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 $\mathbf{A}^* = \mathbf{A}^{-1}$.

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 ED could be complex even for a real matrix.)

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

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

Observe that $\mathbf{B}$ is self-adjoint, 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 [**4, 7**], 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 [**4**, 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 **Q** *and* **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 **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 [**4**, Sec. 5.4] or [**7**, 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 **Q** *and* **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) [**2, 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 $\mathbf{A}\mathbf{x}_2 = \mathbf{0}$. Moreover, $\mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{x}_1 \in Y_1$, so in reality, $\mathbf{A}$ maps X_1 to Y_1.

REMARK 1.4. ... full SVD ...

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

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

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

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

1.6.3. 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

$$\mathbf{x} = \underbrace{\mathbf{V}\mathbf{V}^*\mathbf{x}}_{\in \text{row}(\mathbf{A})} + \underbrace{\big(\mathbf{I} - \mathbf{V}\mathbf{V}^*\big)\mathbf{x}}_{\in \text{null}(\mathbf{A})}.$$

1.6.4. Solving a (least squares) linear system.

CHAPTER 2

Randomized methods for low rank approximation

2.1. Introduction

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 [**5**], 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

$$||\mathbf{A} - \mathbf{U}\mathbf{D}\mathbf{V}^*|| \leq \varepsilon \tag{2.1}$$

unless ε is close to the machine precision. Note that (2.1) 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 [**6**] states that

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

ALGORITHM: 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 $\mathbf{G}$.
(2) Form the sample matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{G}$.
(3) Orthonormalize the columns of the sample matrix $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$.

Stage B:

(4) Form the $(k+p) \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.
(5) Decompose the matrix $\mathbf{B}$ in a singular value decomposition $[\hat{\mathbf{U}},\, \mathbf{D},\, \mathbf{V}] = \texttt{svd}(\mathbf{B})$.
(6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

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.

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 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

$$\underset{n\times n}{\mathbf{A}} \approx \underset{n\times k}{\mathbf{U}}\ \underset{k\times k}{\mathbf{D}}\ \underset{k\times n}{\mathbf{U}^*} \tag{2.2}$$

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

$$\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}. \tag{2.3}$$

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

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

Combine (2.3) and (2.4) to obtain

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

We define

$$\mathbf{C} = \mathbf{Q}^*\mathbf{AQ}. \tag{2.6}$$

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 (2.6). Instead, we will derive a formula for $\mathbf{C}$ that can be evaluated without revisiting $\mathbf{A}$. To this end, multiply (2.6) by $\mathbf{Q}^*\mathbf{G}$ to obtain

$$\mathbf{C}(\mathbf{Q}^*\mathbf{G}) = \mathbf{Q}^*\mathbf{AQQ}^*\mathbf{G} \approx \{\text{Use } (2.4)\} \approx \mathbf{Q}^*\mathbf{AG} = \mathbf{Q}^*\mathbf{Y} \tag{2.7}$$

From (2.4), we know that $\mathbf{AQQ}^* \approx \mathbf{A}$ so we can approximate right hand side in (2.7) 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 $\ell = k + p$)

$$\underset{\ell\times\ell}{\mathbf{C}}\ \underset{\ell\times\ell}{(\mathbf{Q}^*\mathbf{G})} = \underset{\ell\times\ell}{(\mathbf{Q}^*\mathbf{Y})}. \tag{2.8}$$

At first, it may appears that (2.8) 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:

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.

(1) The approximation error in formula (2.5) tends to be larger than the error in (2.3). In fact, with $\varepsilon = \|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$, we have

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

where we used that $\|\mathbf{Q}\mathbf{Q}^*\| \leq 1$ (since $\mathbf{Q}$ is ON, and $\mathbf{Q}\mathbf{Q}^*$ therefore is an ON projection) and that $\|\mathbf{A} - \mathbf{A}\mathbf{Q}\mathbf{Q}^*\| = \|(\mathbf{A} - \mathbf{A}\mathbf{Q}\mathbf{Q}^*)^*\| = \|\mathbf{Q}\mathbf{Q}^*\mathbf{A} - \mathbf{A}\|$. 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.

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

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

Since (2.9) 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

$$\mathbf{C} = \mathbf{Q}_{\rm c}^*\mathbf{A}\mathbf{Q}_r. \tag{2.10}$$

We will derive two relationships that together will determine $\mathbf{C}$ is a manner that is analogous to (2.7). First left multiply (2.10) by $\mathbf{G}_{\rm r}^*\mathbf{Q}_{\rm c}$ to obtain

$$\mathbf{G}_{\rm r}^*\mathbf{Q}_{\rm c}\mathbf{C} = \mathbf{G}_{\rm r}^*\mathbf{Q}_{\rm c}\mathbf{Q}_{\rm c}^*\mathbf{A}\mathbf{Q}_{\rm r} \approx \mathbf{G}_{\rm r}^*\mathbf{A}\mathbf{Q}_{\rm r} = \mathbf{Y}_{\rm r}^*\mathbf{Q}_{\rm r}. \tag{2.11}$$

Next we right multiply (2.10) by $\mathbf{Q}_{\rm r}^*\mathbf{G}_{\rm c}$ to obtain

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

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.

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

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

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.

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

$$\underset{n \times \ell}{\mathbf{\Omega}} \;=\; \underset{n \times n}{\mathbf{D}} \quad \underset{n \times n}{\mathbf{F}} \quad \underset{n \times \ell}{\mathbf{S}}, \tag{2.13}$$

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 (2.13), 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 **??**.

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 [**5**, 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 [**5**] 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*

$$\mathbb{E}\big[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\mathrm{Fro}}\big] \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}. \tag{2.14}$$

The corresponding result for the spectral norm reads

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

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 (2.15) 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 (2.15) 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 [**5**, 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

$$||\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}|| \leq \left(1 + 17\sqrt{1+k/p}\right)\sigma_{k+1} + \frac{8\sqrt{k+p}}{p+1}\left(\sum\nolimits_{j>k}\sigma_j^2\right)^{1/2}, \tag{2.16}$$

with failure probability at most $3\,e^{-p}$, see [**5**, 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

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

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) $\mathbf{Y} = \mathbf{A}\mathbf{G}$;
(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) $\mathbf{Q} = \texttt{orth}(\mathbf{Y})$;
(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.

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:

(1) $\mathbf{G} = \texttt{randn}(n, k+p)$;
(2) $\mathbf{Q} = \texttt{orth}(\mathbf{A}\mathbf{G})$;
(3) **for** $j = 1 : q$
(4) $\quad \mathbf{W} = \texttt{orth}(\mathbf{A}^*\mathbf{Q})$;
(5) $\quad \mathbf{Q} = \texttt{orth}(\mathbf{A}\mathbf{W})$;
(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 [**5**, 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 $\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q\mathbf{A}\mathbf{G}$, and let $\mathbf{Q}$ denote an $m \times (k+p)$ ON matrix resulting from orthonormalizing the columns of $\mathbf{Y}$. Then*

$$\mathbb{E}\big[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|\big] \leq \left[\left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1}^{2q+1} + \frac{e\sqrt{k+p}}{p}\left(\sum\nolimits_{j>k}\sigma_j^{2(2q+1)}\right)^{1/2}\right]^{1/(2q+1)}. \tag{2.17}$$

The bound in (2.17) is slightly hard to read. 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 (2.17)

simplifies to

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

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

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

Observe that $\mathbf{Y}$ 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 [3] and its bibliography. To describe the idea, we first recall from Section 2.4.1 that when $\mathbf{A}$ is Hermitian (which of course is a special case of psd), then it is natural to use the approximation

$$\mathbf{A} \approx \mathbf{Q}\left(\mathbf{Q}^*\mathbf{A}\mathbf{Q}\right)\mathbf{Q}^*. \tag{2.18}$$

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

$$\mathbf{A} \approx (\mathbf{A}\mathbf{Q})\left(\mathbf{Q}^*\mathbf{A}\mathbf{Q}\right)^{-1}(\mathbf{A}\mathbf{Q})^*. \tag{2.19}$$

For both stability and computational efficiency, we typically rewrite (2.19) 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

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

To compute the factor $\mathbf{F}$ numerically, first form the matrices $\mathbf{B}_1 = \mathbf{A}\mathbf{Q}$ 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 (2.19) 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 (2.18). In both cases, we need to twice apply $\mathbf{A}$ to a set of $k+p$ vectors (first in computing $\mathbf{A}\mathbf{G}$, then in computing $\mathbf{A}\mathbf{Q}$). 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 [3]. In particular, Lemma 4 of [3] implies that, in the spectral norm, the Nyström approximation error never exceeds $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|$, and it is often substantially smaller.

ALGORITHM: EIGENVALUE DECOMPOSITION VIA NYSTRÖM METHOD

Given an $n \times n$ positive semidefinite 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.

2.9. Adaptive rank determination

2.10. A blocked method that adaptively determines the rank

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 $\mathbf{v}_1$.

EXERCISE 1. Suppose that $\mathbf{A}$ is a real symmetric matrix with an eigenvalue decomposition (3.1) that satisfies (3.2). Define a sequence of vectors via

$$\mathbf{x}_0 = \texttt{randn}(n, 1), \tag{3.4}$$

$$\mathbf{x}_p = \mathbf{A}\mathbf{x}_{p-1} \qquad p = 1, 2, 3, \ldots, \tag{3.5}$$

so that $\mathbf{x}_p = \mathbf{A}^p \mathbf{x}_0$.

(a) Assume that $\lambda_1 = 1$, and that $\beta = |\lambda_2|/|\lambda_1| < 1$. Prove that the vector $\mathbf{y}_p = \frac{1}{\|\mathbf{x}_p\|}\mathbf{x}_p$ converges to $\pm\mathbf{v}_1$ as $p \to \infty$.
(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 the $\mathbf{y}_p$ will flip back and forth between $\mathbf{v}_1$ and $-\mathbf{v}_1$.
(d) Show that $\|\mathbf{A}\mathbf{y}_p - \lambda_1\mathbf{y}_p\| \to 0$ as $p \to \infty$.

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 $c_1 \neq 0$ with probability 1. (In fact, one can prove that each c_j is a random variable drawn from a normalized Gaussian distribution.)

3.2. Subspace iteration

The power iteration described in Section 3.1 is a very 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. We can express this is matrix form as:

$$\mathbf{X}_0 = \texttt{randn}(n, \ell), \tag{3.6}$$

$$\mathbf{X}_p = \mathbf{A}\mathbf{X}_{j-1} \qquad p = 1, 2, 3, \ldots, \tag{3.7}$$

The end result is a matrix $\mathbf{X}_p$ whose ℓ columns approximately span the same space as the leading ℓ singular vectors. By running Gram-Schmidt on these vectors, we expect to get a set of ℓ orthonormal vectors that should be "decent" approximations to the ℓ dominant eigenvectors.

To improve numerical stability, it is best to orthornomalize in between each iteration:

$$\begin{aligned}
&\mathbf{X}_0 = \texttt{randn}(n, \ell)\\
&\textbf{for } j = 1,\ 2,\ 3, \ldots\\
&\qquad \mathbf{Y} = \mathbf{A}\,\mathbf{X}_{j-1}\\
&\qquad \mathbf{X}_j = \texttt{qr}(\mathbf{Y}, 0)\\
&\textbf{end for}
\end{aligned} \tag{3.8}$$

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

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

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

$$\mathbf{A} = \mathbf{Q}\mathbf{T}\mathbf{Q}^*, \tag{3.9}$$

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

$$\underset{n\times n}{\mathbf{A}} \approx \underset{n\times k}{\mathbf{Q}(:,1:k)} \quad \underset{k\times k}{\mathbf{T}(1:k,1:k)} \quad \underset{k\times n}{\mathbf{Q}(:,1:k)^*}. \tag{3.10}$$

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 (3.9) exists, and derive an algorithm for computing it.

THEOREM 4. *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

$$\mathbf{A}\mathbf{Q} = \mathbf{Q}\mathbf{T}. \tag{3.11}$$

Next, let us write out (3.11) 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

$$t_{21}\,\mathbf{q}_2 = \mathbf{A}\,\mathbf{q}_1 - t_{11}\,\mathbf{q}_1. \tag{3.12}$$

To be precise, (3.12) does not quite determine $\{t_{21},\ \mathbf{q}_2\}$ uniquely. We see that if $\{t_{21},\ \mathbf{q}_2\}$ to (3.12) is one solution of (3.12), 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 (3.11):

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

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

$$t_{32}\,\mathbf{q}_3 = \mathbf{A}\,\mathbf{q}_2 - t_{12}\,\mathbf{q}_1 - t_{22}\,\mathbf{q}_2. \tag{3.13}$$

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

$$\mathbf{A}\,\mathbf{q}_k - t_{k,k-1}\,\mathbf{q}_{k-1} - t_{k,k}\,\mathbf{q}_k. \tag{3.14}$$

does not equal zero.

Now let us consider the special case where the vector in (3.14) 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

$$\mathbf{A}\,\mathbf{q}_k - t_{k,k-1}\,\mathbf{q}_{k-1} - t_{k,k}\,\mathbf{q}_k = \mathbf{0}, \tag{3.15}$$

and consider the subspace

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

We will prove that when (3.15) 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 (3.15). 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 3.1. 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 4. 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 (3.11) 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 3.1 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.

$t_{11} = \mathbf{q}_1^* \mathbf{A} \mathbf{q}_1$
$\mathbf{r}_1 = \mathbf{A} \mathbf{q}_1 - t_{11} \mathbf{q}_1$
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 3.1. 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 $\mathbf{Q}(:, 1) = \mathbf{q}_1$.

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

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

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

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

In other words, we have found one invariant subspace, and the eigenvalues of $\mathbf{T}_{\mathrm{L}}$ 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.

3.4.4. Partial factorization. The eigenvalues of the triangular matrix $\mathbf{T}$ form approximations to the eigenvalues of $\mathbf{A}$. The actual convergence theory is somewhat involved ...

3.4.5. Block Lanczos. The scheme described in Figure 3.1 admits a simple block formulation. The set-up is entirely analogous. We seek a factorization of the form (3.9), 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 3.2.

$\mathbf{T}_{11} = \mathbf{Q}_1^* \mathbf{A} \mathbf{Q}_1$
$\mathbf{R}_1 = \mathbf{A} \mathbf{Q}_1 - \mathbf{T}_{11} \mathbf{Q}_1$
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 3.2. 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 $\mathbf{Q}(:, 1:b) = \mathbf{Q}_1$.

3.5. 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 (3.9) 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

$$\mathbf{A}\mathbf{Q} = \mathbf{Q}\mathbf{H}. \tag{3.16}$$

The first column of (3.16) evaluates to

$$\mathbf{A}\,\mathbf{q}_1 = \mathbf{q}_1\, h_{11} + \mathbf{q}_2\, h_{21}. \tag{3.17}$$

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

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

Rearranging (3.17), we find

$$h_{21}\,\mathbf{q}_2 = \mathbf{A}\,\mathbf{q}_1 - h_{11}\,\mathbf{q}_1, \tag{3.18}$$

Note that if $\{h_{21},\, \mathbf{q}_2\}$ is any solution of (3.12), 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 (3.16):

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

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

$$h_{32}\,\mathbf{q}_3 = \mathbf{A}\,\mathbf{q}_2 - h_{12}\,\mathbf{q}_1 - h_{22}\,\mathbf{q}_2, \tag{3.19}$$

and continue until completion. The resulting algorithm is shown in Figure 3.3.

3.5.1. Blocked Arnoldi. The modification of the basic Arnoldi scheme shown in Figure 3.3 is entirely analogous to the blocking of the Lanczos scheme in Section 3.4.5. The resulting scheme is shown in Figure 3.4.

$h_{11} = \mathbf{q}_1^* \mathbf{A}\, \mathbf{q}_1$
$\mathbf{r}_1 = \mathbf{A}\, \mathbf{q}_1 - h_{11}\, \mathbf{q}_1$
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 3.3. 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 3.4. 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 $\mathbf{Q}(:, 1:b) = \mathbf{Q}_1$.

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 3.3) 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 3.3)
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).

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] 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.

[3] 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.

[4] 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.

[5] 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.

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

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