Revolutions

Daily news about using open source R for big data analysis, predictive modeling, data science, and visualization since 2008

August 04, 2016

Simulating from the Bivariate Normal Distribution in R

by Joseph Rickert

My guess is that a good many statistics students first encounter the bivariate Normal distribution as one or two hastily covered pages in an introductory text book, and then don't think much about it again until someone asks them to generate two random variables with a given correlation structure. Fortunately for R users, a little searching on the internet will turn up several nice tutorials with R code explaining various aspects of the bivariate Normal. For this post, I have gathered together a few examples and tweaked the code a little to make comparisons easier.

Here are five different ways to simulate random samples bivariate Normal distribution with a given mean and covariance matrix.

To set up for the simulations this first block of code defines N, the number of random samples to simulate, the means of the random variables, and and the covariance matrix. It also provides a small function for drawing confidence ellipses on the simulated data.

```
library(mixtools)  #for ellipse

N <- 200 # Number of random samples
set.seed(123)
# Target parameters for univariate normal distributions
rho <- -0.6
mu1 <- 1; s1 <- 2
mu2 <- 1; s2 <- 8

# Parameters for bivariate normal distribution
mu <- c(mu1,mu2) # Mean
sigma <- matrix(c(s1^2, s1*s2*rho, s1*s2*rho, s2^2),
           2) # Covariance matrix

# Function to draw ellipse for bivariate normal data
ellipse_bvn <- function(bvn, alpha){
  Xbar <- apply(bvn,2,mean)
  S <- cov(bvn)
  ellipse(Xbar, S, alpha = alpha, col="red")
}
```

The first method, the way to go if you just want to get on with it, is to use the `mvrnorm()` function from the MASS package.

```
library(MASS)
bvn1 <- mvrnorm(N, mu = mu, Sigma = sigma ) # from MASS package
colnames(bvn1) <- c("bvn1_X1","bvn1_X2")
```

It takes so little code to do the simulation it might be possible to tweet in a homework assignment.

A look at the source code for `mvrnorm()` shows that it uses eignevectors to generate the random samples. The documentation for the function states that this method was selected because it is stabler than the alternative of using a Cholesky decomposition which might be faster.

For the second method, let's go ahead and directly generate generate bivariate Normal random variates with the Cholesky decomposition. Remember that the Cholesky decomposition of sigma (a positive definite matrix) yields a matrix M such that M times its transpose gives sigma back again. Multiplying M by a matrix of standard random Normal variates and adding the desired mean gives a matrix of the desired random samples. A lecture from Colin Rundel covers some of the theory.

```
M <- t(chol(sigma))
# M %*% t(M)
Z <- matrix(rnorm(2*N),2,N) # 2 rows, N/2 columns
bvn2 <- t(M %*% Z) + matrix(rep(mu,N), byrow=TRUE,ncol=2)
colnames(bvn2) <- c("bvn2_X1","bvn2_X2")
```

For the third method we make use of a special property of the bivariate normal that is discussed in almost all of those elementary textbooks. If X1 and X2 are two jointly distributed random variables, then the conditional distribution of X2 given X1 is itself normal with: mean = m2 + r(s2/s1)(X1 - m1) and variance = (1 - r2)s2X2.

Hence, a sample from a bivariate Normal distribution can be simulated by first simulating a point from the marginal distribution of one of the random variables and then simulating from the second random variable conditioned on the first. A brief proof of the underlying theorem is available here.

```
rbvn<-function (n, m1, s1, m2, s2, rho)
  {
     X1 <- rnorm(n, mu1, s1)
     X2 <- rnorm(n, mu2 + (s2/s1) * rho *
           (X1 - mu1), sqrt((1 - rho^2)*s2^2))
     cbind(X1, X2)
  }
```

```
bvn3 <- rbvn(N,mu1,s1,mu2,s2,rho)
colnames(bvn3) <- c("bvn3_X1","bvn3_X2")
```

The fourth method, my favorite, comes from Professor Darren Wiliinson's Gibbs Sampler tutorial. This is a very nice idea; using the familiar bivariate Normal distribution to illustrate the basics of the Gibbs Sampling Algorithm. Note that this looks very much like the previous method, except that now we are alternately sampling from the full conditional distributions.

```
gibbs<-function (n, mu1, s1, mu2, s2, rho)
{
  mat <- matrix(ncol = 2, nrow = n)
  x <- 0
  y <- 0
  mat[1, ] <- c(x, y)
  for (i in 2:n) {
    x <- rnorm(1, mu1 +
          (s1/s2) * rho * (y - mu2), sqrt((1 - rho^2)*s1^2))
    y <- rnorm(1, mu2 +
          (s2/s1) * rho * (x - mu1), sqrt((1 - rho^2)*s2^2))
    mat[i, ] <- c(x, y)
  }
  mat
}
bvn4 <- gibbs(N,mu1,s1,mu2,s2,rho)
colnames(bvn4) <- c("bvn4_X1","bvn4_X2")
```

The fifth and final way uses the `rmvnorm()` function form the mvtnorm package with the singular value decomposition method selected. The functions in this package are overkill for what we are doing here, but mvtnorm is probably the package you would want to use if you are calculating probabilities from high dimensional multivariate distributions. It implements numerical methods for carefully calculating the high dimensional integrals involved that are based on some papers by Professor Alan Genz dating from the early '90s. These methods are briefly explained in the package vignette.

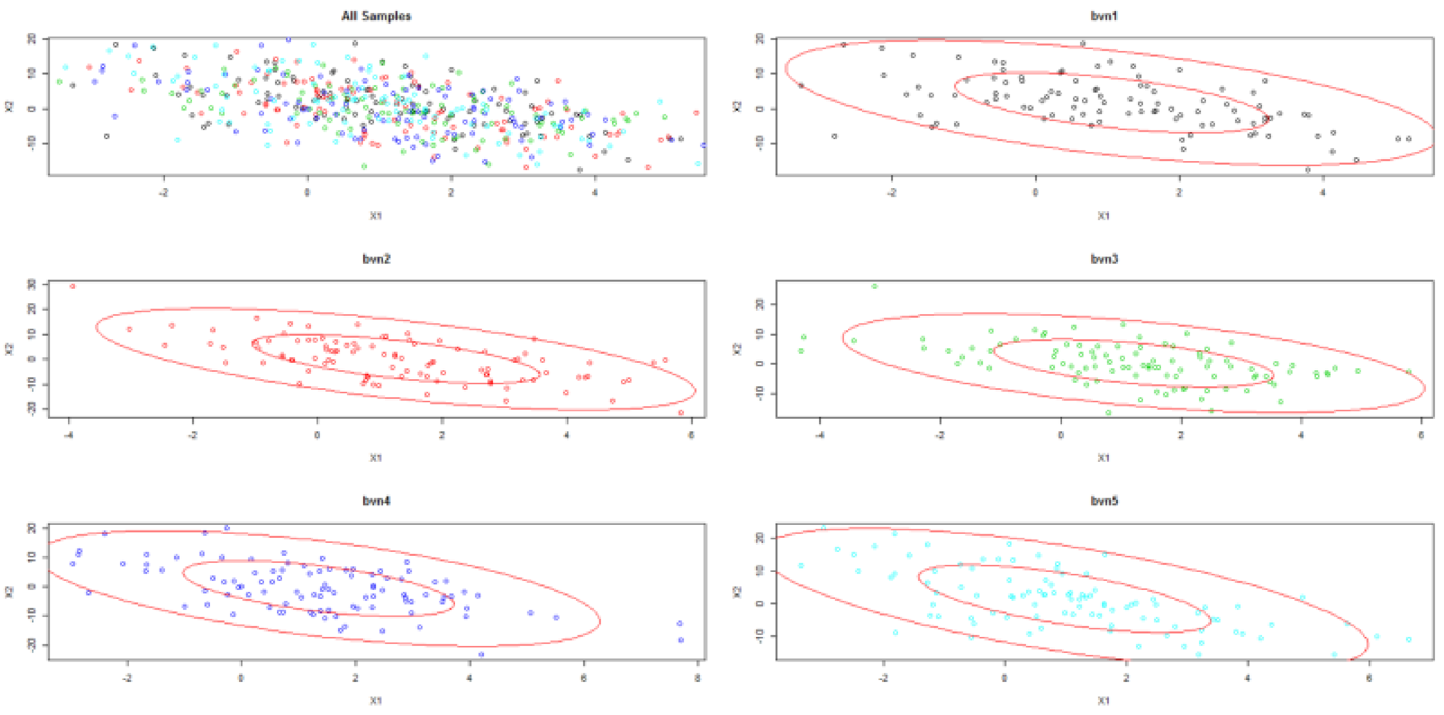
```
library (mvtnorm)
bvn5 <- mvtnorm::rmvnorm(N,mu,sigma, method="svd")
colnames(bvn5) <- c("bvn5_X1","bvn5_X2")
```

Note that I have used the :: operator here to make sure that R uses the `rmvnorm()` function from the mvtnorm package. There is also a `rmvnorm()` function in the mixtools package that I used to get the ellipse function. Loading the packages in the wrong order could lead to the rookie mistake of having the function you want inadvertently overwritten.

Next, we plot the results of drawing just 100 random samples for each method. This allows us to see how the algorithms spread data over the sample space as they are just getting started.

```
bvn <- list(bvn1,bvn2,bvn3,bvn4,bvn5)
```

```
par(mfrow=c(3,2))
plot(bvn1, xlab="X1",ylab="X2",main= "All Samples")
for(i in 2:5){
  points(bvn[[i]],col=i)
}
for(i in 1:5){
  item <- paste("bvn",i,sep="")
  plot(bvn[[i]],xlab="X1",ylab="X2",main=item, col=i)
  ellipse_bvn(bvn[[i]],.5)
  ellipse_bvn(bvn[[i]],.05)
}
par(mfrow=c(1,1))
```



The first plot shows all 500 random samples color coded by the method with which they were generated. The remaining plots show the samples generated by each method. In each of these plots the ellipses mark the 0.5 and 0.95 probability regions, i.e. the area within the ellipses should contain 50% and 95% of the points respectively. Note that bvn4 which uses the Gibbs sampling algorithm looks like all of the rest. In most use cases for the Gibbs it takes the algorithm some time to converge to the target distribution. In our case, we start out with a pretty good guess.

Finally, a word about accuracy: nice coverage of the sample space is not sufficient to produce accurate results. A little experimentation will show that, for all of the methods outlined above, regularly achieving a sample covariance matrix that is close to the target, sigma, requires something on the order of 10,000 samples as is Illustrated below.

```
> sigma
       [,1] [,2]
[1,] 4.0 -9.6
[2,] -9.6 64.0


for(i in 1:5){
  print(round(cov(bvn[[i]]),1))
}

             bvn1_X1 bvn1_X2
bvn1_X1       4.0    -9.5
bvn1_X2      -9.5    63.8

            bvn2_X1 bvn2_X2
bvn2_X1       3.9    -9.5
bvn2_X2      -9.5    64.5

           bvn3_X1 bvn3_X2
bvn3_X1       4.1    -9.8
bvn3_X2      -9.8    63.7

           bvn4_X1 bvn4_X2
bvn4_X1       4.0    -9.7
bvn4_X2      -9.7    64.6

          bvn5_X1 bvn5_X2
bvn5_X1       4.0    -9.6
bvn5_X2      -9.6    65.3
```

Many people coming to R for the first time find it disconcerting to realize that there are several ways to do some fundamental calculation in R. My take is that rather than being a point of frustration, having multiple options indicates that richness of the R language. A close look at the package documentation will often show that yet another method to do something is a response to some subtle need that was not previously addressed. Enjoy the diversity!

Posted by Joseph Rickert at 08:21 in beginner tips, packages, R, statistics | Permalink

Comments

You can follow this conversation by subscribing to the comment feed for this post.

I'm working on a paper surveying this subject. Contact me via Twitter.

Posted by: Driverdaniel67 | August 05, 2016 at 13:38

The comments to this entry are closed.