```c
/* Functions to set GSL rng and generate standard normal random variables */
/* Multithreaded */
/* For use as a shared library in R */
/* Compile with something like: */
/* gcc -fPIC -shared -fopenmp -O3 -march=native gslrand.c -o gslrand.so -lgsl */
/* or */
/* icc -fPIC -shared -openmp -O3 -xHost gslrand.c -o gslrand.so -lgsl */
/* By Adam J. Suarez, Last Edited: 4/3/2015 */

#include <omp.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include <R.h>
#include <Rinternals.h>

const gsl_rng_type *GSL_rng_t;
gsl_rng **GSL_rng;
int GSL_nt;
char L_char='L',N_char='N',R_char='R',T_char='T';

SEXP INIT_GSL_RNG(SEXP SEED){
  int j,seed=asInteger(SEED),i;
  GSL_nt=omp_get_max_threads();

  gsl_rng_env_setup();
  GSL_rng_t = gsl_rng_mt19937;
  GSL_rng = (gsl_rng **) malloc(GSL_nt * sizeof(gsl_rng *));

  omp_set_num_threads(GSL_nt);

#pragma omp parallel for private(i) shared(GSL_rng,GSL_rng_t) schedule(static,1)
  for(j=0;j<GSL_nt;j++){
    i=omp_get_thread_num();
    GSL_rng[i] = gsl_rng_alloc (GSL_rng_t);
    gsl_rng_set(GSL_rng[i],seed+i);
  }

  return R_NilValue;
}

void generate_normal(double *out_v, int n, int nt){
  int j;
#pragma omp parallel for shared(out_v,GSL_rng) num_threads(nt)
   for(j=0;j<n;j++){
       out_v[j] = gsl_ran_gaussian_ziggurat(GSL_rng[omp_get_thread_num()],1);
   }
}

SEXP rnorm_gsl(SEXP N, SEXP NT)
 {
   int n=asInteger(N),nt=asInteger(NT);
   SEXP result = PROTECT(allocVector(REALSXP,n));
   double * out_v = REAL(result);

   generate_normal(out_v,n,nt);

   UNPROTECT(1);
   return result;
 }

SEXP FREE_GSL_RNG(void){
  int j;
  for(j=0;j<GSL_nt;j++) {gsl_rng_free(GSL_rng[j]);}
  return R_NilValue;
}
```



```c
/* Functions to initialize/free CUDA resources and to generate multivariate norm
/* For use as a shared library in R */
/* Compile with something like: */
/* gcc -fPIC -shared -O3 -march=native curand.c -o cudanorm.so -lcudart -lcublas
/* or */
/* icc -fPIC -shared -O3 -xHost curand.c -o cudanorm.so -lcudart -lcublas -lcura
/* By Adam J. Suarez, Last Edited: 4/10/2015 */

#include <cuda.h>
#include <curand.h>
#include <cublas.h>
#include <R.h>
#include <Rinternals.h>

curandGenerator_t CURAND_gen;
cublasHandle_t handle;

SEXP INIT_CURAND_RNG(SEXP SEED){
  curandCreateGenerator(&CURAND_gen, CURAND_RNG_PSEUDO_MTGP32);
  curandSetPseudoRandomGeneratorSeed(CURAND_gen,asInteger(SEED));

  culaInitialize();
  cublasCreate_v2(&handle);

  return R_NilValue;
}

SEXP rmvnorm_cuda(SEXP N, SEXP M, SEXP SIGMA)
{
  size_t n = (size_t) asInteger(N), m=asInteger(M),i;

  double * devData, *dev_sigma;
  cudaMalloc((void **)&devData, n*m*sizeof(double));
  cudaMalloc((void **)&dev_sigma, m*m*sizeof(double));

  SEXP result = PROTECT(allocMatrix(REALSXP,n,m)),SIGMA2 = PROTECT(duplicate(SIG
  double * hostData = REAL(result), * sigma = REAL(SIGMA2),alpha=1.0;

  cudaMemcpy(dev_sigma, sigma, m * m*sizeof(double), cudaMemcpyHostToDevice);

  culaDeviceDpotrf('L',m,dev_sigma,m);

  curandGenerateNormalDouble(CURAND_gen, devData, n*m, 0.0, 1.0);

  cublasDtrmm_v2(handle, CUBLAS_SIDE_RIGHT, CUBLAS_FILL_MODE_LOWER, CUBLAS_OP_T,

  cudaMemcpy(hostData, devData, n * m*sizeof(double), cudaMemcpyDeviceToHost);

  cudaFree(devData);
  cudaFree(dev_sigma);
  UNPROTECT(2);
  return result;
}

SEXP FREE_CURAND_RNG(void){
  cublasDestroy_v2(handle);
  curandDestroyGenerator(CURAND_gen);
  culaShutdown();
  return R_NilValue;
}
```

```
## Program to show the usage of the functions in
## gslrand.c and curand.c
## By Adam J. Suarez, Last Edited: 4/10/2015

## Load shared libraries
dyn.load("gslrand.so")
dyn.load("cudanorm.so")

## Set seed and iniitalize resources
seed <- 1
.Call("INIT_GSL_RNG",seed)
.Call("INIT_CURAND_RNG",seed)

## ## Generate N standard normals using N.cores CPU cores
N <- 10
N.cores <- 1
.Call("rnorm_gsl",N,N.cores)

## ## Make an MxM  covariance matrix
M <- 5
diags <- as.numeric(seq(10,0,length.out = M))
Sigma <- toeplitz(diags)

## ## Generate N samples of M-dimemsional multivariate normals
## ## centered at 0 with covariance Sigma
N <- 10
.Call("rmvnorm_cuda",N,M,Sigma)

## Free resources
## Once you run these, you need to initialize again
## Else you may crash R
.Call("FREE_GSL_RNG")
.Call("FREE_CURAND_RNG")

```



```
> ## Program to show the usage of the functions in
> ## gslrand.c and curand.c
> ## By Adam J. Suarez, Last Edited: 4/10/2015
>
> ## Load shared libraries
> dyn.load("gslrand.so")
> dyn.load("cudanorm.so")
>
> ## Set seed and iniitalize resources
> seed <- 1
> .Call("INIT_GSL_RNG",seed)
NULL
> .Call("INIT_CURAND_RNG",seed)
NULL
>
> ## ## Generate N standard normals using N.cores CPU cores
> N <- 10
> N.cores <- 1
> .Call("rnorm_gsl",N,N.cores)
 [1] -0.4919236382 -1.5460562802  0.0004269484  0.0872218452  0.5142327984
 [6]  0.3651699728  0.0774689638  0.1439099856  0.3195173863 -0.6376160645
>
> ## ## Make an MxM  covariance matrix
> M <- 5
> diags <- as.numeric(seq(10,0,length.out = M))
> Sigma <- toeplitz(diags)
>
> ## ## Generate N samples of M-dimemsional multivariate normals
> ## ## centered at 0 with covariance Sigma
> N <- 10
> .Call("rmvnorm_cuda",N,M,Sigma)
             [,1]       [,2]       [,3]        [,4]       [,5]
 [1,] -1.7304060 -2.3273904 -0.5959233  0.09770317  0.7416234
 [2,] -2.3088785 -1.9422787 -2.4664693 -0.43190943  0.3128057
 [3,]  1.6219573 -1.1736105  2.8222656  1.24104425 -3.5641869
 [4,] -2.3224799 -1.2155825 -1.5161599  2.59440914  5.6963993
 [5,]  2.5883184  0.4255287  0.5497961  2.48933020  1.6423867
 [6,]  1.1899834  2.1250309  4.5491187  2.00538426  4.2117090
 [7,]  0.6364397  1.1423523  2.6739987  4.09681073  3.6542330
 [8,]  2.1666916  0.9757605  2.4092831  1.63436485  1.4712313
 [9,]  0.5349406  3.6933131  7.5955719  8.49207378  6.2229478
[10,]  3.3317695 -1.3361243 -0.9238727  0.51710632 -4.4805197
>
> ## Free resources
> ## Once you run these, you need to initialize again
> ## Else you may crash R
> .Call("FREE_GSL_RNG")
NULL
> .Call("FREE_CURAND_RNG")
NULL
>
>
```