they decay fast, then the spectral norm error and the Frobenius norm error are similar, and the RSVD works well. If they decay slowly, then the RSVD performs OK when errors are measured in the Frobenius norm, but not very well when the spectral norm is the one of interest. To illustrate the difference, let us consider two situations:

Case 1 — fast decay: Suppose that the tail singular values decay exponentially fast, so that for some $\beta \in (0, 1)$ we have $\sigma_j \approx \sigma_{k+1}\,\beta^{j-k-1}$ for $j > k$. Then $\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2} \approx \sigma_{k+1}\left(\sum_{j=k+1}^{\min(m,n)} \beta^2\right)^{1/2} \leq \sigma_{k+1}(1 - \beta^2)^{-1/2}$. As long as β is not very close to 1, we see that the contribution from the tail singular values is very modest in this case.

Case 2 — no decay: Suppose that the tail singular values exhibit *no* decay, so that $\sigma_j = \sigma_{k+1}$ for $j > k$. This represents the worst case scenario, and now $\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2} = \sigma_{k+1}\sqrt{n-k}$. Since we want to allow for n to be very large (say $n = 10^6$), this represents a huge degree of suboptimality.

Fortunately, it is possible to modify the RSVD in such a way that the errors produced are close to optimal in both the spectral and the Frobenius norms. This is achieved by modestly increasing the computational cost. See Section 2.7 and [**11**, Sec. 4.5].

2.6.2. Bounds on the likelihood of large deviations. One can prove that (perhaps surprisingly) the likelihood of a large deviation from the mean depends only on the over-sampling parameter p, and decays extra-ordinarily fast. For instance, one can prove that if $p \geq 4$, then

$$||\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}|| \leq \left(1 + 17\sqrt{1+k/p}\right)\sigma_{k+1} + \frac{8\sqrt{k+p}}{p+1}\left(\sum\nolimits_{j>k}\sigma_j^2\right)^{1/2}, \tag{2.17}$$

with failure probability at most $3\,e^{-p}$, see [**11**, Cor. 10.9].

2.7. An accuracy enhanced randomized scheme

2.7.1. The key idea — power iteration. We mentioned earlier that the basic randomized scheme (see, e.g., Figure 2.1) gives accurate results for matrices whose singular values decay rapidly, but tends to produce suboptimal results when they do not. The theoretical results summarized in Section 2.6 make this claim precise. To recap, suppose that we compute a rank-k approximation to an $m \times n$ matrix $\mathbf{A}$ with singular values $\{\sigma_j\}_j$. The theory shows that the error measured in the spectral norm behaves like $\left(\sum_{j>k}\sigma_j^2\right)^{1/2}$. When the singular values decay slowly, this quantity can be much larger than the theoretically minimal approximation error (which is σ_{k+1}).

Recall that the objective of the randomized sampling is to construct a set of ON vectors $\{\mathbf{q}_j\}_{j=1}^{\ell}$ that capture to high accuracy the space spanned by the k dominant left singular vectors $\{\mathbf{u}_j\}_{j=1}^{k}$ of $\mathbf{A}$. The idea is now to sample not $\mathbf{A}$, but the matrix

$$\mathbf{A}^{(q)} := \left(\mathbf{A}\mathbf{A}^*\right)^q\mathbf{A},$$

where q is a small positive integer (typically, $q = 1$ or $q = 2$). A simple calculation shows that if $\mathbf{A}$ has singular value decomposition $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^*$, then the SVD of $\mathbf{A}^{(q)}$ is

$$\mathbf{A}^{(q)} = \mathbf{U}\,\mathbf{D}^{2q+1}\,\mathbf{V}^*.$$

In other words, $\mathbf{A}^{(q)}$ has the same left singular values as $\mathbf{A}$, while its singular values are $\{\sigma_j^{2q+1}\}_j$. Even when the singular values of $\mathbf{A}$ decay slowly, the singular values of $\mathbf{A}^{(q)}$ tend to decay fast enough for our purposes.

The accuracy enhanced scheme now consists of drawing a Gaussian matrix $\mathbf{G}$ and then forming a sample matrix

$$\mathbf{Y} = \left(\mathbf{A}\mathbf{A}^*\right)^q\mathbf{A}\mathbf{G}.$$

Then orthonormalize the columns of $\mathbf{Y}$ to obtain $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$, and proceed as before. The resulting scheme is shown in Figure 2.4.

REMARK 2.5. The scheme described in Figure 2.4 can lose accuracy due to round-off errors. The problem is that as q increases, all columns in the sample matrix $\mathbf{Y} = \left(\mathbf{A}\mathbf{A}^*\right)^q\mathbf{A}\mathbf{G}$ tend to align closer and closer to the dominant left singular vector. This means that we lose almost all accuracy in regards to the directions of singular values associated with smaller singular vectors. Roughly speaking, if

$$\frac{\sigma_j}{\sigma_1} \leq \epsilon_{\text{mach}}^{1/(2q+1)},$$

then all information associated with the j'th singular more is lost. This issue is explored in more detail in Section 3.2. For now, let us simply state a variation of the scheme in Figure 2.4 that ameliorates the round-off error problem by explicitly orthonormalizing the sample vectors between each iteration:

ALGORITHM: ACCURACY ENHANCED RANDOMIZED SVD

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, an over-sampling parameter p (say $p = 10$), and a small integer q denoting the number of steps in the power iteration.

Outputs: Matrices $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate rank-$(k+p)$ SVD of $\mathbf{A}$. (I.e. $\mathbf{U}$ and $\mathbf{V}$ are orthonormal and $\mathbf{D}$ is diagonal.)

(1) $\mathbf{G} = \texttt{randn}(n, k+p)$;
(2) $\mathbf{Y} = \mathbf{AG}$;
(3) **for** $j = 1 : q$
(4) $\quad \mathbf{Z} = \mathbf{A}^*\mathbf{Y}$;
(5) $\quad \mathbf{Y} = \mathbf{AZ}$;
(6) **end for**
(7) $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$;
(8) $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$;
(9) $[\hat{\mathbf{U}}, \mathbf{D}, \mathbf{V}] = \texttt{svd}(\mathbf{B}, \texttt{'econ'})$;
(10) $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$;

FIGURE 2.4. The accuracy enhanced randomized SVD. If a factorization of precisely rank k is desired, the factorization in Step 5 can be truncated to the k leading terms.

(1) $\mathbf{G} = \texttt{randn}(n, k+p)$;
(2) $\mathbf{Q} = \texttt{orth}(\mathbf{AG})$;
(3) **for** $j = 1 : q$
(4) $\quad \mathbf{W} = \texttt{orth}(\mathbf{A}^*\mathbf{Q})$;
(5) $\quad \mathbf{Q} = \texttt{orth}(\mathbf{AW})$;
(6) **end for**
(7) $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$;
(8) $[\hat{\mathbf{U}}, \mathbf{D}, \mathbf{V}] = \texttt{svd}(\mathbf{B}, \texttt{'econ'})$;
(9) $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$;

This scheme is more costly due to the calls to `orth`. However, this orthonormalization can be executed using *unpivoted* Gram-Schmidt, which is quite fast.

2.7.2. Theoretical results. A detailed error analysis of the scheme described in Figure 2.4 is provided in [**11**, Sec. 10.4]. In particular, the key theorem states the following:

THEOREM 3. *Let $\mathbf{A}$ denote an $m \times n$ matrix, let $p \geq 2$ be an over-sampling parameter, and let q denote a small integer. Draw a Gaussian matrix $\mathbf{G}$ of size $n \times (k+p)$, set $\mathbf{Y} = (\mathbf{AA}^*)^q\mathbf{AG}$, and let $\mathbf{Q}$ denote an $m \times (k+p)$ ON matrix resulting from orthonormalizing the columns of $\mathbf{Y}$. Then*

$$\mathbb{E}\big[\|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\|\big] \leq \left[\left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1}^{2q+1} + \frac{e\sqrt{k+p}}{p}\left(\sum\nolimits_{j>k}\sigma_j^{2(2q+1)}\right)^{1/2}\right]^{1/(2q+1)}. \tag{2.18}$$

The bound in (2.18) is slightly hard to decipher. To simplify it, let us consider the worst case scenario where there is no decay in the singular values beyond the truncation point, so that $\sigma_{k+1} = \sigma_{k+2} = \cdots = \sigma_{\min(m,n)}$. Then (2.18) simplifies to

$$\mathbb{E}\big[\|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\|\big] \leq \left[1 + \sqrt{\frac{k}{p-1}} + \frac{e\sqrt{k+p}}{p}\cdot\sqrt{\min\{m,n\} - k}\right]^{1/(2q+1)}\sigma_{k+1}.$$

In other words, as we increase the exponent q, the power scheme drives factor set in blue to one exponentially fast. This factor represents the degree of "sub-optimality" you can expect to see.

2.7.3. Extended sampling matrix. The scheme described in Section 2.7.1 is slightly wasteful in that it does not directly use all the sampling vectors computed. To further improve accuracy, let us form an "extended" sampling matrix

$$\mathbf{Y} = \big[\mathbf{AG},\, \mathbf{A}^2\mathbf{G},\, \ldots,\, \mathbf{A}^q\mathbf{G}\big].$$

Observe that this new sampling matrix $\mathbf{Y}$ has $q\ell$ columns. Then proceed as before:

$$\mathbf{Q} = \mathtt{qr}(\mathbf{Y}), \qquad \mathbf{B} = \mathbf{Q}^*\mathbf{A}, \qquad [\hat{\mathbf{U}}, \mathbf{D}, \mathbf{V}] = \mathtt{svd}(\mathbf{B}, \mathtt{'econ'}), \qquad \mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}.$$

Note that these computations are all much more expensive than those in Section 2.7.1 since we now work with matrices with $q\ell$ columns, as opposed to ℓ columns earlier. Since the cost of QR factorization, etc, grows quadratically with the number of columns, the difference is very substantial — the cost of dense linear algebra increases by a factor of $O(q^2)$. Consequently, the scheme described here is primarily useful in situations where the computational cost is dominated by applications of $\mathbf{A}$ and $\mathbf{A}^*$, and we want to maximally leverage all interactions with $\mathbf{A}$.

2.8. The Nyström method for positive symmetric definite matrices

When the input matrix $\mathbf{A}$ is positive semidefinite, the *Nyström method* can be used to improve the quality of standard factorizations at almost no additional cost; see [**4**] and its bibliography. To describe the idea, we first recall from Section 2.4.1 that when $\mathbf{A}$ is Hermitian (which of course every psd matrix is), then it is natural to use the approximation

$$\mathbf{A} \approx \mathbf{Q}\left(\mathbf{Q}^*\mathbf{A}\mathbf{Q}\right)\mathbf{Q}^*. \tag{2.19}$$

In contrast, the Nyström scheme builds a more sophisticated rank-k approximation, namely

$$\mathbf{A} \approx (\mathbf{AQ})\left(\mathbf{Q}^*\mathbf{AQ}\right)^{-1}(\mathbf{AQ})^*. \tag{2.20}$$

For both stability and computational efficiency, we typically rewrite (2.20) as

$$\mathbf{A} \approx \mathbf{F}\mathbf{F}^*,$$

where $\mathbf{F}$ is an approximate *Cholesky* factor of $\mathbf{A}$ of size $n \times k$, defined by

$$\mathbf{F} = (\mathbf{AQ})\left(\mathbf{Q}^*\mathbf{AQ}\right)^{-1/2}.$$

To compute the factor $\mathbf{F}$ numerically, first form the matrices $\mathbf{B}_1 = \mathbf{AQ}$ and $\mathbf{B}_2 = \mathbf{Q}^*\mathbf{B}_1$. Observe that $\mathbf{B}_2$ is necessarily psd, which means that we can compute its Cholesky factorization $\mathbf{B}_2 = \mathbf{C}^*\mathbf{C}$. Finally compute the factor $\mathbf{F} = \mathbf{B}_1\mathbf{C}^{-1}$ by performing a triangular solve. The low-rank factorization (2.20) can be converted to a standard decomposition using the techniques from Section 2.2.

The Nyström technique for computing an approximate eigenvalue decomposition is given in Figure 2.5. Let us compare the cost of this method to the more straight-forward method resulting from using the formula (2.19). In both cases, we need to twice apply $\mathbf{A}$ to a set of $k+p$ vectors (first in computing $\mathbf{AG}$, then in computing $\mathbf{AQ}$). But the Nyström method tends to result in substantially more accurate results. Informally speaking, the reason is that by exploiting the psd property of $\mathbf{A}$, we can take one step of power iteration "for free."

For a more formal analysis of the cost and accuracy of the Nyström method, we refer the reader to [**4**]. In particular, Lemma 4 of [**4**] implies that, in the spectral norm, the Nyström approximation error never exceeds $\|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\|$, and it is often substantially smaller.

ALGORITHM: EIGENVALUE DECOMPOSITION VIA THE NYSTRÖM METHOD

Given an $n \times n$ non-negative matrix $\mathbf{A}$, a target rank k and an over-sampling parameter p, this procedure computes an approximate eigenvalue decomposition $\mathbf{A} \approx \mathbf{U}\mathbf{\Lambda}\mathbf{U}^*$, where $\mathbf{U}$ is orthonormal, and $\mathbf{\Lambda}$ is nonnegative and diagonal.

(1) Draw a Gaussian random matrix $\mathbf{G} = \texttt{randn}(n, k+p)$.
(2) Form the sample matrix $\mathbf{Y} = \mathbf{A}\mathbf{G}$.
(3) Orthonormalize the columns of the sample matrix to obtain the basis matrix $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$.
(4) Form the matrices $\mathbf{B}_1 = \mathbf{A}\mathbf{Q}$ and $\mathbf{B}_2 = \mathbf{Q}^*\mathbf{B}_1$.
(5) Perform a Cholesky factorization $\mathbf{B}_2 = \mathbf{C}^*\mathbf{C}$.
(6) Form $\mathbf{F} = \mathbf{B}_1\mathbf{C}^{-1}$ using a triangular solve.
(7) Compute an SVD of the Cholesky factor $[\mathbf{U}, \mathbf{\Sigma}, \sim] = \texttt{svd}(\mathbf{F}, \texttt{'econ'})$.
(8) Set $\mathbf{\Lambda} = \mathbf{\Sigma}^2$.

FIGURE 2.5. The Nyström method. This procedure is applicable only to self-adjoint matrices with non-negative eigenvalues. It involves two applications of $\mathbf{A}$ to matrices with k columns, and consequently has comparable cost to the basic RSVD in Figure 2.1. However, it exploits the symmetry of $\mathbf{A}$ to boost the accuracy.

2.9. Adaptive rank determination with updating of the matrix

2.9.1. Problem formulation. Up to this point, we have assumed that the rank k is given as an input variable to the factorization algorithm. In practical usage, it is common that we are given instead a matrix $\mathbf{A}$ and a computational tolerance ε, and our task is then to determine a matrix $\mathbf{B}_k$ of rank k such that $\|\mathbf{A} - \mathbf{B}_k\| \leq \varepsilon$.

The techniques described in this section are designed for dense matrices stored in RAM. They directly update the matrix, and come with a firm guarantee that the computed low rank approximation is within distance ε of the original matrix. There are many situations where direct updating is not feasible and we can in practice only interact with the matrix via the matrix-vector multiplication (e.g., very large matrices stored out-of-core, sparse matrices, matrices that are defined implicitly). Section 2.10 describes algorithms designed for this environment that use randomized sampling techniques to *estimate* to approximation error.

Recall that for the case where a computational tolerance is given (rather than a rank), the optimal solution is given by the SVD. Specifically, let $\{\sigma_j\}_{k=1}^{\min(m,n)}$ be the singular values of $\mathbf{A}$. Then the minimal rank k for which $\inf\{\|\mathbf{A} - \mathbf{B}\| : \mathbf{B} \text{ has rank } k\}$ is the smallest integer k such that $\sigma_{k+1} \leq \varepsilon$. The algorithms described here will determine a k that is not necessarily optimal, but is typically fairly close.

2.9.2. A basic updating algorithm. Let us start by describing a very general algorithmic template for how to compute an approximate rank-k approximate factorization of a matrix. To be precise, suppose that we are given an $m \times n$ matrix $\mathbf{A}$, and a computational tolerance ε. Our objective is then to determine an integer $k \in \{1, 2, \ldots, \min(m,n)\}$, an $m \times k$ ON matrix $\mathbf{Q}_k$, and a $k \times n$ matrix $\mathbf{B}_k$ such that

$$\underset{m \times n}{\mathbf{A}} - \underset{m \times k}{\mathbf{Q}_k}\,\underset{k \times n}{\mathbf{B}_k} \| \leq \varepsilon.$$

This problem can be solved using the greedy algorithm shown in Figure 2.6 which builds $\mathbf{Q}_k$ and $\mathbf{B}_k$ one column and row at a time.

The algorithm described in Figure 2.6 is a generalization of the basic Gram-Schmidt procedure described in Figure 1.1. The key to understanding how the algorithm works is provided by the identity

$$\mathbf{A} = \mathbf{Q}_j\mathbf{B}_j + \mathbf{A}_j, \qquad j = 0, 1, 2, \ldots, k.$$

The computational efficiency and accuracy of the algorithm depends crucially on how the vector $\mathbf{y}$ is picked on line (4). Let us consider three possible selection strategies:

(1) $\mathbf{Q}_0 = [\,]$; $\mathbf{B}_0 = [\,]$; $\mathbf{A}_0 = \mathbf{A}$; $k = 0$;
(2) **while** $\|\mathbf{A}_k\| > \varepsilon$
(3) $\quad k = k + 1$
(4) $\quad$ Pick a vector $\mathbf{y} \in \text{Ran}(\mathbf{A}_{k-1})$
(5) $\quad \mathbf{q} = \mathbf{y}/\|\mathbf{y}\|$.
(6) $\quad \mathbf{b} = \mathbf{q}^*\mathbf{A}_{k-1}$
(7) $\quad \mathbf{Q}_k = [\mathbf{Q}_{k-1}\ \mathbf{q}]$
(8) $\quad \mathbf{B}_k = \begin{bmatrix} \mathbf{B}_{k-1} \\ \mathbf{b} \end{bmatrix}$
(9) $\quad \mathbf{A}_k = \mathbf{A}_{k-1} - \mathbf{q}\mathbf{b}$
(10) **end for**

FIGURE 2.6. A greedy algorithm for building a low-rank approximation to a given $m \times n$ matrix $\mathbf{A}$ that is accurate to within a given precision ε. To be precise, the algorithm determines an integer k, an $m \times k$ ON matrix $\mathbf{Q}_k$ and a $k \times n$ matrix $\mathbf{B}_k = \mathbf{Q}_k^*\mathbf{A}$ such that $\|\mathbf{A} - \mathbf{Q}_k\mathbf{B}_k\| \leq \varepsilon$. One can easily verify that after the algorithm finishes, we have $\mathbf{A} = \mathbf{Q}_j\mathbf{B}_j + \mathbf{A}_j$ for any $j = 0, 1, 2, \ldots, k$.

Pick the largest remaining column. Suppose we instantiate line (4) by letting $\mathbf{y}$ be simply the largest column of the remainder matrix $\mathbf{A}_{k-1}$.

$$\text{(4)} \qquad \text{Set } j_k = \text{argmax}\{\|\mathbf{A}_{k-1}(:, j)\| : j = 1, 2, \ldots, n\} \text{ and then } \mathbf{y} = \mathbf{A}_{k-1}(:, j_k).$$

With this choice, the algorithm in Figure 2.6 is *precisely* column pivoted Gram-Schmidt (CPQR). This algorithm is reasonably efficient, and often leads to fairly close to optimal low-rank approximation. For instance, when the singular values of $\mathbf{A}$ decay rapidly, CPQR typically determines a numerical rank k that is typically reasonably close to the theoretically exact ε-rank. However, this is not always the case even when the singular values decay rapidly, and the results can be quite poor when the singular values decay slowly.

Pick the locally optimal vector. A choice that is natural, and is conceptually very simple is to pick the vector $\mathbf{y}$ by solving the obvious minimization problem:

$$\text{(4)} \qquad \mathbf{y} = \text{argmin}\{\|\mathbf{A}_{k-1} - \mathbf{y}\mathbf{y}^*\mathbf{A}_{k-1}\| : \|\mathbf{y}\| = 1\}.$$

With this choice, the algorithm will produce matrices that attain the theoretically optimal precision

$$\|\mathbf{A} - \mathbf{Q}_j\mathbf{B}_j\| = \sigma_{j+1}.$$

This tells us that the greediness of the algorithm is not a problem. However, solving the local minimization problem is sufficiently computationally hard that this algorithm is not particularly practical.

A randomized selection strategy. Suppose now that we pick $\mathbf{y}$ by forming a linear combination of the columns of $\mathbf{A}_{k-1}$ with the expansion weights drawn from a normalized Gaussian distribution:

$$\text{(4)} \qquad \text{Draw a Gaussian random vector } \mathbf{g} \in \mathbb{R}^n \text{ and set } \mathbf{y} = \mathbf{A}_{k-1}\mathbf{g}.$$

With this choice, the algorithm becomes logically equivalent to the basic randomized SVD given in Figure 2.1. This means that this choice often leads to a factorization that is close to optimally accurate, and is also computationally efficient. One can attain higher accuracy by trading away some computational efficiency by incorporating a couple of steps of power iteration, and choosing $\mathbf{y} = \left(\mathbf{A}_{k-1}\mathbf{A}_{k-1}^*\right)^q\mathbf{A}_{k-1}\mathbf{g}$ for some small integer q.

2.9.3. A blocked updating algorithm. A key benefit of the randomized greedy algorithm described in Section 2.9.2 is that it can easily be *blocked*. In other words, given a block size ℓ, we can at each step of the iteration draw a set of ℓ Gaussian random vectors, compute the corresponding sample vectors, and then extend the factors $\mathbf{Q}$ and $\mathbf{B}$ by adding ℓ columns and ℓ rows at a time, respectively. The resulting algorithm is shown in Figure 2.7.

(1) $\mathbf{Q} = [\,]; \mathbf{B} = [\,];$
(2) **while** $\|\mathbf{A}\| > \varepsilon$
(3) Draw an $n \times \ell$ random matrix $\mathbf{R}$.
(4) Compute the $m \times \ell$ matrix $\mathbf{Q}_{\text{new}} = \texttt{qr}(\mathbf{AR}, 0)$.
(5) $\mathbf{B}_{\text{new}} = \mathbf{Q}_{\text{new}}^* \mathbf{A}$
(6) $\mathbf{Q} = [\mathbf{Q}\ \mathbf{Q}_{\text{new}}]$
(7) $\mathbf{B} = \begin{bmatrix} \mathbf{B} \\ \mathbf{B}_{\text{new}} \end{bmatrix}$
(8) $\mathbf{A} = \mathbf{A} - \mathbf{Q}_{\text{new}} \mathbf{B}_{\text{new}}$
(9) **end while**

FIGURE 2.7. A greedy algorithm for building a low-rank approximation to a given $m \times n$ matrix $\mathbf{A}$ that is accurate to within a given precision ε. This algorithm is a blocked analogue of the method described in Figure 2.6 and takes as input a block size ℓ. Its output is an ON matrix $\mathbf{Q}$ of size $m \times k$ (where k is a multiple of ℓ) and a $k \times n$ matrix $\mathbf{B}$ such that $\|\mathbf{A} - \mathbf{QB}\| \leq \varepsilon$. For higher accuracy, one can incorporate a couple of steps of power iteration and set $\mathbf{Q}_{\text{new}} = \texttt{qr}((\mathbf{AA}^*)^q \mathbf{AR}, 0)$ on Line (4).

(1) $\mathbf{Q}_0 = [\,]; \mathbf{B}_0 = [\,];$
(2) **for** $j = 1, 2, 3, \ldots$
(3) Draw a Gaussian random vector $\mathbf{g}_j \in \mathbb{R}^n$ and set $\mathbf{y}_j = \mathbf{A}\mathbf{g}_j$
(4) Set $\mathbf{z}_j = \mathbf{y}_j - \mathbf{Q}_{j-1}\mathbf{Q}_{j-1}^* \mathbf{y}_j$ and then $\mathbf{q}_j = \mathbf{z}_j / |\mathbf{z}_j|$.
(5) $\mathbf{Q}_j = [\mathbf{Q}_{j-1}\ \mathbf{q}_j]$
(6) $\mathbf{B}_j = \begin{bmatrix} \mathbf{B}_{j-1} \\ \mathbf{q}_j^* \mathbf{A} \end{bmatrix}$
(7) **end for**

FIGURE 2.8. A randomized range finder that builds an ON-basis $\{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \ldots\}$ for the range of $\mathbf{A}$ one vector at a time. This algorithm is mathematically equivalent to the basic RSVD in Figure 2.1 in the sense that if $\mathbf{G} = [\mathbf{g}_1\ \mathbf{g}_2\ \mathbf{g}_3\ \ldots]$, then the vectors $\{\mathbf{q}_j\}_{j=1}^p$ form an ON-basis for $\mathbf{AG}(:, 1:p)$ for both methods. Observe that $\mathbf{z}_j = \big(\mathbf{A} - \mathbf{Q}_{j-1}\mathbf{Q}_{j-1}^* \mathbf{A}\big)\mathbf{g}_j$, cf. (2.21).

2.10. Adaptive rank determination without updating the matrix

The techniques described in Section 2.9 for computing a low rank approximation to a matrix $\mathbf{A}$ that is valid to a *given tolerance* (as opposed to a *given rank*) are highly computationally efficient whenever the matrix $\mathbf{A}$ itself can be easily updated (e.g. a dense matrix stored in RAM). In this section, we describe algorithms for solving the "given tolerance" problem that do not need to explicitly update the matrix, which comes in handy for sparse matrices, for matrices stored out-of-core, for matrices defined implicitly, etc. The price we have to pay is that we can no longer *guarantee* that the computed factorization $\mathbf{A} \approx \mathbf{QB}$ is accurate to precision ε. Rather, for any given tolerated risk p, we can say that $\|\mathbf{A} - \mathbf{QB}\| \leq \varepsilon$ with probability at least $1 - p$.

As a preliminary step in deriving the update-free scheme, let us reformulate the basic RSVD in Figure 2.1 as the sequential algorithm shown in Figure 2.8 that builds the matrices $\mathbf{Q}$ and $\mathbf{B}$ one vector at a time. We observe that this method is similar to the greedy template shown in Figure 2.6, except that there is not an immediately obvious way to tell when $\|\mathbf{A} - \mathbf{Q}_j \mathbf{B}_j\|$ becomes small enough. However, it is possible to *estimate* this quantity quite easily. The idea is that once $\mathbf{Q}_j$ becomes large enough to capture "most" of the range of $\mathbf{A}$, then then sample vectors $\mathbf{y}_j$ drawn will all approximately lie in the span of $\mathbf{Q}_j$, which is to say that the projected vectors $\mathbf{z}_j$ will become very

Draw standard Gaussian vectors $\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(r)}$ of length n.
For $i = 1, 2, \ldots, r$, compute $\mathbf{y}^{(i)} = \mathbf{A}\boldsymbol{\omega}^{(i)}$.
$j = 0$.
$\mathbf{Q}^{(0)} = [\,]$, the $m \times 0$ empty matrix.
while $\max\left\{\|\mathbf{y}^{(j+1)}\|, \|\mathbf{y}^{(j+2)}\|, \ldots, \|\mathbf{y}^{(j+r)}\|\right\} > \varepsilon/(10\sqrt{2/\pi})$,
    $j = j + 1$.
    Overwrite $\mathbf{y}^{(j)}$ by $\mathbf{y}^{(j)} - \mathbf{Q}^{(j-1)}(\mathbf{Q}^{(j-1)})^*\mathbf{y}^{(j)}$.
    $\mathbf{q}^{(j)} = \mathbf{y}^{(j)}/|\mathbf{y}^{(j)}|$.
    $\mathbf{Q}^{(j)} = [\mathbf{Q}^{(j-1)}\ \mathbf{q}^{(j)}]$.
    Draw a standard Gaussian vector $\boldsymbol{\omega}^{(j+r)}$ of length n.
    $\mathbf{y}^{(j+r)} = \left(\mathbf{I} - \mathbf{Q}^{(j)}(\mathbf{Q}^{(j)})^*\right)\mathbf{A}\boldsymbol{\omega}^{(j+r)}$.
    for $i = (j+1), (j+2), \ldots, (j+r-1)$,
        Overwrite $\mathbf{y}^{(i)}$ by $\mathbf{y}^{(i)} - \mathbf{q}^{(j)}\,\langle \mathbf{q}^{(j)}, \mathbf{y}^{(i)}\rangle$.
    end for
end while
$\mathbf{Q} = \mathbf{Q}^{(j)}$.

FIGURE 2.9. A randomized range finder. Given an $m \times n$ matrix $\mathbf{A}$, a tolerance ε, and an integer r, the algorithm computes an orthonormal matrix $\mathbf{Q}$ such that $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \varepsilon$ holds with probability at least $1 - \min\{m, n\}10^{-r}$. (Adapted from Algorithm 4.2 of [**11**]).

small. In other words, once we start to see a sequence of vectors $\mathbf{z}_j$ that are all very small, we can reasonably deduce that the basis we have on hand very likely covers most of the range of $\mathbf{A}$.

To make the discussion in the previous paragraph more mathematically rigorous, let us first observe that each projected vector $\mathbf{z}_j$ satisfies the relation

$$\mathbf{z}_j = \mathbf{y}_j - \mathbf{Q}_{j-1}\mathbf{Q}_{j-1}^*\mathbf{y}_j = \mathbf{A}\mathbf{g}_j - \mathbf{Q}_{j-1}\mathbf{Q}_{j-1}^*\mathbf{A}\mathbf{g}_j = \big(\mathbf{A} - \mathbf{Q}_{j-1}\mathbf{Q}_{j-1}^*\mathbf{A}\big)\mathbf{g}_j. \tag{2.21}$$

It turns out that if $\mathbf{B}$ is any $m \times n$ matrix, and $\mathbf{g} \in \mathbb{R}^n$ is a standard Gaussian vector, then $\mathbb{E}[\|\mathbf{B}\mathbf{g}\|^2] = \|\mathbf{B}\|_{\text{Fro}}^2$. (We will prove this claim shortly.) In other words, by looking at the norm of the vectors $\mathbf{z}_j$, we obtain estimates of $\|\mathbf{A} - \mathbf{Q}_{j-1}\mathbf{Q}_{j-1}\mathbf{A}\|_{\text{Fro}}$. Since the Frobenius norm is an upper bound for the spectral norm, we consequently also obtain an upper bound of the spectral norm. The precise result that we need is the following, cf. [**20**, Sec. 3.4] and [**11**, Lemma 4.1].

LEMMA 4. *Let $\mathbf{B}$ be a real $m \times n$ matrix. Fix a positive integer r and a real number $\alpha \in (0, 1)$. Draw an independent family $\{\mathbf{g}^{(i)} : i = 1, 2, \ldots, r\}$ of standard Gaussian vectors. Then*

$$\|\mathbf{B}\| \leq \frac{1}{\alpha}\sqrt{\frac{2}{\pi}}\max_{i=1,\ldots,r}\|\mathbf{B}\mathbf{g}^{(i)}\|$$

except with probability α^r.

In applying this result, we set $\alpha = 1/10$, whence it follows that if $\|\mathbf{z}_j\|$ is smaller than the resulting threshold for r vectors in a row, then $\|\mathbf{A} - \mathbf{Q}_j\mathbf{B}_j\| \leq \varepsilon$ with probability at least $1 - 10^{-r}$. The resulting algorithm is shown in Figure 2.9. Observe that choosing $\alpha = 1/10$ will work well only if the singular values decay reasonably fast.

All that remains at this point is to prove our claim that if $\mathbf{g}$ is a Gaussian random vector, then $\|\mathbf{B}\mathbf{g}\|$ is a reasonable estimator for the Frobenius norm of $\mathbf{B}$.

LEMMA 5. *Let $\mathbf{B}$ be an $m \times n$ real matrix. Draw a vector $\mathbf{g} \in \mathbb{R}^n$ from a normalized Gaussian distribution. Then*

$$\mathbb{E}[\|\mathbf{B}\mathbf{g}\|^2] = \|\mathbf{B}\|_{\text{Fro}}^2.$$

PROOF. Let $\mathbf{B}$ have the singular value decomposition $\mathbf{B} = \mathbf{U}\mathbf{D}\mathbf{V}^*$, and set $\tilde{\mathbf{g}} = \mathbf{V}^*\mathbf{g}$. Then

$$\|\mathbf{B}\mathbf{g}\|^2 = \|\mathbf{U}\mathbf{D}\mathbf{V}^*\mathbf{g}\|^2 = \|\mathbf{U}\mathbf{D}\tilde{\mathbf{g}}\|^2 = \|\mathbf{D}\tilde{\mathbf{g}}\|^2 = \sum_{j=1}^{n} \sigma_j^2 \tilde{g}_j^2.$$

Then observe that since the distribution of Gaussian vectors is rotationally invariant, the vector $\tilde{\mathbf{g}} = \mathbf{V}^*\mathbf{g}$ is also a standardized Gaussian vector, and so $\mathbb{E}[\tilde{g}_j^2] = 1$. It follows that

$$\mathbb{E}[\|\mathbf{B}\mathbf{g}\|^2] = \mathbb{E}\left[\sum_{j=1}^{n} \sigma_j^2 \tilde{g}_j^2\right] = \sum_{j=1}^{n} \sigma_j^2 \mathbb{E}[\tilde{g}_j^2] = \sum_{j=1}^{n} \sigma_j^2 = \|\mathbf{B}\|_{\text{Fro}}^2,$$

which completes the proof. □

CHAPTER 3

Power iteration and Krylov methods

Consider a basic question: Suppose that we are given a matrix $\mathbf{A}$ and seek to compute approximations to the dominant eigenvectors and the corresponding eigenvalues. The technique in Section 1.5.4 is one option that works well for dense matrices whose singular values decay fairly rapidly. In this section, we briefly survey an alternative set of techniques based on iterations and Krylov methods. These techniques have at least two persuasive advantages:

- They interact with the matrix only via the matrix-vector multiplication. This is particularly good for very large sparse matrices.
- They are in certain environments more accurate than the methods described in Section 1.5.4.

For simplicity, we restrict attention to square matrices.

3.1. The basic power iteration

Let $\mathbf{A}$ be an $n \times n$ real symmetric matrix whose eigenvalues decay in magnitude. Suppose first that we seek simply to compute an approximation to the dominante eigenvalue and its corresponding eigenvector. We suppose that $\mathbf{A}$ has an eigenvalue decomposition

$$\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^* = \sum_{j=1}^{n} \lambda_j \, \mathbf{v}_j \mathbf{v}_j^*, \tag{3.1}$$

with the eigenvalues ordered by magnitude so that

$$|\lambda_1| \geq |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_1| \geq 0. \tag{3.2}$$

Of course, the assumption is at this point that we do not know $\mathbf{V}$ and $\mathbf{D}$, we use them simply for the analysis. Now let us draw a random vector $\mathbf{x}$ as the start of the iteration. Since $\{\mathbf{v}_j\}_{j=1}^n$ forms an orthonormal basis for $\mathbb{R}^n$, the vector $\mathbf{x}$ has an expansion

$$\mathbf{x} = \sum_{j}^{n} c_j \, \mathbf{v}_j, \tag{3.3}$$

for some (unknown) coefficients $\{c_j\}_{j=1}^n$. Using that for any positive integer p we have $\mathbf{A}^p \mathbf{v}_j = \lambda_j^p \mathbf{v}_j$, we easily find that

$$\mathbf{A}^p \mathbf{x} = \sum_{j}^{n} c_j \, \lambda_j^p \, \mathbf{v}_j.$$

We see that if the eigenvalues strictly decay, so that $|\lambda_1| > |\lambda_2|$, then the expansion coefficient of $\mathbf{v}_1$ will as $p \to \infty$ become more and more dominant (compared to the other terms), and the vector $\mathbf{A}^p \mathbf{x}$ will start to align better and better with $\mathbf{v}_1$.

3.2. Subspace iteration

The power iteration described in Section 3.1 is a slightly primitive algorithm. Its convergence rate is not that good unless the ratio $|\lambda_2|/|\lambda_1|$ is small. Moreover, it only computes a single eigenvector. Suppose now that we seek to determine approximations to the top ℓ eigenvectors. A natural idea is then to run ℓ independent instantiations of the single vector power iteration. The resulting algorithm is shown as the "Basic subspace iteration" in Figure 3.1. This algorithm shows how to compute a sample matrix $\mathbf{Y}_p = \mathbf{A}^p \mathbf{G}$, where $\mathbf{G}$ is a Gaussian random matrix. The

Basic subspace iteration:

$\mathbf{G} = \texttt{randn}(n, \ell)$
$\mathbf{Y}_0 = \mathbf{G}$
for $j = 1, 2, 3, \ldots, p$
    $\mathbf{Y}_j = \mathbf{A}\mathbf{Y}_{j-1}$
end for
$[\mathbf{Q}, \mathbf{R}] = \texttt{qr}(\mathbf{Y}_p, 0)$

Stabilized subspace iteration:

$\mathbf{G} = \texttt{randn}(n, \ell)$
$\mathbf{Q}_0 = \mathbf{G}$
for $j = 1, 2, 3, \ldots, p$
    $\mathbf{Z}_j = \mathbf{A}\mathbf{Q}_{j-1}$
    $[\mathbf{Q}_j, \mathbf{R}_j] = \texttt{qr}(\mathbf{Z}_j, 0)$
end for

FIGURE 3.1. Two versions of subspace iteration that are mathematically equivalent when executed in exact arithmetic (cf. Theorem 6). The basic version is faster and often works well. However, when p is high and/or the singular values of $\mathbf{A}$ decay rapidly, the basic version loses accuracy due to round-off errors; in this case, the stabilized version should be used.

columns in $\mathbf{Y}_p$ will consist of independent samples from the range of $\mathbf{A}$, and upon orthonormalization we obtain a matrix $\mathbf{Q}$ whose columns form an orthonormal basis for a space that approximately aligns with the space spanned by the ℓ dominant eigenvectors of $\mathbf{A}$.

The basic subspace iteration shown on the left in Figure 3.1 is numerically fragile. The problem is that as j increases, all the columns in the matrix $\mathbf{Y}_j = \mathbf{A}^j\mathbf{G}$ will align more and more closely with the dominant eigenvectors. Due to the effects of round-off errors, the contributions from the "lesser" eigenvectors will as j increases lose accuracy, and will eventually get lost completely. To combat this effect, it is common to perform orthonormalization in between each iteration, which results in the stabilized scheme shown on the right in Figure 3.1.

We will next prove that the two schemes shown in Figure 3.1 are mathematically equivalent when executed in exact arithmetic.

THEOREM 6. *Let $\mathbf{A}$ be an $n \times n$ matrix. Let ℓ be an integer such that $1 \leq \ell \leq n$, and let $\mathbf{G}$ be an $n \times \ell$ Gaussian matrix. Suppose that the rank of $\mathbf{A}$ is at least ℓ. Let $\mathbf{Q}$ and $\mathbf{Q}_p$ denote the outputs of the two algorithms shown in Figure 3.1. Then*

$$Col(\mathbf{Q}) = Col(\mathbf{Q}_p) = Col(\mathbf{A}^p\mathbf{G}).$$

Before proving Theorem 6, let us first establish a simple auxiliary result:

LEMMA 7. *Let $\mathbf{B}$ be an $m \times k$ matrix where $m \geq k$, and let $\mathbf{H}$ be a $k \times k$ matrix. Then*

$$Col(\mathbf{BH}) \subseteq Col(\mathbf{B}).$$

If additionally, $\mathbf{H}$ is invertible, then

$$Col(\mathbf{BH}) = Col(\mathbf{B}).$$

PROOF OF LEMMA 7. For first that $\mathbf{x} \in \text{Col}(\mathbf{BH})$. Then there exists a vector $\mathbf{y}$ such that $\mathbf{x} = \mathbf{BHy}$. Set $\mathbf{y}' = \mathbf{Hy}$. Then $\mathbf{x} = \mathbf{BHy} = \mathbf{By}'$ so $\mathbf{x} \in \text{Col}(\mathbf{B})$. Next suppose that $\mathbf{H}$ is invertible, and that $\mathbf{x} \in \text{Col}(\mathbf{B})$. Then there exists a vector $\mathbf{y}$ such that $\mathbf{x} = \mathbf{By}$. Set $\mathbf{y}' = \mathbf{H}^{-1}\mathbf{y}$. Then $\mathbf{x} = \mathbf{By} = \mathbf{BHy}'$ so $\mathbf{x} \in \text{Col}(\mathbf{BH})$. □

PROOF OF THEOREM 6. Observe that $\mathbf{Y}_j = \mathbf{A}^j\mathbf{G}$. Observe that when $\mathbf{A}$ is symmetric, we have

$$\text{Rank}(\mathbf{A}) = \text{Rank}(\mathbf{A}^j), \qquad j = 1, 2, 3, \ldots$$

Due to properties of Gaussian random matrices (see ???), this means that $\mathbf{Y}_j$ has precisely rank ℓ for every j. This means that $\mathbf{R}$ is invertible, and so Lemma 7 implies that

$$\text{Col}(\mathbf{Q}) = \text{Col}(\mathbf{A}^p\mathbf{G}).$$

It remains to prove that $\text{Col}(\mathbf{Z}_j) = \text{Col}(\mathbf{Y}_j)$ for every j (that is, that orthonormalizing between each application of $\mathbf{A}$ does not change the span of the vectors in the sampling matrix). For $j = 1$, this is trivially true since

$$\mathbf{Y}_1 = \mathbf{AG} = \mathbf{Z}_1.$$

Since $\mathbf{Z}_1$ has rank ℓ, it follows that $\mathbf{R}_1$ is invertible. Using that $\mathbf{Z}_1 = \mathbf{Q}_1\mathbf{R}_1$, we then find that

$$\mathbf{Z}_2 = \mathbf{A}\mathbf{Q}_1 = \mathbf{A}\mathbf{Z}_1\mathbf{R}_1^{-1} = \mathbf{A}^2\mathbf{G}\mathbf{R}_1^{-1}.$$

Lemma 7 now asserts that $\text{Col}(\mathbf{Z}_2) = \text{Col}(\mathbf{Y}_2)$. It follows that $\mathbf{Z}_2$ has full rank, and consequently, the factor $\mathbf{R}_2$ in the factorization $\mathbf{Z}_2 = \mathbf{Q}_2\mathbf{R}_2$ is invertible. Then

$$\mathbf{Z}_3 = \mathbf{A}\mathbf{Q}_2 = \mathbf{A}\mathbf{Z}_2\mathbf{R}_2^{-1} = \mathbf{A}^3\mathbf{G}\mathbf{R}_2^{-1}\mathbf{R}_1^{-1}.$$

Lemma 7 asserts that $\text{Col}(\mathbf{Z}_3) = \text{Col}(\mathbf{Y}_3)$. Continuing in the obvious way, we establish the result for every j. □

REMARK 3.1 (Non-symmetric matrices). For a matrix $\mathbf{A}$ that is not symmetric, the power method described in this section can be modified to a technique for computing a matrix $\mathbf{Q}$ whose columns form an ON basis for a space that approximates the space spanned by the ℓ dominant left singular vectors of $\mathbf{A}$. The idea is to form the sample matrix $\mathbf{Y}_p = \left(\mathbf{A}\mathbf{A}^*\right)^p\mathbf{G}$, and then simply orthonormalize to obtain $[\mathbf{Q}_p, \sim, \sim] = \texttt{qr}(\mathbf{Y}_p, 0)$. In fact, we described this algorithm in Section 2.7.1 with the result summarized in Figure 2.4. A more numerically stable technique is described in Remark 2.5.

3.3. The general idea of Krylov methods

Suppose that $\mathbf{A}$ is an $n \times n$ Hermitian matrix, and let us consider the basic (single-vector) power iteration described in Section 3.1. The idea is to take some starting vector $\mathbf{g}$ (using a Gaussian random vector is often a good choice), and then construct a sequence by setting $\mathbf{y}_1 = \mathbf{A}\mathbf{g}$, and $\mathbf{y}_{j+1} = \mathbf{A}\mathbf{y}_j$. The end result is that after k steps, we use the vector $\mathbf{y}_k/\|\mathbf{y}_k\|$ as an approximation to the dominant eigenvector. Note that the information in the vectors $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{k-1}\}$ is not used. To simplify slightly, the idea in Krylov methods is to use the full information contained in these vectors to simultaneously construct approximations to the k leading eigenvalues and eigenvectors. To be precise, suppose that we are given an $n \times n$ matrix $\mathbf{A}$, some starting vector $\mathbf{g}$, and a positive integer k. We then define the *Krylov subspace* $\mathcal{K}_k(\mathbf{A}, \mathbf{g})$ as linear space

$$\mathcal{K}_k(\mathbf{A}, \mathbf{g}) = \text{Span}(\mathbf{g}, \mathbf{A}\mathbf{g}, \mathbf{A}^2\mathbf{g}, \ldots, \mathbf{A}^{k-1}\mathbf{g}).$$

The idea is then to restrict $\mathbf{A}$ to this lower dimensional space, and use the eigenvalues of the resulting $k \times k$ matrix to approximate the eigenvalues of $\mathbf{A}$. To formalize, suppose that $\{\mathbf{q}_j\}_{j=1}^k$ is an ON set of vectors obtained by performing Gram-Schmidt on the set $\{\mathbf{A}^{j-1}\mathbf{g}\}_{j=1}^k$. Then set

$$\mathbf{Q} = \left[\mathbf{q}_1\ \mathbf{q}_2\ \cdots\ \mathbf{q}_k\right],$$

and define the matrix $\mathbf{T}$ via

$$\mathbf{T} = \mathbf{Q}^*\mathbf{A}\mathbf{Q}.$$

The relationship between the eigenvalues of $\mathbf{T}$ and the eigenvalues of $\mathbf{A}$ is very well understood. A simple property to prove is that if the eigenvalues of $\mathbf{A}$ decay rapidly in magnitude, then the dominant eigenvalues of $\mathbf{T}$ closely approximate the dominant eigenvalues of $\mathbf{A}$.

3.4. The Lanzcos algorithm

3.4.1. Problem formulation. Let $\mathbf{A}$ be an $n \times n$ Hermitian matrix, and let $\mathbf{q}_1$ be a given starting vector of unit length (a good generic choice is to draw a Gaussian random vector $\mathbf{g}$ and to set $\mathbf{q}_1 = \mathbf{g}/\|\mathbf{g}\|$). The idea of the Lanczos iteration is to build, one vector at a time, an ON basis $\{\mathbf{q}_j\}_{j=1}^k$ for the Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{q}_1) = \text{Span}(\mathbf{q}_1, \mathbf{A}\mathbf{q}_1, \mathbf{A}^2\mathbf{q}_1, \ldots, \mathbf{A}^{k-1}\mathbf{q}_1)$. We formulate this task as incrementally building a matrix factorization

$$\underset{n\times n}{\mathbf{A}} = \underset{n\times n}{\mathbf{Q}}\ \underset{n\times n}{\mathbf{T}}\ \underset{n\times n}{\mathbf{Q}^*}, \tag{3.4}$$

where $\mathbf{Q}$ is a unitary matrix whose first column is $\mathbf{q}_1$, and where $\mathbf{T}$ is tridiagonal (and Hermitian). In our derivation, we will treat the factorization as a full and exact factorization, but typically the objective is to stop after k steps and obtain an approximate factorization

$$\underset{n\times n}{\mathbf{A}} \approx \underset{n\times k}{\mathbf{Q}(:, 1:k)}\ \underset{k\times k}{\mathbf{T}(1:k, 1:k)}\ \underset{k\times n}{\mathbf{Q}(:, 1:k)^*}. \tag{3.5}$$

In terms of notation, we use

$$\mathbf{Q} = [\mathbf{q}_1\ \mathbf{q}_2\ \ldots], \qquad \mathbf{T} = \begin{bmatrix} t_{11} & t_{12} & 0 & 0 & \cdots \\ t_{21} & t_{22} & t_{23} & 0 & \cdots \\ 0 & t_{23} & t_{33} & t_{34} & \cdots \\ 0 & 0 & \vdots & \vdots & \end{bmatrix}.$$

3.4.2. The Lanczos iteration. We will simultaneously prove that the factorization (3.4) exists, and derive an algorithm for computing it.

THEOREM 8. *Let $\mathbf{A}$ be an $n \times n$ matrix, and let $\mathbf{q}_1$ be a unit vector. Then:*

(a) *There exist a triangular matrix $\mathbf{T}$ and an orthonormal matrix $\mathbf{Q}$ whose first column is $\mathbf{q}_1$ such that*

$$\mathbf{A} = \mathbf{Q}\mathbf{T}\mathbf{Q}^*.$$

(b) *If $\mathbf{q}_1$ does not belong to an invariant subspace of $\mathbf{A}$, then the factorization is unique, up to scaling of the columns of $\mathbf{Q}$ by a unitary complex number.*

Proof: Let us rewrite the factorization $\mathbf{A} = \mathbf{Q}\mathbf{T}\mathbf{Q}^*$ as

$$\mathbf{A}\mathbf{Q} = \mathbf{Q}\mathbf{T}. \tag{3.6}$$

Next, let us write out (3.6) column by column. For the first column, we get

$$\mathbf{A}\,\mathbf{q}_1 = \mathbf{q}_1\,t_{11} + \mathbf{q}_2\,t_{21}.$$

Multiplying from the left by $\mathbf{q}_1^*$ we find

$$\mathbf{q}_1^*\,\mathbf{A}\,\mathbf{q}_1 = t_{11},$$

which uniquely determines t_{11}. We temporarily assume that $\mathbf{A}\,\mathbf{q}_1 - t_{11}\,\mathbf{q}_1 \neq \mathbf{0}$. Then t_{21} and $\mathbf{q}_2$ are both determined by the relation

$$t_{21}\,\mathbf{q}_2 = \mathbf{A}\,\mathbf{q}_1 - t_{11}\,\mathbf{q}_1. \tag{3.7}$$

To be precise, (3.7) does not quite determine $\{t_{21},\ \mathbf{q}_2\}$ uniquely. We see that if $\{t_{21},\ \mathbf{q}_2\}$ to (3.7) is one solution of (3.7), and if θ is a complex number such that $|\theta| = 1$, then $\{\theta\,t_{21},\ \bar{\theta}\,\mathbf{q}_2\}$ is another solution. This is the only ambiguity involved, however (as long as the right hand side is not zero). Next we proceed to compare the second column in (3.6):

$$\mathbf{A}\,\mathbf{q}_2 = \mathbf{q}_1\,t_{12} + \mathbf{q}_2\,t_{22} + \mathbf{q}_3\,t_{32}.$$

Multiply by $\mathbf{q}_2^*$ to find t_{22}, then determine $\{t_{32},\ \mathbf{q}_3\}$ from

$$t_{32}\,\mathbf{q}_3 = \mathbf{A}\,\mathbf{q}_2 - t_{12}\,\mathbf{q}_1 - t_{22}\,\mathbf{q}_2. \tag{3.8}$$

We see that the process can be continued until completion, with all quantities uniquely determined (up to scaling by a unitary complex scalar) as long as at step k, the vector

$$\mathbf{A}\,\mathbf{q}_k - t_{k,k-1}\,\mathbf{q}_{k-1} - t_{k,k}\,\mathbf{q}_k. \tag{3.9}$$

does not equal zero.

Now let us consider the special case where the vector in (3.9) does happen to be zero at step k. In this case, we must have $t_{k+1,k} = 0$, but the vector $\mathbf{q}_{k+1}$ is not uniquely determined. *Existence* of the factorization is not a problem in this case — simply choose any vector $\mathbf{q}_{k+1}$ that is of unit length and is perpendicular to $\text{Span}\{\mathbf{q}_j\}_{j=1}^k$. To establish the claim of *uniqueness* in part (b), suppose that

$$\mathbf{A}\,\mathbf{q}_k - t_{k,k-1}\,\mathbf{q}_{k-1} - t_{k,k}\,\mathbf{q}_k = \mathbf{0}, \tag{3.10}$$

and consider the subspace

$$V = \text{Span}\{\mathbf{q}_1,\ \mathbf{q}_2,\ \ldots\ \mathbf{q}_k\}.$$

We will prove that when (3.10) holds, V is an invariant subspace of $\mathbf{A}$. Suppose $\mathbf{v} \in V$. Then $\mathbf{v} = \sum_{j=1}^k c_j\mathbf{q}_j$ for some scalars $\{c_j\}$. Then $\mathbf{A}\mathbf{v} = \sum_{j=1}^k c_j\mathbf{A}\mathbf{q}_j$. When $j < k$ we know that $\mathbf{A}_j\mathbf{q}_j \in V$ by construction, and the fact that $\mathbf{A}\mathbf{q}_k \in V$ follows from (3.10). Consequently, V is an invariant subspace of $\mathbf{A}$ which contradicts the assumptions in (b) since $\mathbf{q}_1 \in V$.

The procedure is summarized in Figure 3.2. Some remarks on the theoretical result:

(1) If the initial vector $\mathbf{q}_1$ is a random vector drawn from a uniform distribution on the unit sphere in $\mathbb{C}^n$, then the likelihood of $\mathbf{q}_1$ being exactly contained in some invariant subspace is zero.
(2) While the likelihood of any $t_{j-i,j}$ being precisely zero is zero, it can in practice easily be a very small number. This means that the procedure is highly unstable — a small perturbation in $\mathbf{q}_1$ leads to dramatic changes in $\mathbf{Q}$.

At this point, we have derived an algorithm for producing the matrix factorization $\mathbf{A} = \mathbf{Q}\mathbf{T}\mathbf{Q}^*$ with the properties stated in Theorem 8. We have not yet shown that for any k, the set $\{\mathbf{q}_j\}_{j=1}^k$ forms an ON basis for the Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{q}_1) = \text{Span}(\mathbf{q}_1, \mathbf{A}\mathbf{q}_1, \mathbf{A}^2\mathbf{q}_1, \ldots, \mathbf{A}^{k-1}\mathbf{q}_1)$. For simplicity, let us consider the generic case where $\mathbf{q}_1$ does not belong to any invariant subspace of $\mathbf{A}$. Looking at the first column of (3.6) we see that

$$\mathbf{A}\mathbf{q}_1 = \mathbf{q}_1 t_{1,1} + \mathbf{q}_2 t_{2,1}.$$

Since $t_{2,1}\mathbf{q}_2 = \mathbf{A}\mathbf{q}_1 - t_{1,1}\mathbf{q}_1$, we see that $\mathbf{q}_2 \in \text{Span}(\mathbf{q}_1, \mathbf{A}\mathbf{q}_1)$. Since $\{\mathbf{q}_1, \mathbf{q}_2\}$ is an orthonormal set, the claim holds for $k = 2$. For $k = 3$, let us consider the second column:

$$\mathbf{A}\mathbf{q}_2 = \mathbf{q}_1 t_{1,2} + \mathbf{q}_2 t_{2,2} + \mathbf{q}_3 t_{3,2}.$$

We see that

$$\mathbf{q}_3 \in \text{Span}(\mathbf{q}_1, \mathbf{q}_2, \mathbf{A}\mathbf{q}_2) \subseteq \text{Span}(\mathbf{q}_1, \mathbf{A}\mathbf{q}_1, \mathbf{A}^2\mathbf{q}_1),$$

where in the second relation we used that $\mathbf{q}_2 \in \text{Span}(\mathbf{q}_1, \mathbf{A}\mathbf{q}_1)$. The general claim now follows by a straightforward induction argument.

3.4.3. Implementation issues. The algorithm described in Figure 3.2 can in principle be executed while only keeping a couple of "long" vectors in memory. The recursion relation implicitly guarantees that the columns of $\mathbf{Q}$ remain orthonormal. In *practice*, this is hideously unstable. There are many ways to fix the situation, the easiest is to explicitly reorthogonalize $\mathbf{r}$ to $\text{Span}\{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k\}$.

If the orthogonality of the basis vectors is maintained scrupulously, then the instability in the map $\{\mathbf{A}, \mathbf{q}_1\} \mapsto \{\mathbf{Q}, \mathbf{T}\}$ is irrelevant. Indeed, whenever we are close to hitting an invariant subspace (*i.e.*, we come across a small $t_{k-1,k}$), then picking basically "any" direction for the new $\mathbf{q}_k$ (that is orthogonal to the previous basis vectors, of course) will work just fine.

To illustrate why a small $t_{k-1,k}$ is not only not bad, but actually a great thing, suppose that for some k, we discover that

$$\mathbf{r}_k = \mathbf{A}\,\mathbf{q}_k - t_{k-1,k}\,\mathbf{q}_{k-1} - t_{k,k}\,\mathbf{q}_k = \mathbf{0}.$$

What this means is that $\mathbf{T}$ and $\mathbf{Q}$ take the form

$$\mathbf{Q} = [\mathbf{Q}_{\text{L}}\ \mathbf{Q}_{\text{R}}], \qquad \mathbf{T} = \begin{bmatrix} \mathbf{T}_{\text{L}} & \mathbf{0} \\ \mathbf{0} & \mathbf{T}_{\text{R}} \end{bmatrix},$$

where the partition is done so that $\mathbf{Q}_{\text{L}}$ has k columns, and $\mathbf{T}_{\text{LL}}$ is of size $k \times k$. This is to say that

$$\mathbf{A} = \mathbf{Q}_{\text{L}}\,\mathbf{T}_{\text{L}}\,\mathbf{Q}_{\text{L}}^* + \mathbf{Q}_{\text{R}}\,\mathbf{T}_{\text{R}}\,\mathbf{Q}_{\text{R}}^*.$$

In other words, we have found one invariant subspace, and the eigenvalues of $\mathbf{T}_{\text{L}}$ are exactly the eigenvalues associated with this subspace!

In practice, what often happens is that we find some $t_{k-1,k}$ that is small but not quite zero. Then the vector $\mathbf{q}_{k+1}$ will have a large component of noise in its direction. As long as we make sure that $\mathbf{q}_{k+1}$ is truly orthogonal to all previous basis vectors, however, this is fine.

$t_{11} = \mathbf{q}_1^* \mathbf{A}\, \mathbf{q}_1$
$\mathbf{r}_1 = \mathbf{A}\, \mathbf{q}_1 - t_{11}\, \mathbf{q}_1$
for $j = 2, 3, \ldots, n$
  $t_{j-1,j} = ||\mathbf{r}_{j-1}||$
  $\mathbf{q}_j = \frac{1}{\|\mathbf{r}_{j-1}\|}\, \mathbf{r}_{j-1}$
  $t_{j,j} = \mathbf{q}_j^* \mathbf{A}\, \mathbf{q}_j$
  $\mathbf{r}_j = \mathbf{A}\, \mathbf{q}_j - t_{j-1,j}\, \mathbf{q}_{j-1} - t_{j,j}\, \mathbf{q}_j$
end

FIGURE 3.2. The basic Lanczos scheme. Given a matrix $\mathbf{A}$ (which is accessed only as an operator $\mathbf{x} \mapsto \mathbf{A}\,\mathbf{x}$) and a starting vector $\mathbf{q}_1$ such that $||\mathbf{q}_1|| = 1$, compute an orthonormal matrix $\mathbf{Q}$ and a triangular matrix $\mathbf{T}$ such that $\mathbf{A} = \mathbf{Q}\mathbf{T}\mathbf{Q}^*$, and $\mathbf{Q}(:,1) = \mathbf{q}_1$.

3.4.4. Partial factorization and convergence analysis. In the derivation of the Lanczos procedure, we treated it as a technique for computing a full factorization (3.4). In practice, the objective is generally to run just k steps (for $k \ll n$) to obtain an approximate rank-k factorization like (3.5). The hope here is that the eigenvalues of $\mathbf{T}(1:k, 1:k)$, or at least some of them, will form good approximations to the eigenvalues of $\mathbf{A}$. As it happens, this convergence is usually fast. Moreover, the convergence analysis is surprisingly sharp and informative. Here we provide just a brief preview, for details see [**19**, Lecture 36] or [**7**, Sec. 9.1].

As before, suppose that $\mathbf{A}$ is a given $n \times n$ Hermitian vector, and that $\mathbf{b}$ is a given starting vector for the Lanczos iteration (so that $\mathbf{q}_1 = \mathbf{b}/\|\mathbf{b}\|$). Let $\mathbf{Q}$ and $\mathbf{T}$ be the matrices resulting from the algorithm shown in Figure 3.2, and set

$$\mathbf{T}_k = \mathbf{T}(1:k, 1:k) = \begin{bmatrix} t_{11} & t_{12} & 0 & 0 & \cdots & 0 \\ t_{21} & t_{22} & t_{23} & 0 & \cdots & 0 \\ 0 & t_{32} & t_{33} & t_{34} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & 0 & & t_{kk} \end{bmatrix}, \qquad k = 1, 2, \ldots, n.$$

Let p_k denote the characteristic polynomial of $\mathbf{T}_k$ so that

$$p_k(z) = \det(\mathbf{T}_k - z\,\mathbf{I}_k).$$

Our objective is now to show that the zeros of p_k, which are the eigenvalues of $\mathbf{T}_k$, form good approximations to the dominant eigenvalues of $\mathbf{A}$. One result in this direction states that

$$\|p_k(\mathbf{A})\,\mathbf{b}\| \leq \|p(\mathbf{A})\,\mathbf{b}\|, \qquad \forall p \in P_k^{\infty}, \tag{3.11}$$

where P_k^{∞} is the set of k'th order "monic polynomials" which is the set of all polynomials of the form $p(z) = z^k + c_{k-1}z^{k-1} + c_{k-2}z^{k-2} + \cdots + c_1 z + c_0$ (observe that the leading term has coefficient one). At first glance, it might not be obvious why (3.11) is helpful. To elucidate the relation, let

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^*$$

denote the eigenvalue decomposition of $\mathbf{A}$. Then for any polynomial p we have

$$p(\mathbf{A}) = \mathbf{U}p(\mathbf{D})\mathbf{U}^*.$$

This means that

$$\|p(\mathbf{A})\,\mathbf{b}\| = \|\mathbf{U}p(\mathbf{D})\mathbf{U}^*\,\mathbf{b}\| = \{\text{Set } \mathbf{b}' = \mathbf{U}^*\mathbf{b}\} = \|p(\mathbf{D})\,\mathbf{b}'\| = \left(\sum_{i=1}^{n} |p(\lambda_i)\,b_i'|^2\right)^{1/2}.$$

To minimize this quantity, there is enormous pressure to pick a polynomial p whose zeros very closely approximate the dominant eigenvalues of $\mathbf{A}$, which is precisely what we want.

$\mathbf{T}_{11} = \mathbf{Q}_1^* \mathbf{A} \mathbf{Q}_1$
$\mathbf{R}_1 = \mathbf{A} \mathbf{Q}_1 - \mathbf{T}_{11} \mathbf{Q}_1$
for $j = 2, 3, \ldots, r$
  $[\mathbf{Q}_j, \mathbf{T}_{j,j-1}] = \mathtt{qr}(\mathbf{R}_{j-1})$
  $\mathbf{T}_{j,j} = \mathbf{Q}_j^* \mathbf{A} \mathbf{Q}_j$
  $\mathbf{R}_j = \mathbf{A} \mathbf{Q}_j - \mathbf{T}_{j-1,j} \mathbf{Q}_{j-1} - \mathbf{T}_{j,j} \mathbf{Q}_j$
end

FIGURE 3.3. The blocked Lanczos scheme. Given an $n \times n$ Hermitian matrix $\mathbf{A}$ and an $n \times b$ starting matrix $\mathbf{Q}_1$ such that $\mathbf{Q}_1^* \mathbf{Q}_1 = \mathbf{I}_b$, the scheme computes an orthonormal matrix $\mathbf{Q}$ and a block triangular matrix $\mathbf{T}$ such that $\mathbf{A} = \mathbf{Q}\mathbf{T}\mathbf{Q}^*$, and $\mathbf{Q}(:, 1:b) = \mathbf{Q}_1$.

3.4.5. Block Lanczos. The scheme described in Figure 3.2 admits a simple block formulation. The set-up is entirely analogous. We seek a factorization of the form (3.4), with

$$\mathbf{Q} = [\mathbf{Q}_1 \; \mathbf{Q}_2 \; \cdots \; \mathbf{Q}_r], \qquad \mathbf{T} = \begin{bmatrix} \mathbf{T}_{11} & \mathbf{T}_{21}^* & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{T}_{21} & \mathbf{T}_{22} & \mathbf{T}_{32}^* & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{T}_{23} & \mathbf{T}_{33} & \mathbf{T}_{43}^* & \cdots \\ \mathbf{0} & \mathbf{0} & \vdots & \vdots & \end{bmatrix}.$$

Each block $\mathbf{T}_{ij}$ is of size $b \times b$, each block $\mathbf{Q}_j$ is of size $n \times b$, and we assume that there are r blocks so that $n = rb$ (if n is not an even multiple of b the scheme can trivially be modified to allow the last block to be treated to have fewer columns). Suppose that $\mathbf{Q}_1$ is a given $n \times b$ orthonormal matrix. Then

$$\mathbf{Q}_1 \mathbf{T}_{11} + \mathbf{Q}_2 \mathbf{T}_{21} = \mathbf{A} \mathbf{Q}_1.$$

Left multiply by $\mathbf{Q}_1^*$ and used that $\mathbf{Q}_i^* \mathbf{Q}_j = \delta_{ij} I$ to obtain

$$\mathbf{T}_{11} = \mathbf{Q}_1^* \mathbf{A} \mathbf{Q}_1.$$

Then

$$\mathbf{Q}_2 \mathbf{T}_{21} = \mathbf{A} \mathbf{Q}_1 - \mathbf{Q}_1 \mathbf{T}_{11}.$$

Now preform a *QR-factorization* of the right-hand side to obtain the matrices $\mathbf{Q}_2$ and $\mathbf{T}_{21}$. Proceed analogously. The result is given in Figure 3.3.

3.5. The Arnoldi process

Let $\mathbf{A}$ be an $n \times n$ matrix, not necessarily Hermitian. In this case, it is too much to ask for a factorization (3.4) where the middle factor is tridiagonal. Instead, we relax the requirements and seek a factorization

$$\mathbf{A} = \mathbf{Q}\mathbf{H}\mathbf{Q}^*,$$

where $\mathbf{H}$ is in "Hessenberg" form

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{14} & \cdots \\ h_{21} & h_{22} & h_{23} & h_{24} & \cdots \\ 0 & h_{32} & h_{33} & h_{34} & \cdots \\ 0 & 0 & h_{43} & h_{44} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix}.$$

The derivation follows the Lanczos derivation completely. Rewrite the factorization as

$$\mathbf{A}\mathbf{Q} = \mathbf{Q}\mathbf{H}. \tag{3.12}$$

The first column of (3.12) evaluates to

$$\mathbf{A} \mathbf{q}_1 = \mathbf{q}_1 h_{11} + \mathbf{q}_2 h_{21}. \tag{3.13}$$

Multiplying from the left by $\mathbf{q}_1^*$ we find

$$h_{11} = \mathbf{q}_1^* \mathbf{A} \mathbf{q}_1.$$

$h_{11} = \mathbf{q}_1^* \mathbf{A}\, \mathbf{q}_1$
$\mathbf{r}_1 = \mathbf{A}\, \mathbf{q}_1 - h_{11}\, \mathbf{q}_1$
for $j = 2, 3, \ldots, n$
  $h_{j-1,j} = ||\mathbf{r}_{j-1}||$
  $\mathbf{q}_j = \frac{1}{\|\mathbf{r}_{j-1}\|}\, \mathbf{r}_{j-1}$
  $\mathbf{H}(1:j, j) = \mathbf{Q}(:, 1:j)^*\, \mathbf{A}\, \mathbf{q}_j$
  $\mathbf{r}_j = \mathbf{A}\, \mathbf{q}_j - \mathbf{Q}(:, 1:j)\mathbf{H}(1:j, j)$
end

FIGURE 3.4. The basic Arnoldi scheme. Given a matrix $\mathbf{A}$ (which is accessed only as an operator $\mathbf{x} \mapsto \mathbf{A}\,\mathbf{x}$) and a starting vector $\mathbf{q}_1$ such that $\|\mathbf{q}_1\| = 1$, compute an orthonormal matrix $\mathbf{Q}$ and a Hessenberg matrix $\mathbf{H}$ such that $\mathbf{A} = \mathbf{Q}\mathbf{H}\mathbf{Q}^*$, and $\mathbf{Q}(:, 1) = \mathbf{q}_1$.

$\mathbf{H}_{11} = \mathbf{Q}_1^*\, \mathbf{A}\, \mathbf{Q}_1$
$\mathbf{R} = \mathbf{A}\, \mathbf{Q}_1 - \mathbf{Q}_1 \mathbf{H}_{11}$
for $j = 2, 3, \ldots$
  $[\mathbf{Q}_j, \mathbf{R}_j] = \mathtt{qr}(\mathbf{R}, 0)$
  $\mathbf{H}_{j,j-1} = \mathbf{R}_j$
  $\begin{bmatrix} \mathbf{H}_{1,j} \\ \mathbf{H}_{2,j} \\ \vdots \\ \mathbf{H}_{j,j} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^* \\ \mathbf{Q}_2^* \\ \vdots \\ \mathbf{Q}_j^* \end{bmatrix} \mathbf{A}\, \mathbf{Q}_j$
  $\mathbf{R}_j = \mathbf{A}\, \mathbf{Q}_j - \sum_{i=1}^{j} \mathbf{Q}_i \mathbf{H}_{i,j}$
end

FIGURE 3.5. The blocked Arnoldi scheme. Given an $n \times n$ matrix $\mathbf{A}$ and an $n \times b$ starting block $\mathbf{Q}_1$ such that $\mathbf{Q}_1^* \mathbf{Q}_1 = \mathbf{I}_b$, the schemes computes an orthonormal matrix $\mathbf{Q}$ and a block Hessenberg matrix $\mathbf{H}$ such that $\mathbf{A} = \mathbf{Q}\mathbf{H}\mathbf{Q}^*$, and $\mathbf{Q}(:, 1:b) = \mathbf{Q}_1$.

Rearranging (3.13), we find

$$h_{21}\, \mathbf{q}_2 = \mathbf{A}\, \mathbf{q}_1 - h_{11}\, \mathbf{q}_1, \tag{3.14}$$

Note that if $\{h_{21},\, \mathbf{q}_2\}$ is any solution of (3.7), and if θ is a complex number such that $|\theta| = 1$, then $\{\theta\, h_{21},\, \bar{\theta}\, \mathbf{q}_2\}$ is another solution. Next we proceed to compare the second column in (3.12):

$$\mathbf{A}\, \mathbf{q}_2 = \mathbf{q}_1\, h_{12} + \mathbf{q}_2\, h_{22} + \mathbf{q}_3\, h_{32}.$$

Multiply by $\mathbf{q}_2^*$ to find h_{22}, then determine $\{h_{32},\, \mathbf{q}_3\}$ from

$$h_{32}\, \mathbf{q}_3 = \mathbf{A}\, \mathbf{q}_2 - h_{12}\, \mathbf{q}_1 - h_{22}\, \mathbf{q}_2, \tag{3.15}$$

and continue until completion. The resulting algorithm is shown in Figure 3.4.

3.5.1. Blocked Arnoldi. The modification of the basic Arnoldi scheme shown in Figure 3.4 is entirely analogous to the blocking of the Lanczos scheme in Section 3.4.5. The resulting scheme is shown in Figure 3.5.

3.6. A comparison of randomized methods and Krylov methods

The randomized methods described in Chapter 2 and the Krylov methods described in this chapter are similar in that they both seek to construct a "thin" orthonormal matrix $\mathbf{Q}$ such that

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}.$$

The two classes of techniques are similar in that the both interact with $\mathbf{A}$ only via its action on given sets of vectors or thin matrices. (For non-Hermitian matrices, the randomized techniques also need application of $\mathbf{A}^*$, while Krylov methods generally do not.)

Let us first compare a basic single-vector Krylov method (e.g. Figure 3.4) with the basic randomized method (e.g. Figure 2.1). To keep the comparison simple, suppose that we take ℓ steps of the Krylov method, and use a Gaussian random matrix with ℓ columns in the randomized scheme. Then, to simplify greatly, the methods compare as follows:

	Basic RSVD scheme (Figure 2.1)	Basic Arnoldi scheme (Figure 3.4)
Interaction with $\mathbf{A}$:	ℓ matrix-vector multiplications that can all be executed independently of each other. In particular, they can be executed in parallel as a matrix-matrix multiplication.	ℓ matrix-vector multiplications that have to be executed consecutively, one at a time.
Speed:	For a fixed ℓ, faster.	For a fixed ℓ, slower.
Accuracy:	For a fixed ℓ less accurate, in particular if the singular values decay slowly.	For a fixed ℓ, more accurate.
Other:	Also requires application of $\mathbf{A}^*$ to ℓ vectors.	Can sometimes get high accuracy not only on leading singular values, but also on the smallest ones.

The randomized power method described in Section 2.7 and the blocked Krylov methods described in Sections 3.4.5 and 3.5.1 represent two ways to bridge the gap between the extreme versions of the algorithms to obtain hybrid methods that have some of the best qualities of both schemes. These methods are conceptually very close (in particular the randomized method with an "extended sampling matrix" described in Section 2.7.3). The key point here is that by using a Gaussian random matrix as the starting point, one can often get away with taking a very small number of steps in the "power iteration" and thereby increasing computational speed (primarily through a reducing the amount of communication required).

3.7. Exercises

EXERCISE 1. Suppose that $\mathbf{A}$ is a real $n \times n$ symmetric matrix with eigenpairs $\{\lambda_j, \mathbf{v}_j\}_{j=1}^n$, ordered so that $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$. Define a sequence of vectors via $\mathbf{x}_0 = \texttt{randn}(n, 1)$, and then $\mathbf{x}_p = \mathbf{A}\mathbf{x}_{p-1}$, so that $\mathbf{x}_p = \mathbf{A}^p \mathbf{x}_0$.

(a) Set $\beta = |\lambda_2|/|\lambda_1|$ and $\mathbf{y}_p = (1/\|\mathbf{x}_p\|)\,\mathbf{x}_p$. Assume that $\lambda_1 = 1$ and that $\beta < 1$. Prove that as $p \to \infty$, the vectors $\{\mathbf{y}_p\}$ converge either to $\mathbf{v}_1$ or to $-\mathbf{v}_1$.
(b) What is the speed of convergence of $\{\mathbf{y}_p\}$?
(c) Assume again that $\beta = |\lambda_2|/|\lambda_1| < 1$, but now drop the assumption that $\lambda_1 = 1$. Prove that your answers in (a) and (b) are still correct, with the exception that if λ_1 is negative, then it is the vector $(-1)^p \mathbf{y}_p$ that converges instead.

In solving this problem, you are allowed to use that when $\mathbf{x}_0$ is drawn from a Gaussian distribution (which is what the Matlab command `randn` does) it has a series expansion $\mathbf{x}_0 = \sum_{j=1}^n c_j \mathbf{v}_j$ where $c_j \neq 0$ with probability 1. (In fact, one can prove that each c_j is a random variable drawn from a normalized Gaussian distribution.)

CHAPTER 4

Interpretation of data: The CUR and Interpolative Decompositions

4.1. The interpolative decomposition

4.1.1. Basic idea. Any matrix $\mathbf{A}$ of size $m \times n$ and rank k, where $k < \min(m, n)$, admits a so call "interpolative decomposition (ID)" which takes the form

$$\underset{m \times n}{\mathbf{A}} = \underset{m \times k}{\mathbf{C}} \quad \underset{k \times n}{\mathbf{Z}}, \tag{4.1}$$

where the matrix $\mathbf{C}$ is given by a subset of the columns of $\mathbf{A}$ and where $\mathbf{Z}$ is well-conditioned in a sense that we will make precise shortly. The ID has several advantages, as compared to, e.g., the QR or SVD factorizations:

- If $\mathbf{A}$ is sparse or non-negative, then $\mathbf{C}$ shares these properties.
- The ID require less memory to store than either the QR or the SVD.
- Finding the indices associated with the spanning columns is often helpful in *data interpretation.*
- In the context of numerical algorithms for discretizing PDEs and integral equations, the ID often preserves "the physics" of a problem in a way that the QR or SVD do not.

One shortcoming of the ID is that when $\mathbf{A}$ is not of precisely rank k, then the approximation error by the best possible rank-k ID can be substantially larger than the theoretically minimal error. (In fact, the ID and the column pivoted QR factorizations are closely related, and they attain *exactly* the same minimal error.)

4.1.2. Notation. While the ID is in many ways a very "natural" factorization, the notation required to fully describe it is slightly cumbersome. Consider again the factorization (4.1). We let J_{s} denote an index vector of length k that identifies the k columns that form $\mathbf{C}$ so that

$$\mathbf{C} = \mathbf{A}(:, J_{\mathrm{s}}).$$

The columns listed in J_{s} are called the "skeleton" columns. We put the remaining indices in a vector of "residual" indices J_{r} so that we obtain a disjoint partitioning of the index set

$$[1, 2, 3, \ldots, n] = J_{\mathrm{s}} \cup J_{\mathrm{r}}.$$

Next observe that the relation (4.1) implies that the matrix $\mathbf{Z}$ must contain as a submatrix the $k \times k$ identity matrix $\mathbf{I}_k$. Specifically, restricting (4.1) to the k columns identified in J_{s}, we see that it must necessarily be the case that

$$\mathbf{Z}(:, J_{\mathrm{s}}) = \mathbf{I}_k.$$

The remaining $n - k$ columns of $\mathbf{Z}$ consist of a $k \times (n - k)$ matrix

$$\mathbf{T} = \mathbf{Z}(:, J_{\mathrm{r}}).$$

We refer to the entries of $\mathbf{T}$ as *expansion coefficients* since each column of $\mathbf{T}$ contains the expansion coefficients used to build the corresponding column of $\mathbf{A}$ using the k skeleton columns as a basis. Finally, let $\mathbf{P}$ denote the $n \times n$ permutation matrix defined by $\mathbf{P} = \mathbf{I}_n(:, [J_{\mathrm{s}}, J_{\mathrm{r}}])$, so that

$$\mathbf{AP} = \mathbf{A}(:, [J_{\mathrm{s}}, J_{\mathrm{r}}]) = \big[\mathbf{C}, \mathbf{A}(:, J_{\mathrm{r}})\big].$$

Then

$$\mathbf{Z} = \big[\mathbf{I}_k \; \mathbf{T}\big]\mathbf{P}^*.$$

4.1.3. Three flavors of ID: The row, column, and double-sided ID. Sections 4.1.1 and 4.1.2 describe a factorization where we use a subset of the *columns* of $\mathbf{A}$ to space its *column space*. Naturally, this factorization has a sibling which uses the *rows* of $\mathbf{A}$ to span its *row space.* In other words $\mathbf{A}$ also admits the factorization

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{X}}\ \underset{k\times n}{\mathbf{R}}, \tag{4.2}$$

where $\mathbf{R}$ is a matrix consisting of k rows of $\mathbf{A}$, and where $\mathbf{X}$ is a matrix that contains the $k \times k$ identity matrix. We let $I_{\rm s}$ denote the index vector of length k that marks the "skeleton" rows so that $\mathbf{R} = \mathbf{A}(I_{\rm s}, :)$.

Finally, there exists a so called *double-sided ID* which takes the form

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{X}}\ \underset{k\times k}{\mathbf{A}_{\rm s}}\ \underset{k\times n}{\mathbf{Z}}, \tag{4.3}$$

where $\mathbf{X}$ and $\mathbf{Z}$ are the same matrices as those that appear in (4.1) and (4.2), and where $\mathbf{A}_{\rm s}$ is the $k \times k$ submatrix of $\mathbf{A}$ given by

$$\mathbf{A}_{\rm s} = \mathbf{A}(I_{\rm s}, J_{\rm s}).$$

In this chapter, we will for the most part discuss techniques for computing the column ID. With very minor modifications, these techniques can be used to compute the row ID and the double-sided ID too.

4.1.4. Storage requirements of the ID. Suppose that $\mathbf{A}$ is dense. Then we find that the amount of storage required to store the various factors are:

- *Row-ID:* Storing $\mathbf{R}$ and $\mathbf{X}$ require kn and $k(m-k)$ floats, respectively.
- *Column-ID:* Storing $\mathbf{C}$ and $\mathbf{Z}$ require km and $k(n-k)$ floats, respectively.
- *Double-ID:* Storing $\mathbf{A}_{\rm s}$, $\mathbf{X}$, and $\mathbf{Z}$ require k^2, $k(m-k)$, and $k(n-k)$ floats, respectively.

In every case, the storage requirements add up to $k(m+n-k)$. This is slightly better than the cost of storing the QR or SVD factorizations (which are $k(m+n-k/2)$ and $k(m+n)$, respectively).

For sparse matrices, the storage advantage of the ID becomes more dramatic. In this case, $\mathbf{C}$ and $\mathbf{R}$ will be sparse too, which can greatly cut down on the cost. For the double-sided ID, we can play some games to further reduce storage requirements. Suppose that $\mathbf{A}$ has exact rank k, and has a column ID

$$\mathbf{A} = \mathbf{C}\mathbf{Z}, \tag{4.4}$$

where $\mathbf{C} = \mathbf{A}(:, J_{\rm s})$. Suppose further that we find a good set $I_{\rm s}$ of spanning rows of $\mathbf{C}$, as we do in constructing the doublesided ID. Then restrict (4.4) to the rows in $I_{\rm s}$ to obtain

$$\mathbf{A}(I_{\rm s}, :) = \mathbf{C}(I_{\rm s}, :)\mathbf{Z}. \tag{4.5}$$

Now observe that $\mathbf{C}(I_{\rm s}, :) = \mathbf{A}(I_{\rm s}, J_{\rm s})$ to find that $\mathbf{Z}$ satisfies the relation

$$\mathbf{A} = \big(\mathbf{A}(I_{\rm s}, J_{\rm s})\big)^{-1}\,\mathbf{A}(I_{\rm s}, :). \tag{4.6}$$

In order to use the factorization (4.4), we see that there is no need to store $\mathbf{Z}$ explicitly: We only need $\mathbf{A}(I_{\rm s}, :)$ and $\big(\mathbf{A}(I_{\rm s}, J_{\rm s})\big)^{-1}$, which are both cheap to store (since $\mathbf{A}(I_{\rm s}, :)$ is sparse, and $\big(\mathbf{A}(I_{\rm s}, J_{\rm s})\big)^{-1}$ is tiny).

REMARK 4.1. In some environments, it is not necessary to store $\mathbf{C}$ and $\mathbf{R}$ explicitly. If the matrix entries are cheap to generate on the fly, then it is enough to just store the index vectors $I_{\rm s}$ and $J_{\rm s}$.

4.2. Existence of the ID

That any matrix of precise rank k admits factorization like (4.1), (4.2), and (4.3), is a direct consequence of the definition of the concept of rank.[1] A slightly deeper result is that there must always exist a *well-conditioned ID*. To be precise, we will prove that there exist IDs for which every entry of the "basis matrices" $\mathbf{X}$ and $\mathbf{Z}$ is bounded in modulus by one.

[1] Suppose $\mathbf{A}$ has rank k, so that its column space has dimension k. This means that the set of columns of $\mathbf{A}$ must contain a set of at least k independent vectors (otherwise the rank would be less than k). Once a set of k linearly independent columns has been fixed, we see that any of the remaining columns can be expressed as a linear combination of the chosen k (otherwise the rank would be higher than k).

THEOREM 9. *Let $\mathbf{A}$ be an $m \times n$ matrix of precisely rank k, where $k < \min(m, n)$. Then $\mathbf{A}$ admits a factorization*

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{C}}\ \underset{k\times n}{\mathbf{Z}}, \tag{4.7}$$

where $\mathbf{C} = \mathbf{A}(:, J_{\rm s})$ for some index vector $J_{\rm s}$ of length k contained in $[1, 2, \ldots, n]$, and where

$$\max_{i,j} |\mathbf{Z}(i, j)| \leq 1.$$

PROOF. Let us first consider a special case where $m = k$ so that $\mathbf{A}$ is of size $k \times n$. Then define the index vector $J_{\rm s}$ as the set of k indices that maximize the determinant of $\mathbf{A}(:, J_{\rm s})$. In other words:

$$J_{\rm s} = \text{argmax}\{|\det(\mathbf{A}(:, J))| : \ J \text{ is a subset of size } k \text{ of } [1, 2, \ldots, n]\}. \tag{4.8}$$

(The max may or may not unique, this does not matter.) Let $J_{\rm r}$ denote an index vector containing the "remaining" indices so that $J_{\rm s} \cup J_{\rm r} = [1, 2, \ldots, n]$ in a disjoint union. Let $\mathbf{P}$ denote the permutation matrix for which

$$\mathbf{AP} = [\mathbf{A}(:, J_{\rm s}) \quad \mathbf{A}(:, J_{\rm r})]. \tag{4.9}$$

Observe that $\mathbf{A}(:, J_{\rm s})$ must be non-singular, so we can rewrite (4.9) as

$$\mathbf{AP} = \mathbf{A}(:, J_{\rm s})\,[\mathbf{I}_k \quad \mathbf{A}(:, J_{\rm s})^{-1}\mathbf{A}(:, J_{\rm r})]. \tag{4.10}$$

Setting

$$\mathbf{T} := \mathbf{A}(:, J_{\rm s})^{-1}\mathbf{A}(:, J_{\rm r}), \tag{4.11}$$

we find that we can write (4.10) as

$$\mathbf{A} = \underbrace{\mathbf{A}(:, J_{\rm s})}_{=:\mathbf{C}}\ \underbrace{[\mathbf{I}_k \quad \mathbf{T}]\mathbf{P}^*}_{=:\mathbf{Z}}. \tag{4.12}$$

We will next prove that every element of $\mathbf{T}$ has modulus at most one. Let $\{j_i\}_{i=1}^n$ denote the chosen permutation of $[1, 2, \ldots, n]$ such that

$$[J_{\rm s}\ J_{\rm r}] = [j_1\ j_2\ j_3\ \cdots\ j_n].$$

We now find that the definition of $\mathbf{T}$ in (4.11) implies that

$$\underbrace{\big[\mathbf{c}_{j_1}\ \mathbf{c}_{j_2}\ \cdots\ \mathbf{c}_{j_k}\big]}_{=\mathbf{A}(:,J_{\rm s})}\mathbf{T} = \underbrace{\big[\mathbf{c}_{j_{k+1}}\ \mathbf{c}_{j_{k+2}}\ \cdots\ \mathbf{c}_{j_n}\big]}_{=\mathbf{A}(:,J_{\rm r})}.$$

Cramer's rule provides us with explicit solution formulas for each element of $\mathbf{T}$. For instance,

$$\mathbf{T}(1,1) = \frac{\det\big[\mathbf{c}_{j_{k+1}}\ \mathbf{c}_{j_2}\ \mathbf{c}_{j_3}\ \mathbf{c}_{j_4}\ \cdots\ \mathbf{c}_{j_k}\big]}{\det\big[\mathbf{c}_{j_1}\ \mathbf{c}_{j_2}\ \mathbf{c}_{j_3}\ \mathbf{c}_{j_4}\ \cdots\ \mathbf{c}_{j_k}\big]}, \qquad \mathbf{T}(1,2) = \frac{\det\big[\mathbf{c}_{j_{k+2}}\ \mathbf{c}_{j_2}\ \mathbf{c}_{j_3}\ \mathbf{c}_{j_4}\ \cdots\ \mathbf{c}_{j_k}\big]}{\det\big[\mathbf{c}_{j_1}\ \mathbf{c}_{j_2}\ \mathbf{c}_{j_3}\ \mathbf{c}_{j_4}\ \cdots\ \mathbf{c}_{j_k}\big]},$$

$$\mathbf{T}(2,1) = \frac{\det\big[\mathbf{c}_{j_1}\ \mathbf{c}_{j_{k+1}}\ \mathbf{c}_{j_3}\ \mathbf{c}_{j_4}\ \cdots\ \mathbf{c}_{j_k}\big]}{\det\big[\mathbf{c}_{j_1}\ \mathbf{c}_{j_2}\ \mathbf{c}_{j_3}\ \mathbf{c}_{j_4}\ \cdots\ \mathbf{c}_{j_k}\big]}, \qquad \mathbf{T}(2,2) = \frac{\det\big[\mathbf{c}_{j_2}\ \mathbf{c}_{j_{k+2}}\ \mathbf{c}_{j_3}\ \mathbf{c}_{j_4}\ \cdots\ \mathbf{c}_{j_k}\big]}{\det\big[\mathbf{c}_{j_1}\ \mathbf{c}_{j_2}\ \mathbf{c}_{j_3}\ \mathbf{c}_{j_4}\ \cdots\ \mathbf{c}_{j_k}\big]},$$

et cetera. Observe that in each formula, the denominator consists of the determinant formed by the k columns of $\mathbf{A}$ that maximize the modulus of the determinant. The numerator is obtained by swapping out one of the columns from the maximizing set by a different column. It follows directly from the definition of $J_{\rm s}$ in (4.8) that each such ratio must be bounded by one in modulus.

All that remains is to relax our temporary assumption that the matrix $\mathbf{A}$ must have precisely k rows. In the general case, observe that when $\mathbf{A}$ has exact rank k, it admits a factorization

$$\underset{m\times n}{\mathbf{A}} = \underset{m\times k}{\mathbf{E}}\ \underset{k\times n}{\mathbf{F}}, \tag{4.13}$$

for some matrices $\mathbf{E}$ and $\mathbf{F}$ (their properties do not matter beyond the fact that they both have precise rank k). We have shown that $\mathbf{F}$ admits an ID factorization with the required properties, that is,

$$\underset{k\times n}{\mathbf{F}} = \underset{k\times k}{\mathbf{F}(:, J_{\rm s})}\ \underset{k\times n}{\mathbf{Z}}, \tag{4.14}$$

where every entry of $\mathbf{Z}$ is bounded by one in modulus. Combining (4.13) and (4.14) we get

$$\mathbf{A} = \mathbf{E}\mathbf{F}(:, J_{\rm s})\mathbf{Z}. \tag{4.15}$$

Since $\mathbf{Z}(:, J_{\rm s}) = \mathbf{I}_k$, we can restrict (4.15) to the columns in $J_{\rm s}$ to see that

$$\mathbf{A}(:, J_{\rm s}) = \mathbf{E}\mathbf{F}(:, J_{\rm s}), \tag{4.16}$$

Combining (4.15) and (4.16) we finally get

$$\mathbf{A} = \mathbf{A}(:, J_{\rm s})\mathbf{Z}, \tag{4.17}$$

which is the factorization we sought. □

4.3. Deterministic techniques for computing the ID

4.3.1. Computing the ID from the column pivoted QR. The ID and the column pivoted QR factorization (CPQR) are closely related. To demonstrate this, let $\mathbf{A}$ be an $m \times n$ matrix. At first, let us assume that $\mathbf{A}$ has exact rank k, and that we have computed its CPQR so that

$$\underset{m \times n}{\mathbf{A}(:, J)} = \underset{m \times k}{\mathbf{Q}_1} \quad \underset{k \times n}{\mathbf{R}}. \tag{4.18}$$

(Our choice to endow the "Q" factor with a subscript is to keep notation consistent with the general case where $\mathbf{A}$ does not have exact rank k.) Now let us partition

$$J = \begin{bmatrix} J_{\rm s} & J_{\rm r} \end{bmatrix}, \tag{4.19}$$

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \end{bmatrix}, \tag{4.20}$$

in such a way that $J_{\rm s}$ holds the first k indices in J, and $\mathbf{R}_{11}$ is the leading $k \times k$ submatrix of $\mathbf{R}$. Observe that $J_{\rm s}$ points to the k columns on $\mathbf{A}$ that were chosen as "pivots." Set

$$\mathbf{C} = \mathbf{A}(:, J_{\rm s}). \tag{4.21}$$

Then (4.18) can be written

$$\begin{bmatrix} \mathbf{C} & \mathbf{A}(:, J_{\rm r}) \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1\mathbf{R}_{11} & \mathbf{Q}_1\mathbf{R}_{12} \end{bmatrix}. \tag{4.22}$$

Looking at the first k columns of (4.22) we see that it must be the case that

$$\mathbf{C} = \mathbf{Q}_1\mathbf{R}_{11}.$$

Next, observe that $\mathbf{R}_{11}$ is necessarily invertible (since $\mathbf{A}$ has rank k, and the pivoting procedure ensures that the columns in $\mathbf{C}$ are linearly independent). This means that (4.22) can be rewritten as

$$\mathbf{A} = \mathbf{C} \underbrace{\begin{bmatrix} \mathbf{I}_k & \mathbf{T} \end{bmatrix} \mathbf{P}^*}_{=:\mathbf{Z}}, \tag{4.23}$$

where $\mathbf{P}$ is the permutation matrix associated with J (as usual), and where

$$\mathbf{T} := \mathbf{R}_{11}^{-1}\mathbf{R}_{12}. \tag{4.24}$$

Equation (4.23) tells us that the CPQR factorization is in fact an ID in (very slight) disguise.

Next let us consider the case where $\mathbf{A}$ does not have *exact* rank k. Suppose that we have computed the full CPQR, which then takes then form

$$\mathbf{A}(:, J) = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{0} & \mathbf{R}_{22} \end{bmatrix}, \tag{4.25}$$

where $\mathbf{Q}_1$ holds the first k columns of $\mathbf{Q}$ and $\mathbf{R}_{11}$ is $k \times k$. Let us again partition the index vector as dictated by (4.19), define $\mathbf{C}$ via (4.21), and $\mathbf{T}$ via (4.24). Then we obtain the approximate rank-k factorization

$$\mathbf{A} \approx \mathbf{Q}_1 \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \end{bmatrix} \mathbf{P}^* = \mathbf{Q}_1\mathbf{R}_{11} \begin{bmatrix} \mathbf{I}_k & \mathbf{R}_{11}^{-1}\mathbf{R}_{12} \end{bmatrix} \mathbf{P}^* = \mathbf{C} \begin{bmatrix} \mathbf{I}_k & \mathbf{T} \end{bmatrix} \mathbf{P}^* = \mathbf{C}\,\mathbf{Z}.$$

We also have an exact expression for the approximation error from (4.25):

$$\mathbf{A} - \mathbf{C}\mathbf{Z} = \mathbf{Q}\mathbf{R}\mathbf{P}^* - \mathbf{Q}_1\begin{bmatrix}\mathbf{R}_{11} & \mathbf{R}_{12}\end{bmatrix}\mathbf{P}^* = \begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2\end{bmatrix} \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{0} & \mathbf{R}_{22} \end{bmatrix} \mathbf{P}^* - \mathbf{Q}_1\begin{bmatrix}\mathbf{R}_{11} & \mathbf{R}_{12}\end{bmatrix}\mathbf{P}^* = \begin{bmatrix}\mathbf{0} & \mathbf{Q}_2\mathbf{R}_{22}\end{bmatrix}\mathbf{P}^*.$$

In other words, the approximation error in the ID is $\|\begin{bmatrix}\mathbf{0} & \mathbf{Q}_2\mathbf{R}_{22}\end{bmatrix}\mathbf{P}^*\| = \|\mathbf{R}_{22}\|$, which is exactly the same as the approximation error in the truncated CPQR from which we obtained it.

REMARK 4.2. In practice, one typically only take k steps of the CPQR algorithm. At that point, we have a partial factorization

$$\mathbf{AP} \approx \mathbf{Q}_1\begin{bmatrix}\mathbf{R}_{11} & \mathbf{R}_{12}\end{bmatrix} + \begin{bmatrix}\mathbf{0} & \mathbf{B}\end{bmatrix}$$

where $\mathbf{B}$ contains the parts of the $n-k$ non-pivot columns that remain after projecting away from the span of $\mathbf{Q}_1$. Clearly $\mathbf{B}\tilde{\mathbf{P}} = \mathbf{Q}_2\mathbf{R}_{22}$ for some permutation matrix $\tilde{\mathbf{P}}$, but one does not need to carry out these steps in order to determine the approximation error since $\|\mathbf{B}\| = \|\mathbf{R}_{22}\|$.

4.3.2. Optimality of the QR procedure.

4.3.3. Computing the row ID and the double sided ID. First observe that the *row* ID is obtained by applying the procedure described in Section 4.3.1 to the *rows* of $\mathbf{A}$. In other words, the first step is to compute a partial CPQR of $\mathbf{A}^*$.

For the double sided ID, we start by computing a column ID,

$$\mathbf{A} = \mathbf{A}(:, J_{\mathrm{s}})\,\mathbf{Z} + \mathbf{E}, \tag{4.26}$$

for some "error matrix" $\mathbf{E}$. Then simply compute a row ID of the matrix $\mathbf{C} = \mathbf{A}(:, J_{\mathrm{s}})$ containing the k chosen columns,

$$\mathbf{A}(:, J_{\mathrm{s}}) = \mathbf{X}\,\mathbf{A}(I_{\mathrm{s}}, J_{\mathrm{s}}). \tag{4.27}$$

Observe that this second factorization is *exact* since the matrix $\mathbf{A}(:, J_{\mathrm{s}})$ has k columns, and therefore necessarily has rank k (provided that $\mathbf{A}$ has rank at least k). Combining (4.26) and (4.27), we obtain

$$\mathbf{A} = \mathbf{A}(:, J_{\mathrm{s}})\,\mathbf{Z} + \mathbf{E} = \mathbf{X}\mathbf{A}(I_{\mathrm{s}}, J_{\mathrm{s}})\,\mathbf{Z} + \mathbf{E}, \tag{4.28}$$

which is the double-sided ID we sought. Observe that the error term $\mathbf{E}$ is identical in (4.26) and (4.28); in other words, the error in the double-sided ID is identical to the error in the one-sided ID with which we start (which in turn is identical to the error in the original CPQR).

The three algorithms for computing the three flavors of the ID via the CPQR are summarized in Figure 4.1.

REMARK 4.3. The double-sided ID could also be computed by independently computing the row and column IDs of the original matrix $\mathbf{A}$, as follows:

$$[J_{\mathrm{s}},\ \mathbf{Z}] = \texttt{ID\_col}(\mathbf{A}, k) \qquad \text{and} \qquad [I_{\mathrm{s}},\ \mathbf{X}] = \texttt{ID\_row}(\mathbf{A}, k). \tag{4.29}$$

Then we automatically find that

$$\mathbf{A} \approx \mathbf{X}\,\mathbf{A}(I_{\mathrm{s}}, J_{\mathrm{s}})\,\mathbf{Z}.$$

While there are circumstances where this procedure may be beneficial, it is in general more expensive to execute. Moreover, the method in Figure 4.1 is guaranteed to produce a factorization whose approximation error is exactly the same as the error incurred by the first column pivoted ID. Computing the double sided ID via (4.25) involves two independent approximations that each incurs its own error; these two errors combine to a form an overall error that can easily exceed that of the method in Figure 4.1.

4.4. Computing interpolative decompositions via randomized sampling

The randomized range finder described in Section 2.3 is particularly effective when used in conjunction with the interpolative decomposition. To illustrate the process, consider for a moment the task of factorizing an $m \times n$ matrix $\mathbf{A}$ of *exact* rank k (we will discuss the case of approximate rank shortly). Using the randomized sampling technique, we would then draw an $n \times k$ random matrix $\mathbf{G}$, form a sample matrix $\mathbf{Y} = \mathbf{AG}$, orthonormalize its columns to form the matrix $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$, and finally form the rank-k factorization

$$\mathbf{A} \approx \mathbf{Q}\left(\mathbf{Q}^*\mathbf{A}\right).$$

Compute a column ID so that $\mathbf{A} \approx \mathbf{A}(:, J_s)\,\mathbf{Z}$.
function $[J_s,\ \mathbf{Z}] = \texttt{ID\_col}(\mathbf{A}, k)$
  $[\mathbf{Q}, \mathbf{R}, J] = \texttt{qr}(\mathbf{A}, 0)$;
  $\mathbf{T} = (\mathbf{R}(1:k, 1:k))^{-1}\mathbf{R}(1:k, (k+1):n)$;
  $\mathbf{Z} = \texttt{zeros}(k, n)$
  $\mathbf{Z}(:, J) = [\mathbf{I}_k\ \mathbf{T}]$;
  $J_s = J(1:k)$;

Compute a row ID so that $\mathbf{A} \approx \mathbf{X}\,\mathbf{A}(I_s, :)$.
function $[I_s,\ \mathbf{X}] = \texttt{ID\_row}(\mathbf{A}, k)$
  $[\mathbf{Q}, \mathbf{R}\,J] = \texttt{qr}(\mathbf{A}^*, 0)$;
  $\mathbf{T} = (\mathbf{R}(1:k, 1:k))^{-1}\mathbf{R}(1:k, (k+1):n)$;
  $\mathbf{X} = \texttt{zeros}(m, k)$
  $\mathbf{X}(J, :) = [\mathbf{I}_k\ \mathbf{T}]^*$;
  $I_s = J(1:k)$;

Compute a double-sided ID so that $\mathbf{A} \approx \mathbf{X}\,\mathbf{A}(I_s, J_s)\,\mathbf{Z}$.
function $[I_s, J_s, \mathbf{X}, \mathbf{Z}] = \texttt{ID\_double}(\mathbf{A}, k)$
  $[J_s,\ \mathbf{Z}] = \texttt{ID\_col}(\mathbf{A}, k)$;
  $[I_s,\ \mathbf{X}] = \texttt{ID\_row}(\mathbf{A}(:, J_s), k)$;

FIGURE 4.1. Deterministic algorithms for computing the column, row, and double-sided ID via the column pivoted QR factorization. The input is in every case an $m \times n$ matrix $\mathbf{A}$ and a target rank k. Since the algorithms are based on the CPQR, it is elementary to modify them to the situation where a tolerance rather than a rank is given. (Recall that the errors resulting from these ID algorithms are *identical* to the error in the first CPQR factorization executed.)

This process requires the evaluation of the matrix-matrix product $\mathbf{Q}^*\mathbf{A}$. It turns out that by using the ID instead, we can skip this step. Instead of applying Gram-Schmidt to the *columns* of $\mathbf{Y}$, we apply Gram-Schmidt to the *rows* of $\mathbf{Y}$ to find a good spanning set of rows. To be precise, we execute the command *rows*,

$$[\mathbf{X}, I_s] = \texttt{ID\_row}(\mathbf{Y}, k).$$

Then

$$\mathbf{Y} = \mathbf{X}\mathbf{Y}(I_s, :). \tag{4.30}$$

Now, automatically (!), the couple $\{\mathbf{X}, I_s\}$ also forms an ID for $\mathbf{A}$,

$$\mathbf{A} = \mathbf{X}\mathbf{A}(I_s, :). \tag{4.31}$$

(See Remark 4.4 for details.) The beauty here is that once we have the sample matrix $\mathbf{Y}$, we do not need to revisit all of $\mathbf{A}$ to construct a rank-k factorization — all we need to do is to extract the k rows of $\mathbf{A}$ indicated by the index vector I_s.

By combining the idea of using an SRFT for "Stage A" (at complexity $O(mn\log(k))$) with an ID for "Stage B" (at complexity $O((m+n)k^2)$), we get the accelerated algorithm for constructing an ID described in Figure 4.3. With a slight modification to Stage B, we get the $O(mn\log(k))$ algorithm for computing an SVD described in Figure 4.4. More details (including an error analysis for the case when $\mathbf{A}$ has only approximate rank k) can be found in [**11**, Sec. 5.2].

REMARK 4.4. It is perhaps not immediately clear why the ID $\{\mathbf{X}, I_s\}$ computed for $\mathbf{Y}$ should automatically also be an ID for $\mathbf{A}$. To see why this should be, first observe that since the columns of $\mathbf{Y}$ form a basis for the columns of $\mathbf{A}$, there must exist a $k \times n$ matrix $\mathbf{F}$ such that

$$\mathbf{A} = \mathbf{Y}\mathbf{F}. \tag{4.32}$$

Now insert the relation (4.30) into (4.32):

$$\mathbf{A} = \mathbf{X}\mathbf{Y}(I_s, :)\mathbf{F}. \tag{4.33}$$

ALGORITHM: RANDOMIZED ID

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, an over-sampling parameter p (say $p = 10$), and a small integer q denoting the number of power iterations taken.

Outputs: An $m \times k$ interpolation matrix $\mathbf{X}$ and an index vector $I_{\rm s} \in \mathbb{N}^k$ such that $\mathbf{A} \approx \mathbf{X}\mathbf{A}(I_{\rm s}, :)$.

(1) $\mathbf{G} = \texttt{randn}(n, k+p)$;
(2) $\mathbf{Y} = \mathbf{A}\mathbf{G}$;
(3) **for** $j = 1 : q$
(4) $\quad \mathbf{Z} = \mathbf{A}^*\mathbf{Y}$;
(5) $\quad \mathbf{Y} = \mathbf{A}\mathbf{Z}$;
(6) **end for**
(7) Form an ID of the $n \times (k+p)$ sample matrix: $[\mathbf{X}, I_{\rm s}] = \texttt{ID\_row}(\mathbf{Y}, k)$.

FIGURE 4.2. An $O(mnk)$ algorithm for computing an interpolative decomposition of $\mathbf{A}$ via randomized sampling. For $q = 0$, the scheme is fast (but not quite as fast as the scheme in Figure 4.3) and accurate for matrices whose singular values decay rapidly. For matrices whose singular values decay slowly, one should pick a larger q (say $q = 1$ or 2) to improve accuracy at the cost of longer execution time. If accuracy better than $\epsilon_{\rm mach}^{1/(2q+1)}$ is desired, then the scheme should be modified to incorporate orthonormalization as described in Remark 2.5.

ALGORITHM: FAST RANDOMIZED ID

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p (say $p = k$).

Outputs: An $m \times k$ interpolation matrix $\mathbf{X}$ and an index vector $I_{\rm s} \in \mathbb{N}^k$ such that $\mathbf{A} \approx \mathbf{X}\mathbf{A}(I_{\rm s}, :)$.

Stage A:
(1) Form an $n \times (k+p)$ SRFT $\mathbf{\Omega}$.
(2) Form the sample matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{\Omega}$.

Stage B:
(3) Form an ID of the $n \times (k+p)$ sample matrix: $[\mathbf{X}, I_{\rm s}] = \texttt{ID\_row}(\mathbf{Y}, k)$.

FIGURE 4.3. An $O(mn\log(k))$ algorithm for computing an interpolative decomposition of $\mathbf{A}$.

Restrict (4.33) to the rows in $I_{\rm s}$, and exploit that $\mathbf{X}(I_{\rm s}, :) = \mathbf{I}_k$ to find

$$\mathbf{A}(I_{\rm s}, :) = \mathbf{X}(I_{\rm s}, :)\mathbf{Y}(I_{\rm s}, :)\mathbf{F} = \mathbf{Y}(I_{\rm s}, :)\mathbf{F}. \tag{4.34}$$

Finally, insert (4.34) in (4.33) to attain (4.31).

4.5. The CUR Decomposition

4.5.1. Basic definitions. A rank k CUR factorization of a matrix $\mathbf{A} \in \mathbb{C}^{m\times n}$ is given by

$$\underset{m\times n}{\mathbf{A}} \quad \approx \quad \underset{m\times k}{\mathbf{C}} \quad \underset{k\times k}{\mathbf{U}} \quad \underset{k\times n}{\mathbf{R}},$$

where $\mathbf{C} = \mathbf{A}(:, J_{\rm s})$ consists of k columns of $\mathbf{A}$, and $\mathbf{R} = \mathbf{A}(I_{\rm s}, :)$ consists of k rows of $\mathbf{A}$. The decomposition is typically obtained in three steps [**16**]: (1) Some scheme is used to assign a weight or the so called leverage score (of importance) to each column and row in the matrix. This is typically done either using the ℓ_2 norms of the columns and rows or by using the leading singular vectors of $\mathbf{A}$ [**5**]. (2) The matrices $\mathbf{C}$ and $\mathbf{R}$ are constructed via

ALGORITHM: FAST RANDOMIZED SVD

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, and an over-sampling parameter p (say $p = k$).

Outputs: Matrices $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}$ in an approximate rank-$(k+p)$ SVD of $\mathbf{A}$. (I.e. $\mathbf{U}$ and $\mathbf{V}$ are orthonormal and $\mathbf{\Sigma}$ is diagonal.)

Stage A:

(1) Form an $n \times (k+p)$ SRFT $\mathbf{\Omega}$.
(2) Form the sample matrix $\mathbf{Y} = \mathbf{B}\,\mathbf{\Omega}$.

Stage B:

(3) Form an ID of the sample matrix $\mathbf{Y}$: $[\mathbf{X}, I_{\rm s}] = \texttt{ID\_row}(\mathbf{Y}, k)$.
(4) Compute the QR decomposition of the interpolation matrix $[\mathbf{Q}, \mathbf{R}] = \texttt{qr}(\mathbf{X})$.
(5) Extract $k+p$ rows of $\mathbf{A}$: $\mathbf{A}^{\rm rows} = \mathbf{A}(I_{\rm s}, :)$.
(6) Multiply $\mathbf{R}$ and $\mathbf{A}^{\rm rows}$ to form the $(k+p) \times n$ matrix $\mathbf{F} = \mathbf{R}\mathbf{A}^{\rm rows}$.
(7) Decompose the matrix $\mathbf{F}$ in a singular value decomposition $[\hat{\mathbf{U}},\ \mathbf{\Sigma},\ \mathbf{V}] = \texttt{svd}(\mathbf{F})$.
(8) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

FIGURE 4.4. An $O(mn\log(k))$ algorithm for computing a partial SVD. If an SVD of exactly rank k is desired, then truncate the factors computed in Step (7).

a randomized sampling procedure, using the leverage scores to assign a sampling probability to each column and row. (3) The $\mathbf{U}$ matrix is computed via:

$$\mathbf{U} \approx \mathbf{C}^{\dagger}\mathbf{A}\mathbf{R}^{\dagger}, \tag{4.35}$$

with $\mathbf{C}^{\dagger}$ and $\mathbf{R}^{\dagger}$ being the pseudoinverses of $\mathbf{C}$ and $\mathbf{R}$.

Many techniques for computing CUR factorizations have been proposed. In particular, we mention the recent work of Sorensen and Embree [**17**] on the DEIM-CUR method. A number of standard CUR algorithms is implemented in the software package rCUR [**2**]. The methods in the rCUR package utilize eigenvectors to assign weights to columns and rows of $\mathbf{A}$. Computing the eigenvectors exactly amounts to doing the SVD which is very expensive. However, instead of the full SVD, when a CUR of rank k is required, we can utilize instead the randomized SVD algorithm [**11**] to compute an approximate SVD of rank k at substantially lower cost.

4.5.2. Conditioning of CUR. For matrices whose singular value experience substantial decay, the accuracy of the CUR factorization can deteriorate due to effects of ill-conditioning. To simplify slightly, one would normally expect the leading k singular values of $\mathbf{C}$ and $\mathbf{R}$ to be roughly similar in size to the leading k singular values of $\mathbf{A}$. Since low-rank factorizations are most useful when applied to matrices whose singular values decay reasonably rapidly, we would *typically* expect $\mathbf{C}$ and $\mathbf{R}$ to be highly ill-conditioned, with condition numbers roughly on the order of $\sigma_1(\mathbf{A})/\sigma_k(\mathbf{A})$. Hence, in the typical case, evaluation of the formula (4.35) can be expected to result in substantial loss of accuracy due to accumulation of round-off errors. Observe that the ID does not suffer from this problem; in (4.3), the matrix $\mathbf{A}_{\rm skel}$ tends to be ill-conditioned, but it does not need to be inverted. (The matrices $\mathbf{X}$ and $\mathbf{Z}$ are well-conditioned.)

Example: *Consider an $m \times n$ matrix $\mathbf{A}$ of rank 2, whose doublesided ID is given by*

$$\mathbf{A} = \mathbf{X}\mathbf{A}_{\rm s}\mathbf{Y}, \quad \text{where} \quad \mathbf{X} = \begin{bmatrix} \mathbf{I} \\ \mathbf{S} \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} \mathbf{I} & \mathbf{T} \end{bmatrix}, \quad \mathbf{A}_{\rm s} = \begin{bmatrix} 1 & 0 \\ 0 & \beta \end{bmatrix}, \tag{4.36}$$

for some small number β. Then the CUR factorization of $\mathbf{A}$ takes the form

$$\mathbf{A} = \mathbf{A}(:, [1,2]) \begin{bmatrix} 1 & 0 \\ 0 & 1/\beta \end{bmatrix} \mathbf{A}([1,2], :). \tag{4.37}$$

Observe that while the factorization (4.36) is perfectly well-conditioned, the factorization (4.37) is numerically problematic whenever β is small.

4.5.3. Storage requirements of CUR and ID. For a dense matrix, all matrix decompositions that we have discussed require roughly $k(m+n)$ floats. When k is close to m and n in size, then terms of order k^2 matter too, and we find that the precise requirements are (cf. Section 4.1.4):

Number of floats required to store a dense $m \times n$ matrix of rank k:				
	CUR	SVD	QR	ID (all three flavors)
Storage requirement:	$k(m+n+k)$	$k(m+n)$	$k(m+n-k/2)$	$k(m+n-k)$

In other words, the CUR is the most expensive, and the ID is the least expensive.

When $\mathbf{A}$ is sparse, then the ID and the CUR decomposition gain a large advantage since the factors $\mathbf{C}$ and $\mathbf{R}$ inherit sparsity from the original matrix. At first glance, the ID might seem more expensive in this requirement since it appears to require the storage of the dense matrices of expansion coefficients $\mathbf{X}$, $\mathbf{Z}$, or both. But, as we showed in Section 4.1.4, this problem can easily be fixed, so that the CUR and the ID have the same storage requirements:

Storage requirements of CUR and ID: One dense $k \times k$ matrix, and k rows and k columns of $\mathbf{A}$.

4.6. Deterministic techniques for computing the CUR

In this section, we describe a deterministic technique we label the CUR-ID for computing the CUR factorization from the double-sided ID. We start with a heuristic derivation in Section 4.6.1, and then proceed to prove that the error incurred is modest in Section 4.6.2.

4.6.1. Converting a double-sided ID to a CUR decomposition. The CUR-ID algorithm is based on the double-sided ID and as a starting point, we assume the factorization (4.3) has been computed using the procedures described in Section 4.3.3. In other words, we assume that the index vectors $I_{\rm s}$ and $J_{\rm s}$, and the basis matrices $\mathbf{X}$ and $\mathbf{Z}$, are all available. We then define

$$\mathbf{C} = \mathbf{A}(:, J_{\rm s}) \quad \text{and} \quad \mathbf{R} = \mathbf{A}(I_{\rm s}, :). \tag{4.38}$$

Consequently, $\mathbf{C}$ and $\mathbf{R}$ are respectively subsets of columns and of rows of $\mathbf{A}$, with $J_{\rm s}$ and $I_{\rm s}$ determined by the column pivoted QR factorizations. Next we construct a $k \times k$ matrix $\mathbf{U}$ such that $\mathbf{A} \approx \mathbf{CUR}$. We know that

$$\mathbf{A} \approx \mathbf{C}\,\mathbf{Z}, \tag{4.39}$$

and we seek a factor $\mathbf{U}$ such that

$$\mathbf{A} \approx \mathbf{C}\,\mathbf{U}\,\mathbf{R}. \tag{4.40}$$

By inspecting (4.39) and (4.40), we find that we achieve our objective if we determine a matrix $\mathbf{U}$ such that

$$\underset{k\times k}{\mathbf{U}} \quad \underset{k\times n}{\mathbf{R}} \quad = \quad \underset{k\times n}{\mathbf{Z}}. \tag{4.41}$$

Unfortunately, (4.41) is an over-determined system, but at least intuitively, it seems plausible that it should have a fairly accurate solution, given that the rows of $\mathbf{R}$ and the rows of $\mathbf{Z}$ should, by construction, span roughly the same space (namely, the space spanned by the k leading right singular vectors of $\mathbf{A}$). Solving (4.41) in the least-square sense, we arrive at our definition of $\mathbf{U}$:

$$\mathbf{U} := \mathbf{Z}\mathbf{R}^{\dagger}. \tag{4.42}$$

The resulting algorithm is summarized in Figure 4.5.

REMARK 4.5. Observe that for purposes of data interpretation, the double-sided ID is just as useful as the CUR, and does not suffer from problems of ill-conditioning, cf. Section 4.5.2.

REMARK 4.6. The techniques for computing the double sided ID that we have described are based on two single-sided IDs. Observe that these can be executed in a few different ways. For instance, either of the following two methods work well:

ALGORITHM: CUR FROM COLUMN PIVOTED QR

Inputs: An $m \times n$ matrix $\mathbf{A}$, and a target rank k.

Outputs: A $k \times k$ matrix $\mathbf{U}$, and index vector $I_{\rm s}$ and $J_{\rm s}$ such that $\mathbf{A} \approx \mathbf{A}(:, J_{\rm s})\mathbf{U}\mathbf{A}(I_{\rm s}, :)$.

(1) $[I_{\rm s}, J_{\rm s}, \mathbf{X}, \mathbf{Z}] = \texttt{ID\_double}(\mathbf{A}, k)$.
(2) $\mathbf{U} = \mathbf{Z}(\mathbf{A}(I_{\rm s}, :))^{\dagger}$.

FIGURE 4.5. An algorithm for computing a CUR factorization via two column pivoted QR factorizations. *Very important:* When the double-sided ID is constructed, it is in this context crucial that the second ID is performed only the subset of columns picked by the first ID, not on the full matrix, see Remark 4.6.

Method 1:
$[J_{\rm s},\ \mathbf{Z}] = \texttt{ID\_col}(\mathbf{A}, k)$;
$[I_{\rm s},\ \mathbf{X}] = \texttt{ID\_row}(\mathbf{A}(:, J_{\rm s}), k)$;

Method 2:
$[J_{\rm s},\ \mathbf{Z}] = \texttt{ID\_col}(\mathbf{A}, k)$;
$[I_{\rm s},\ \mathbf{X}] = \texttt{ID\_row}(\mathbf{A}, k)$;

The first step of both methods is the same: perform CPQR on the columns of $\mathbf{A}$ to find the spanning columns. In the second step, Method 1 performs a row ID using only the columns chosen in the first step, while Method 2 performs a row ID on all of $\mathbf{A}$. For purposes of computing the double-sided ID, we generally recommend the use of Method 1 since it is computationally more efficient, but either method works fine. When we compute the CUR decomposition, however, it is crucial that Method 1 is used. The reason is that Method 1 inherently seeks to construct a matrix $\mathbf{A}(I_{\rm s}, J_{\rm s})$ that is as well conditioned as possible. When using Method 2, one can easily end up with a matrix $\mathbf{A}(I_{\rm s}, J_{\rm s})$ that is far worse conditioned than is necessary.

4.6.2. Error analysis of the CUR-ID algorithm. The construction of $\mathbf{C}$, $\mathbf{U}$, and $\mathbf{R}$ in Section 4.6.1 was based on heuristics. We next demonstrate that the approximation error is comparable to the error resulting from the original QR-factorization. First, let us define $\mathbf{E}$ and $\tilde{\mathbf{E}}$ as the errors in the column and row IDs of $\mathbf{A}$, respectively,

$$\mathbf{A} = \mathbf{C}\,\mathbf{Z} + \mathbf{E}, \tag{4.43}$$

$$\mathbf{A} = \mathbf{X}\,\mathbf{R} + \tilde{\mathbf{E}}. \tag{4.44}$$

Recall that $\mathbf{E}$ is a quantity we can control by continuing the original QR factorization of $\mathbf{A}$ until $\|\mathbf{E}\|$ is smaller than some given threshold. We will prove that $\|\tilde{\mathbf{E}}\|$ cannot be all that much larger than $\|\mathbf{E}\|$, which leads us to the following error bound for the algorithm in Figure 4.5:

THEOREM 10. *Let $\mathbf{A}$ be an $m \times n$ matrix. Suppose that we have computed a column ID of rank k for $\mathbf{A}$ with error term $\mathbf{E}$, as in (4.43). Suppose further that we next compute the row ID $\{I_{\rm s}, \mathbf{X}\}$ of $\mathbf{C}$, and use this row ID to approximate $\mathbf{A}$, resulting in an approximation error $\tilde{\mathbf{E}}$, as in (4.44). Then*

$$\|\tilde{\mathbf{E}}\| \leq (1 + \|\mathbf{T}\|)\,\|\mathbf{E}\|, \tag{4.45}$$

where $\mathbf{T}$ is the $k \times (m-k)$ matrix of expansion coefficients obtained while computing the row ID of $\mathbf{C}$. Additionally, it holds that

$$\|\mathbf{A} - \mathbf{C}\mathbf{U}\mathbf{R}\| \leq (2 + \|\mathbf{T}\|)\,\|\mathbf{E}\|. \tag{4.46}$$

The error bound in Theorem 10 involves the term $\|\mathbf{T}\|$. We can provide a simplistic bound for this quantity as follows: Recall that the matrix $\mathbf{T}$ contains the expansion coefficients in the interpolative decomposition of $\mathbf{C}$. These can be guaranteed **[8, 12]** to all be bounded by $1 + \nu$ in magnitude for any positive number ν. The cost increases as $\nu \to 0$, but for, e.g., $\nu = 1$, the cost is very modest. Consequently, we find that for either the spectral or the Frobenius norm, we can easily guarantee $\|\mathbf{T}\| \leq (1 + \nu)\sqrt{k(n-k)}$, with practical norm often far smaller.

We will prove Theorem 10 in two steps. Lemma 11 asserts that when (4.43) and (4.44) hold, then $\|\mathbf{A} - \mathbf{C}\mathbf{U}\mathbf{R}\| \leq \mathbf{E} + \tilde{\mathbf{E}}$. Then Lemma 12 provides the information we need to bound $\|\tilde{\mathbf{E}}\|$.

LEMMA 11. *Let $\mathbf{A}$ be an $m \times n$ matrix that satisfies the approximate factorizations (4.43) and (4.44). Suppose further that $\mathbf{R}$ is full rank, and that the $k \times k$ matrix $\mathbf{U}$ is defined by (4.42). Then*

$$\|\mathbf{A} - \mathbf{CUR}\| \leq \|\mathbf{E}\| + \|\tilde{\mathbf{E}}\|. \tag{4.47}$$

PROOF. Using first (4.42) and then (4.43), we find

$$\mathbf{A} - \mathbf{CUR} = \mathbf{A} - \mathbf{CZR}^{\dagger}\mathbf{R} = \mathbf{A} - (\mathbf{A} - \mathbf{E})\mathbf{R}^{\dagger}\mathbf{R} = \big(\mathbf{A} - \mathbf{AR}^{\dagger}\mathbf{R}\big) + \mathbf{ER}^{\dagger}\mathbf{R}. \tag{4.48}$$

To bound the term $\mathbf{A} - \mathbf{AR}^{\dagger}\mathbf{R}$ we use (4.44) and the fact that $\mathbf{RR}^{\dagger}\mathbf{R} = \mathbf{R}$ to achieve

$$\mathbf{A} - \mathbf{AR}^{\dagger}\mathbf{R} = \mathbf{A} - (\mathbf{WR} + \tilde{\mathbf{E}})\mathbf{R}^{\dagger}\mathbf{R} = \mathbf{A} - \mathbf{WR} - \tilde{\mathbf{E}}\mathbf{R}^{\dagger}\mathbf{R} = \tilde{\mathbf{E}} - \tilde{\mathbf{E}}\mathbf{R}^{\dagger}\mathbf{R} = \tilde{\mathbf{E}}(\mathbf{I} - \mathbf{R}^{\dagger}\mathbf{R}). \tag{4.49}$$

Inserting (4.49) into (4.48) and taking the norms of the result, we get

$$\|\mathbf{A} - \mathbf{CUR}\| = \|\tilde{\mathbf{E}}(\mathbf{I} - \mathbf{R}^{\dagger}\mathbf{R}) + \mathbf{ER}^{\dagger}\mathbf{R}\| \leq \|\tilde{\mathbf{E}}(\mathbf{I} - \mathbf{R}^{\dagger}\mathbf{R})\| + \|\mathbf{ER}^{\dagger}\mathbf{R}\| \leq \|\tilde{\mathbf{E}}\| + \|\mathbf{E}\|,$$

where in the last step we used that $\mathbf{RR}^{\dagger}$ and $\mathbf{I} - \mathbf{RR}^{\dagger}$ are both orthonormal projections. □

LEMMA 12. *Let $\mathbf{A}$ be an $m \times n$ matrix that admits the factorization (4.43), with error term $\mathbf{E}$. Suppose further that $I = [I_{\mathrm{s}}, I_{\mathrm{r}}]$ and $\mathbf{T}$ form the output of the row ID of the matrix $\mathbf{C}$, so that*

$$\mathbf{C} = \mathbf{XC}(I_{\mathrm{s}}, :), \qquad \textit{where} \qquad \mathbf{X} = \mathbf{P}\begin{bmatrix} \mathbf{I} \\ \mathbf{T}^* \end{bmatrix}, \tag{4.50}$$

and where $\mathbf{P}$ is the permutation matrix for which $\mathbf{PA}(I, :) = \mathbf{A}$. Now define the matrix $\mathbf{R}$ via

$$\mathbf{R} = \mathbf{A}(I_{\mathrm{s}}, :). \tag{4.51}$$

Observe that $\mathbf{R}$ consists of the k rows of $\mathbf{A}$ selected in the skeletonization of $\mathbf{C}$. Finally, set

$$\mathbf{F} = \big[-\mathbf{T}^* \;\; \mathbf{I}\big]\mathbf{P}^*. \tag{4.52}$$

Then the product $\mathbf{XR}$ approximates $\mathbf{A}$, with a residual error

$$\tilde{\mathbf{E}} = \mathbf{A} - \mathbf{XR} = \mathbf{P}\begin{bmatrix} \mathbf{0} \\ \mathbf{FE} \end{bmatrix}. \tag{4.53}$$

PROOF. From the definitions of $\mathbf{X}$ in (4.50) and $\mathbf{R}$ in (4.51) we find

$$\begin{aligned} \mathbf{A} - \mathbf{XR} = \mathbf{PA}(I, :) - \mathbf{XR} &= \mathbf{P}\begin{bmatrix} \mathbf{A}(I_{\mathrm{s}}, :) \\ \mathbf{A}(I_{\mathrm{r}}, :) \end{bmatrix} - \mathbf{P}\begin{bmatrix} \mathbf{I} \\ \mathbf{T}^* \end{bmatrix}\mathbf{A}(I_{\mathrm{s}}, :) \\ &= \mathbf{P}\begin{bmatrix} \mathbf{0} \\ \mathbf{A}(I_{\mathrm{r}}, :) - \mathbf{T}^*\mathbf{A}(I_{\mathrm{s}}, :) \end{bmatrix} = \mathbf{P}\begin{bmatrix} \mathbf{0} \\ \mathbf{FA} \end{bmatrix}. \end{aligned} \tag{4.54}$$

To bound the term $\mathbf{FA}$ in (4.54), we invoke (4.43) to obtain

$$\mathbf{FA} = \mathbf{FCV}^* + \mathbf{FE} = \{\text{Insert (4.50)}\} = \mathbf{FXC}(I_{\mathrm{s}}, :)\mathbf{V}^* + \mathbf{FE} = \mathbf{FE}, \tag{4.55}$$

since $\mathbf{FX} = \mathbf{0}$ due to (4.50) and (4.52). Finally, insert (4.55) into (4.54) to obtain (4.53). □

PROOF OF THEOREM 10. Equation (4.53) allows us to bound the norm of the error $\tilde{\mathbf{E}}$ in (4.44). Simply observe that the definition of $\mathbf{F}$ in (4.52) implies that for any matrix $\mathbf{B}$ we have:

$$\mathbf{FB} = \big[-\mathbf{T}^* \;\; \mathbf{I}\big]\mathbf{P}^*\mathbf{B} = \big[-\mathbf{T}^* \;\; \mathbf{I}\big]\begin{bmatrix} \mathbf{B}(I_{\mathrm{s}}, :) \\ \mathbf{B}(I_{\mathrm{r}}, :) \end{bmatrix} = -\mathbf{T}^*\mathbf{B}(I_{\mathrm{s}}, :) + \mathbf{B}(I_{\mathrm{r}}, :),$$

so that:

$$\|\mathbf{FB}\| = \|\mathbf{B}(I_{\mathrm{r}}, :) - \mathbf{T}^*\mathbf{B}(I_{\mathrm{s}}, :)\| \leq \|\mathbf{B}(I_{\mathrm{r}}, :)\| + \|\mathbf{T}\|\,\|\mathbf{B}(I_{\mathrm{s}}, :)\| \leq (1 + \|\mathbf{T}\|)\,\|\mathbf{B}\|. \tag{4.56}$$

We can now prove (4.45), by applying the bound (4.56) to (4.53), with $\mathbf{E}$ in place of $\mathbf{B}$:

$$\|\tilde{\mathbf{E}}\| = \left\|\mathbf{P}\begin{bmatrix} \mathbf{0} \\ \mathbf{FE} \end{bmatrix}\right\| \leq \left\|\begin{bmatrix} \mathbf{0} \\ \mathbf{FE} \end{bmatrix}\right\| = \|\mathbf{FE}\| \leq (1 + \|\mathbf{T}\|)\,\|\mathbf{E}\|.$$

To establish (4.46), we use (4.47) and (4.45):

$$\|\mathbf{A} - \mathbf{CUR}\| \leq \|\mathbf{E}\| + \|\tilde{\mathbf{E}}\| \leq (2 + \|\mathbf{T}\|)\,\|\mathbf{E}\|.$$

ALGORITHM: RANDOMIZED CUR

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank k, an over-sampling parameter p (say $p = 10$), and a small integer q denoting the number of power iterations taken.

Outputs: An $k \times k$ matrix $\mathbf{U}$ and index vectors $I_{\rm s}$ and $J_{\rm s}$ of length k such that $\mathbf{A} \approx \mathbf{A}(:, J_{\rm s})\, \mathbf{U}\, \mathbf{A}(I_{\rm s}, :)$.

(1) $\mathbf{G} = \texttt{randn}(k + p, m)$;
(2) $\mathbf{Y} = \mathbf{G}\mathbf{A}$;
(3) **for** $j = 1 : q$
(4) $\quad \mathbf{Z} = \mathbf{Y}\mathbf{A}^*$;
(5) $\quad \mathbf{Y} = \mathbf{Z}\mathbf{A}$;
(6) **end for**
(7) Form an ID of the $(k + p) \times n$ sample matrix: $[\mathbf{Z}, J_{\rm s}] = \texttt{ID\_col}(\mathbf{Y}, k)$.
(8) Form an ID of the $m \times k$ matrix of chosen columns: $[\sim, I_{\rm s}] = \texttt{ID\_row}(\mathbf{A}(:, J_{\rm s}), k)$.
(9) Determine the linking matrix $\mathbf{U}$ by solving the least squares equation $\mathbf{U}\mathbf{A}(I_{\rm s}, :) = \mathbf{Z}$.

FIGURE 4.6. A randomized algorithm for computing a CUR decomposition of $\mathbf{A}$ via randomized sampling. For $q = 0$, the scheme is fast and accurate for matrices whose singular values decay rapidly. For matrices whose singular values decay slowly, one should pick a larger q (say $q = 1$ or 2) to improve accuracy at the cost of longer execution time. If accuracy better than $\epsilon_{\rm mach}^{1/(2q+1)}$ is desired, then the scheme should be modified to incorporate orthonormalization as described in Remark 2.5.

□

4.7. Randomized techniques for computing the CUR

In Section 4.6 we demonstrated that the CUR decomposition of a matrix can be readily computed from a double-sided ID, as long as care is taken to ensure that the matrix $\mathbf{A}(I_{\rm s}, J_{\rm s})$ at the center of the double-sided ID is as well conditioned as possible. If we combine this scheme with the randomized method for computing the double-sided ID described in Section 4.4, we then obtain a randomized algorithm for computing the CUR decomposition. The resulting method is shown in Figure 4.6.

Bibliography

[1] L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

[2] András Bodor, István Csabai, Michael Mahoney, and Norbert Solymosi. rCUR: an R package for CUR matrix decomposition. *BMC Bioinformatics*, 13(1), 2012.

[3] Jack J Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain S Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software (TOMS)*, 16(1):1–17, 1990.

[4] P. Drineas and M. W. Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *J. Mach. Learn. Res.*, 6:2153–2175, 2005.

[5] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. Relative-error CUR matrix decompositions. *SIAM J. Matrix Anal. Appl.*, 30(2):844–881, 2008.

[6] A. Frieze, R. Kannan, and S. Vempala. Fast Monte Carlo algorithms for finding low-rank approximations. *J. ACM*, 51(6):1025–1041, 2004. (electronic).

[7] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[8] Ming Gu and Stanley C. Eisenstat. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM J. Sci. Comput.*, 17(4):848–869, 1996.

[9] N. Halko. *Randomized methods for computing low-rank approximations of matrices*. PhD thesis, Applied Mathematics, University of Colorado at Boulder, 2012.

[10] Nathan Halko, Per-Gunnar Martinsson, Yoel Shkolnisky, and Mark Tygert. An algorithm for the principal component analysis of large data sets. *SIAM Journal on Scientific Computing*, 33(5):2580–2594, 2011.

[11] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[12] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proc. Natl. Acad. Sci. USA*, 104(51):20167–20172, 2007.

[13] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the approximation of matrices. Technical Report Yale CS research report YALEU/DCS/RR-1361, Yale University, Computer Science Department, 2006.

[14] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the decomposition of matrices. *Appl. Comput. Harmon. Anal.*, 30(1):47–68, 2011.

[15] P.G. Martinsson and S. Voronin. A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices., 2015. To appear in *SIAM Journal on Scientific Computation,* Arxiv.org report #1503.07157.

[16] Nikola Mitrovic, Muhammad Tayyab Asif, Umer Rasheed, Justin Dauwels, and Patrick Jaillet. Cur decomposition for compression and compressed sensing of large-scale traffic data. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, pages 1475–1480. IEEE, 2013.

[17] D. C. Sorensen and M. Embree. A DEIM Induced CUR Factorization. *ArXiv e-prints*, July 2014.

[18] G. W. Stewart. On the early history of the singular value decomposition. *SIAM Rev.*, 35(4):551–566, 1993.

[19] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. Siam, 1997.

[20] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335–366, 2008.