Convergence rates of Markov chain Monte Carlo methods

by: David Clement[*]

1. Introduction

The following paper deals with the convergence rates of Markov Chain Monte Carlo (MCMC) algorithms. Section 2 provides a short review of the Metropolis-Hastings algorithm and quantitative measures of rate of convergence. It also mentions qualitative convergence, irreducibility, and recurrence, but those areas are not explored further. Section 3 introduces the "frog example" (Rosenthal, 1995a) and examines different proposal distributions and their rates of convergence to stationarity for a series of specific target distributions. This is done with eigenvalue analysis, as well as simulations. Section 4 deals with bounding these eigenvalues using a method from Diaconis and Saloff-Coste (1993). Section 5 discusses two other methods of bounding variation distance - minorisation conditions and logarithmic-Sobolev inequalities.

2.1 Markov chain Monte Carlo

As explained in Evans and Rosenthal (2003), Monte Carlo techniques generate random variables having certain discrete distributions. However, once it becomes too complex, it is not obvious how to simulate the distribution anymore. This is where Markov chains can be used. If we can find a Markov chain with a stationary distribution that is the same as the desired probability distribution $\{\pi(i)\}$ (the target distribution),

[*] Written as part of an undergraduate NSERC award in the summer of 2003 under the supervision of Jeffrey S. Rosenthal, Professor, Department of Statistics, University of Toronto.

then we can run the Markov chain for a long time, say *N* iterations, and note what state the chain is in after these *N* iterations. The probability that the chain is in state *i* will be approximately the same as the probability that our discrete random variable equals *i*.

We can get an idea of how far our approximation is from the target distribution by taking *M* samples using the Markov chain, finding the proportion of samples that ended up in state *i* (this is $\mu_N(i)$) for each *i* in the sample space, and then adding up the absolute value of the difference between $\mu_N(i)$ and $\pi(i)$ for all *i* and multiplying by ½. i.e. the total variation distance is $\|\mu_N - \pi\| = \frac{1}{2}\sum_i |\mu_N(i) - \pi(i)|$. This variation distance will not be zero because there is Markov chain error (*N* is not large enough) and there is Monte Carlo error (*M* is not large enough).

We are still left with the problem of finding the Markov chain with the desired stationary distribution. As Evans and Rosenthal (2003), and many others before them, explain, the Metropolis-Hastings algorithm is one good way to do this.

2.2 Metropolis-Hastings algorithm

This algorithm proposes a new point on the Markov chain which is either accepted or rejected. If the point is accepted, the Markov chain moves to the new point. If the point is rejected, the Markov chain remains in the same state. By choosing the acceptance probability correctly, we create a Markov chain which has $\{\pi(i)\}$ as a stationary distribution.

We begin with a state space χ and a probability distribution $\{\pi(i)\}$ on χ. Then we choose a proposal distribution $\{q_{ij} : i, j \in \chi\}$ with $q_{ij} \geq 0$ and $\sum_{j \in S} q_{ij} = 1$ for each $i \in \chi$. Given that $X_n = i$, X_{n+1} is computed as follows.

1. Choose $Y_{n+1} = j$ according to the Markov chain $\{q_{ij}\}$
2. Set $\alpha_{ij} = \min\left\{\frac{\pi(j)q_{ji}}{\pi(i)q_{ij}}\right\}$
3. With probability α_{ij}, let $X_{n+1} = Y_{n+1} = j$ (accept proposal)

   Otherwise, let $X_{n+1} = X_n = i$ (reject proposal)

As desired, this will create a chain with stationary distribution $\{\pi(i)\}$

Proof (Evans and Rosenthal, 2003 and Grimmett and Stirzaker, 2001)

Lemma 1 If $\pi_i P(X_{n+1} = j \mid X_n = i) = \pi_j P(X_{n+1} = i \mid X_n = j)$ for all *i, j* (known as reversibility) then π is a stationary distribution of *P*.

Proof of Lemma 1 If π satisfies the above conditions, then

$$\sum_i \pi_i p_{ij} = \sum_i \pi_j p_{ji} = \pi_j \sum_i p_{ji} = \pi_j$$

and so $\pi = \pi P$, which makes π a stationary distribution.

If we can show that the Markov chain resulting from the Metropolis-Hastings algorithm is reversible with respect to $\{\pi(i)\}$, then it follows that $\{\pi(i)\}$ is the stationary distribution by *Lemma 1*.

Obviously $\pi_i P(X_{n+1} = j \mid X_n = i) = \pi_j P(X_{n+1} = i \mid X_n = j)$ holds if $i = j$, so we will only consider $i \neq j$. If $X_n = i$, we need $Y_{n+1} = j$ to be proposed and accepted if we are to get $X_{n+1} = j$. Hence,

$$P(X_{n+1} = j \mid X_n = i) = q_{ij}\alpha_{ij}$$

$$= q_{ij} \min\left\{1, \frac{\pi_j q_{ji}}{\pi_i q_{ij}}\right\}$$

$$= \min\left\{q_{ij}, \frac{\pi_j q_{ji}}{\pi_i}\right\}$$

Multiplying both sides by π_i, we get

$$\pi_i P(X_{n+1} = j \mid X_n = i) = \min\{\pi_i q_{ij}, \pi_j q_{ji}\}$$

Similarly, we find that

$$\pi_j P(X_{n+1} = i \mid X_n = j) = \min\{\pi_i q_{ij}, \pi_j q_{ji}\}$$

The left sides of the above equations must be equal, so we have shown reversibility.

The Metropolis-Hastings algorithm can also be used for continuous random variables by using densities and continuous proposal distributions, but we will not explore that here. It is also possible to extend the theorem to the multidimensional case. The "Gibbs sampler", which we will now discuss very briefly, is a special case of the multivariate Metropolis-Hastings algorithm.

2.3 Gibbs sampler

Suppose $\pi(\cdot)$ is the joint distribution of $(x_1, x_2, ..., x_N)$. Here, we update each component based on the value of each other component

$$\pi(x_i \mid x_{-i}) = \pi(x_i \mid x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_N)$$

These distributions are called full conditionals. The Gibbs sampler is a special case of the Metropolis-Hastings algorithm where the proposals q are the full conditionals and the acceptance probability is always one.

The updating of components can be done systematically or randomly. The above is from Gibbs (2000).

2.4 Quantitative measures of convergence

In addition to the formula for total variation distance given in 2.1, there are two other equivalent definitions (see Rosenthal, 1995a). All three are listed below. Here, ν_1 and ν_2 are probability measures, and χ is the state space.

i) $\|\nu_1 - \nu_2\| = \frac{1}{2}\sum_{x\in\chi}|\nu_1(x) - \nu_2(x)|$

ii) $\|\nu_1 - \nu_2\| = \sup_{A\in\chi}|\nu_1(A) - \nu(A)|$ (supremum is taken over measurable subsets A)

iii) $\|\nu_1 - \nu_2\| = \sup_{\substack{f:\chi\to R \\ 0\le f(x)\le 1}} \left|E_{\nu_1}(f) - E_{\nu_2}(f)\right|$ (where E stands for expected value)

We will deal only with the first one in this paper.

We will also deal with $L^2\left(\frac{1}{\pi}\right)$ and $L^2(\pi)$ norms, which have connections to variation distance and eigenvalue analysis. These L^2 norms are defined as:

$$\|\nu_1 - \nu_2\|_{L^2\left(\frac{1}{\pi}\right)} = \sqrt{\sum_{x\in\chi}\left(\frac{((\nu_1 - \nu_2)(x))^2}{\pi(x)}\right)} \quad \text{and} \quad \|\nu_1 - \nu_2\|_{L^2(\pi)} = \sqrt{\sum_{x\in\chi}\left(((\nu_1 - \nu_2)(x))^2\pi(x)\right)}$$

In this paper, we will be using $(\nu_1 - \nu_2)(x) = \mu_k(x) - \pi(x)$.

2.5 Relationship between these measures of convergence

$$\|\mu_k - \pi\|_{\text{var}} = \frac{1}{2}\sum_{x\in\chi}|\mu_k(x) - \pi(x)|$$

$$= \frac{1}{2}\left\langle |\mu_k - \pi|, \pi \right\rangle_{L^2\left(\frac{1}{\pi}\right)}$$

$$\le \frac{1}{2}\|\mu_k - \pi\|_{L^2\left(\frac{1}{\pi}\right)} \cdot \|\pi\|_{L^2\left(\frac{1}{\pi}\right)} \quad \text{by Cauchy-Schwarz inequality}$$

$$= \frac{1}{2}\|\mu_k - \pi\|_{L^2\left(\frac{1}{\pi}\right)}\sqrt{\sum_{x\in\chi}\pi(x)\pi(x)\frac{1}{\pi(x)}}$$

$$= \frac{1}{2}\|\mu_k - \pi\|_{L^2\left(\frac{1}{\pi}\right)}$$

Hence, the variation distance is less than or equal to ½ the $L^2\left(\frac{1}{\pi}\right)$ norm. Unfortunately, no nice inequality can be obtained between variation distance and the $L^2(\pi)$ norm, since we end up with $\sqrt{\sum_{x\in\chi}\frac{1}{\pi(x)}}$ instead of the square root of 1 like we did above.

2.6 Qualitative convergence

Gibbs (2000) and Barndorff-Nielsen, Cox and Kluppelberg (2001) both give the following two definitions which help to decide whether a chain converges quickly enough to be useful in simulations.

A chain is geometrically ergodic if

$$\|P^t(x,\cdot) - \pi(\cdot)\| \le M(x)\rho^t \quad \text{for finite } M(x) \text{ and } \rho < 1$$

A chain is uniformly ergodic if for all x

$$\left\| P^t(x,\cdot) - \pi(\cdot) \right\| \le M\rho^t \text{ for finite } M \text{ and } \rho < 1$$

2.7 Irreducibility and recurrence

This information can also be found in Gibbs (2000) and Barndorff-Nielsen, Cox and Kluppelberg (2001). We did not explore it further in this work.

Define τ_A to be the time of the first visit to *A* for any $A \in \chi$ (χ is the state space). That is $\tau_A = \min\{t : X_t \in A\}$ ($\tau_A = \infty$ if there are no visits to *A*). A Markov chain is called $\phi - \mathit{irreducible}$ if there exists a non-zero probability measure on χ such that for any $A \in \chi$

$$\phi(A) > 0 \Rightarrow P(\tau_A < \infty \mid X_0 = x) > 0 \quad \text{for all } x \in \chi$$

To state the above in words, any set with a positive φ measure has a positive probability of being hit from any starting point *x*.

To have the result hold for all initial states (instead of all but a probability-zero exceptional state), we need a slightly stronger notion called *Harris recurrence*. This guarantees that for any $A \in \chi$,

$$\phi(A) > 0 \Rightarrow P(\tau_A < \infty \mid X_0 = x) = 1 \quad \text{for all } x \in \chi$$

Harris recurrence is important to have for a simulation so that you do not run your chain from the exceptional initial state.

3.1 The frog example

Consider the "frog example" (Rosenthal, 1995a) in which there are 1000 lily-pads arranged in a circle, numbered in order from 0 to 999. Suppose that a frog makes a jump every minute, beginning on pad 0, and also suppose that he has an equal probability of landing on any lily-pad within *m* pads (nodes) of his previous location. Rosenthal considers *m* = 1 and shows convergence to a uniform target distribution π (the frog having an equal probability of being at each pad) is a very slow process. In fact, $\|\mu_k - \pi\|$ > 0.1 until *k* exceeds 122,301. Obviously, convergence will occur more quickly if *m* is increased, or if shortcuts are added to allow larger jumps. In the case of the uniform target, *m* = 500 will give convergence immediately. The situation is more complicated when a non-uniform target is desired, and the Metropolis-Hastings algorithm is used. In the uniform case, no matter what *m* we choose, $q_{ij} = q_{ji}$, and $\pi(i) = \pi(j)$ so $\alpha_{ij} = 1$, $\forall i, j$, and we always accept the proposal. For non-uniform target distributions, we would expect an optimal *m* whose proposals lead to a Markov chain that converges the quickest. This *m* will not be 500 because that leads to too many rejections, but it will not be too low either because the jumps will be too small in that case. This introduces the idea of optimal scaling discussed by Roberts and Rosenthal (2001), although they deal almost exclusively with the continuous case. Later in section 3, we will visit optimal scaling for the family of target distributions discussed in this paper.

3.2 Family of target distributions used

$\pi(x) \propto (d(0,x)+1)^{-\beta}$ where *d*(*a, b*) is the shortest distance from *a* to *b* on the circle. β equal to 0.5, 1.0, 2.0, 5.0, 10, 20 were all studied. Clearly the higher β is, the higher $\pi(0)$ is. However, even this distribution is not peaked enough at 0 for a study of minorisation conditions to give reasonable bounds. In that section (5.2), we will also look at $\pi(x) \propto e^{-\left(d(0,x)^{\beta}\right)}$ for $\beta = 2$ and 5.

3.3 Study of eigenvalues

It is known that the second highest eigenvalue of a matrix P is related to the rate of convergence to its stationary distribution. Using the proposal distribution and the acceptance rate, the matrix $\{P(x, y)\}$ with stationary distribution $\{\pi(x)\}$ is created for the different β and different m's. See Appendix #1 for the MATLAB program that does this and finds the second largest eigenvalue of this matrix. Also, see Figure #1(a-g) below for a table of the results with 1000 nodes.

Figure #1(a) $\beta = 0.5$

m	2nd highest eigenvalue
180	0.87424681286953
190	0.87166589971567
191	0.87162483521290
192	**0.87161209098862**
193	0.87162491319268
195	0.87171715104516
200	0.87223907689506

Figure #1(b) $\beta = 1$

m	2nd highest eigenvalue
100	0.96454525127127
110	0.96211699380438
112	0.96202225156711
113	**0.96201625073779**
114	0.96203448162669
115	0.96207438489506
125	0.96320546327069

Figure #1(c) β = 2

m	2nd highest eigenvalue
50	0.98610315652020
58	0.98244791674512
59	0.98224599638054
60	**0.98220726967340**
61	0.98229619505990
62	0.98245773197201
65	0.98309090524785

Figure #1(d) β = 5

m	2nd highest eigenvalue
10	0.99819871895349
30	0.98657696383820
32	0.98494713857452
33	**0.98410552801069**
34	0.98443694799826
35	0.98487533604633
40	0.98674255645486

Figure #1(e) β = 10

m	2nd highest eigenvalue
10	0.99562710970574
20	0.98496386158779
23	0.98084123291182
24	**0.97955124183204**
25	0.98035315391692
27	0.98178201545008
50	0.99007931534401

Figure #1(f) β = 20

m	2nd highest eigenvalue
10	0.98836991948443
15	0.97682800076837
16	0.97420529009865
17	**0.97149478015797**
18	0.97297292140747
20	0.97560970956284
25	0.98039211945248

Figure #1(g) summary

β	m	2nd highest eigenvalue	rejection rate
0.5	192	0.87161209098862	0.2123
1	113	0.96201625073779	0.4997
2	60	0.98220726967340	0.9005
5	33	0.98410552801069	0.9811
10	24	0.97955124183204	0.9795
20	17	0.97149478015797	0.9714

The m column in Figure #1(g) shows the optimal m for that particular β. As expected, there is an optimal m that is not too high and not too low. We also see that β of 2 and 5 have the slowest long-run convergence rate since their second highest eigenvalues for the optimal m are the highest.

Unfortunately, the rejection rate column (calculated by multiplying π's and α's) shows no link to an optimal rejection percentage like that found in Roberts and Rosenthal (2001). The probability of acceptance to a node other than the current one was also looked at, but there was no pattern there either.

A quantitative bound on variation distance and value of the $L^2(\pi)$ norm can be found using all the eigenvalues, instead of only the second highest one. See Rosenthal (1995a) for the proofs. We find that:

$$\|\mu_k - \pi\|_{\text{var}} \leq \frac{1}{2}\sum_{x=0}^{n-1}\sum_{m=1}^{n-1}|a_m v_m(x)\|\lambda_m|^k$$

$$\|\mu_k - \pi\|_{L^2(\pi)} = \sum_{m=1}^{n-1}|a_m|^2|\lambda_m|^{2k}$$

where k is the number of iterations before taking an approximation to the stationary distribution, n is the number of nodes, $v_0,...,v_{n-1}$ are a basis of left eigenvectors of P corresponding to $\lambda_0,...,\lambda_{n-1}$ respectively and a_m are the unique coefficients satisfying

$\mu_0 = a_0 v_0 + a_1 v_1 + \cdots + a_{n-1} v_{n-1}$. The end of Appendix #1 has the code used to study these bounds.

Ideally, we would be able to use the above equations (even though they only provide an upper bound to the variation distance and $L^2(\pi)$ norm) to find the optimal *m* instead of running several long simulations. We will explore this idea now.

3.4 Connection between bounds using eigenvalues and simulation

When doing simulations using MCMC, Markov chain error and Monte carlo error can both be problems as mentioned previously. The Markov chain error can be controlled by constantly adding iterations (using a high constant number of runs to avoid a high Monte Carlo error). When adding more iterations does not help reduce the variation distance very much, a satisfactory number has been reached. The Monte Carlo error, however, is not as easy to avoid. It should go down approximately as the square root of the number of runs goes up, but in the frog example, it is very hard to have this error lower than the actual difference in variation distance between two *m*'s. Figure #2(b) below shows the variation distance after different numbers of runs for *m* = 24 and β = 10 with 400 iterations and 1000 nodes. This almost completely gets rid of the Markov chain error as can be seen in Figure #2(a), but even though the runs are well into the millions, the Monte Carlo error would need more runs to become satisfactorily low. This can use up a lot of computational time. Each entry below is completely separate; i.e. the data for 3 million runs does not affect the data for 5 million runs.

Figure #2(a)

iterations	variation distance
100	2.4325e-4
120	1.59147e-4
140	1.02601e-4
160	7.5728e-5
180	6.70294e-5
200	2.85437e-5
220	2.61725e-5
240	1.53473e-5
260	1.31236e-5
280	1.62452e-5
300	1.37689e-5
320	2.93243e-5
340	3.0797e-5
360	3.9397e-5
380	4.78014e-5
400	4.95985e-5

Figure #2(b)

number of runs (in millions)	variation distance
3	1.09555e-5
5	4.95985e-5
7	1.60423e-5
9	8.93828e-6
12	6.93959e-6
14	7.95605e-6
16	5.50603e-6
18	2.28559e-5
20	1.53241e-5
24	2.68304e-6
28	6.20349e-6
32	1.50024e-5

We can see by these figures that 32 million runs still will not be enough to avoid drastic Monte Carlo error. Another example is shown below in Figure #3 (again 400 iterations and 1000 nodes are used).

Figure #3

number of runs	m = 24 var. dist.	m = 50 var. dist.
3 million	1.09555e-5	2.02942e-5
5 million	4.95985e-5	6.60949e-5

In the previous section, we saw that β = 10 had the fastest convergence for m = 24 according to the eigenvalues. Obviously we expect m = 24 to converge much quicker in our simulation than m = 50 since the second highest eigenvalue was much lower. But in the above figure, m = 50 converged more after 3 million runs than m = 24 did after 5 million runs.

To actually show the connection between eigenvalue analysis and simulations, only 20 nodes were used with 900 million runs and 40 iterations to make the Monte Carlo error very small. The target distribution was proportional to $(d(0,x)+1)^{-2}$. The simulation C program can be seen in Appendix #2. Figure #4 below shows the simulation results, and Figure #5 shows the eigenvalue analysis.

Figure #4

m	Variation distance	$L^2(\pi)$	$L^2\left(\frac{1}{\pi}\right)$
2	8.005e-3	3.818e-3	2.571e-2
3	1.230e-3	6.347e-4	3.987e-3
4	7.031e-4	3.108e-4	1.794e-3
5	7.103e-4	4.537e-4	1.714e-3
6	1.103e-3	7.711e-4	2.410e-3
7	1.956e-3	1.364e-3	3.992e-3

Figure #5

m	2nd eigenvalue	$\frac{1}{2}\sum_{x=0}^{19}\sum_{m=1}^{19}\lvert a_m v_m(x)\rvert\lvert\lambda_m\rvert^{40}$	$\sum_{m=1}^{19}\lvert a_m\rvert^2\lvert\lambda_m\rvert^{80}$
2	0.93167	8.512e-3	4.519e-5
3	0.88472	1.369e-3	1.554e-6
4	0.84846	4.403e-4	2.025e-7
5	0.84243	4.989e-4	2.695e-7
6	0.85373	9.235e-4	9.470e-7
7	0.86722	1.755e-3	3.460e-6

The inequalities from 3.3 do not all hold since we have not completely eliminated the Monte Carlo error. However, we have done enough runs to have $m = 4$ show itself as the quickest converging m in both variation distance and $L^2(\pi)$ norm after 40 iterations just as the eigenvalues predicted for both. Doing more than 900 million runs only makes this even clearer.

4.1 Comparison of eigenvalues through Dirichlet forms

The following theory comes from work by Diaconis and Saloff-Coste (1993). Assume that $P(x, y)$, $\pi(x)$ is a reversible, irreducible chain (as our frog example's chain is) on a finite set χ, and let $l^2(\chi)$ have the scalar product $\langle f,g\rangle = \sum_{x\in\chi} f(x)g(x)\pi(x)$. Also, the operator $f \to Pf$, with $Pf(x) = \sum f(y)P(x,y)$ is self-adjoint on l^2 (because of reversibility) with eigenvalues $\beta_0 = 1 > \beta_1 \geq \beta_2 \geq \cdots \geq \beta_{|X|-1} \geq -1$. Let the Dirichlet form ε be defined as

$$\varepsilon(f,f) = \langle (I-P)f, f\rangle = \frac{1}{2}\sum_{x,y}(f(x)-f(y))^2\pi(x)P(x,y)$$

Proof of the second equality above

$$\langle (I-P)f, f\rangle = \sum_x \pi(x) f(x)((I-P)(f))(x)$$

$$= \sum_x \left[\pi(x) f(x) \left(f(x) - \sum_y f(y) P(x,y) \right) \right]$$

$$= \sum_x \pi(x)(f(x))^2 - \sum_{x,y} \pi(x) f(x) f(y) P(x,y)$$

$$= \sum_{x,y} \pi(x) P(x,y)(f(x))^2 - \sum_{x,y} \pi(x) f(x) f(y) P(x,y)$$

$$= \frac{1}{2} \sum_{x,y} \begin{pmatrix} \pi(x)P(x,y)(f(x))^2 + \pi(y)P(y,x)(f(y))^2 \\ -2\pi(x) f(x) f(y) P(x,y) \end{pmatrix}$$

$$= \frac{1}{2} \sum_{x,y} \begin{pmatrix} \pi(x)P(x,y)(f(x))^2 + \pi(x)P(x,y)(f(y))^2 \\ -2\pi(x) f(x) f(y) P(x,y) \end{pmatrix}$$

$$= \frac{1}{2} \sum_{x,y} \left(\pi(x) P(x,y) \left((f(x))^2 - 2 f(x) f(y) + (f(y))^2 \right) \right)$$

$$= \frac{1}{2} \sum_{x,y} (f(x) - f(y))^2 \pi(x) P(x,y)$$

Now if $\tilde{P}(x,y)$, $\tilde{\pi}$ is a second reversible Markov chain on *X*, we get

$$(*) \qquad \beta_i \le 1 - \frac{a}{A}\left(1 - \tilde{\beta}_i\right), \quad \text{if } \tilde{\varepsilon} \le A\varepsilon,\ \tilde{\pi} \ge a\pi$$

It is important to be able to find *A* to come up with a bound. Let $\tilde{P}, \tilde{\pi}$ and P, π be reversible Markov chains on a finite set χ. Assume that we know the eigenvalues of $\tilde{P}, \tilde{\pi}$ and we are trying to bound the eigenvalues of P, π.

For each pair $x \neq y$ with $\tilde{P}(x,y) > 0$, fix a predetermined sequence of steps $x = x_0, x_1, x_2, \ldots, x_k = y$ with $P(x_i, x_{i+1}) > 0$. This sequence will be called a path γ_{xy} of length $|\gamma_{xy}| = k$. Fix *E* to be the set of "edges" for *P* (i.e. the *x, y* such that *P*(*x, y*) > 0) and fix $\tilde{E}(e) = \tilde{E}(z,w)$ to be the set of paths that contain *e* = (*z, w*). Then if

$$A = \max_{(z,w)\in E} \left\{ \frac{1}{\pi(z)P(z,w)} \sum_{\tilde{E}(z,w)} |\gamma_{xy}| \tilde{\pi}(x) \tilde{P}(x,y) \right\}$$

we have $\tilde{\varepsilon} \leq A\varepsilon$. The proof of this can be seen in Diaconis and Saloff-Coste (1993).

4.2 Comparison in the frog example

Our goal is to see how tight the upper bound on eigenvalues is for a fixed β and different *m*'s. To save computational time, only 200 nodes are used instead of 1000. It is hoped that results can be extrapolated to that case or even the general case. Experimentation was done to find which *m* value gave the lowest second eigenvalue for *nodes* = 200 and β = 10. An *m* of 9 was found to do so. In Figure #6, *m*1 and *m*2 are the *m* values for P, π and $\tilde{P}, \tilde{\pi}$ respectively. We are bounding the second highest eigenvalues of P, π in terms of those of $\tilde{P}, \tilde{\pi}$. See Appendix #3 for the C program that finds the value of *A*.

Figure #6

$m1$	$m2$	2nd eig. for $m2$	A	2nd eig. for $m1$	upper bound
7	9	0.94726373	4.47357	0.95831927	0.988212
8	9	0.94726373	2.87749	0.94900079	0.981673
10	9	0.94726373	1.10526	0.95228623	**0.952286**
11	9	0.94726373	1.21053	0.95643526	**0.956435**
50	9	0.94726373	5.31579	0.9900793	**0.990079**
9	8	0.94900079	1.11765	0.94726373	0.954369
9	10	0.95228623	2.91186	0.94726373	0.983614
9	50	0.9900793	368.415	0.94726373	0.99973

The upper bound is perfect for values of $m1$ which are higher than $m2$ and also have higher second eigenvalues in the corresponding matrix (see boldface in the table above). This leads us to believe that the second highest eigenvalues for the two m's must be related only in terms of $m1$ and $m2$ in these cases. To see why this is, look at how A is determined.

When $m1$ is higher than $m2$, there is only one path that contains the edge from z to w $\left(\forall(z,w)\in E\right)$. This path goes directly from z to w, and hence has length of one. This implies the following

$$A = \frac{\tilde{\pi}(z)\,\tilde{P}(z,w)}{\pi(z)\,P(z,w)} = \frac{\frac{1}{2m2+1}}{\frac{1}{2m1+1}} = \frac{2m1+1}{2m2+1}$$

In this case, $\tilde{\pi}$ and π are the same, so they cancel out above. This also means $\tilde{\pi} \geq a\pi$ when $a = 1$. Hence, we have the relation

$$\beta_1 = 1 - \frac{2m2+1}{2m1+1}\left(1-\tilde{\beta}_1\right) \text{ (where } \beta_1, \tilde{\beta}_1 \text{ are the second highest eigenvalues)}$$

when $m1$ is higher than $m2$ and also has a higher second eigenvalue in the corresponding transition matrix.

4.3 The value of A in the frog example

Theorem In the frog example, A is always greater than or equal to 1 when we are comparing two chains with different m values.

Proof Assume $m2 < m1$. Then the path length is always one, the π's cancel out, and we are left with only $\frac{\tilde{P}(z,w)}{P(z,w)}$ which is greater than 1.

Now assume $m1 < m2$.

$$A = \max_{(z,w)\in E}\left[\frac{1}{\pi(z)P(z,w)}\sum_{\tilde{E}(z,w)}\left\lceil\frac{m2}{m1}\right\rceil\tilde{\pi}(x)\tilde{P}(x,y)\right]$$

$$= \max_{(z,w)\in E}\left[\frac{1}{\pi(z)\left[\frac{\min\left(\frac{\pi(w)}{\pi(z)},1\right)}{2m1+1}\right]}\sum_{\tilde{E}(z,w)}\left[\left\lceil\frac{m2}{m1}\right\rceil\tilde{\pi}(x)\frac{\min\left(\frac{\tilde{\pi}(y)}{\tilde{\pi}(x)},1\right)}{2m2+1}\right]\right]$$

$$= \max_{(z,w)\in E}\left[\frac{2m1+1}{\min(\pi(w),\pi(z))}\sum_{\tilde{E}(z,w)}\left\lceil\frac{m2}{m1}\right\rceil\frac{\min\left(\tilde{\pi}(y),\tilde{\pi}(x)\right)}{2m2+1}\right]$$

$$= \max_{(z,w)\in E}\left[\sum_{\tilde{E}(z,w)}\left\lceil\frac{m2}{m1}\right\rceil\frac{2m1+1}{2m2+1}\frac{\min\left(\tilde{\pi}(y),\tilde{\pi}(x)\right)}{\min(\pi(w),\pi(z))}\right]$$

take $\pi(w) \le \tilde{\pi}(x), \forall x$

$$\geq \sum \left\lceil \frac{m2}{m1} \right\rceil \frac{2m1+1}{2m2+1}$$

$$\geq 2\left\lceil \frac{m2}{m1} \right\rceil \frac{2m1+1}{2m2+1}$$

$$> 2\left\lceil \frac{m2}{m1} \right\rceil \frac{2m1}{2m2+1}$$

$$\geq \frac{4m2}{2m2+1}$$

$$> 1$$

Where $\left\lceil \frac{m2}{m1} \right\rceil$ is the lowest integer greater than or equal to $m2/m1$. i.e. the path length, which is at least 2 when $m2 > m1$.

This completes the proof. (Note: $A = 1$ when $m1 = m2$)

Since A is always at least 1, when we want to use (*) from 4.1 to give an upper bound to the second highest eigenvalue of one chain in terms of another, it gives a meaningless bound when we use the higher second eigenvalue as known and try to bound the lower second eigenvalue. The bound will be even greater than the higher second eigenvalue.

Although A is always at least 1 in the frog example, that is not true of every reversible $\tilde{P}, \tilde{\pi}$ and P, π. Take for example,

$$P = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix} \quad \text{and} \quad \tilde{P} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

Here, both P and $\tilde{P}$ have stationary distributions $\begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$. For each pair $x \neq y$, the path from x to y can be of length 1, and for all (*z, w*), there is only one pair (*x, y*) where $x \neq y$ that has (*z, w*) as an edge. Hence,

$$A = \frac{1}{\frac{1}{3} \cdot \frac{1}{2}} \left(\frac{1}{3} \cdot \frac{1}{3} \right) = \frac{2}{3}$$

This actually gives perfect bounds on the eigenvalues using (*) since P has eigenvalues 1, $-\frac{1}{2}$, $-\frac{1}{2}$ and $\tilde{P}$ has eigenvalues 1, 0, 0.

5.1 Minorisation conditions

The following theory comes from Rosenthal (1995b). Proofs of the lemmas and theorems can be found there. The numbering is the same in the following to avoid confusion. Some theorems are skipped here.

Theorem 1 Suppose that a Markov chain *P(x, dy)* on state space χ satisfies the following (a minorisation condition)

$$P^{k_0}(x, A) \geq \varepsilon Q(A), \quad \forall x \in R \subseteq \chi \text{ for all measurable subsets } A \subseteq \chi$$

where $Q(\cdot)$ is a probability measure on χ,

k_0 is a positive integer, and $\varepsilon > 0$

Let $X^{(k)}, Y^{(k)}$ be two realizations of the Markov chain defined jointly from any starting distribution and let

$$t_1 = \inf\{m : (X^{(m)}, Y^{(m)}) \in R \times R\}, \quad \text{and for } i > 1 \text{ let}$$

$$t_i = \inf\{m : m \geq t_{i-1} + k_0, (X^{(m)}, Y^{(m)}) \in R \times R\}$$

Then set $N_k = \max\{i : t_i < k\}$. Now for any $j > 0$,

$$\left\| L(X^{(k)}) - L(Y^{(k)}) \right\| \leq (1-\varepsilon)^{\left[\frac{j}{k_0}\right]} + P(N_k < j)$$

where $[r]$ is the greatest integer not exceeding r

The above theorem is usually applied with $L(Y^{(0)}) = \pi$ and used to find the distance to stationarity. If R can be kept small, then we can find a larger ε, which will clearly give a tighter bound.

Lemma 3 Let t_i be the "k_0-delayed hitting times of $R \times R$" as in the theorem above and let $r_i = t_i - t_{i-1}$ (with $r_1 = t_1$) represent the "k_0-delayed i^{th} return time to $R \times R$".

Then for any $\alpha > 1$, $P(N_k < j) \leq \alpha^{-k} E\left(\prod_{i=1}^{j} \alpha^{r_i} \right)$

If we bound exponential moments $E(\alpha^{r_i})$ of the return times of $(X^{(k)}, Y^{(k)})$ to $R \times R$, we can get a tighter bound. The following lemma does this using an auxiliary function h whose expectation is decreasing when $(X^{(k)}, Y^{(k)}) \in R \times R$. This is related to "drift conditions".

Lemma 4 Let $X^{(k)}$ and $Y^{(k)}$ be two Markov chains on χ defined jointly just as in Theorem 1 with $R \in \chi$ and k_0-delayed i^{th} return times to $R \times R$ as in Lemma 3.

Suppose there is an $\alpha > 1$ and a function $h : \chi \times \chi \to \Re$ such that h is always greater than or equal to 1 and

$$E\left(h\left(X^{(1)}, Y^{(1)}\right) \mid X^{(0)} = x, Y^{(0)} = y\right) \le \alpha^{-1} h(x, y),\ \forall (x, y) \in R \times R$$

Then

$E\left(\alpha^{r_i}\right) \le E\left(h\left(X^{(0)}, Y^{(0)}\right)\right)$ and for $i > 1$ and any choices of $r_1, \ldots, r_{i-1}$,

$$E\left(\alpha^{r_i} \mid r_1, \ldots, r_{i-1}\right) \le \alpha^{k_0} \sup_{(x,y)\in R\times R} E\left(h\left(X^{(1)}, Y^{(1)}\right) \mid X^{(0)} = x, Y^{(0)} = y\right)$$

Now we obtain the following,

Theorem 5 Suppose a Markov chain $P(x, dy)$ satisfies Theorem 1 and Lemma 4, then if $v = L(X^{(0)})$ is the initial distribution and π is the stationary distribution, then for any $j > 0$, the variation distance to π after k iterations satisfies

$$\left\| L(X^{(k)}) - \pi \right\| \le (1-\varepsilon)^{\left\lfloor \frac{j}{k_0} \right\rfloor} + \alpha^{-k+(j-1)k_0} A^{j-1} E_{v\times\pi}\left[h\left(X^{(0)}, Y^{(0)}\right)\right]$$

where $A = \sup_{(x,y)\in R\times R} E\left(h\left(X^{(k_0)}, Y^{(k_0)}\right) \mid X^{(0)} = x, Y^{(0)} = y\right)$ and $X^{(0)}$ is distributed according to v and $Y^{(0)}$ according to π.

Choosing j to be a small multiple of k is often a good idea, and a good choice for h is often of the form $h(x, y) = 1 + (x_i - a)^2 + (y_i - a)^2$ where x is a vector and x_i is the i^{th} coordinate. This works well when x_i drifts exponentially quickly towards the value a.

The following lemma is good for establishing a minorisation condition $P^{k_0}(x, A) \ge \varepsilon Q(A)$ (see Theorem 1) in practice.

Lemma 6

a) Suppose that a Markov chain transition kernel P on state space χ satisfies

$P^{k_1}(x, R_2) \geq \varepsilon_1$, $\forall x \in R_1$ and

$P^{k_2}(x,\cdot) \geq \varepsilon_2 Q(\cdot)$, $\forall x \in R_2$ for some probability measure $Q(\cdot)$ on χ

Then the minorisation condition is satisfied when $k_0 = k_1 + k_2$ with $R = R_1$ and $\varepsilon = \varepsilon_1 \varepsilon_2$.

b) Given a positive integer k_0 and a subset $R \subseteq \chi$, there exists a probability measure $Q(\cdot)$ so that

$P^{k_0}(x,\cdot) \geq \varepsilon Q(\cdot)$, $\forall x \in R$

where $\varepsilon = \int_{\chi} \left(\inf_{x \in R} P^{k_0}(x, dy) \right)$

Theorem 5 can also be stated in another way using a drift condition on the original chain rather than on the coupled chain.

Theorem 12 Suppose that a Markov chain *P*(*x, dy*) on state space χ satisfies the drift condition

$E\left(V(X^{(1)}) \mid X^{(0)} = x \right) \leq \lambda V(x) + b$, $x \in \chi$ for some $V : \chi \rightarrow R^{\geq 0}$ and some $\lambda < 1$ and $b < \infty$

and further satisfies a minorisation condition

$P(x,\cdot) \geq \varepsilon Q(\cdot)$, $\forall x \in \chi$

with $V(x) \le d$ for some $\varepsilon > 0$, some probability measure $Q(\cdot)$ on χ and some $d > \dfrac{2b}{1-\lambda}$. Then for any $0 < r < 1$, beginning in the initial distribution *v*, we have

$$\left\| L(X^{(k)}) - \pi \right\| \le (1-\varepsilon)^{rk} + (\alpha^{-(1-r)} A^r)^k (1 + \frac{b}{1-\lambda} + E_v(V(X_0))$$

where

$$\alpha^{-1} = \frac{1+2b+\lambda d}{1+d} < 1, \quad A = 1 + 2(\lambda d + b)$$

5.2 Minorisation condition in the frog example

Using Theorem 5, we wrote a program (see Appendix #4) that finds the bound on the variation distance $\|\mu_k - \pi\|$. With the target distribution used in the eigenvalue analysis and simulations, the bounds were barely less than 1, so a more peaked target distribution was used here to see if the bounds could be significantly better than 1. This target has $\pi(x) \propto e^{-\left(d(0,x)^{\beta}\right)}$. We used β of 2 and 5.

It appears that *R* with just a single state is a good choice for any value of *m* since *R* with more elements makes ε less than 1, which more than offsets any increase in α. *R* of only node 0 appears to be a good choice since $\left(\dfrac{h(x,y)}{E\left(h(X^{(1)}, Y^{(1)}) \mid X^{(0)} = x, Y^{(0)} = y\right)} \right)$, where $h(x,y) = 1 + d(0,x)^{h\exp} + d(0,y)^{h\exp}$, reaches its minimum over *x* and *y* with *x* and *y* equal to zero, which makes α less than 1. We have also chosen $k_0 = 1$. Figure #7 gives a summary of the bounds for different *m* and *hexp* with β = 5, 100 nodes and 200 iterations.

Figure #7

hexp	*m*	bound
15	1	2.08e-8
	2	9.01e-13
	3	4.91e-9
25	1	7.27e-13
	2	9.01e-13
	3	4.91e-9
30	1	7.64e-15
	2	9.02e-13
	3	4.92e-9
40	1	2.41e-18
	2	2.89e-12
	3	2.42e-8

Once *hexp* gets bigger than 40, it reaches a point where the increase in *h* for different *x* and *y* more than offsets the very small chance of going to a node with a greater distance from node zero. This makes the bound quite large since $\alpha < 1$. For *hexp* lower than 15, α is lower since the decrease in *h* for different *x* and *y* is not large enough to make a big difference in $\left(\frac{h(x,y)}{E\left(h(X^{(1)},Y^{(1)}) \mid X^{(0)} = x, Y^{(0)} = y\right)}\right)$.

Figure #7 does show that for $\beta = 5$, $m = 1$ has the lowest bound (arrived at with *hexp* = 40). In fact, according to the eigenvalue analysis, $m = 1$ does converge the fastest. The second eigenvalue for $m = 1$ is 0.6667 compared to 0.7265 for $m = 2$. The variation distances after 200 iterations for these *m*'s are on the order of 10^{-36} and 10^{-28} respectively. So although the minorisation bound did give the ideal *m* in this case, the bound was so poor, that it would not be as reliable as eigenvalue analysis in this case.

Turning our attention to $\beta = 2$, we would expect since this stationary distribution is not as peaked at node zero that the minorisation bound would not be as good. The eigenvalue analysis here tells us that $m = 1$, 2, or 3 should converge the quickest since the eigenvalues for m = 1, 2 and 3 respectively are 0.7380, 0.7280 and 0.7468, which put the variation distance around the order of 10^{-26} after 200 iterations, whereas m = 4 has an eigenvalue of 0.8030 which after 200 iterations puts the variation distance on the order of 10^{-20}. However, taking *hexp* = 2.1 and m = 4 gives a bound of 0.000167 on the variation distance, which is lower than we get for m < 4 for any value of *hexp*. Not surprisingly, since the bounds are so poor, we cannot tell which m will convergence the fastest by looking at the minorisation condition bounds. The same thing happens when we use the same family of target distributions as in the eigenvalue analysis and simulations.

5.3 Logarithmic-Sobolev inequalities

In addition to minorisation conditions, log-Sobolev inequalities can be used to bound the convergence of a Markov chain to its stationary distribution. These inequalities often give tighter bounds. The following theory comes from work by Diaconis and Saloff-Coste (1996). Proofs of statements made here can be found in that paper.

Let χ be a finite state space and let $P(x,y) \geq 0$ with $\sum_y P(x,y) = 1$. Also, let the continuous time semigroup associated with *P* be $H_t = \exp(-t(I-P))$. Denote its kernel by $H_t^x(y) = H_t(x,y)$ which is the distribution at time *t* of the chain started at *x*.

We know that $H_t^x(y) \to \pi(y)$ and we want to bound $\|H_t^x - \pi\|_{TV}$. One way to do this is using the ℓ^2 distance with respect to π. i.e.

$$2\|H_t^x - \pi\|_{TV} \leq \|(H_t^x/\pi) - 1\|_{2,\pi}$$

This norm can be represented as an operator norm,

$$\max_x \left\| (H_t^x/\pi) - 1 \right\|_{2,\pi} = \left\| H_t - E \right\|_{2\to\infty}$$ where $E : f \to Ef$ is the operator that associates *f* to its mean with respect to π.

Now if we break up *t* into $t_1 + t_2$ where both t_1 and t_2 are greater than or equal to zero, we get

$$\left\| H_t - E \right\|_{2\to\infty} \le \left\| H_{t_1} \right\|_{2\to\infty} \left\| H_{t_2} - E \right\|_{2\to 2}$$

We can bound the $2 \to 2$ norm in terms of the second eigenvalue

$$\lambda = \min\left(\frac{\varepsilon(f,f)}{Var(f)} : Var(f) > 0 \right)$$

where $\varepsilon(f,f) = \frac{1}{2}\sum_{x,y} |f(x) - f(y)|^2 P(x,y)\pi(x)$ and

$$Var(f) = \frac{1}{2}\sum_{x,y} |f(x) - f(y)|^2 \pi(y)\pi(x)$$

One bound using the second eigenvalue (following from taking t_1 = 0) is

$$4\left\| H_t^x - \pi \right\|_{\text{var}}^2 \le \frac{1}{\pi_*} e^{-2\lambda t} \qquad \text{(eq. 1)}$$

The above bound can be improved upon by taking positive t_1's. To do this we need bounds on the decay of $\|H\|_{2\to\infty} = \max_x \|H_t^x / \pi\|_2$ as a function of *t*. Log-Sobolev inequalities can be used to estimate this decay.

A log-Sobolev inequality is of the form

$$L(f) \le C\varepsilon(f,f) \qquad \text{(eq. 2)}$$

where $L(f) = \sum |f|^2 \log\left(\frac{|f|^2}{\|f\|_2^2} \right)\pi$

Define the log-Sobolev constant of the chain (P,π) by

$$\alpha = \min\left(\frac{\varepsilon(f,f)}{L(f)} : L(f) \neq 0 \right)$$

Then $\frac{1}{\alpha}$ is the smallest constant C such that eq. 2 holds for all f. α always satisfies $\alpha \leq \frac{\lambda}{2}$ and is sometimes equal to $\frac{\lambda}{2}$. Using this log-Sobolev inequality, we can get $2\left\|H_t^x - \pi\right\|_{TV}^2 \leq \left(\log\frac{1}{\pi_*} \right) e^{-2\alpha t}$ which is an improvement on eq. 1 roughly when $\frac{1}{\lambda}\log\frac{1}{\pi_*} \geq \frac{1}{\alpha}\log\log\frac{1}{\pi_*}$, which it does in our variations of the frog example. However, log-Sobolev inequalities were not explored further than this.

References

Barndorff-Nielsen, Ole E., Cox, David R. and Kluppelberg, Claudia (2001), *Complex Stochastic Systems*. Chapman & Hall/CRC, New York.

Diaconis, P. and Saloff-Coste, L. (1993), Comparison theorems for reversible Markov chains. Ann. Appl. Prob. **3**, 696-730.

Diaconis, P. and Saloff-Coste, L. (1996), Logarithmic Sobolev inequalities for finite Markov chains. Ann. Appl. Prob. **6**, 695-750.

Evans, Michael J. and Rosenthal, J.S. (2003), *Probability and Statistics: The Science of Uncertainty.* W.H. Freeman and Company, New York.

Gibbs, Alison L. (2000). Convergence of Markov Chain Monte Carlo algorithms with applications to image restoration. PhD thesis. University of Toronto.

Grimmett, Geoffrey and Stirzaker, David (2001), *Probability and Random Processes 3rd ed.* Oxford University Press, New York.

Roberts, Gareth O. and Rosenthal, J.S. (2001), Optimal scaling for various Metropolis-Hastings algorithms. Statistical Science **16**, 351-367.

Rosenthal, J.S. (1995a), Convergence rates of Markov chains. SIAM Review **37**, 387-405.

Rosenthal, J.S. (1995b), Minorization Conditions and Convergence Rates for Markov Chain Monte Carlo. Journal of the American Statistical Association **90**, 558-566.

Appendix #1 (2nd eigenvalue)

```
% The following MATLAB .m program finds the value of the second highest
% eigenvalue in absolute value of the matrix P from the 'frog example'.
% A lower second highest eigenvalue generally means the matrix converges
% more quickly to stationarity.

m = 9;            % P(x,y) = 1/(2m+1) for all x, y within m of each other
nodes = 200;      % P is a nodes*nodes matrix
beta = 10;        % target distribution is proportional to (dist(x,1)+1)^-b
                  % where dist(x,1) is the number of nodes to get from x to 1

% the following two loops calculate the stationary (target) distribution
% of the matrix P
denom = 0;
for i = 1:nodes  % array indices cannot be zero here, but this is allowed for by subtracting
                 % one everywhere
  min = i-1;
  if nodes-i+1 < min
    min = nodes-i+1;  % distance from the current node to 1
  end
  denom = denom + (min + 1)^(-beta);
end
for i = 1:nodes
  min = i-1;
  if nodes-i+1 < min
    min = nodes-i+1;
  end
  pi(1, i) = (min + 1)^(-beta)/denom;    % stationary distribution of P
end

for i = 1:nodes
   P(i, i) = 1/(2*m+1);
   for j = 1:nodes
      if pi(1, j) >= pi(1, i)
         alpha = 1;       % alpha is the probability that we jump from the
                          % current node, i, to the proposed node, j
      else
         alpha = pi(1, j)/pi(1, i);
      end

      if 0 < abs(i-j) & abs(i-j) < (m+1)
         P(i, j) = alpha/(2*m+1);
         P(i, i) = P(i, i) + (1-alpha)/(2*m+1);
      elseif abs(i-j) >= (nodes-m)       % the highest nodes and the lowest
                                         % nodes have edges between them too
         P(i, j) = alpha/(2*m+1);
```

```
        P(i, i) = P(i, i) + (1-alpha)/(2*m+1);
      elseif (m+1) <= abs(i-j) & abs(i-j) < (nodes-m)
        P(i, j) = 0;
      end
  end
end

y = eig(P);      % finds a vector of the eigenvalues; the following loop
                 % isolates the second highest one in absolute value
max = 0;
max2 = 0;
for k = 1:nodes
  if abs(y(k, 1)) > max
    max2 = max;
    max = abs(y(k, 1));
  elseif abs(y(k, 1))>max2 & max>= abs(y(k, 1))
    max2 = abs(y(k, 1));
  end
end
max2


% the following can be added on to the above program to find bounds
% on variation distance using all the eigenvalues; a slight modification
% can be made to see bounds on L2(pi) distance
iters = 40;
Q = P'; % we want left eigenvectors of P which will be right eigenvectors
        % of Q
sum = 0;
[evecs, evals] = eig(Q);
start = zeros(nodes, 1);
start(1,1) = 1;
am = evecs\start;  % finding a sub m for all m
for i = 1:nodes
  for j = 2:nodes
    sum = sum + abs(am(j,1)*evecs(i,j)) * abs(evals(j,j))^iters;
  end
end
sum = sum/2;
sum  % bound on variation distance
```

<u>Appendix #2</u> (metropolishastings)

```
/* The following C program studies the speed of convergence of a matrix to
   its stationary distribution using the Metropolis-Hastings algorithm.
   The 'frog example' is used again */

#define NODES 20            /* size of the state space */
#define ITERATIONS 40   /* number of jumps before we take an approximate
                              sample from the stationary distribution */
#define RUNS 500000       /* number of trials that are averaged together to
                              estimate the stationary distribution */
#define M 5                      /* all nodes within M of each other have an edge
                               between them */
#define BETA 2                /* target distribution is proportional to (1 +
                               distance to node zero)^-BETA */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

void simulate();
int distance (int);

int main()
{
 srand((unsigned) time(NULL));
 simulate();
 exit(0);
}

void simulate()
{
 int edges[NODES][NODES], total[NODES], sum[NODES];
 int current, temp, partsum, proposal, c, i, j, rejects, min;
 double sdist[NODES], target[NODES];
 double rejsum=0, vardist=0, l2distpi=0, l2distpiinv=0, denom=0;
 double randomnum, randomnum2, alpha, rejperc;
 FILE *fp;
 fp = fopen("outputfile", "w");

 /* setting all values in arrays to zero so we can add them */
 for (i=0; i<NODES; i++)
 {
  for (j=0; j<NODES; j++)
    edges[i][j] = 0;
   total[i] = 0;
   sum[i] = 0;
 }

 /* setting all the edges with equal weight */
 for (i=0; i<NODES; i++)
   for (j=0; j<=M; j++)
   {
    temp = (i+j) % NODES;
    edges[i][temp] = 1;
    edges[temp][i] = 1;
   }

 /* finding the number of edges from each node (actually just 2M+1 in
    this case, but written more generally to allow changes to code) */
 for (i=0; i<NODES; i++)
   for(j=0; j<NODES; j++)
    sum[i] += edges[i][j];

 /* calculating the given target distribution for NODES and BETA
    we don't actually need pi - only the ratios, but it's good to have it */
 for (i=0; i<NODES; i++)
 {
   min = distance(i);
   denom += pow(min+1, -BETA);
 }
 for (i=0; i<NODES; i++)
 {
   min = distance(i);
   target[i] = pow(min+1, -BETA)/denom;
 }

 /* the actual simulation */
 for (i=0; i<RUNS; i++)
 {
   rejects = 0; /* keeps track of rejections to find what percentage of
                   proposals are rejected (this hopefully leads to a
                   connection to 'optimal scaling'), resets often to
                   avoid overflow */
   current = 0; /* starting the simulation at node 0 here */
   for (j=0; j<ITERATIONS; j++)
   {
    partsum = 0;
    randomnum = rand()/(RAND_MAX+1.0)*sum[current];
    /* starting the loop M spots away from the current node maximizes
       efficiency since this is where all the edges are */
```

```
      c = (current + NODES - M) % NODES;
      while (partsum < randomnum)
      {
        partsum += edges[current][c];
        proposal = c;
        c = (c+1) % NODES;
      }

      randomnum2 = rand()/(RAND_MAX+1.0);
      /* sum is q's reciprocal in this case */
      alpha = target[proposal]*sum[current]/(target[current]*sum[proposal]);
      if (randomnum2 < alpha)    /* accepting the proposed jump */
        current = proposal;
      else
        rejects++;
    }
    rejsum += ((double)rejects)/ITERATIONS;
    total[current]++;
  }

  /* calculating the different types of norms to quantitatively study how
      well we approximated the target distribution */
  for (i=0; i<NODES; i++)
  {
    sdist[i] = ((double)total[i])/RUNS;
    vardist += fabs(sdist[i]-target[i]);
    l2distpi += pow(sdist[i]-target[i], 2.0)*target[i];
    l2distpiinv += pow(sdist[i]-target[i], 2.0)/target[i];
  }
  vardist = 0.5*vardist;
  l2distpi = sqrt(l2distpi);
  l2distpiinv = sqrt(l2distpiinv);
  rejperc = 100*rejsum/RUNS;
  fprintf (fp, "Variation distance is %g\n", vardist);
  fprintf (fp, "L2 dist(pi) is %g\n", l2distpi);
  fprintf (fp, "L2 dist(1/pi) is %g\n", l2distpiinv);
  fprintf (fp, "Rejection percentage is %g\n", rejperc);
  fclose(fp);
}

int distance(int i)
{
  if (NODES-i < i)
    return NODES-i;
  return i;
}
```

Appendix #3 (findingA)

```
/* The following C program finds the value of A (specifically for our frog
   example) used in eigenvalue comparison in Diaconis and Saloff-Coste
   (1993), "Comparison theorems for reversible Markov chains."  Ann. Appl.
   Prob. 3, 696-730 */

#define NODES 200      /* the size of the state space */
#define M1 10          /* P1 has edges between nodes within M1 of each other */
#define M2 9           /* P2 has edges between nodes within M2 of each other */
#define BETA 10        /* the target distribution is proportional to
                          (dist(x,0)+1)^-BETA where dist(x,0) is the distance
                          from node x to node 0 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int findLength(int, int, int, int);
int distance (int);

int main()
{
 int min, i, j, z, w, x, y, length;
 int searchstart, searchstart2, searchend, searchend2;
 double alpha, max=0, sum, denom=0;
 double P1[NODES][NODES], P2[NODES][NODES], pi1[NODES], pi2[NODES];

 for (i=0; i<NODES; i++)
 {
  min = distance(i);
  denom = denom + pow(min+1.0, -BETA);
 }
 for (i=0; i<NODES; i++)
 {
  min = distance(i);
  pi1[i] = pow(min+1.0, -BETA)/denom;
  pi2[i] = pi1[i];   /* the stationary distributions of P2 and P1 */
 }

 for (i=0; i<NODES; i++)
 {
  P1[i][i] = 1.0/(2*M1+1);
  P2[i][i] = 1.0/(2*M2+1);
  for (j=0; j<NODES; j++)
  {
    if (pi1[j] >= pi1[i])
      alpha = 1;  /* alpha is the probability that the proposed state, j,
                     is accepted given that we are in state i; this is
                     the same for P1 and P2 since they have the same
                     stationary distribution.  However, the proposals
                     are different. */
    else
      alpha = pi1[j]/pi1[i];

    if ((0 < abs(i-j)) && (abs(i-j) < M1+1))
    {
      P1[i][j] = alpha/(2*M1+1);
      P1[i][i] = P1[i][i] + (1-alpha)/(2*M1+1);
    }
    else if (abs(i-j) >= NODES-M1) /* the highest nodes are connected to
                                      the lowest nodes */
    {
      P1[i][j] = alpha/(2*M1+1);
      P1[i][i] = P1[i][i] + (1-alpha)/(2*M1+1);
    }
    else if ((M1+1 <= abs(i-j)) && (abs(i-j) < NODES-M1))
      P1[i][i] = 0;

    if ((0 < abs(i-j)) && (abs(i-j) < M2+1))
    {
      P2[i][j] = alpha/(2*M2+1);
      P2[i][i] = P2[i][i] + (1-alpha)/(2*M2+1);
    }
    else if (abs(i-j) >= NODES-M2)
    {
      P2[i][j] = alpha/(2*M2+1);
      P2[i][i] = P2[i][i] + (1-alpha)/(2*M2+1);
    }
    else if ((M2+1 <= abs(i-j)) && (abs(i-j) < NODES-M2))
      P2[i][j] = 0;
  }
}

for (z=0; z<NODES; z++)
{
  /* an edge of P cannot be outside the following boundaries */
  searchstart = (z+NODES-M1) % NODES;
  searchend = (z+M1) % NODES;
  if (searchstart > searchend)
    searchend += NODES;   /* allows wrapping around from NODES-1 to 0 */
  for (i=searchstart; i<=searchend; i++)
```

```
    {
      w = i % NODES;
      sum = 0;
      for (x=0; x<NODES; x++)
      {
        searchstart2 = (x+NODES-M2) % NODES;
        searchend2 = (x+M2) % NODES;
        if (searchstart2 > searchend2)
          searchend2 += NODES;
        for (j=searchstart2; j<=searchend2; j++)
        {
          y = j % NODES;
          length = findLength(z, w, x, y);
          sum = sum + length*pi2[x]*P2[x][y];
        }
      }
      sum = sum/(pi1[z]*P1[z][w]);
      if (sum > max)
        max = sum;
    }
  }
  printf ("A is %g\n", max);
  exit(0);
}

/* returns the path length from x to y if (z,w) was one of the steps on
   the path, otherwise returns 0 */
int findLength(int z, int w, int x, int y)
{
  int left, right, indicator, rem, min, len;

  indicator = 0; /* changes to 1 when (z,w) is found to be a step on the
                    path from x to y */
  /* the edge comes by moving clockwise, e.g. 1->2*/
  if (((x<y) && (y-x < NODES/2)) || ((y<x) && (x-y >= NODES/2)))
  {
    left = x;
    right = (left + M1) % NODES;
    while ((abs(left-y) > M1) && (abs(left-y)<NODES-M1))
    {
      if ((left==z) && (right==w))
      {
        indicator = 1;
        left = y;   /* ending the while loop */
      }
      right = (right+M1) % NODES; /* updating the values for the next
```

```
                                       iteration */
      left = (left+M1) % NODES;
    }
    if ((left==z) && (y==w))
      indicator = 1;
  }

  /* the edge comes by moving counterclockwise, e.g. 2->1 */
  else if (((x<y) && (y-x >= NODES/2)) || ((y<x) && (x-y < NODES/2)))
  {
    right = x;
    left = (x+NODES-M1) % NODES;
    while ((abs(right-y)>M1) && (abs(right-y)<NODES-M1))
    {
      if ((right==z) && (left==w))
      {
        indicator = 1;
        right = y;
      }
      right = (right+NODES-M1) % NODES;
      left = (left+NODES-M1) % NODES;
    }
    if ((right==z) && (y==w))
      indicator = 1;
  }

  min = distance(abs(x-y));    /* the shortest distance from x to y */
  rem = min % M1;
  if (rem==0)
    len = indicator*min/M1; /* len is the quickest way to get from x to y
                               jumping at most M1 nodes at time, but only
                               if indicator=1 */
  else
    len = indicator*(min/M1+1);
  return len;
}

/* finds the shortest distance to node 0 from node i*/
int distance(int i)
{
  if (NODES-i < i)
    return NODES-i;
  return i;
}
```

<u>Appendix #4</u> (minorisation)

```
/* This C program uses the theory from Rosenthal (1995b) applied to the
   frog example to find a bound on the total variation distance after 200
   iterations */

#define NODES 100   /* the size of the state space */
#define M 4         /* P has edges between nodes within M of each other */
#define ITERS 200   /* the number of jumps before we take an approximate
                       sample from the stationary distribution */
#define BETA 2      /* the target distribution is proportional to
                       e^(-((distance to node zero)^BETA)) */
#define HEXP 2      /* the exponent in the calculation of h below */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int distance(int);

int main()
{
 int min, i, j, x, y, disti, distj, distx, disty;
 double P[NODES][NODES], pi[NODES];
 double bestbound=1, Eh0=0, Eh1=0, alph=100, epsilon, A=0, denom=0,
   acceptalpha, bound, temp;

 /* the following two loops calculate the stationary
   distribution of the chain */
 for (i=0; i<NODES; i++)
 {
  min = distance(i);
  denom = denom + exp(-pow(min, BETA));
 }
 for (j=0; j<NODES; j++)
 {
  min = distance(j);
  pi[j] = exp(-pow(min, BETA))/denom;
 }

 /* calculates a transition matrix with the desired stationary
   distribution */
 for (i=0; i<NODES; i++)
 {
  P[i][i] = 1.0/(2*M+1);
  for (j=0; j<NODES; j++)
 {
   if (pi[j] > pi[i])
      acceptalpha = 1;
   else
   {
      disti = distance(i);
      distj = distance(j);
      /* to avoid dividing zero by zero; this makes P slightly
         different than actually specified, but the difference is
         minor, and the same thing had to be done in the eigenvalue
         calculation */
      if ((pi[i]<pow(10, -317)) && (distj>disti))
        acceptalpha = 0;
      else if ((pi[i]<pow(10, -317)) && (distj<=disti))
        acceptalpha = 1;
      else
        acceptalpha = pi[j]/pi[i];
   }

   if ((0 < abs(i-j)) && (abs(i-j) < M+1))
   {
      P[i][j] = acceptalpha/(2*M+1);
      P[i][i] = P[i][i] + (1-acceptalpha)/(2*M+1);
   }
   else if (abs(i-j) >= NODES-M)
   {
      P[i][j] = acceptalpha/(2*M+1);
      P[i][i] = P[i][i] + (1-acceptalpha)/(2*M+1);
   }
   else if ((M+1 <= abs(i-j)) && (abs(i-j) < NODES-M))
      P[i][j] = 0;
 }
}

/* finds expected value of h after 1 iteration and alph using
   h = 1 + dist(x,0)^HEXP + dist(y,0)^HEXP and taking the low value of
   temp since inequality holds for all x, y
   covering all pairs of x and y isn't necessary with this particular h
   since x will equal y, but it's nice to have the code in case we want
   to change h */
for (x=1; x<NODES; x++)
{
 for (y=x; y<NODES; y++)
 {
  Eh1 = 0;
  for (i=0; i<NODES; i++)
```

```
      {
        min = distance(i);
        /* adding the two sums at the same time */
        Eh1 += pow(1.0*min, HEXP)*(P[x][i]+P[y][i]);
      }
      Eh1++;
      distx = distance(x);
      disty = distance(y);
      temp = (1+pow(distx, HEXP)+pow(disty, HEXP))/Eh1;
      if (temp < alph)
        alph = temp;
    }
  }

  epsilon = 1; /* for R = only one state */

  /* finds A; we're using k0 = 1 */
  for (i=0; i<NODES; i++)
  {
    min = distance(i);
    A += pow(min, HEXP)*P[0][i];
  }
  A = A*2 + 1;

  /* finds expected value of h for the initial distribution */
  for (i=0; i<NODES; i++)
  {
    min = distance(i);
    Eh0 += pow(min, HEXP)*pi[i];
  }
  Eh0 += 2; /* since the initial state is always zero here */

  /* tries different j's to find the best bound; in our case, the best j
     will always be 1 since epsilon is 1, but it's nice to have the code
     here if other cases are to be explored */
  for (j=1; j<=ITERS; j++)
  {
    bound = pow(1-epsilon, j) +
            Eh0*pow(A, j-1)*pow(alph, -ITERS+(j-1));
    if (bound < bestbound)
      bestbound = bound;
  }

  printf ("Bound is %g\n", bestbound);
  exit(0);
}
```

```
int distance (int i)
{
 if (NODES-i < i)
   return NODES-i;
 return i;
}
```