The important lines subtract from \( v = a_j \) its projection onto each \( q_i \):

\[
 r_{kj} = \sum_{i=1}^{m} q_{ik} v_{ij} \quad \text{and} \quad v_{ij} = v_{ij} - q_{ik} r_{kj}.
\]

Starting from \( a, b, c = a_1, a_2, a_3 \) this code will construct \( q_1, B, q_2, C, q_3 \):

\[
q_1 = a_1 / \|a_1\| \quad \text{and} \quad B = a_2 - (q_1^T a_2) q_1 \quad q_2 = B / \|B\| \\
C^* = a_3 - (q_1^T a_3) q_1 \quad C = C^* - (q_2^T C^*) q_2 \quad q_3 = C / \|C\|
\]

Equation (12) subtracts off projections as soon as the new vector \( q_k \) is found. This change to "subtract one projection at a time" is called *modified Gram-Schmidt*. That is numerically more stable than equation (8) which subtracts all projections at once.

```
for j = 1:n
  v = A(:,j);
  for i = 1:j-1
    R(i,j) = Q(:,i)^*v;
    v = v - R(i,j)*Q(:,i);
  end
  R(j,j) = norm(v);
  Q(:,j) = v/R(j,j);
end
```

To recover column \( j \) of \( A \), undo the last step and the middle steps of the code:

\[
R(j,j)q_j = (v \text{ minus its projections}) = (\text{column } j \text{ of } A) - \sum_{i=1}^{j-1} R(i,j)q_i.
\]

**Moving the sum to the far left, this is column } j} \text{ in the multiplication } A = QR.**

**Confession** Good software like LAPACK, used in good systems like MATLAB and Octave and Python, will not use this Gram-Schmidt code. There is now a better way. “Householder reflections” produce the upper triangular \( R \), one column at a time, exactly as elimination produces the upper triangular \( U \).

Those reflection matrices \( I - 2uu^T \) will be described in Chapter 9 on numerical linear algebra. If \( A \) is tridiagonal we can simplify even more to use 2 by 2 rotations. The result is always \( A = QR \) and the MATLAB command is \([Q, R] = qr(A)\). I believe that Gram-Schmidt is still the good process to understand, even if the reflections or rotations lead to a more perfect \( Q \).
The vectors \( a \) and \( A \) and \( q_1 \) are all along a single line.

The vectors \( a, b \) and \( A, B \) and \( q_1, q_2 \) are all in the same plane.

The vectors \( a, b, c \) and \( A, B, C \) and \( q_1, q_2, q_3 \) are in one subspace (dimension 3).

At every step \( a_1, \ldots, a_k \) are combinations of \( q_1, \ldots, q_k \). Later \( q \)'s are not involved. The connecting matrix \( R \) is triangular, and we have \( A = QR \):

\[
\begin{bmatrix}
a & b & c \\
q_1 & q_2 & q_3 \\
q_1^T a & q_1^T b & q_1^T c \\
q_2^T b & q_2^T c \\
q_3^T c \\
\end{bmatrix}
= A = QR.
\]

\( A = QR \) is Gram-Schmidt in a nutshell. Multiply by \( Q^T \) to see why \( R = Q^T A \).

(Gram-Schmidt) From independent vectors \( a_1, \ldots, a_n \), Gram-Schmidt constructs orthonormal vectors \( q_1, \ldots, q_n \). The matrices with these columns satisfy \( A = QR \). Then \( R = Q^T A \) is upper triangular because later \( q \)'s are orthogonal to earlier \( a \)'s.

Here are the \( a \)'s and \( q \)'s from the example. The \( i, j \) entry of \( R = Q^T A \) is row \( i \) of \( Q^T \) times column \( j \) of \( A \). This is the dot product of \( q_i \) with \( a_j \):

\[
A = \begin{bmatrix}
1 & 2 & 3 \\
-1 & 0 & -3 \\
0 & -2 & 3 \\
\end{bmatrix} = \begin{bmatrix}
1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{3} \\
-1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{3} \\
0 & -2/\sqrt{6} & 1/\sqrt{3} \\
\end{bmatrix} \begin{bmatrix}
\sqrt{2} & \sqrt{2} & \sqrt{18} \\
0 & -\sqrt{6} & -\sqrt{6} \\
0 & 0 & -\sqrt{3} \\
\end{bmatrix} = QR.
\]

The lengths of \( A, B, C \) are the numbers \( \sqrt{2}, \sqrt{6}, \sqrt{3} \) on the diagonal of \( R \). Because of the square roots, \( QR \) looks less beautiful than \( LU \). Both factorizations are absolutely central to calculations in linear algebra.

Any \( m \) by \( n \) matrix \( A \) with independent columns can be factored into \( QR \). The \( m \) by \( n \) matrix \( Q \) has orthonormal columns, and the square matrix \( R \) is upper triangular with positive diagonal. We must not forget why this is useful for least squares: \( A^T A \) equals \( R^T Q^T QR = R^T R \). The least squares equation \( A^T A \bar{x} = A^T b \) simplifies to \( Rx = Q^T b \):

Least squares \( R^T R \bar{x} = R^T Q^T b \ \text{or} \ Q \bar{x} = Q^T b \ \text{or} \ \bar{x} = R^{-1} Q^T b \)

Instead of solving \( Ax = b \), which is impossible, we solve \( R \bar{x} = Q^T b \) by back substitution—which is very fast. The real cost is the \( mn^2 \) multiplications in the Gram-Schmidt process, which are needed to construct the orthogonal \( Q \) and the triangular \( R \).

Below is an informal code. It executes equations (11) and (12), for \( k = 1 \) then \( k = 2 \) and eventually \( k = n \). The last line of that code normalizes to unit vectors \( q_j \):

Divide by length \( q_j = \text{unit vector} \)

\[
q_{jj} = \left( \sum_{i=1}^{m} v_{ij}^2 \right)^{1/2} \quad \text{and} \quad q_{ij} = \frac{v_{ij}}{r_{jj}} \quad \text{for} \quad i = 1, \ldots, m.
\]
qr demo -- QR factorization using LAPACK
Written November 26 by Eric Olson

This program uses LAPACKE compiled with ATLAS BLAS. The example performs the following steps:

1. Given A we call LAPACKE_dgeqrf which overwrites A with the results of the QR decomposition.

2. Extract the upper triangular matrix from the output and store it as the matrix R.

3. Call LAPACKE_dormqr to multiply R by Q and store the product overwriting matrix R.

4. Compute the error between QR and the original matrix.

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <lapacke.h>

void vecprint(int n, double v[n]) {
    int j;
    for (j = 0; j < n; j++)
        printf("%10.2e%c", v[j], j == n - 1 ? '\n' : ' ');
}

void matprint(int n, double A[n][n]) {
    int i;
    for (i = 0; i < n; i++)
        vecprint(n, A[i]);
}

int main() {
    printf("qr demo -- QR factorization using LAPACK Version 2\n";
    "Written November 26 by Eric Olson\n\n");
    int i, j, n = 5;
    double A[n][n], tau[n];
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            A[i][j] = i * n + j;

    printf("A=\n");
    matprint(n, A);
    LAPACKE_dgeqrf(LAPACK_ROW_MAJOR, n, n, A[0], n, tau);
    printf("output from dgeqrf=\n");
    matprint(n, A);
    printf("tau=\n");
    vecprint(n, tau);
    double R[n][n];
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            R[i][j] = j < i ? 0 : A[i][j];

    printf("R=\n");
    matprint(n, R);
    LAPACKE_dormqr(LAPACK_ROW_MAJOR, 'L', 'N',
        n, n, n, A[0], n, tau, R[0], n);
    printf("QR=\n");
    matprint(n, R);

    double rmserr = 0.0;
for(i = 0; i < n; i++)
    for(j = 0; j < n; j++) {  
        double t = R[i][j] - (i * n + j);
        rmserr += t * t;
    }
    rmserr = sqrt(rmserr / n / n);
printf("error=\n%23.15e\n", rmserr);
return 0;