Data Blocking

Jon K. Nilsen

Department of Physics and Scientific Computing Group
University of Oslo, N-0316 Oslo, Norway

Spring 2008

Outline

Data Blocking

- Why blocking?
- What is blocking?
- Blocking in parallel VMC
- Example

Why blocking?

Statistical analysis

- Monte Carlo simulations can be treated as *computer experiments*
- The results can be analysed with the same statistics tools we would use in analysing laboraty experiments
- As in all other experiments, we are looking for expectation values and an estimate of how accurate they are, i.e., the error

Why blocking?

Statistical analysis

- As in other experiments, Monte Carlo experiments have two classes of errors:
  - Statistical errors
  - Systematic errors
- Statistical errors can be estimated using standard tools from statistics
- Systematic errors are method specific and must be treated differently from case to case. (In VMC a common source is the step length)

What is blocking?

Blocking

- Say that we have a set of samples from a Monte Carlo experiment
- Assuming (wrongly) that our samples are uncorrelated our best estimate of the standard deviation of the mean $\bar{m}$ is given by

$$\sigma = \sqrt{\frac{1}{n-1}\left(\bar{m^2} - \bar{m}^2\right)}$$

- If the samples are correlated it can be showed that

$$\sigma = \sqrt{\frac{1+2\tau/\Delta t}{n-1}\left(\bar{m^2} - \bar{m}^2\right)}$$

where τ is the correlation time (the time between a sample and the next uncorrelated sample) and Δt is time between each sample

What is blocking?

Blocking

- If $\Delta t \gg \tau$ our first estimate of σ still holds
- Much more common that $\Delta t < \tau$
- In the method of data blocking we divide the sequence of samples into blocks
- We then take the mean $\bar{m}_i$ of block $i = 1 \ldots n_{blocks}$ to calculate the total mean and variance
- The size of each block must be so large that sample j of block i is not correlated with sample j of block $i + 1$
- The correlation time τ would be a good choice

What is blocking?

Blocking

- Problem: We don't know τ
- Solution: Make a plot of std. dev. as a function of block size
- The estimate of std. dev. of correlated data is too low $\rightarrow$ the error will increase with increasing block size until the blocks are uncorrelated, where we reach a plateau
- When the std. dev. stops increasing the blocks are uncorrelated

Implementation

Main ideas

- Do a parallel Monte Carlo simulation, storing all samples to files (one per process)
- Do the statistical analysis on these files, independently of your Monte Carlo program
- Read the files into an array
- Loop over various block sizes
- For each block size n_b, loop over the array in steps of n_b taking the mean of elements $in_b, \ldots, (i+1)n_b$
- Take the mean and variance of the resulting array
- Write the results for each block size to file for later analysis

Implementation

Example

- The files vmc_para.cpp and vmc_blocking.cpp contains a parallel VMC simulator (see Mortens slides for details) and a program for doing blocking on the samples from the resulting set of files
- Will go through the parts related to blocking

Implementation

Parallel file output

- The total number of samples from all processes may get very large
- Hence, storing all samples on the master node is not a scalable solution
- Instead we store the samples from each process in separate files
- Must make sure this files have different names

String handling

```
ostringstream ost;
ost << "blocks_rank" << my_rank << ".dat";
blockofile.open(ost.str().c_str(), ios::out | ios::
    binary);
```

Implementation

Parallel file output

- Having separated the filenames it's just a matter of taking the samples and store them to file
- Note that there is no need for communication between the processes in this procedure

File dumping

```
all_energies = new double[number_cycles+1];
mc_sampling(max_variations, number_cycles, cumulative_e,
      cumulative_e2,
            all_energies);

blockofile.write((char*)(all_energies+1),
                  number_cycles*sizeof(double));
blockofile.close();
```

Implementation

Reading the files

- Reading the files is only about mirroring the output
- To make life easier for ourselves we find the filesize, and hence the number of samples by using the C function `stat`

File loading

```
struct stat result;
if(stat("blocks_rank0.dat", &result) == 0){
  local_n = result.st_size/sizeof(double);
  n = local_n*n_procs;
}

double* mc_results = new double[n];
for(int i=0; i<n_procs; i++){
  ostringstream ost;
  ost << "blocks_rank" << i << ".dat";
  ifstream infile;
  infile.open(ost.str().c_str(), ios::in | ios::binary);
  infile.read((char*)&(mc_results[i*local_n]),result.st_size);
  infile.close();
}
```

Implementation

Blocking

- Loop over block sizes $in_b, \ldots, (i+1)n_b$

Loop over block sizes

```
for(int i=0; i<n_block_samples; i++){
   block_size = min_block_size+i*block_step_length;
   blocking(mc_results, n, block_size, res);
   mean  = res[0];
   sigma = res[1];
   outfile << block_size << "\t" << mean << "\t"
           << sqrt(sigma/((n/block_size)-1.0))
           << endl;
}
```

Implementation

Blocking

- The blocking itself is now just a matter of finding the number of blocks (note the integer division) and taking the mean of each block
- Note the pointer aritmetic: Adding a number *i* to an array pointer moves the pointer to element *i* in the array

Blocking function

```
void blocking(double *vals, int n_vals, int
    block_size, double *res){
  int n_blocks = n_vals/block_size;
  double* block_vals = new double[n_blocks];
  for(int i=0; i<n_blocks; i++)
    block_vals[i] = mean(vals+i*block_size,
        block_size);
  meanvar(block_vals, n_blocks, res);
}
```