

Timing comparison

Michael Braun

2015-02-04

This vignette summarizes run times for sampling from, and computing the log density of, an MVN random variable using functions from the *sparseMVN* package (this one), and the *mvtnorm* package. For all of these tests, the covariance/precision matrices have the “block-arrow” structure of a hierarchical model with p heterogeneous variables across m units, and k population variables. The “using sparseMVN” vignettes describes this sparsity pattern in more detail.

The following function is called for each combination of N , m , k , p and $prec$.

```
require(plyr)
require(dplyr)
require(Matrix)
require(mvtnorm)

compare <- function(D) {

  N <- D$N ## number of draws
  p <- D$p ## heterogeneous variables
  k <- D$k ## population variables
  m <- D$m ## number of agents
  prec <- D$prec

  mu <- rep(0,p*m+k) ## assume mean at origin
  Q1 <- tril(kronecker(Matrix(seq(0.1,p,length=p*p),p,p),diag(m)))
  Q2 <- cBind(Q1,Matrix(0,m*p,k))
  Q3 <- rBind(Q2,cBind(Matrix(rnorm(k*m*p),k,m*p),Diagonal(k)))
  CV.sparse <- tcrossprod(Q3)
  CV.dense <- as(CV.sparse,"matrix") ## dense covariance

  ## creates a dCHMsimpl object
  chol.time <- system.time(chol.CV <- Cholesky(CV.sparse))

  if (prec) {
    solve.time <- system.time(sigma <- solve(CV.dense))
  } else {
    solve.time <- NA
    sigma <- CV.dense
  }

  ## draw random samples using rmvn.sparse
  tr.sp <- system.time(x.sp <- rmvn.sparse(N, mu, chol.CV,prec=prec))

  ## computing log densities using dmvn.sparse
  td.sp <- system.time(d.sp <- dmvn.sparse(x.sp, mu, chol.CV, prec=prec))

  ## computing same log densities using dmvnorm
  td.dens <- system.time(d.dens <- dmvnorm(x.sp,mu,sigma,log=TRUE))
}
```

```

## sampling a comparable matrix using rmvnorm
tr.dens <- system.time(x.dens <- rmvnorm(N, mu, sigma, method="chol"))

res <- data.frame(
  N=N, m=m, p=p, k=k, prec=prec, r.sparse=tr.sp["elapsed"],
  r.dense=tr.dens["elapsed"],
  d.sparse=td.sp["elapsed"],
  d.dense=td.dens["elapsed"],
  chol.time=chol.time["elapsed"],
  solve.time=solve.time["elapsed"],
  density.check=as.logical(all.equal(d.sp, d.dens))
)
return(res)
}

```

We set levels for each factor, and run multiple reps of `compare`, averaging across reps. For each test, we will draw, or compute the density for, N samples. The covariance or precision matrix will have a block-arrow structure, with m blocks. Each block is $p \times p$, and there are k variables in the “arrowhead,” or margin, of the matrix.

Each rep is run in parallel using the *doParallel* package. You may need to change the method of parallelization or the number of cores.

Computation times (in seconds) were generated on a 2013-era Mac Pro, with 12 2.7GHz Xeon E5 processors and 64GB of RAM.

```

require(doParallel)
registerDoParallel(cores=12)
N <- 50 ## number of samples
m <- c(20, 200, 1000, 2000) ## number of heterogeneous units
p <- 3 ## size of each "block"
k <- 3 ## variables in the margin, or "arrowhead"
prec <- c(FALSE, TRUE) ## a covariance (FALSE) or precision (TRUE) matrix?

D <- expand.grid(rep=1:12, N=N, m=m,
                p=p, k=k, prec=prec)
R <- ddply(D, c("rep", "N", "m", "p", "k", "prec"), compare, .parallel=TRUE) %>%
  group_by(N, m, p, k, prec) %>%
  summarize(sample.sparse=mean(r.sparse),
            sample.dense=mean(r.dense),
            density.sparse=mean(d.sparse),
            density.dense=mean(d.dense),
            cholesky=mean(chol.time),
            inverse=mean(solve.time)
  )

```

Now let's put the table together. First, let's compare times for sampling from an MVN, when providing a covariance matrix.

```

require(tidyr)
tmp <- gather(R, val, time, c(sample.sparse, sample.dense, density.sparse, density.dense))
tmp <- separate(tmp, val, c("stat", "method")) %>%
  spread(method, time) %>%
  mutate(nnz=(m+1)*p^2 + 2*k*m*p,

```

```

nels=(m*p+k)^2,
pct.nnz=nnz/nels
)

```

```

require(knitr)
tab.samp <- tmp %>% filter(stat %in% c("sample") & prec==FALSE) %>%
  select(-N) %>%
    select(m, pct.nnz, dense, sparse)
kable(tab.samp)

```

| m | pct.nnz | dense | sparse |
|------|---------|--------|--------|
| 20 | 0.138 | 0.007 | 0.005 |
| 200 | 0.015 | 0.087 | 0.011 |
| 1000 | 0.003 | 4.529 | 0.049 |
| 2000 | 0.001 | 13.168 | 0.081 |

We see that as the percentage of non-zero elements goes down, there is more time savings from using `rmvn.sparse` over `rmvnorm`.

There is a similar pattern when computing the log density.

```

tab.dens <- tmp %>% filter(stat %in% c("density") & prec==FALSE) %>%
  select(-N) %>%
    select(m, pct.nnz, dense, sparse)
kable(tab.dens)

```

| m | pct.nnz | dense | sparse |
|------|---------|-------|--------|
| 20 | 0.138 | 0.001 | 0.005 |
| 200 | 0.015 | 0.082 | 0.033 |
| 1000 | 0.003 | 2.884 | 1.921 |
| 2000 | 0.001 | 7.599 | 6.409 |

These patterns hold if we were to provide the precision matrix instead of the covariance matrix. First, the random sampling times:

```

tab.samp <- tmp %>% filter(stat %in% c("sample") & prec==TRUE) %>%
  select(-N) %>%
    select(m, pct.nnz, dense, sparse)
kable(tab.samp)

```

| m | pct.nnz | dense | sparse |
|------|---------|--------|--------|
| 20 | 0.138 | 0.006 | 0.004 |
| 200 | 0.015 | 0.108 | 0.020 |
| 1000 | 0.003 | 5.252 | 1.326 |
| 2000 | 0.001 | 14.837 | 4.645 |

And now, the log density times:

```

tab.dens <- tmp %>% filter(stat %in% c("density") & prec==TRUE) %>%
  select(-N) %>%
  select(m, pct.nnz, dense, sparse)
kable(tab.dens)

```

| m | pct.nnz | dense | sparse |
|------|---------|--------|--------|
| 20 | 0.138 | 0.002 | 0.005 |
| 200 | 0.015 | 0.083 | 0.006 |
| 1000 | 0.003 | 4.099 | 0.016 |
| 2000 | 0.001 | 11.909 | 0.021 |

The *sparseMVN* functions require a Cholesky decomposition of the covariance/precision matrix. However, the *mvtnorm* functions require a matrix inversion when converting a precision matrix to a covariance matrix. The following table summarizes the times of each of those steps. Of course, if we start with a covariance matrix, no inversion is necessary to use the *mvtnorm* functions.

```

tab2 <- tmp %>% filter(stat=="density") %>%
  select(prec, m, pct.nnz, cholesky, inverse) %>%
  arrange(prec, m)
kable(tab2)

```

| prec | m | pct.nnz | cholesky | inverse |
|-------|------|---------|----------|---------|
| FALSE | 20 | 0.138 | 0.000 | NA |
| FALSE | 200 | 0.015 | 0.000 | NA |
| FALSE | 1000 | 0.003 | 0.003 | NA |
| FALSE | 2000 | 0.001 | 0.006 | NA |
| TRUE | 20 | 0.138 | 0.000 | 0.001 |
| TRUE | 200 | 0.015 | 0.001 | 0.027 |
| TRUE | 1000 | 0.003 | 0.004 | 2.911 |
| TRUE | 2000 | 0.001 | 0.006 | 10.246 |

Thus, we see that the time needed for a sparse Cholesky decomposition is negligible, even for large matrices. However, the time it takes to create a dense covariance matrix by inverting a dense precision matrix can be quite substantial. Thus, if you are starting with a sparse precision matrix, the efficiency gains from using *sparseMVN* over *mvtnorm* also include the avoidance of this additional matrix inversion step.