Technical Vignette 6: Solving systems of equations, generating multivariate normal draws, and inverting matrices efficiently in R

Christopher Paciorek, Department of Statistics, University of California, Berkeley, and Department of Biostatistics, Harvard School of Public Health

Version 1.0: January 2012

Computing solutions to systems of linear equations is a common component of many statistical calculations. Often, one has a positive definite matrix, Q, and needs to calculate $Q^{-1}b$. The best way to do this to make use of the Cholesky decomposition: $Q = LL^{\top} = U^{\top}U$, where U is upper triangular. Note that what follows is just as relevant for matrix b as for vector b.

The approach involves several tricks for reducing computation. First, working with the Cholesky is more efficient than using the LU decomposition, which is what is used by R's `solve()` function. Second, we do the Cholesky once, saving it if multiple solutions that use the same Q matrix are needed. Finally, we try to do as few backsolves and forwardsolves as possible.

Backsolve and forwardsolve

A quick note on backsolves and forwardsolves. If you have a lower-triangular L, a forwardsolve is the calculation $L^{-1}b$. A backsolve is the calculation $U^{-1}b$ for upper-triangular U.

In R, we can do either type of solve with either L or U without an explicit transpose:
$U^{-1}b$: `backsolve(U, b)`
$L^{-1}b = U^{\top -1}b$: `backsolve(U, transpose = TRUE)`
$L^{-1}b$: `forwardsolve(L, b)`
$U^{-1}b = L^{\top -1}b$: `forwardsolve(L, b, transpose = TRUE)`

Working with dense Q

First, find the Cholesky decomposition of the dense positive definite Q:

```
> U = chol(Q)
```

- To calculate $Q^{-1}b$ do
```
  > backsolve(U, backsolve(U, b, transpose = TRUE)
```
- To calculate the quadratic form, $b^{\top}Q^{-1}b$, do
```
  > vec = backsolve(U, b, transpose = TRUE)
  > sum(vec^2)
```

- To calculate $y = Lb = U^\top b$ (which gives $\mathrm{Cov}(y) = Q$), do
  ```
  > crossprod(U, b)
  ```
- To calculate $y = L^{\top -1} b = U^{-1} b$ (which gives $\mathrm{Cov}(y) = Q^{-1}$), do
  ```
  > backsolve(U, b)
  ```

Working with sparse Q

Here I recommend Reinhard Furrer's *spam* package. First, find the Cholesky decomposition of the dense positive definite Q:

```
  > library(spam)
  > Q = as.spam(Q)
  > U = chol.spam(Q)
```

Note that this produces a Cholesky of a reordered Q (see notes below for some consequences of this). Also note that if you've previously calculated the Cholesky of a matrix with the same sparsity pattern as Q (i.e., the same pattern of where the zeros are), a faster way to do the Cholesky is:

```
  > U = update.spam.chol.NgPeyton(R, Q) # R should be the (spam type)
Cholesky of the matrix with the same sparsity pattern as Q; if Q and
U are just being updated, then you would use U in place of R
```

- To calculate $Q^{-1}b$ do
  ```
  > backsolve(U, forwardsolve(U, b))
  ```
- To calculate the quadratic form, $b^\top Q^{-1} b$, do
  ```
  > vec = forwardsolve(U, b)
  > sum(vec^2)
  ```
- To calculate $y = Lb = U^\top b$ (which gives $\mathrm{Cov}(y) = Q$), do
  ```
  > iord = ordering(U, inv = TRUE)
  > (t(U) %*% b)[iord]
  ```
  CAUTION: the sparse Cholesky involves a permutation to improve efficiency, so the result is not the same in any realization as what you get using the approach for the dense matrix above, but the result is the same in distribution.
- To calculate $y = L^{\top -1} = U^{-1} b$ (which gives $\mathrm{Cov}(y) = Q^{-1}$), do
  ```
  > backsolve(U, b)
  ```
  CAUTION: the sparse Cholesky involves a permutation to improve efficiency, so the result is not the same in any realization as what you get using the approach for the dense matrix above, but the result is the same in distribution.

NOTE: In R, take `P=I[ord,]` where `ord=ordering(U, inverse = FALSE)` and `I` is the identity (see help(ordering)). U is actually the Cholesky of $PQP^\top$. `backsolve(U, b)` actually computes $P^\top (P^\top U)^{-1} b$ and `forwardsolve(U, b)` actually computes $P^\top (P^\top U^T)^{-1} b$. Matrix manipulations show that the calculations above are legitimate and give the desired result.

Generating a multivariate random normal draw using the precision matrix

Suppose you want to generate $y \sim \mathcal{N}(Q^{-1}a, Q^{-1})$, as arises in working with Markov random fields, where you are given the precision matrix, Q. Here's an efficient approach if Q is dense.

```
> U=chol(Q)
> b = rnorm(nrow(Q))
> backsolve(U, backsolve(U, a, transpose = TRUE) + b)
```

Here's how if you have sparse Q:

```
> U = chol.spam(Q) # assuming Q is already a spam object
> b = rnorm(nrow(Q))
> backsolve(U, forwardsolve(U, a) + b)
```