Stat 8931 Spin Glass Homework

Charles J. Geyer

October 12, 2005

1 Introduction

1.1 Model

The *Edwards-Anderson spin glass model* is a spatial lattice process on a square lattice with nearest neighbors. It is like the Ising model explained in the notes but with haphazard coupling constants. Its unnormalized density is

$$h(x) = \exp\left(\frac{1}{2\tau} \sum_{(i,j)\in E} \beta_{ij} x_i x_j\right) \tag{1}$$

where, as in the Ising model, the random variables x_i take values in $\{-1, +1\}$ and the edges $(i, j) \in E$ are nearest neighbors in the lattice. In the Edwards-Anderson model the β_{ij} are themselves random variables, taken to be IID standard normal. However, we are uninterested in the joint distribution of the β_{ij} and the x_i. We are only interested in the conditional distribution of the x_i given the β_{ij}. Thus we will always consider the β_{ij} as fixed known numbers. (The only point of the Gaussian distribution here is to give the β_{ij} haphazard values. Gaussianity itself is uninteresting here.) To finish the model specification we must specify boundary conditions, which we take to be periodic. The parameter τ plays the role of temperature.

1.2 Coupling Coefficients

We need two matrices of coupling coefficients (coupling to neighbor to the right and coupling to the down neighbor). We generate these as follows.

```
> n <- 6
> set.seed(42)
> br <- matrix(rnorm(n^2), n, n)
> bd <- matrix(rnorm(n^2), n, n)
```

To not rely on the R random numbers changing, we write these out to a file and read them back in.

```
> foo <- try(scan("betas.txt"))
> if (inherits(foo, "try-error")) {
+     write(c(br, bd), file = "betas.txt")
+     foo <- scan("betas.txt")
+ }
> n <- sqrt(length(foo)/2)
> n
```

```
[1] 6
```

```
> br <- matrix(foo[1:n^2], n, n)
> bd <- matrix(foo[n^2 + 1:n^2], n, n)
> br
```

```
           [,1]        [,2]       [,3]       [,4]       [,5]       [,6]
[1,]  1.3709580  1.51152200 -1.3888610 -2.4404670  1.8951930  0.4554501
[2,] -0.5646982 -0.09465904 -0.2787888  1.3201130 -0.4304691  0.7048373
[3,]  0.3631284  2.01842400 -0.1333213 -0.3066386 -0.2572694  1.0351040
[4,]  0.6328626 -0.06271410  0.6359504 -1.7813080 -1.7631630 -0.6089264
[5,]  0.4042683  1.30487000 -0.2842529 -0.1719174  0.4600974  0.5049551
[6,] -0.1061245  2.28664500 -2.6564550  1.2146750 -0.6399949 -1.7170090
```

```
> bd
```

```
            [,1]       [,2]       [,3]        [,4]       [,5]        [,6]
[1,] -0.78445900  0.7581632 -0.4314462  0.08976065 -0.3672346  0.33584810
[2,] -0.85090760 -0.7267048  0.6556479  0.27655070  0.1852306  1.03850600
[3,] -2.41420800 -1.3682810  0.3219253  0.67928880  0.5818237  0.92072860
[4,]  0.03612261  0.4328180 -0.7838390  0.08983289  1.3997370  0.72087820
[5,]  0.20599860 -0.8113932  1.5757280 -2.99309000 -0.7272920 -1.04311900
[6,] -0.36105730  1.4441010  0.6428993  0.28488300  1.3025430 -0.09018639
```

The tricky code using the R `try` function is how one does something that may cause an error and respond to the error. Here we write the file only if it does not already exist (more precisely, if it does not already exist or can not be read without error). If the first `scan` command works, then we just use its result.

We will consider these betas fixed throughout the exercise. The temperature parameter τ will be variable. We start with

```
> tau <- 1
```

1.3 Coding Sets

Because we have negative coupling coefficients, the Swendsen-Wang algorithm does not apply, and the only algorithms we know for updating x while preserving this equilibrium distribution are variable-at-a-time Metropolis or Gibbs. It is a well-known trick in lattice processes to update using *coding sets* (introduced by Julian Besag). If we think of our square lattice as colored like a chess board, we see that all neighbors of white nodes are black and vice versa. As mentioned in the notes, the conditional distribution of one variable given the rest depend only on the values of its neighbors. Thus white nodes are conditionally independent given black nodes and vice versa. Hence a block Gibbs (or block Metropolis) sampler that updates using coding sets as blocks can use the coding sets to make the block updates very efficient.

With periodic boundary conditions, we must have n and hence n^2 even in order for coding sets to work properly.

1.4 Data

We take make up data on an $n \times n$ lattice. This means we have n^2 nodes and $2n^2$ edges in the graph. In order to do block updates efficiently in R we precalculate lots of things.

```
> i <- matrix(seq(1, n^2), n, n)
> i
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
```

```
> ir <- cbind(i[, -1], i[, 1])
> ir
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    7   13   19   25   31    1
[2,]    8   14   20   26   32    2
[3,]    9   15   21   27   33    3
```

```
[4,]   10   16   22   28   34    4
[5,]   11   17   23   29   35    5
[6,]   12   18   24   30   36    6

> il <- cbind(i[, n], i[, -n])
> il

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   31    1    7   13   19   25
[2,]   32    2    8   14   20   26
[3,]   33    3    9   15   21   27
[4,]   34    4   10   16   22   28
[5,]   35    5   11   17   23   29
[6,]   36    6   12   18   24   30

> id <- rbind(i[-1, ], i[1, ])
> id

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    2    8   14   20   26   32
[2,]    3    9   15   21   27   33
[3,]    4   10   16   22   28   34
[4,]    5   11   17   23   29   35
[5,]    6   12   18   24   30   36
[6,]    1    7   13   19   25   31

> iu <- rbind(i[n, ], i[-n, ])
> iu

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    6   12   18   24   30   36
[2,]    1    7   13   19   25   31
[3,]    2    8   14   20   26   32
[4,]    3    9   15   21   27   33
[5,]    4   10   16   22   28   34
[6,]    5   11   17   23   29   35

> x <- matrix(1, n, n)
```

If `i` is used as an index into the vector `x` of spin variables, so `x[i]` is just the vector arranged in an array, `x[i]` is the same as `x`, then

- `x[ir]` is the vector of neighbors to the right,
- `x[il]` is the vector of neighbors to the left,
- `x[iu]` is the vector of neighbors upwards, and
- `x[id]` is the vector of neighbors downwards

so the unnormalized log density is

```
sum((x * x[ir] * br + x * x[id] * bd) / tau)
```

and the unnormalized log conditional density of x given the rest is

```
function(x) x * (x[ir] * br + x[id] * bd +
    x[il] * br[il] + x[iu] * br[iu]) / tau
```

or, more precisely, this is the vector of such conditional probabilities, but such a vector can be combined only over a coding set, which is given by

```
> co <- (i + (1 - n%%2) * col(i))%%2
> co
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    0    1    0    1    0    1
[2,]    1    0    1    0    1    0
[3,]    0    1    0    1    0    1
[4,]    1    0    1    0    1    0
[5,]    0    1    0    1    0    1
[6,]    1    0    1    0    1    0
```

```
> ico0 <- i[co == 0]
> ico1 <- i[co == 1]
```

2 Sampler

2.1 Elementary Block Update

Thus we can do two block Gibbs updates, one for each coding set, as follows

```
> block.gibbs <- function(x, tau) {
+     foo <- x[ir] * br + x[id] * bd + x[il] * br[il] + x[iu] *
+         br[iu]
```

```
+     foo <- foo/tau
+     p <- 1/(1 + exp(-2 * foo))
+     x[ico0] <- as.numeric(runif(n^2/2) < p[ico0]) * 2 - 1
+     foo <- x[ir] * br + x[id] * bd + x[il] * br[il] + x[iu] *
+         br[iu]
+     foo <- foo/tau
+     p <- 1/(1 + exp(-2 * foo))
+     x[ico1] <- as.numeric(runif(n^2/2) < p[ico1]) * 2 - 1
+     return(x)
+ }

> print(x)

     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    1    1    1    1    1
[2,]    1    1    1    1    1    1
[3,]    1    1    1    1    1    1
[4,]    1    1    1    1    1    1
[5,]    1    1    1    1    1    1
[6,]    1    1    1    1    1    1
```

2.2 Doing a Run

By symmetry every X_i has mean zero, which also variance one, because the variance is then $E(X_i^2)$ and $X_i^2 = 1$ always.

Since our process is non-homogeneous in the coupling constants every neighbor pair (X_i, X_j) is different. Of the five first and second order moments of the pair, we know four (the means and variances), but each pair has a (possibly) different correlation.

Let us take as the functional of the state of interest the whole state vector (all the X_i) and the products $X_i X_j$ for neighbor pairs. We know the X_i should have mean zero, but this provides a useful convergence check. (All useful convergence checks "cheat" in this way. They use a known property of the equilibrium distribution.)

```
> nbatch <- 100
> blen <- 100

> batch <- matrix(NA, nbatch, 3 * n^2)
> for (ibatch in 1:nbatch) {
+     xbatch <- rep(0, 3 * n^2)
```

```
+     for (iiter in 1:blen) {
+         x <- block.gibbs(x, tau)
+         xbatch <- xbatch + as.numeric(c(x, x * x[ir], x * x[id]))
+     }
+     batch[ibatch, ] <- xbatch/blen
+ }
> mu <- apply(batch, 2, mean)
> mcse <- apply(batch, 2, sd)/sqrt(nbatch)
> xmu <- matrix(mu[1:n^2], n, n)
> xmcse <- matrix(mcse[1:n^2], n, n)
> xxrmu <- matrix(mu[n^2 + 1:n^2], n, n)
> xxrmcse <- matrix(mcse[n^2 + 1:n^2], n, n)
> xxdmu <- matrix(mu[2 * n^2 + 1:n^2], n, n)
> xxdmcse <- matrix(mcse[2 * n^2 + 1:n^2], n, n)

> round(xmu, 3)

       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,]  0.015  0.006  0.015 -0.020  0.023  0.035
[2,]  0.005  0.008 -0.008  0.021  0.023  0.021
[3,]  0.006  0.012  0.009  0.030 -0.031  0.020
[4,] -0.010  0.012  0.018  0.028 -0.027  0.027
[5,]  0.018 -0.004 -0.007 -0.012  0.029  0.039
[6,]  0.050 -0.009 -0.010  0.011  0.039 -0.050

> round(xmcse, 3)

      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,] 0.035 0.044 0.028 0.045 0.044 0.038
[2,] 0.021 0.032 0.026 0.045 0.044 0.025
[3,] 0.017 0.016 0.016 0.035 0.027 0.032
[4,] 0.014 0.015 0.021 0.035 0.037 0.038
[5,] 0.017 0.016 0.029 0.044 0.031 0.025
[6,] 0.034 0.049 0.050 0.050 0.031 0.035

> round(xxrmu, 3)

       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,]  0.786  0.184 -0.480 -0.964  0.768  0.376
[2,]  0.147 -0.213 -0.502  0.938  0.108  0.289
[3,]  0.190  0.553 -0.041 -0.462 -0.428  0.561
```

```
[4,]  0.388 -0.056  0.357 -0.871 -0.873 -0.292
[5,]  0.334  0.543  0.398 -0.229  0.139  0.460
[6,] -0.203  0.954 -0.943  0.306 -0.635 -0.915
```

```
> round(xxrmcse, 3)
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,] 0.006 0.019 0.015 0.002 0.010 0.015
[2,] 0.010 0.015 0.012 0.003 0.018 0.009
[3,] 0.010 0.007 0.015 0.013 0.012 0.007
[4,] 0.010 0.011 0.011 0.005 0.005 0.008
[5,] 0.010 0.008 0.012 0.020 0.019 0.009
[6,] 0.025 0.004 0.003 0.022 0.009 0.004
```

```
> round(xxdmu, 3)
```

```
       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,]  0.551  0.682 -0.812 -0.914  0.960  0.250
[2,] -0.321 -0.231  0.173  0.742 -0.326  0.509
[3,]  0.028  0.437 -0.040  0.204  0.415  0.743
[4,]  0.023 -0.005 -0.500 -0.256 -0.661  0.063
[5,]  0.337  0.073  0.496 -0.870  0.323 -0.699
[6,] -0.058  0.777 -0.063  0.365  0.348 -0.518
```

```
> round(xxdmcse, 3)
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,] 0.010 0.008 0.006 0.004 0.003 0.016
[2,] 0.011 0.013 0.013 0.010 0.017 0.011
[3,] 0.010 0.008 0.011 0.020 0.012 0.007
[4,] 0.010 0.012 0.011 0.024 0.008 0.020
[5,] 0.011 0.016 0.012 0.007 0.017 0.008
[6,] 0.018 0.012 0.021 0.027 0.022 0.019
```

An examination of all 108 autocorrelation plots (one for each coordinate of the vector of batch means) shows that perhaps doubling the batch size is necessary for the mean coordinates (those contributing to `xmu`). We would also like the standard errors to be smaller. Let's redo.

```
> nbatch <- 200
> blen <- 1000
```

```
> batch <- matrix(NA, nbatch, 3 * n^2)
> for (ibatch in 1:nbatch) {
+     xbatch <- rep(0, 3 * n^2)
+     for (iiter in 1:blen) {
+         x <- block.gibbs(x, tau)
+         xbatch <- xbatch + as.numeric(c(x, x * x[ir], x * x[id]))
+     }
+     batch[ibatch, ] <- xbatch/blen
+ }
> mu <- apply(batch, 2, mean)
> mcse <- apply(batch, 2, sd)/sqrt(nbatch)
> xmu <- matrix(mu[1:n^2], n, n)
> xmcse <- matrix(mcse[1:n^2], n, n)
> xxrmu <- matrix(mu[n^2 + 1:n^2], n, n)
> xxrmcse <- matrix(mcse[n^2 + 1:n^2], n, n)
> xxdmu <- matrix(mu[2 * n^2 + 1:n^2], n, n)
> xxdmcse <- matrix(mcse[2 * n^2 + 1:n^2], n, n)
> round(xmu, 3)

       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,]  0.003  0.007 -0.008  0.007 -0.007 -0.006
[2,] -0.001  0.007  0.006 -0.007 -0.007  0.001
[3,]  0.004 -0.006 -0.006 -0.004  0.000  0.005
[4,] -0.006 -0.005  0.002  0.009 -0.009  0.009
[5,] -0.004 -0.001  0.006  0.009  0.007 -0.002
[6,] -0.001  0.011  0.012 -0.012 -0.003  0.001

> round(xmcse, 3)

      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,] 0.009 0.011 0.006 0.011 0.011 0.010
[2,] 0.005 0.009 0.006 0.011 0.011 0.005
[3,] 0.004 0.004 0.004 0.009 0.006 0.007
[4,] 0.003 0.004 0.005 0.008 0.008 0.008
[5,] 0.004 0.004 0.007 0.011 0.007 0.005
[6,] 0.007 0.012 0.012 0.012 0.007 0.007

> round(xxrmu, 3)

       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,]  0.791  0.193 -0.494 -0.967  0.779  0.385
```

```
[2,]  0.160 -0.201 -0.498  0.936  0.127  0.290
[3,]  0.194  0.544 -0.039 -0.494 -0.448  0.559
[4,]  0.391 -0.051  0.350 -0.870 -0.873 -0.288
[5,]  0.321  0.552  0.412 -0.224  0.111  0.438
[6,] -0.191  0.954 -0.940  0.288 -0.615 -0.915
```

```
> round(xxrmcse, 3)
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,] 0.001 0.004 0.004 0.001 0.002 0.005
[2,] 0.002 0.004 0.003 0.001 0.004 0.002
[3,] 0.002 0.002 0.003 0.003 0.003 0.002
[4,] 0.002 0.003 0.002 0.001 0.001 0.002
[5,] 0.002 0.002 0.002 0.005 0.004 0.002
[6,] 0.006 0.001 0.001 0.006 0.003 0.001
```

```
> round(xxdmu, 3)
```

```
       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,]  0.557  0.687 -0.819 -0.917  0.964  0.246
[2,] -0.315 -0.228  0.160  0.750 -0.360  0.528
[3,]  0.032  0.423 -0.028  0.218  0.419  0.744
[4,]  0.011 -0.013 -0.505 -0.260 -0.656  0.041
[5,]  0.325  0.103  0.516 -0.868  0.314 -0.696
[6,] -0.060  0.775 -0.056  0.355  0.371 -0.517
```

```
> round(xxdmcse, 3)
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,] 0.002 0.002 0.001 0.001 0.001 0.004
[2,] 0.003 0.003 0.003 0.002 0.004 0.002
[3,] 0.003 0.002 0.003 0.005 0.003 0.002
[4,] 0.002 0.002 0.002 0.005 0.002 0.004
[5,] 0.002 0.003 0.002 0.002 0.004 0.002
[6,] 0.004 0.003 0.006 0.008 0.005 0.004
```

3 Lower Temperature

So far, so good. But how about if we lower the temperature?

```
> tau <- 0.2
```

```
> nbatch <- 200
> blen <- 1000
> batch <- matrix(NA, nbatch, 3 * n^2)
> for (ibatch in 1:nbatch) {
+     xbatch <- rep(0, 3 * n^2)
+     for (iiter in 1:blen) {
+         x <- block.gibbs(x, tau)
+         xbatch <- xbatch + as.numeric(c(x, x * x[ir], x * x[id]))
+     }
+     batch[ibatch, ] <- xbatch/blen
+ }
> mu <- apply(batch, 2, mean)
> mcse <- apply(batch, 2, sd)/sqrt(nbatch)
> xmu <- matrix(mu[1:n^2], n, n)
> xmcse <- matrix(mcse[1:n^2], n, n)
> xxrmu <- matrix(mu[n^2 + 1:n^2], n, n)
> xxrmcse <- matrix(mcse[n^2 + 1:n^2], n, n)
> xxdmu <- matrix(mu[2 * n^2 + 1:n^2], n, n)
> xxdmcse <- matrix(mcse[2 * n^2 + 1:n^2], n, n)
> round(xmu, 3)

       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,]  1.000  1.000  0.986 -1.000  1.000  1.000
[2,]  0.970  0.978 -0.986  1.000  1.000  0.940
[3,]  0.710 -0.494 -0.494  0.999 -0.954  0.938
[4,] -0.710 -0.494 -0.095  0.937 -0.937  0.937
[5,]  0.982  0.993  0.998  0.999  0.937  1.000
[6,]  1.000  1.000  1.000 -1.000  0.996 -1.000

> round(xmcse, 3)

      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,] 0.000 0.000 0.001 0.000 0.000 0.000
[2,] 0.010 0.001 0.001 0.000 0.000 0.023
[3,] 0.011 0.010 0.010 0.000 0.018 0.024
[4,] 0.011 0.010 0.013 0.024 0.024 0.024
[5,] 0.001 0.000 0.000 0.000 0.024 0.000
[6,] 0.000 0.000 0.000 0.000 0.001 0.000

> round(xxrmu, 3)
```

```
       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,]  1.000  0.986 -0.986 -1.000  1.000  1.000
[2,]  0.948 -0.965 -0.986  1.000  0.940  0.965
[3,] -0.660  0.938 -0.494 -0.955 -0.983  0.717
[4,]  0.725 -0.379 -0.032 -1.000 -1.000 -0.715
[5,]  0.985  0.996  0.997  0.936  0.937  0.982
[6,]  1.000  1.000 -1.000 -0.996 -0.996 -1.000
```

```
> round(xxrmcse, 3)
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,] 0.000 0.001 0.001 0.000 0.000 0.000
[2,] 0.010 0.001 0.001 0.000 0.023 0.012
[3,] 0.009 0.001 0.010 0.017 0.007 0.008
[4,] 0.009 0.015 0.014 0.000 0.000 0.009
[5,] 0.000 0.000 0.000 0.024 0.024 0.001
[6,] 0.000 0.000 0.000 0.001 0.001 0.000
```

```
> round(xxdmu, 3)
```

```
       [,1]   [,2]   [,3]   [,4]   [,5]   [,6]
[1,]  0.970  0.978 -1.000 -1.000  1.000  0.940
[2,]  0.679 -0.515  0.487  0.999 -0.954  0.998
[3,] -0.457  0.938 -0.379  0.938  0.983  0.999
[4,] -0.698 -0.492 -0.097  0.936 -1.000  0.937
[5,]  0.982  0.993  0.998 -0.999  0.940 -1.000
[6,]  1.000  1.000  0.986  1.000  0.996 -1.000
```

```
> round(xxdmcse, 3)
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
[1,] 0.010 0.001 0.000 0.000 0.000 0.023
[2,] 0.021 0.009 0.009 0.000 0.018 0.001
[3,] 0.018 0.001 0.015 0.024 0.006 0.001
[4,] 0.011 0.010 0.013 0.024 0.000 0.024
[5,] 0.001 0.000 0.000 0.000 0.023 0.000
[6,] 0.000 0.000 0.001 0.000 0.001 0.000
```

Now our "convergence diagnostic" (available only because of a known symmetry in the problem) diagnoses complete failure. The `xmu` results are completely wrong. The standard errors for them are also completely wrong (we know the right answer, exactly zero, thus the *correct* MCSE are one in this case).

4 The Assigned Homework

Finally we get to the homework assignment.

1. Figure out what sort of `nbatch` and `blen` this sampler with `tau ==` 0.2 needs to get (a) working with all of the reported MC estimates and MCSE apparently correct and (b) all of the MCSE less than 0.001. Note that you do not actually need to get the MCSE below 0.001. You need to run long enough so that you are getting apparently correct results and can thus infer using the "square root law" how long you would need to run to get all MCSE below 0.001.

2. Implement a parallel tempering sampler with one "helper" chain the same model except for a different `tau`. Make the helper `tau` such that (a) the helper chain mixes better (is that higher temperature or lower?) and (b) the acceptance rate of Metropolis rejection for the swap steps is in the range 20–30%. Then proceed as in part 1 figuring out what sort of `nbatch` and `blen` this sampler needs to get all of the MCSE for the distribution of interest (with `tau ==` 0.2) less than 0.001.