

## Householder $QR$

Algorithmically, this method is very close to G.E. with no pivoting. There, a Gauss transform  $M_k = I + m_k e_k^T$  was used to introduce zeros below the  $(k, k)$  element of  $A^{(k-1)}$ , giving  $A_G^{(k)} = M_k A_G^{(k-1)}$ . Here, a Householder reflector  $H_k = I - \beta u_k u_k^T$  replaces the Gauss transform as the operator that introduces zeros below the  $(k, k)$  element:  $A_H^{(k)} = H_k A_H^{(k-1)}$ .

Let  $A \in \mathbb{R}^{m \times n}$ , with  $m \geq n$ , and let  $p = \min(n, m - 1)$ . The Householder  $QR$  factorization of  $A$  can be coarsely described as

```

A(0) = A
For k = 1:p
    Compute u so that (I - βuuT)A(k-1) has zeros below its (k, k) entry
    Compute A(k) = HkA(k-1)  % this 'updates' A
End

```

There are important computational details to consider yet, but mathematically it is this simple. When the loop terminates, we have  $H_p \cdots H_2 H_1 A = R$ , and defining (the orthogonal matrix)  $Q^T \equiv H_p \cdots H_2 H_1$  gives  $Q^T A = R$ , or  $A = QR$ . Now the details:

We know that if  $u$  is a Householder vector for  $x$ , then  $u$  is a multiple of  $x \pm \|x\|_2 e_1$ , and that (with  $\beta = 2/(u^T u)$ )  $Hx = (I - \beta uu^T)x = \pm \|x\|_2 e_1$ . As with G.E. we can view the  $k^{\text{th}}$  step as

$$A^{(k)} = \begin{bmatrix} R^{(k)} & X^{(k)} \\ 0 & \tilde{A}^{(k)} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & \tilde{H}_k \end{bmatrix} \begin{bmatrix} R^{(k-1)} & X^{(k-1)} \\ 0 & \tilde{A}^{(k-1)} \end{bmatrix} = H_k A^{(k-1)},$$

where  $R^{(k)}$  is  $k \times k$  upper triangular, and  $\tilde{A}^{(k)}$  is  $(m - k) \times (n - k)$ . So  $u$  in the first line of the loop above is  $u_k = (0, \tilde{u}_k^T)^T$ , where  $\tilde{u}_k$  is a Householder vector for  $x = \tilde{A}^{(k-1)} e_1$ .

We rarely actually form the Householder reflector matrices. We simply save the  $\tilde{u}_k$  vectors. If we want to compute  $C = HB = (I - \beta v v^T)B$  for some matrix  $B$  (as in the ‘update’ in the loop above), we simply compute

$$C = B - ((\beta v)(v^T B)).$$

The parenthesis are purposefully placed to suggest efficient implementation. If  $H$  is  $m \times m$  and  $B$  is  $m \times p$ , then the cost of computing  $C$  this way is about  $4mp$  flops (whereas forming  $H$  and computing  $HB$  would require over  $2m^2p$  flops). In the loop above, we use  $\tilde{u}_k$  in place of  $v$ , and  $\tilde{A}^{(k-1)}$  in place of  $B$ . In practice we don’t save the  $A^{(k)}$  separately; instead, we overwrite  $A$  at each step. In Matlab, taking  $v = \tilde{u}_k$ , the update (all but first column of  $\tilde{H}_k \tilde{A}^{(k-1)}$  overwriting  $A$ ) might look like

$$A(k:m, k+1:n) = A(k:m, k+1:n) + (\beta * v) * (v' * A(k:m, k+1:n));$$

We do not explicitly have the matrix  $Q$ , but by saving the  $\tilde{u}_k$ ’s, we have the ‘factored form’ of  $Q$ : all the information needed to compute  $QB$  or  $Q^T C$  for any  $B$  or  $C \in \mathbb{R}^{m \times p}$ .

The cost of this factorization is about  $\sum_{k=0}^{p-1} 4(m - k)(n - k) = 2mn^2 - 2n^3/3 + O(mn)$  flops. Overwriting  $A$  with  $R = Q^T A$  allows us to store all but the first entry of each  $\tilde{u}_k$  in positions  $A(k+1:m, k)$  ( $r_{kk}$  is in the way of the first entry of  $\tilde{u}_k$ ). Typically an extra  $n$ -vector is used to store the first element (the ‘head’) of each  $\tilde{u}_k$ .