5.1.3 Computing the Householder Vector

There are a number of important practical details associated with the determination of a Householder matrix, i.e., the determination of a Householder vector. One concerns the choice of sign in the definition of v in (5.1.2). Setting

$$v_1 = x_1 - \| x \|_2$$

has the nice property that Px is a positive multiple of e_1. But this recipe is dangerous if x is close to a positive multiple of e_1 because severe cancellation would occur. However, the formula

$$v_1 \;=\; x_1 - \| x \|_2 \;=\; \frac{x_1^2 - \| x \|_2^2}{x_1 + \| x \|_2} \;=\; \frac{-(x_2^2 + \cdots + x_n^2)}{x_1 + \| x \|_2}$$

suggested by Parlett (1971) does not suffer from this defect in the $x_1 > 0$ case.

In practice, it is handy to normalize the Householder vector so that $v(1) = 1$. This permits the storage of $v(2{:}n)$ where the zeros have been introduced in x, i.e., $x(2{:}n)$. We refer to $v(2{:}n)$ as the *essential part* of the Householder vector. Recalling that $\beta = 2/v^Tv$ and letting **length**(x) specify vector dimension, we obtain the following encapsulation:

Algorithm 5.1.1 (Householder Vector) Given $x \in \mathbb{R}^n$, this function computes $v \in \mathbb{R}^n$ with $v(1) = 1$ and $\beta \in \mathbb{R}$ such that $P = I_n - \beta vv^T$ is orthogonal and $Px = \| x \|_2 e_1$.

function: $[v, \beta] =$ **house**(x)
  $n =$ **length**(x)
  $\sigma = x(2{:}n)^T x(2{:}n)$
  $v = \begin{bmatrix} 1 \\ x(2{:}n) \end{bmatrix}$
  if $\sigma = 0$
    $\beta = 0$
  else
    $\mu = \sqrt{x(1)^2 + \sigma}$
    if $x(1) <= 0$
      $v(1) = x(1) - \mu$
    else
      $v(1) = -\sigma/(x(1) + \mu)$
    end
    $\beta = 2v(1)^2/(\sigma + v(1)^2)$
    $v = v/v(1)$
  end

A production version of Algorithm 5.1.1 may involve a preliminary scaling of the x vector ($x \leftarrow x/\| x \|$) to avoid overflow.

5.1.4 Applying Householder Matrices

It is critical to exploit structure when applying a Householder reflection to a matrix. If $A \in \mathbb{R}^{m\times n}$ and $P = I - \beta vv^T \in \mathbb{R}^{m\times m}$, then

$$PA = \left(I - \beta vv^T\right) A = A - vw^T$$

where $w = \beta A^T v$. Likewise, if $P = I - \beta vv^T \in \mathbb{R}^{n\times n}$, then

$$AP = A\left(I - \beta vv^T\right) = A - wv^T$$

where $w = \beta Av$. Thus, an m-by-n Householder update involves a matrix-vector multiplication and an outer product update. It requires $4mn$ flops. Failure to recognize this and to treat P as a general matrix increases work by an order of magnitude. *Householder updates never entail the explicit formation of the Householder matrix.*

Both of the above Householder updates can be implemented in a way that exploits the fact that $v(1) = 1$. This feature can be important in the computation of PA when m is small and in the computation of AP when n is small.

As an example of a Householder matrix update, suppose we want to overwrite $A \in \mathbb{R}^{m\times n}$ ($m \geq n$) with $B = Q^T A$ where Q is an orthogonal matrix chosen so that $B(j+1{:}m, j) = 0$ for some j that satisfies $1 \leq j \leq n$. In addition, suppose $A(j{:}m, 1{:}j-1) = 0$ and that we want to store the essential part of the Householder vector in $A(j+1{:}m, j)$. The following instructions accomplish this task:

$$\begin{aligned}
&[v, \beta] \;=\; \textbf{house}(A(j{:}m, j)) \\
&A(j{:}m, j{:}n) = (I_{m-j+1} - \beta vv^T)A(j{:}m, j{:}n) \\
&A(j+1{:}m, j) = v(2{:}m-j+1)
\end{aligned}$$

From the computational point of view, we have applied an order $m-j+1$ Householder matrix to the bottom $m-j+1$ rows of A. However, mathematically we have also applied the m-by-m Householder matrix

$$\tilde{P} = \begin{bmatrix} I_{j-1} & 0 \\ 0 & P \end{bmatrix} = I_m - \beta\tilde{v}\tilde{v}^T \qquad \tilde{v} = \begin{bmatrix} 0 \\ v \end{bmatrix}$$

to A in its entirety. Regardless, the "essential" part of the Householder vector can be recorded in the zeroed portion of A.

5.1.5 Roundoff Properties

The roundoff properties associated with Householder matrices are very favorable. Wilkinson (1965, pp. 152-62) shows that **house** produces a Householder vector $\hat{v}$ very near the exact v. If $\hat{P} = I - 2\hat{v}\hat{v}^T/\hat{v}^T\hat{v}$ then

$$\| \hat{P} - P \|_2 = O(\mathbf{u})$$

meaning that $\hat{P}$ is *orthogonal to machine precision.* Moreover, the computed updates with $\hat{P}$ are close to the exact updates with P :

$$\begin{aligned} fl(\hat{P}A) &= P(A+E) \qquad \| E \|_2 = O(\mathbf{u}\| A \|_2) \\ fl(A\hat{P}) &= (A+E)P \qquad \| E \|_2 = O(\mathbf{u}\| A \|_2) \end{aligned}$$

5.1.6 Factored Form Representation

Many Householder based factorization algorithms that are presented in the following sections compute products of Householder matrices

$$Q = Q_1 Q_2 \cdots Q_r \qquad Q_j = I - \beta_j v^{(j)} v^{(j)T} \tag{5.1.3}$$

where $r \le n$ and each $v^{(j)}$ has the form

$$v^{(j)} = (\underbrace{0, 0, \cdots 0,}_{j-1} 1\; v^{(j)}_{j+1}, \cdots, v^{(j)}_n)^T .$$

It is usually not necessary to compute Q explicitly even if it is involved in subsequent calculations. For example, if $C \in \mathbb{R}^{n\times q}$ and we wish to compute $Q^T C$, then we merely execute the loop

for $j = 1{:}r$
    $C = Q_j C$
end

The storage of the Householder vectors $v^{(1)} \cdots v^{(r)}$ and the corresponding β_j (if convenient) amounts to a *factored form* representation of Q. To illustrate the economies of the factored form representation, suppose that we have an array A and that $A(j+1{:}n, j)$ houses $v^{(j)}(j+1{:}n)$, the essential part of the jth Householder vector. The overwriting of $C \in \mathbb{R}^{n\times q}$ with $Q^T C$ can then be implemented as follows:

for $j = 1{:}r$
    $v(j{:}n) = \begin{bmatrix} 1 \\ A(j+1{:}n, j) \end{bmatrix}$ (5.1.4)
    $C(j{:}n,:) = (I - \beta_j v(j{:}n) v(j{:}n)^T) C(j{:}n,:)$
end

This involves about $2qr(2n-r)$ flops. If Q is explicitly represented as an n-by-n matrix, Q^TC would involve $2n^2q$ flops.

Of course, in some applications, it is necessary to explicitly form Q (or parts of it). Two possible algorithms for computing the Householder product matrix Q in (5.1.3) are *forward accumulation,*

$$
\begin{array}{l}
Q = I_n \\
\textbf{for } j = 1{:}r \\
\qquad Q = QQ_j \\
\textbf{end}
\end{array}
$$

and *backward accumulation,*

$$
\begin{array}{l}
Q = I_n \\
\textbf{for } j = r{:}-1{:}1 \\
\qquad Q = Q_jQ \\
\textbf{end}
\end{array}
$$

Recall that the leading $(j-1)$-by-$(j-1)$ portion of Q_j is the identity. Thus, at the beginning of backward accumulation, Q is "mostly the identity" and it gradually becomes full as the iteration progresses. This pattern can be exploited to reduce the number of required flops. In contrast, Q is full in forward accumulation after the first step. For this reason, backward accumulation is cheaper and the strategy of choice:

$$
\begin{array}{l}
Q = I_n \\
\textbf{for } j = r{:}-1{:}1 \\
\qquad v(j{:}n) = \begin{bmatrix} 1 \\ A(j+1{:}n, j) \end{bmatrix} \\
\qquad Q(j{:}n, j{:}n) = (I - \beta_j v(j{:}n)v(j{:}n)^T)Q(j{:}n, j{:}n) \\
\textbf{end}
\end{array}
\tag{5.1.5}
$$

This involves about $4(n^2r - nr^2 + r^3/3)$ flops.

5.1.7 A Block Representation

Suppose $Q = Q_1 \cdots Q_r$ is a product of n-by-n Householder matrices as in (5.1.3). Since each Q_j is a rank-one modification of the identity, it follows from the structure of the Householder vectors that Q is a rank-r modification of the identity and can be written in the form

$$Q = I + WY^T \tag{5.1.6}$$

where W and Y are n-by-r matrices. The key to computing the *block representation* (5.1.6) is the following lemma.