Solving Linear Systems and Computational Matrix Algebra

Want to solve the system of linear equations

$$Ax = b$$

This occurs many places in Statistics

- Linear regression: $X^T X\hat{\beta} = X^T y$
- Non-linear least squares, Generalized Linear Models, iteratively reweighted least squares.
- Optimization and root finding

  Newton-Raphson in the multivariate setting has the form

$$x_n = x_{n-1} - D^{-1}(x_{n-1})g(x_{n-1})$$

While the last case doesn't look like it, it fits into this problem since $D^{-1}(x_{n-1})g(x_{n-1})$ is the solution to $D(x_{n-1})y = g(x_{n-1})$.

Now, assuming that everything is nice (things are full rank, A is invertible, etc), the problem is easily solved by finding A^{-1} and calculating

$$x = A^{-1}b$$

However this is usually a suboptimal approach

- It has a higher computational burden
- Numerically less stable – additional computational errors creep in

Instead is usually preferable to solve the system of equations directly.

For example, the code in S-Plus and R for fitting linear models, `lm()`, uses this approach, though their implementations are slightly different.

In S-Plus, you can base the calculations on the QR (default), Choleski, or Singular Value (SVD) decompositions. In R, only the QR approach is available.

More generally, if you want to invert a matrix in S-Plus/R you use the `solve()` command.

In Matlab, to avoid calculating unnecessary determinants, they have created matrix functions / (mldivide) and \ (mrdivide).

`A/B` is equivalent to AB^{-1}

`A\B` is equivalent to $A^{-1}B$

Actually `A/B` is done by `(B'\A')'`.

From the Matlab help page for `inv`, the matrix inversion function:

On a 300 MHz, laptop computer the statements

```
n = 500;
Q = orth(randn(n,n));
d = logspace(0,-10,n);
A = Q*diag(d)*Q';
x = randn(n,1);
b = A*x;
tic, y = inv(A)*b; toc
err = norm(y-x)
res = norm(A*y-b)
```

produce

```
elapsed_time = 1.4320
err = 7.3260e-006
res = 4.7511e-007
```

while the statements

```
tic, z = A\b, toc
err = norm(z-x)
res = norm(A*z-b)
```

produce

```
elapsed_time = 0.6410
err = 7.1209e-006
res = 4.4509e-015
```

A is a matrix with a high condition number (10^{10}), which is expected to have numerical accuracy problems.

The accuracy of both procedures is similar (based on err lines)

`A\b` is much faster. Usually the speed increase is by a factor of 2 to 3.

The errors from a residual point of view (`A*z-b`, where `z` is a solution of `A*x = b`), the residuals based on `A\b` are much smaller.

Wide range of approaches for solving $Ax = b$, some which depend on the form of A.

Often in problems in Statistics, A will be symmetric, as with $X^T X$ or covariance matrices.

This structure can be exploited to achieve more efficient solutions.

LU Decomposition and Gaussian Elimination

Any square nonsingular matrix A can be factored as $LU = A$, where L is lower triangular and U is upper triangular.

The procedures for creating these matrices are equivalent to the Gaussian Elimination procedure taught in every linear/matrix algebra class.

So to solve the system via this decomposition can be done in two steps

1) Solve $Ly = b$ by forward substitution

2) Solve $Ux = y$ by back substitution.

To give an idea of how backward and forward substitution work, let

$$U = \begin{bmatrix} 1 & 2 & 3 \\ & 4 & 5 \\ & & 6 \end{bmatrix}; \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad y = \begin{bmatrix} 11 \\ 14 \\ 12 \end{bmatrix}$$

Then for backward substitution

$$6x_3 = 12 \Rightarrow x_3 = 2$$
$$4x_2 + 5 \times 2 = 14 \Rightarrow x_2 = 1$$
$$1x_1 + 2 \times 1 + 3 \times 2 = 11 \Rightarrow x_1 = 3$$

Forward substitution is similar.

S-Plus/R has `forwardsolve` and `backsolve` for dealing with triangular matrices. I haven't been able to find the routines, but they must be there to get L and U.

Now this approach is useful for any square nonsingular matrix, no other assumptions are needed.

To determine L and U, there are recursive formulas for them which just involve simple arithmetic.

For many of the problems we might be interested in, this approach is often of little interest.

Choleski decomposition

Definition: A matrix A is positive definite if $x^T Ax \geq 0$ for all x and equality only holds for the 0 vector.

For example, most variance matrices are positive definite (though some are positive semidefinite). Also $X^T X$ from linear models are positive definite.

Assume that A is symmetric and positive definite.

The Choleski decomposition factors A as

$$A = U^T U$$

where U is upper triangular. U can be determined by a simple recursive formula.

Thus $Ax = b$ can easily be solved by the steps

1) Solve $U^T y = b$ by forward substitution

2) Solve $Ux = y$ by backwards substitution

The same as for the LU decomposition.

One advantage is the computation burden for getting the Choleski decomposition is roughly half that for the LU decomposition.

There are other uses for this decomposition.

- Random number generation

  Draw realizations from $N_p(\mu, \Sigma)$ where $\Sigma = U^T U$.

  Assuming that you can draw $Z \sim N_p(0, I)$.

  Then $U^T Z \sim N_p(\mu, \Sigma)$.

Note: U can be any matrix satisfying $\Sigma = U^T U$, but the Choleski decomposition is often a good way of getting it.

U is sometimes referred to as a matrix square root. Note it is not unique, unless A is diagonal.

- Variance calculations

  Often variance = $b^T A^{-1} b$.

  e.g., A is the information matrix associated with $\hat{\theta}$ and your interested in $b^T \hat{\theta}$.

  Or as in the regression

  $$\text{Var}(c^T \hat{\beta}) = \sigma^2 c^T (X^T X)^{-1} c.$$

  $b^T A^{-1} b$ can be determined by

  1) Solve $U^T d = b$ by forward substitution

  2) $b^T A^{-1} b = d^T d$

Sweep Operator

An approach for least squares problems.

Allows for simultaneous computation of $\hat{\beta}$, Var($\hat{\beta}$), and SSE = $(y - \hat{y})^T (y - \hat{y})$.

It is also useful for dealing with calculations involving conditional Normal distributions.

Suppose $A = (a_{ij})$ is an $m \times m$ symmetric matrix.

Sweeping on the k^{th} diagonal entry $a_{kk} \neq 0$ of A yields a new symmetric matrix $\hat{A} = (\hat{a}_{ij})$ with entries

$$\hat{a}_{kk} = -\frac{1}{a_{kk}}$$

$$\hat{a}_{ik} = \frac{a_{ik}}{a_{kk}}$$

$$\hat{a}_{kj} = \frac{a_{kj}}{a_{kk}}$$

$$\hat{a}_{ij} = a_{ij} - \frac{a_{ik} a_{kj}}{a_{kk}}$$

for $i \neq k$ and $j \neq k$.

This action can be undone with inverse sweeping on the k^{th} diagonal entry. Inverse sweeping yields a matrix $\breve{A} = (\breve{a}_{ij})$ with entries

$$\breve{a}_{kk} = -\frac{1}{a_{kk}}$$

$$\breve{a}_{ik} = -\frac{a_{ik}}{a_{kk}}$$

$$\breve{a}_{kj} = -\frac{a_{kj}}{a_{kk}}$$

$$\breve{a}_{ij} = a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}}$$

If all m diagonal entries are swept, the final result is related to A^{-1}.

Both sweeping and inverse sweeping preserve symmetry, so the calculations only need to be done for either the upper or lower triangular part of A.

Note: The sweep operator is closely tied to Gaussian elimination. Actually it is a symmetrized version of Gauss-Jordan pivoting.

Why is the sweep operator interesting?

Proposition 7.5.2

Let the symmetric matrix A be partitioned as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

If possible, sweeping on the diagonal entries of A_{11} yields

$$\hat{A} = \begin{bmatrix} -A_{11}^{-1} & A_{11}^{-1}A_{12} \\ A_{21}A_{11}^{-1} & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{bmatrix}$$

So the sweeping rule on blocks conforms to the same rules as sweeping on the matrix entry by entry.

Also, if you can sweep on the diagonal entries in more than one order, you get the same answer.

Proof: See Lange, page 83.

Let X be multivariate normal and partition X and its mean and variance so that

$$X = \begin{pmatrix} Y \\ Z \end{pmatrix}; \quad \mu = \begin{pmatrix} \mu_Y \\ \mu_Z \end{pmatrix}; \quad \Sigma = \begin{bmatrix} \Sigma_Y & \Sigma_{YZ} \\ \Sigma_{ZY} & \Sigma_Z \end{bmatrix}$$

Lets sweep the Σ_Y part of Σ. The result based on the above is

$$\hat{\Sigma} = \begin{bmatrix} -\Sigma_Y^{-1} & \Sigma_Y^{-1}\Sigma_{YZ} \\ \Sigma_{ZY}\Sigma_Y^{-1} & \Sigma_Z - \Sigma_{ZY}\Sigma_Y^{-1}\Sigma_{YZ} \end{bmatrix}$$

The conditional distribution of $Z \mid Y$ has

$$E[Z \mid Y = y] = \mu_Z + \Sigma_{ZY}\Sigma_Y^{-1}(y - \mu_Y)$$
$$\mathrm{Var}(Z \mid Y = y) = \Sigma_Z - \Sigma_{ZY}\Sigma_Y^{-1}\Sigma_{YZ}$$

Exactly the quantities we need come from the sweep operator.

Sweeping directly on

$$\begin{bmatrix} \Sigma_Y & \Sigma_{YZ} & \mu_Y - y \\ \Sigma_{ZY} & \Sigma_Z & \mu_Z \\ (\mu_Y - y)^T & \mu_Z^T & 0 \end{bmatrix}$$

will give the mean and variance immediately.

It also gives us what we need in regression.

Sweeping the upper block of

$$\begin{bmatrix} X^T X & X^T y \\ y^T X & y^T y \end{bmatrix}$$

yields

$$\begin{bmatrix} -(X^TX)^{-1} & (X^TX)^{-1}X^Ty \\ y^TX(X^TX)^{-1} & y^Ty - y^TX(X^TX)^{-1}X^Ty \end{bmatrix}$$

A well known result is that

$$\begin{aligned} \text{SSE} &= (y-\hat{y})^T(y-\hat{y}) \\ &= y^Ty - y^TX(X^TX)^{-1}X^Ty \end{aligned}$$

Since the sweep operator is invertible, it makes it easy to update the fit when a covariate is added or dropped.

While the sweep algorithm is useful, it is not used as much as it used to be for regression problems.

More popular today are methods that factor X directly instead of X^TX.

QR decomposition

Let Q be an $n \times n$ orthogonal matrix. Then is can be shown that

$$\left\| Q^Ty - Q^TX\beta \right\|_2^2 = \left\| y - X\beta \right\|_2^2$$

where $\|x\|_2^2 = x^Tx$.

So minimizing either formula will give the least squares estimate.

Suppose that a Q can be found such that

$$Q^T X = \begin{bmatrix} R_{p\times p} \\ \mathbf{0}_{(n-p)\times p} \end{bmatrix}$$

where R is upper triangular and $\mathbf{0}$ is a matrix of all zeros.

Partition $Q = (Q_1, Q_2)$ where Q_1 contains the first p columns of Q. Then

$$\begin{aligned} \left\| Q^T y - Q^T X\beta \right\|_2^2 &= \left\| \begin{pmatrix} Q_1^T y - R\beta \\ Q_2^T y \end{pmatrix} \right\|_2^2 \\ &= \left\| Q_1^T y - R\beta \right\|_2^2 + \left\| Q_2^T y \right\|_2^2 \end{aligned}$$

and $\left\| Q_1^T y - R\beta \right\|_2^2$ is minimized by $\hat{\beta} = R^{-1} Q_1^T y$ and thus is the least squares estimator.

Note that $R^{-1} Q_1^T = (X^T X)^{-1} X^T$.

One way the matrix Q can be determined with the use of Householder transformations. These applying each of these transformations to y can be used to get $Q_1^T y$.

Then the least squares estimate is determined by solving $R\beta = Q_1^T y$ by backward substitution.

This is the default approach in S-Plus for `lm()`, the approach for `lm()` in R, and the approach for `regress` in Matlab.

I suspect other Statistics packages also use this approach for reasons that I will show in a few minutes.

Aside: The QR decomposition can be used for solving general systems of equations.

The system $Ax = b$ can be transformed to $Rx = Q^T b$.

I believe this approach is the default for `solve()` is S-Plus, and the only approach used in R.

Advantages of the QR approach in regression

Estimation of σ^2

The usual estimate σ^2 in regression is

$$\hat{\sigma}^2 = \frac{\|y - X\beta\|_2^2}{n - p}$$

As mentioned before, we can calculate the numerator by

$$\left\| y - X\hat{\beta} \right\|_2^2 = \left\| Q^T y - Q^T X\hat{\beta} \right\|_2^2$$

Now for the least squares estimator, this is just

$$\left\| Q^T y - Q^T X\hat{\beta} \right\|_2^2 = \left\| Q_2^T y \right\|_2^2$$

which is just the sums of squares of the last $n - p$ elements of $Q^T y$.

Aside: I believe that this underlines a popular proof that in regression SSE ~ χ^2_{n-p}.

In addition, $\text{Var}(a^T\hat{\beta}) = \sigma^2 a^T (X^T X)^{-1} a$.

Since $X = Q_1 R$, $X^T X = R^T Q_1^T Q_1 R = R^T R$.

($Q_1^T Q_1 = I_{pxp}$ since $Q_1^T Q_1$ is the upper $p \times p$ block of $Q^T Q$ which is the identity matrix)

So $a^T (X^T X)^{-1} a = a^T (R^T R)^{-1} a = \left\| (R^T)^{-1} a \right\|_2^2$.

This can be easily calculated by a backward substitution followed by summing the squared terms.

Also important in regression is the "hat" matrix

$$H = X(X^T X)^{-1} X^T,$$

which is involved with

- fitted values: $\hat{y} = Hy$
- residuals: $e = (I - H)y$
- variances of fits: $\text{Var}(\hat{y}) = \sigma^2 H$
- variance of residuals: $\text{Var}(e) = \sigma^2 (I - H)$
- influence statistics (leverages, deleted residuals, changes in fits, Cook's distance, etc)

This matrix can be calculated by $H = Q_1 Q_1^T$.

Sequential Sums of Squares

Most statistics packages, as part of their regression output, include sequential sums of squares as part of there output. For example, from R

```
> anova(swiss.lm)

Analysis of Variance Table
Response: Infant.Mortality
             Df  Sum Sq Mean Sq F value   Pr(>F)
Education     1   3.850   3.850  0.5585 0.458920
Agriculture   1  10.215  10.215  1.4820 0.230103
Fertility     1  79.804  79.804 11.5780 0.001454 **
Residuals    43 296.386   6.893
```

These can also be determined easily as $Q^T = H_p H_{p-1} \ldots H_2 H_1$, where H_j is the Householder transformation based on the j^{th} column of X, so.

$$Q^T y = H_p H_{p-1} \ldots H_2 H_1 y,$$

If we only use the first j,

$$y^{(j)} = H_j H_{j-1} \ldots H_2 H_1 y,$$

this will give us the information on the first using only the first j predictors and.

$$SSE(\beta_1, \ldots, \beta_j) = \sum_{i=j+1}^{n} \left(y_i^{(j)} \right)^2$$

$$SSE(\beta_j \mid \beta_1, \ldots, \beta_{j-1}) = \left(y_j^{(j)} \right)^2$$

Another situation where this approach can be useful for model building, in particular, forward stepwise procedures.

Assume that you have j variables in the model

Interested in which of the remaining $p - j$ variables should be added next.

For simplicity assume that variables 1 through j are already in the model.

Let $H_k^{(j)}$ be the Householder transformation for column k after columns 1 through j have been performed.

Then $y^{(k,j)} = H_k^{(j)} y^{(j)}$ contains the residual information when variable k is added to the model.

$\left(y_{j+1}^{(k,j)}\right)^2$ measures the improvement in the fit by adding variable k to the model.

Stepwise deletion can be also be done through the use of Householder transformations, but it's not as efficient and usually isn't done that way.

Computational efficiency of QR

QR is not the most computationally efficient algorithm.

- The LU decomposition to solve $Ax = b$ needs about half as many operations as QR.
- Using Choleski to solve the normal equations in least squares again requires about half the computions as QR.

However QR is often preferred for its numerical stability, not it efficiency. Also for the regression case, it brings a lot more along as well.

Additional reference:

Gray R (2003). Bio 248: Advanced Statistical Computing - Course Notes. These are the lecture notes for a course offered by Robert Gray in the Harvard School of Public Health.

These are available on the course web site on the Books and Articles page of the References section of the site.