CS 542G: QR, Weighted Least Squares, MLS

Robert Bridson

October 6, 2008

1 The QR Factorization

We established the Gram-Schmidt process last time as a start towards an alternative algorithm for solving least squares problems: while the normal equations approach (using Cholesky factorization) is very attractive when A^TA is well-conditioned, it is prone to failure on more challenging problems that should nevertheless be numerically solvable. The last problem we faced with Gram-Schmidt was how to get back the least-squares solution in terms of the original basis (corresponding to the columns of A) as opposed to the orthonormal basis (the columns of Q).

Slightly rewriting the statement of the Gram-Schmidt process from last time makes the relationship between A and Q clearer:

$$\begin{aligned}
a_1 &= \|\tilde{q}_1\| q_1 \\
a_2 &= \|\tilde{q}_2\| q_2 + q_1 (q_1 \cdot a_2) \\
&\vdots \\
a_i &= \|\tilde{q}_i\| q_i + \sum_{j=1}^{i-1} q_j (q_j \cdot a_i) \\
&\vdots
\end{aligned}$$

Introduce the following labeling for the coefficients computed during Gram-Schmidt:

$$\begin{aligned}
R_{ii} &= \|\tilde{q}_i\| \\
R_{ji} &= q_j \cdot a_i \quad \text{for } j < i \\
R_{ji} &= 0 \quad \text{for } j > i
\end{aligned}$$

What we have defined is an upper triangular matrix R, such that the above statement of Gram-Schmidt can be summarized as:

$$A = QR$$

This is called the QR factorization of the matrix A: expressing it as a product of an orthogonal matrix Q and an upper triangular matrix R.

The least squares estimate of the data points we arrived at last time was $Q\beta \approx f$, with β solved as $\beta = Q^T f$. Now we can write

$$\begin{aligned} Q\beta &= QRR^{-1}\beta \\ &= AR^{-1}\beta \\ &= A\alpha \end{aligned}$$

In other words, the solution α in terms of the original columns of A is $\alpha = R^{-1}\beta$, i.e. the solution of the system $R\alpha = \beta$. Solving with an upper triangular matrix is simple, so we are done.

Just as translating row-reduction to the LU matrix factorization opened up new possibilities for algorithms, writing Gram-Schmidt as a QR factorization leads to better alternatives.

1.1 Modified Gram-Schmidt

The classical Gram-Schmidt process relies on previous q vectors being perfectly orthogonal: if they weren't, subtracting off the components of a_i in all the previous directions would not generally result in a vector that's orthogonal to any of them. Unfortunately, with floating-point arithmetic we can't expect perfect orthogonality of the previous vectors, and sure enough, it is well documented that classical Gram-Schmidt does a very bad job of producing an orthogonal Q. The resulting loss of accuracy in the least square solve at least partly defeats the point of doing something other than the normal equations.

However, a very simple modification can make Gram-Schmidt useful, giving an algorithm creatively named **Modified Gram-Schmidt** (MGS). Rather than compute all the coefficients $(q_j \cdot a_i)$ to or-

thogonalize a column a_i simultaneously, we compute them one by one, updating the vector as we go:

$$\begin{aligned}
\tilde{q}_i^{(0)} &= a_i \\
\tilde{q}_i^{(1)} &= \tilde{q}_i^{(0)} - q_1(q_1 \cdot \tilde{q}_i^{(0)}) \\
\tilde{q}_i^{(2)} &= \tilde{q}_i^{(1)} - q_2(q_2 \cdot \tilde{q}_i^{(1)}) \\
&\vdots \\
\tilde{q}_i^{(i-1)} &= \tilde{q}_i^{(i-2)} - q_{i-1}(q_{i-1} \cdot \tilde{q}_i^{(i-2)}) \\
q_i &= \frac{\tilde{q}_i^{(i-1)}}{\|\tilde{q}_i^{(i-1)}\|}
\end{aligned}$$

In this way, q_i can be guaranteed to be orthogonal to q_{i-1} up to machine precision, and in general leads to a much more orthogonal Q and more accurate solve.

1.2 Householder QR

However, there are even more accurate algorithms available. MGS can be viewed as applying a sequence of elementary column operations to A to make it orthogonal: $Q = AR^{-1}$. The problem we fought was trying to make the resulting Q accurately orthogonal in floating point arithmetic. A different strategy is to apply a sequence of elementary orthogonal operations to A to make it upper triangular: $R = Q^T A$. The main question is what sort of elementary orthogonal operation can do the job?

Two main orthogonal operations have been considered: **Givens rotations** and **Householder reflections**. We'll skip over the former, which essentially consists of a planar rotation along two chosen axes, and focus on the more popular Householder reflections.

A Householder reflection can be geometrically visualized as picking a plane, characterized by some normal vector v, and reflecting any vector u it is applied to across that plane. That is, we leave the components of u that are orthogonal to v unchanged, but negate the component in the direction of v—or equivalently subtract twice that component off. More formally, the action of the reflection is as follows:

$$Hu = u - 2\frac{v}{\|v\|}\left(\frac{v}{\|v\|} \cdot u\right)$$

We can write this as multiplication with the following matrix H:

$$H = I - 2\frac{vv^T}{v^T v}$$

(A good exercise is to verify multiplying Hu is equivalent to the first formula.)

This Householder matrix H, a rank one update to the identity, is clearly symmetric; it's also easily verified to be orthogonal as expected by multipling $H^T H$ out:

$$\begin{aligned} H^T H &= \left(I - 2\frac{vv^T}{v^T v}\right)\left(I - 2\frac{vv^T}{v^T v}\right) \\ &= I - 2\frac{vv^T}{v^T v} - 2\frac{vv^T}{v^T v} + 4\frac{vv^T vv^T}{(v^T v)(v^T v)} \\ &= I - 4\frac{vv^T}{v^T v} + 4\frac{v(v^T v)v^T}{(v^T v)(v^T v)} \\ &= I \end{aligned}$$

Clearly a product of Householder reflections is also orthogonal.

We use Householder reflections to compute the QR factorization in a sequential process much like row-reduction, constructing Q^T as a product of reflections that combined make Q upper triangular. The first reflection is designed to make the first column of A zero below the diagonal; the second reflection makes the second column of the updated matrix zero below the diagonal and so on.

Let's take a closer look at the first reflection, how to choose v based on the first column x of A so that Hx is zero below the diagonal—i.e. has all zero entries except its first. The action of H is to subtract a multiple of v from the input vector, so if that zeroes out all entries except the first, all but the first entry of v must be proportional to the entries in x. Therefore, we can choose $v = x + \alpha e$, where α is some scalar we need to determine, and e is the first column of the identity matrix: $(1, 0, \ldots, 0)$. Plugging this in, with $x_1 = e^T x$ the first entry of x, gives

$$\begin{aligned} Hx &= x - 2\frac{(x+\alpha e)(x+\alpha e)^T}{(x+\alpha e)^T(x+\alpha e)}x \\ &= x - 2\frac{(x+\alpha e)(x^T x + \alpha x_1)}{x^T x + 2\alpha x_1 + \alpha^2} \\ &= \left[1 - \frac{2(\|x\|^2 + \alpha x_1)}{\|x\|^2 + 2\alpha x_1 + \alpha^2}\right] x - \frac{2\alpha(\|x\|^2 + \alpha x_1)}{\|x\|^2 + 2\alpha x_1 + \alpha^2} e \end{aligned}$$

For this answer to be zero after the first entry, i.e. proportional to e, the coefficient in front of x must be zero:

$$\begin{aligned} 1 - \frac{2(\|x\|^2 + \alpha x_1)}{\|x\|^2 + 2\alpha x_1 + \alpha^2} &= 0 \\ \Leftrightarrow \quad \|x\|^2 + 2\alpha x_1 + \alpha^2 - 2(\|x\|^2 + \alpha x_1) &= 0 \\ \Leftrightarrow \quad \alpha^2 &= \|x\|^2 \\ \Leftrightarrow \quad \alpha &= \pm\|x\| \end{aligned}$$

There are two possible choices for α. This makes sense if you draw what's happening, say in 2D: for a given arbitrary vector x, there are two planes of reflection which map it to be parallel to the first axis.

From the standpoint of reducing cancellation error, it makes sense to choose the sign of α to match the sign of x_1, so that $v = x + \alpha e$ is computed with all significant digits. It's not hard to work out this gives

$$Hx = (-\text{sign}(x_1)\|x\|, 0, \ldots, 0)^T$$

which is the first column of R. The remaining columns of A are also updated with this reflection.

The second reflection can be computed in much the same, considering the second column of the updated matrix. However, we want to make sure that this second reflection doesn't perturb the first column (which is already in upper triangular form), and thus we ensure that the second v has a zero first entry. We build it from the diagonal and lower triangular part of the second column, along with the second column from the identity matrix, but otherwise in the same way as before. Later reflection vectors continue in the same pattern, being zero above the "diagonal" entry.

The final result is a string of Householder reflections $H_k H_{k-1} \cdots H_2 H_1$ that applied to A give R:

$$Q^T = H_k H_{k-1} \cdots H_2 H_1$$

and so

$$Q = H_1 H_2 \cdots H_k$$

Actually multiplying these out to form Q explicitly is, however, fairly expensive. It's preferable just to store the vector for each reflection, and when multiplication by Q or Q^T is needed just apply the reflections in the correct order. In fact, since the nonzero parts of the reflection vectors fit in a lower triangular matrix, it's possible to set up a Householder QR algorithm that runs in-place, overwriting A as it goes (with a little trickery to deal with the diagonal); this is what LAPACK does for least squares problems.

1.3 Connection to Normal Equations

It's instructive to take a look at what the normal equations are in terms of the QR factorization:

$$\begin{aligned} A^T A &= (QR)^T (QR) \\ &= R^T Q^T Q R \\ &= R^T R \end{aligned}$$

(using the fact $Q^T Q = I$.) The last line is the product of a lower triangular matrix R^T with its tranpose, which tells us the R from QR is in fact the transpose of the Cholesky factor of $A^T A$.

Computing R this way (and presumably then $Q = AR^{-1}$) would be madness, since just forming the normal equations is enough to destroy accuracy for tough problems. However, this can be a useful conceptual tool to reason about QR.

1.4 Solving Linear Systems

The QR factorization, particularly when computed with Householder reflections, also provides a particularly robust way of solving regular linear systems of equations: $Ax = b$ is solved as $x = R^{-1}(Q^T b)$. The error bounds are more favourable than LU with pivoting, however, it's about twice as expensive and for this reason is not commonly used for regular linear systems.

2 Weighted Least Squares

In some scenarios, there is more information included in the approximation problem we are solving with least squares. For example, we might know that some data points are more reliable (have less error) than others, or that it's more important for the approximating function to come close to some data points than others. In other words, we may want the solution to force some entries in the residual to be closer to zero than they would be in regular least squares (and then necessarily have other entries be further from zero).

We can do this by introducing non-negative weights $w_i \geq 0$ when measuring the norm of the residual:

$$\min w_1 r_1^2 + \ldots + w_n r_n^2$$

This is **weighted least squares**. We can express it more conveniently by introducing a diagonal weight matrix $W = \mathrm{diag}(w_1, \ldots, w_n)$:

$$W = \begin{pmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_n \end{pmatrix}$$

The weighted residual norm squared is then $r^T W r$.

We can solve this problem with the same approach we took to get the normal equations for plain least squares, differentiating $r^T W r$ with respect to α (where our approximation to the data f is $A\alpha$), and setting it to zero:

$$\begin{aligned} 0 &= \frac{\partial}{\partial \alpha}(f - A\alpha)^T W (f - A\alpha) \\ &= \frac{\partial}{\partial \alpha}\left(f^T W f - 2\alpha^T A^T W f + \alpha^T A^T W A \alpha\right) \\ &= -2A^T W f + 2A^T W A^T \alpha \end{aligned}$$

We then get the weighted normal equations:

$$A^T W A \alpha = A^T W f$$

Since the weights were nonegative, we can take their square roots to get another real diagonal matrix $W^{1/2} = \mathrm{diag}(\sqrt{w_1}, \ldots, \sqrt{w_n})$ satisfying $(W^{1/2})^2 = W$, and rewrite the weighted normal equations as:

$$(W^{1/2}A)^T(W^{1/2}A)\alpha = (W^{1/2}A)^T(W^{1/2}f)$$

These are now the normal equations for a plain least squares problem (and thus are clearly SPD as well), with rescaled matrix $W^{1/2}A$ and data $W^{1/2}f$; we can instead solve it with QR.

3 Moving Least Squares

Least squares provides a great tool for dealing with relatively simple (though noisy) data, where we have a good idea for the space of plausible approximating functions. However, for richer data sets there may not be any "simple" function we could expect to provide a good approximation: for example, a map of the depth of the ocean bottom in a coastal region is probably going to be a lot more complicated than a simple low degree polynomial.

However, as long as a function is smooth, Taylor's theorem tells us it can be well approximated with a low degree polynomial in a small region around a selected point. That polynomial may not be valid further away if the function is complicated, but locally it should be good.

Let's start simple, with a **moving average** (or "windowed" average): one way to estimate the value of a function at any point is simply to take the average of all nearby sample values. Nearby could be defined as any x_i within distance R of x. The name "moving" derives from a mental model of moving the evaluation point x through the domain, with an attached "window" of radius R into the data moving with it, averaging just those data points seen in the window.

Averages are in fact the least squares fit of a constant value to the data: the mean is the value closest to all data points in a least squares sense. By introducing a window to only include nearby data points, we have in fact a crude weighted least squares problem. The weight w_i for data point i is actually a function of the evaluation point x, defined as:

$$w_i(x) = \begin{cases} 1 & : \quad \|x - x_i\| \leq R \\ 0 & : \quad \text{otherwise} \end{cases}$$

The weighted least squares problem is then to estimate $f(x)$, for a given x, as the value that optimizes:

$$f(x) = \arg\min_y \sum_{i=1}^{n} w_i(x) \, (f_i - y)^2$$

The solution to this is just

$$f(x) = \frac{\sum_{i=1}^{n} w_i(x) f_i}{\sum_{i=1}^{n} w_i(x)}$$

which is of course an average over the data points within radius R of x.

The moving average we just computed has a number of problems. One is that it discontinuously jumps between values: as the window moves, a jump will occur when suddenly a data point is no longer included in the average, or a data point is included for the first time. In fact, the function is piecewise constant, since it only changes at these events. This is not a good model of an underlying smooth function.

We can smooth this out by using a smoother set of weight functions $w_i(x)$. Quite often they are chosen to be translated versions of the same radially symmetric kernel function: $w_i(x) = w(\|x - x_i)$. This kernel function should be non-negative (so that weighted least squares makes sense), biggest at zero and tailing off towards infinity (so that nearer data points count more than further ones). A Gaussian e^{-x^2} fits the bill, though similarly shaped splines that hit zero at some finite radius may be more efficient (far data points needn't be included in the calculation). This approximation scheme (weighted averages using a smooth weight function) is sometimes called **Franke approximation**.

Another problem is that constants don't do a very accurate job of approximating smooth functions. Instead, as we suggested above based on Taylor's theorem, we can use a richer space of low degree polynomials—include linear or maybe even quadratic terms in the least squares fit. Using the same smooth weight functions gives a method called **Moving Least Squares** (MLS).