CS322 Lecture Notes: QR factorization and orthogonal transformations

Steve Marschner
Cornell University

2–5 March 2007

In this lecture I'll talk about orthogonal matrices and their properties, discuss how they can be used to compute a matrix factorization, called the QR factorization, that is similar in some ways to the LU factorization we studied earlier but with an orthogonal factor replacing the lower triangular one, then show how the Q and R factors can be used to compute solutions to least squares problems.

1 Orthogonal matrices

A matrix is orthogonal if its columns are unit length and mutually perpendicular. That is, if we name the columns $\mathbf{q}_j$ so that $\mathbf{Q} = \begin{bmatrix}\mathbf{q}_1 \cdots \mathbf{q}_n\end{bmatrix}$, then $\|\mathbf{q}_j\| = 1$ for all j and $\mathbf{q}_i \cdot \mathbf{q}_j = 0$ whenever $i \neq j$.

Orthogonal matrices are useful for many kinds of manipulations where we'd like to preserve a lot of the properties of the vectors or matrices we are transforming.

Properties of orthogonal matrices. For a square orthogonal matrix Q:

- $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$, or $\mathbf{q}_i^T\mathbf{q}_j = \delta_{ij}$ (the columns are orthonormal)
- $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$ (the rows are also orthonormal)
- $\mathbf{Q}^{-1} = \mathbf{Q}^T$ (because of the previous two properties, the transpose is the inverse)
- $\det \mathbf{Q} = \pm 1$
- $\|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$ (orthogonal transformation preserves lengths in the usual Euclidan norm, or 2-norm)
- $(\mathbf{Q}\mathbf{x}) \cdot (\mathbf{Q}\mathbf{y}) = \mathbf{x} \cdot \mathbf{y}$ (orthogonal transformation preserves angles)

Aside: Matrix norms. In the same way that it's useful to measure the size, or length, or magnitude, of a vector, it's also useful to have some ways to measure the "magnitude" of a linear transformation. This leads to the idea of a *matrix norm*, which is a way of measuring size for matrices that behaves like length for vectors. I will just go so far as to mention a couple of them here.

Motivated by the idea that one could think of the magnitude of a scalar being a measurement of how much it makes things bigger or smaller when you multiply by it:

$$|a| = \frac{|ax|}{|x|}$$

we can define a similar thing for matrices. The difference is that a matrix doesn't magnify all vectors the same amount, so we have to pick a particular vector. To get the matrix 2-norm, we take the largest magnification factor over all nonzero vectors:

$$\|\mathbf{A}\|_2 = \max_{\mathbf{x}\neq 0} \frac{\|\mathbf{Ax}\|_2}{\|\mathbf{x}\|_2}$$

The trouble with the matrix 2-norm is that it is not trivial to compute: you have to know which vector $\mathbf{x}$ to use. We will learn how to do this, but when something a little more direct is needed, we can use the *Frobenius* norm, which is computed just like the vector 2-norm, summing the squares of all the elements in the matrix without regard for rows and columns, then taking the square root:

$$\|\mathbf{A}\|_F^2 = \sum_{ij} |a_{ij}|^2$$

Note that you can rearrange this sum as the sum of squares of row norms or the sum of squares of column norms.

A few facts about orthogonal matrices and matrix norms:

- $\|\mathbf{Q}\|_2 = 1$ (obvious because $\mathbf{Q}$ preserves length)
- $\|\mathbf{QA}\|_2 = \|\mathbf{A}\|_2$ (because $\mathbf{QAx}$ has the same length as $\mathbf{Ax}$)
- $\|\mathbf{QA}\|_F = \|\mathbf{A}\|_F$ (because the norms of all the columns are preserved)

Geometric intuition for orthogonal transformations. Orthogonal transformations correspond to the geometric ideas of rotation and mirror reflection. You can also think of an orthogonal transformation as a change of coordinates from one orthonormal basis to another.

Geometrically, we know that an orthonormal basis is more convenient than just any old basis, because it is easy to compute coordinates of vectors with respect to such a basis (Figure 1).

Computing coordinates in an orthonormal basis using dot products instead of a linear system is exactly the same idea as inverting an orthogonal matrix using $\mathbf{Q}^T$ rather than inverting a general matrix by the much more expensive computation of $\mathbf{A}^{-1}$.

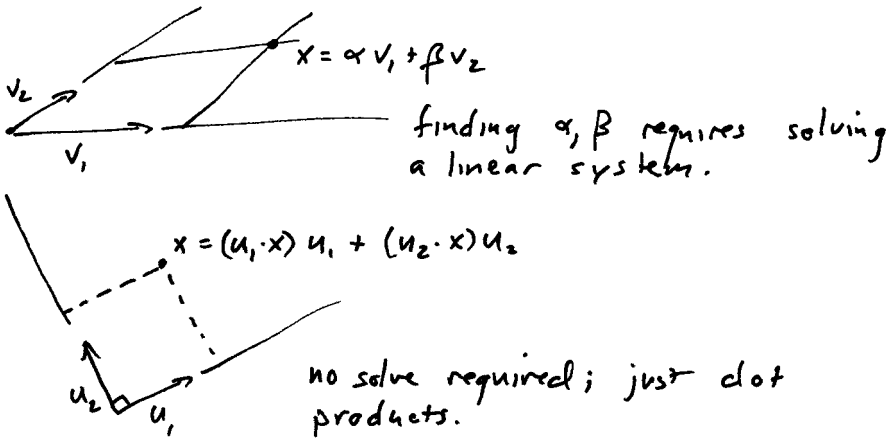
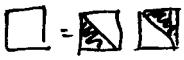


Figure 1: Computing coordinates in arbitrary (top) and orthonormal (bottom) bases.

2 The QR factorization

In view of this idea of coordinate systems, let's look back at the LU factorization:

$$\mathbf{A} = \mathbf{LU}$$



We can think of this equation as L changing the coordinate system in which we express the "output" of **A**. In the new coordinate system the transformation defined by **A** happens to have a convenient form: it is represented now by an upper triangular matrix. **U** "does the same thing as" **A** but just returns its result in a different coordinate system.

So when we solve $\mathbf{Ax} = \mathbf{b}$ by computing

$$\mathbf{y} = \mathbf{L}\backslash\mathbf{b}$$
$$\mathbf{x} = \mathbf{U}\backslash\mathbf{y}$$

the vector **y** is just **x** expressed in a coordinate system where **A** becomes convenient to work with.

Now, maybe we can do one better than LU by finding not just a coordinate system in which our transformation becomes upper triangular, but an *orthogonal* coordinate system in which our transformation becomes upper triangular. This is the basic idea of a new matrix factorization, the QR factorization, which factors **A** into a the product of an orthogonal matrix and an upper triangular matrix:

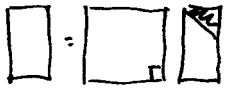
$$\mathbf{A} = \mathbf{QR}$$

This factorization has some similarities to LU:

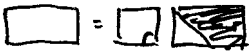
- **L** is easy to invert; so is **Q**.
- **U** is upper triangular; so is **R**.

but **Q** will preserve norm, dot products, etc. at the same time! This makes this factorization very suitable for questions where norm is important, and leads to better (more accurate) methods for least squares problems.

Preview: one other difference is that QR can be applied to non-square matrices, resulting in factors that look like this:



or this:



2.1 Computing the QR factorization

When we talked earlier about computing the LU factorization, we reduced **A** to the upper triangular matrix **U** by applying a sequence of special lower triangular matrices with simple structure, known as Gauss transformations. The **L** factor was then the inverse product of all those transformations.

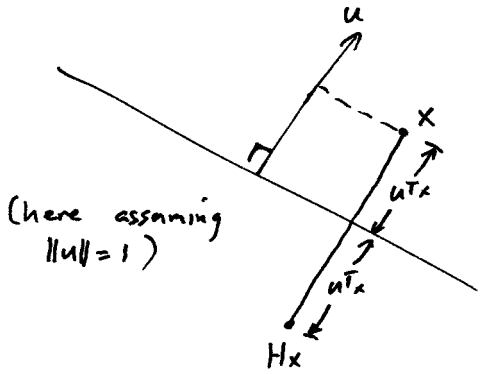
We'll compute the QR factorization similarly: we'll reduce **A** to the upper triangular matrix **R** by applying a sequence of special orthogonal transformations with simple structure, known as Householder reflections. The **Q** factor is then the inverse product of all those reflections.

Householder reflections. Householder matrices are orthogonal matrices (they are reflections) that are convenient for introducing zeros into a matrix, in the same way that Gauss transformations are.

A Householder matrix is defined by a nonzero vector **u**, and it's just a reflection along the **u** direction. (Another way to say this is that it's a mirror reflection across the subspace orthogonal to **u**.) Algebraically,

$$\mathbf{H} = I - 2\frac{\mathbf{u}\mathbf{u}^T}{\|\mathbf{u}\|^2} \quad \text{so} \quad \mathbf{H}\mathbf{x} = I - 2\frac{\mathbf{u}\mathbf{u}^T\mathbf{x}}{\|\mathbf{u}\|^2}$$

The geometric connection is given by this picture:

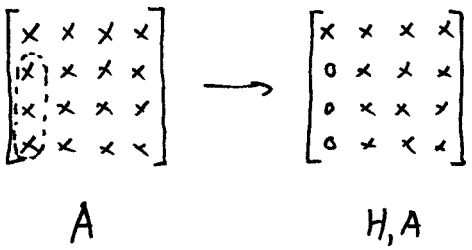


(Exercise: show that $\mathbf{H}$ is symmetric and orthogonal.)

The key thing about a Householder reflection is that it differs from the identity by a rank-1 matrix (the columns of the outer product $\mathbf{u}\mathbf{u}^T$ are all parallel to $\mathbf{u}$). The product of a matrix with $\mathbf{H}$ is called a "rank-1 update" and is efficient to compute.

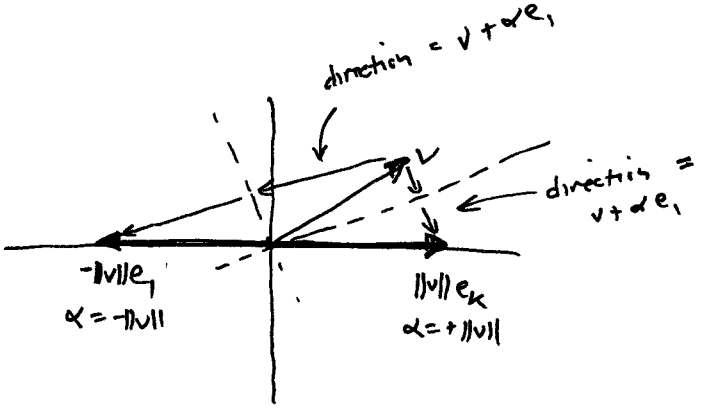
(Note that a Gauss transformation can be written in the same way: $G = I - \tau \mathbf{e}_k^T$. It is also a rank-1 update, but also has a sparse structure.)

QR factorization algorithm. The algorithm to compute the QR factorization using Householder reflections proceeds very much like the LU algorithm. In the first step, we apply a transformation that will zero out everything in the first column below the $(1, 1)$ entry.



We want to apply a transform that maps the first column to $\begin{bmatrix}\alpha & 0 & 0 & 0\end{bmatrix}^T$ for some α. Because orthogonal transforms preserve norm, we know that $|\alpha| = \|\mathbf{v}\|$ where $\mathbf{v}$ is the first column.

It turns out that the Householder vector to do this is $\mathbf{v} - \alpha \mathbf{e}_k$, as suggested by this picture:



(exercise: prove that the Householder transformation defined by this vector does in fact map $\mathbf{v}$ to $\mathbf{e}_k$). When we choose the sign of alpha, we should choose it to be opposite the sign of v_k to avoid loss of precision due to cancellation.

To run the whole algorithm, we repeat this process n times, each time looking at a smaller block of the matrix and zeroing out the subdiagonal in the next column.

When we apply a Householder reflection, we *do not* form the Householder matrix and then multiply; this would take $O(m^2n)$ time. Rather, we take advantage of the outer product structure:

$$(\mathbf{I} - 2\mathbf{u}\mathbf{u}^T)\mathbf{A} = \mathbf{A} - 2\mathbf{u}(\mathbf{u}^T\mathbf{A})$$

This operation consists of a matrix-vector multiplication and an outer product update. Each costs $2mn$ floating-point operations (flops).

Differences between LU and QR:

- Gauss transforms vs. Householder matrices
- Asymptotic flops $n^3/3$ vs. $2n^3/3$.
- Only square matrices vs. rectangular or square
- Requires pivoting vs. requires no pivoting (for full rank)

3 Using QR to solve least squares problems.

The real attraction of QR is its usefulness in solving non-square linear systems. What is important is that the $\mathbf{Q}$ factor provides orthonormal bases for the span of the columns of $\mathbf{A}$.

Aside: Subspaces A matrix has four subspaces associated with it: the *range*, the *null space* and the orthogonal complements of these two spaces.

The range is the set of everything you can produce by multiplying vectors by $\mathbf{A}$. This is the span of the columns of A:

$$\begin{aligned} \text{ran}(A) &= \{\mathbf{A}\mathbf{x} | \mathbf{x} \in I\!R^n\} \\ \text{null}(A) &= \{\mathbf{x} | \mathbf{A}\mathbf{x} = 0\} \\ V^\perp &= \{\mathbf{y} | \forall \mathbf{x} \in V, \mathbf{x} \cdot \mathbf{y} = 0\} \end{aligned}$$

Note that $\text{ran}(A)$ is the span of the columns and $\text{null}(A)^\perp$ is the span of the rows.

For full rank matrices these ideas come into play depending on the shape of $\mathbf{A}$. If A is tall ($m > n$), then there are not enough columns to span $I\!R^m$, so $\text{ran}(A)$ is less than all of $I\!R^m$ and $\text{ran}(A)^\perp$ is nontrivial. Conversely there are plenty of rows, so as long as $\mathbf{A}$ is full rank there is no null space (or rather $\text{null}(A) = \{\mathbf{0}\}$). If $\mathbf{A}$ is wide ($n > m$) then there are not enough rows, so that $\text{null}(\mathbf{A})$ and $\text{null}(\mathbf{A})^\perp$ are nontrivial; conversely there are plenty of columns so $\text{ran}(A) = I\!R^m$ in the full-rank case.

These four ideas: $\text{ran}(A)$, $\text{null}(A)$, $\text{ran}(A)^\perp$, and $\text{null}(A)^\perp$, are important in thinking about rank and about least squares.

Overdetermined systems. If we are solving

$$\mathbf{A}\mathbf{x} \approx \mathbf{b}$$

for tall $\mathbf{A}$, recall that the key characteristic of the solution is that the residual is orthogonal to the columns of $\mathbf{A}$. Another way to say this is that $\mathbf{A}\mathbf{x} \in \text{ran}(A)$ and $\mathbf{A}\mathbf{x} - \mathbf{b} \in \text{ran}(A)^\perp$. The matrix $\mathbf{Q}$ provides bases for these two subspaces:

The product $\mathbf{A}\mathbf{x}$ is a linear combination of the columns of $\mathbf{Q}_1$, so $\mathbf{Q}_1$ is a basis for the range of $\mathbf{A}$. Since $\mathbf{Q}_2$ multiplies with the zero part of $\mathbf{R}$, the result never has a component in any of those directions, so $\mathbf{Q}_2$ is a basis for $\text{ran}(A)^\perp$.

In the least squares setting, we can use $\mathbf{Q}$ to transform the problem into coordinates where these two spaces are in separate rows of the system, and then we can pay attention only to the relevant components. The reasoning is as follows: we want to minimize

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$$

but transforming that difference by an orthogonal matrix won't change the norm:

$$\|\mathbf{Q}^T(\mathbf{A}\mathbf{x} - \mathbf{b})\|^2 = \|\mathbf{Q}^T\mathbf{A}\mathbf{x} - \mathbf{Q}^T\mathbf{b}\|^2 = \|\mathbf{R}\mathbf{x} - \mathbf{Q}^T\mathbf{b}\|^2$$

Writing this in block form, we can separate the first n rows from the rest:

$$\left\| \begin{bmatrix} \mathbf{R}_1 \\ 0 \end{bmatrix} \mathbf{x} - \begin{bmatrix} \mathbf{Q}_1^T\mathbf{b} \\ \mathbf{Q}_2^T\mathbf{b} \end{bmatrix} \right\|^2 = \left\| \begin{bmatrix} \mathbf{R}_1\mathbf{x} - \mathbf{Q}_1^T\mathbf{b} \\ \mathbf{Q}_2^T\mathbf{b} \end{bmatrix} \right\|^2 = \|\mathbf{R}_1\mathbf{x} - \mathbf{Q}_1^T\mathbf{b}\| + \|\mathbf{Q}_2^T\mathbf{b}\|$$

The second term in this equation does not depend on $\mathbf{x}$, so it is irrelevant to the minimization. Minimizing the first term is minimizing the norm of a square

system; if $\mathbf{R}_1$ is nonsingular (which it is if $\mathbf{A}$ is), then the minimum norm is zero and the solution to $\mathbf{R}_1\mathbf{x} = \mathbf{Q}_1^T\mathbf{b}$ can be found by back substitution. The second term is the residual of the least squares system.

Thus the QR factorization, like the normal equations followed by LU factorization, reduces the overdetermined system to a square one with an upper triangular matrix. The difference is that it does so via a single orthogonal transformation, which results (as we will see when we learn more about accuracy and condition numbers) in a more accurate solution.

Note that we don't actually need $\mathbf{Q}_2$ to compute the solution; the QR factorization can be computed more efficiently if only $\mathbf{Q}_1$ is computed, rather than all of $\mathbf{Q}$. Matlab calls this the "economy size" $\mathbf{Q}$ and you can get it via `[Q,R] = qr(A, 0)`.

Underdetermined systems. In this case we have the opposite problem: many values for $\mathbf{x}$ will solve the system exactly, but we need to choose one in particular. One sensible choice in many contexts is the minimum-norm solution. In terms of the spaces we discussed earlier, the component of the minimum-norm solution has in any direction in the null space of $\mathbf{A}$ os zero—for if $\mathbf{x}$ had any component in the null space, it could be subtracted off without changing $\mathbf{Ax}$, resulting in a smaller-norm solution.

The solution of an underdetermined system requires bases for the row spaces $\text{null}(A)$ and $\text{null}(A)^\perp$, which can be had by factoring $\mathbf{A}^T$:

$$\mathbf{QR} = \mathbf{A}^T \quad \text{so} \quad \mathbf{A} = \mathbf{R}^T\mathbf{Q}^T = \begin{bmatrix}\mathbf{R}_1^T & 0\end{bmatrix}\begin{bmatrix}\mathbf{Q}_1^T \\ \mathbf{Q}_2^T\end{bmatrix}$$

For any $\mathbf{x}$, $\mathbf{Ax} = \mathbf{R}_1^T\mathbf{Q}_1^T\mathbf{x} + \mathbf{0}\mathbf{Q}_2^T\mathbf{x}$. The components of $\mathbf{x}$ in the directions in $\mathbf{Q}_1$ affect the result, but the components in the directions in $\mathbf{Q}_2$ do not. So $\mathbf{Q}^2$ is a basis for $\text{null}(\mathbf{A})$. Working out the solution to the underconstrained system we have

$$\mathbf{Ax} = \mathbf{R}_1^T\mathbf{Q}_1^T\mathbf{x} + \mathbf{0}\mathbf{Q}_2^T\mathbf{x} = \mathbf{b}$$

so $\mathbf{Q}_1^T\mathbf{x}$ has to be $\mathbf{R}_1^T\backslash\mathbf{b}$. On the other hand $\mathbf{Q}_2^T\mathbf{x}$ can be anything—it doesn't affect the system at all. To get the minimum norm we set it to zero, leading to the solution

$$\mathbf{x} = \mathbf{Q}\begin{bmatrix}\mathbf{R}_1^T\backslash\mathbf{b} \\ \mathbf{0}\end{bmatrix} = \mathbf{Q}_1(\mathbf{R}_1^T\backslash\mathbf{b})$$

Sources

- Our textbook: Moler, *Numerical Computing in Matlab*. Chapter 5.
- Golub and Van Loan, *Matrix Computations*, Third edition. Johns Hopkins Univ. Press, 1996. Chapter 5.