Monte Carlo Methods, Courant Institute, Spring 2013

http://www.math.nyu.edu/faculty/goodman/teaching/MonteCarlo12/

Always check the class message board on the NYU Classes site from home.nyu.edu before doing any work on the assignment.

Assignment 3, due March 26

Corrections: (none yet).

1. (*More practice with Gaussians and linear algebra. This is a piece of analysis that sets up a programming exercise below.*) Consider the linear Gaussian recurrence relation

$$X_{n+1} = AX_n + BZ_n \ , \tag{1}$$

with $X_n \in \mathbb{R}^2$ and $B = diag(1, 2)$, and

$$A = \begin{pmatrix} .7 & .2 \\ .2 & .7 \end{pmatrix} \quad , \quad \text{and} \quad B = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \ .$$

Find the steady state covariance matrix and the eigenfunction, $g(x)$, of the forward operator L^*, and the eigenfunction $v(x)$, of the generator, L, that corresponds to eigenvalue $\lambda = .81 = .9^2$. We saw in class that g is the second derivative of the invariant density in an eigenvector direction. From the definition of generator, v must satisfy

$$E[v(X_{n+1}) \mid X_n = x] = \lambda v(x) \ .$$

You can do this with polynomials. Hint: The eigenvectors of

$$A = \begin{pmatrix} a & b \\ b & a \end{pmatrix} \quad \text{are} \quad q_+ = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad q_+ = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \ .$$

2. (*It is important to test software that you write. It is particularly important for scientific computing software. Within scientific computing, it is particularly important for Monte Carlo. This exercise asks you to write a piece of software and do a non-trivial test. You have to do lots of calculation and programming correctly to make this work.*) Write a procedure in C++ that computes the empirical auto-covariance function of a time series of length n. The input should be an array of length T, $(Y_1, \ldots, Y_T)$, and an integer, n. The output should be an array $\widehat{C}(t)$ for $t = 0, \ldots, n$.

$$\widehat{C}(t) = \frac{1}{T - t - 1} \sum_{k=1}^{T-t} \left(Y_{k+t} - \overline{Y} \right) \left(Y_k - \overline{Y} \right) \ ,$$

for $t = 0, \ldots, n$. The efficient way to do this, for large T and n, is to use the FFT. Feel free to do it this way. But it is also OK to use the

direct formulas. These should be fine for our applications with $T \leq 10^6$ and $n \leq 10^3$. Test your program by implementing the linear Gaussian recurrence of Exercise 1. Take $Y_k = v(X_k)$, where v is the eigenfunction of L corresponding to $\lambda = .81$. Print the computed and exact values $\widehat{C}(t)$, and $C(t)$, until t is so large that $\widehat{C}(t)$ is within noise of being zero. You can make a plot if you want, but a printout is enough. Use several values of T ranging from not very big to as big as your computer can handle easily. Your program should consist of three C++ files (and header files, and a makefile, and a possible python file), one for `main`, one for `timeSeries`, and one for `acor`. `Main` is the supervisor, doing all array memory allocation and calling the others. `TimeSeries` runs the X process and generates the Y_k. These are returned to `main` using array passing. `Acor` computes the numbers $\widehat{C}(t)$. The parameters should not be hard coded. Write the linear recurrence for a generic symmetric matrix, with parameters $a = .7$ and $b = .2$. Hint: Download the program from Exercise 3 first. It is always easier to create a program using pieces of another program. You can use the Exercise 3 program or the one from Week 1.

3. Download the tarball `Week2.tar` and the sample output file `Week2Output`. If you un-tar the file (`tar -xvf Week2.tar`) and type `make fTest`, you should get output that matches `Week2Output`. This program runs slowly on my computer. It may run even worse on yours. You will want to `ctrl C` to stop the program after the first few pieces of output. Notice how little the pictures change from output to output and note how many MCMC sweeps there are between pictures. This is a slow MCMC algorithm.

   (a) Verify that the probability distribution is being sampled correctly by running on a 3×3 lattice. Estimate $P(X - x)$ from the MCMC (count how many times each configuration is taken) and compare that to the theoretical probability $f(x)$. Some of the procedures included with the code should help. If you are feeling ambitious, try 4×4 or even 5×5. The latter requires extra coding. Do it if you are very interested.

   (b) Write a program that computes the number of occupied sites in the right half of the array. Output this time series to Python and plot it as a function of t. Do this for a few different β values (inverse temperature) to see the difference between high temperature free movement and low temperature slowness. You should not output this statistic for every sample, but only for samples separated by a sampling period. Choose a sampling period appropriately. Use a value of n that is as large as your computer can handle conveniently while still producing a reasonably long time series. Look at the time series plot to identify the *burn in* time.

   (c) Compute and plot the a the auto-covariance function of this time series after you have left off the burn in time. Try to estimate the

auto-correlation time from this plot. Do this for high and low temperature to see the difference.

(d) The MCMC moves in the program propose swaps between neighbors. Modify the program to propose swaps between randomly chosen sites. Repeat part (a) to see that you did it correctly. Repeat part (b) to see whether this method is better. The answer to that question depends on β.