Block Algorithms

In some cases, it is useful to develop block versions of matrix computations. Block algorithms work with several rows or columns at once, instead of a single vector, and with matrices instead of scalars. This can be productive from an efficiency standpoint, if a given computer architecture is particularly well suited to working with vectors of a certain length or matrices of a certain size.

However, block algorithms can also have mathematically desirable properties. For example, the factorization $A = LDL^T$, where $A = A^T$ is symmetric, cannot be readily adapted to a skew-symmetric matrix $A$, for which $A = -A^T$. To see this, we attempt to satisfy the requirement that $(LDL^T) = -(LDL^T)$, where $L$ is unit lower triangular and $D$ is diagonal. It follows from the nonsingularity of $L$ that we must have $D = -D^T$, but this implies $d_{ii} = -d_{ii}$ for $i = 1, 2, \ldots, n$, which yields the unfortunate conclusion that $D = 0$. Therefore, such a factorization of a skew-symmetric matrix is not possible.

However, if $n = 2p$ for some integer $p$, we can compute a block $LDL^T$ factorization of a skew-symmetric matrix, $A = LDL^T$, where

$$L = \begin{bmatrix}
L_{11} & L_{12} \\
L_{21} & L_{22} \\
& \ddots \\
L_{p1} & L_{p2} & \cdots & L_{pp}
\end{bmatrix}, \quad D = \begin{bmatrix}
D_1 & \\
& D_2 \\
& \ddots \\
& & D_p
\end{bmatrix},$$

with each $L_{ij}$ and each $D_j$ being a $2 \times 2$ matrix. Then, to satisfy the requirement that $D = -D^T$, we can require that each $D_j$ have the form

$$D_j = \begin{bmatrix}
0 & d_j \\
-d_j & 0
\end{bmatrix},$$

for some scalar $d_j$. We say that $L$ is block unit lower-triangular, and that $D$ is block diagonal. The algorithm for the $LDL^T$ factorization can be adapted to compute this block factorization by replacing scalar multiplication with multiplication by a $2 \times 2$ matrix, and division by a scalar by multiplication by the inverse of a $2 \times 2$ matrix.

It is important to take care that operations are performed in the correct order, since matrix multiplication, unlike scalar multiplication, is not commutative. For example, consider the main
step in Gaussian elimination,

\[ a_{ik} = a_{ik} - m_{ij}a_{jk}, \quad m_{ij} = \frac{a_{ij}}{a_{jj}}. \]

To generalize this to a block version of Gaussian elimination, we note that we must multiply each block \( A_{jk} \) on the left by a “multiplier” \( M_{ij} \), because we are performing a block row operation, which is equivalent to multiplying on the left by an elementary “block row” matrix. However, to obtain a result of \( A_{ik} = 0 \) when \( k = j \), we must have \( A_{ij} - M_{ij}A_{jj} = 0 \), which yields

\[ M_{ij} = A_{ij}A_{jj}^{-1}. \]

That is, we must “divide” on the right for block Gaussian elimination to produce the correct result.

**Gram-Schmidt Orthogonalization**

Givens rotations or Householder reflections can be used to compute the “full” QR decomposition

\[ A = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \]

where \( Q \) is an \( m \times m \) orthogonal matrix, and \( R_1 \) is an \( n \times n \) upper-triangular matrix that is nonsingular if and only if \( A \) is of full column rank (that is, \( \text{rank}(A) = n \)).

It can be seen from the above partitions of \( Q \) and \( R \) that \( A = Q_1R_1 \). Furthermore, it can be shown that \( \text{range}(A) = \text{range}(Q_1) \), and \( (\text{range}(A))^\perp = \text{range}(Q_2) \). In fact, for \( k = 1, \ldots, n \), we have

\[ \text{span}\{a_1, \ldots, a_k\} = \text{span}\{q_1, \ldots, q_k\}, \]

where

\[ A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix}, \quad Q = \begin{bmatrix} q_1 & \cdots & q_m \end{bmatrix} \]

are column partitions of \( A \) and \( Q \), respectively.

We now examine two methods for computing the “thin” or “economy-size” QR factorization \( A = Q_1R_1 \), which is sufficient for solving full-rank least-squares problems, as the least-squares solution \( x \) that minimizes \( \| b - Ax \|_2 \) can be obtained by solving the upper-triangular system \( R_1x = Q_1^Tb \) by back substitution.

**Classical Gram-Schmidt**

Consider the “thin” QR factorization

\[ A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} = \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix} \begin{bmatrix} r_{11} & \cdots & r_{1n} \\
\vdots & \ddots & \vdots \\
r_{nn} & & r_{nn} \end{bmatrix} = QR. \]
From the above matrix product we can see that \( a_1 = r_{11}q_1 \), from which it follows that

\[
grace{11} = \pm \| a_1 \|_2, \quad q_1 = \frac{1}{\| a_1 \|_2} a_1.
\]

For convenience, we choose the + sign for \( r_{11} \).

Next, by taking the inner product of both sides of the equation \( a_2 = r_{12}q_1 + r_{22}q_2 \) with \( q_1 \), and imposing the requirement that the columns of \( Q \) form an orthonormal set, we obtain

\[
r_{12} = q_1^T a_2, \quad r_{22} = \| a_2 - r_{12}q_1 \|_2, \quad q_2 = \frac{1}{r_{22}} (a_2 - r_{12}q_1).
\]

In general, we use the relation

\[
a_k = \sum_{j=1}^{k} r_{jk} q_j
\]

to obtain

\[
q_k = \frac{1}{r_{kk}} \left( a_k - \sum_{j=1}^{k-1} r_{jk} q_j \right), \quad r_{jk} = q_j^T a_k.
\]

Note that \( q_k \) can be rewritten as

\[
q_k = \frac{1}{r_{kk}} \left( I - \sum_{j=1}^{k-1} (q_j^T a_k) q_j \right) = \frac{1}{r_{kk}} \left( a_k - \sum_{j=1}^{k-1} q_j q_j^T a_k \right) = \frac{1}{r_{kk}} \left( I - \sum_{j=1}^{k-1} q_j q_j^T \right) a_k.
\]

If we define \( P_i = q_i q_i^T \), then \( P_i \) is a symmetric projection that satisfies \( P_i^2 = P_i \), and \( P_i P_j = \delta_{ij} \). Thus we can write

\[
q_k = \frac{1}{r_{kk}} \left( I - \sum_{j=1}^{k-1} P_j \right) a_k = \frac{1}{r_{kk}} \prod_{j=1}^{k-1} (I - P_j) a_k.
\]

Unfortunately, Gram-Schmidt orthogonalization, as described, is numerically unstable, because, for example, if \( a_1 \) and \( a_2 \) are almost parallel, then \( a_2 - r_{12}q_1 \) is almost zero, and roundoff error becomes significant due to catastrophic cancellation.

**Modified Gram-Schmidt**

The *Modified Gram-Schmidt* method alleviates the numerical instability of “Classical” Gram-Schmidt. Recall

\[
A = Q_1 R_1 = \begin{bmatrix} r_{11}q_1 & r_{12}q_1 + r_{22}q_2 & \cdots \end{bmatrix}
\]

We define

\[
C^{(k)} = \sum_{i=1}^{k-1} q_i r_i^T, \quad r_i^T = \begin{bmatrix} 0 & \cdots & 0 & r_{ii} & r_{i,i+1} & \cdots & r_{in} \end{bmatrix}
\]

3
which means
\[ A - C^{(k)} = \begin{bmatrix} 0 & 0 & \cdots & 0 & A^{(k)} \end{bmatrix}, \]
because the first \( k - 1 \) columns of \( A \) are linear combinations of the first \( k - 1 \) columns of \( Q_1 \), and the contributions of these columns of \( Q_1 \) to all columns of \( A \) are removed by subtracting \( C^{(k)} \).

If we write
\[ A^{(k)} = \begin{bmatrix} z_k & B_k \end{bmatrix} \]
then, because the \( k \)th column of \( A \) is a linear combination of the first \( k \) columns of \( Q_1 \), and the contributions of the first \( k - 1 \) columns are removed in \( A^{(k)} \), \( z_k \) must be a multiple of \( q_k \). Therefore,
\[ r_{kk} = \|z_k\|_2, \quad q_k = \frac{1}{r_{kk}}z_k. \]
We then compute
\[ \begin{bmatrix} r_{k,k+1} & \cdots & r_{kn} \end{bmatrix} = q_k^T B_k \]
which yields
\[ A^{(k+1)} = B_k - q_k \begin{bmatrix} r_{k,k+1} & \cdots & r_{kn} \end{bmatrix}. \]
This process is numerically stable.

Note that Modified Gram-Schmidt computes the entries of \( R_1 \) row-by-row, rather than column-by-column, as Classical Gram-Schmidt does. This rearrangement of the order of operations, while mathematically equivalent to Classical Gram-Schmidt, is much more stable, numerically, because each entry of \( R_1 \) is obtained by computing an inner product of a column of \( Q_1 \) with a modified column of \( A \), from which the contributions of all previous columns of \( Q_1 \) have been removed.

To see why this is significant, consider the inner products
\[ u^T v, \quad u^T(v + w), \]
where \( u^T w = 0 \). The above inner products are equal, but suppose that \( |u^T v| \ll \|w\| \). Then \( u^T v \) is a small number that is being computed by subtraction of potentially large numbers, which is susceptible to catastrophic cancellation.

It can be shown that Modified Gram-Schmidt produces a matrix \( \tilde{Q}_1 \) such that
\[ \tilde{Q}_1^T \tilde{Q}_1 = I + E_{MGS}, \quad \|E_{MGS}\| \approx \kappa_2(A), \]
and \( \tilde{Q}_1 \) can be computed in approximately \( 2mn^2 \) flops (floating-point operations), whereas with Householder QR,
\[ \tilde{Q}_1^T \tilde{Q}_1 = I + E_n, \quad \|E_n\| \approx u, \]
with \( \tilde{Q}_1 \) being computed in approximately \( 2mn^2 - 2n^2/3 \) flops to factor \( A \) and an additional \( 2mn^2 - 2n^2/3 \) flops to obtain \( \tilde{Q}_1 \), the first \( n \) columns of \( Q \). That is, Householder QR is much less sensitive to roundoff error than Gram-Schmidt, even with modification, although Gram-Schmidt is more efficient if an explicit representation of \( Q_1 \) desired.