Sensitivity of Markov chains for wireless protocols

Problem presented by

Keith Briggs

BT

Problem statement

Many wireless network protocols lead to Markov chain models with a transition matrix $P(\alpha)$ depending on some parameters α. From this the steady state distribution $\mathbf{z}(\alpha)$ can be computed, and from this the final desired quantities $q(\alpha)$, which might be for example the throughput of the protocol. To optimise q efficiently we wish to have its gradient with respect to α. There are formulas for $d\mathbf{z}/d\alpha$ in terms of $(I-P)^{\#}$ but what are the best numerical techniques for obtaining the derivatives, and are they stable? Can we also obtain the derivatives of the mixing time and mean hitting times H?

The Study Group showed that in general the techniques of sparse linear algebra and preconditioning should be used for calculating $\mathbf{z}$, H and their derivatives, and sparse eigenvalue software for mixing times, rather than any numerical computation of $(I-P)^{\#}$. For a particular example of interest (the Bianchi chain) these methods were applied and shown to be very efficient, and the same can be expected for any chain with a similar "escalator" structure.

Study Group contributors

David Allwright (Smith Institute)
Paul Dellar (Imperial College)
Jens Gravesen (Technical University of Denmark)
Jan Van Lent (University of Bath)
Rob Scheichl (University of Bath)
Maxim Zyskin (University of Bristol)

Report prepared by

David Allwright (Smith Institute)
Paul Dellar (Imperial College)

1 Introduction

Network communication protocols such as the IEEE 802.11 wireless protocol are currently best modelled as Markov chains. In these situations we have some protocol parameters α, and a transition matrix $P(\alpha)$ from which we can compute the steady state (equilibrium) distribution $\mathbf{z}(\alpha)$ and hence final desired quantities $q(\alpha)$, which might be for example the throughput of the protocol. Typically the chain will have thousands of states, and a particular example of interest is the Bianchi chain defined later. Generally we want to optimise q, perhaps subject to some constraints that also depend on the Markov chain. To do this efficiently we need the gradient of q with respect to α, and therefore need the gradient of $\mathbf{z}$ and other properties of the chain with respect to α. The matrix formulas available for this (*e.g.* in [12] and [4]) involve the so-called *fundamental matrix* $(I-P)^{\#} = (I-P+\mathbf{1}\mathbf{z}^{\mathsf{T}})^{-1} - \mathbf{1}\mathbf{z}^{\mathsf{T}}$, where $\mathbf{z}$ is the equilibrium state and $\mathbf{1}$ a vector of 1s. Then $d\mathbf{z}^{\mathsf{T}}/d\alpha = \mathbf{z}^{\mathsf{T}}\,(dP/d\alpha)\,(I-P)^{\#}$, but is this the best numerical method, and is it stable? Are there approximate gradients available which are faster and still sufficiently accurate? In some cases BT would like to do the whole calculation in computer algebra, and get a series expansion of the equilibrium $\mathbf{z}$ with respect to a parameter in P. In addition to the steady state $\mathbf{z}$, the same questions arise for the mixing time and the mean hitting times. The mixing time can here be thought of as $(|\log|\lambda_2||)^{-1}$ or $1/(1-|\lambda_2|)$ where $|\lambda_2|$ is the second largest absolute value of the eigenvalues of P (although Example 26 of [1] shows this may not always have such a close link to mixing as is sometimes claimed).

Two qualitative features that were brought to the Study Group's attention were:

- the transition matrix P is large, but *sparse*.
- the systems of linear equations to be solved are generally singular and need some additional normalisation condition, such as is provided by using the fundamental matrix $(I-P)^{\#}$.

We also note a third highly important property regarding applications of numerical linear algebra:

- the transition matrix P is *asymmetric*.

A realistic dimension for the matrix P in the Bianchi model [3] described below is 8064×8064, but on average there are only a few nonzero entries per column. Merely storing such a large matrix in dense form would require nearly 0.5 GBytes using 64-bit floating point numbers, and computing its LU factorisation takes around 80 seconds on a modern microprocessor. It is thus highly desirable to employ specialised algorithms for sparse matrices. These algorithms are generally divided between those only applicable to symmetric matrices, the most prominent being the conjugate-gradient (CG) algorithm for solving linear equations (*e.g.* [8]), and those applicable to general matrices. A similar division is present in the literature on numerical eigenvalue problems.

In the problems encountered at the Study Group, the singular matrices of dimension n had rank $n-1$. We restored each matrix to full rank by replacing one row or column to impose a suitable normalisation condition: examples of how this was done will be given in Section 3.1.

We shall also show how the derivatives of the steady state and eigenvalues with respect to the parameters in P may be computed in terms of the corresponding derivatives of P. These computations may be formulated using matrix perturbation theory, as commonly met for Hermitian matrices in the context of quantum mechanics.

This study is motivated by the Bianchi model [3] for which some quantities of interest, notably the equilibrium population, may be written down explicitly. However, the numerical techniques employed should be applicable much more widely. The key feature that we exploit is the presence of "escalator" structures in the transition matrix. Escalators correspond to deterministic increments of a backoff time counter in the communication protocol. In linear algebra terms, the escalators contribute transposed Jordan blocks to the transition matrix. The spectra of the transition matrices shown below in Figure 5 are typical of the behaviour of Jordan blocks under the addition of small perturbations. Each Jordan block has a single degenerate eigenvalue that splits into a circular pattern when perturbed. These spectra are highly undesirable for efficient performance of iterative methods like GMRES [14], but the lower triangular part of the matrix (or even just the bidiagonal matrix of the unperturbed Jordan block) makes a highly effective preconditioner. A concrete example is given below in Section 4.3.

1.1 The Bianchi chain

A particular example of interest is the Bianchi model of an RTS/CTS WAP protocol. This has parameters

$$
\begin{aligned}
W &= \text{minimum congestion window} && (1)\\
m &= \text{value such that } 2^m W \text{ is the maximum congestion window} && (2)\\
p &= \text{packet loss probability, computed from the number of users.} && (3)
\end{aligned}
$$

The states of the system form $m+1$ downward-going escalators $E_0, E_1, \ldots, E_m$ of heights $W_0, W_1, \ldots, W_m$ where $W_i = 2^i W$. The states are labelled by (i,k) where $i \in \{0,1,\ldots,m\}$ labels the backoff stage (*i.e.* which escalator), and $k \in \{0,1,\ldots,W_i-1\}$ is the backoff time counter (height on that escalator). The dynamics may be summarised by the following rules.

- From (i,k) with $k \geq 1$, move with probability one to state $(i,k-1)$, one step down E_i.
- From $(i,0)$ (the bottom of E_i) jump with probability $1-p$ to a random point on escalator E_0.
- From $(i,0)$ jump with probability p to a random point on escalator $E_{\min(i+1,m)}$.

In formulae,

$$\begin{aligned}
p(i,k;i,k-1) &= 1 \text{ for each } i \text{ and for } 1 \le k \le W_i - 1, && (4)\\
p(i,0;0,k) &= \frac{1-p}{W_0} \text{ for each } i \text{ and for } 0 \le k \le W_0 - 1, && (5)\\
p(i,0;i+1,k) &= \frac{p}{W_{i+1}} \text{ for } 0 \le i \le m-1 \text{ and for } 0 \le k \le W_{i+1} - 1, && (6)\\
p(m,0;m,k) &= \frac{p}{W_m} \text{ for } 0 \le k \le W_m - 1, && (7)
\end{aligned}$$

where we use $p(i,k;j,l)$ to denote the 1-step transition probability from (i,k) to (j,l). We can perhaps picture the state as a boy playing on this set of downwards-moving escalators, making random jumps according to these rules when he reaches the *bottom* of an escalator, but strictly obeying a sign that says "No jumping on the escalators". The system is represented schematically in the (i,k)-plane in Figure 1.

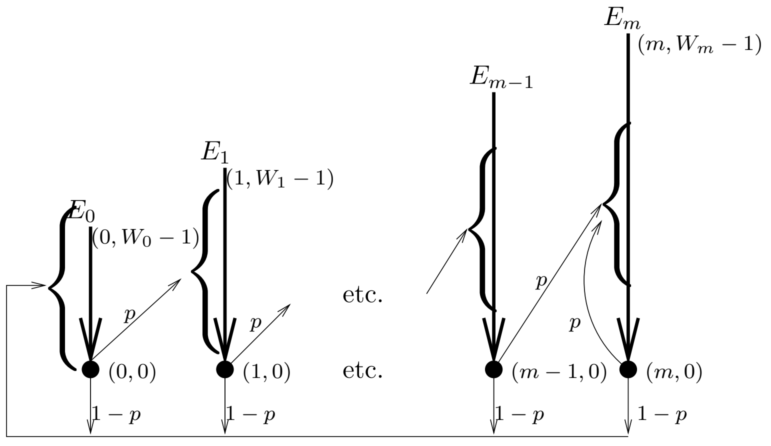


Figure 1: Schematic diagram of the Bianchi Markov chain. When the system is on an escalator E_i it descends deterministically at unit rate. When it is at the bottom of E_i it jumps with probability $1-p$ to a random state of E_0, and with probability p to a random state of E_{i+1} (or E_m if $i = m$).

There are $n = (2^{m+1} - 1)W$ states, and the case $m = 3$, $W = 32$ has 480 states, about the smallest reasonable model. But m might in practice be up to 5, and W up to 128, in which case $n = 8064$. The values of p of interest are generally small, perhaps up to 0.05 but with 0.01 as more typical. To arrange the states in a linear order we take first E_0 with k increasing, then E_1 *etc*, so (i,k) maps to $\alpha = 1 + k + (2^i - 1)W$ with $1 \le \alpha \le n$.

Figure 2 shows the sparsity pattern of the transition matrix for two different choices of m and W: the black entries are those that are positive, the others are 0. Both choices give much smaller matrices than would be realistic. Choosing $m = 5$ and $W = 128$ gives an 8064×8064 matrix that would be difficult to plot.

The diagram in Figure 3 shows the transitions for $m = 2$, $W = 4$, and Figure 4 is for $m = 3$, $W = 6$. The "escalators" visible in these graphs and in the sparsity pattern diagrams are a typical structure, whatever the values of W and m.

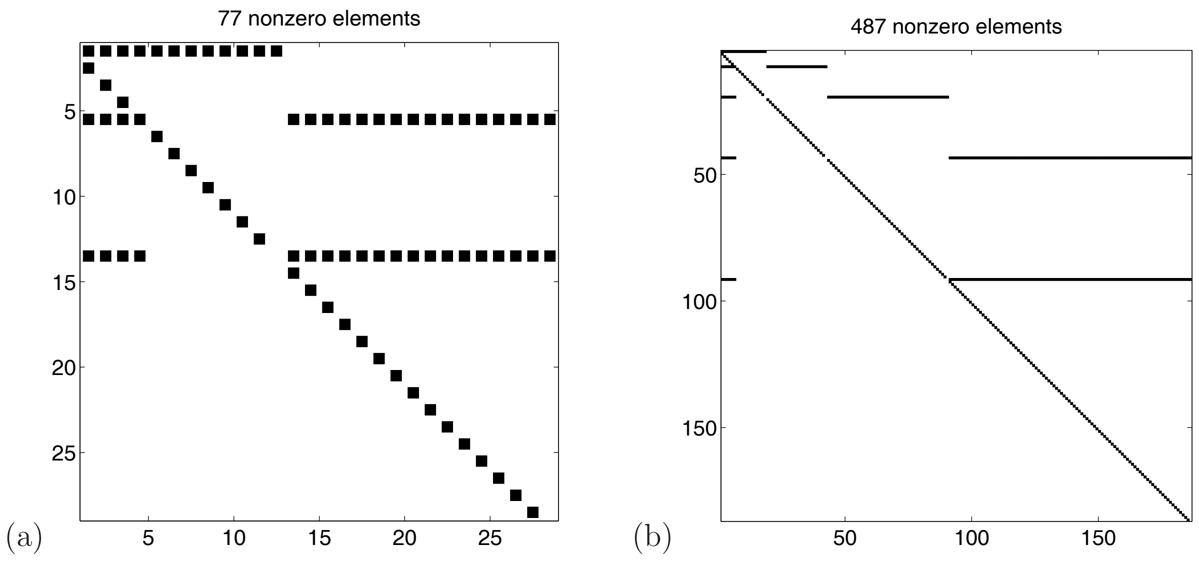


Figure 2: Nonzero elements of the Bianchi transition matrix for (a) $m = 2$ and $W = 4$, and (b) $m = 4$ and $W = 6$.

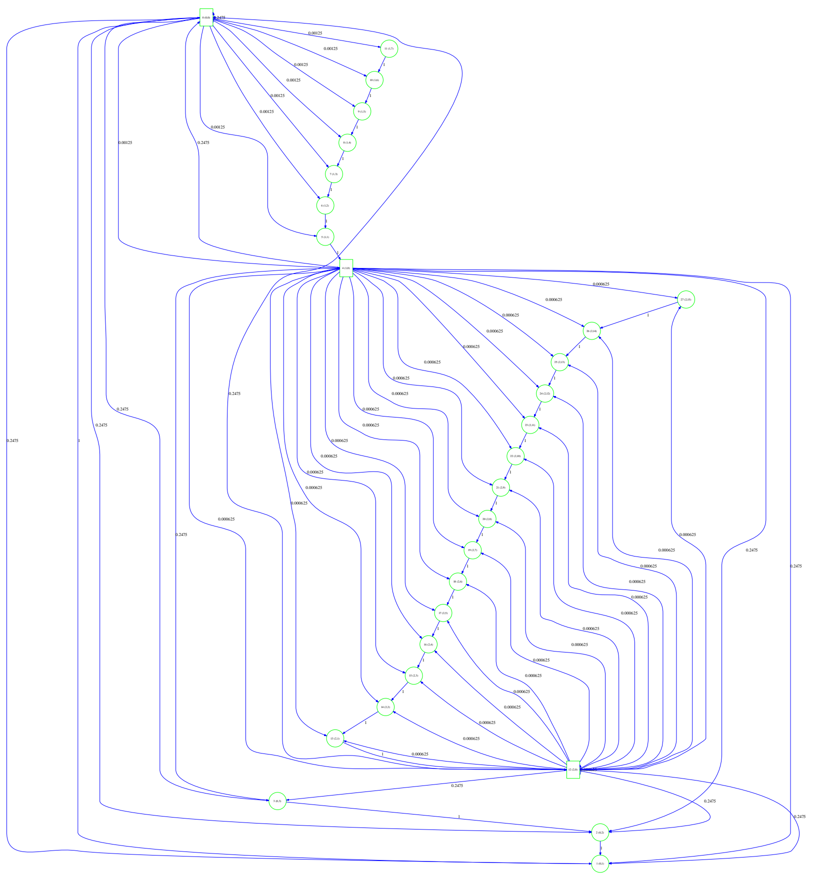


Figure 3: State graph for the Bianchi model $m = 2$, $W = 4$.

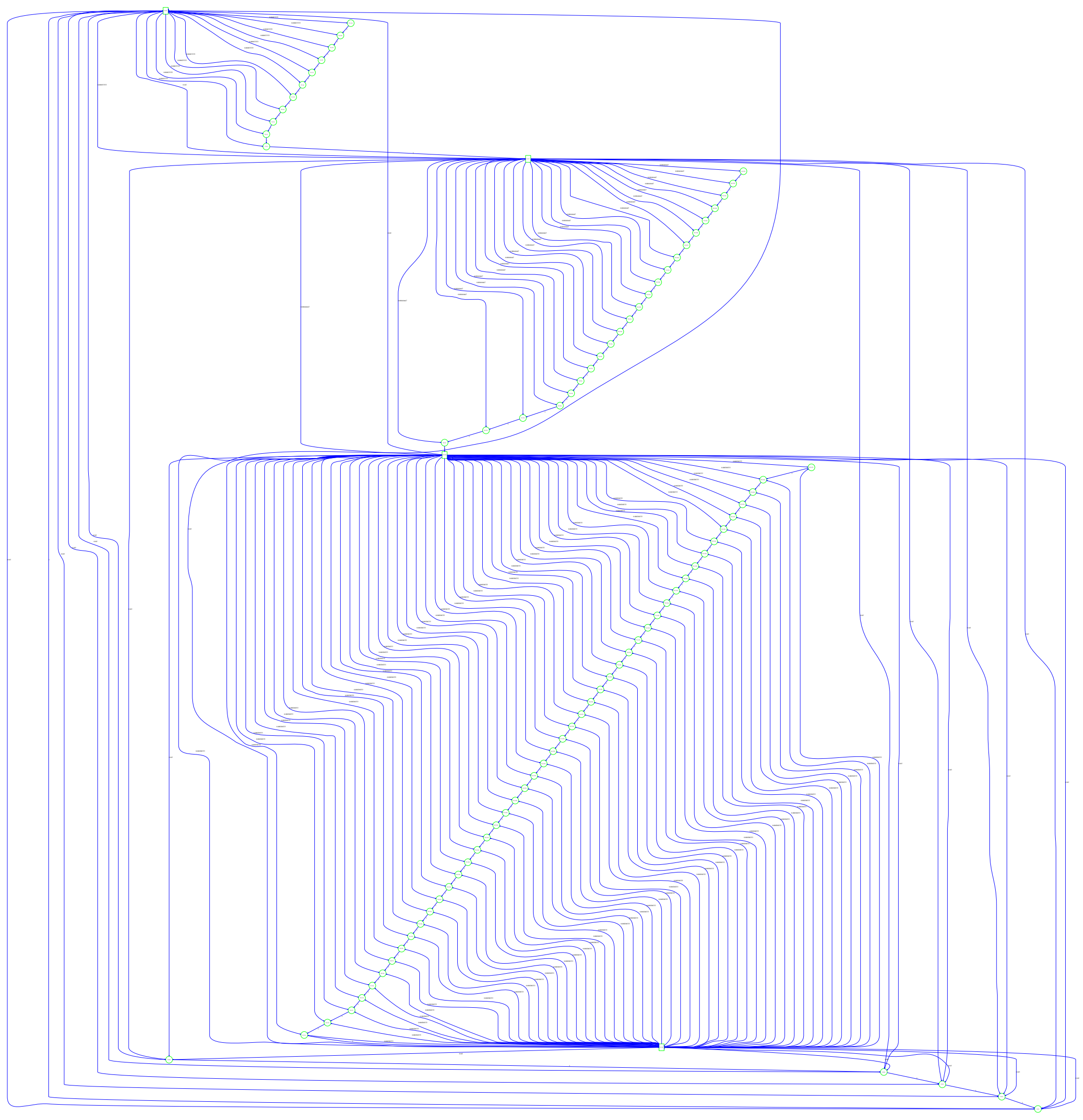


Figure 4: State graph for the Bianchi model, $m = 3$, $W = 6$.

Figure 5 shows the distribution of the eigenvalues of the transition matrix for the same two choices of m and W as Figure 2. As for any transition matrix, there is an eigenvalue $\lambda = 1$ whose left eigenvector gives the equilibrium population, as described later. All other eigenvalues lie inside the unit circle as shown. The distribution of eigenvalues on circles is typical of perturbed Jordan blocks as shown in Appendix B.

2 General notation and theory

For a discrete-time Markov chain with states denoted by i, j, *etc*, we shall let

$$p_{ij} = \mathbb{P}\mathrm{r}(X_{t+1} = j | X_t = i) \tag{8}$$

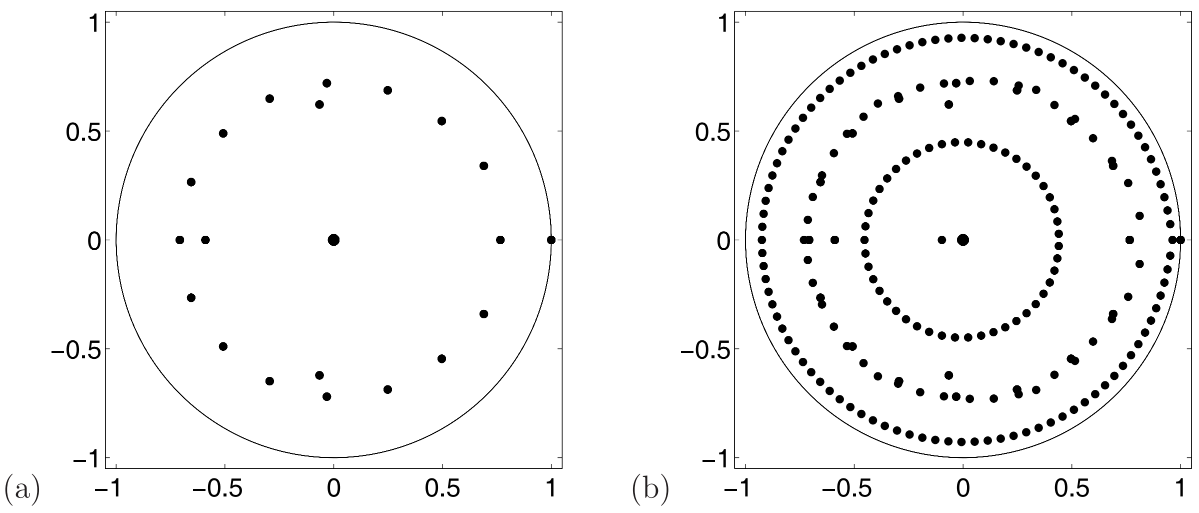


Figure 5: Complex eigenvalues of the Bianchi transition matrix for (a) $m = 2$ and $W = 4$, and (b) $m = 4$ and $W = 6$. The unit circle is also shown.

be the 1-step transition probability from i to j, and n be the number of states.[1] We also think of the states as the vertices of a directed graph, in which there is an edge from i to j if and only if $p_{ij} > 0$, and these are the graphs illustrated in Figures 3 and 4. We shall assume that the states form a single closed class, *i.e.* for any states i and j there is a path from i to j in that graph. We shall also assume that the chain is aperiodic, *i.e.* it is not possible to partition the states into $r \geq 2$ sets A_1, A_2, …, A_r such that from each A_s you can only get to A_{s+1}, and from A_r only to A_1. A sufficient condition for this is that some $p_{ii} > 0$, or that from some i to some j there are paths of coprime lengths. When these conditions hold, the Markov chain has steady state probabilities

$$z_i = \lim_{t\to\infty} \left(\mathbb{P}\mathrm{r}(X_t = i)\right), \tag{9}$$

and the system is ergodic in the sense that z_i is also the long-term proportion of the time that is spent in state i. Coming now to the matrix formulation, the matrix $P = (p_{ij})$ has $P\mathbf{1} = \mathbf{1}$ where $\mathbf{1}$ is a column vector with every entry 1. So $\lambda_1 = 1$ is an eigenvalue of P, and under the conditions we have assumed, each other eigenvalue lies in $|\lambda| < 1$. When the initial probabilities of the system being in state i form a row vector $\mathbf{p}_0^\mathsf{T}$, the probabilities at times $1, 2, \ldots$ are $\mathbf{p}_0^\mathsf{T}P$, $\mathbf{p}_0^\mathsf{T}P^2$ *etc.* Since $\mathbf{z}^\mathsf{T} = (z_1, z_2, \ldots, z_n)$ is the steady-state distribution, it therefore obeys $\mathbf{z}^\mathsf{T}P = \mathbf{z}^\mathsf{T}$, so it is the left eigenvector for the eigenvalue $\lambda_1 = 1$. The normalisation is fixed by the fact that it is a probability distribution over the states and so $\sum z_i = \mathbf{z}^\mathsf{T}\mathbf{1} = 1$. So from the viewpoint of linear algebra, $\mathbf{z}^\mathsf{T}$ is determined by $\mathbf{z}^\mathsf{T} = \mathbf{z}^\mathsf{T}P$ together with the condition $\mathbf{z}^\mathsf{T}\mathbf{1} = 1$ that resolves the ambiguity caused by 1 being an eigenvalue of P.

In general if the eigenvalues λ_i of P are distinct then there will be left eigenvectors $\mathbf{z}_i^\mathsf{T}$

[1]Of course when we come the the particular example of the Bianchi chain, this i and j will each have to be replaced by an index α that encodes the *pair* of indices (i, k) in the Bianchi description.

and right eigenvectors $\mathbf{v}_i$ such that

$$\mathbf{z}_i^{\mathsf{T}} P = \lambda_i \mathbf{z}_i^{\mathsf{T}}, \qquad P\mathbf{v}_i = \lambda_i \mathbf{v}_i, \qquad \mathbf{z}_i^{\mathsf{T}} \mathbf{v}_j = \delta_{ij}, \tag{10}$$

with $\lambda_1 = 1$, $\mathbf{z}_1^{\mathsf{T}} = \mathbf{z}^{\mathsf{T}}$, $\mathbf{v}_1 = \mathbf{1}$, and then

$$P = \sum_{i=1}^{n} \lambda_i v_i z_i^T = \mathbf{1}\mathbf{z}^{\mathsf{T}} + \sum_{i=2}^{n} \lambda_i \mathbf{v}_i \mathbf{z}_i^{\mathsf{T}}. \tag{11}$$

Any initial probability distribution over the states, $\mathbf{p}_0^{\mathsf{T}}$, will then have the form $\mathbf{p}_0^{\mathsf{T}} = \mathbf{z}^{\mathsf{T}} + \sum_{i=2}^{n} \xi_i \mathbf{z}_i^{\mathsf{T}}$ where $\xi_i = \mathbf{p}_0^{\mathsf{T}} \mathbf{v}_i$, and after r steps of the Markov process it will become

$$\mathbf{p}_0^{\mathsf{T}} P^r = \mathbf{z}^{\mathsf{T}} + \sum_{i=2}^{n} \lambda_i^r \xi_i \mathbf{z}_i^{\mathsf{T}}. \tag{12}$$

So this tends to $\mathbf{z}^{\mathsf{T}}$ as $r \to \infty$ and the rate of convergence is governed by the subdominant eigenvalue, *i.e.* $|\lambda_2|$ if we arrange the eigenvalues by decreasing magnitude,

$$\lambda_1 = 1 > |\lambda_2| \geq |\lambda_3| \geq \ldots \qquad . \tag{13}$$

So $\mathbf{z}_2^{\mathsf{T}}$ is the most persistent (slowest decaying) non-equilibrium behaviour.

In the circumstances we are considering, the mean time to reach state i starting from any other state j is finite, called the mean hitting time (or just the hitting time) and we denote it by

$$H_{ji} = \mathbb{E}\mathrm{x}(\text{least } t \text{ such that } X_t = i | X_0 = j), \qquad (j \neq i). \tag{14}$$

The equations for these come from considering how we get from j to i. We must first take 1 step, and then if we have gone to some $k \neq i$ the mean number of steps from there to i is H_{ki}, so

$$H_{ji} = 1 + \sum_{k \neq i} p_{jk} H_{ki} \qquad (j \neq i). \tag{15}$$

For fixed i these are $n-1$ equations for the $n-1$ unknowns H_{ji} and uniquely determine them. The matrix involved is $I - P$ with row and column i deleted. A further property of the hitting times is that

$$\frac{1}{z_i} = 1 + \sum_{k \neq i} p_{ik} H_{ki}, \tag{16}$$

since $1/z_i$ is the mean recurrence time for state i, *i.e.* the mean time between successive visits to i, which we can write down by the same argument that led to (15). If we define a matrix H_0 to be H_{ij} off the diagonal and 0 on the diagonal then these two equations are combined in

$$H_0 + \mathrm{diag}(1/z_i) = J + PH_0, \tag{17}$$

where every entry of J is 1: *off* the diagonal this is (15), and *on* the diagonal it is (16). Although (17) alone does not determine H_0 because $I - P$ is singular, the additional condition that H_0 has zero diagonal elements resolves that ambiguity.

2.1 Derivatives with respect to parameters

When the matrix P depends on a vector of parameters α in a differentiable way, we can find the derivatives of the various properties of the chain by differentiating the defining equations. So from $\mathbf{z}^\mathsf{T} P = \mathbf{z}^\mathsf{T}$ and $\mathbf{z}^\mathsf{T}\mathbf{1} = 1$ we have

$$\left(\mathbf{z}^\mathsf{T}\right)' P + \mathbf{z}^\mathsf{T} P' = \left(\mathbf{z}^\mathsf{T}\right)', \quad \left(\mathbf{z}^\mathsf{T}\right)' \mathbf{1} = 0, \tag{18}$$

where $'$ denotes differentiation with respect to α. So, like $\mathbf{z}^\mathsf{T}$ itself, the vector $\left(\mathbf{z}^\mathsf{T}\right)'$ obeys a system of linear equations with matrix $I - P$, together with a condition that resolves the ambiguity caused by 1 being an eigenvalue of P.

For differentiating the subdominant eigenvalue λ_2, we deal first with the case where λ_2 is real with $|\lambda_2| > |\lambda_3|$. Then λ_2 is an isolated eigenvalue so its left and right eigenvectors are well-defined and from differentiating the conditions (10) for $i = 2$ we find that

$$\lambda_2' = \mathbf{z}_2^\mathsf{T} P' \mathbf{v}_2. \tag{19}$$

Equally, if λ_2 and λ_3 are a complex conjugate pair, with $|\lambda_2| = |\lambda_3| > |\lambda_4|$ then λ_2 is isolated and the same result holds, and of course, $\mathbf{z}_3^\mathsf{T}$, $\mathbf{v}_3$ and λ_3' are just the conjugates of $\mathbf{z}_2^\mathsf{T}$, $\mathbf{v}_2$ and λ_2'. However, when λ_2 is real with $|\lambda_2| = |\lambda_3|$ then there is generally a failure of differentiability. Either $\lambda_2 = \lambda_3$ in which case the eigenvectors are either not defined or not unique and perturbing P by $O(\epsilon)$ splits the eigenvalues by $O(\sqrt{\epsilon})$. Or $\lambda_2 \neq \lambda_3$ in which case even though the eigenvalues of P vary continuously there is a discontinuous switch in which of them is subdominant. The same kind of failure of differentiability will happen when λ_2 is complex and $|\lambda_2| = |\lambda_4|$. These cases are not generic as far as P is concerned. However, when one tries to solve an optimisation problem where the objective function or constraints may involve λ_2, then this kind of difficulty can easily arise because the optimum may well be at a point of non-differentiability of $|\lambda_2|$. For instance if one tries to minimise $M = |\lambda_2|$ subject to some constraints then one can easily envisage this optimum occurring for a configuration where P has several eigenvalues on the circle $|\lambda| = M$, but where any feasible perturbation causes some of those eigenvalues to move into $|\lambda| > M$. So the fact that $|\lambda_2|$ is not everywhere differentiable could be of more importance than its non-genericity might suggest.

To differentiate the hitting times, we differentiate (17) and see

$$H_0' - \operatorname{diag}(z_i'/z_i^2) = P' H_0 + P H_0', \tag{20}$$

so again H_0' obeys a system of linear equations with matrix $I - P$ and with the condition of zero diagonal elements to resolve the ambiguity caused by 1 being an eigenvalue of P.

Whether these equations are solved analytically or numerically, these formulae are the key to calculating the derivatives.

3 Numerical methods

As explained in the introduction, our aim here is to describe numerical methods that will be effective for this kind of problem in general, with a large sparse transition matrix

P having an escalator structure. The particular illustrations will use the Bianchi chain as the example.

3.1 Equilibrium population

The equilibrium population, $\mathbf{z}_1^{\mathsf{T}}$ in our notation, is given by

$$\mathbf{z}_1^{\mathsf{T}}(I - P) = 0\,, \tag{21}$$

and to define a unique solution we impose the condition $\mathbf{z}_1^{\mathsf{T}}\mathbf{1} = 1$ that makes $\mathbf{z}_1^{\mathsf{T}}$ a normalised probability distribution.

When the states of the Markov chain form a single closed class, the matrix $I - P$ is rank-deficient by 1, and we found experimentally that replacing the leftmost column of the matrix by a column of ones $(1, \ldots, 1)^{\mathsf{T}}$ created a matrix with full rank. We chose the leftmost column so that this dense column lies in the lower triangular part of the matrix. This is convenient when using the lower triangular part of the matrix as a preconditioner for iterative methods, as in Section 4.3 below, but in principle any other column could be chosen. One could also add 1 to each entry in the original column of $I - P$ instead of replacing the existing entries with 1s. The resulting linear system, written in partitioned form, is

$$\mathbf{z}_1^{\mathsf{T}}\begin{pmatrix} 1 & \\ \vdots & [I-P]_{2\ldots n} \\ 1 & \end{pmatrix} = (1, 0, \ldots, 0), \tag{22}$$

where $[I - P]_{2\ldots n}$ denotes columns 2 to n of the $n \times n$ matrix $I - P$. The transpose of equation (22) may be treated using numerical methods for solving systems of full rank in the standard form $A\,\mathbf{x} = \mathbf{b}$, as described in Section 4 below.

An alternative approach to finding eigenvectors is to solve the obvious (notionally singular) linear system instead of changing one row or column to impose a normalisation condition. There are matrices where the "eigenvector" computed by changing a row or column varies wildly depending upon precisely which row or column is chosen. (We chose column 1 to keep the triangular structure.) By contrast, a naive solution of the notionally singular system, possibly changed by O(machine epsilon) to avoid a genuine division by zero, gives the eigenvector to very high precision. The notional singularity of the matrix makes the coefficient in front of the eigenvector highly inaccurate, but that is of no interest anyway. This is what the MATLAB `eigs` routine does internally to find eigenvectors.

The hitting times are also found by solving the linear algebra problem (17) with the same matrix $I - P$.

4 Numerical solution of equilibrium equations

As noted before, realistic-sized problems involve large, sparse, and asymmetric matrices. Since matrices with these properties arise in many applications, notably from discrete approximations to partial differential equations, many numerical methods have been developed for solving linear systems involving large, sparse matrices that perform better than standard methods like LU factorisation. In the Study Group we pursued three different approaches — two that are applicable to generic matrices, and one that makes greater use of the special structure of the Bianchi matrices. These approaches are described in the subsections below, while results and timings will be presented in the conclusions in Section 8.

4.1 Sparse direct factorisation method

The standard numerical method for solving a system of linear equations, conventionally written as $A\mathbf{x} = \mathbf{b}$, is by factorising the matrix A into the product of a lower triangular matrix L and an upper triangular matrix U. In other words, one writes $A = LU$, which inspires the name "LU factorisation". The original linear system then decomposes into two separate linear systems,

$$A\mathbf{x} = \mathbf{b} \Leftrightarrow LU\mathbf{x} = \mathbf{b} \Leftrightarrow L\mathbf{y} = \mathbf{b} \text{ and } U\mathbf{x} = \mathbf{y}, \tag{23}$$

whose solutions $\mathbf{y}$ and $\mathbf{x}$ may be found by forward and backwards substitution respectively. The factors are commonly rendered unique by requiring L to have unit entries on its diagonal. Efficient routines to compute the factors L and U for generic matrices may be found in packages such as LAPACK [2]. LU factorisation is a "direct method" for solving linear equations, because one would find the exact solution after a finite number of computations, in the absence of round-off error associated with finite precision arithmetic.

Even if the original matrix A is sparse, with a large proportion of zero entries, the computed factors L and U typically contain many more nonzero entries than A. This phenomenon is known as "fill-in". However, the same system of linear equations $A\mathbf{x} = \mathbf{b}$ may be rewritten in many different ways by arranging the individual equations in different orders. Each rearrangement is equivalent to a permutation of the elements of the vectors $\mathbf{x}$ and $\mathbf{b}$, and of the rows and/or columns of the matrix A. Various algorithms have been developed to find permutations that are likely to reduce the number of nonzero entries in the factors, by analysing the pattern of nonzero entries in the original matrix. These algorithms often use techniques from the theories of graphs and trees.

MATLAB has some sparse matrix capabilities [7] including an interface to the package UMFPACK [6] that solves large, sparse linear systems by a sparse LU factorisation. Figure 6 shows the results of computing the sparse factors L and U for two Bianchi matrices, those with $m = 2$ and $W = 4$, and $m = 4$ and $W = 6$. For each matrix, the figure shows first the permuted matrix, then the lower factor L, and finally the upper factor U. It is remarkable that the factors L and U share the high degree of sparsity of the original matrix P. Nonzero elements in all three matrices (P, L, U) are confined to a narrow band on the diagonal, plus a few dense rows or columns. Table 1 shows that this trend continues to much larger Bianchi matrices, with L containing roughly 2 nonzero elements per row, and U containing roughly 1.5 nonzero elements per row. By contrast,

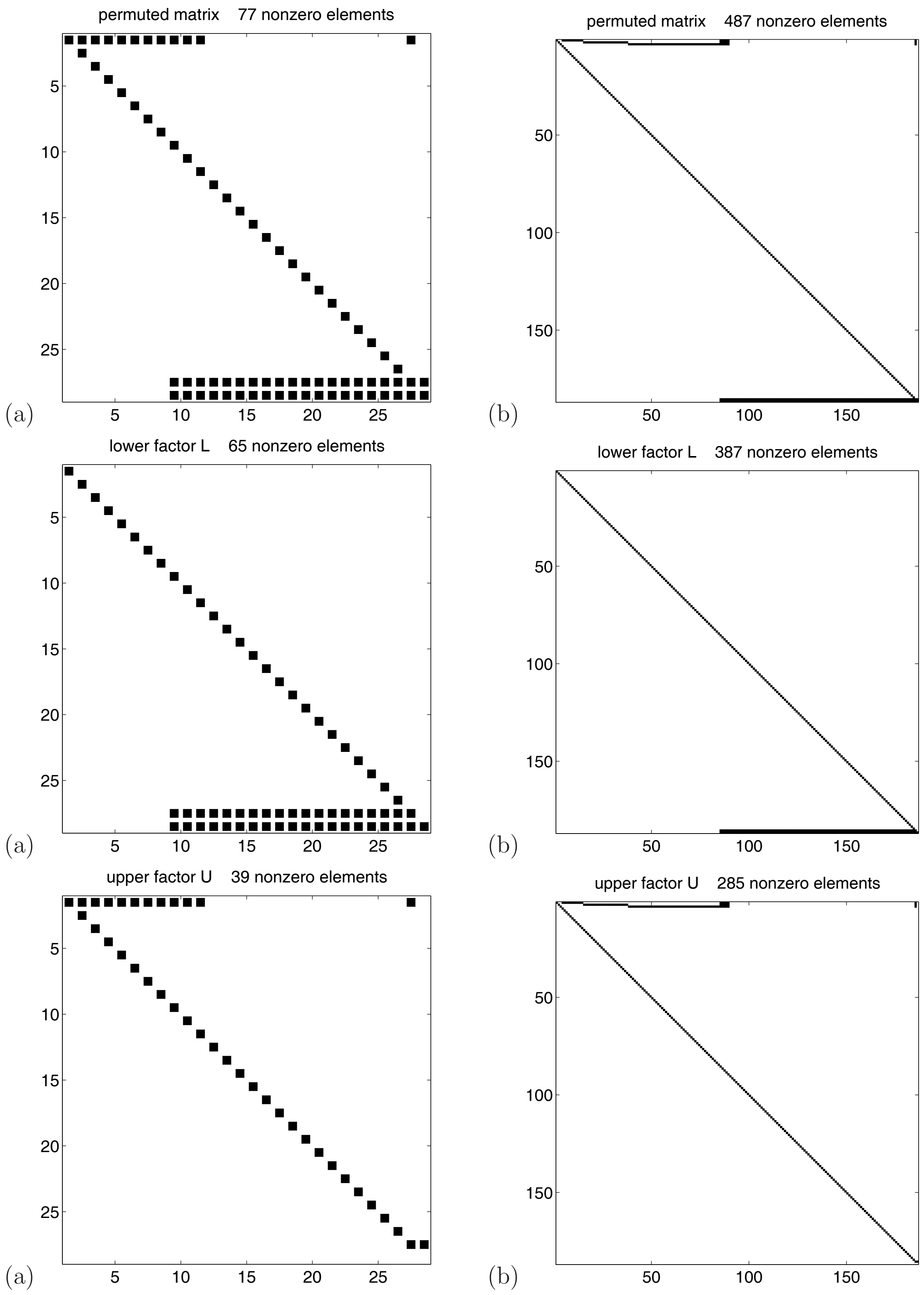


Figure 6: Sparse LU factors of the permuted Bianchi transition matrices for (a) $m = 2$ and $W = 4$, and (b) $m = 4$ and $W = 6$.

Parameters	n	nnz(P)		nnz(L)		nnz(U)	
$m = 2$, $W = 4$	28	77	(2.75n)	65	(2.32n)	39	(1.39n)
$m = 4$, $W = 64$	186	487	(2.62n)	387	(2.08n)	285	(1.53n)
$m = 5$, $W = 128$	8064	20858	(2.59n)	16509	(2.05n)	12412	(1.54n)
$m = 8$, $W = 256$	130816	329207	(2.52n)	262397	(2.01n)	197625	(1.51n)

Table 1: Sparsity properties for LU factors of permuted Bianchi matrices P. The number of nonzero elements (nnz) for P, the factors L and U, and the average number of nonzero elements per row. These averages, or the sparsity fractions, appear roughly constant as the matrix dimension n increases.

the factors of matrices arising as discrete approximations to elliptic partial differential equations typically contain many more nonzero entries than their product.

Note for an efficient implementation that many sparse matrix packages have the facility to perform the structural analysis, permutation, and symbolic factorisation once for a given pattern of non-zero entries. A recalculation of just the numerical coefficients in the factors after changing parameters in the original matrix will then be much faster than performing the whole sparse factorisation from scratch each time.

4.2 Block direct factorisation

The previous approach using a sparse direct method is applicable, at least in principle, to arbitrary matrices. The sparse matrix package determines a reordering of the matrix designed to lead to sparse factors L and U. For the particular case of the Bianchi matrices, a natural permutation collects the $m+1$ dense rows together at the top. After this permutation of a Bianchi matrix P, the matrix appearing in equation (22) for the equilibrium population,

$$A = \begin{pmatrix} 1 & \\ \vdots & [I-P]_{2\ldots n} \\ 1 & \end{pmatrix} \tag{24}$$

has the block structure illustrated in Figure 7,

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \tag{25}$$

where A_{11} is a small $(m+1) \times (m+1)$ dense matrix. The much larger $(n-m-1) \times (n-m-1)$ matrix A_{22} is bidiagonal, and thus easily invertible. The matrix A_{12} is quite dense, while the matrix A_{21} below the diagonal contains only a column of 1s from the normalisation, plus an additional m nonzero elements.

The bidiagonal structure of A_{22} motivates a block factorisation of the matrix A according to

$$A = UL \quad \text{or} \quad \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} I & A_{12}A_{22}^{-1} \\ Z & I \end{pmatrix} \begin{pmatrix} S & Z \\ A_{21} & A_{22} \end{pmatrix}, \tag{26}$$

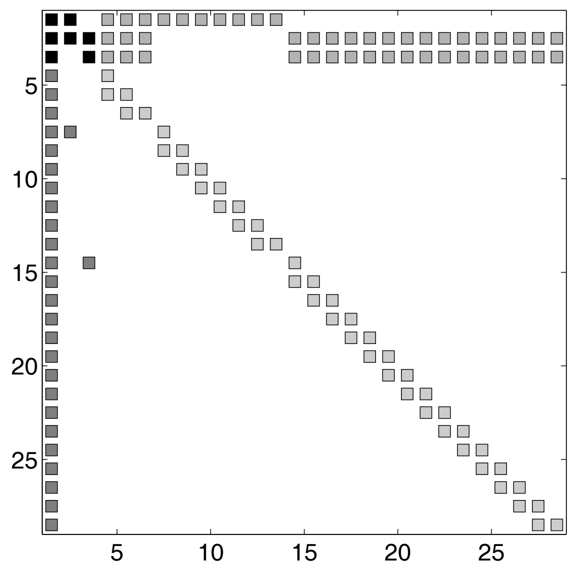
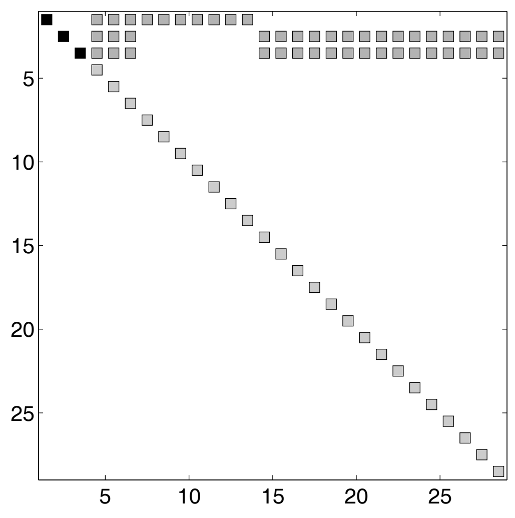


Figure 7: Block structure of the modified and permuted Bianchi transition matrix for $m = 2$ and $W = 4$.

(*U*) 

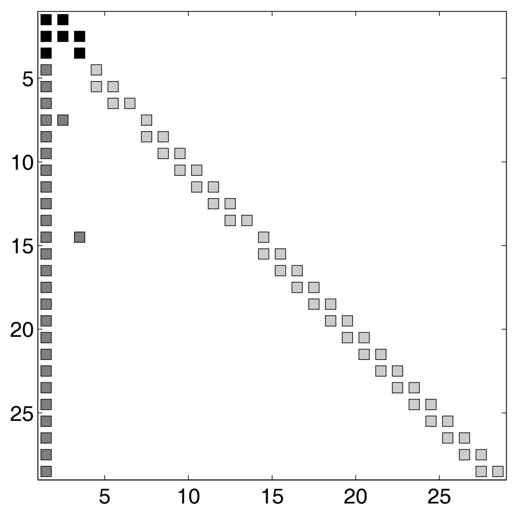
(*L*) 

Figure 8: Block factors U and L of the modified and permuted Bianchi transition matrix for $m = 2$ and $W = 4$.

where the $(m+1)\times(m+1)$ matrix S is the Schur complement of A_{22},

$$S = A_{11} - A_{12}\,A_{22}^{-1}\,A_{21} \tag{27}$$

and Z represents matrices of zeros. The matrices U and L are illustrated in Figure 8. The matrix L is only block lower triangular, not strictly lower triangular, due to the dense block S in the upper left corner.

A half-way step towards the solution of the linear system $\mathbf{x}^\mathsf{T} A = \mathbf{b}^\mathsf{T}$ may be achieved by writing $\mathbf{y}^\mathsf{T} L = \mathbf{b}^\mathsf{T}$, or in block form

$$\left(\mathbf{b}_1^\mathsf{T}, \mathbf{b}_2^\mathsf{T}\right) = \left(\mathbf{y}_1^\mathsf{T}, \mathbf{y}_2^\mathsf{T}\right)\begin{pmatrix} S & Z \\ A_{21} & A_{22}\end{pmatrix}. \tag{28}$$

Here $\mathbf{b}^\mathsf{T} = (\mathbf{b}_1^\mathsf{T}, \mathbf{b}_2^\mathsf{T})$ denotes a partitioning of the vector $\mathbf{b}^\mathsf{T}$ into $m+1$ and $n-m-1$ elements respectively, and similarly for $\mathbf{y}^\mathsf{T} = (\mathbf{y}_1^\mathsf{T}, \mathbf{y}_2^\mathsf{T})$. These partitionings correspond to the earlier partitioning of the matrix A into blocks in (25). Simplifying the right hand side of (28) gives

$$\mathbf{b}_2^\mathsf{T} = \mathbf{y}_2^\mathsf{T} A_{22}, \quad \mathbf{b}_1^\mathsf{T} = \mathbf{y}_1^\mathsf{T} S + \mathbf{y}_2^\mathsf{T} A_{21}. \tag{29}$$

The matrix A_{22} is large but bidiagonal. Computing $\mathbf{y}_2$ therefore requires only $O(n-m-1)$ operations to solve the linear system involving A_{22} by substitution. Computing $\mathbf{y}_1$ then requires a multiplication of $\mathbf{y}_2^\mathsf{T}$ by A_{21}, followed by the solution of a small $(m+1)\times(m+1)$ linear system involving S. The solution $\mathbf{x}$ follows easily from $\mathbf{y}$ using the block inverse of U,

$$\mathbf{x}^\mathsf{T} = \mathbf{y}^\mathsf{T} U^{-1} = \left(\mathbf{y}_1^\mathsf{T}, \mathbf{y}_2^\mathsf{T}\right)\begin{pmatrix} I & -A_{12}\,A_{22}^{-1} \\ Z & I\end{pmatrix} = \left(\mathbf{y}_1^\mathsf{T}, \mathbf{y}_2^\mathsf{T} - \mathbf{y}_1^\mathsf{T} A_{12}\,A_{22}^{-1}\right). \tag{30}$$

Computation of $\mathbf{y}_1^\mathsf{T} A_{12}\,A_{22}^{-1}$ requires only a matrix multiplication by A_{12}, followed by another solution of a bidiagonal system involving A_{22}.

4.3 An iterative method: preconditioned GMRES

The previous two approaches are "direct methods". They perform some fixed amount of computation that yields the exact solution, at least in exact arithmetic. For many problems it is preferable to adopt an iterative approach that yields successively better approximations to the exact solution with each iteration, halting as soon as one obtains a sufficiently accurate approximation. For example, it seems reasonable to be content with an approximate solution whose accuracy is comparable with that of the underlying floating point arithmetic, a relative error of about 10^{-15}.

GMRES (generalised minimum residual) by Saad and Schultz [14] is one of a family of iterative methods for solving linear systems that finds successive approximate solutions $\mathbf{x}_0, \mathbf{x}_1, \ldots$ to the linear system $A\mathbf{x} = \mathbf{b}$ in the form of polynomials in the matrix A multiplying the right hand side vector $\mathbf{b}$. Thus $\mathbf{x}_m = \mathsf{P}_m(A)\mathbf{b}$, where P_m is a polynomial of degree m. The $\mathbf{x}_m$ are therefore elements of the successive Krylov spaces $\mathcal{K}_m$ defined by

$$\mathcal{K}_m = \operatorname{span}\{\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \ldots, A^{m-1}\mathbf{b}\} \tag{31}$$

with the property that $\mathcal{K}_m \subseteq \mathcal{K}_{m+1}$ for all $m \geq 1$. This family of methods, often called Krylov space methods, typically converge to an adequate solution in far fewer iterations than the classical methods like Jacobi or Gauss–Seidel iteration.

In GMRES each $\mathbf{x}_m$ is chosen to be the element of $\mathcal{K}_m$ that minimises the ℓ_2 norm of the residual $\mathbf{r}_m = A\,\mathbf{x}_m - \mathbf{b}$. By constructing orthonormal bases of the successive Krylov spaces, finding each minimising element $\mathbf{x}_m$ reduces to one further multiplication of a vector by the matrix A, followed by the solution of an $m \times m$ Hessenberg system of linear equations. (The nonzero elements of a Hessenberg matrix are confined to the upper triangular part and the first sub-diagonal.) Since each Krylov space $\mathcal{K}_m$ contains its predecessors $\mathcal{K}_{m-1}, \mathcal{K}_{m-2}, \ldots$, the sequence of residuals is guaranteed to be non-increasing if the computations are performed in exact arithmetic.

GMRES is also guaranteed to find the exact solution after a number of iterations equal to the dimension of the matrix ($m = n$) again assuming exact arithmetic. However, one hopes to find an acceptable approximate solution in far fewer iterations, or when $m \ll n$. A good heuristic for the number of necessary iterations is the number of distinct eigenvalues, or eigenvalue clusters, of the matrix.

The eigenvalues of the Bianchi matrices are distributed on circles, as shown in Fig. 5, which is not encouraging for the application of Krylov space methods. Indeed, we shall find that convergence of the bare GMRES algorithm is exceedingly slow, taking even longer than a direct solution of the linear system by dense LU factorisation.

However, the situation may be salvaged by replacing the original linear system $A\mathbf{x} = \mathbf{b}$ with the equivalent system

$$(M^{-1}A)\,\mathbf{x} = M^{-1}\mathbf{b}, \tag{32}$$

where in principle M may be any invertible matrix. As suggested by the parentheses in (32), the GMRES algorithm may be applied to the product matrix $M^{-1}A$ instead of to A alone. Convergence will be much more rapid if a matrix M, called the preconditioner (*e.g.* [13, 8, 15]) can be found that satisfies the following two properties:

- linear systems $M\mathbf{y} = \mathbf{z}$ are relatively efficient to solve (equivalently, $\mathbf{y} = M^{-1}\mathbf{z}$ is easy to compute);
- M approximates A in the sense that $M^{-1}A$ has a more tightly clustered spectrum than A.

These two properties conflict, since $M = I$ satisfies the first property perfectly, yet offers no help towards satisfying the second property. Conversely, $M = A$ satisfies the second property perfectly, yet if linear systems involving A were easy to solve there would be no need for GMRES.

For the Bianchi matrices, we found that the lower triangular part of the original matrix made an extremely effective preconditioner. A linear system with a lower triangular matrix is extremely easy to solve by back substitution, and the preconditioned GMRES then converged to machine accuracy in just a few iterations, as shown in Figure 9. This behaviour is a consequence of the Bianchi matrices' structure as a collection of

Jordan blocks, one per escalator, coupled by a small number of dense rows. In fact, just the lower bidiagonal part of the matrix would be sufficient, but in our MATLAB implementation there was only a very small improvement in performance over using the full lower triangular part, despite the bidiagonal preconditioner having only 2/3 the number of nonzero elements. GMRES stagnates without a preconditioner at all, making only tiny reductions in the residual with successive iterations, as shown by the almost horizontal dotted line in Figure 9.

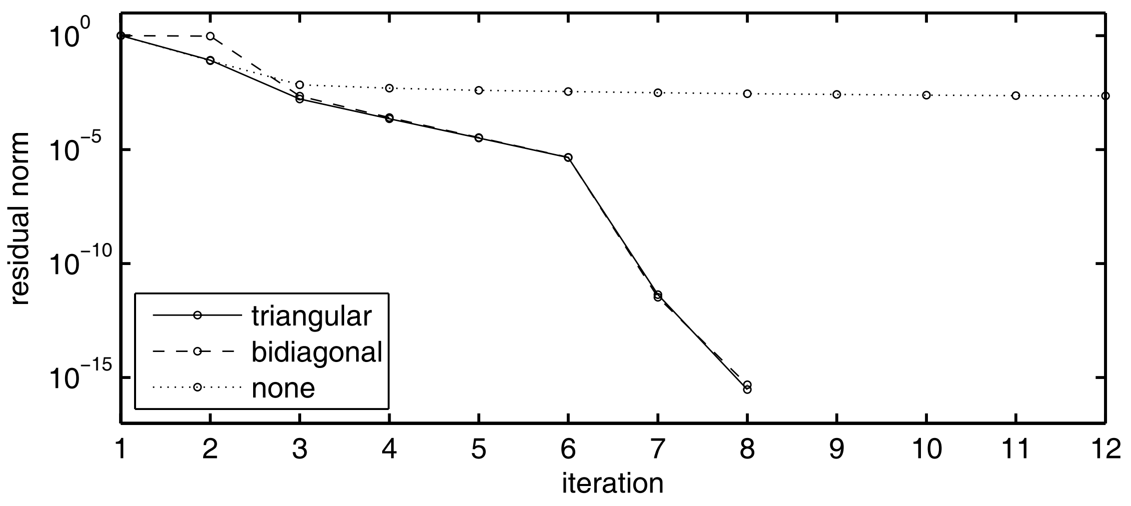


Figure 9: Performance of GMRES for computing the equilibrium population of the Bianchi matrix with $m = 5$ and $W = 128$. The residual norm at each iteration is shown for the algorithm using a lower triangular preconditioner, a lower bidiagonal preconditioner, and no preconditioner.

A Jordan block has a single degenerate eigenvalue with multiplicity equal to the dimension of the block. The degenerate eigenvalue splits into a circle of single eigenvalues under the perturbation caused by the dense rows of the Bianchi matrix (recall Fig. 5). Using the unperturbed Jordan block (a bidiagonal matrix) as a preconditioner collapses the circle of eigenvalues onto the unit eigenvalue, except for a single distinct eigenvalue that is displaced by the perturbation. The mathematical details are given in Appendix B.

5 Numerical eigenvalues: mixing time estimates

As explained in Section 2, estimating how quickly the population approaches equilibrium depends on finding the eigenvalues of P, and in particular the subdominant eigenvalue defined by (13).

There is no "direct method" for computing eigenvalues, since computing the eigenvalues of an $n \times n$ matrix is equivalent to finding the roots of the corresponding characteristic polynomial of degree n. No finite computation can determine the roots of an arbitrary quintic or higher polynomial, so finding the eigenvalues of even a 5×5 matrix requires an iterative procedure.

QR iteration, or the QR algorithm, is the standard technique for computing eigenvalues. An arbitrary matrix A may be expressed as the product of an orthogonal matrix Q and an upper triangular matrix R. The successive columns of the orthogonal matrix Q form orthonormal bases for the spaces spanned by successive columns $a_1, a_2, \ldots, a_n$ of the matrix A,

$$\mathrm{span}\{a_1\} \subset \mathrm{span}\{a_1, a_2\} \subset \cdots \subset \mathrm{span}\{a_1, a_2, \ldots, a_n\}. \tag{33}$$

The matrices Q and R may be constructed by modified Gram–Schmidt, Householder, or Givens rotations, all of which are exact in exact arithmetic.

The QR *algorithm* takes a matrix A, computes its factors Q and R, then multiplies them together backwards to obtain a new matrix $A^{(1)}$. Continuing this procedure,

$$Q^{(k)} R^{(k)} = A^{(k-1)}, \quad A^{(k)} = R^{(k)} Q^{(k)}, \quad \text{for } k = 2, 3, \ldots \tag{34}$$

the factors $Q^{(k)}$ and $A^{(k)}$ converge towards a Schur factorisation of the original matrix A,

$$A = Q^{(k)} A^{(k)} {Q^{(k)}}^{\mathsf{T}}. \tag{35}$$

Convergence may be improved using shifts, and typically becomes cubic, tripling the number of significant digits with each iteration. The matrices $A^{(k)}$ converge to a matrix that is close to upper triangular, and would be upper triangular in complex arithmetic. In real arithmetic, complex eigenvalue pairs $\lambda_{\pm} = \sigma \pm i\omega$ are represented by 2×2 blocks $\begin{pmatrix} \sigma & \omega \\ -\omega & \sigma \end{pmatrix}$ on the diagonal. Once convergence is complete, typically after $O(n)$ iterations, the eigenvalues may be read off from the diagonal of $A^{(k)}$. The Schur factorisation (35) is preferred for computational work because the orthogonal matrix $Q^{(k)}$ cannot become ill-conditioned, unlike the matrix that would bring A into Jordan normal form.

An efficient implementation for asymmetric matrices first reduces the matrix A to Hessenberg form, which may be done exactly using $O(n^3)$ operations in exact arithmetic. The Hessenberg form then allows an efficient QR iteration using only $O(n^2)$ operations per iteration. Section 7.6 of [8] covers the asymmetric eigenvalue problems. Other books such as [15] cover the simpler eigenvalue problem for symmetric matrices.

5.1 Finding a few eigenvalues of a sparse matrix

For a Bianchi matrix of realistic size, a standard dense eigenvalue routine such as DGEEV from LAPACK [2] would require prohibitive amounts of storage and computation. Moreover, the sparse structure of the Bianchi matrices, that was preserved in the sparse factors L and U, is not preserved in the factors Q and R, or even by a reduction to Hessenberg form. These matrices are all full, or close to full.

However, we do not need to find all the eigenvalues of the Bianchi matrix, as found by routines like DGEEV, but only the two eigenvalues with largest real parts in order to determine λ_2 and the mixing time t_{mix}. The Arnoldi algorithm, which is closely related to GMRES, constructs successive Hessenberg matrices of some small, specified size such that the eigenvalues of the Hessenberg matrices approximate the largest few eigenvalues of the original matrix. Like GMRES, the original matrix is only employed through computing matrix-vector products, which are cheap to compute when the original matrix is sparse. Although the Arnoldi algorithm thus bears some resemblance to the straightforward power method, computing successive powers $A^m\mathbf{b}$ for $m = 1, 2, \ldots$ and some fixed vector $\mathbf{b}$ in the hope of approximating the eigenvector corresponding to the eigenvalue of largest modulus, the Arnoldi algorithm makes much greater use of the information in the sequence of vectors $\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, \ldots A^m\mathbf{b}$, which is nothing other than a basis for the Krylov space $\mathcal{K}_m$ defined in equation (31). The MATLAB sparse eigenvalue routine `eigs` provides a convenient interface to an implementation of the Arnoldi algorithm called ARPACK [11].

The Arnoldi algorithm is likely to perform poorly for an unmodified Bianchi matrix, just as GMRES performs poorly without a preconditioner, because λ_2 is merely one of many eigenvalues of approximately equal modulus arranged in a circle. Although the art of preconditioning eigenvalue problems is still in its infancy, the simple technique of "shift and invert" suffices to find λ_2 relatively easily. Given an estimate σ for λ_2, the matrix $(P-\sigma I)^{-1}$ has an eigenvalue $(\lambda_2-\sigma)^{-1}$ that will be much larger in modulus than any of the other eigenvalues $(\lambda_i-\sigma)^{-1}$ if σ is closer to λ_2 than to any of the other eigenvalues λ_i of the original matrix P.

A good estimate for λ_2 is available from an asymptotic formula for Bianchi matrices when 2^mW is sufficiently large (see Section 7.2 and Appendix A) but in numerical experiments the much cruder estimate $\sigma = 1 - 10^{-4}$ sufficed when using `eigs` to compute the two largest eigenvalues of $(P-\sigma I)^{-1}$. Some examples of mixing times are shown in Figure 12 and compared with the asymptotic formula. By default, the MATLAB implementation `eigs` in "shift and invert" mode computes an LU factorisation of $P - \sigma I$ internally. As one would expect from the relative performance of the algorithms for solving linear systems (see Section 8), replacing the general purpose LU factorisation with a call to the specialised block factorisation algorithm described in Section 4.2 offers a substantial gain in performance.

5.2 Second left eigenvector $\mathbf{z}_2^\mathsf{T}$ and right eigenvector $\mathbf{v}_2$

For some purposes we may also need to know the eigenvectors corresponding to λ_2, *i.e.* the states of the system that are involved in the slowest convergence to equilibrium in (12). These are specified by

$$\mathbf{z}_2^\mathsf{T} P = \lambda_2 \mathbf{z}_2^\mathsf{T}, \quad P\mathbf{v}_2 = \lambda_2 \mathbf{v}_2, \tag{36}$$

and one can solve for $\mathbf{z}_2^\mathsf{T}$, $\mathbf{v}_2$, and λ_2 using a sparse eigenvalue solver, *e.g.* `eigs` in MATLAB or ARPACK. Computing the eigenvectors is a straightforward and inexpensive addition to computing just the eigenvalue. Many routines, such as `eigs` in MATLAB, compute only right eigenvectors, so one must compute left eigenvectors by calling `eigs` again with the transpose of the original matrix.

They can be normalised as in (10) so that

$$\mathbf{z}_2^{\mathsf{T}}\mathbf{v}_2 = 1, \tag{37}$$

but note that this still does not define either $\mathbf{z}_2^{\mathsf{T}}$ or $\mathbf{v}_2$ uniquely.

Given both the left and right eigenvectors, the condition number of the second eigenvalue is given by the reciprocal of their normalised inner product,

$$\operatorname{cond}(\lambda_2) = \frac{||\mathbf{z}_2||_2||\mathbf{v}_2||_2}{\mathbf{z}_2^{\mathsf{T}}\mathbf{v}_2}. \tag{38}$$

Figure 10 shows the condition numbers of the first and second eigenvalues for two Bianchi matrices with $W = 128$ and $m = 3$ and $m = 5$. Varying W had little effect upon these condition numbers.

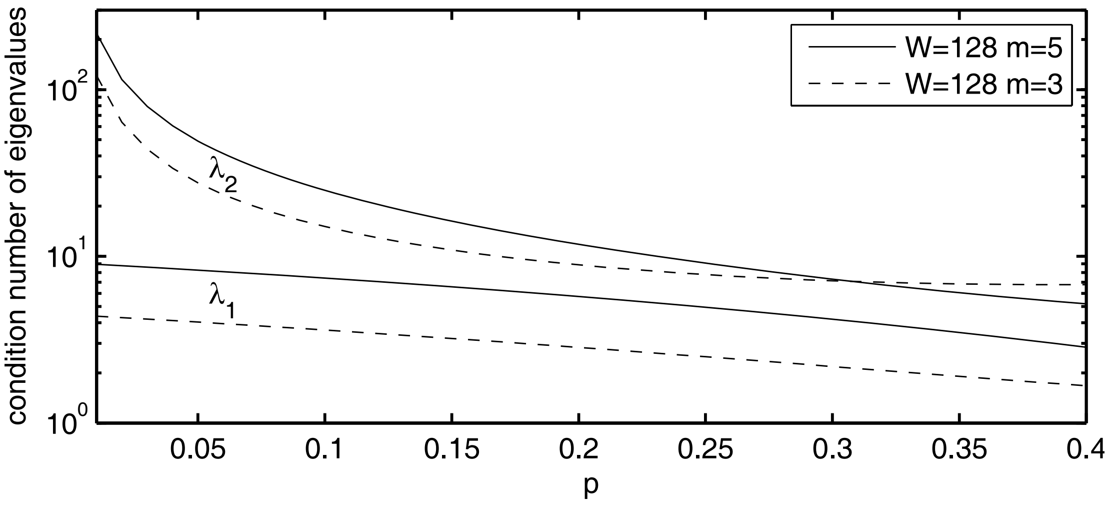


Figure 10: Condition numbers of the first two eigenvalues as p varies for two Bianchi matrices with $W = 128$, $m = 5$ and $W = 128$, $m = 3$.

6 Derivatives with respect to parameters

The material in this section is all based on Section 1.6 (page 15 onwards) of Hinch [9].

6.1 Derivative of the equilibrium population

As mentioned in Section 2 the derivative of the steady state is given by (18) which we write here as

$$\left(\mathbf{z}_1^{\mathsf{T}}\right)'(I - P) = \mathbf{z}_1^{\mathsf{T}} P', \tag{39a}$$

$$\left(\mathbf{z}_1^{\mathsf{T}}\right)'\mathbf{1} = 0. \tag{39b}$$

Equation (39a) is rank deficient by one, so replace one column of the matrix $I - P$ by a column of ones $(1, \ldots, 1)$ to impose the normalisation condition. Notice that this is the same linear system (22) that we solved to find $\mathbf{z}_1^\mathsf{T}$ in the first place, only the right hand sides are different. This is a generic feature of perturbation theory applied to linear equations.

6.2 Derivative of the second eigenvalue and vectors

The derivative of the second eigenvalue is found by (19) earlier, subject to the caveats in that section. To calculate the derivatives of the second eigenvectors if necessary, we differentiate equations (36) with respect to the parameter and obtain

$$\left(\mathbf{z}_2^\mathsf{T}\right)' (\lambda_2 I - P) = -\mathbf{z}_2^\mathsf{T} \left(\lambda_2' I - P'\right), \quad (\lambda_2 I - P)\, \mathbf{v}_2' = -\left(\lambda_2' I - P'\right) \mathbf{v}_2. \tag{40}$$

Differentiating the normalisation $\mathbf{z}_2^\mathsf{T}\mathbf{v}_2 = 1$ gives $\left(\mathbf{z}_2^\mathsf{T}\right)' \mathbf{v}_2 + \mathbf{z}_2^\mathsf{T}\mathbf{v}_2' = 0$, which we choose to split into the two independent conditions

$$\left(\mathbf{z}_2^\mathsf{T}\right)' \mathbf{v}_2 = 0, \quad \text{and} \quad \mathbf{z}_2^\mathsf{T}\mathbf{v}_2' = 0. \tag{41}$$

(We can impose this because it is still arbitrary how a scalar factor is partitioned between $\mathbf{z}_2^\mathsf{T}$ and $\mathbf{v}_2$.) The two linear systems now separate into one for $\left(\mathbf{z}_2^\mathsf{T}\right)'$,

$$\begin{aligned} \left(\mathbf{z}_2^\mathsf{T}\right)' (\lambda_2 I - P) &= -\mathbf{z}_2^\mathsf{T} \left(\lambda_2' I - P'\right), && \text{(42a)} \\ \left(\mathbf{z}_2^\mathsf{T}\right)' \mathbf{v}_2 &= 0 && \text{(42b)} \end{aligned}$$

and another one for $\mathbf{v}_2'$,

$$\begin{aligned} (\lambda_2 I - P)\, \mathbf{v}_2' &= -\left(\lambda_2' I - P'\right) \mathbf{v}_2\,, && \text{(43a)} \\ \mathbf{z}_2^\mathsf{T}\mathbf{v}_2' &= 0. && \text{(43b)} \end{aligned}$$

As before, these are the same linear systems that one would solve to find the left and right eigenvectors in the first place, only with different right hand sides.

7 Bianchi Markov chain

For the Bianchi chain we here give the explicit formulae for the steady state, an asymptotic result for the subdominant eigenvalue, and explicit formulae for the hitting times. The explicit formulae enable exact derivatives to be calculated, and the asymptotic formula allows good approximate derivatives to be calculated.

7.1 Steady state

The states of the Bianchi Markov chain clearly form a single closed class if $0 < p < 1$, and it is aperiodic (*e.g.* since $p(0,0;0,0) = 1/W > 0$) so the general theory applies. We denote the steady state probability distribution of the Bianchi chain by $z(i,k)$, so the balance equations we need to satisfy are

$$\begin{aligned} z(i,k) &= \sum_{j,l} z(j,l)p(j,l;i,k), && \text{(44a)} \\ \sum_{i,k} z(i,k) &= 1. && \text{(44b)} \end{aligned}$$

We shall first find a solution $Z(i,k)$ of the homogeneous system (44a) and then normalise at the end. For the state at the top of E_i, (44a) says that $Z(i, W_i - 1) = f_i/W_i$ where f_i is the total flow into E_i from the bottoms of the other escalators, which is

$$f_i = \begin{cases} (1-p)(Z_0 + Z_1 + \ldots + Z_m) & \text{if } i = 0, \\ pZ_{i-1} & \text{if } 1 \le i \le m-1, \\ p(Z_{m-1} + Z_m) & \text{if } i = m. \end{cases} \tag{45}$$

Here we are using Z_i to denote $Z(i,0)$. Then (44a) at any state (i,k) below the top of E_i gives $Z(i,k) = Z(i,k+1) + f_i/W_i$. So we obtain

$$Z(i,k) = \frac{W_i - k}{W_i} f_i. \tag{46}$$

For consistency at the states with $k = 0$ we therefore need $Z_i = f_i$ for each i, and solving this with the definition (45) and choosing the normalisation with $Z_0 = 1$ we obtain

$$Z_i = \begin{cases} p^i & \text{if } 0 \le i \le m-1, \\ \dfrac{p^m}{1-p} & \text{if } i = m. \end{cases} \tag{47}$$

The normalised steady state is therefore

$$z(i,k) = \frac{W_i - k}{W_i} \frac{Z_i}{S}, \tag{48}$$

where the normalisation constant is

$$\begin{aligned} S &= \sum_{i=0}^{m} \sum_{k=0}^{W_i - 1} \frac{W_i - k}{W_i} Z_i && \text{(49)} \\ &= \sum_{i=0}^{m} \frac{W_i + 1}{2} Z_i && \text{(50)} \\ &= \underbrace{\left(\frac{1}{2(1-p)} + \frac{W}{2(1-2p)} \right)}_{S_0} - \underbrace{\left(\frac{W p^{m+1} 2^{m-1}}{(1-p)(1-2p)} \right)}_{S_1} && \text{(51)} \\ &= S_0 - S_1. && \text{(52)} \end{aligned}$$

The factor $1 - 2p$ in the denominator comes from summing the geometric progression with ratio $2p$, which occurs because the system combines a probability p of going from one escalator to the next with a growth by a factor of 2 in their sizes. The value $p = \frac{1}{2}$ is a removable singularity of S and in fact when $p = \frac{1}{2}$, $S = 1 + W(1 + m/2)$ so there is no difficulty about calculating the steady state in that case, although in practice $p < \frac{1}{2}$. Another way of thinking of the significance of $p = \frac{1}{2}$ is to consider the system with m infinite: it is still persistent, in that the probability of returning to any state is 1, but it has a change of behaviour at $p = \frac{1}{2}$. For $0 < p < \frac{1}{2}$ it is *positive* persistent, *i.e.* the expected time to return to any state from itself is finite. But for $\frac{1}{2} \leq p < 1$ the infinite Markov chain is *null* persistent, *i.e.* the expected time to return is infinite, essentially because the time spent descending the large escalators, $O(2^i)$, outweighs the infrequency of reaching them, $O(p^i)$.

7.2 Subdominant eigenvalue for the Bianchi chain

One way to think of the eigenvalues of P is to consider what they are for $p = 0$ and how they are perturbed from that when $p > 0$. When $p = 0$ each escalator other than E_0 is transient, and in fact each E_i for $1 \leq i \leq m$ corresponds to W_i eigenvalues all zero, in the form of a Jordan block of size W_i. The only persistent behaviour of the system is on E_0 where the eigenvalue equation is

$$\lambda^W = Y_0 =_{\text{def}} \left(\frac{1 + \lambda + \lambda^2 + \ldots + \lambda^{W_0 - 1}}{W_0} \right), \tag{53}$$

with one eigenvalue at 1. When this is perturbed by small $p > 0$, it is to be expected that the perturbed Jordan block of size W_i will produce W_i non-zero eigenvalues of order p^{1/W_i}, and this is the case computationally as we have seen. We can also obtain analytically some results on these lines. The eigenvalue equation from solving $\lambda \mathbf{z}^\mathsf{T} = \mathbf{z}^\mathsf{T} P$ turns out to be

$$\lambda^{W_0} = Y_0(1-p)\left(1 + \frac{Y_1 p}{\lambda^{W_1}}\left(1 + \frac{Y_2 p}{\lambda^{W_2}}\left(1 + \ldots \left(1 + \frac{Y_m p}{\lambda^{W_m} - Y_m p}\right)\right)\right)\right), \tag{54}$$

where each Y_i is like Y_0 but with W_i in place of W. If this is multiplied up by $\lambda_m^W - Y_m p$ then every term has that as a factor except the innermost term from the product on the right. However, that term is of order p^m, which we expect to be small, so it is reasonable to hope that the roots of

$$\lambda^{W_m} = Y_m p = \left(\frac{1 + \lambda + \lambda^2 + \ldots + \lambda^{W_m - 1}}{W_m} \right) p \tag{55}$$

will be a good approximation to the subdominant eigenvalue of P. In fact there is a natural interpretation of this too: the most persistent non-equilibrium behaviour of the system is playing on the largest escalator, because when we reach the bottom of that we have probability p of jumping back onto it for another go. So it is plausible that we can approximate the subdominant eigenvalue by that of a simplified system consisting of an escalator of height W_m with probability p of recycling from the bottom and $1-p$ of going to some absorbing state. The eigenvalue equation for the $\lambda \neq 1$ of that simplified system is just (55). This is considered further in Appendix A, where the method of obtaining derivatives of the approximate eigenvalue with respect to parameters is also given.

In fact the next set of W_{m-1} eigenvalues are all exactly zero, and form an *unperturbed* Jordan block. To see this, first note that the transition probabilities from the bottoms of E_{m-1} and E_m are equal. So P has two identical rows, and so 0 is an eigenvalue, with a left (row) eigenvector having ± 1 at those states $(m-1,0)$ and $(m,0)$. If we consider a row vector with entries ± 1 at $(m-1,k)$ and (m,k) (for $k \le W_{m-1}-1$) then the action of P pushes those entries down the escalators to the bottoms and then annihilates them. So we have a Jordan block of size W_{m-1} and eigenvalue exactly 0 as asserted.

7.3 Hitting times for the Bianchi Markov chain

We shall denote the mean hitting time from (j,l) to (i,k) by $H(j,l;i,k)$, and we shall show that they are given by (56) for $j=i$, by (60) for $j<i$, and by (61) for $j>i$. For $j=i$ we have

$$H(i,l;i,k) = \begin{cases} l-k & \text{if } l>k, \\ l-k+1/z(i,k) & \text{if } l<k, \end{cases} \tag{56}$$

since if $l>k$ the state simply descends E_i deterministically from (i,l) to (i,k) in $l-k$ steps, while if $l<k$ the mean recurrence time $1/z(i,k)$ from state (i,k) can be considered as made up of $k-l$ deterministic steps down to (i,l) followed by a mean of $H(i,l;i,k)$ steps to return to (i,k) again, so $1/z(i,k) = k-l+H(i,l;i,k)$ as required.

To obtain $H(j,l;i,k)$ for $j \ne i$, the first thing to note is that $H(j,l;i,k) = l + H(j,0;i,k)$, since to get from (j,l) to (i,k) for $j \ne i$ we must first take the l deterministic steps down to $(j,0)$. So we let $u_j = H(j,0;i,k)$ and obtain and solve the appropriate recurrence relation, which is

$$u_j = \begin{cases} 1+(1-p)\left(\dfrac{W-1}{2}+u_0\right)+p\left(\dfrac{W_{j+1}-1}{2}+u_{j+1}\right) & \text{if } 0 \le j \le i-2 \\ & \text{or } i+1 \le j \le m-1, \\ 1+(1-p)\left(\dfrac{W-1}{2}+u_0\right)+p\left(\dfrac{W_i-1}{2}-k+\dfrac{k}{W_i z(i,k)}\right) & \text{if } j=i-1, \\ 1+(1-p)\left(\dfrac{W-1}{2}+u_0\right)+p\left(\dfrac{W_m-1}{2}+u_m\right) & \text{if } j=m. \end{cases} \tag{57}$$

The first case (57a) arises because from $(j,0)$ we must take 1 jump, and then with probability $1-p$ that first jump takes us to a random point of E_0, in which case we have a mean of $(W-1)/2$ steps to get down to $(0,0)$ followed by u_0 to hit (i,k) from there; or with probability p that first jump takes us to a random point of E_{j+1}, in which case we have a mean of $(W_{j+1}-1)/2$ steps to get down to $(j+1,0)$ and then u_{j+1} steps from there. The final case (57c) where $j=m$ is the same except that $j+1$ is replaced by m. The special case $j=i-1$ in (57b) is also similar except that when we jump to E_i with probability p we can calculate exactly the expected hitting time on (i,k) from a random point of E_i by the result (56). We therefore have the recurrence relation

$$u_j = c+(1-p)u_0+pW2^j+pu_{j+1}, \quad \text{for } 0 \le j \le i-2 \text{ and } i+1 \le j \le m-1, \tag{58}$$

where $c = 1 + (1-p)(W-1)/2 - p/2$. The general solution of this is

$$u_j = \frac{c}{1-p} + u_0 + \frac{pW2^j}{1-2p} + \frac{A}{p^j}, \tag{59}$$

so this will hold with the constant A taking one value A_0 for $0 \le j \le i-1$ and a different value A_1 for $i+1 \le j \le m$, and we have to find those two constants and u_0. For the range $0 \le j \le i-1$, consistency at $j = 0$ gives $A_0 = -S_0$. Then matching the condition on u_{i-1} given by (57b) determines the value of u_0 and produces

$$H(j,l;i,k) = l - k + \frac{k}{W_i z(i,k)} + \frac{pW(2^j - 2^i)}{1-2p} + \frac{S_0}{p^i} - \frac{S_0}{p^j} \quad \text{for } j < i. \tag{60}$$

When the value of u_0 has been fixed by this, the value of A_1 for the range $i+1 \le j \le m$ is fixed by (57c) and turns out to be $A_1 = -S_1$, so

$$H(j,l;i,k) = l - k + \frac{k}{W_i z(i,k)} + \frac{pW(2^j - 2^i)}{1-2p} + \frac{S_0}{p^i} - \frac{S_1}{p^j} \quad \text{for } j > i, \tag{61}$$

where the only change from (60) is replacing S_0 by S_1 in the last term. The apparent singularities at $p = \frac{1}{2}$ are in fact removable like those of $z(i,k)$ earlier.

8 Conclusions and further work

We have found and demonstrated the appropriate numerical techniques for analysis of Markov chains with the escalator structure typified by the Bianchi chain. This analysis covers the computation of the steady state, mixing time and hitting times, and their derivatives with respect to the system parameters. We have illustrated these techniques for the Bianchi chain. We have also carried out an exact calculation of the steady state and hitting times of the Bianchi chain, and an asymptotic calculation of the mixing time that is accurate for the regions of interest. These exact and asymptotic results also allow computation of derivatives.

We now give some details of the timing to show the efficiency of the methods described here over "unthinking" use of standard software. The Bianchi matrix with $m = 5$ and $W = 128$ has 8064×8064 elements, and would occupy 0.5 GBytes when stored as a dense matrix using 64 bit floating point numbers. We found that the CPU time (in seconds) required to find the equilibrium population $\mathbf{z}_1^\mathsf{T}$ using various different techniques implemented in MATLAB (version 7.1) may be broken down as:

Creating the sparse matrix	1.40s
Sparse LU factorisation (interface to UMFPACK)	0.16s
Custom block factorisation	0.04s
Preconditioned GMRES	0.10s
Dense LU factorisation (interface to LAPACK)	41.00s

These timings were made on a Intel Pentium D830 based workstation with two CPU cores, running MATLAB version 7.1 for 64 bit Linux. The dense direct solution algorithm made effective use of the two CPU cores through MATLAB's internal use of the Intel Mathematics Kernel Library (MKL) for dense linear algebra. The other tasks used only one core.

The preconditioned GMRES brought the residual down to $\sim 10^{-15}$ after eight iterations when $p = 0.1$. The convergence rate seems to be roughly uniform over the whole range $0 \le p \le 1$, with some improvement for very small values of p. By contrast, GMRES with no preconditioning brought the residual down to 4×10^{-9} only after 150 seconds and 10000 iterations (including 100 restarts). Preconditioned GMRES is thus 400 times faster than the dense method, while the custom block factorisation is 1000 times faster. Even a general-purpose sparse LU factorisation is 250 times faster than a dense LU factorisation. In all these numerical experiments, especially for the very large Bianchi matrices listed in Table 1, the time taken to create the sparse matrix (and also its derivative) far outweighed any subsequent computations.

The computational times taken to compute the second eigenvalue, and one of its eigenvectors, were:

Creating the sparse matrix	1.4s
`eigs` using built-in sparse LU factorisation	2.8s
`eigs` using custom block factorisation	0.5s
Dense eigenvalue computation (interface to LAPACK)	4 hours

The last figure, 4 hours, is an estimate extrapolated from the time taken to compute all the eigenvalues of Bianchi matrices of 1/2 and 1/4 the size, in other words 2016×2016 and 4032×4032 instead of 8064×8064.

A The Bianchi subdominant eigenvalue

For fixed p and $W_m = 2^m W$ large, the second largest eigenvalue λ_2 is given asymptotically as the root close to 1 of

$$\frac{p}{W_m}\left(1 + \lambda + \cdots + \lambda^{W_m - 1}\right) = \lambda^{W_m - 1}. \tag{62}$$

Summing the geometric progression, substituting $\lambda_2 = 1 - \xi/W_m$, and taking the limit $W_m \to \infty$ of both sides, we obtain the transcendental equation

$$\frac{e^{\xi} - 1}{\xi} = \frac{1}{p}. \tag{63}$$

The solution ξ may be expressed as

$$\xi = -\mathcal{W}_{-1}(-pe^{-p}) - p, \tag{64}$$

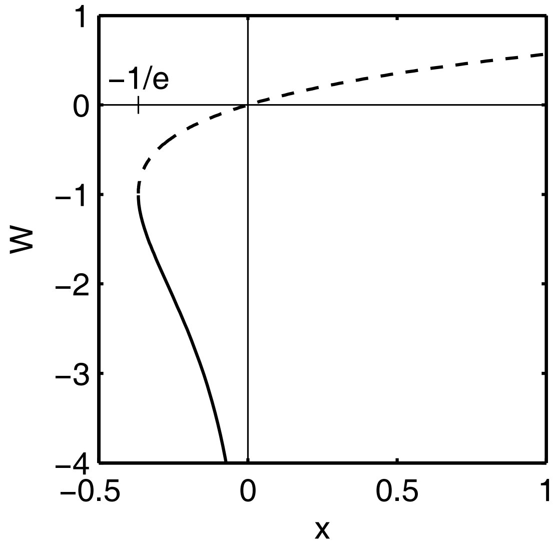


Figure 11: The two real branches of the Lambert W function. The branch $\mathcal{W}_{-1}(x)$ relevant to the solution of equation (66) is shown solid, while the principal branch $\mathcal{W}_0(x)$ is shown dotted. The branches meet at $x = -1/e$ with $\mathcal{W}_{-1}(x) = \mathcal{W}_0(x) = -1$.

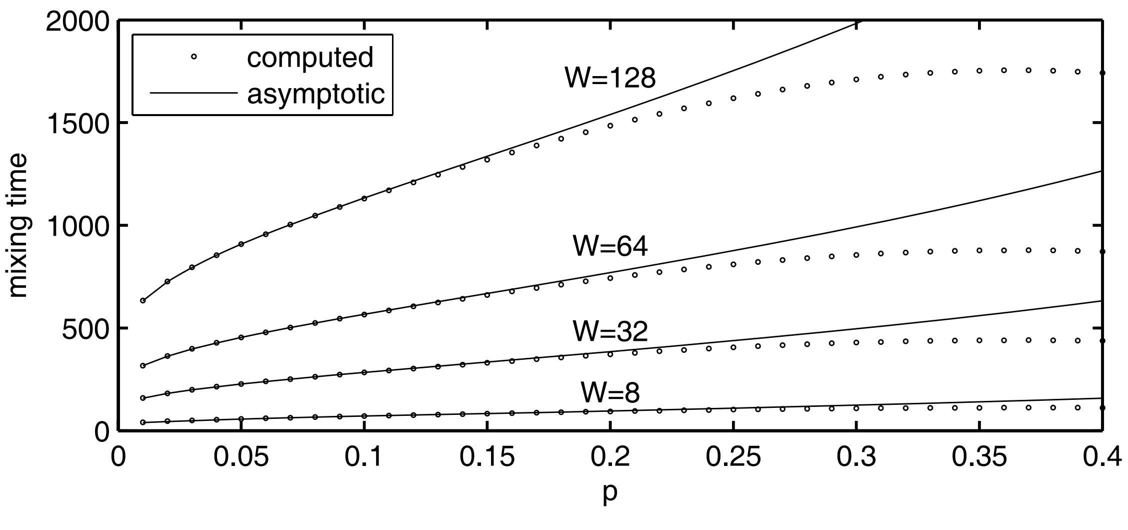


Figure 12: The mixing times $t_{\text{mix}} = (1 - \lambda_2)^{-1}$ for four different Bianchi matrices with $m = 5$ and $W = 8, 32, 64, 128$. The dots result from computing λ_2 using `eigs`, while the continuous lines show $t_{\text{mix}} = W_m/\xi$ with ξ given by the asymptotic formula (64).

where $\mathcal{W}_{-1}(x)$ is the branch of the Lambert W function satisfying $\mathcal{W}_{-1}(x)\exp[\mathcal{W}_{-1}(x)] = x$, and $\mathcal{W}_{-1}(x) < -1$ for x in the open interval $(-e^{-1}, 0)$. The two real branches of the Lambert W function are shown in Figure 11. Corless *et al.* [5] have published a survey of the history, properties, and numerical aspects of the Lambert W function.

Figure 12 compares the mixing time t_{mix} as given by (64),

$$t_{\text{mix}} = \frac{1}{1-\lambda_2} = \frac{W_m}{\xi}, \tag{65}$$

with the mixing times computed numerically using `eigs` for four different sizes of Bianchi matrix. Although equations (62) and (63) were derived for large values of $W_m = 2^m W$, these results suggest that the approximation is excellent for $p \lesssim 0.2$ over a wide range of matrix sizes. The solution of the original equation (62) and the further approximation (63) are indistinguishable, even for the smallest matrix shown with $m = 5$ and $W = 8$. For an even smaller matrix with $m = 3$ and $W = 8$, solving the original equation (62) gives some improved agreement with the exact eigenvalues over solving (63).

Taking logarithms of (63) and rearranging gives

$$\xi = \log(1/p) + \log(\xi + p), \tag{66}$$

which is more amenable to approximate solution. When $p \ll 1$, and thus $\xi \sim \log(1/p) \gg 1$, one may neglect the $O(p/\xi)$ contribution from $\log(1 + p/\xi)$ and approximate (66) by

$$\xi = \log(1/p) + \log(\xi). \tag{67}$$

The $\mathcal{W}_{-1}$ branch of the Lambert W function satisfies (from [10])

$$\mathcal{W}_{-1}(z) + \log \mathcal{W}_{-1}(z) = \log z, \tag{68}$$

so judicious use of $\log(-z) = \log z \pm i\pi$ gives the solution to (67) as

$$\xi = -\mathcal{W}_{-1}(-p). \tag{69}$$

An asymptotic expansion of the Lambert W function is given by equation (4.19) of [5]. With the correct choice of signs for the $\mathcal{W}_{-1}$ branch, this gives

$$\xi = L_1 + L_2 + \frac{L_2}{L_1} + \frac{L_2(2 - L_2)}{2L_1^2} + \cdots \tag{70}$$

where $L_1 = \log(1/p)$ and $L_2 = \log\log(1/p)$. The same expansion may be obtained using an iterative procedure, taking $\xi_0 = \log(1/p)$ and setting $\xi_{n+1} = \log(1/p) + \log \xi_n$, then expanding logarithms of sums (see [9]).

Given the appearance of $L_2 = \log\log(1/p)$, it is not surprising that this asymptotic expansion converges very slowly unless p is extremely small. However, two iterations of Newton's method to $\xi = \log(\xi/p + 1)$, beginning with $\xi_0 = \log(1/p)$, gives an excellent, though unwieldy, approximation to the solution of the earlier equation (66) for $p \lesssim 0.3$.

For analytical work it would probably be preferable to use known properties of the Lambert W function directly. For example,

$$\frac{d}{dz}\mathcal{W}_{-1}(z) = \frac{\mathcal{W}_{-1}(z)}{z(1+\mathcal{W}_{-1}(z))}, \tag{71}$$

(from [5]) so the derivative of (64) gives the derivative of ξ with respect to p as

$$\frac{d\xi}{dp} = -\frac{p+\mathcal{W}_{-1}(-pe^{-p})}{p(1+\mathcal{W}_{-1}(-pe^{-p}))} = -\frac{\xi}{p(\xi+p-1)}. \tag{72}$$

For the mixing time $t_{\text{mix}} = W_m/\xi$,

$$\frac{d}{dp}\left(\frac{W_m}{\xi}\right) = \frac{W_m}{p(1+\mathcal{W}_{-1}(-pe^{-p}))(p+\mathcal{W}_{-1}(-pe^{-p}))} = \frac{W_m}{p\xi(\xi+p-1)}. \tag{73}$$

B Preconditioning a perturbed Jordan block

Consider the $n \times n$ Jordan block with eigenvalue $-\mu$, the minus sign being for later convenience, and make a perturbation ϵ to the bottom left matrix element. The resulting matrix

$$J_\epsilon = \begin{pmatrix} -\mu & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & -\mu & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & -\mu & 1 & \dots & 0 & 0 \\ \vdots & & & \ddots & \ddots & & \vdots \\ 0 & 0 & \dots & \dots & -\mu & 1 & 0 \\ 0 & 0 & \dots & \dots & \dots & -\mu & 1 \\ \epsilon & 0 & \dots & \dots & \dots & 0 & -\mu \end{pmatrix} \tag{74}$$

has characteristic polynomial $(\lambda+\mu)^n - \epsilon = 0$. The eigenvalues are thus given by

$$\lambda_m = -\mu + \epsilon^{1/n} e^{2\pi i m/n} \text{ for } m = 0, \dots, n-1. \tag{75}$$

The single n-fold degenerate eigenvalue of the unperturbed Jordan block J_0 splits under the perturbation into n separate eigenvalues evenly distributed on a circle of radius $\epsilon^{1/n}$ around the unperturbed eigenvalue $-\mu$ (recall Figure 5). Iterative methods like GMRES based on Krylov spaces may therefore be expected to perform poorly for the matrix J_ϵ, requiring n iterations to converge in exact arithmetic.

The inverse of the unperturbed Jordan block J_0 is the (non-cyclic) Toeplitz matrix

$$J_0^{-1} = -\begin{pmatrix} \mu^{-1} & \mu^{-2} & \mu^{-3} & \mu^{-4} & \dots & \mu^{1-n} & \mu^{-n} \\ 0 & \mu^{-1} & \mu^{-2} & \mu^{-3} & \dots & \mu^{2-n} & \mu^{1-n} \\ 0 & 0 & \mu^{-1} & \mu^{-2} & \dots & \mu^{3-n} & \mu^{2-n} \\ \vdots & & \ddots & \ddots & & \vdots & \\ 0 & 0 & \dots & \dots & \mu^{-1} & \mu^{-2} & \mu^{-3} \\ 0 & 0 & \dots & \dots & \dots & \mu^{-1} & \mu^{-2} \\ 0 & 0 & \dots & \dots & \dots & 0 & \mu^{-1} \end{pmatrix}. \tag{76}$$

Using the matrix J_0^{-1} as a preconditioner on the left for the perturbed matrix J_ϵ, we obtain a rank-1 update of the identity matrix,

$$J_0^{-1}J_\epsilon = \begin{pmatrix} 1-\epsilon\mu^{-n} & 0 & 0 & 0 & \dots & 0 & 0 \\ -\epsilon\mu^{1-n} & 1 & 0 & 0 & \dots & 0 & 0 \\ -\epsilon\mu^{2-n} & 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & & & \ddots & \ddots & & \vdots \\ -\epsilon\mu^{-3} & 0 & \dots & \dots & 1 & 0 & 0 \\ -\epsilon\mu^{-2} & 0 & \dots & \dots & 0 & 1 & 0 \\ -\epsilon\mu^{-1} & 0 & \dots & \dots & 0 & 0 & 1 \end{pmatrix}, \tag{77}$$

with $O(\epsilon)$ modifications to the first column. The matrix $J_0^{-1}J_\epsilon$ has characteristic polynomial

$$(\lambda-1)^{n-1}\left(\lambda-1+\epsilon\mu^{-n}\right)=0, \tag{78}$$

with an $(n-1)$-fold degenerate eigenvalue $\lambda = 1$, and a single distinct eigenvalue at $\lambda = 1-\epsilon\mu^{-n}$ due to the perturbation. One therefore expects GMRES to converge in two iterations when applied to the preconditioned matrix $J_0^{-1}J_\epsilon$. In a concrete implementation one would not compute the inverse matrix J_0^{-1}, but instead compute solutions to the upper bidiagonal linear system $J_0\mathbf{x} = \mathbf{b}$ by back substitution.

References

[1] Aldous, D., & Fill, J. A. A second look at general Markov chains. `http://www.stat.berkeley.edu/~aldous/RWG/Chap9.pdf`.

[2] Anderson, E. *et al.* 1999 *LAPACK Users' Guide*, 3rd edn. Philadelphia: SIAM.

[3] Bianchi, G. 2000 Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE J. Selected Areas Commun.* **18**, 535–547.

[4] Campbell, S. L. & Meyer, C. D. Jr. 1991 *Generalized inverses of linear transformations.* Dover reprint.

[5] Corless, R. M., Gonnet, G., Hare, D. E. G., Jeffrey, D. J. & Knuth, D. E. 1996 On the Lambert W function. *Adv. Comput. Math.* **5**, 329–359.

[6] Davis, T. A. 2004 Algorithm 832: UMFPACK – an unsymmetric-pattern multifrontal method with a column pre-ordering strategy. *ACM Trans. Math. Software* **30**, 196–199.

[7] Gilbert, R. J., Moler, C. & Schreiber, R. 1992 Sparse matrices in MATLAB: design and implementation. *SIAM J. Matrix Anal. Appl.* **13**, 333–356.

[8] Golub, G. H. & Van Loan, C. F. 1996 *Matrix Computations*, 3rd edn. Baltimore: Johns Hopkins University Press.

[9] Hinch, E. J. 1991 *Perturbation Methods*. Cambridge, UK: Cambridge University Press.

[10] Jeffrey, D. J., Hare, D. E. G. & Corless, R. M. 1996 Unwinding the branches of the Lambert W function. *Math. Scientist* **21**, 1–7.

[11] Lehoucq, R. B., Sorensen, D. C. & Yang, C. 1998 *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Philadelphia: SIAM, available from `http://www.caam.rice.edu/software/ARPACK`.

[12] Meyer, C. D. 1994 Sensitivity of the stationary distribution of a Markov chain. *SIAM J. Matrix Anal. Appl.* **15**, 715–728.

[13] Saad, Y. 1996 *Iterative Methods for Sparse Linear Systems*. Boston: PWS, available from `http://www-users.cs.umn.edu/~saad/books.html`. Second edition (2003) published by SIAM, Philadelphia.

[14] Saad, Y. & Schultz, M. H. 1985 GMRES – a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.* **7**, 856–869.

[15] Trefethen, L. N. & Bau, D. 1997 *Numerical Linear Algebra*. Philadelphia: SIAM.