

## Lecture 4: The QR Decomposition

## 1.2.3. QR decomposition.

The technique of reflecting a vector onto the first coordinate axis (i.e., zeroing out all but the first entry) that we developed in the previous lecture is the essential building block for our first method for constructing the QR decomposition of a general matrix  $\mathbf{A} \in \mathbb{C}^{m \times n}$ , arguably the most fundamental technique in all of numerical linear algebra. We wish to write  $\mathbf{A} = \mathbf{Q}\mathbf{R}$ , where  $\mathbf{Q} \in \mathbb{C}^{m \times m}$  is a unitary matrix and  $\mathbf{R} \in \mathbb{C}^{m \times n}$  is upper triangular (i.e.,  $r_{jk} = 0$  if  $j > k$ ). In general, we shall assume that  $m \geq n$  throughout; this covers the most common situations encountered in applications, and saves us from making a few technical caveats along the way.

We shall follow this methodology: repeatedly apply Householder reflectors on the left of  $\mathbf{A}$ , with the  $j$ th transformation zeroing entries below the diagonal entry in the  $j$ th column. Since the product of unitary matrices is also unitary,<sup>†</sup> the product of these Householder reflectors forms a unitary matrix; call it  $\mathbf{Q}^*$ . Then we have  $\mathbf{Q}^*\mathbf{A} = \mathbf{R}$ , which implies  $\mathbf{A} = \mathbf{Q}\mathbf{R}$ . Now, let us fill in the details.

For simplicity, we shall work in real arithmetic: suppose  $\mathbf{A} \in \mathbb{R}^{m \times n}$ , and write  $\mathbf{A}$  in the form

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \widehat{\mathbf{A}}_1 \end{bmatrix},$$

where  $\mathbf{a}_1 \in \mathbb{R}^m$  and  $\widehat{\mathbf{A}}_1 \in \mathbb{R}^{m \times (n-1)}$ . To construct a Householder transformation that will reflect  $\mathbf{a}_1$ , the first column of  $\mathbf{A}$ , onto the first coordinate direction  $\mathbf{e}_1 \in \mathbb{R}^m$ , we set

$$\mathbf{v}_1 = \mathbf{a}_1 - \|\mathbf{a}_1\|_2 \mathbf{e}_1,$$

as described in the last lecture. This choice<sup>‡</sup> gives

$$\begin{aligned} \mathbf{H}(\mathbf{v}_1)\mathbf{A} &= \begin{bmatrix} \mathbf{H}(\mathbf{v}_1)\mathbf{a}_1 & \mathbf{H}(\mathbf{v}_1)\widehat{\mathbf{A}}_1 \end{bmatrix} \\ &= \begin{bmatrix} \|\mathbf{a}_1\|_2 \mathbf{e}_1 & \mathbf{H}(\mathbf{v}_1)\widehat{\mathbf{A}}_1 \end{bmatrix}. \end{aligned}$$

Define  $\mathbf{Q}_1 = \mathbf{H}(\mathbf{v}_1)$ . Furthermore, set  $r_{11} = \|\mathbf{a}_1\|_2$ , let  $[r_{12}, r_{13}, \dots, r_{1n}]$  denote the first row of  $\mathbf{Q}_1\widehat{\mathbf{A}}_1$ , and let  $\mathbf{A}_2 \in \mathbb{R}^{(m-1) \times (n-1)}$  denote the remaining portion of  $\mathbf{H}(\mathbf{v}_1)\widehat{\mathbf{A}}_1$ . With this notation, we have

$$\mathbf{Q}_1\mathbf{A} = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & \boxed{\mathbf{A}_2} \\ \vdots & & & \\ 0 & & & \end{pmatrix}.$$

<sup>†</sup>Proof: If  $\mathbf{Q}_1$  and  $\mathbf{Q}_2$  are unitary, then  $(\mathbf{Q}_1\mathbf{Q}_2)^*(\mathbf{Q}_1\mathbf{Q}_2) = \mathbf{Q}_2^*\mathbf{Q}_1^*\mathbf{Q}_1\mathbf{Q}_2 = \mathbf{I}$ .

<sup>‡</sup>Actually, we could just as easily reflect  $\mathbf{a}_1$  to the vector  $-\|\mathbf{a}_1\|_2 \mathbf{e}_1$ . The choice of sign has important implications for *numerical stability*, i.e., the robustness of the algorithm to rounding errors on a computer. For this reason, one does not wish to reflect  $\mathbf{a}_1$  to a nearby vector, as subtracting two like quantities can result in a phenomenon called *catastrophic cancellation* that we shall discuss in more detail in later lectures on floating point computer arithmetic. The choice of  $\mathbf{v}$  that reflects  $\mathbf{a}_1$  farthest is  $\mathbf{v}_1 = \mathbf{a}_1 + \text{sign}(a_{11})\|\mathbf{a}_1\|_2 \mathbf{e}_1$ , i.e.,  $\mathbf{a}_1$  is reflected to  $-\text{sign}(a_{11})\|\mathbf{a}_1\|_2 \mathbf{e}_1$ . For complex  $\mathbf{A}$ , we take  $\mathbf{v}_1 = \mathbf{a}_1 \pm e^{i\theta}\|\mathbf{a}_1\|_2 \mathbf{e}_1$ , where  $\theta$  is the argument of the first entry in  $\mathbf{a}_1$ .

The key now is to reflect the first column of  $\mathbf{A}_2$  onto the first coordinate direction,  $\mathbf{e}_1 \in \mathbb{R}^{m-1}$ . On its own, this would be a simple procedure: Like before, partition  $\mathbf{A}_2$  into

$$\mathbf{A}_2 = \begin{bmatrix} \mathbf{a}_2 & \widehat{\mathbf{A}}_2 \end{bmatrix}$$

with first column  $\mathbf{a}_2 \in \mathbb{R}^{m-1}$ . The required reflector then takes the form

$$\mathbf{v}_2 = \mathbf{a}_2 - \|\mathbf{a}_2\|_2 \mathbf{e}_1,$$

so that  $\mathbf{H}(\mathbf{v}_2) \in \mathbb{R}^{(m-1) \times (m-1)}$ .

However, to build a **QR** decomposition, we need to apply the reflector to  $\mathbf{Q}_1 \mathbf{A}$ , not just to the submatrix  $\mathbf{A}_2$ . Procedures like this arise often in matrix decomposition algorithms, and they are a bit fragile: *we want to alter the submatrix  $\mathbf{A}_1$  without disturbing the zero entries we have already created in the first column of  $\mathbf{Q}_1 \mathbf{A}$ .* In this case, the fix is simple: define

$$\mathbf{Q}_2 = \begin{pmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \mathbf{H}(\mathbf{v}_2) \end{pmatrix},$$

which one can verify is also a unitary matrix. Now, we have

$$\mathbf{Q}_2 \mathbf{Q}_1 \mathbf{A} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & \cdots & r_{1n} \\ 0 & r_{22} & r_{23} & \cdots & r_{2n} \\ 0 & 0 & & & \\ \vdots & \vdots & & & \\ 0 & 0 & \boxed{\mathbf{A}_3} & & \end{pmatrix}$$

with  $\mathbf{A}_3 \in \mathbb{R}^{(m-2) \times (n-2)}$ .

Now that we have two columns with zeros below the diagonal, the pattern for future eliminations should be clear. In general,

$$\mathbf{Q}_k = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{H}(\mathbf{v}_k) \end{pmatrix},$$

where  $\mathbf{I}$  is the  $(k-1) \times (k-1)$  identity matrix, and  $\mathbf{H}(\mathbf{v}_k) \in \mathbb{R}^{(m-k+1) \times (m-k+1)}$ . Since  $\mathbf{A}$  has  $n$  columns, we require  $n$  reflectors<sup>§</sup> to zero out all subdiagonal entries of  $\mathbf{A}$ . All together, we have

$$\mathbf{Q}_n \mathbf{Q}_{n-1} \cdots \mathbf{Q}_1 \mathbf{A} = \mathbf{R}.$$

Since our ultimate goal is to obtain a factorization of  $\mathbf{A}$ , we will now move all the unitary matrices to the right hand side of the equation. We can do this by multiplying on the left by  $\mathbf{Q}_n^*$ , then  $\mathbf{Q}_{n-1}^*$ , and so on. Since Householder reflectors are Hermitian matrices ( $\mathbf{Q}_k^* = \mathbf{Q}_k$  for all  $k$ ), we have

$$\mathbf{A} = \mathbf{Q}_1 \mathbf{Q}_2 \cdots \mathbf{Q}_n \mathbf{R}.$$

We define

$$\mathbf{Q} = \mathbf{Q}_1 \mathbf{Q}_2 \cdots \mathbf{Q}_n,$$

and thus arrive at

$$\mathbf{A} = \mathbf{Q} \mathbf{R}.$$

---

<sup>§</sup>If  $m = n$ , then the final column of  $\mathbf{A}$  has no subdiagonal entries, so only  $n - 1$  transformations are necessary.

We have just produced a constructive proof to the following theorem.

**Theorem.** For any matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$  with  $m \geq n$ , there exists a unitary matrix  $\mathbf{Q} \in \mathbb{R}^{m \times m}$  and an upper triangular matrix  $\mathbf{R} \in \mathbb{R}^{m \times n}$  such that  $\mathbf{A} = \mathbf{QR}$ .

The following MATLAB code provides a basic implementation of the QR algorithm described above. To ease your translation of mathematics into MATLAB, this code is given in an extremely inefficient manner. Can you find at least two or three ways to speed up this code while still producing the same  $\mathbf{Q}$  and  $\mathbf{R}$ ?

```
function [Q,R] = slow_householder_qr(A)
% Compute the QR factorization of real A using Householder reflectors
% ** implementation designed for clarity, not efficiency **

[m,n] = size(A);
Q = eye(m);
for k=1:min(m-1,n)
    ak = A(k:end,k); % vector to be zeroed out
    vk = ak + sign(ak(1))*norm(ak)*[1;zeros(m-k,1)]; % construct v_k that defines the reflector
    Hk = eye(m-k+1) - 2*vk*vk'/(vk'*vk); % construct reflector
    Qk = [eye(k-1) zeros(k-1,m-k+1); zeros(m-k+1,k-1) Hk];
    A = Qk*A; % update A
    Q = Q*Qk; % accumulate Q
end
R = A;
```

#### 1.2.4. Built-in MATLAB commands for the QR decomposition.

While it is important to understand the fundamentals of the QR decomposition, you should not use `slow_householder_qr.m` (or your personal implementation) to compute industrial-strength QR decompositions. Excellent versions, both computationally efficient and stable to round-off errors, are available. The LAPACK library, a free collection of numerical linear algebra routines in FORTRAN, is the most popular source for such robust codes.<sup>¶</sup> These codes, which descend from the famous EISPACK and LINPACK libraries, have been written, refined, and tested over several decades; as such, it contains excellent implementations of the best algorithms.

MATLAB (from version 6.0 onward) uses compiled LAPACK routines for its basic linear algebra computations. To compute a QR decomposition of the matrix  $\mathbf{A}$  in MATLAB, simply type

```
[Q,R] = qr(A);
```

Many applications give rise to matrices  $\mathbf{A} \in \mathbb{R}^{m \times n}$  with  $m$  much larger than  $n$ . In this case, you might rightly be concerned about forming the  $m \times m$  matrix  $\mathbf{Q}$ , which will require far more storage than the matrix  $\mathbf{A}$  itself. (Moreover, columns  $n + 1, \dots, m$  of  $\mathbf{Q}$  are superfluous, in that they multiply against zero entries of  $\mathbf{R}$ .) There are two common solutions to this concern, details of which follow in the next lecture. First, you can simply store the vectors  $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$  used to form the Householder reflectors: with this data stored, you can compute the vector  $\mathbf{Q}\mathbf{x}$  for any  $\mathbf{x}$  without explicitly forming the matrix  $\mathbf{Q}$ . The second approach is to compute the QR factorization

<sup>¶</sup>You can download LAPACK (and get reference information) from the NETLIB mathematical software repository, [www.netlib.org](http://www.netlib.org).

through a completely different method, Gram-Schmidt orthogonalization. This procedure results in a *skinny QR decomposition*,  $\mathbf{A} = \mathbf{QR}$  with  $\mathbf{Q} \in \mathbb{C}^{m \times n}$ ,  $\mathbf{R} \in \mathbb{C}^{n \times n}$ , and  $\mathbf{Q}^* \mathbf{Q} = \mathbf{I}$ ), which you can compute in matrix with the command

```
[Q,R] = qr(A,0);
```

Compare some timings in MATLAB for `qr(A)` and `qr(A,0)` for matrices  $\mathbf{A}$  with  $m \gg n$ . This should convince you that for large  $m$ , you will generally want to avoid forming the full-size  $\mathbf{Q}$ !