Notes for 2016-02-29

Sparse least squares and Q-less QR

Suppose we want to solve a full-rank least squares problem in which A is large and sparse. In principle, we could solve the problem via the normal equations

$$A^T Ax = A^T b,$$

or introduce $A = QR$ and multiply $A^T Ax = R^T Rx = b$ by R^{-T} to find

$$Rx = R^{-T} A^T b = Q^T b.$$

Note that there is a very close relation between these approaches; the matrix R in the QR decomposition is a Cholesky factor of $A^T A$ in the normal equations (possibly scaled by a diagonal matrix with ± 1 on the diagonal). As we have discussed, it may not be advantageous to use the normal equations directly, since forming and factoring $A^T A$ brings in the square of the condition number. On the other hand, in a sparse setting it's not necessarily such a good idea to use the usual QR approach, since even if A and $A^T A$ are sparse, Q in general will not be. Consequently, when A is sparse, we would typically use the following steps[1]

1. Possibly permute the columns of A so that the Cholesky factor of $A^T A$ (or the factor R, which has the same structure) remains sparse. See `help colamd` for an example of a generally good permutation.

2. Compute a "Q-less" QR decomposition, e.g. `R = qr(A,0)` in MATLABwhere A is sparse. This does not compute the (usually very dense) Q factor explicitly. It also does not form $A^T A$ explicitly. If the right hand side b is known initially, the MATLAB `qr` function can compute $Q^T b$ implicitly at the same time it does the QR factorization.

3. Compute $Q^T b$ as $R^{-T}(A^T b)$, since the latter computation involves only a sparse multiply and a sparse triangular solve.

4. Solve $Rx = Q^T b$.

[1]In MATLAB, you can also use backslash to solve a least squares problem, and it will do the right thing if A is sparse.

It's not a bad idea to do iterative refinement after this — see `help qr` in MATLAB:

```
x = R\(R'\(A'*b))
r = b-A*x;
e = R\(R'\(A'*r));
x = x + e;
```

Of course, just as some square sparse systems cannot be re-ordered so that the LU factorization will be sparse, some sparse least squares problems cannot be re-ordered so that the QR factorization will be sparse. Sometimes, these problems can be more effectively treated by iterative methods of the sort we will discuss later in the class.

Weight patiently

So far, we have considered least squares problems involving the standard Euclidean norm in $\mathbb{R}^n$. But while we clearly take advantage of Euclidean structure in thinking about least squares, nothing requires that we look to the *standard* inner product for that structure. For example, consider the problem

$$\text{minimize } \|Ax - b\|_M^2$$

where M is a general symmetric and positive definite matrix. We can approach this problem through a generalization of the normal equations, e.g.

$$A^T M A x = A^T M b.$$

Alternately, we can observe that if $M = R_M^T R_M$ is a Cholesky factorization, then the M-norm least squares problem is equivalent to a least squares problem with respect to the standard inner product:

$$\text{minimize } \|R_M(Ax - b)\|^2.$$

Frequently, we care about *weighted* least squares problems

$$\text{minimize } \sum_{i=1}^m w_i^2 r_i^2$$

where w_i^2 are a set of weights. This may be appropriate if some rows represent measures that are more certain than others; for example, we may try to identify and down-weight outliers that do not fit the general trend of the rest of the data. Alternately, we may incorporate weights if different rows are naturally scaled differently. A weighted least squares problem can be re-phrased as a standard least squares problem:

$$\text{minimize } \|D^{-1}(Ax-b)\|^2$$

where $D = \operatorname{diag}(w_1, \ldots, w_m)$ is a diagonal scaling matrix corresponding to the weights.

Weighted least squares makes a good building block for solving more general fitting problems. The *iteratively reweighted least squares* procedure (IRLS) is used to solve problems of the form

$$\text{minimize } \sum_{i=1}^{m} l(r_i)$$

where l is some positive-definite loss function. If we let $l(r) = r^2$, we get the standard least squares problem; but standard least squares problems are sensitive to outliers, and may not be appropriate when measurement errors come from a non-Gaussian distribution with heavy tails. In some procedures, one tries to identify outliers statically, then downweights them in an ordinary least squares problem. In other procedures, one uses a loss function l that grows less quickly as a function of the argument; for example, $l(r) = |r|$ corresponds to *least absolute deviation* fitting. A popular choice due to Huber combines the least squares and least absolute deviation loss:

$$l(r) = \begin{cases} \frac{r^2}{2\epsilon}, & |r| \leq \epsilon \\ |r| - \frac{\epsilon}{2}, & |r| > \epsilon. \end{cases}$$

There are other choices as well, such as the Tukey biweight function. The idea in IRLS is to approximate the solution to the general loss minimization throguh a sequence of least square problems; at each step, weights $w_j = l'(\hat{r}_j/s)/\hat{r}_j$, where s is a scale parameter and $\hat{r}$ is the residual vector from the previous step. We will return to IRLS later in the class when we talk about iterations for nonlinear least squares problems.

Consider constraint

Weighted least squares is a good way to construct fits in which some equations are more important than others. But what if we want to enforce some equations *exactly*? That is, we consider the *linearly constrained least squares problem*

$$\text{minimize } \|Ax - b\|^2 \text{ s.t. } Bx = c$$

where $B \in \mathbb{R}^{l \times n}$ with $l < n$.

There are several approaches to solving this problem. One in particular relates to the picture we saw when discussing the QR approach to solving least squares problems. We start by finding some solution y to the constraint equation $By = c$, and form a *homogeneous* problem in terms of a correction $z = x - y$

$$\text{minimize } \|Az - s\|^2 \text{ s.t. } Bz = 0, \quad s = b - Ay.$$

In the usual QR approach, we consider an orthogonal transformation to the residual space such that part of the residual is independent of x and part of the residual can be set to zero. In the constrained setting, we also want to transform the space where the variable lives. That is, suppose we write

$$B^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} R_B \\ 0 \end{bmatrix}$$

Then the condition $Bz = 0$ is equivalent to the condition that $z = U_1 w$. Therefore, we want to solve the problem

$$\text{minimize } \|AU_1 w - s\|^2.$$

This method of handling constraints is fine in the dense case. In the sparse case, we might prefer an alternate approach based on the method of *Lagrange multipliers*, which we will return to in more detail later in the semester. In this method, we want to find a stationary point for the *augmented Lagrangian*

$$L(x, \lambda) = \frac{1}{2}\|Ax - b\|^2 + \lambda^T(Bx - c).$$

Differentiating with respect to x and λ, we have

$$\begin{bmatrix} \delta x \\ \delta \lambda \end{bmatrix}^T \begin{bmatrix} A^T Ax + B^T \lambda - A^T b \\ Bx - c \end{bmatrix} = 0,$$

i.e.

$$\begin{bmatrix} A^T A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} A^T b \\ c \end{bmatrix}.$$

Applying block Gaussian elimination, we have

$$B(A^T A)^{-1} B^T \lambda = B(A^T A)^{-1} A^T b - c.$$

Using the economy factorization $A = QR$, and introducing $M = BR^{-1}$, we have

$$(M^T M)\lambda = MQ^T b - c.$$

From there, we can solve for λ and then do some algebra, which leads to the following MATLAB code:

```
% Factorizations independent of RHS
[Q,R] = qr(A, 0);
M = B/R;
[Q2,T] = qr(M,0);

% Part that depends on the RHS
x0 = R\(Q'*b);              % Unconstrained solve
lambda = T\(T'\(B*x0-c)); % Compute multipliers
x = x0-R\(M'*lambda); % Correction to unconstrained solve
```

Problems to ponder

1. Derive the generalized normal equations for minimizing $\|Ax - b\|_M^2$.
2. Write a MATLAB code to solve the linearly constrained least squares problem using the first approach described (QR decomposition of the constraint matrix).
3. Complete the algebra to get from the extended system formulation for the linearly constrained least squares problem to the MATLAB code at the end of the notes.