WE'RE HERE! WE'RE QR!

Good morning.

Thank you for the link to the website. I loved it. Loved. I didnt see any bunnies, so maybe I need to update my Flash player or something. Still.

I am actually for other reasons pleased that you shared said site with me. The main Big Trick, or so I am led to believe, for beating numerical dyspepsia when doing linear solving is to avoid calculating matrix inverses. Hence QR factorization. (I should note here that I visited the office of a colleague who actually possesses a brain stem, and she assures me that the silver bullet is in fact SVD). The nutshell of said gizmo is that we write our matrix X as a product QR, where Q has orthonormalcolumns (this is the special sauce) and R is upper triangular (with positive entries on the diagonal).

How does that go? Well, despite there being no bunnies, it's so damn cute you just want to pinch it. Plus, it's beautifully algorithmic. We assume here that the columns of X are linearly independent to begin with; this is going to be true for real-world floating-point data with probability 1.

To Gram the ol' Schmidt – and you can surely skip this little nugget of pedantry if this is a familiar thing to you – recall the dot product of two n-vectors $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$ is $\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + \cdots + v_n w_n$ and that the norm of a vector $\mathbf{v}$ is $\|\mathbf{v}\| = \sqrt{\mathbf{v} \cdot \mathbf{v}}$. Now write $X = [\mathbf{x}_1\, \mathbf{x}_2\, \ldots\, \mathbf{x}_r]$ in columns. Let

$$\begin{aligned} \mathbf{u}_1 &= \frac{\mathbf{x}_1}{\|\mathbf{x}_1\|}, \\ \mathbf{u}_2 &= \frac{\mathbf{x}_2 - (\mathbf{x_2} \cdot \mathbf{u}_1)\mathbf{u}_1}{\|\mathbf{x}_2 - \mathbf{u}_1\|}, \\ \mathbf{u}_3 &= \frac{\mathbf{x}_3 - (\mathbf{x_3} \cdot \mathbf{u}_1)\mathbf{u}_1 - (\mathbf{x_3} \cdot \mathbf{u}_2)\mathbf{u}_2}{\|\mathbf{x}_3 - (\mathbf{x_3} \cdot \mathbf{u}_1)\mathbf{u}_1 - (\mathbf{x_3} \cdot \mathbf{u}_2)\mathbf{u}_2\|}, \\ \textit{etc.} \end{aligned}$$

so that $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r$ are orthonormal – that is, $\mathbf{u}_i \cdot \mathbf{u}_i = 1$ and $\mathbf{u}_i \cdot \mathbf{u}_j = 0$ for $i \neq j$ – and form the matrix $Q = [\mathbf{u}_1\, \mathbf{u}_2\, \ldots\, \mathbf{u}_r]$.

The next bit is "theory," which, as it turns out, doesn't need to be implemented in code as such; I still include it because, well, I'm kind of anal that way, and plus it's cool. At this point the first column of Q spans the same subspace as the first column of X, the first two columns of Q span the same subspace as the first two columns of X, and so on. Since this is the case we can find coefficients a_{ij} such

that

$$\begin{aligned}
\mathbf{x}_1 &= a_{11}\mathbf{u}_1, \\
\mathbf{x}_2 &= a_{12}\mathbf{u}_1 + a_{22}\mathbf{u}_2, \\
&\vdots \\
\mathbf{x}_k &= a_{1k}\mathbf{u}_1 + a_{2k}\mathbf{u}_2 + \cdots + a_{kk}\mathbf{u}_k \\
&\vdots
\end{aligned}$$

where note that for each k the indices of the r's only run up to k, so set $a_{ij} = 0$ whenever $i > j$. There's a little twist here if we want the diagonal entries of R to be positive (which I don't really see why we do, but anyway): If any a_{kk} turns out to be negative, change Q by making $\mathbf{u}_k$ into $-\mathbf{u}_k$; this won't destroy orthogonality, and now the new a_{kk} becomes positive. Now form the matrix

$$R = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1r} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & \dots & arr \end{bmatrix}$$

and note that

$$QR = [\mathbf{u}_1\, \mathbf{u}_2\, \dots\, \mathbf{u}_r] \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1r} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & \dots & a_{rr} \end{bmatrix} = [\mathbf{x}_1\, \mathbf{x}_2\, \dots\, \mathbf{x}_r] = X(!!)$$

But as I mentioned, that little pirouette wasn't computationally necessary, thanks to the following beautiful card trick. Since the columns of Q are orthonormal and since Q^TQ just dots the rows of Q with one another, Q^TQ is the $r \times r$ identity matrix! Since $X = QR$ we get

$$R = (Q^TQ)R = Q^T(QR) = Q^TX;$$

in other words, once we know Q and simply know that R exists, we calculate it as $R = Q^TX$.

Well, and good, but how does this help us with least-squares dealiebobs? With the least-squares equation of the form

$$X\boldsymbol{\beta} = \mathbf{y},$$

recall that the normal equations have the form

$$X^TX\boldsymbol{\beta} = X^T\mathbf{y}.$$

When we factor $X = QR$ as above, the normal equations become

$$R^TQ^TQR\boldsymbol{\beta} = R^TQ^T\mathbf{y};$$

since Q^TQ is the $r \times r$ identity matrix, we get the further simplification

$$R^TR\boldsymbol{\beta} = R^TQ^T\mathbf{y}.$$

Since R is an upper-triangular matrix with nonzero entries on the diagonal, it is nonsingular and therefore so is R^T, so the equations simplify finally to

$$R\boldsymbol{\beta} = Q^T\mathbf{y};$$

this equation we solve not by using inverse matrices but by backsolving, since R is already in beautiful upper-triangular form.

Now, depending on how far down the rabbit-hole one wants to go, one can also asperse the numerical stability of Gram-Schmidt itself and seek out other methods of finding Q. I am less familiar with these other options; the most prominent of them is the so-called Householder transformation, which if I understand correctly involves dressing up like June Cleaver.