sparseMVN: An R Package for Multivariate Normal Functions with Sparse Covariance and Precision Matrices

Michael Braun
Edwin L. Cox School of Business
Southern Methodist University

Abstract

The sparseMVN package exploits sparsity in covariance and precision matrices to speed up multivariate normal simpulation and density computation.

Keywords: multivariate normal, sparse matrices.

Simulation from, and calculating the density of, a multivariate normal (MVN) distribution linear algebra operations, such as multiplying a matrix with a vector or another matrix, decomposing a matrix into factors, and solving a system of linear equations. Without futher constraints on the covariance or precision matrix, the memory requirements and computational complexity of these component steps grow faster than the dimension of the MVN-distributed random variable. Because existing methods of working with the MVN distribution in R treat the covariance matrix as dense, these methods are not practical tools for working with high-dimensional problems.

However, in many cases the covariance or precision matrix is sparse. The **sparseMVN** package supplies functions that exploit that sparsity. The **sparseMVN** package provides functions to sample from a multivariate normal (MVN) distribution, and compute its density, when the covariance or precision matrix is sparse. By exploiting this sparsity, we can reduce the amount of computational resources that are needed for matrix storage, and for linear algebra routines like matrix-vector multiplication, solving linear systems, and computing Cholesky factors. Sparse matrix structures store only the row and column indices, and the values, of the non-zero elements of the matrix. All other elements are assumed to be zero, so they do not need to be stored explicitly. Many linear algebra libraries, such as those called by the **Matrix** package, include routines that are optimized for sparse matrices. These routines are much faster than their dense matrix counterparts because they effectively skip over atomic operations that involve the zeros in the matrices.

1. The multivariate normal distribution

Let $x \in \mathbb{R}^K$ be a realization of random variable $X \sim \mathbf{MVN}(\mu, \Sigma)$, where $\mu \in \mathbb{R}^K$ is a vector, and $\Sigma \in \mathbb{R}^{K \times K}$ is a positive-definite covariance matrix.

The probability density of x is

$$f(x) = (2\pi)^{-\frac{k}{2}} |\Sigma|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(x-\mu)' \Sigma^{-1} (x-\mu)\right] \tag{1}$$

A typical approach to computing this density is

1. compute a lower triagular matrix root L, such that $\Sigma = LL^\top$;
2. solve for z in the triangular linear system $Lz = (x - \mu)$;
3. compute $\log|\Sigma| = 2\sum_{k=1}^{K} L_{kk}$; and
4. compute $\log f(x) = -\frac{k}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}z'z$

The matrix root L could be generated via an eigenvalue, singular value, or Cholesky decomposition. The Cholesky factor of a symmetric, positive definite matrix is the unique factor L for which all of the diagonal elements are positive, so we will maintain usage of that definition.

Simulating a MVN random variable also requires a Cholesky decomposition and a triangular matrix multiplication:

1. compute the Cholesky decomposition L, such that $\Sigma = LL^\top$;
2. generate $z \in \mathbb{R}^K$ by simulating from a standard normal distribution k times;
3. compute $x = Lz + \mu$

We know that x is a sample from $\mathbf{MVN}(\mu, \Sigma)$ because

$$\mathbf{E}(Lz) = 0, \text{ and} \tag{2}$$

$$\mathbf{cov}(Lz) = \mathbf{E}\left(Lzz^\top L^\top\right) = L\,\mathbf{E}\left(zz^\top\right)L^\top = LL^\top = \Sigma \tag{3}$$

The computational complexity is $\mathcal{O}\left(k^3\right)$ for a Cholesky decomposition, and $\mathcal{O}\left(k^2\right)$ for solving the triangular linear system (Golub and Van Loan 1996), so the total computational cost grows cubically with the dimension of the random variable. Additionally, the storage requirement for Σ grows quadratically with k.

The `dmvnorm` and `rmvnorm` functions in the **mvtnorm** package (Genz, Bretz, Miwa, Mi, Leisch, Scheipl, and Hothorn 2012) generally follow these algorithms for density computation and random value simulation, respectively. They are well-established implementations, but are limited in some important ways.

1. Each call to the function involves a new matrix factorization step. This can be costly for applications in which x or μ changes from call to call, but Σ does not.
2. In some applications, the precision matrix Σ^{-1}, and not the covariance matrix Σ, is readily available (e.g., estimating the asymptotic covariance from the inverse of a Hessian of a maximum likelihood estimator). To use the **mvtnorm** functions, Σ^{-1} would first have to be inverted explicitly.

3. The **mvtnorm** functions treat Σ as if it were dense, even if there is a large proportion of structural zeros.

The **sparseMVN** package addresses these limitations.

1. The `rmvn.sparse` and `dmvn.sparse` functions take as their matrix argument a sparse Cholesky decomposition. The user does need to do this explicitly beforehand, but once it is done, it does not have to be done again.
2. Both functions include an argument to identify the sparse Cholesky decomposition as a factor of a covariance matrix (`prec=FALSE`) or precision matrix (`prec=TRUE`).

Algorithms for computing the MVN density, and simulating MVN random variates, require only slight modifications when Σ^{-1} is available instead of Σ. To compute the density:

1. compute the Cholesky decomposition of Σ^{-1}, Λ, such that $\Sigma^{-1} = \Lambda\Lambda^\top$;
2. compute $z = \Lambda^\top (x - \mu)$;
3. compute $\log|\Sigma| = -2\sum_{k=1}^{K} \Lambda_{kk}$; and
4. compute $\log f(x) = -\frac{k}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}z'z$

For simulation,

1. compute the Cholesky decomposition of Σ^{-1}, Λ, such that $\Sigma^{-1} = \Lambda\Lambda^\top$;
2. generate $z \in \mathbb{R}^K$ by simulating from a standard normal distribution k times;
3. solve for x in the triangular linear system $\Lambda^\top (x - \mu) = z$

Even when either Σ or Σ^{-1} are dense, there may be advantages to using **sparseMVN** instead of **mvtnorm**. For example, if the user were starting with a large, dense precision matrix Σ^{-1}, and computing MVN densities repeatedly, it may take less time to compute the Cholesky decomposition of $\Sigma^{-1]}$ once, than to invert it and have the **mvtnorm** functions decompose Σ over and over. Nevertheless, the main purpose of **sparseMVN** is to exploit sparsity in either Σ or Σ^{-1} when it exists.

1.1. Sparsity

By "sparse matrix," we mean a matrix for which relatively few elements are non-zero. For example, consider the Hessian of a log posterior density for a hierarchical model with outcomes that are conditionally independent across N heterogeneous units. Each unit is characterized by a vector $\beta_i \in \mathbb{R}^k$ for $i = 1, \ldots, N$, and depend on p population-level variables. In this model, there are $M = Nk + p$ total variables. The cross partial derivatives between any two elements of β_i, or between any β_i and a population-level parameter, could be non-zero, but the cross-partial derivatives between an element in β_i and an element in any other β_j for $j \neq i$ must always be zero under the conditional independence assumption.

When N is large, the proportion of non-zero elements in the Hessian is small, and thus the Hessian is sparse. For example, if $N = 5,\ k = 2$, and $p = 2$, then there are 12 total variables, and the Hessian will have the following "block-arrow" pattern.

```
# 12 x 12 sparse Matrix of class "lgCMatrix"
#
#  [1,] | | . . . . . . . . | |
#  [2,] | | . . . . . . . . | |
#  [3,] . . | | . . . . . . | |
#  [4,] . . | | . . . . . . | |
#  [5,] . . . . | | . . . . | |
#  [6,] . . . . | | . . . . | |
#  [7,] . . . . . . | | . . | |
#  [8,] . . . . . . | | . . | |
#  [9,] . . . . . . . . | | | |
# [10,] . . . . . . . . | | | |
# [11,] | | | | | | | | | | | |
# [12,] | | | | | | | | | | | |
```

There are 144 elements in this symmetric matrix. If the matrix is stored in the standard **base** R dense format, memory is reserved for all 144 values, even though only 64 values are non-zero, and only 38 values are unique. For larger matrices, the reduction in memory requirements by storing the matrix in a sparse format can be substantion. If $N = 1000$, there are 2002 variables in the problem, and more than 4 million elements in the Hessian. However, only 12004 of those elements are non-zero. If we work with only the lower triangle of the Hessian we only need to work with only 7003 unique values. The dense matrix requires 30.6 Mb of RAM, while a sparse matrix of the *dsCMatrix* class requires only 91.5 Kb.

2. Using the sparseMVN package

The **sparseMVN** package provides `rmvn.sparse` and `dmvn.sparse` as alternatives to `rmvnorm` and `dmvnorm`. The signatures are

```
rmvn.sparse(ndraws, mu, CH, prec=TRUE)
dmvn.sparse(x, mu, CH, prec=TRUE)
```

`mvn.sparse` returns a matrix X with each sample in a row. The mean vector μ has q elements. CH is either a *dCHMsimpl* or *dCHMsuper* object, and is computed using the funcCholesky function in the **Matrix** package. The `prec` argument identifies if CH is the decomposition of either the covariance (Σ, `prec=FALSE`) or precision (Σ^{-1}, `prec=TRUE`) matrix. These functions do require the user to compute the Cholesky decomposition beforehand, but this needs to be done only once (as long as Σ does not change).

More details about the `Cholesky` function are available in the **Matrix** package, but it is a simple function to use. The first argument is a sparse symmetric Matrix stored as a *dsCMatrix* object. As far as we know, there is no particular need to deviate from the defaults of the remaining arguments. If `Cholesky` uses a fill-reducing permutation to compute `CH`, the functions in **sparseMVN** will handle that directly, with no additional user intervention required.

Do not use the `chol` function in base R. `Cholesky` is designed for decomposing *sparse* matrices, which involves permuting rows and columns to maintain sparsity of the factor. The *dCHM-*

simple and *dCHMsuper* objects store this permutation, which `rmvn.sparse` and `dmvn.sparse` need.

2.1. An example

Suppose we want to generate R samples from X, where $X \sim \mathbf{MVN}(\mu, \Sigma)$, and the structure of Σ is defined by our example model. In the following code, we first construct the mean vector μ. Then, we construct a random "block arrow" covariance matrix for a given N, p, and k. We use functions from the **Matrix** package to ensure that Σ is stored as a sparse matrix in compressed format.

```
require(Matrix)
set.seed(123)
N <- 5  ## number of blocks in sparse covariance matrix
p <- 2 ## size of each block
k <- 2  ##
R <- 10

## mean vector
mu <- seq(-3,3,length=k*N+p)

## build random block-arrow covariance/precision matrix for test
Q1 <- tril(kronecker(diag(N), Matrix(seq(0.1,1.1,length=k*k),k,k)))
Q2 <- Matrix(rnorm(N*k*p), p, N*k)
Q3 <- tril(0.2*diag(p))
Sigma <- rBind(cBind(Q1, Matrix(0, N*k, p)), cBind(Q2, Q3))
Sigma <- Matrix::tcrossprod(Sigma)
class(Sigma)

# [1] "dsCMatrix"
# attr(,"package")
# [1] "Matrix"
```

Next, we compute the Cholesky decomposition of Σ using the `Cholesky` function, and call `rmvn.sparse`, noting that `chol.Sigma` is the decomposition of a *covariance* matrix, and not a precision matrix.

```
chol.Sigma <- Matrix::Cholesky(Sigma)  ## creates a dCHMsimpl object
x <- rmvn.sparse(R, mu, chol.Sigma, prec=FALSE)
```

Each row of `x` is a sample, and each column is a variable.

The `dmvn.sparse` function returns the log density for each row. Since we have already computed the Cholesky decomposition for Σ, we do not need to do it again.

```
d <- dmvn.sparse(x, mu, chol.Sigma, prec=FALSE)
str(d)

#  num [1:10] -1.707 -1.823 -0.775 0.332 -4.417 ...
```

Sometimes the precision matrix Σ^{-1} is more readily available than Σ. For example, the negative Hessian of a log posterior density at the posterior mode is the precision of the normal approximation to that density. Let $\Sigma^{-1} = \Lambda\Lambda'$ represent the Cholesky decomposition of Σ^{-1}. To sample x, we sample z as before, solve $\Lambda' x = z$, and then add μ. Since $\mathbf{E}(z) = 0$ and $\mathbf{E}(zz') = I_k$, we have $\mathbf{E}(x) = \mu$, and $cov(xx') = \mathbf{E}\left(\Lambda'^{-1} z z' \Lambda^{-1}\right) = \Lambda'^{-1}\Lambda^{-1} = (\Lambda\Lambda')^{-1} = \Sigma$. Then, if we let $y = \Lambda'(x - \mu)$, the log density is

$$\log f(x) = -\frac{k}{2}\log(2\pi) + |\Lambda| - \frac{1}{2} y'y \tag{4}$$

By setting the argument `prec=TRUE`, `rmvn.sparse` and `dmvn.sparse` will recognize the Cholesky decomposition as being for a *precision* matrix instead of a covariance matrix. Without this option, the user would need to explicitly invert Σ beforehand. Even if Σ^{-1} were sparse, there is no guarantee that Σ would be (and vice versa). `rmvn.sparse` and `dmvn.sparse` let the user work with either the covariance or precision matrix, depending on which is more convenient.

3. Timing

A timing comparison is available as a separate vignette.

4. Other packages for creating and using sparse matrices

4.1. sparseHessianFD

Suppose you have a objective function that has a sparse Hessian (e.g., the log posterior density for a hierarchical model). You have an R function that computes the value of the objective, and another function that computes its gradient. You may also need the Hessian, either for a nonlinear optimization routine, or as the negative precision matrix of an MVN approximation.

It's hard enough to get the gradient, but the derivation of the Hessian might be too tedious or complicated for it to be worthwhile. However, it should not be too hard to identify which elements of the Hessian are non-zero. If you have both the gradient, and the Hessian emphpattern, then you can use the **sparseHessianFD** package (Braun 2015) to estimate the Hessian itself.

The **sparseHessianFD** package estimates the Hessian numerically, but in a way that exploits the fact that the Hessian is sparse, and that the pattern is known. The package contains functions that return the Hessian as a sparse *dgCMatrix*. This object can be coerced into a *dsCMatrix*, which in turn can be used by `rmvn.sparse` and `dmvn.sparse`.

4.2. trustOptim

The **trustOptim** package provides a nonlinear optimization routine that takes the Hessian as a sparse *dgCMatrix* object. This optimizer is useful for unconstrained optimization of a high-dimensional objective function with a sparse Hessian. It uses a trust region algorithm, which may be more stable and robust than line search approaches. Also, it applies a stopping rule based on the norm of the gradient, as opposed to whether the algorithm makes "sufficient

progress." (Many optimizers, especially `optim` in **base** R, stop too early, before the gradient is truly flat).

5. Other

Since a large proportion of elements in the matrix are zero, we need to store only the row and column indices, and the values, of the unique non-zero elements. The efficiency gains in **sparseMVN** come from storing the covariance or precision matrix in a compressed format without explicit zeros, and applying linear algebra routines that are optimized for those sparse matrix structures. The **Matrix** package calls sparse linear algebra routines that are implemented in the **CHOLMOD** library (Chen, Davis, Hager, and Rajamanickam 2008; Davis and Hager 1999, 2009); more information about these routines is available there.

The exact amount of time and memory that are saved by saving the covariance/precision matrix in a sparse format depends on the sparsity pattern. But for the hierarchical model example from earlier in this section, the number of non-zero elements grows only linearly with N. The result is that all of the steps of sampling from an MVN also grow linearly with N. Section 4 of Braun and Damien (2014) explains why this is so.

References

Braun M (2015). "sparseHessianFD: An R package for estimating sparse Hessians." URL http://cran.r-project.org/package=sparseHessianFD.

Braun M, Damien P (2014). "Scalable Rejection Sampling for Bayesian Hierarchical Models." 1401.8236.

Chen Y, Davis TA, Hager WW, Rajamanickam S (2008). "Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate." **35**(3), 1–14. doi:10.1145/1391989.1391995.

Davis TA, Hager WW (1999). "Modifying a Sparse Cholesky Factorization." **20**(3), 606–627. doi:10.1137/S0895479897321076.

Davis TA, Hager WW (2009). "Dynamic Supernodes in Sparse Cholesky Update/Downdate and Triangular Solves." **35**(4), 1–23. doi:10.1145/1462173.1462176.

Genz A, Bretz F, Miwa T, Mi X, Leisch F, Scheipl F, Hothorn T (2012). "mvtnorm: Multivariate Normal and t Distributions."

Golub GH, Van Loan CF (1996). *Matrix Computations*. 3rd edition. Johns Hopkins University Press.

Affiliation:

Michael Braun
Edwin L. Cox School of Business

Southern Methodist University
6212 Bishop Blvd.
Dallas, TX 75275
E-mail: braunm@smu.edu
URL: http://www.smu.edu/Cox/Departments/FacultyDirectory/BraunMichael