Chapter 10

Batching of MCMC runs

As described in chapter 9, the C program `mcrcm` is given a set of parameters as follows: L, T, interaction type and interaction parameter α, algorithm type (i.e. SO, SAR, band-update, or worm) and number of sweeps. Then a sequence of random permutations is generated, and sample means of random variables are computed over that sequence. The result is, for example, that the system energy H had sample mean 4939.7 with sample standard deviation 155.6.

One wishes, however, to find patterns in such data. In particular, as described in chapter 11: for various interactions α, for larger and larger L permitting extrapolation to the thermodynamic limit, one wishes to estimate the critical temperature T_c at which various order parameters (section 3.7) have a point of non-analyticity in the infinite-volume limit.

The C language was chosen for `mcrcm`, due to its efficiency for large-scale computations. (Data sets discussed in this dissertation have taken approximately 5.5 CPU-years; the choice of C has proved worthwhile, as an early Python implementation ran a factor of 40 times slower.) For the relatively lightweight task of scheduling parallel-processing tasks over multiple parameter values, extracting and collating sample statistics of random variables, and viewing the results — tasks for which the CPU time is measured in minutes, at most — it suffices to use easier-to-code scripting languages such as Bash or Python. Examples are shown in subsequent sections. The author has developed a flexible Python module, `taskutil.py`, for automating most of the tasks described in this chapter. However, such content is non-mathematical, of dubious value to an already lengthy mathematics dissertation. Equivalent, but briefer and simpler, scripting snippets in Bash will be shown instead.

10.1 Collecting data over multiple parameter values

Given a choice of parameter `nacc` (number of sweeps), and choices of parameters L, T, and α as described in chapter 2, a simple example of running `mcrcm` programs to compute sample statistics of random variables is as follows:

```
nacc=100000
datadir=sar_nacc_${nacc}
Ls="40 60 80"
Ts="6.40 6.50 6.60 6.70 6.80 6.90 7.00 7.10 7.20"
alphas="0.000 0.001 0.002"
mkdir -p $datadir
```

```
for L in $Ls; do
    for T in $Ts; do
        for alpha in $alphas; do
            file=$datadir/L_${L}_T_${T}_rell_alpha_${alpha}.txt
            mcrcm L=$L T=$T rell alpha0=$alpha nacc=$nacc > $file
        done
    done
done
```

For example, one of the loop iterations will execute the command

```
mcrcm L=40 T=6.70 rell alpha0=0.001 nacc=100000
```

and direct the output (as described in section 9.18) to the file

```
sar_nacc_100000/L_40_T_6.70_rell_alpha_0.001.txt,
```

the contents of which were shown in section 9.18. In total, 81 such files will be produced. Next one may wish to, say, select out only the values of `mean_fS` — the sample mean of f_S (sections 3.6 and 9.14) — from all 81 files.

The author's `taskutil.py` module implements this basic idea, with various elaborations. For example, one might divide the 81 tasks into 3 tasks for 27 processors each, with processors running in parallel. One might also wish to implement restart logic in case of unexpected downtime (e.g. due to a thunderstorm), wherein the script sees if a given file has already been completed rather than launching an already-completed task.

10.2 Extracting data over multiple parameter values

Given a list of data files as described in the previous section, one may wish to extract out a specified sample statistic for a specified random variable. A simple sample script which does this is

```
RV_name="mean_fS"
nacc=100000
datadir=sar_nacc_${nacc}
Ls="40 60 80"
Ts="6.40 6.50 6.60 6.70 6.80 6.90 7.00 7.10 7.20"
alphas="0.000 0.001 0.002"
for alpha in $alphas; do
    echo "alpha = $alpha"
    for L in $Ls; do
        for T in $Ts; do
            file=$datadir/L_${L}_T_${T}_rell_alpha_${alpha}.txt
```

```
            RV=`grep $RV_name $file | awk '{print $NF}'`
            echo $L $T $RV
        done
    done
    echo ""
done
```

```
alpha = 0.000
40 6.40 1.0506240
40 6.50 0.8573933
...

alpha = 0.1
40 6.40 1.3394773
40 6.50 1.2505133
...
```

Such output may be analyzed or plotted as desired. Examples were shown throughout chapter 9.

10.3 Parallel processing

As discussed in section 10.1, when one is examining a set of (L, T, α) parameter values, one invokes an `mcrcm` executable for each particular triple of (L, T, α). If sufficiently many processors are available, there is no reason one cannot run, say, $(L = 40, T = 6.5, \alpha = 0.1)$ at the same time as $(L = 40, T = 6.5, \alpha = 0.2)$. In high-performance computing jargon, such parallelism is called *trivially parallel* or *embarrassingly parallel*. The author uses three such paradigms:

- On a single-processor laptop, `mcrcm` programs are launched in sequence, as in section 10.1.
- On the University of Arizona Department of Mathematics chivo cluster, which is four hosts with two CPUs each, one might (to be civil to other users) pick three hosts, running one parameter set on each, further prefixing with Unix `nice -10`.
- On the University of Arizona High Performance Computing Center's ICE cluster, which is a single host with over 1000 CPUs shared by dozens of on-campus researchers, one might request, say, 32 CPUs and divide the parameter set among those CPUs.

What has not been implemented is parallelization of `mcrcm` itself. As of this writing, there is no need to do so.