Chapter 10

Numerical Solution of Markov Chains

10.1 Introduction

We now turn our attention to the computation of stationary and transient distributions of Markov chains. For the computation of stationary distributions, we consider first the case when the number of states can be large but finite. In particular, we examine direct methods based on Gaussian elimination, point and block iterative methods such as point and block Gauss–Seidel and decompositional methods that are especially well suited to Markov chains that are almost reducible. Second, when the Markov chain has an infinite state space and the pattern of nonzero elements in the transition matrix is block repetitive, we develop and analyze the matrix geometric and matrix analytic methods. As for the computation of transient distributions, we describe methods that are appropriate for small-scale Markov chains, as well as the popular uniformization method and methods based on differential equation solvers for large-scale chains. Throughout this chapter, our concern is with stationary, homogeneous Markov chains only.

10.1.1 Setting the Stage

We begin by considering finite, irreducible Markov chains. Such Markov chains have a unique stationary probability distribution π whose elements are strictly greater than zero. When the Markov chain is also aperiodic, this unique stationary distribution is also the steady-state distribution. Let P be the transition probability matrix of a finite, irreducible discrete-time Markov chain. Then π is the left-hand eigenvector corresponding to the dominant (and simple) unit eigenvalue of P, normalized so that its elements sum to 1:

$$\pi P = \pi \quad \text{with} \quad \pi e = 1. \tag{10.1}$$

If the Markov chain evolves in continuous time rather than in discrete time and its infinitesimal generator is denoted by Q, then its stationary distribution may be found from the system of linear equations

$$\pi Q = 0 \quad \text{with} \quad \pi e = 1. \tag{10.2}$$

Observe that both these equations may be put into the same form. We may write the first, $\pi P = \pi$, as $\pi(P - I) = 0$ thereby putting it into the form of Equation (10.2). Observe that $(P - I)$ has all the properties of an infinitesimal generator, namely, all off-diagonal elements are nonnegative; row sums are equal to zero and diagonal elements are equal to the negated sum of off-diagonal row elements. On the other hand, we may discretize a continuous-time Markov chain. From

$$\pi Q = 0 \quad \text{with} \quad \pi e = 1$$

we may write

$$\pi(Q\Delta t + I) = \pi \quad \text{with} \quad \pi e = 1, \tag{10.3}$$

thereby posing it in the form of Equation (10.1). In the discretized Markov chain, transitions take place at intervals Δt, Δt being chosen sufficiently small that the probability of two transitions taking place in time Δt is negligible, i.e., of order $o(\Delta t)$. One possibility is to take

$$\Delta t \leq \frac{1}{\max_i |q_{ii}|}.$$

In this case, the matrix $(Q\Delta t + I)$ is stochastic and the stationary[1] probability vector π of the continuous-time Markov chain, obtained from $\pi Q = 0$, is identical to that of the discretized chain, obtained from $\pi(Q\Delta t + I) = \pi$. It now follows that numerical methods designed to compute the stationary distribution of discrete-time Markov chains may be used to compute the stationary distributions of continuous-time Markov chains, and vice versa.

Example 10.1 In the previous chapter we saw that one very simple method for computing the stationary distribution of a discrete-time Markov chain, is to successively compute the probability distribution at each time step until no further change in the distribution is observed. We shall now apply this method (which we shall later refer to as the *power method*) to the *continous-time* Markov chain whose infinitesimal generator is given by

$$Q = \begin{pmatrix} -4 & 4 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ 0 & 2 & -4 & 2 \\ 0 & 0 & 1 & -1 \end{pmatrix}.$$

Setting $\Delta t = 1/6$, we obtain

$$Q\Delta t + I = \begin{pmatrix} 1-4/6 & 4/6 & 0 & 0 \\ 3/6 & 1-6/6 & 3/6 & 0 \\ 0 & 2/6 & 1-4/6 & 2/6 \\ 0 & 0 & 1/6 & 1-1/6 \end{pmatrix} = \begin{pmatrix} 1/3 & 2/3 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 1/6 & 5/6 \end{pmatrix}.$$

Beginning with $\pi^{(0)} = (1,\ 0,\ 0,\ 0)$, and successively computing $\pi^{(n)} = \pi^{(n-1)}(Q\Delta t + I)$ for $n = 1, 2, \ldots$, we obtain

$$\begin{aligned}
\pi^{(0)} &= (1.0000 \quad 0.0000 \quad 0.0000 \quad 0.0000),\\
\pi^{(1)} &= (0.3333 \quad 0.6667 \quad 0.0000 \quad 0.0000),\\
\pi^{(2)} &= (0.4444 \quad 0.2222 \quad 0.3333 \quad 0.0000),\\
\pi^{(3)} &= (0.2593 \quad 0.4074 \quad 0.2222 \quad 0.1111),\\
&\ \vdots \\
\pi^{(10)} &= (0.1579 \quad 0.1929 \quad 0.2493 \quad 0.3999),\\
&\ \vdots \\
\pi^{(25)} &= (0.1213 \quad 0.1612 \quad 0.2403 \quad 0.4773),\\
&\ \vdots \\
\pi^{(50)} &= (0.1200 \quad 0.1600 \quad 0.2400 \quad 0.4800),
\end{aligned}$$

and, correct to four decimal places, no further change is observed in computing the distribution at higher step values. Thus we may take (0.1200, 0.1600, 0.2400, 0.4800) to be the stationary distribution of the continuous-time Markov chain represented by the infinitesimal generator Q.

Thus, depending on how we formulate the problem, we may obtain the stationary distribution of a Markov chain (discrete *or* continuous time) by solving either $\pi P = \pi$ or $\pi Q = 0$.

[1] We caution the reader whose interest is in transient solutions (i.e., probability distributions at an arbitrary time t) that those of the discretized chain, represented by the transition probability matrix $Q\Delta t + I$, are not the same as those of the continuous-time chain, represented by the infinitesimal generator Q. However, both have the same stationary distribution.

From the perspective of a numerical analyst, these are two different problems. The first is an *eigenvalue/eigenvector* problem in which the stationary solution vector π is the left-hand eigenvector corresponding to a unit eigenvalue of the transition probability matrix P. The second is a *linear equation* problem in which the desired vector π is obtained by solving a homogeneous (right-hand side identically equal to zero) system of linear equations with singular coefficient matrix Q. In light of this, it behooves us to review some eigenvalue/eigenvector properties of stochastic matrices and infinitesimal generators.

10.1.2 Stochastic Matrices

A matrix $P \in \Re^{n\times n}$ is said to be a *stochastic* matrix if it satisfies the following three conditions:

1. $p_{ij} \geq 0$ for all i and j.
2. $\sum_{\text{all } j} p_{ij} = 1$ for all i.
3. At least one element in each column differs from zero.

Matrices that obey condition 1 are called nonnegative matrices, and stochastic matrices form a proper subset of them. Condition 2 implies that a transition is guaranteed to occur from state i to at least one state in the next time period (which may be state i again). Condition 3 specifies that, since each column has at least one nonzero element, there are no *ephemeral* states, i.e., states that could not possibly exist after the first time transition. In much of the literature on stochastic matrices this third condition is omitted (being considered trivial). In the remainder of this text we also shall omit this condition. We shall now list some important properties concerning the eigenvalues and eigenvectors of stochastic matrices.

Property 10.1.1 *Every stochastic matrix has an eigenvalue equal to 1.*

Proof: Since the sum of the elements of each row of P is 1, we must have

$$Pe = e,$$

where $e = (1, 1, \ldots, 1)^T$. It immediately follows that P has a unit eigenvalue.

Corollary 10.1.1 *Every infinitesimal generator Q has a zero eigenvalue.*

Proof: This follows immediately from the fact that $Qe = 0$.

Property 10.1.2 *The eigenvalues of a stochastic matrix must have modulus less than or equal to 1.*

Proof: To prove this result we shall use the fact that for any matrix A,

$$\rho(A) \leq \|A\|_\infty = \max_j \left(\sum_{\text{all } k} |a_{jk}| \right),$$

where $\rho(A)$ denotes the *spectral radius* (magnitude of the largest eigenvalue) of A. The *spectrum* of A is the set of eigenvalues of A. For a stochastic matrix P, $\|P\|_\infty = 1$, and therefore we may conclude that

$$\rho(P) \leq 1.$$

Hence, no eigenvalue of a stochastic matrix P can exceed 1 in modulus.

Notice that this property, together with Property 10.1.1, implies that the spectral radius of a stochastic matrix is 1, i.e.,

$$\rho(P) = 1.$$

Property 10.1.3 *The right-hand eigenvector corresponding to a unit eigenvalue $\lambda_1 = 1$ of a stochastic matrix P is given by e where $e = (1, 1, \ldots)^T$.*

Proof: Since the sum of each row of P is 1, we have $Pe = e = \lambda_1 e$.

Property 10.1.4 *The vector π is a stationary probability vector of a stochastic matrix P iff it is a left-hand eigenvector corresponding to a unit eigenvalue.*

Proof: By definition, a stationary probability vector π, does not change when post-multiplied by a stochastic transition probability matrix P, which implies that

$$\pi = \pi P.$$

Therefore π satisfies the eigenvalue equation

$$\pi P = \lambda_1 \pi \quad \text{for } \lambda_1 = 1.$$

The converse is equally obvious.

The following additional properties apply when P is the stochastic matrix of an *irreducible* Markov chain.

Property 10.1.5 *The stochastic matrix of an irreducible Markov chain possesses a simple unit eigenvalue.*

The theorem of Perron and Frobenius is a powerful theorem which has applicability to irreducible Markov chains [50]. Property 10.1.5 follows directly from the Perron–Frobenius theorem.

Property 10.1.6 *For any irreducible Markov chain with stochastic transition probability matrix P, let*

$$P(\alpha) = I - \alpha(I - P), \tag{10.4}$$

where $\alpha \in \Re' \equiv (-\infty, \infty)\backslash\{0\}$ (i.e., the real line with zero deleted). Then 1 is a simple eigenvalue of every $P(\alpha)$, and associated with this unit eigenvalue is a uniquely defined positive left-hand eigenvector of unit 1-norm, which is precisely the stationary probability vector π of P.

Proof: Notice that the spectrum of $P(\alpha)$ is given by $\lambda(\alpha) = 1 - \alpha(1 - \lambda)$ for λ in the spectrum of P. Furthermore, as can be verified by substituting from Equation (10.4), the left-hand eigenvectors of P and $P(\alpha)$ agree in the sense that

$$x^T P = \lambda x^T \quad \text{if and only if} \quad x^T P(\alpha) = \lambda(\alpha) x^T \quad \text{for all } \alpha.$$

In particular, this means that regardless of whether or not $P(\alpha)$ is a stochastic matrix, $\lambda(\alpha) = 1$ is a simple eigenvalue of $P(\alpha)$ for all α, because $\lambda = 1$ is a simple eigenvalue of P. Consequently, the entire family $P(\alpha)$ has a unique positive left-hand eigenvector of unit 1-norm associated with $\lambda(\alpha) = 1$, and this eigenvector is precisely the stationary distribution π of P.

Example 10.2 Consider the 3×3 stochastic matrix given by

$$P = \begin{pmatrix} .99911 & .00079 & .00010 \\ .00061 & .99929 & .00010 \\ .00006 & .00004 & .99990 \end{pmatrix}. \tag{10.5}$$

Its eigenvalues are 1.0, .9998, and .9985. The maximum value of α that we can choose so that $P(\alpha)$ is stochastic is $\alpha_1 = 1/.00089$. The resulting stochastic matrix that we obtain is given by

$$P(\alpha_1) = \begin{pmatrix} 0 & .88764 & .11236 \\ .68539 & .20225 & .11236 \\ .06742 & .04494 & .88764 \end{pmatrix},$$

and its eigenvalues are 1.0, .77528, and −.68539. Yet another choice, $\alpha_2 = 10{,}000$, yields a $P(\alpha_2)$ that is not stochastic. We find

$$P(\alpha_2) = \begin{pmatrix} -7.9 & 7.9 & 1.0 \\ 6.1 & -6.1 & 1.0 \\ 0.6 & 0.4 & 0.0 \end{pmatrix}.$$

Its eigenvalues are 1.0, −1.0, and −14.0. The left-hand eigenvector corresponding to the unit eigenvalue for all three matrices is

$$\pi = (.22333,\ .27667,\ .50000)$$

—the stationary probability vector of the original stochastic matrix.

This theorem has important consequences for the convergence of certain iterative methods used to find stationary distributions of discrete- and continuous-time Markov chains. As we shall see later, the rate of convergence of these methods depends upon the separation of the largest (in modulus) and second largest eigenvalues. This theorem may be used to induce a larger separation, as can be seen in Example 10.5 in moving from P to $P(\alpha_1)$, with a resulting faster convergence rate.

10.1.3 The Effect of Discretization

Let us now find the values of $\Delta t > 0$ for which the matrix $P = Q\Delta t + I$ is stochastic.

Example 10.3 Consider a two-state Markov chain whose infinitesimal generator is

$$Q = \begin{pmatrix} -q_1 & q_1 \\ q_2 & -q_2 \end{pmatrix},$$

with $q_1, q_2 \geq 0$. The transition probability matrix is then

$$P = Q\Delta t + I = \begin{pmatrix} 1 - q_1\Delta t & q_1\Delta t \\ q_2\Delta t & 1 - q_2\Delta t \end{pmatrix}.$$

Obviously the row sums are equal to 1. To ensure that $0 \leq q_1\Delta t \leq 1$ and $0 \leq q_2\Delta t \leq 1$, we require that $0 \leq \Delta t \leq q_1^{-1}$ and $0 \leq \Delta t \leq q_2^{-1}$. Let us assume, without loss of generality, that $q_1 \geq q_2$. Then $0 \leq \Delta t \leq q_1^{-1}$ satisfies both conditions. To ensure that $0 \leq 1 - q_1\Delta t \leq 1$ and $0 \leq 1 - q_2\Delta t \leq 1$ again requires that $\Delta t \leq q_1^{-1}$. Consequently the maximum value that we can assign to Δt subject to the condition that P be stochastic is $\Delta t = 1/\max_i |q_i|$.

Similar results hold for a general stochastic matrix $P = Q\Delta t + I$ to be stochastic, given that Q is an infinitesimal generator. As before, for any value of Δt, the row sums of P are unity, since by definition the row sums of Q are zero. Therefore we must concern ourselves with the values of Δt that guarantee the elements of P lie in the interval [0,1]. Let q be the size of the largest off-diagonal element:

$$q = \max_{i,j\ i \neq j} (q_{ij}) \quad \text{and} \quad q_{ij} \geq 0 \quad \text{for all } i, j.$$

Then $0 \leq p_{ij} \leq 1$ holds if $0 \leq q_{ij}\Delta t \leq 1$, which is true if $\Delta t \leq q^{-1}$. Now consider a diagonal element $p_{ii} = q_{ii}\Delta t + 1$. We have

$$0 \leq q_{ii}\Delta t + 1 \leq 1$$

or

$$-1 \leq q_{ii}\Delta t \leq 0.$$

The right-hand inequality holds for all $\Delta t \geq 0$, since q_{ii} is negative. The left-hand inequality $q_{ii}\Delta t \geq -1$ is true if $\Delta t \leq -q_{ii}^{-1}$, i.e., if $\Delta t \leq |q_{ii}|^{-1}$. It follows then that, if $0 \leq \Delta t \leq (\max_i |q_{ii}|)^{-1}$, the matrix P is stochastic. (Since the diagonal elements of Q equal the negated sum of the off-diagonal elements in a row, we have $\max_i |q_{ii}| \geq \max_{i\neq j}(q_{ij})$.) Thus, Δt must be less than or equal to the reciprocal of the absolute value of the largest diagonal element of Q.

The choice of a suitable value for Δt plays a crucial role in some iterative methods for determining the stationary probability vector from Equation (10.3). As we mentioned above, the rate of convergence is intimately related to the magnitude of the eigenvalues of P. As a general rule, the closer the magnitudes of the subdominant eigenvalues are to 1, the slower the convergence rate. We would therefore like to maximize the distance between the largest eigenvalue, $\lambda_1 = 1$, and the subdominant eigenvalue (the eigenvalue that in modulus is closest to 1). Notice that, as $\Delta t \to 0$, the eigenvalues of P *all* tend to unity. This would suggest that we choose Δt to be as large as possible, subject only to the constraint that P be a stochastic matrix. However, choosing $\Delta t = (\max_i |q_{ii}|)^{-1}$ does not necessarily guarantee the maximum separation of dominant and subdominant eigenvalues. It is simply a good heuristic.

Example 10.4 Consider the (2×2) case as an example. The eigenvalues of P are the roots of the characteristic equation $|P - \lambda I| = 0$, i.e.,

$$\begin{vmatrix} 1 - q_1\Delta t - \lambda & q_1\Delta t \\ q_2\Delta t & 1 - q_2\Delta t - \lambda \end{vmatrix} = 0.$$

These roots are $\lambda_1 = 1$ and $\lambda_2 = 1 - \Delta t(q_1 + q_2)$. As $\Delta t \to 0$, $\lambda_2 \to \lambda_1 = 1$. Also notice that the left-hand eigenvector corresponding to the unit eigenvalue λ_1 is independent of the choice of Δt. We have

$$(q_2/(q_1+q_2) \;\; q_1/(q_1+q_2)) \begin{pmatrix} 1 - q_1\Delta t & q_1\Delta t \\ q_2\Delta t & 1 - q_2\Delta t \end{pmatrix} = (q_2/(q_1+q_2) \;\; q_1/(q_1+q_2)).$$

This eigenvector is, of course, the stationary probability vector of the Markov chain and as such must be independent of Δt. The parameter Δt can only affect the speed at which matrix iterative methods converge to this vector. As we mentioned above, it is often advantageous to choose Δt to be as large as possible, subject only to the constraint that the matrix P be a stochastic matrix. Intuitively, we may think that by choosing a large value of Δt we are marching more quickly toward the stationary distribution. With small values we essentially take only small steps, and therefore it takes longer to arrive at our destination, the stationary distribution.

10.2 Direct Methods for Stationary Distributions

10.2.1 Iterative versus Direct Solution Methods

There are two basic types of solution method in the field of numerical analysis: solution methods that are *iterative* and solution methods that are *direct*. Iterative methods begin with an initial approximation to the solution vector and proceed to modify this approximation in such a way that, at each step or iteration, it becomes closer and closer to the true solution. Eventually, it becomes equal to the true solution. At least that is what we hope. If no initial approximation is known, then a guess is made or an arbitrary initial vector is chosen instead. Sometimes iterative methods fail to converge to the solution. On the other hand, a direct method attempts to go straight to the final solution. A certain number of well defined steps must be taken, at the end of which the solution has been computed. However, all is not that rosy, for sometimes the number of steps that must be taken is prohibitively large and the buildup of rounding error can, in certain cases, be substantial.

Iterative methods of one type or another are by far the most commonly used methods for obtaining the stationary probability vector from either the stochastic transition probability matrix or from the infinitesimal generator. There are several important reasons for this choice. First, an examination of the standard iterative methods shows that the only operation in which the matrices are involved, is their multiplication with one or more vectors—an operation which leaves the transition matrices unaltered. Thus compact storage schemes, which minimize the amount of memory required to store the matrix and which in addition are well suited to matrix multiplication, may be conveniently implemented. Since the matrices involved are usually large and very sparse, the savings made by such schemes can be considerable. With direct equation solving methods, the elimination of one nonzero element of the matrix during the reduction phase often results in the creation of several nonzero elements in positions which previously contained zero. This is called *fill-in* and not only does it make the organization of a compact storage scheme more difficult, since provision must be made for the deletion and the insertion of elements, but in addition, the amount of fill-in can often be so extensive that available memory can be exhausted. A successful direct method must incorporate a means of overcoming these difficulties.

Iterative methods have other advantages. Use may be made of good initial approximations to the solution vector and this is especially beneficial when a series of related experiments is being conducted. In such circumstances the parameters of one experiment often differ only slightly from those of the previous; many will remain unchanged. Consequently, it is to be expected that the solution to the new experiment will be close to that of the previous and it is advantageous to use the previous result as the new initial approximation. If indeed there is little change, we should expect to compute the new result in relatively few iterations.

Also, an iterative process may be halted once a prespecified tolerance criterion has been satisfied, and this may be relatively lax. For example, it may be wasteful to compute the solution of a mathematical model correct to full machine precision when the model itself contains errors of the order of 5–10%. In contrast, a direct method must continue until the final specified operation has been carried out. And lastly, with iterative methods, the matrix is never altered and hence the buildup of rounding error is, to all intents and purposes, nonexistent.

For these reasons, iterative methods have traditionally been preferred to direct methods. However, iterative methods have a major disadvantage in that often they require a very long time to converge to the desired solution. Direct methods have the advantage that an upper bound on the time required to obtain the solution may be determined before the calculation is initiated. More important, for certain classes of problems, direct methods can result in a much more accurate answer being obtained in *less time*. Since iterative method will in general require less memory than direct methods, these latter can only be recommended if they obtain the solution in less time. Unfortunately, it is often difficult to predict when a direct solver will be more efficient than an iterative solver.

In this section we consider direct methods for computing the stationary distribution of Markov chains while in the next section we consider basic iterative methods. Some methods, such as *preconditioned projection methods* may be thought of as a combination of the direct and iterative approaches, but their study is beyond the scope of this text. Readers wishing further information on projection methods should consult one of the standard texts, such as [47, 50].

10.2.2 Gaussian Elimination and LU Factorizations

Direct equation solving methods for obtaining the stationary distribution of Markov chains are applied to the system of equations

$$\pi Q = 0, \;\; \pi \geq 0, \;\; \pi e = 1, \tag{10.6}$$

i.e., a homogeneous system of n linear equations in which the n unknowns $\pi_i, \; i = 1, 2, \ldots n$ are the components of the stationary distribution vector, π. The vector e is a column vector whose elements are all equal to 1. The system of equations (10.6) has a solution other than the trivial solution ($\pi_i = 0$, for all i) if and only if the determinant of the coefficient matrix is zero, i.e., if and only if the coefficient matrix is singular. Since the determinant of a matrix is equal to the product of its eigenvalues and since Q possesses a zero eigenvalue, the singularity of Q and hence the existence of a non-trivial solution, follows. The standard direct approaches for solving systems of linear equations are based on the method of Gaussian elimination (GE) and related LU factorizations. Gaussian elimination is composed of two phases, a reduction phase during which the coefficient matrix is brought to upper triangular form, and a backsubstitution phase which generates the solution from the reduced coefficient matrix. The first (reduction) phase is the computationally expensive part of the algorithm, having an $O(n^3)$ operation count when the matrix is full. The backsubstitution phase has order $O(n^2)$. A detailed discussion of Gaussian elimination and LU factorizations can be found in Appendix B. Readers unfamiliar with the implementation of these methods should consult this appendix before proceeding.

Gaussian Elimination for Markov Chains

Consider the Gaussian elimination approach for computing the stationary distribution of a finite, irreducible continuous-time Markov chain. In this case the system of equations, $\pi Q = 0$, is homogeneous, the coefficient matrix is singular and there exists a unique solution vector π. We proceed by means of an example.

Example 10.5 Suppose we seek to solve

$$(\pi_1, \pi_2, \pi_3, \pi_4)\begin{pmatrix} -4.0 & 1.0 & 2.0 & 1.0 \\ 4.0 & -9.0 & 2.0 & 3.0 \\ 0.0 & 1.0 & -3.0 & 2.0 \\ 0.0 & 0.0 & 5.0 & -5.0 \end{pmatrix} = (0, 0, 0, 0).$$

Although not strictly necessary, we begin by transposing both sides of this equation, thereby putting it into the standard textbook form, $Ax = b$, for implementing Gaussian elimination. We get

$$Ax = \begin{pmatrix} -4.0 & 4.0 & 0.0 & 0.0 \\ 1.0 & -9.0 & 1.0 & 0.0 \\ 2.0 & 2.0 & -3.0 & 5.0 \\ 1.0 & 3.0 & 2.0 & -5.0 \end{pmatrix}\begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = b.$$

We now proceed to perform the reduction phase of Gaussian elimination. The first step is to use the first equation to eliminate the nonzero elements in column 1 that lie below the pivot element, $a_{11} = -4$. This is accomplished by adding multiples of row 1 into rows 2 through $n = 4$. Once this has been done, the next step is to eliminate nonzero elements in column 2 that lie below the pivot element in row 2 (the newly modified $a'_{22} = -8$) by adding multiples of row 2 into rows 3 through $n = 4$. And so the reduction phase of Gaussian elimination continues until all nonzero elements below the diagonal have been eliminated—at which point the reduction phase of the algorithm has been completed. Following the notation introduced in Appendix B, we obtain the *reduction phase:*

$$\begin{array}{r|} \text{Multipliers} \\ 0.25 \\ 0.50 \\ 0.25 \end{array} \begin{pmatrix} -4.0 & 4.0 & 0.0 & 0.0 \\ \hline 1.0 & -9.0 & 1.0 & 0.0 \\ 2.0 & 2.0 & -3.0 & 5.0 \\ 1.0 & 3.0 & 2.0 & -5.0 \end{pmatrix} \Longrightarrow \left(\begin{array}{r|rrr} -4.0 & 4.0 & 0.0 & 0.0 \\ 0.0 & -8.0 & 1.0 & 0.0 \\ 0.0 & 4.0 & -3.0 & 5.0 \\ 0.0 & 4.0 & 2.0 & -5.0 \end{array}\right)\begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

$$\begin{array}{l|} \text{Multipliers} \\ \\ 0.50 \\ 0.50 \end{array} \left(\begin{array}{r|rrr} -4.0 & 4.0 & 0.0 & 0.0 \\ 0.0 & -8.0 & 1.0 & 0.0 \\ \hline 0.0 & 4.0 & -3.0 & 5.0 \\ 0.0 & 4.0 & 2.0 & -5.0 \end{array}\right) \Longrightarrow \left(\begin{array}{rr|rr} -4.0 & 4.0 & 0.0 & 0.0 \\ 0.0 & -8.0 & 1.0 & 0.0 \\ \hline 0.0 & 0.0 & -2.5 & 5.0 \\ 0.0 & 0.0 & 2.5 & -5.0 \end{array}\right) \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

$$\begin{array}{l|} \text{Multipliers} \\ \\ \\ 1.0 \end{array} \left(\begin{array}{rr|rr} -4.0 & 4.0 & 0.0 & 0.0 \\ 0.0 & -8.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & -2.5 & 5.0 \\ \hline 0.0 & 0.0 & 2.5 & -5.0 \end{array}\right) \Longrightarrow \left(\begin{array}{rrr|r} -4.0 & 4.0 & 0.0 & 0.0 \\ 0.0 & -8.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & -2.5 & 5.0 \\ \hline 0.0 & 0.0 & 0.0 & 0.0 \end{array}\right) \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

At the end of these three steps, the coefficient matrix has the following upper triangular form;

$$\begin{pmatrix} -4.0 & 4.0 & 0.0 & 0.0 \\ 0.0 & -8.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & -2.5 & 5.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix} \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Observe that, since the pivotal elements are the largest in each column, the multipliers do not exceed 1. Explicit pivoting which is used in general systems of linear equation to ensure that multipliers do not exceed 1, is generally not needed for solving Markov chain problems, since the elements along the diagonal are the largest in each column and this property is maintained during the reduction phase.

We make one additional remark concerning the reduction process. In some cases it can be useful to keep the multipliers for future reference. Since elements below the diagonal are set to zero during the reduction, these array positions provide a suitable location into which the multipliers may be stored. In this case, the coefficient matrix ends up as

$$\begin{pmatrix} -4.0 & 4.0 & 0.0 & 0.0 \\ 0.25 & -8.0 & 1.0 & 0.0 \\ 0.50 & 0.5 & -2.5 & 5.0 \\ 0.25 & 0.5 & 1.0 & 0.0 \end{pmatrix}$$

but only the part on and above the diagonal participates in the backsubstitution phase.

We now turn to the backsubstitution phase wherein π_4 is found from the last equation of the reduced system, π_3 found from the second last equation and so on until π_1 is obtained from the first equation. However, the last row contains all zeros (we ignore all multipliers that may have been stored during the reduction step) and does not allow us to compute the value of π_4. This should not be unexpected, because the coefficient matrix is singular and we can only compute the solution to a multiplicative constant. One possibility is to assign the value 1 to π_4 and to compute the value of the other π_i in terms of this value. Doing so, we obtain

$$\begin{array}{llcl} \text{Equation 4:} & & & \pi_4 = 1, \\ \text{Equation 3:} & -2.5\,\pi_3 + 5 \times 1 = 0 & \Longrightarrow & \pi_3 = 2, \\ \text{Equation 2:} & -8\,\pi_2 + 1 \times 2 = 0 & \Longrightarrow & \pi_2 = 0.25, \\ \text{Equation 1:} & -4\,\pi_1 + 4 \times 0.25 = 0 & \Longrightarrow & \pi_1 = 0.25. \end{array}$$

Therefore our computed solution is $(1/4,\ 1/4,\ 2,\ 1)$. However, this is not the stationary distribution vector just yet, since the sum of its elements does not add to 1. The stationary distribution vector is obtained after normalization, i.e., once we divide each element of this vector by the sum of all its components. Since all the components are positive, this is the same as dividing each element by the 1-norm of the vector. In our example, we have

$$\|\pi\|_1 = |\pi_1| + |\pi_2| + |\pi_3| + |\pi_4| = 3.5,$$

and the stationary distribution vector, the normalized computed solution, is therefore

$$\pi = \frac{2}{7}(1/4,\ 1/4,\ 2,\ 1).$$

Another way to look on this situation is to observe that the system of equations $\pi Q = 0$ does not tell the whole story. We also know that $\pi e = 1$. The n equations of $\pi Q = 0$ provide only $n - 1$ *linearly independent* equations, but together with $\pi e = 1$, we have a complete basis set. For example, it is possible to replace the last equation of the original system with $\pi e = 1$ which eliminates the need for any further normalization. In this case the coefficient matrix becomes nonsingular, the right-hand side becomes nonzero and a unique solution, the stationary probability distribution, is computed. Of course, it is not necessary to replace the last equation of the system by this normalization equation. Indeed, any equation could be replaced. However, this is generally undesirable, for it will entail more numerical computation. For example, if the first equation is replaced, the first row of the coefficient matrix will contain all ones and the right-hand side will be $e_1 = (1, 0, \ldots, 0)^T$. The first consequence of this is that during the reduction phase, the entire sequence of elementary row operations must be performed on the right-hand side vector, e_1, whereas if the last equation is replaced, the right-hand side is unaffected by the elementary row operations. The second and more damaging consequence is that substantial fill-in, the situation in which elements of the coefficient matrix that were previously zero become nonzero, will occur since a multiple of the first row, which contains all ones, will be added into higher numbered rows and a cascading effect will undoubtedly occur in all subsequent reduction steps. An equally viable alternative to replacing the last equation with $\pi e = 1$ is to set the last component π_n to one, perform the back substitution and then to normalize the solution thus computed, just as we did in Example 10.5.

To summarize, Gaussian elimination for solving the balance equations arising from finite, irreducible, continuous-time Markov chains, consists of the following three-step algorithm, where we have taken $A = Q^T$ (i.e., the element a_{ij} in the algorithm below is the rate of transition from state j into state i).

Algorithm 10.1: Gaussian Elimination for Continuous-Time Markov Chains

```
1. The reduction step:
       For $i = 1, 2, \ldots, n-1$ :
           $a_{ji} = -a_{ji}/a_{ii}$            for all $j > i$       % multiplier for row $j$
           $a_{jk} = a_{jk} + a_{ji}a_{ik}$     for all $j, k > i$    % reduce using row $i$

2. The backsubstitution step:
       $x_n = 1$                                % set last component equal to 1
       For $i = n-1, n-2, \ldots, 1$ :
           $x_i = -\left[\sum_{j=i+1}^{n} a_{ij}x_j\right]/a_{ii}$    % backsubstitute to get $x_i$

3. The final normalization step:
       norm $= \sum_{j=1}^{n} x_j$              % sum components
       For $i = 1, 2, \ldots, n$ :
           $\pi_i = x_i/$ norm                  % component $i$ of stationary probability vector
```

An alternative to the straightforward application of Gaussian elimination to Markov chains as described above, is the *scaled Gaussian elimination* algorithm. In this approach, each equation of the system $Q^T\pi^T = 0$ is scaled so that the element on each diagonal is equal to -1. This is accomplished by dividing each element of a row by the absolute value of its diagonal element, which is just the same as dividing both left and right sides of each equation by the same value.

Example 10.6 Observe that the two sets of equations

$$\begin{pmatrix} -4.0 & 4.0 & 0.0 & 0.0 \\ 1.0 & -9.0 & 1.0 & 0.0 \\ 2.0 & 2.0 & -3.0 & 5.0 \\ 1.0 & 3.0 & 2.0 & -5.0 \end{pmatrix} \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

and

$$\begin{pmatrix} -1 & 1 & 0 & 0 \\ 1/9 & -1 & 1/9 & 0 \\ 2/3 & 2/3 & -1 & 5/3 \\ 1/5 & 3/5 & 2/5 & -1 \end{pmatrix} \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

are identical from an equation solving point of view.

This has the effect of simplifying the backsubstitution phase. It does not significantly decrease the the number of numerical operations that must be performed, but rather facilitates programming the Gaussian elimination method for Markov chains. The previous three-step algorithm now becomes:

ALGORITHM 10.2: SCALED GAUSSIAN ELIMINATION FOR CONTINUOUS-TIME MARKOV CHAINS

1. The reduction step:

```
For $i = 1, 2, \ldots, n-1$:
    $a_{ik} = -a_{ik}/a_{ii}$              for all $k > i$        % scale row $i$
    $a_{jk} = a_{jk} + a_{ji}a_{ik}$       for all $j, k > i$     % reduce using row $i$
```

2. The backsubstitution step:

```
$x_n = 1$                                   % set last component equal to1
For $i = n-1, n-2, \ldots, 1$:
    $x_i = \sum_{j=i+1}^{n} a_{ij}x_j$      % backsubstitute to get $x_i$
```

3. The final normalization step:

```
norm $= \sum_{j=1}^{n} x_j$          % sum components
For $i = 1, 2, \ldots, n$:
    $\pi_i = x_i /$ norm             % component $i$of stationary probability vector
```

Observe that during the reduction step no attempt is made actually to set the diagonal elements to -1: it suffices to realize that this must be the case. Nor is any element below the diagonal set to zero, for exactly the same reason. If the multipliers are to be kept, then they may overwrite these subdiagonal positions. At the end of the reduction step, the elements *strictly above* the diagonal of A contain that portion of the upper triangular matrix U needed for the backsubstitution step.

Example 10.7 Beginning with

$$A = \begin{pmatrix} -4.0 & 4.0 & 0.0 & 0.0 \\ 1.0 & -9.0 & 1.0 & 0.0 \\ 2.0 & 2.0 & -3.0 & 5.0 \\ 1.0 & 3.0 & 2.0 & -5.0 \end{pmatrix},$$

the matrices obtained for each different value of $i = 1, 2, 3$ during the reduction phase are

$$A_1 = \begin{pmatrix} -4.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & -8.0 & 1.0 & 0.0 \\ 2.0 & 4.0 & -3.0 & 5.0 \\ 1.0 & 4.0 & 2.0 & -5.0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} -4.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & -8.0 & 0.125 & 0.0 \\ 2.0 & 4.0 & -2.5 & 5.0 \\ 1.0 & 4.0 & 2.5 & -5.0 \end{pmatrix},$$

$$A_3 = \begin{pmatrix} -4.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & -8.0 & 0.125 & 0.0 \\ 2.0 & 4.0 & -2.5 & 2.0 \\ 1.0 & 4.0 & 2.5 & 0.0 \end{pmatrix}.$$

In this last matrix, only the elements above the diagonal contribute to the backsubstitution step. Given that the diagonal elements are taken equal to -1, this means that the appropriate upper triangular matrix is

$$U = \begin{pmatrix} -1.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & -1.0 & 0.125 & 0.0 \\ 0.0 & 0.0 & -1.0 & 2.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}.$$

The backsubstitution step, beginning with $x_4 = 1$, successively gives $x_3 = 2 \times 1 = 2$, $x_2 = 0.125 \times 2 = 0.25$ and $x_1 = 1.0 \times 0.25 = 0.25$ which when normalized gives exactly the same result as before.

LU Factorizations for Markov Chains

When the coefficient matrix in a system of linear equations $Ax = b$ can be written as the product of a lower triangular matrix L and an upper triangular matrix U, then

$$Ax = LUx = b$$

and the solution can be found by first solving (by forward substitution) $Lz = b$ for an intermediate vector z and then solving (by backward substitution) $Ux = z$ for the solution x. The upper triangular matrix obtained by the reduction phase of Gaussian elimination provides an upper triangular matrix U to go along with a lower triangular matrix L whose diagonal elements are all equal to 1 and whose subdiagonal elements are the multipliers with a minus sign in front of them. In the Markov chain context, the system of equations is homogeneous and the coefficient matrix is singular,

$$Q^T x = (LU)x = 0.$$

If we now set $Ux = z$ and attempt to solve $Lz = 0$, we find that, since L is nonsingular (it is a triangular matrix whose diagonal elements are all equal to 1) we must have $z = 0$. This means that we may proceed directly to the back substitution on $Ux = z = 0$ with $u_{nn} = 0$. It is evident that we may assign any nonzero value to x_n, say $x_n = \eta$, and then determine, by simple backsubstitution, the remaining elements of the vector x in terms of η. We have $x_i = c_i \eta$ for some constants $c_i, i = 1, 2, ..., n$, and $c_n = 1$. Thus the solution obtained depends on the value of η.

There still remains one equation that the elements of a probability vector must satisfy, namely that the sum of the probabilities must be 1. Normalizing the solution obtained from solving $Ux = 0$ yields the desired unique stationary probability vector π.

Example 10.8 With the matrix Q^T of Example 10.5, given by

$$Q^T = \begin{pmatrix} -4.0 & 4.0 & 0.0 & 0.0 \\ 1.0 & -9.0 & 1.0 & 0.0 \\ 2.0 & 2.0 & -3.0 & 5.0 \\ 1.0 & 3.0 & 2.0 & -5.0 \end{pmatrix},$$

we find the following LU decomposition:

$$L = \begin{pmatrix} 1.00 & 0.0 & 0.0 & 0.0 \\ -0.25 & 1.0 & 0.0 & 0.0 \\ -0.50 & -0.5 & 1.0 & 0.0 \\ -0.25 & -0.5 & -1.0 & 1.0 \end{pmatrix}, \quad U = \begin{pmatrix} -4.0 & 4.0 & 0.0 & 0.0 \\ 0.0 & -8.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & -2.5 & 5.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix},$$

whose product LU is equal to Q^T. The upper triangular matrix U is just the reduced matrix given by Gaussian elimination while the lower triangular matrix L is the identity matrix with the negated multipliers placed below the diagonal. Forward substitution on $Lz = 0$ successively gives $z_1 = 0,\ z_2 = 0,\ z_3 = 0$, and $z_4 = 0$. Since we know in advance that this must be the case, there is no real need to carry out these operations, but instead we may go straight to the backsubstitution stage to compute x from $Ux = z = 0$. The sequence of operations is identical to that of the backsubstitution phase of Example 10.5.

The above discussion concerns *finite, irreducible* Markov chains. If the Markov chain is decomposable into k irreducible closed communicating classes, these k irreducible components may be solved independently as separate irreducible Markov chains. If Gaussian elimination is applied to the entire transition rate matrix of a Markov chain with $k > 1$ separable components, the reduction phase will lead to a situation in which there are k rows whose elements are all equal to zero. This is because such a matrix possesses k eigenvalues all equal to zero; only $n - k$ of the equations are linearly independent. Although our algorithms can be adjusted to take this situation into account, it is much easier to treat each of the k components separately.

An issue of concern in the implementation of direct methods is that of the data structure used to hold the coefficient matrix. Frequently the matrices generated from Markov models are too large to permit regular two-dimensional arrays to be used to store them in computer memory. Since these matrices are usually very sparse, it is economical, and indeed necessary, to use some sort of packing scheme whereby only the nonzero elements and their positions in the matrix are stored. The most suitable candidates for solution by direct methods are Markov chains whose transition matrix is small, of the order of a few hundred, or when it is *banded*, i.e., the only nonzero elements of the coefficient matrix are not too far from the diagonal. In this latter case, it means that an ordering can be imposed on the states such that no single step transition from state i will take it to states numbered greater than $i + \delta$ or less than $i - \delta$. All fill-in will occur within a distance δ of the diagonal, and the amount of computation per step is proportional to δ^2.

Direct methods are generally not recommended when the transition matrix is large and *not* banded, due to the amount of fill-in that can quickly overwhelm available storage capacity. When the coefficient matrix is generated row by row the following approach allows matrices of the order of several thousand to be handled by a direct method. The first row is generated and stored in a compacted form, i.e., only the nonzero element and its position in the row is stored. Immediately after the second row has been obtained, it is possible to eliminate the element in position (2,1) by adding a multiple of the first row to it. Row 2 may now be compacted and stored. This process may be continued so that when the i-th row of the coefficient matrix is generated, rows 1 through

$(i-1)$ have been derived, reduced to upper triangular form, compacted and stored. The first $(i-1)$ rows may therefore be used to eliminate all nonzero elements in row i from column positions $(i, 1)$ through $(i, i-1)$, thus putting it into the desired triangular form. Note that since this reduction is performed on Q^T, it is the columns of the infinitesimal generator that are required to be generated one at a time and not its rows.

This method has a distinct advantage in that once a row has been generated in this fashion, no more fill-in will occur into this row. It is suggested that a separate storage area be reserved to hold temporarily a single unreduced row. The reduction is performed in this storage area. Once completed, the reduced row may be compacted into any convenient form and appended to the rows which have already been reduced. In this way no storage space is wasted holding subdiagonal elements which, due to elimination, have become zero, nor in reserving space for the inclusion of additional elements. The storage scheme should be chosen bearing in mind the fact that these rows will be used in the reduction of further rows and also later in the algorithm during the back-substitution phase.

This approach cannot be used for solving general systems of linear equations because it inhibits a pivoting strategy from being implemented. It is valid when solving irreducible Markov chains since pivoting is not required in order for the reduction phase to be performed in a stable manner.

The Grassmann–Taksar–Heyman Advantage

It is appropriate at this point to mention a version of Gaussian elimination that has attributes that appear to make it even more stable than the usual version. This procedure is commonly referred to as the GTH (Grassmann–Taksar–Heyman) algorithm. In GTH the diagonal elements are obtained by summing off-diagonal elements rather than performing a subtraction: it is known that subtractions can sometimes lead to loss of significance in numerical computations. These subtractions occur in forming the diagonal elements during the reduction process. Happily, it turns out that at the end of each reduction step, the unreduced portion of the matrix is the transpose of a transition rate matrix in its own right. This has a probabilistic interpretation based on the restriction of the Markov chain to a reduced set of states and it is in this context that the algorithm is generally developed. It also means that the diagonal elements may be formed by adding off-diagonal elements and placing a minus sign in front of this sum instead of performing a single subtraction. When in addition, the concept of scaling is also introduced into the GTH algorithm the need to actually form the diagonal elements disappears (all are taken to be equal to -1). In this case, the scale factor is obtained by taking the reciprocal of the sum of off-diagonal elements.

Example 10.9 Let us return to Example 10.7 and the matrices obtained at each step of Gaussian elimination with scaling.

$$\left(\begin{array}{|cccc}\hline -\mathbf{4.0} & 4.0 & 0.0 & 0.0\\ \mathbf{1.0} & -9.0 & 1.0 & 0.0\\ \mathbf{2.0} & 2.0 & -3.0 & 5.0\\ \mathbf{1.0} & 3.0 & 2.0 & -5.0\end{array}\right) \longrightarrow \left(\begin{array}{c|ccc}-4.0 & 1.0 & 0.0 & 0.0\\ \hline 1.0 & -\mathbf{8.0} & 1.0 & 0.0\\ 2.0 & \mathbf{4.0} & -3.0 & 5.0\\ 1.0 & \mathbf{4.0} & 2.0 & -5.0\end{array}\right)$$

$$\longrightarrow \left(\begin{array}{cc|cc}-4.0 & 1.0 & 0.0 & 0.0\\ 1.0 & -8.0 & 0.125 & 0.0\\ \hline 2.0 & 4.0 & -\mathbf{2.5} & 5.0\\ 1.0 & 4.0 & \mathbf{2.5} & -5.0\end{array}\right) \longrightarrow \left(\begin{array}{ccc|c}-4.0 & 1.0 & 0.0 & 0.0\\ 1.0 & -8.0 & 0.125 & 0.0\\ 2.0 & 4.0 & -2.5 & 2.0\\ \hline 1.0 & 4.0 & 2.5 & \mathbf{0.0}\end{array}\right).$$

Observe that the submatrix contained in each lower right-hand block is the transpose of a transition rate matrix with one less state than its predecessor. Consider the first reduction step, whereby elements in positions a_{21}, a_{31}, and a_{41} are to be eliminated. Off-diagonal elements in row 1 are scaled by dividing each by the sum $a_{21} + a_{31} + a_{41} = 1 + 2 + 1$, although for this first row, it is just as easy to use the absolute value of the diagonal element. After the first row is scaled, it must be added into the second row to eliminate the a_{21} element. While this should result in the subtraction $-9+1$ into position a_{22}, we choose instead to ignore this particular operation and leave the a_{22} element equal to -9. Throughout the entire reduction process, the diagonal elements are left unaltered. Once the elements a_{21}, a_{31} and a_{41} have been eliminated, we are ready to start the second reduction step, which begins by forming the next scale factor as $a_{32}+a_{42}=4+4$, the sum of elements below a_{22}. The process continues in this fashion, first computing the scale factor by summing below the current diagonal element, then scaling all elements to the right of the current diagonal element and and finally by carrying out the reduction step. These steps are shown below.

$$\left(\begin{array}{|rrrr}\hline \mathbf{-4.0} & 4.0 & 0.0 & 0.0\\ \mathbf{1.0} & -9.0 & 1.0 & 0.0\\ \mathbf{2.0} & 2.0 & -3.0 & 5.0\\ \mathbf{1.0} & 3.0 & 2.0 & -5.0\end{array}\right) \overset{\text{scale}}{\longrightarrow} \left(\begin{array}{|rrrr}\mathbf{-4.0} & 1.0 & 0.0 & 0.0\\ \hline \mathbf{1.0} & -9.0 & 1.0 & 0.0\\ \mathbf{2.0} & 2.0 & -3.0 & 5.0\\ \mathbf{1.0} & 3.0 & 2.0 & -5.0\end{array}\right)$$

$$\overset{\text{reduce}}{\longrightarrow} \left(\begin{array}{r|rrr}-4.0 & 1.0 & 0.0 & 0.0\\ \hline 1.0 & \mathbf{-9.0} & 1.0 & 0.0\\ 2.0 & \mathbf{4.0} & -3.0 & 5.0\\ 1.0 & \mathbf{4.0} & 2.0 & -5.0\end{array}\right) \overset{\text{scale}}{\longrightarrow} \left(\begin{array}{r|rrr}-4.0 & 1.0 & 0.0 & 0.0\\ 1.0 & \mathbf{-9.0} & 0.125 & 0.0\\ \hline 2.0 & \mathbf{4.0} & -3.0 & 5.0\\ 1.0 & \mathbf{4.0} & 2.0 & -5.0\end{array}\right)$$

$$\overset{\text{reduce}}{\longrightarrow} \left(\begin{array}{rr|rr}-4.0 & 1.0 & 0.0 & 0.0\\ 1.0 & -9.0 & 0.125 & 0.0\\ \hline 2.0 & 4.0 & \mathbf{-3.0} & 5.0\\ 1.0 & 4.0 & \mathbf{2.5} & -5.0\end{array}\right) \overset{\text{scale}}{\longrightarrow} \left(\begin{array}{rr|rr}-4.0 & 1.0 & 0.0 & 0.0\\ 1.0 & -9.0 & 0.125 & 0.0\\ 2.0 & 4.0 & -3.0 & 2.0\\ \hline 1.0 & 4.0 & 2.5 & \mathbf{-5.0}\end{array}\right).$$

Algorithm 10.3: GTH for Continuous-Time Markov Chains with $A = Q^T$

```
1. The reduction step:
       For $i = 1, 2, \ldots, n-1$:
           $a_{ik} = a_{ik} / \sum_{j=i+1}^{n} a_{ji}$    for all $k > i$             % scale row $i$
           $a_{jk} = a_{jk} + a_{ji} a_{ik}$             for all $j, k > i,\ k \neq j$   % reduce using row $i$
2. The backsubstitution step:
       $x_n = 1$                                   % set last component equal to1
       For $i = n-1, n-2, \ldots, 1$:
           $x_i = \sum_{j=i+1}^{n} a_{ij} x_j$                 % backsubstitute to get $x_i$
3. The final normalization step:
       norm $= \sum_{j=1}^{n} x_j$        %sum components
       For $i = 1, 2, \ldots, n$:
           $\pi_i = x_i/$norm        % component $i$ of stationary probability vector
```

Comparing this algorithm with the scaled Gaussian elimination algorithm, we see that only the first step has changed, and within that step only in the computation of the scale factor. The GTH implementation requires more numerical operations than the standard implementation but this may be offset by a gain in precision when the matrix Q is ill conditioned. The extra additions are not very costly when compared with the overall cost of the elimination procedure, which leads to the conclusion that the GTH advantage should be exploited where possible in elimination procedures.

If the transition rate matrix is stored in a two-dimensional or band storage structure, access is easily available to both the rows and columns of Q, and there is no difficulty in implementing GTH. Unfortunately, the application of GTH is not quite so simple when the coefficient matrix is stored in certain compacted representations.

Matlab code for Gaussian Elimination

```
function [pi] = GE(Q)

A = Q’;
n = size(A);
for i=1:n-1
    for j=i+1:n
        A(j,i) = -A(j,i)/A(i,i);
    end
    for j=i+1:n
        for k=i+1:n
            A(j,k) = A(j,k) + A(j,i)*A(i,k);
        end
    end
end

x(n) = 1;
for i=n-1:-1:1
    for j=i+1:n
        x(i) = x(i) + A(i,j)*x(j);
    end
    x(i) = -x(i)/A(i,i);
end

pi = x/norm(x,1);
```

Matlab code for Scaled Gaussian Elimination

```
function [pi] = ScaledGE(Q)

A = Q’;
n = size(A);
for i=1:n-1
   for k=i+1:n
      A(i,k) = -A(i,k)/A(i,i);
   end
   for j=i+1:n
      for k=i+1:n
         A(j,k) = A(j,k) + A(j,i)*A(i,k);
      end
   end
end

x(n) = 1;
for i=n-1:-1:1
```

```
    for j=i+1:n
        x(i) = x(i) + A(i,j)*x(j);
    end
end

pi = x/norm(x,1);
```

Matlab code for GTH

```
function [pi] = GTH(Q)

A = Q';
n = size(A);
for i=1:n-1
    scale = sum(A(i+1:n,i));
    for k=i+1:n
        A(i,k) = A(i,k)/scale;
    end
    for j=i+1:n
        for k=i+1:n
            A(j,k) = A(j,k) + A(j,i)*A(i,k);
        end
    end
end

x(n) = 1;
for i=n-1:-1:1
    for j=i+1:n
        x(i) = x(i) + A(i,j)*x(j);
    end
end

pi = x/norm(x,1);
```

10.3 Basic Iterative Methods for Stationary Distributions

Iterative methods for solving systems of equations begin with some approximation to, or guess at, the solution, and successively apply numerical operations designed to make this approximation approach the true solution. The coefficient matrix is not altered during the execution of the algorithm which makes iterative methods well suited to compacted storage schemes. An item of constant concern with iterative methods is their rate of convergence, the speed at which the initial approximation approaches the solution.

10.3.1 The Power Method

Perhaps the approach that first comes to mind when we need to find the stationary distribution of an finite, ergodic, discrete-time Markov chain, is to let the chain evolve over time, step by step, until it reaches its stationary distribution. Once the probability vector no longer changes as the process evolves from some step n to step $n+1$, that vector can be taken as the stationary probability vector, since at that point we have $zP = z$.

Example 10.10 Consider a discrete-time Markov chain whose matrix of transition probabilities is

$$P = \begin{pmatrix} .0 & .8 & .2 \\ .0 & .1 & .9 \\ .6 & .0 & .4 \end{pmatrix}. \tag{10.7}$$

If the system starts in state 1, the initial probability vector is given by

$$\pi^{(0)} = (1,\ 0,\ 0).$$

Immediately after the first transition, the system will be either in state 2, with probability .8, or in state 3, with probability .2. The vector $\pi^{(1)}$, which denotes the probability distribution after one transition (or one step) is thus

$$\pi^{(1)} = (0,\ .8,\ .2).$$

Notice that this result may be obtained by forming the product $\pi^{(0)}P$.

The probability of being in state 1 after two time steps is obtained by summing (over all i) the probability of being in state i after one step, given by $\pi_i^{(1)}$, multiplied by the probability of making a transition from state i to state 1. We have

$$\sum_{i=1}^{3} \pi_i^{(1)} p_{i1} = \pi_1^{(1)} \times .0 + \pi_2^{(1)} \times .0 + \pi_3^{(1)} \times .6 = .12.$$

Likewise, the system will be in state 2 after two steps with probability .08 ($= .0\times.8+.8\times.1+.2\times.0$), and in state 3 with probability .8 ($= .0 \times .2 + .8 \times .9 + .2 \times .4$). Thus, given that the system begins in state 1, we have the following probability distribution after two steps:

$$\pi^{(2)} = (.12,\ .08,\ .8).$$

Notice once again that $\pi^{(2)}$ may be obtained by forming the product $\pi^{(1)}P$:

$$\pi^{(2)} = (.12,\ .08,\ .8) = (0.0,\ 0.8,\ 0.2) \begin{pmatrix} .0 & .8 & .2 \\ .0 & .1 & .9 \\ .6 & .0 & .4 \end{pmatrix} = \pi^{(1)}P.$$

We may continue in this fashion, computing the probability distribution after each transition step. For any integer k, the state of the system after k transitions is obtained by multiplying the probability vector obtained after $(k-1)$ transitions by P. Thus

$$\pi^{(k)} = \pi^{(k-1)}P = \pi^{(k-2)}P^2 = \cdots = \pi^{(0)}P^k.$$

At step $k = 25$, we find the probability distribution to be

$$\pi = (.2813,\ .2500,\ .4688),$$

and thereafter, correct to four decimal places,

$$(.2813,\ .2500,\ .4688) \begin{pmatrix} .0 & .8 & .2 \\ .0 & .1 & .9 \\ .6 & .0 & .4 \end{pmatrix} = (.2813,\ .2500,\ .4688),$$

which we may now take to be the stationary distribution (correct to four decimal places).

When the Markov chain is finite, aperiodic, and irreducible (as in Example 10.10), the vectors $\pi^{(k)}$ converge to the stationary probability vector π regardless of the choice of initial vector. We have

$$\lim_{k\to\infty} \pi^{(k)} = \pi.$$

This method of determining the stationary probability vector is referred to as the *power method* or *power iteration*. The power method is well known in the context of determining the right-hand eigenvector corresponding to a dominant eigenvalue of a matrix, A, and it is in this context that we shall examine its convergence properties. However, recall that the stationary distribution of a Markov chain is obtained from the *left-hand* eigenvector, so that in a Markov chain context, the matrix A must be replaced with P^T, the transpose of the transition probability matrix. Let A be a square matrix of order n. The power method is described by the iterative procedure

$$z^{(k+1)} = \frac{1}{\xi_k} A z^{(k)}, \tag{10.8}$$

where ξ_k is a normalizing factor, typically $\xi_k = \|Az^{(k)}\|_\infty$, and $z^{(0)}$ is an arbitrary starting vector. Although this formulation of the power method incorporates a normalization at *each* iteration, whereby each element of the newly formed iterate is divided by ξ_k, this is not strictly necessary. Normalization may be performed less frequently.

To examine the rate of convergence of the power method, let A have eigensolution

$$Ax_i = \lambda_i x_i, \qquad i = 1, 2, \ldots, n,$$

and suppose that

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|.$$

Let us further assume that the initial vector may be written as a linear combination of the eigenvectors of A, i.e.,

$$z^{(0)} = \sum_{i=1}^{n} \alpha_i x_i.$$

The rate of convergence of the power method may then be determined from the relationship

$$z^{(k)} = A^k z^{(0)} = A^k \sum_{i=1}^{n} \alpha_i x_i = \sum_{i=1}^{n} \alpha_i A^k x_i = \sum_{i=1}^{n} \alpha_i \lambda_i^k x_i = \lambda_1^k \left\{ \alpha_1 x_1 + \sum_{i=2}^{n} \alpha_i \left(\frac{\lambda_i}{\lambda_1} \right)^k x_i \right\}. \tag{10.9}$$

It may be observed that the process converges to the dominant eigenvector x_1. The rate of convergence depends on the ratios $|\lambda_i|/|\lambda_1|$ for $i = 2, 3, \ldots, n$. The smaller these ratios, the quicker the summation on the right-hand side tends to zero. It is, in particular, the magnitude of the subdominant eigenvalue, λ_2, that determines the convergence rate. The power method will not perform satisfactorily when $|\lambda_2| \approx |\lambda_1|$. Obviously major difficulties arise when $|\lambda_2| = |\lambda_1|$.

Example 10.11 Returning to the 3×3 example given by Equation (10.7), the eigenvalues of P are $\lambda_1 = 1$ and $\lambda_{2,3} = -.25 \pm .5979i$. Thus $|\lambda_2| \approx .65$. Notice that $.65^{10} \approx .01$, $.65^{25} \approx 2 \times 10^{-5}$, and $.65^{100} \approx 2 \times 10^{-19}$. Table 10.1 presents the probability distribution of the states of this example at specific steps, for each of three different starting configurations. After 25 iterations, no further changes are observed in the first four digits for any of the starting configurations. The table shows that approximately two decimal places of accuracy have been obtained after 10 iterations and four places after 25 iterations, which coincides with convergence guidelines obtained from the subdominant eigenvalue. Furthermore, after 100 iterations it is found that the solution is accurate to full machine precision, again as suggested by the value of $.65^{100}$.

We hasten to point out that the magnitude of $|\lambda_2|^k$ does not *guarantee* a certain number of decimal places of accuracy in the solution, as might be construed from the preceding example. As a general rule, the number of decimal places accuracy is determined from a *relative* error norm; a relative error norm of 10^{-j} yielding approximately j decimal places of accuracy. However, some flexibility in the Markov chain context may be appropriate since both matrix and vectors have unit 1-norms.

Table 10.1. Convergence in power method.

Step	Initial state			Initial state			Initial state		
	1.0	.0	.0	.0	1.0	.0	.0	.0	1.0
1	.0000	.8000	.2000	.0000	.1000	.9000	.6000	.0000	.4000
2	.1200	.0800	.8000	.5400	.0100	.4500	.2400	.4800	.2800
3	.4800	.1040	.4160	.2700	.4330	.2970	.1680	.2400	.5920
4	.2496	.3944	.3560	.1782	.2593	.5626	.3552	.1584	.4864
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
10	.2860	.2555	.4584	.2731	.2573	.4696	.2827	.2428	.4745
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
25	.2813	.2500	.4688	.2813	.2500	.4688	.2813	.2500	.4688

Although omitted from Equation (10.9), in the general formulation of the power method, it is usually necessary to normalize successive iterates, since otherwise the term λ_1^k may cause successive approximations to become too large (if $\lambda_1 > 1$) or too small (if $\lambda_1 < 1$) and may result in overflow or underflow. Additionally, this normalization is required to provide a standardized vector with which to implement convergence testing. However, in Markov chain problems, the coefficient matrix has 1 as a dominant eigenvalue ($\lambda_1 = 1$) and the requirement for periodic normalization of iterates in the power method disappears. Indeed, if the initial starting approximation is a probability vector, all successive approximations will also be probability vectors. As we noted earlier, when the power method is applied in the Markov chain context, it is the left-hand eigenvector corresponding to a unit eigenvalue that is required and so the matrix to which the method is applied is P^T and the above iteration, Equation (10.8), takes the form

$$z^{(k+1)} = P^T z^{(k)}. \tag{10.10}$$

It is known that the unit eigenvalue of a stochastic matrix is a dominant eigenvalue and that if the matrix is irreducible, there are no other unit eigenvalues. When the matrix is periodic, however, there exist other eigenvalues on the unit circle, which are different from 1 but whose modulus is equal to 1. A straightforward application of the power method in this case will fail. This situation may be circumvented by a slight modification that leaves the unit eigenvalue and its corresponding eigenvector unchanged. The matrix P is usually obtained from the infinitesimal generator by means of the relationship

$$P = (Q\Delta t + I),$$

where $\Delta t \leq 1/\max_i |q_{ii}|$. If Δt is chosen so that $\Delta t < 1/\max_i |q_{ii}|$, the resulting stochastic matrix has diagonal elements $p_{ii} > 0$ and therefore cannot be periodic. Under these conditions (irreducible and aperiodic), the power method can be guaranteed to converge. Its rate of convergence is governed by the ratio $|\lambda_2|/|\lambda_1|$, i.e., by $|\lambda_2|$.

Unfortunately, the difference between theoretical conditions for the convergence of an iterative method and its observed behavior in practical situations can be quite drastic. What in theory will converge may take such a large number of iterations that for all practical purposes the method should be considered unworkable. This occurs in the power method when the modulus of the subdominant eigenvalue, $|\lambda_2|$, is close to unity. For example, stochastic matrices that are nearly completely decomposable (NCD) arise frequently in modeling physical and mathematical systems; such matrices have subdominant eigenvalues that are necessarily close to 1. In these cases the power method will converge extremely slowly.

Given the fact that it often takes many iterations to achieve convergence, it may be thought that a more economical approach is to repeatedly square the matrix P. Let $k = 2^m$ for some integer m. Using the basic iterative formula $\pi^{(k)} = \pi^{(k-1)}P$ requires k iterations to obtain $\pi^{(k)}$; each iteration includes a matrix-vector product. Repeatedly squaring the matrix requires only m matrix products to determine P^{2^m}, from which $\pi^{(k)} = \pi^{(0)}P^k$ is quickly computed. It may be further speculated that, since a matrix-vector product requires n^2 multiplications and a matrix-matrix product requires n^3, that the squaring approach is to be recommended when $mn^3 < 2^m n^2$, i.e., when $nm < 2^m$. Unfortunately, this analysis completely omits the fact that the matrix P is usually large and sparse. Thus, a matrix-vector product requires only n_z multiplications, where n_z is the number of nonzero elements in P. The matrix-squaring operation will increase the number of nonzero elements in the matrix (in fact, for an irreducible matrix, convergence will not be attained before all the elements have become nonzero), thereby increasing not only the number of multiplications needed but also the amount of memory needed. It is perhaps memory requirements more than time constraints that limit the applicability of matrix powering.

10.3.2 The Iterative Methods of Jacobi and Gauss–Seidel

Iterative solution methods are frequently obtained from the specification of a problem as an equation of the form $f(x) = 0$. The function $f(x)$ may be a linear function, a nonlinear function, or even a system of linear equations, in which case $f(x) = Ax - b$. An iterative method is derived from $f(x) = 0$ by writing it in the form $x = g(x)$ and then constructing the iterative process

$$x^{(k+1)} = g(x^{(k)})$$

with some initial approximation $x^{(0)}$. In other words, the new iterate is obtained by inserting the value at the previous iterate into the right-hand side. The standard and well-known iterative methods for the solution of systems of linear equations are the methods of Jacobi, Gauss–Seidel, and successive overrelaxation (SOR). These methods derive from a nonhomogeneous system of linear equations

$$Ax = b, \quad \text{or, equivalently,} \quad Ax - b = 0,$$

an iterative formula of the form

$$x^{(k+1)} = Hx^{(k)} + c, \qquad k = 0, 1, \ldots. \tag{10.11}$$

This is accomplished by splitting the coefficient matrix A. Given a splitting

$$A = M - N$$

with nonsingular M, we have

$$(M - N)x = b$$

or

$$Mx = Nx + b,$$

which leads to the iterative procedure

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b = Hx^{(k)} + c, \qquad k = 0, 1, \ldots.$$

The matrix $H = M^{-1}N$ is called the *iteration* matrix and it is the eigenvalues of this matrix that determine the rate of convergence of the iterative method. The methods of Jacobi and Gauss–Seidel differ in their choice M and N. We begin with the method of Jacobi.

Consider a nonhomogeneous system of linear equations, $Ax = b$ in which $A \in \Re^{(4\times 4)}$ is nonsingular and $b \neq 0$. Writing this in full, we have

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= b_1,\\
a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= b_2,\\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= b_3,\\
a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= b_4.
\end{aligned}$$

Bringing all terms with off-diagonal components a_{ij}, $i \neq j$ to the right-hand side, we obtain

$$\begin{aligned}
a_{11}x_1 &= \phantom{-a_{21}x_1} -a_{12}x_2 - a_{13}x_3 - a_{14}x_4 + b_1,\\
a_{22}x_2 &= -a_{21}x_1 \phantom{- a_{32}x_2} - a_{23}x_3 - a_{24}x_4 + b_2,\\
a_{33}x_3 &= -a_{31}x_1 - a_{32}x_2 \phantom{- a_{43}x_3} - a_{34}x_4 + b_3,\\
a_{44}x_4 &= -a_{41}x_1 - a_{42}x_2 - a_{43}x_3 \phantom{- a_{34}x_4} + b_4.
\end{aligned}$$

We are now ready to convert this into an iterative procedure. Taking components of x on the right-hand side to be the old values (the values computed at iteration k) allows us to assign new values as follows:

$$\begin{aligned}
a_{11}x_1^{(k+1)} &= \phantom{-a_{21}x_1^{(k)}} -a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - a_{14}x_4^{(k)} + b_1,\\
a_{22}x_2^{(k+1)} &= -a_{21}x_1^{(k)} \phantom{- a_{32}x_2^{(k)}} - a_{23}x_3^{(k)} - a_{24}x_4^{(k)} + b_2,\\
a_{33}x_3^{(k+1)} &= -a_{31}x_1^{(k)} - a_{32}x_2^{(k)} \phantom{- a_{43}x_3^{(k)}} - a_{34}x_4^{(k)} + b_3,\\
a_{44}x_4^{(k+1)} &= -a_{41}x_1^{(k)} - a_{42}x_2^{(k)} - a_{43}x_3^{(k)} \phantom{- a_{34}x_4^{(k)}} + b_4.
\end{aligned} \tag{10.12}$$

This is the *Jacobi* iterative method. In matrix form, A is *split* as $A = D - L - U$ where[2]

- D is a diagonal matrix,
- L is a strictly lower triangular matrix,
- U is a strictly upper triangular matrix,

and so the method of Jacobi becomes equivalent to

$$Dx^{(k+1)} = (L + U)x^{(k)} + b$$

or

$$x^{(k+1)} = D^{-1}(L + U)x^{(k)} + D^{-1}b.$$

Notice that the diagonal matrix D must be nonsingular for this method to be applicable. Thus the method of *Jacobi* corresponds to the splitting $M = D$ and $N = (L + U)$. Its iteration matrix is given by

$$H_J = D^{-1}(L + U).$$

In the Markov chain context, the system of equations whose solution we seek is

$$\pi Q = 0, \quad \text{or, equivalently,} \quad Q^T\pi^T = 0.$$

For notational convenience, set $x = \pi^T$ and let $Q^T = D - (L + U)$. The matrix D is nonsingular, since for all j, $d_{jj} \neq 0$ and so D^{-1} exists. Once the kth approximation $x^{(k)}$ has been formed, the next approximation is obtained by solving the system of equations

$$Dx^{(k+1)} = (L + U)x^{(k)}$$

[2] The matrices L and U should not be confused with the LU factors obtained from direct methods such as Gaussian elimination.

or

$$x^{(k+1)} = D^{-1}(L+U)x^{(k)}.$$

In scalar form,

$$x_i^{(k+1)} = \frac{1}{d_{ii}} \left\{ \sum_{j \neq i} \left(l_{ij} + u_{ij} \right) x_j^{(k)} \right\}, \qquad i = 1, 2, \ldots, n. \tag{10.13}$$

Example 10.12 Consider a four-state Markov chain with stochastic transition probability matrix

$$P = \begin{pmatrix} .5 & .5 & 0 & 0 \\ 0 & .5 & .5 & 0 \\ 0 & 0 & .5 & .5 \\ .125 & .125 & .25 & .5 \end{pmatrix}.$$

Since we are given P rather than Q, we need to write $\pi P = \pi$ as $\pi(P-I) = 0$ and take $Q = P - I$:

$$Q = \begin{pmatrix} -.5 & .5 & 0 & 0 \\ 0 & -.5 & .5 & 0 \\ 0 & 0 & -.5 & .5 \\ .125 & .125 & .25 & -.5 \end{pmatrix}.$$

Transposing this, we obtain the system of equations

$$\begin{pmatrix} -.5 & 0 & 0 & .125 \\ .5 & -.5 & 0 & .125 \\ 0 & .5 & -.5 & .250 \\ 0 & 0 & .5 & -.500 \end{pmatrix} \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Writing this in full, we have

$$\begin{aligned} -.5\pi_1 + 0\pi_2 + 0\pi_3 + .125\pi_4 &= 0, \\ .5\pi_1 - .5\pi_2 + 0\pi_3 + .125\pi_4 &= 0, \\ 0\pi_1 + .5\pi_2 - .5\pi_3 + .25\pi_4 &= 0, \\ 0\pi_1 + 0\pi_2 + .5\pi_3 - .5\pi_4 &= 0, \end{aligned}$$

or

$$\begin{aligned} -.5\pi_1 &= & & & & - .125\pi_4, \\ -.5\pi_2 &= -.5\pi_1 & & & & - .125\pi_4, \\ -.5\pi_3 &= & & -.5\pi_2 & & - .25\pi_4, \\ -.5\pi_4 &= & & & -.5\pi_3. & \end{aligned} \tag{10.14}$$

From this we can write the iterative version,

$$\begin{aligned} -.5\pi_1^{(k+1)} &= & & & & - .125\pi_4^{(k)}, \\ -.5\pi_2^{(k+1)} &= -.5\pi_1^{(k)} & & & & - .125\pi_4^{(k)}, \\ -.5\pi_3^{(k+1)} &= & & -.5\pi_2^{(k)} & & - .25\pi_4^{(k)}, \\ -.5\pi_4^{(k+1)} &= & & & -.5\pi_3^{(k)}, & \end{aligned}$$

which leads to

$$\pi_1^{(k+1)} = .25\pi_4^{(k)}, \quad \pi_2^{(k+1)} = \pi_1^{(k)} + .25\pi_4^{(k)}, \quad \pi_3^{(k+1)} = \pi_2^{(k)} + .5\pi_4^{(k)}, \quad \pi_4^{(k+1)} = \pi_3^{(k)}.$$

We may now begin the iterative process. Starting with

$$\pi^{(0)} = (.5,\ .25,\ .125,\ .125),$$

we obtain

$$\begin{aligned}
\pi_1^{(1)} &= & & .25\pi_4^{(0)} = .25 \times .125 = .03125,\\
\pi_2^{(1)} &= \pi_1^{(0)} & & + .25\pi_4^{(0)} = .5 + .25 \times .125 = .53125,\\
\pi_3^{(1)} &= \quad \pi_2^{(0)} & & + .5\pi_4^{(0)} = .25 + .5 \times .125 = .31250,\\
\pi_4^{(1)} &= \qquad \pi_3^{(0)} & & = .12500.
\end{aligned}$$

In this particular example, the sum of the components of $\pi^{(1)}$ is equal to 1, so a further normalization is not necessary. In fact, since at any iteration $k+1$,

$$\sum_{i=1}^{4} \pi_i^{(k+1)} = .25\pi_4^{(k)} + \pi_1^{(k)} + .25\pi_4^{(k)} + \pi_2^{(k)} + .50\pi_4^{(k)} + \pi_3^{(k)} = \sum_{i=1}^{4} \pi_i^{(k)} = 1,$$

the sum of the components of all the approximations to the stationary distribution will always be equal to 1, provided the initial approximation has components that sum to 1. Continuing with the iterative Jacobi procedure, we obtain the following sequence of approximations:

$$\begin{aligned}
\pi^{(0)} &= (.50000,\ .25000,\ .12500,\ .12500),\\
\pi^{(1)} &= (.03125,\ .53125,\ .31250,\ .12500),\\
\pi^{(2)} &= (.03125,\ .06250,\ .59375,\ .31250),\\
\pi^{(3)} &= (.078125,\ .109375,\ .21875,\ .59375).\\
&\ \vdots
\end{aligned}$$

Notice that substitution of $Q^T = D - (L + U)$ into $Q^T x = 0$ gives $(L + U)x = Dx$, and since D is nonsingular, this yields the eigenvalue equation

$$D^{-1}(L + U)x = x, \tag{10.15}$$

in which x is seen to be the right-hand eigenvector corresponding to a unit eigenvalue of the matrix $D^{-1}(L + U)$. This matrix will immediately be recognized as the iteration matrix for the method of Jacobi, H_J. That H_J has a unit eigenvalue is obvious from Equation (10.15). Furthermore, from the zero-column-sum property of Q^T, we have

$$d_{jj} = \sum_{i=1,\ i\neq j}^{n} (l_{ij} + u_{ij}), \qquad j = 1, 2, \ldots,$$

with $l_{ij},\ u_{ij} \le 0$ for all $i,\ j,\ i \neq j$, and it follows directly from the theorem of Gerschgorin that no eigenvalue of H_J can have modulus greater than unity. This theorem states that the eigenvalues of any square matrix A of order n lie in the union of the n circular disks with centers $c_i = a_{ii}$ and radii $r_i = \sum_{j=1, j\neq i}^{n} |a_{ij}|$. The stationary probability vector π is therefore the eigenvector corresponding to a dominant eigenvalue of H_J, and the method of Jacobi is identical to the power method applied to the iteration matrix H_J.

Example 10.13 Returning to the previous example, the Jacobi iteration matrix is given by

$$H_J = \begin{pmatrix} -.5 & 0 & 0 & 0\\ 0 & -.5 & 0 & 0\\ 0 & 0 & -.5 & 0\\ 0 & 0 & 0 & -.5 \end{pmatrix}^{-1} \begin{pmatrix} 0 & 0 & 0 & -.125\\ -.5 & 0 & 0 & -.125\\ 0 & -.5 & 0 & -.250\\ 0 & 0 & -.5 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & .25\\ 1.0 & 0 & 0 & .25\\ 0 & 1.0 & 0 & .50\\ 0 & 0 & 1.0 & 0 \end{pmatrix}.$$

The four eigenvalues of this matrix are

$$\lambda_1 = 1.0, \quad \lambda_2 = -.7718, \quad \lambda_{3,4} = -0.1141 \pm 0.5576i.$$

The modulus of λ_2 is equal to 0.7718. Observe that $0.7718^{50} = .00000237$, which suggests about five to six places of accuracy after 50 iterations.

We now move on to the method of *Gauss–Seidel*. Usually the computations specified by Equation (10.13) in the method of Jacobi are carried out sequentially; the components of the vector $x^{(k+1)}$ are obtained one after the other as $x_1^{(k+1)}, x_2^{(k+1)}, \ldots, x_n^{(k+1)}$. When evaluating $x_i^{(k+1)}$, only components of the previous iteration $x^{(k)}$ are used, even though elements from the current iteration $x_j^{(k+1)}$ for $j < i$ are available and are (we hope) more accurate. The Gauss–Seidel method makes use of these most recently available component approximations. This may be accomplished by simply overwriting elements as soon as a new approximation is determined.

Referring back to Equation (10.12) and rewriting it using the most recent values, we obtain

$$\begin{aligned}
a_{11}x_1^{(k+1)} &= \phantom{-a_{21}x_1^{(k+1)}} - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - a_{14}x_4^{(k)} + b_1,\\
a_{22}x_2^{(k+1)} &= -a_{21}x_1^{(k+1)} \phantom{- a_{12}x_2^{(k)}} - a_{23}x_3^{(k)} - a_{24}x_4^{(k)} + b_2,\\
a_{33}x_3^{(k+1)} &= -a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)} \phantom{- a_{13}x_3^{(k)}} - a_{34}x_4^{(k)} + b_3,\\
a_{44}x_4^{(k+1)} &= -a_{41}x_1^{(k+1)} - a_{42}x_2^{(k+1)} - a_{43}x_3^{(k+1)} \phantom{- a_{14}x_4^{(k)}} + b_4.
\end{aligned}$$

Observe that in the second equation, the value of the newly computed first component, x_1, is used, i.e., we use $x_1^{(k+1)}$ rather than $x_1^{(k)}$. Similarly, in the third equation we use the new values of x_1 and x_2, and finally in the last equation we use the new values of all components other than the last. With n linear equations in n unknowns, the ith equation is written as

$$a_{ii}x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} \right), \quad i = 1, 2, \ldots, n. \tag{10.16}$$

Rearranging these equations so that all new values appear on the left-hand side, we find

$$\begin{aligned}
a_{11}x_1^{(k+1)} &= -a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - a_{14}x_4^{(k)} + b_1,\\
a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} &= - a_{23}x_3^{(k)} - a_{24}x_4^{(k)} + b_2,\\
a_{31}x_1^{(k+1)} + a_{32}x_2^{(k+1)} + a_{33}x_3^{(k+1)} &= - a_{34}x_4^{(k)} + b_3,\\
a_{41}x_1^{(k+1)} + a_{42}x_2^{(k+1)} + a_{43}x_3^{(k+1)} + a_{44}x_4^{(k+1)} &= b_4.
\end{aligned} \tag{10.17}$$

Using the same $D - L - U$ splitting as for Jacobi, the Gauss–Seidel iterative method is equivalent to

$$(D - L)x^{(k+1)} = Ux^{(k)} + b. \tag{10.18}$$

This is just the matrix representation of the system of equations (10.17). It may be written as

$$x^{(k+1)} = D^{-1}(Lx^{(k+1)} + Ux^{(k)} + b)$$

or

$$x^{(k+1)} = (D - L)^{-1}Ux^{(k)} + (D - L)^{-1}b. \tag{10.19}$$

Thus the iteration matrix for the method of Gauss–Seidel is given by

$$H_{GS} = (D - L)^{-1}U.$$

This iterative method corresponds to the splitting $M = (D - L)$ and $N = U$ and is applicable only when the matrix $D - L$ is nonsingular. Gauss–Seidel usually, but not always, converges faster than Jacobi.

For homogeneous systems of equations such as those which arise in Markov chains, the right-hand side is zero and Equation (10.19) becomes

$$x^{(k+1)} = (D-L)^{-1}Ux^{(k)}, \quad \text{i.e.,} \quad x^{(k+1)} = H_{GS}x^{(k)}.$$

Furthermore, since the diagonal elements of D are all nonzero, the inverse, $(D - L)^{-1}$, exists. The stationary probability vector $\pi = x^T$ obviously satisfies $H_{GS}x = x$, which shows that x is the right-hand eigenvector corresponding to a unit eigenvalue of H_{GS}. As a consequence of the Stein–Rosenberg theorem ([53], p. 70) and the fact that the corresponding Jacobi iteration matrix H_J possesses a dominant unit eigenvalue, the unit eigenvalue of the matrix H_{GS} is a dominant eigenvalue. The method of Gauss–Seidel is therefore identical to the power method applied to H_{GS}.

Example 10.14 We now apply Gauss–Seidel to the problem previously solved by the method of Jacobi:

$$\begin{aligned}
-.5\pi_1 &= & & & -.125\pi_4,\\
-.5\pi_2 &= -.5\pi_1 & & & -.125\pi_4,\\
-.5\pi_3 &= & -.5\pi_2 & & -.25\pi_4,\\
-.5\pi_4 &= & & -.5\pi_3. &
\end{aligned}$$

In the Gauss–Seidel iterative scheme, this becomes

$$\begin{aligned}
-.5\pi_1^{(k+1)} &= & & & -.125\pi_4^{(k)},\\
-.5\pi_2^{(k+1)} &= -.5\pi_1^{(k+1)} & & & -.125\pi_4^{(k)},\\
-.5\pi_3^{(k+1)} &= & -.5\pi_2^{(k+1)} & & -.25\pi_4^{(k)},\\
-.5\pi_4^{(k+1)} &= & & -.5\pi_3^{(k+1)}, &
\end{aligned}$$

$$\pi_1^{(k+1)} = .25\pi_4^{(k)}, \quad \pi_2^{(k+1)} = \pi_1^{(k+1)} + .25\pi_4^{(k)}, \quad \pi_3^{(k+1)} = \pi_2^{(k+1)} + .5\pi_4^{(k)}, \quad \pi_4^{(k+1)} = \pi_3^{(k+1)}.$$

We may now begin the iterative process. Starting with

$$\pi^{(0)} = (.5,\ .25,\ .125,\ .125),$$

we obtain

$$\begin{aligned}
\pi_1^{(1)} &= & & .25\pi_4^{(0)} &&= .25 \times .125 = .03125,\\
\pi_2^{(1)} &= \pi_1^{(1)} & & + .25\pi_4^{(0)} &&= .03125 + .25 \times .125 = .06250,\\
\pi_3^{(1)} &= & \pi_2^{(1)} & + .5\pi_4^{(0)} &&= .06250 + .5 \times .125 = .12500,\\
\pi_4^{(1)} &= & \pi_3^{(1)} & &&= .12500.
\end{aligned}$$

Notice that the sum of elements in $\pi^{(1)}$ does not add up to 1, and so it becomes necessary to normalize. We have

$$\|\pi^{(1)}\|_1 = 0.34375,$$

so dividing each element by 0.34375, we obtain

$$\pi^{(1)} = (0.090909,\ 0.181818,\ 0.363636,\ .363636) = \frac{1}{11}(1,\ 2,\ 4,\ 4).$$

By continuing this procedure, the following sequence of approximations is computed:

$$\pi^{(1)} = (0.090909,\ 0.181818,\ 0.363636,\ .363636),$$
$$\pi^{(2)} = (0.090909,\ 0.181818,\ 0.363636,\ .363636),$$
$$\pi^{(3)} = (0.090909,\ 0.181818,\ 0.363636,\ .363636),$$

and so on. For this particular example, Gauss–Seidel converges in only one iteration! To see why this is so, we need to examine the iteration matrix

$$H_{GS} = (D - L)^{-1}U. \tag{10.20}$$

In the example considered above, we have

$$D = \begin{pmatrix} -.5 & 0 & 0 & 0 \\ 0 & -.5 & 0 & 0 \\ 0 & 0 & -.5 & 0 \\ 0 & 0 & 0 & -.5 \end{pmatrix}, \quad L = \begin{pmatrix} 0 & 0 & 0 & 0 \\ -.5 & 0 & 0 & 0 \\ 0 & -.5 & 0 & 0 \\ 0 & 0 & -.5 & 0 \end{pmatrix}, \quad U = \begin{pmatrix} 0 & 0 & 0 & -.125 \\ 0 & 0 & 0 & -.125 \\ 0 & 0 & 0 & -.25 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

and

$$H_{GS} = (D - L)^{-1}U = \begin{pmatrix} 0 & 0 & 0 & .25 \\ 0 & 0 & 0 & .50 \\ 0 & 0 & 0 & 1.0 \\ 0 & 0 & 0 & 1.0 \end{pmatrix}.$$

Since U has nonzeros only in the last column, it must be the case that H_{GS} has nonzeros only in the last column and hence the only nonzero eigenvalue of H_{GS} must be the last diagonal element. More generally, the eigenvalues of upper or lower triangular matrices are equal to the diagonal elements of the matrix. Since the rate of convergence of the power method when applied to Markov chain problems depends on the magnitude of the subdominant eigenvalue (here equal to 0), convergence must occur immediately after the first iteration which is indeed what we observe in this example.

As indicated in Equation (10.13), the method of Gauss–Seidel corresponds to computing the i^{th} component of the current approximation from $i = 1$ through n, i.e., from top to bottom. To denote specificially the direction of solution, this is sometimes referred to as *forward* Gauss–Seidel. A *backward* Gauss–Seidel iteration takes the form

$$(D - U)x^{(k+1)} = Lx^{(k)}, \quad k = 0, 1, \ldots,$$

and corresponds to computing the components from bottom to top. Forward and backward iterations in a Jacobi setting are meaningless, since in Jacobi only components of the previous iteration are used in the updating procedure. As a general rule of thumb, a forward iterative method is usually recommended when the preponderance of the elemental mass is to be found below the diagonal, for in this case the iterative method essentially works with the inverse of the lower triangular portion of the matrix, $(D - L)^{-1}$, and, intuitively, the closer this is to the inverse of the entire matrix, the faster the convergence. Ideally, in a general context, a splitting should be such that M is chosen as close to Q^T as possible, subject only to the constraint that M^{-1} be easy to find. On the other hand, a backward iterative scheme works with the inverse of the upper triangular portion, $(D - U)^{-1}$, and is generally recommended when most of the nonzero mass lies above the diagonal. However, some examples that run counter to this "intuition" are known to exist.

10.3.3 The Method of Successive Overrelaxation

In many ways, the *successive overrelaxation method* (SOR) resembles the Gauss–Seidel method. When applied to $Ax = b$, a linear system of n equations in n unknowns, the i^{th} component of the

$(k+1)^{\text{th}}$ iteration is obtained from

$$a_{ii}x_i^{(k+1)} = a_{ii}(1-\omega)x_i^{(k)} + \omega\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right), \quad i = 1, 2, \ldots, n.$$

Observe that the expression within the large parentheses on the right-hand side completely constitutes the method of Gauss–Seidel as defined by Equation (10.16) and that SOR reduces to Gauss–Seidel when ω is set equal to 1. A *backward* SOR relaxation may also be written. For $\omega > 1$, the process is said to be one of *overrelaxation*; for $\omega < 1$ it is said to be *underrelaxation*. It may be shown that the SOR method converges only if $0 < \omega < 2$. This is a necessary, but not sufficient, condition for convergence.

The choice of an optimal, or even a reasonable, value for ω has been the subject of much study, especially for problems arising in the numerical solution of partial differential equations. Some results have been obtained for certain classes of matrices but unfortunately, little is known at present about the optimal choice of ω for arbitrary nonsymmetric linear systems. If a series of related experiments is to be conducted, it may well be worthwhile to carry out some numerical experiments to try to determine a suitable value; some sort of adaptive procedure might be incorporated into the algorithm. For example, it is possible to begin iterating with a value of $\omega = 1$ and, after some iterations have been carried out, to estimate the rate of convergence from the computed approximations. The value of ω may now be augmented to 1.1, say, and after some further iterations a new estimate of the rate of convergence computed. If this is better than before, ω should again be augmented, to 1.2, say, and the same procedure used. If the rate of convergence is not as good, the value of ω should be diminished.

When applied to the homogeneous system $Q^T x = (D-L-U)x = 0$, the SOR method becomes

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \omega\left\{\frac{1}{d_{ii}}\left(\sum_{j=1}^{i-1} l_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{n} u_{ij}x_j^{(k)}\right)\right\}, \quad i = 1, 2, \ldots, n,$$

or in matrix form

$$x^{(k+1)} = (1-\omega)x^{(k)} + \omega\left\{D^{-1}\left(Lx^{(k+1)} + Ux^{(k)}\right)\right\}. \tag{10.21}$$

Rearranging, we find

$$(D-\omega L)x^{(k+1)} = [(1-\omega)D + \omega U]x^{(k)}$$

or

$$x^{(k+1)} = (D-\omega L)^{-1}[(1-\omega)D + \omega U]x^{(k)}, \tag{10.22}$$

and thus the iteration matrix for the SOR method is

$$H_\omega = (D-\omega L)^{-1}[(1-\omega)D + \omega U].$$

It corresponds to the splitting $M = \omega^{-1}[D-\omega L]$ and $N = \omega^{-1}[(1-\omega)D + \omega U]$.

From Equation (10.22), it is evident that the stationary probability vector is the eigenvector corresponding to a unit eigenvalue of the SOR iteration matrix. However, with the SOR method, it is not necessarily true that this unit eigenvalue is the dominant eigenvalue, because the eigenvalues depend on the choice of the relaxation parameter ω. It is possible that H_ω has eigenvalues which are strictly greater than 1. When the unit eigenvalue is the dominant eigenvalue, then the SOR method is identical to the power method applied to H_ω. The value of ω that maximizes the difference between this unit eigenvalue and the subdominant eigenvalue of H_ω is the optimal choice for the relaxation parameter, and the convergence rate achieved with this value of ω can be a considerable improvement over that of Gauss–Seidel.

To summarize to this point, we have now seen that the power method may be used to obtain π from one of four sources: P^T, H_J, H_{GS}, and H_ω. The eigenvalues (with the exception of the unit eigenvalue) will not be the same from one matrix to the next, and sometimes a considerable difference in the number of iterations required to obtain convergence may be observed. Since the computational effort to perform an iteration step is the same in all four cases, it is desirable to apply the power method to the matrix that yields convergence in the smallest number of iterations, i.e., to the matrix whose subdominant eigenvalues are, in modulus, furthest from unity.

The following Matlab code performs a fixed number of iterations of the basic Jacobi, forward Gauss–Seidel, or SOR method on the system of equation $Ax = b$. It accepts an input matrix A (which may be set equal to an infinitesimal generator Q^T), an initial approximation x_0, a right-hand side vector b (which may be set to zero—in which case, a normalization must also be performed), and the number of iterations to be carried out, *itmax*. It returns the computed solution, *soln*, and a vector containing the residual computed at each iteration, *resid*. It is not designed to be a "production" code, but rather simply a way to generate some results in order to get some intuition into the performance characteristics of these methods.

Matlab code for Jacobi/Gauss–Seidel/SOR

```
function [soln,resid] = gs(A,x0,b,itmax)
%  Performs ``itmax'' iterations of Jacobi/Gauss-Seidel/SOR on Ax = b

  [n,n] = size(A); L = zeros(n,n); U = L;  D = diag(diag(A));
  for i = 1:n,
      for j = 1:n,
           if i<j, U(i,j) = -A(i,j); end
           if i>j, L(i,j) = -A(i,j); end
      end
  end
  M = inv(D-L);  B = M*U;        %  B is GS iteration matrix
 %M = inv(D);    B = M*(L+U); % Use this for Jacobi
 %w = 1.1; b = w*b; M = inv(D-w*L); B = M*((1-w)*D + w*U) % Use this for SOR

  for iter = 1:itmax,
      soln = B*x0+M*b;
      if norm(b,2) == 0 soln = soln/norm(soln,1); end % Normalize when b=0.
      resid(iter) = norm(A*soln-b,2);
      x0 = soln;
  end
  resid = resid';
  if norm(b,2) == 0 soln = soln/norm(soln,1); end  % Normalize when b = 0.
```

10.3.4 Data Structures for Large Sparse Matrices

We focus next on some algorithmic details that must be taken into account when implementing iterative methods for solving large-scale Markov chains. When the transition matrix is larger than several hundred it becomes impractical to keep it in a two-dimensional array, the format in which we are used to seeing matrices. In most cases the transition matrix is sparse, since each state generally can reach only a small number of states in a single step, and hence there are only a few nonzero elements in each row. In this section we shall consider approaches for storing this matrix efficiently, by taking advantage of its sparsity. One of the major advantages that iterative methods have over direct methods is that no modification of the elements of the transition matrix occurs during the execution of the algorithm. Thus, the matrix may be stored once and for all in some convenient compact form without the need to provide mechanisms to handle insertions (due to zero elements becoming nonzero) and deletions (due to the elimination of nonzero elements). As a constraint, the storage scheme used should not hinder the numerical operations that must be conducted on the matrix. The basic numerical operation performed by the iterative methods we consider, in fact, the only numerical operation performed on the matrix, is its pre- and postmultiplication by a vector, i.e., $z = Ax$ and $z = A^T x$.

One simple approach is to use a real (double-precision) one-dimensional array aa to store the nonzero elements of the matrix and two integer arrays ia and ja to indicate, respectively, the row and column positions of these elements. Thus, if the nonzero element a_{ij} is stored in position k of aa, i.e., $aa(k) = a_{ij}$, we have $ia(k) = i$ and $ja(k) = j$.

Example 10.15 The (4×4) matrix A given by

$$A = \begin{pmatrix} -2.1 & 0.0 & 1.7 & 0.4 \\ 0.8 & -0.8 & 0.0 & 0.0 \\ 0.2 & 1.5 & -1.7 & 0.0 \\ 0.0 & 0.3 & 0.2 & -0.5 \end{pmatrix}$$

may be stored as

aa :	−2.1	1.7	0.4	−0.8	0.8	−1.7	0.2	1.5	−0.5	0.3	0.2
ia :	1	1	1	2	2	3	3	3	4	4	4
ja :	1	3	4	2	1	3	1	2	4	2	3

or as

aa :	−2.1	0.8	0.2	−0.8	1.5	0.3	1.7	−1.7	0.2	0.4	−0.5
ia :	1	2	3	2	3	4	1	3	4	1	4
ja :	1	1	1	2	2	2	3	3	3	4	4

or yet again as

aa :	−2.1	−0.8	−1.7	−0.5	1.7	0.8	0.4	0.2	0.3	1.5	0.2
ia :	1	2	3	4	1	2	1	4	4	3	3
ja :	1	2	3	4	3	1	4	3	2	2	1

and so on. In the first case, the matrix is stored by rows, by columns in the second case, and in a random fashion in the third (although diagonal elements are given first).

Irrespective of the order in which the elements of the matrix A are entered into aa, the following algorithm, in which n_z denotes the number of nonzero elements in A, computes the product $z = Ax$.

ALGORITHM 10.4: SPARSE MATRIX-VECTOR MULTIPLICATION I

```
1. Set $z(i) = 0$ for all $i$.
2. For $next = 1$ to $n_z$ do
   • Set $nrow = ia(next)$.
   • Set $ncol = ja(next)$.
   • Compute $z(nrow) = z(nrow) + aa(next) \times x(ncol)$.
```

To perform the product $z = A^T x$ it suffices simply to interchange the arrays ia and ja. This algorithm is based on the fact that when multiplying a matrix by a vector, each element of the matrix is used only once: element a_{ij} is multiplied with x_j and constitutes one term of the inner product $\sum_{j=1}^n a_{ij} x_j = z_i$, the i^{th} component of the result z. It follows that the elements in the array aa can be treated consecutively from first to last, at the end of which the matrix-vector product will have been formed.

A more efficient storage scheme can be implemented if a partial ordering is imposed on the positions of the nonzero elements in the array aa. Consider the case when the nonzero elements of the matrix are stored by rows; elements of row i precede those of row $i+1$ but elements within a row may or may not be in order. This is frequently the case with Markov chains, since it is usual to generate all the states that may be reached in a single step from a given state i before generating the states that can be reached from the next state, $i+1$. Hence the matrix is generated row by row. When the nonzero elements are stored by rows in this fashion, it is possible to dispense with the integer array *ia* and to replace it with a smaller array. The most commonly used compact storage scheme uses the elements of *ia* as pointers into the arrays aa and *ja*. The k^{th} element of *ia* denotes the position in aa and *ja* at which the first element of row k is stored. Thus, we always have $ia(1) = 1$. Additionally, it is usual to store the first empty position of aa and *ja* in position $(n+1)$ of *ia*. Most often this means that $ia(n+1) = n_z + 1$. The number of nonzero elements in row i is then given by $ia(i+1) - ia(i)$. This makes it easy to immediately go to any row of the matrix, even though it is stored in a compact form—the $ia(i+1) - ia(i)$ nonzero elements of row i begin at $aa[ia(i)]$. This *row-wise* packing scheme is sometimes referred to as the *Harwell-Boeing format*.

Example 10.16 Consider, once again, the same 4×4 matrix:

$$A = \begin{pmatrix} -2.1 & 0.0 & 1.7 & 0.4 \\ 0.8 & -0.8 & 0.0 & 0.0 \\ 0.2 & 1.5 & -1.7 & 0.0 \\ 0.0 & 0.3 & 0.2 & -0.5 \end{pmatrix}.$$

In this row-wise packing scheme, A may be stored as

aa :	−2.1	1.7	0.4	−0.8	0.8	−1.7	0.2	1.5	−0.5	0.3	0.2
ja :	1	3	4	2	1	3	1	2	4	2	3
ia :	1	4	6	9	12						

It is not necessary for the elements in any row to be in order; it suffices that all the nonzero elements of row i come before those of row $i+1$ and after those of row $i-1$. Using this storage scheme, the matrix-vector product $z = Ax$ may be computed by

Algorithm 10.5: Sparse Matrix-Vector Multiplication II

```
1. For $i = 1$ to $n$ do
   • Set $sum = 0$.
   • Set $initial = ia(i)$.
   • Set $last = ia(i+1) - 1$.
   • For $j = initial$ to $last$ do
       ○ Compute $sum = sum + aa(j) \times x(ja(j))$.
   • Set $z(i) = sum$.
```

In our discussion of the SOR algorithm, it may have appeared to have been numerically more complex than the simple power method or Gauss–Seidel and that incorporation of a sparse storage data structure would be more challenging. However, this is not the case. The SOR method requires only a matrix-vector multiplication per iteration, the same as the power method, or for that matter the Jacobi or Gauss–Seidel method. When programming SOR, we use the formula

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right).$$

By scaling the matrix so that $a_{ii} = 1$ for all i and setting $b_i = 0$, for all i, this reduces to

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} - \omega\left(\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right)$$

$$= x_i^{(k)} - \omega\left(\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} + a_{ii}x_i^{(k)} + \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right).$$

At iteration k, the program may be written (assuming A is stored in the row-wise compact form just described) simply as

Algorithm 10.6: Sparse SOR

```
1. For $i = 1$ to $n$ do
   • Set $sum = 0$.
   • Set $initial = ia(i)$.
   • Compute $last = ia(i+1) - 1$.
   • For $j = initial$ to $last$ do
      ◦ Compute $sum = sum + aa(j) \times x(ja(j))$.
   • Compute $x(i) = x(i) - \omega \times sum$.
```

Observe that the only difference between this and the straightforward matrix-vector multiply algorithm, Algorithm 10.5, occurs in the very last line. In the SOR algorithm, the elements of x are determined sequentially and hence it is possible to overwrite them with their new values as soon as they have been computed. The computation of element $x(i+1)$ does not begin until the new value of $x(i)$ has already been computed.

When using the SOR algorithm to obtain the stationary distribution of a Markov chain, the matrix A above must be replaced by Q^T, the transpose of the infinitesimal generator. This may pose a problem if Q is generated by rows, as is often the case. If it is not possible to generate Q by columns (which means that for each state we need to find the states that can access this state in one step), then it becomes necessary to transpose the matrix and to do so without expanding it into full two-dimensional format. If sufficient storage is available to store the compacted matrix *and* its compacted transpose, then the operation of transposition can be effected in $O(n_z)$ operations, where n_z is the number of nonzero elements stored. If space is not available for a second compacted copy, then the transposition may be carried out in place in $O(n_z \log n_z)$ operations, using a standard sorting procedure. The moral is obviously to try to store the matrix Q by columns. Unfortunately, in many Markov chain applications, it is much more convenient to determine all destination states that occur from a given source state (row-wise generation) than to determine all source states which lead to a given destination state (column-wise generation).

10.3.5 Initial Approximations, Normalization, and Convergence

During the iterative process, a suitably chosen initial approximation is successively modified until it converges to the solution. This poses three problems:

- What should we choose as the initial vector?
- What happens to this vector at each iteration?
- How do we know when convergence has occurred?

These three questions are answered in this section. When choosing an initial starting vector for an iterative method, it is tempting to choose something simple, such as a vector whose components are all zero except for one or two entries. If such a choice is made, care must be taken to ensure that the initial vector is not deficient in some component of the basis of the solution vector, otherwise a vector containing all zeros may arise, and the process will never converge.

Example 10.17 Consider the (2×2) transition rate matrix

$$Q^T = \begin{pmatrix} -\lambda & \mu \\ \lambda & -\mu \end{pmatrix} = (D - L - U).$$

Applying the Gauss–Seidel method with $x^{(0)} = (1, 0)^T$ yields $x^{(k)} = (0,\ 0)^T$ for all $k \geq 1$, since

$$x^{(1)} = (D-L)^{-1}Ux^{(0)} = (D-L)^{-1}\begin{pmatrix} 0 & -\mu \\ 0 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = (D-L)^{-1}\begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

and all successive vectors $x^{(k)},\ k = 1, 2, \ldots,$ are identically equal to zero.

If some approximation to the solution is known, then it should be used. If this is not possible, the elements of the initial iterate may be assigned random numbers uniformly distributed between 0 and 1 and then normalized to produce a probability vector. A simpler approach is to set each element equal to $1/n$, where n is the number of states.

We now turn to the second question. At each iteration the current approximation must be multiplied by the iteration matrix. In methods other than the power method, the result of this multiplication may not be a probability vector, i.e., the sum of the components of the computed vector may not be one, even when the initial approximation is chosen to be a probability vector. This vector, however, may be made into a probability vector by normalization, by dividing each component by the sum of the components. In large-scale Markov chains, this can be a costly operation, since the sum of all n components must be computed and then followed by n divisions—indeed it can almost double the complexity per iteration. In most cases, it is unnecessary to constrain successive approximations to be probability vectors. Normalization is only required prior to conducting a test for convergence, since we need to compare standardized vectors, and to prevent problems of overflow and underflow of the elements of the successive approximations.

Underflow. While this is not fatal, it often gives rise to an undesirable error message. This may be avoided by periodically checking the magnitude of the elements and by setting those that are less than a certain threshold (e.g., 10^{-25}) to zero. Note that the underflow problem can arise even when the approximations are normalized at each iteration, since some elements of the solution vector, although strictly positive, may be extremely small. The concern when normalization is not performed is that *all* of the elements will become smaller with each iteration until they are all set to zero. This problem can be avoided completely by a periodic scan of the approximation to ensure that at least one element exceeds a certain minimum threshold and to initiate a normalization when this test fails.

Overflow. With a reasonable starting vector, overflow is unlikely to occur, since the eigenvalues of the iteration matrices should not exceed one. All doubt can be eliminated by keeping a check on the magnitude of the largest element and normalizing the iterate if this element exceeds a certain maximum threshold, say 10^{10}.

The final question to be answered concerns knowing when convergence has occurred. This generally comes under the heading of *convergence testing and numerical accuracy*. The number of iterations k needed to satisfy a tolerance criterion ϵ may be obtained approximately from the relationship

$$\rho^k = \epsilon, \quad \text{i.e., } k = \frac{\log \epsilon}{\log \rho},$$

where ρ is the spectral radius of the iteration matrix. In Markov chain problems, the magnitude of the subdominant eigenvalue is used in place of ρ. Thus, when we wish to have six decimal places of accuracy, we set $\epsilon = 10^{-6}$ and find that the number of iterations needed for different spectral radii is as follows:

ρ	.1	.5	.6	.7	.8	.9	.95	.99	.995	.999
k	6	20	27	39	62	131	269	1, 375	2,756	13,809

Since the size of the subdominant eigenvalue is seldom known in advance, the usual method of testing for convergence is to examine some norm of the difference of successive iterates. When this difference becomes less than a certain prespecified tolerance, the iterative procedure is stopped. This is a satisfactory approach when the procedure converges relatively rapidly and the magnitude of the largest element of the vector is of order unity. However, when the procedure is converging slowly, it is possible that the difference in successive iterates is smaller than the tolerance specified, even though the vector may be far from the solution.

Example 10.18 Consider the infinitesimal generator

$$Q = \begin{pmatrix} -.6 & 0 & .6 & 0 \\ .0002 & -.7 & 0 & .6998 \\ .1999 & .0001 & -.2 & 0 \\ 0 & .5 & 0 & -.5 \end{pmatrix}.$$

Using the Gauss–Seidel method with initial approximation $\pi^{(0)} = (.25, .25, .25, .25)$, we find, after 199 and 200 iterations,

$$\pi^{(199)} = (0.112758,\ 0.228774,\ 0.338275,\ 0.3201922),$$
$$\pi^{(200)} = (0.112774,\ 0.228748,\ 0.338322,\ 0.3201560),$$

which appears to give about four decimal places of accuracy. If the tolerance criterion had been set to $\epsilon = 0.001$, the iterative procedure would have stopped prior to iteration 200. However, the true solution is

$$\pi = (0.131589,\ 0.197384,\ 0.394768,\ 0.276259)$$

and the Gauss–Seidel method will eventually converge onto this solution. In this example, the subdominant eigenvalue of the Gauss–Seidel iteration matrix H_{GS} is 0.9992 and over 6,000 iterations are needed to achieve an accuracy of just $\epsilon = .01$.

This problem may be overcome by testing, not successive iterates $\|\pi^{(k)} - \pi^{(k-1)}\| < \epsilon$, but rather iterates spaced further apart, e.g.,

$$\|\pi^{(k)} - \pi^{(k-m)}\| < \epsilon.$$

Ideally, m should be determined as a function of the convergence rate. A simple but less desirable alternative is to allow m to assume different values during the iteration procedure. For example, when the iteration number k is

$$\begin{array}{cl} k < 100 & \text{let} m = 5, \\ 100 \le k < 500 & \text{let } m = 10, \\ 500 \le k < 1{,}000 & \text{let } m = 20, \\ k \ge 1,000 & \text{let } m = 50. \end{array}$$

A second problem arises when the approximations converge to a vector in which the elements are all small. Suppose the problem has 100,000 states, and all are approximately equally probable. Then each element of the solution is approximately equal to 10^{-5}. If the tolerance criterion is set at 10^{-3} and the initial vector is chosen such that $\pi_i^{(0)} = 1/n$ for all i, then the process will probably "converge" after one iteration! This same problem can arise in a more subtle context if the approximations are not normalized before convergence testing *and* the choice of initial vector results in the iterative method converging to a vector in which all components are small. This may happen even though some elements of the solution may be large relative to others. For example, it happens in the (2×2) Markov chain of Example 10.17 when the initial approximation is chosen as $(\xi,\ 10^{-6})$ for any value of ξ. (Note that the opposite effect will occur if the approximations converge to a vector with all elements relatively large, e.g., if the initial vector is $(\xi,\ 10^{6})$ in Example 10.17.)

A solution to this latter aspect of the problem (when vectors are not normalized) is, of course, to normalize the iterates before testing for convergence. If this normalization is such that it produces a probability vector, the original problem (all of the components may be small) still remains. A better choice of normalization in this instance is $\|\pi^{(k)}\|_\infty = 1$ (i.e., normalize so that the largest element of $\pi^{(k)}$ is equal to 1—only the final normalization needs to produce a probability vector). A better solution, however, and the one that is recommended, is to use a relative measure, e.g.,

$$\max_i \left(\frac{|\pi_i^{(k)} - \pi_i^{(k-m)}|}{|\pi_i^{(k)}|} \right) < \epsilon.$$

This effectively removes the exponent from consideration in the convergence test and hence gives a better estimate of the precision that has been achieved.

Another criterion that has been used for convergence testing is to check the size of the residuals (the magnitude of $\|\pi^{(k)} Q\|$), which should be small. Residuals will work fine in many and perhaps even most modeling problems. Unfortunately, a small residual does not always imply that the error in the solution vector is also small. In ill-conditioned systems the residual may be very small indeed, yet the computed solution may be hopelessly inaccurate. A small residual is a necessary condition for the error in the solution vector to be small—but it is not a sufficient condition. The most suitable approach is to check the residual after the relative convergence test indicates that convergence has been achieved. In fact, it is best to envisage a battery of convergence tests, all of which must be satisfied before the approximation is accepted as being sufficiently accurate.

We now turn to the frequency with which the convergence test should be administered. Often it is performed during each iteration. This may be wasteful, especially when the matrix is very large and the iterative method is converging slowly. Sometimes it is possible to estimate the rate of convergence and to determine from this rate the approximate numbers of iterations that must be performed. It is now possible to proceed "full steam ahead" and carry out this number of iterations without testing for convergence or normalizing. For rapidly converging problems this may result in more iterations being performed than is strictly necessary, thereby achieving a more accurate result than the user actually needs. When the matrix is large and an iteration costly, this may be undesirable. One possibility is to carry out only a proportion (say 80–90%) of the estimated number of iterations before beginning to implement the battery of convergence tests.

An alternative approach is to implement a relatively inexpensive convergence test (e.g., using the relative difference in the first nonzero component of successive approximations) at each iteration. When this simple test is satisfied, more rigorous convergence tests may be initiated.

10.4 Block Iterative Methods

The iterative methods we have examined so far are sometimes referred to as *point* iterative methods in order to distinguish them from their *block* counterparts. Block iterative methods are generalizations of point iterative methods and can be particularly beneficial in Markov chain problems in which the state space can be meaningfully partitioned into subsets. In general such block iterative methods require more computation per iteration, but this is offset by a faster rate of convergence. Let us partition the defining homogeneous system of equations $\pi Q = 0$ as

$$(\pi_1,\ \pi_2,\ \ldots,\ \pi_N)\begin{pmatrix} Q_{11} & Q_{12} & \cdots & Q_{1N} \\ Q_{21} & Q_{22} & \cdots & Q_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{N1} & Q_{N2} & \cdots & Q_{NN} \end{pmatrix} = 0.$$

We now introduce the block splitting

$$Q^T = D_N - (L_N + U_N),$$

where D_N is a block diagonal matrix and L_N and U_N are, respectively, strictly lower and upper triangular block matrices. We have

$$D_N = \begin{pmatrix} D_{11} & 0 & \cdots & 0 \\ 0 & D_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D_{NN} \end{pmatrix},$$

$$L_N = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ L_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{N1} & L_{N2} & \cdots & 0 \end{pmatrix}, \quad U_N = \begin{pmatrix} 0 & U_{12} & \cdots & U_{1N} \\ 0 & 0 & \cdots & U_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}.$$

In analogy with Equation (10.18), the block Gauss–Seidel method is given by

$$(D_N - L_N)x^{(k+1)} = U_N x^{(k)}$$

and corresponds to the splitting $M = (D_N - L_N)$; $N = U_N$. The i^{th} block equation is given by

$$D_{ii}x_i^{(k+1)} = \left(\sum_{j=1}^{i-1} L_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{N} U_{ij}x_j^{(k)}\right) \qquad i = 1, 2, \ldots, N, \tag{10.23}$$

where the subvectors x_i are partitioned conformally with D_{ii}, $i = 1, 2, \ldots, N$. This implies that at each iteration we must now solve N systems of linear equations

$$D_{ii}x_i^{(k+1)} = z_i, \quad i = 1, 2, \ldots, N, \tag{10.24}$$

where

$$z_i = \left(\sum_{j=1}^{i-1} L_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{N} U_{ij}x_j^{(k)}\right), \quad i = 1, 2, \ldots, N.$$

The right-hand side z_i can always be computed before the i^{th} system has to be solved. If our Markov chain is irreducible, the N systems of equations (10.24) are nonhomogeneous and have nonsingular coefficient matrices. We may use either direct or iterative methods to solve them. Naturally, there is no requirement to use the same method to solve all the diagonal blocks. Instead, it is possible to tailor methods to particular block structures. If a direct method is used, then an LU decomposition of block D_{ii} may be formed once and for all before beginning the iteration, so that solving $D_{ii}x_i^{(k+1)} = z_i$, $i = 1, \ldots, N$, in each global iteration simplifies to a forward and backward substitution. The nonzero structure of the blocks may be such that this is a particularly efficient approach. For example, if the diagonal blocks are themselves diagonal matrices, or if they are upper or lower triangular matrices or even tridiagonal matrices, then it is very easy to obtain their LU decomposition, and a block iterative method becomes very attractive.

If the diagonal blocks are large and do not possess a suitable nonzero structure, it may be appropriate to use matrix iterative methods (such as point Gauss–Seidel) to solve these block equations—in which case we can have multiple inner (or local) iterative methods (one per block thus analyzed) within an outer (or global) iteration. A number of tricks may be used to speed up this process. First, the solution computed using any block D_{ii} at iteration k should be used as the initial approximation to the solution using this same block at iteration $k+1$. Second, it is hardly worthwhile computing a highly accurate solution in early (outer) iterations. We should require only a small number of digits of accuracy until the global process begins to converge. One convenient way to achieve this is to carry out only a fixed, small number of iterations for each inner solution. Initially, this will not give much accuracy, but when combined with the first suggestion, the accuracy achieved will increase from one outer iteration to the next.

Intuitively, it is expected that for a given transition rate matrix Q, the larger the block sizes (and thus the smaller the number of blocks), the fewer the number of (outer) iterations needed for convergence. This has been shown to be true under fairly general assumptions on the coefficient matrix for general systems of equations (see [53]). In the special case of only one block, the method degenerates to a standard direct method and we compute the solution in a single "iteration." The reduction in the number of iterations that usually accompanies larger blocks is offset to a certain degree by an increase in the number of operations that must be performed at each iteration. However, in some important cases it may be shown that there is no increase. For example, when the matrix is block tridiagonal (as in quasi-birth-death processes) and the diagonal blocks are also tridiagonal, it may be shown that the computational effort per iteration is the same for both point and block iterative methods. In this case the reduction in the number of iterations makes the block methods very efficient indeed.

In a similar vein to the block Gauss–Seidel method, we may also define a *block* Jacobi method

$$D_{ii}x_i^{(k+1)} = \left(\sum_{j=1}^{i-1} L_{ij}x_j^{(k)} + \sum_{j=i+1}^{N} U_{ij}x_j^{(k)}\right), \quad i = 1, 2, \ldots, N,$$

and a *block* SOR method

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \omega\left\{D_{ii}^{-1}\left(\sum_{j=1}^{i-1} L_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{N} U_{ij}x_j^{(k)}\right)\right\}, \quad i = 1, 2, \ldots, N.$$

Example 10.19 We apply the block Gauss–Seidel method to find the stationary distribution of the continuous-time Markov chain with infinitesimal generator given by

$$Q = \begin{pmatrix} -4.0 & 2.0 & 1.0 & 0.5 & 0.5 \\ 0.0 & -3.0 & 3.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -1.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 & -5.0 & 4.0 \\ 1.0 & 0.0 & 0.0 & 1.0 & -2.0 \end{pmatrix}.$$

We put the first three states in the first subset and the remaining two states into a second subsest. Transposing Q and writing out the system of equations, we have

$$Q^T x = \begin{pmatrix} D_{11} & -U_{12} \\ -L_{21} & D_{22} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \left(\begin{array}{ccc|cc} -4.0 & 0.0 & 0.0 & 1.0 & 1.0 \\ 2.0 & -3.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 3.0 & -1.0 & 0.0 & 0.0 \\ \hline 0.5 & 0.0 & 0.0 & -5.0 & 1.0 \\ 0.5 & 0.0 & 1.0 & 4.0 & -2.0 \end{array}\right)\begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \hline \pi_4 \\ \pi_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \hline 0 \\ 0 \end{pmatrix}.$$

Equation (10.23) becomes

$$D_{ii}x_i^{(k+1)} = \left(\sum_{j=1}^{i-1} L_{ij}x_j^{(k+1)} + \sum_{j=i+1}^{2} U_{ij}x_j^{(k)} \right), \quad i = 1, 2,$$

and leads to the two block equations

$$i = 1: \quad D_{11}x_1^{(k+1)} = \left(\sum_{j=1}^{0} L_{1j}x_j^{(k+1)} + \sum_{j=2}^{2} U_{1j}x_j^{(k)} \right) = U_{12}x_2^{(k)},$$

$$i = 2: \quad D_{22}x_2^{(k+1)} = \left(\sum_{j=1}^{1} L_{2j}x_j^{(k+1)} + \sum_{j=3}^{2} U_{2j}x_j^{(k)} \right) = L_{21}x_1^{(k+1)}.$$

Writing these block equations in full, they become

$$\begin{pmatrix} -4.0 & 0.0 & 0.0 \\ 2.0 & -3.0 & 0.0 \\ 1.0 & 3.0 & -1.0 \end{pmatrix} \begin{pmatrix} \pi_1^{(k+1)} \\ \pi_2^{(k+1)} \\ \pi_3^{(k+1)} \end{pmatrix} = - \begin{pmatrix} 1.0 & 1.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{pmatrix} \begin{pmatrix} \pi_4^{(k)} \\ \pi_5^{(k)} \end{pmatrix}$$

and

$$\begin{pmatrix} -5.0 & 1.0 \\ 4.0 & -2.0 \end{pmatrix} \begin{pmatrix} \pi_4^{(k+1)} \\ \pi_5^{(k+1)} \end{pmatrix} = - \begin{pmatrix} 0.5 & 0.0 & 0.0 \\ 0.5 & 0.0 & 1.0 \end{pmatrix} \begin{pmatrix} \pi_1^{(k+1)} \\ \pi_2^{(k+1)} \\ \pi_3^{(k+1)} \end{pmatrix}.$$

We shall solve these block equations using *LU* decompositions. Since D_{11} is lower triangular, the first subsystem may be solved by *forward substitution* alone:

$$D_{11} = \begin{pmatrix} -4.0 & 0.0 & 0.0 \\ 2.0 & -3.0 & 0.0 \\ 1.0 & 3.0 & -1.0 \end{pmatrix} = L \times I.$$

Forming an *LU* decomposition of the second subsystem, we have

$$D_{22} = \begin{pmatrix} -5.0 & 1.0 \\ 4.0 & -2.0 \end{pmatrix} = \begin{pmatrix} 1.0 & 0.0 \\ -0.8 & 1.0 \end{pmatrix} \begin{pmatrix} -5.0 & 1.0 \\ 0.0 & -1.2 \end{pmatrix} = L \times U.$$

Taking the initial distribution to be

$$\pi^{(0)} = (0.2,\ 0.2,\ 0.2,\ 0.2,\ 0.2)^T$$

and substituting the first block equation, we find

$$\begin{pmatrix} -4.0 & 0.0 & 0.0 \\ 2.0 & -3.0 & 0.0 \\ 1.0 & 3.0 & -1.0 \end{pmatrix} \begin{pmatrix} \pi_1^{(1)} \\ \pi_2^{(1)} \\ \pi_3^{(1)} \end{pmatrix} = - \begin{pmatrix} 1.0 & 1.0 \\ 0.0 & 0.0 \\ 0.0 & 0.0 \end{pmatrix} \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix} = - \begin{pmatrix} 0.4 \\ 0.0 \\ 0.0 \end{pmatrix}.$$

Forward substitution successively gives

$$\pi_1^{(1)} = 0.1000, \quad \pi_2^{(1)} = 0.0667, \quad \text{and} \quad \pi_3^{(1)} = 0.3000.$$

The second block system now becomes

$$\begin{pmatrix} 1.0 & 0.0 \\ -0.8 & 1.0 \end{pmatrix} \begin{pmatrix} -5.0 & 1.0 \\ 0.0 & -1.2 \end{pmatrix} \begin{pmatrix} \pi_4^{(1)} \\ \pi_5^{(1)} \end{pmatrix} = - \begin{pmatrix} 0.5 & 0.0 & 0.0 \\ 0.5 & 0.0 & 1.0 \end{pmatrix} \begin{pmatrix} 0.1000 \\ 0.0667 \\ 0.3000 \end{pmatrix} = - \begin{pmatrix} 0.0500 \\ 0.3500 \end{pmatrix}.$$

This is in the standard form $LUx = b$ from which the solution is computed by first obtaining z from $Lz = b$ and then x from $Ux = z$. From

$$\begin{pmatrix} 1.0 & 0.0 \\ -0.8 & 1.0 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = -\begin{pmatrix} 0.0500 \\ 0.3500 \end{pmatrix},$$

we obtain

$$z_1 = -0.0500 \quad \text{and} \quad z_2 = -0.3900.$$

Now, solving

$$\begin{pmatrix} -5.0 & 1.0 \\ 0.0 & -1.2 \end{pmatrix} \begin{pmatrix} \pi_4^{(1)} \\ \pi_5^{(1)} \end{pmatrix} = \begin{pmatrix} -0.0500 \\ -0.3900 \end{pmatrix},$$

we find

$$\pi_4^{(1)} = 0.0750 \quad \text{and} \quad \pi_5^{(1)} = 0.3250.$$

We now have

$$\pi^{(1)} = (0.1000,\ 0.0667,\ 0.3000,\ 0.0750,\ 0.3250)$$

which, when normalized so that the elements sum to 1, gives

$$\pi^{(1)} = (0.1154,\ 0.0769,\ 0.3462,\ 0.0865,\ 0.3750).$$

The next iteration may now be initiated. Interested readers who perform some additional iterations will see that, in this example, all subsequent iterations yield exactly the same result. Indeed, for this specific example, the block Gauss–Seidel method requires only a single iteration to obtain the solution to full machine precision. A examination of the matrix U_N reveals that this matrix has only two nonzero elements in positions $(1, 4)$ and $(1, 5)$ and they are both equal. It must follow that the iteration matrix can have only two nonzero columns, the fourth and fifth, and they must be identical. It may be concluded that the iteration matrix has an eigenvalue equal to 1 and four equal to 0, which explains the fact that convergence is achieved in a single iteration.

The following code is a Matlab implementation of the *block Gauss–Seidel method*. The program accepts a stochastic matrix P; a partitioning vector ni, whose i^{th} component stores the length of the i^{th} block; and two integers: *itmax1*, which denotes the number of outer iterations to perform, and *itmax2*, which denotes the number of iterations to use to solve each of the blocks *if* Gauss–Seidel is used to solve these blocks. The program returns the solution vector π and a vector of residuals. It calls the Gauss–Seidel program given previously.

Matlab code for Block Gauss–Seidel

```
function [x,res] = bgs(P,ni,itmax1,itmax2)
[n,n] = size(P); [na,nb] = size(ni);

%         BLOCK Gauss-Seidel FOR P^T x = x

bl(1) = 1;                                     %  Get beginning and end
for k = 1:nb, bl(k+1) = bl(k)+ni(k); end       %  points of each block
x = ones(n,1)/n;                               %  Initial approximation

%%%%%%%%%%%%%%%%%%%%%%%%  BEGIN OUTER LOOP  %%%%%%%%%%%%%%%%%%%%%%%%%
for iter = 1:itmax1,
    for m = 1:nb,                              % All diagonal blocks
```

```
          A = P(bl(m):bl(m+1)-1,bl(m):bl(m+1)-1)';  % Get A_mm
          b = -P(1:n,bl(m):bl(m+1)-1)'*x+A*x(bl(m):bl(m+1)-1);  %  RHS
          z = inv(A-eye(ni(m)))*b;                    %  Solve for z

%         ***  To solve the blocks using Gauss--Seidel     ***
%         ***  instead of a direct method, substitute     ***
%         ***  the next two lines for the previous one.   ***
%**       x0 = x(bl(m):bl(m+1)-1);                   % Get starting vector
%**       [z,r] = gs(A-eye(ni(m)),x0,b,itmax2);      % Solve for z

          x(bl(m):bl(m+1)-1) = z;                    %  Update x
      end
      res(iter) = norm((P'-eye(n))*x,2);             %  Compute residual
end
x = x/norm(x,1); res = res';
```

10.5 Decomposition and Aggregation Methods

A decompositional approach to solving Markov chains is intuitively very attractive since it appeals to the principle of divide and conquer: if the model is too large or complex to analyze in toto, it is divided into subsystems, each of which is analyzed separately, and a global solution is then constructed from the partial solutions. Ideally the problem is broken into subproblems that can be solved independently and the global solution is obtained by "pasting" together the subproblem solutions. Although it is rare to find Markov chains that can be divided into independent subchains, it is not unusual to have Markov chains in which this condition almost holds. An important class of problems that frequently arise in Markov modeling are those in which the state space may be partitioned into disjoint subsets with strong interactions among the states of a subset but with weak interactions among the subsets themselves. Such problems are sometimes referred to as *nearly completely decomposable (NCD), nearly uncoupled,* or *nearly separable.* It is apparent that the assumption that the subsystems are independent and can therefore be solved separately does not hold. Consequently an error arises. This error will be small if the assumption is approximately true. An irreducible NCD stochastic matrix P may be written as

$$P = \begin{pmatrix} P_{11} & P_{12} & \dots & P_{1N} \\ P_{21} & P_{22} & \dots & P_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N1} & P_{N2} & \dots & P_{NN} \end{pmatrix},$$

where

$$||P_{ii}|| = O(1), \qquad i = 1, 2, \dots, N,$$

and

$$||P_{ij}|| = O(\epsilon), \qquad i \neq j.$$

In addition to its unit eigenvalue, such a matrix possesss $N - 1$ eigenvalues extremely close to 1. None of the point iterative methods discussed previously is effective in handling this situation. On the other hand, block and decompositional methods can be very effective.

We begin by examining what happens when the off-diagonal blocks are all zero, i.e.,

$$(\pi_1, \pi_2, \ldots, \pi_N)\begin{pmatrix} P_{11} & 0 & \ldots & 0 & 0 \\ 0 & P_{22} & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & P_{N-1N-1} & 0 \\ 0 & 0 & \ldots & 0 & P_{NN} \end{pmatrix} = (\pi_1, \pi_2, \ldots, \pi_N).$$

In this case, each P_{ii} is a stochastic matrix. The Markov chain is reducible into N distinct irreducible classes and a stationary distribution can be found for each class. Each π_i can be found directly from

$$\pi_i P_{ii} = \pi_i, \quad i = 1, 2, \ldots, N.$$

This allows us to envisage the following approximate solution procedure when the Markov chain is NCD rather than completely decomposable:

(a) Solve the diagonal blocks as if they are independent.
The solution obtained for each block should provide an approximation to the probability of being in the different states of that block, conditioned on being in that block.
(b) Estimate the probability of being in each block.
This will allow us to remove the condition in part (a).
(c) Combine (a) and (b) into a global approximate solution.

We now examine each of these three steps in more detail.

Step (a): Solve blocks as if independent
The initial step in approximating the solution of $\pi P = \pi$ when $P_{ij} \neq 0$ is to assume that the system is completely decomposable and to compute the stationary probability distribution for each component. A first problem that arises is that the P_{ii} are not stochastic but rather strictly substochastic. A simple way around this problem is to simply ignore it; i.e., to work directly with the substochastic matrices P_{ii} themselves. In other words, we may use the normalized eigenvector corresponding to the Perron root (the eigenvalue closest to 1) of block P_{ii} as the probability vector whose elements denote the probabilities of being in the different states of the block, conditioned on being in this block.

Example 10.20 To help, we shall illustrate the procedure using the 8 × 8 Courtois matrix, an examination of which reveals that it is nearly completely decomposable (NCD) into a block of order 3, a second of order 2, and a third of order 3:

$$P = \left(\begin{array}{ccc|cc|ccc} .85 & .0 & .149 & .0009 & .0 & .00005 & .0 & .00005 \\ .1 & .65 & .249 & .0 & .0009 & .00005 & .0 & .00005 \\ .1 & .8 & .0996 & .0003 & .0 & .0 & .0001 & .0 \\ \hline .0 & .0004 & .0 & .7 & .2995 & .0 & .0001 & .0 \\ .0005 & .0 & .0004 & .399 & .6 & .0001 & .0 & .0 \\ \hline .0 & .00005 & .0 & .0 & .00005 & .6 & .2499 & .15 \\ .00003 & .0 & .00003 & .00004 & .0 & .1 & .8 & .0999 \\ .0 & .00005 & .0 & .0 & .00005 & .1999 & .25 & .55 \end{array}\right).$$

For this matrix, we have the following blocks, Perron roots, and corresponding left-hand eigenvectors:

$$P_{11} = \begin{pmatrix} .85 & .0 & .149 \\ .1 & .65 & .249 \\ .1 & .8 & .0996 \end{pmatrix}, \quad \lambda_{1_1} = .99911, \quad u_1 = (.40143,\ .41672,\ .18185),$$

$$P_{22} = \begin{pmatrix} .7 & .2995 \\ .399 & .6 \end{pmatrix}, \quad \lambda_{2_1} = .99929, \quad u_2 = (.57140,\ .42860),$$

$$P_{33} = \begin{pmatrix} .6 & .2499 & .15 \\ .1 & .8 & .0999 \\ .1999 & .25 & .55 \end{pmatrix}, \quad \lambda_{3_1} = .9999, \quad u_3 = (.24074,\ .55563,\ .20364).$$

To summarize, in part (a), the left-hand eigenvector u_i of length n_i corresponding to the eigenvalue closest to 1, λ_{i_1}, in each block $i,\ 1 \le i \le N$, is computed. In other words, we solve the N eigenvalue problems

$$u_i P_{ii} = \lambda_{i_1} u_i, \qquad u_i e = 1, \quad i = 1, 2, \ldots, N.$$

Step (b): Estimate block probabilities

The second problem is that, once we have computed the stationary probability vector for each block, simply concatenating them together will not give a probability vector. The elements of each subvector sum to 1. We still need to weight each subvector by the probability of being in its subblock of states. The probability distributions computed from the P_{ii} are conditional probabilities in the sense that they express the probability of being in a given state of subset $i,\ i = 1, 2, \ldots, N$, conditioned on the fact that the Markov chain is in one of the states of that subset. We need to remove that condition. To determine the probability of being in a given block of states we need to construct a matrix whose element ij gives the probability of a transition from block i to block j. This is an $(N \times N)$ stochastic matrix that characterizes the interactions among blocks. To construct this matrix, which is called the *coupling matrix*, we need to shrink each block P_{ij} of P down to a single element. The stationary distribution of the coupling matrix provides the block probabilities, the weights needed to form the global approximation.

In terms of the running example, we need to find weights $\xi_1,\ \xi_2$, and ξ_3 such that

$$(\xi_1 u_1,\ \xi_2 u_2,\ \xi_3 u_3)$$

is an approximate solution to π. Here ξ_i is the proportion of time we spend in block i. We need to shrink our original (8×8) stochastic matrix down to a (3×3) stochastic matrix. This is accomplished by first replacing each row of each block by the sum of its elements. The sum of the elements of row k of block ij gives the probability of leaving state k of block i and entering into (one of the states of) block j. It no longer matters which particular state of block j is this destination state. Mathematically, the operation performed for each block is $P_{ij}e$.

Example 10.21 Summing across the block rows of the Courtois matrix gives

$$\left(\begin{array}{l|l|l} .999 & .0009 & .0001 \\ .999 & .0009 & .0001 \\ .9996 & .0003 & .0001 \\ \hline .0004 & .9995 & .0001 \\ .0009 & .999 & .0001 \\ \hline .00005 & .00005 & .9999 \\ .00006 & .00004 & .9999 \\ .00005 & .00005 & .9999 \end{array}\right).$$

The next step is to use these results to find the probability of leaving (any state of) block i to enter (any state of) block j. This means that we must reduce each column subvector, $P_{ij}e$, to a scalar. As we have just noted, the k^{th} element of $P_{ij}e$ is the probability of leaving state k of block i and entering into block j. To determine the total probability of leaving (any state of) block i to enter into (any state of) block j we need to sum the elements of this vector after each element has been weighed by the probability of being in that state (given that the Markov chain is in one of the states of that block). These weighing factors may be obtained from the elements of the stationary probability vector. They are the components of $\pi_i/||\pi_i||_1$. The ij^{th} element of the reduced $(N \times N)$ matrix is therefore given by

$$(C)_{ij} = \frac{\pi_i}{||\pi_i||_1} P_{ij} e = \phi_i P_{ij} e,$$

where $\phi_i = \pi_i/||\pi_i||_1$. If P is an irreducible stochastic matrix, then C also is irreducible and stochastic. Let ξ denote its left eigenvector, i.e., $\xi C = \xi$ and $\xi e = 1$. The i^{th} component of ξ is the stationary probability of being in (one of the states of) block i. It is easy to show that

$$\xi = (||\pi_1||_1,\ ||\pi_2||_1,\ \ldots,\ ||\pi_N||_1).$$

Of course, the vector π is not yet known, so that it is not possible to compute the weights $||\pi_i||_1$. However, they may be approximated by using the probability vector computed from each of the individual P_{ii}, i.e., by setting $\phi_i = u_i$. Consequently, the weights ξ_i can be estimated and an approximate solution to the stationary probability vector π obtained. Any of the previously used methods to find stationary probability vectors may be used.

Example 10.22 Performing these operations, we obtain the following coupling matrix for the Courtois example:

$$(.40143, .41672, .18185,\ |\ .57140, .42860,\ |\ .24074, .55563, .20364)\left(\begin{array}{l|l|l} .999 & .0009 & .0001 \\ .999 & .0009 & .0001 \\ .9996 & .0003 & .0001 \\ \hline .0004 & .9995 & .0001 \\ .0009 & .999 & .0001 \\ \hline .00005 & .00005 & .9999 \\ .00006 & .00004 & .9999 \\ .00005 & .00005 & .9999 \end{array}\right)$$

$$= \left(\begin{array}{l|l|l} .99911 & .00079 & .00010 \\ \hline .00061 & .99929 & .00010 \\ \hline .00006 & .00004 & .99990 \end{array}\right) = C.$$

Its eigenvalues are 1.0, .9998, and .9985 and its stationary probability vector is

$$\xi = (.22252,\ .27748,\ .50000).$$

Step (c): Compute global approximation
We are now in a position to form the final approximation to the stationary probability vector. It is given by the approximation

$$\pi \approx (\xi_1 u_1,\ \xi_2 u_2,\ \ldots,\ \xi_N u_N),$$

where the u_i are approximations to $\pi_i/\|\pi_i\|_1$.

Example 10.23 In the running example we obtain the approximate solution

$$\pi^* = (.08932,\ .09273,\ .04046,\ .15855,\ .11893,\ .12037,\ .27781,\ .10182),$$

which may be compared to the exact solution

$$\pi = (.08928,\ .09276,\ .04049,\ .15853,\ .11894,\ .12039,\ .27780,\ .10182).$$

To summarize, an approximation to the stationary probability vector of an NCD Markov chain may be obtained by first solving each of the blocks separately, then forming and solving the coupling matrix, and finally constructing the approximate solution from these pieces. It is implicitly understood that the states have been ordered so that the transition probability matrix has the required NCD block structure. Algorithmically, the entire procedure may be written as

Algorithm 10.7: NCD Decomposition Approximation

```
1. Solve the individual blocks: $u_i P_{ii} = \lambda_{i_1} u_i,\ \ u_i e = 1$ for $i = 1, 2, \ldots, N$.
2. (a) Form the coupling matrix: $(C)_{ij} = u_i P_{ij} e$.
   (b) Solve the coupling problem: $\xi = \xi C,\ \ \xi e = 1$.
3. Construct the approximate solution: $\pi^* = (\xi_1 u_1,\ \xi_2 u_2,\ \ldots,\ \xi_N u_N)$.
```

The question now arises as to whether we can incorporate this approximation back into the decomposition algorithm to get an even better approximation. Notice that the u_i are used to form the coupling matrix. If we replace them with the new approximations $\xi_i u_i$, we will obtain exactly the same solution to the coupling matrix as before. Hence, there will be no change in the computed approximation. However, it was found that applying a power step to the approximation before plugging it back into the decomposition method had a very salutory effect. Later this power step was replaced by a block Gauss–Seidel step and became known as a disaggregation step; forming and solving the matrix C being the aggregation step. The entire procedure is referred to as *iterative aggregation/disaggregation (IAD)*.

The algorithm is presented below. The iteration number is indicated by a superscript in parentheses on the appropriate variable names. The initial vector is designated as $\pi^{(0)}$. This may be the approximation obtained from the simple decomposition approach, or it may be just a random selection. Observe that many of the steps have corresponding steps in the decomposition algorithm. For instance, the formation of the coupling matrix (Step 3) is identical in both. Step 4(a) has its counterpart in the formation of the computed approximation in the decomposition algorithm, while Step 4(b) corresponds to solving the different blocks, except that in the IAD algorithm, a block Gauss–Seidel step is used.

Algorithm 10.8: Iterative Aggregation/Disaggregation

```
1. Let $\pi^{(0)} = (\pi_1^{(0)}, \pi_2^{(0)}, \ldots, \pi_N^{(0)})$ be a given initial approximation
   to the solution $\pi$, and set $m = 1$.
2. Compute $\phi^{(m-1)} = (\phi_1^{(m-1)}, \phi_2^{(m-1)}, \ldots, \phi_N^{(m-1)})$, where
       $$\phi_i^{(m-1)} = \frac{\pi_i^{(m-1)}}{||\pi_i^{(m-1)}||_1}, \quad i = 1,\ 2,\ \ldots,\ N.$$
3. (a) Form the coupling matrix: $C_{ij}^{(m-1)} = \phi_i^{(m-1)} P_{ij} e$ for $i, j = 1, 2, \ldots, N$.
   (b) Solve the coupling problem: $\xi^{(m-1)} C^{(m-1)} = \xi^{(m-1)},\ \ \xi^{(m-1)} e = 1$.
4. (a) Construct the row vector
       $$z^{(m)} = (\xi_1^{(m-1)} \phi_1^{(m-1)},\ \xi_2^{(m-1)} \phi_2^{(m-1)},\ \ldots,\ \xi_N^{(m-1)} \phi_N^{(m-1)}).$$
   (b) Solve the following $N$ systems of equations to find $\pi^{(m)}$:
       $$\pi_k^{(m)} = \pi_k^{(m)} P_{kk} + \sum_{j>k} z_j^{(m)} P_{jk} + \sum_{j<k} \pi_j^{(m)} P_{jk}, \quad k = 1, 2,\ \ldots,\ N. \qquad (10.25)$$
5. Normalize and conduct a test for convergence.
   If satisfactory, then stop and take $\pi^{(m)}$ to be the required
   solution vector.
   Otherwise set $m = m + 1$ and go to step 2.
```

In these methods, it is important that the matrix has the block structure needed by the algorithms and it may be necessary to reorder the states to get this property. Only after reordering the states can we guarantee that the resulting transition matrix will have a property that directly reflects the structural characteristics of the NCD system. If the partitioning provided to the algorithm does not match the decomposability characteristics of the matrix, the convergence behavior may be much less satisfactory. A reordering of the states can be accomplished by treating the Markov chain as a directed graph in which edges with small weights (probabilities) are removed. A graph algorithm must then be used to find the connected components of $\hat{P} + \hat{P}^T$, where $\hat{P}$ is the modified transition probability matrix. The complexity of the algorithm is $O(|V| + |E|)$, where $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. Details are provided in [14].

Example 10.24 When applied to the Courtois matrix, we find that both the iterative aggregation and disaggregation (IAD) method and the block Gauss–Seidel (BGS) method are very effective. The table below shows that convergence is achieved to full machine precision in only four iterations with the IAD method and nine iterations with BGS. In both cases, the diagonal block equations are solved using LU decomposition.

IAD and BGS residuals for the Courtois NCD matrix

Iteration	IAD residual	BGS residual
	$1.0e{-}05\times$	$1.0e{-}05\times$
1	0.93581293961421	0.94805408435419
2	0.00052482104506	0.01093707688215
3	0.00000000280606	0.00046904081241
4	0.00000000000498	0.00002012500900
5	0.00000000000412	0.00000086349742
6	0.00000000000351	0.00000003705098
7	0.00000000000397	0.00000000158929
8	0.00000000000529	0.00000000006641
9	0.00000000000408	0.00000000000596
10	0.00000000000379	0.00000000000395

We now turn our attention to some implementation details. The critical points are Steps 3 and 4(b). In Step 3, it is more efficient to compute $P_{ij}e$ only once for each block and to store it somewhere for use in all future iterations. This is only possible if sufficient memory is available; otherwise it is necessary to compute it each time it is needed. To obtain the vector ξ in Step 3(b), any of the methods discussed in the previous section may be used, since the vector ξ is simply the stationary probability vector of an irreducible stochastic matrix C.

In Step 4(b), each of the N systems of equations in (10.25) can be written as $Bx = r$ where $B = (I - P_{kk})^T$ and

$$r^T = \sum_{j>k} z_j P_{jk} + \sum_{j<k} \pi_j P_{jk}, \ k = 1, 2, \ldots, N.$$

In all cases, P_{kk} is a strictly substochastic matrix so that B is nonsingular. The vector r will have small norm if the system is NCD. If a direct method is used, the LU decomposition of $(I - P_{kk})$, $k = 1, 2, \ldots, N$, need only be performed once, since this remains unchanged from one iteration to the next. If an iterative method is used we have an iteration algorithm within an iteration algorithm. In this case it is advantageous to perform only a small number of iterations, (e.g., 8–12 of the Gauss–Seidel method) each time a solution of $(I - P_{kk})^T x = r$ is needed but to use the final approximation at one step as the initial approximation the next time the solution of that same subsystem is needed.

Example 10.25 Returning again to the Courtois matrix, the table below shows the number of iterations needed to achieve full machine precision when the diagonal block equations in the IAD method are solved using the Gauss–Seidel iterative method. It can be seen that now an additional iteration is needed.

IAD: Gauss–Seidel for Block solutions	
Iteration	Residual: $\hat{\pi}(I - P)$
	$1.0e-03\times$
1	0.14117911369086
2	0.00016634452597
3	0.00000017031189
4	0.00000000015278
5	0.00000000000014
6	0.00000000000007
7	0.00000000000006
8	0.00000000000003
9	0.00000000000003
10	0.00000000000006

Also, if we check the convergence of the inner Gauss–Seidel method for the diagonal blocks—as shown in the table below for the first diagonal block of size (3×3)—it can be seen that the iterations stagnate after just a few steps. After about six iterations during each global iteration, progress slows to a crawl. It is for this reason that only a small number of iterations should be used when iterative methods are used to solve the (inner) block equations.

	Global iteration			
Inner Iteration	1	2	3	4
1	0.0131607445	0.000009106488	0.00000002197727	0.00000000002345
2	0.0032775892	0.000002280232	0.00000000554827	0.00000000000593
3	0.0008932908	0.000000605958	0.00000000142318	0.00000000000151
4	0.0002001278	0.000000136332	0.00000000034441	0.00000000000037
5	0.0001468896	0.000000077107	0.00000000010961	0.00000000000011
6	0.0001124823	0.000000051518	0.00000000003470	0.00000000000003
7	0.0001178683	0.000000055123	0.00000000003872	0.00000000000002
8	0.0001156634	0.000000053697	0.00000000003543	0.00000000000002
9	0.0001155802	0.000000053752	0.00000000003596	0.00000000000002
10	0.0001149744	0.000000053446	0.00000000003562	0.00000000000002
11	0.0001145044	0.000000053234	0.00000000003552	0.00000000000002
12	0.0001140028	0.000000052999	0.00000000003535	0.00000000000002
13	0.0001135119	0.000000052772	0.00000000003520	0.00000000000002
14	0.0001130210	0.000000052543	0.00000000003505	0.00000000000002
15	0.0001125327	0.000000052316	0.00000000003490	0.00000000000002
16	0.0001120464	0.000000052090	0.00000000003475	0.00000000000002

The following Matlab code may be used to experiment with IAD methods. It mirrors the block Gauss–Seidel method in its parameters. It was used to produce the results provided above.

Matlab code for Iterative Aggregation/Disaggregation

```
function [soln,res] = kms(P,ni,itmax1,itmax2)
[n,n] = size(P); [na,nb] = size(ni);

%%%%%%% ITERATIVE AGGREGATION/DISAGGREGATION FOR pi*P = pi %%%%%%%%%%%%%%

bl(1) = 1;                                          %  Get beginning and end
for k = 1:nb, bl(k+1) = bl(k)+ni(k); end            %  points of each block

E = zeros(n,nb);                                    %  Form (n x nb) matrix E
next = 0;                                      %  (This is needed in forming
for i = 1:nb,                                       %   the coupling matrix: A )
    for k = 1:ni(i); next = next+1; E(next,i) = 1; end
end
Pije = P*E;                 %  Compute constant part of coupling matrix
Phi = zeros(nb,n);          %  Phi, used in forming the coupling matrix,
for m = 1:nb,               %  keeps normalized parts of approximation
    for j = 1:ni(m), Phi(m,bl(m)+j-1) = 1/ni(m); end
end

A = Phi*Pije;                                       %  Form the coupling matrix A
AA = (A-eye(nb))';    en = [zeros(nb-1,1);1];
xi = inv([AA(1:nb-1,1:nb);ones(1,nb)])*en;     %  Solve the coupling matrix
z = Phi'*xi;                                        %  Initial approximation


%%%%%%%%%%%%%%%%%%%%%%%%%  BEGIN OUTER LOOP  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%


for iter = 1:itmax1,

    for m = 1:nb,              %  Solve all diag. blocks; Pmm y = b
        Pmm = P(bl(m):bl(m+1)-1,bl(m):bl(m+1)-1);
                                    %  Get coefficient block, Pmm
        b = z(bl(m):bl(m+1)-1)'*Pmm-z'*P(1:n,bl(m):bl(m+1)-1);     %  RHS

        y = inv(Pmm'-eye(ni(m)))*b';  %  Substitute this line for the 2
                     % lines below it, to solve Pmm by iterative method.
        %x0 = z(bl(m):bl(m+1)-1);        %  Get new starting vector
        %[y,resid] = gs(Pmm'-eye(ni(m)),x0,b',itmax2);
                                        % y is soln for block Pmm
        for j = 1:ni(m), z(bl(m)+j-1) = y(j); end %  Update solution vector
        y = y/norm(y,1);                          %  Normalize y

        for j = 1:ni(m),
            Phi(m,bl(m)+j-1) = y(j);              %  Update Phi
        end
    end
    pi = z;
    res(iter) = norm((P'-eye(n))*pi,2);       % Compute residual
    A = Phi*Pije; AA = (A-eye(nb))';          % Form the coupling matrix A
    xi = inv([AA(1:nb-1,1:nb);ones(1,nb)])*en; % Solve the coupling matrix
    z  = Phi'*xi;                             % Compute new approximation
end

soln = pi; res = res';
```

10.6 The Matrix Geometric/Analytic Methods for Structured Markov Chains

We now turn to the numerical solution of Markov chains whose transition matrices have a special block structure—a block structure that arises frequently when modeling queueing systems—and examine an approach pioneered by Neuts [39, 40]. In the simplest case, these matrices are infinite block tridiagonal matrices in which the three diagonal blocks repeat after some initial period. To capture this nonzero structure, we write such a matrix as

$$\begin{pmatrix} B_{00} & B_{01} & 0 & 0 & 0 & 0 & \cdots \\ B_{10} & A_1 & A_2 & 0 & 0 & 0 & \cdots \\ 0 & A_0 & A_1 & A_2 & 0 & 0 & \cdots \\ 0 & 0 & A_0 & A_1 & A_2 & 0 & \cdots \\ & & & \ddots & \ddots & \ddots & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}, \tag{10.26}$$

in which submatrices $A_0,\ A_1$, and A_2 are square and have the same dimension; the matrix B_{00} is also square and need not have the same size as A_1, and the dimensions of B_{01} and B_{10} are defined to be in accordance with the dimensions of B_{00} and A_1. A transition matrix having this structure arises when each state of the Markov chain can be represented as a pair $\{(\eta, k), \eta \geq 0,\ 1 \leq k \leq K\}$ and the states ordered, first according to increasing value of the parameter η and, for states with the same η value, by increasing value of k. This has the effect of grouping the states into "levels" according to their η value. The block tridiagonal effect is achieved when transitions are permitted only between states of the same level (diagonal blocks), to states in the next highest level (superdiagonal blocks), and to states in the adjacent lower level (subdiagonal blocks). The repetitive nature of the blocks themselves arises if, after boundary conditions are taken into consideration (which gives the initial blocks $B_{00},\ B_{01}$, and B_{10}) the transition rates/probabilities are identical from level to level. A Markov chain whose transition matrix has this block tridiagonal structure is said to belong to the class of *quasi-birth-death* (QBD) processes.

Example 10.26 Consider the Markov chain whose state transition diagram is shown in Figure 10.1.

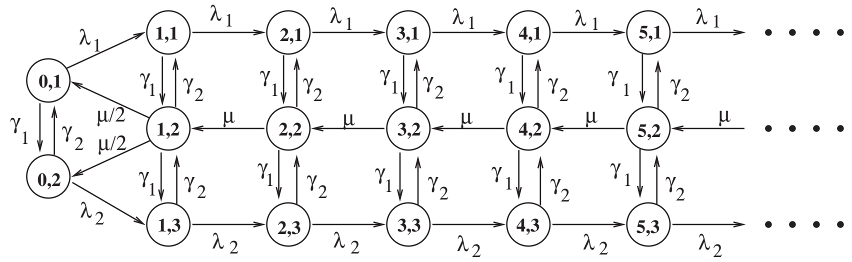

Figure 10.1. State transition diagram for an *M/M/*1-type process.

Its transition rate matrix is

$$
Q = \left(\begin{array}{cc|ccc|ccc|ccc|ccc|ccc|c}
* & \gamma_1 & \lambda_1 \\
\gamma_2 & * & & & \lambda_2 \\
\hline
 & & * & \gamma_1 & & \lambda_1 \\
\mu/2 & \mu/2 & \gamma_2 & * & \gamma_1 \\
 & & & \gamma_2 & * & & & \lambda_2 \\
\hline
 & & & & & * & \gamma_1 & & \lambda_1 \\
 & & & \mu & & \gamma_2 & * & \gamma_1 \\
 & & & & & & \gamma_2 & * & & & \lambda_2 \\
\hline
 & & & & & & & & * & \gamma_1 & & \lambda_1 \\
 & & & & & & \mu & & \gamma_2 & * & \gamma_1 \\
 & & & & & & & & & \gamma_2 & * & & & \lambda_2 \\
\hline
 & & & & & & & & & & & * & \gamma_1 & & \lambda_1 \\
 & & & & & & & & & \mu & & \gamma_2 & * & \gamma_1 \\
 & & & & & & & & & & & & \gamma_2 & * & & & \lambda_2 \\
\hline
 & & & & & & & & & & & & \ddots & & & \ddots & & \ddots
\end{array}\right)
$$

and has the typical block tridiagonal structure which makes it an ideal candidate for solution by the numerical techniques described in this section. Its diagonal elements, marked by asterisks, are such that the sum across each row is zero. We have the following block matrices:

$$
A_0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \mu & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad A_1 = \begin{pmatrix} -(\gamma_1 + \lambda_1) & \gamma_1 & 0 \\ \gamma_2 & -(\mu + \gamma_1 + \gamma_2) & \gamma_1 \\ 0 & \gamma_2 & -(\gamma_2 + \lambda_2) \end{pmatrix}, \quad A_2 = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \lambda_2 \end{pmatrix},
$$

and

$$
B_{00} = \begin{pmatrix} -(\gamma_1 + \lambda_1) & \gamma_1 \\ \gamma_2 & -(\gamma_2 + \lambda_2) \end{pmatrix}, \quad B_{01} = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & 0 & \lambda_2 \end{pmatrix}, \quad B_{10} = \begin{pmatrix} 0 & 0 \\ \mu/2 & \mu/2 \\ 0 & 0 \end{pmatrix}.
$$

The most common extensions of this simplest case are matrices which are block upper Hessenberg (referred to as *M/G/*1 type and solved using the matrix analytic approach) and those that are block lower Hessenberg (referred to as *G/M/*1 type, solved using the matrix geometric approach). Both are treated in this text. Results are also available for the case in which the parameters η and k are both finite, which results in a finite Markov chain, as well as for the case in which transitions may be dependent on the level η. These last extensions are beyond the scope of this text and the interested reader is invited to consult the references.

10.6.1 The Quasi-Birth-Death Case

Quasi-birth-death processes can be conveniently and efficiently solved using the *matrix geometric method*. We begin by considering the case when the blocks of a QBD process are reduced to single elements. Consider, for example, the following infinite infinitesimal generator:

$$
Q = \begin{pmatrix}
-\lambda & \lambda & & & \\
\mu & -(\lambda + \mu) & \lambda & & \\
 & \mu & -(\lambda + \mu) & \lambda & \\
 & & \mu & -(\lambda + \mu) & \lambda \\
 & & & \ddots & \ddots & \ddots
\end{pmatrix}.
$$

This may be associated with a random walk problem in which the probability of moving from any state k to state $k+1$ is $\lambda/(\lambda+\mu)$ and the probability of moving from state k to state $k-1$ is $\mu/(\lambda+\mu)$. In a later chapter, we shall associate this infinitesimal generator with the *M/M/*1 queue. From $\pi Q = 0$, we may write $-\lambda\pi_0 + \mu\pi_1 = 0$, which gives $\pi_1 = (\lambda/\mu)\pi_0$, and in general

$$\lambda\pi_{i-1} - (\lambda+\mu)\pi_i + \mu\pi_{i+1} = 0.$$

We proceed by induction to show that $\pi_{i+1} = (\lambda/\mu)\pi_i$ for $i = 1, 2, \ldots$. We have already established the basis clause $\pi_1 = (\lambda/\mu)\pi_0$. From the inductive hypothesis, we have $\pi_i = (\lambda/\mu)\pi_{i-1}$ and hence

$$\pi_{i+1} = \left(\frac{\lambda+\mu}{\mu}\right)\pi_i - \left(\frac{\lambda}{\mu}\right)\pi_{i-1} = \left(\frac{\lambda}{\mu}\right)\pi_i,$$

which is the desired result. It now follows that

$$\pi_i = \left(\frac{\lambda}{\mu}\right)^i \pi_0 = \rho^i \pi_0,$$

where $\rho = \lambda/\mu$. Thus, once π_0 is known, the remaining values, π_i, $i = 1, 2, \ldots$, may be determined recursively. A similar result exists when Q is a QBD process: In this case, the parameter ρ becomes a square matrix R of order K and the components π_i of the stationary distribution become subvectors of length K.

Let Q be the infinitesimal generator of a QBD process. Then

$$Q = \begin{pmatrix} B_{00} & B_{01} & 0 & 0 & 0 & 0 & \cdots \\ B_{10} & A_1 & A_2 & 0 & 0 & 0 & \cdots \\ 0 & A_0 & A_1 & A_2 & 0 & 0 & \cdots \\ 0 & 0 & A_0 & A_1 & A_2 & 0 & \cdots \\ & & & \ddots & \ddots & \ddots & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix},$$

and the stationary distribution is obtained from $\pi Q = 0$. Let π be partitioned conformally with Q, i.e.,

$$\pi = (\pi_0, \pi_1, \pi_2, \ldots),$$

where

$$\pi_i = (\pi(i,1), \pi(i,2), \ldots \pi(i,K))$$

for $i = 0, 1, \ldots$, and $\pi(i,k)$ is the probability of finding the system in state (i,k) at steady state. This gives the following equations:

$$\begin{aligned} \pi_0 B_{00} + \pi_1 B_{10} &= 0, \\ \pi_0 B_{01} + \pi_1 A_1 + \pi_2 A_0 &= 0, \\ \pi_1 A_2 + \pi_2 A_1 + \pi_3 A_0 &= 0, \\ &\vdots \\ \pi_{i-1} A_2 + \pi_i A_1 + \pi_{i+1} A_0 &= 0, \quad i = 2, 3, \ldots. \end{aligned}$$

In analogy with the point situation, it may be shown that there exists a constant matrix R such that

$$\pi_i = \pi_{i-1} R \quad \text{for } i = 2, 3, \ldots. \tag{10.27}$$

The subvectors π_i are said to be *geometrically* related to each other since

$$\pi_i = \pi_1 R^{i-1} \quad \text{for } i = 2, 3, \ldots. \tag{10.28}$$

If the subvectors π_0 and π_1 and the *rate matrix* R can be found then the remaining subvectors of the stationary distribution may be formed using Equation (10.27). Returning to

$$\pi_{i-1}A_2 + \pi_i A_1 + \pi_{i+1}A_0 = 0$$

and substituting from Equation (10.28), we obtain, for $i = 2, 3, \ldots,$

$$\pi_1 R^{i-2}A_2 + \pi_1 R^{i-1}A_1 + \pi_1 R^i A_0 = 0,$$

i.e.,

$$\pi_1 R^{i-2}\left(A_2 + RA_1 + R^2A_0\right) = 0.$$

It is now apparent that R can be computed from

$$\left(A_2 + RA_1 + R^2A_0\right) = 0. \tag{10.29}$$

The simplest way to accomplish this is by successive substitution. From Equation (10.29) and using the fact that A_1 must be nonsingular, we have

$$A_2A_1^{-1} + R + R^2A_0A_1^{-1} = 0,$$

i.e.,

$$R = -A_2A_1^{-1} - R^2A_0A_1^{-1} = -V - R^2W,$$

where $V = A_2A_1^{-1}$ and $W = A_0A_1^{-1}$. This leads to the successive substitution procedure proposed by Neuts, namely,

$$R_{(0)} = 0, \quad R_{(k+1)} = -V - R_{(k)}^2W, \quad k = 0, 1, 2, \ldots. \tag{10.30}$$

Neuts has shown that the sequence of matrices $R_{(k)},\ k = 0, 1, 2, \ldots,$ is nondecreasing and converges to the rate matrix R. The process is halted once successive differences are less than a specified tolerance criterion. Unfortunately, this simple approach has the disadvantage of frequently requiring many iterations before a sufficiently accurate matrix R is obtained. On the other hand, a *logarithmic reduction algorithm* developed by Latouche and Ramaswami [27] has extremely fast quadratic convergence (the number of decimal places doubles at each iteration). The development of this algorithm is beyond the scope of this text: it is presented in pseudocode and without additional comment, at the end of this section. There are also situations in which the rate matrix R can be computed explicitly, without the need to conduct any iterations at all.

The only remaining problem is the derivation of π_0 and π_1. The first two equations of $\pi Q = 0$ are

$$\pi_0B_{00} + \pi_1B_{10} = 0,$$
$$\pi_0B_{01} + \pi_1A_1 + \pi_2A_0 = 0.$$

Replacing π_2 with $\pi_1 R$ and writing these equations in matrix form, we obtain

$$(\pi_0, \pi_1)\begin{pmatrix} B_{00} & B_{01} \\ B_{10} & A_1 + RA_0 \end{pmatrix} = (0, 0). \tag{10.31}$$

Given the rate matrix R and blocks $B_{00},\ B_{01},\ B_{10},\ A_1$, and A_0, this system may be solved to obtain π_0 and π_1. Since this is a homogeneous system of equations, the computed solution needs to be normalized so that the components of π sum to 1. In other words, we insist that $\pi e = 1$. Thus

$$\begin{aligned}1 = \pi e &= \pi_0 e + \pi_1 e + \sum_{i=2}^{\infty} \pi_i e \\ &= \pi_0 e + \pi_1 e + \sum_{i=2}^{\infty} \pi_1 R^{i-1} e \\ &= \pi_0 e + \sum_{i=1}^{\infty} \pi_1 R^{i-1} e = \pi_0 e + \sum_{i=0}^{\infty} \pi_1 R^i e.\end{aligned}$$

This implies the condition

$$\pi_0 e + \pi_1 \left(\sum_{i=0}^{\infty} R^i \right) e = 1.$$

The eigenvalues of R lie *inside* the unit circle, which means that $(I - R)$ is nonsingular and hence

$$\left(\sum_{i=0}^{\infty} R^i \right) = (I - R)^{-1}. \tag{10.32}$$

This enables us to complete the normalization of the vectors π_0 and π_1 by computing

$$\alpha = \pi_0 e + \pi_1 (I - R)^{-1} e$$

and dividing the computed subvectors π_0 and π_1 by α.

In the simpler case, when B_{00} is the same size as the A blocks and when $B_{01} = A_2$, then $\pi_i = \pi_0 R^i$ for $i = 1, 2, \ldots$. In this case the system of equations (10.31) can be replaced with the simpler system, $\pi_0(B_{00} + RB_{10}) = 0$, from which π can be computed and then normalized so that $\pi_0 (I - R)^{-1} e = 1$.

In a previous discussion about a random walk problem, it was observed that the Markov chain is positive recurrent only if the probability of moving to higher-numbered states is strictly less than the probability of moving to lower-numbered states. A similar condition exists for a QBD process to be ergodic, namely, that the *drift* to higher-numbered levels must be strictly less than the *drift* to lower levels. Let the stationary distribution of the infinitesimal generator $A = A_0 + A_1 + A_2$ be denoted by π_A. For a QBD process to be ergodic, the following condition must hold:

$$\pi_A A_2 e \;<\; \pi_A A_0 e. \tag{10.33}$$

Recall that the elements of A_2 move the process up to a higher-numbered level while those of A_0 move it down a level. Indeed, it is from this condition that Neuts shows that the spectral radius of R is strictly less than 1 and consequently that the matrix $I - R$ is nonsingular.

When the Markov chain under consideration is a discrete-time Markov chain characterized by a stochastic transition probability matrix rather than the continuous-time version discussed in this section, then an almost identical analysis can be performed: it suffices to replace $-A_1^{-1}$ with $(I - A_1)^{-1}$. But be careful: in the first case, A_1 is the repeating diagonal block in a transition rate matrix while in the second it is the repeating diagonal block in a stochastic matrix. The formulation of the equations is the same, just the values in the blocks change according to whether the global matrix is an infinitesimal generator or a stochastic matrix. Before proceeding to an example, we summarize the steps that must be undertaken when solving a QBD process by the matrix geometric method:

1. Ensure that the matrix has the requisite block structure.
2. Use Equation (10.33) to ensure that the Markov chain is ergodic.

3. Use Equation (10.30) to compute the matrix R.
4. Solve the system of equations (10.31) for π_0 and π_1.
5. Compute the normalizing constant α and normalize π_0 and π_1.
6. Use Equation (10.27) to compute the remaining components of the stationary distribution vector.

Example 10.27 We shall apply the matrix geometric method to the Markov chain of Example 10.26 using the following values of the parameters:

$$\lambda_1 = 1,\ \lambda_2 = .5,\ \mu = 4,\ \gamma_1 = 5,\ \gamma_2 = 3.$$

The infinitesimal generator is then given by

$$Q = \left(\begin{array}{rr|rrr|rrr|rrr|r}
-6 & 5.0 & 1 & & & & & & & & & \\
3 & -3.5 & & & .5 & & & & & & & \\
\hline
 & & -6 & 5 & & 1 & & & & & & \\
2 & 2.0 & 3 & -12 & 5.0 & & & & & & & \\
 & & & 3 & -3.5 & & & .5 & & & & \\
\hline
 & & & & & -6 & 5 & & 1 & & & \\
 & & & 4 & & 3 & -12 & 5.0 & & & & \\
 & & & & & & 3 & -3.5 & & & .5 & \\
\hline
 & & & & & & \ddots & & \ddots & & & \ddots
\end{array}\right).$$

1. The matrix obviously has the correct QBD structure.
2. We check that the system is stable by verifying Equation (10.33). The infinitesimal generator matrix

$$A = A_0 + A_1 + A_2 = \begin{pmatrix} -5 & 5 & 0 \\ 3 & -8 & 5 \\ 0 & 3 & -3 \end{pmatrix}$$

has stationary probability vector

$$\pi_A = (.1837,\ .3061,\ .5102)$$

and

$$.4388 = \pi_A A_2 e\ <\ \pi_A A_0 e = 1.2245.$$

3. We now initiate the iterative procedure to compute the rate matrix R. The inverse of A_1 is

$$A_1^{-1} = \begin{pmatrix} -.2466 & -.1598 & -.2283 \\ -.0959 & -.1918 & -.2740 \\ -.0822 & -.1644 & -.5205 \end{pmatrix},$$

which allows us to compute

$$V = A_2 A_1^{-1} = \begin{pmatrix} -.2466 & -.1598 & -.2283 \\ 0 & 0 & 0 \\ -.0411 & -.0822 & -.2603 \end{pmatrix} \quad \text{and}$$

$$W = A_0 A_1^{-1} = \begin{pmatrix} 0 & 0 & 0 \\ -.3836 & -.7671 & -1.0959 \\ 0 & 0 & 0 \end{pmatrix}.$$

Equation (10.30) becomes

$$R_{(k+1)} = \begin{pmatrix} .2466 & .1598 & .2283 \\ 0 & 0 & 0 \\ .0411 & .0822 & .2603 \end{pmatrix} + R_{(k)}^2 \begin{pmatrix} 0 & 0 & 0 \\ .3836 & .7671 & 1.0959 \\ 0 & 0 & 0 \end{pmatrix},$$

and iterating successively, beginning with $R_{(0)} = 0$, we find

$$R_{(1)} = \begin{pmatrix} .2466 & .1598 & .2283 \\ 0 & 0 & 0 \\ .0411 & .0822 & .2603 \end{pmatrix}, \quad R_{(2)} = \begin{pmatrix} .2689 & .2044 & .2921 \\ 0 & 0 & 0 \\ .0518 & .1036 & .2909 \end{pmatrix},$$

$$R_{(3)} = \begin{pmatrix} .2793 & .2252 & .3217 \\ 0 & 0 & 0 \\ .0567 & .1134 & .3049 \end{pmatrix}, \quad \cdots .$$

Observe that the elements are nondecreasing, as predicted by Neuts. After 48 iterations, successive differences are smaller than 10^{-12}, at which point

$$R_{(48)} = \begin{pmatrix} .2917 & .2500 & .3571 \\ 0 & 0 & 0 \\ .0625 & .1250 & .3214 \end{pmatrix}.$$

4. Proceeding to the boundary conditions,

$$(\pi_0, \pi_1) \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & A_1 + RA_0 \end{pmatrix} = (\pi_0, \pi_1) \left(\begin{array}{rr|rrr} -6 & 5.0 & 1 & 0 & 0 \\ 3 & -3.5 & 0 & 0 & .5 \\ \hline 0 & 0 & -6 & 6.0 & 0 \\ 2 & 2.0 & 3 & -12.0 & 5.0 \\ 0 & 0 & 0 & 3.5 & -3.5 \end{array}\right) = (0, 0).$$

We can solve this by replacing the last equation with the equation $\pi_{0_1} = 1$, i.e., setting the first component of the subvector π_0 to 1. The system of equations becomes

$$(\pi_0, \pi_1) \left(\begin{array}{rr|rrr} -6 & 5.0 & 1 & 0 & 1 \\ 3 & -3.5 & 0 & 0 & 0 \\ \hline 0 & 0 & -6 & 6.0 & 0 \\ 2 & 2.0 & 3 & -12.0 & 0 \\ 0 & 0 & 0 & 3.5 & 0 \end{array}\right) = (0, 0 \mid 0, 0, 1)$$

with solution

$$(\pi_0, \pi_1) = (1.0,\ 1.6923,\ \mid .3974,\ .4615,\ .9011).$$

5. The normalization constant is

$$\begin{aligned} \alpha &= \pi_0 e + \pi_1 (I - R)^{-1} e \\ &= (1.0,\ 1.6923)e + (.3974,\ .4615,\ .9011) \begin{pmatrix} 1.4805 & .4675 & .7792 \\ 0 & 1 & 0 \\ .1364 & .2273 & .15455 \end{pmatrix} e \\ &= 2.6923 + 3.2657 = 5.9580, \end{aligned}$$

which allows us to compute

$$\pi_0/\alpha = (.1678,\ .2840), \quad \text{and} \quad \pi_1/\alpha = (.0667,\ .0775,\ .1512).$$

6. Successive subcomponents of the stationary distribution are now computed from $\pi_k = \pi_{k-1}R$. For example,

$$\pi_2 = \pi_1 R = (.0667,\ .0775,\ .1512)\begin{pmatrix} .2917 & .2500 & .3571 \\ 0 & 0 & 0 \\ .0625 & .1250 & .3214 \end{pmatrix} = (.0289,\ .0356,\ .0724),$$

$$\pi_3 = \pi_2 R = (.0289,\ .0356,\ .0724)\begin{pmatrix} .2917 & .2500 & .3571 \\ 0 & 0 & 0 \\ .0625 & .1250 & .3214 \end{pmatrix} = (.0130,\ .0356,\ .0336),$$

and so on.

Steps 2 through 6 of the basic matrix geometric method for QBD processes are embedded in the following Matlab code in which n is the order of the blocks A_i and m is the order of B_{00}:

Matlab code for QBD processes

```
function[R,pi0,pi1] = mgm(n,A0,A1,A2,m,B00,B01,B10)

%              Matrix Geometric Method

%%%%%%%%%%%%%%  Form Neuts' R matrix   %%%%%%%%%%%%%%%
     V = A2 * inv(A1);  W = A0 * inv(A1);
     R = zeros(n,n);  Rbis = -V - R*R * W;
     iter = 1;
     while (norm(R-Rbis,1) > 1.0e-10)
        R = Rbis;  Rbis = -V - R*R * W;
        iter = iter+1;
     end
     R = Rbis;

%%%%%%%%  Form and solve boundary equations   %%%%%%%%
     C = [B00,B01;B10,A1+R*A0];
     z = zeros(n+m,1); z(1) = 1;
     Cn = [C(1:n+m,1:n+m-1),z];
     rhs = zeros(1,n+m); rhs(n+m) = 1;
     pi = rhs*inv(Cn);
     pi0 = pi(1:m);   pi1 = pi(m+1:m+n);

%%%%%%%%%%%%  Normalize computed solution   %%%%%%%%%%%%
     alpha = norm(pi0,1)+ norm(pi1*inv(eye(n)-R),1);
     pi0 = pi0/alpha;  pi1 = pi1/alpha;

%%%%%%%%%%%  Compute successive subvectors   %%%%%%%%%%%
     pi2 = pi1*R;  pi3 = pi2*R;   % and so on.
```

Algorithm 10.10: Logarithmic Reduction—Quadratic Convergence for QBD Processes

```
1. Initialize: i = 0;
            $C_0 = (I - A_1)^{-1}A_0$;      $C_2 = (I - A_1)^{-1}A_2$;   if stochastic, or
   %%%      $C_0 = -A_1^{-1}A_0$;           $C_2 = -A_1^{-1}A_2$;        if infinitesimal generator.
            $T = C_0$;                      $S = C_2$;
```

```
2. While  $\|e - Se\|_\infty < \epsilon$:
    • $i = i + 1$;
    • $A_1^* = C_0C_2 + C_2C_0$;   $A_0^* = C_0^2$;   $A_2^* = C_2^2$;
    • $C_0 = (I - A_1^*)^{-1}A_0^*$;   $C_2 = (I - A_1^*)^{-1}A_2^*$;
    • $S = S + TC_2$;   $T = TC_0$;
3. Termination
    $G = S$;   $U = A_1 + A_0G$;   $R = A_0(I - U)^{-1}$.
```

10.6.2 Block Lower Hessenberg Markov Chains

In a lower Hessenberg matrix A, all elements a_{ij} must be zero for values of $j > i + 1$. In other words, if moving from top right to bottom left, we designate the three diagonals of a tridiagonal matrix as the superdiagonal, the diagonal, and the subdiagonal, then a lower Hessenberg matrix can have nonzero elements only on and below the superdiagonal. For example, the following matrix is a 6×6 lower Hessenberg matrix:

$$A = \begin{pmatrix} a_{00} & a_{01} & 0 & 0 & 0 & 0 \\ a_{10} & a_{11} & a_{12} & 0 & 0 & 0 \\ a_{20} & a_{21} & a_{22} & a_{23} & 0 & 0 \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & 0 \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{50} & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix}.$$

Block lower Hessenberg matrices, found in *G/M/*1-type stochastic processes, are just the block counterparts of lower Hessenberg matrices. In a similar manner, we can define block *upper* Hessenberg matrices, which are found in *M/G/*1-type stochastic processes, as block matrices whose only nonzero blocks are on or above the diagonal blocks and along the subdiagonal block. One is essentially the block transpose of the other. This leads us to caution the reader about the notation used to designate the individual blocks. With block lower Hessenberg matrices, the nonzero blocks are numbered from right to left as A_0, A_1, A_2, ..., while in block upper Hessenberg matrices they are numbered in the reverse order, from left to right. As we have just seen, we choose to denote the blocks in QBD processes (which are simultaneously both upper and lower block Hessenberg) from left to right, i.e., using the block upper Hesssenberg notation. Our interest in this section is with Markov chains whose infinitesimal generators Q have the following repetitive block lower Hessenberg structure:

$$Q = \begin{pmatrix} B_{00} & B_{01} & 0 & 0 & 0 & 0 & 0 & \cdots \\ B_{10} & B_{11} & A_0 & 0 & 0 & 0 & 0 & \cdots \\ B_{20} & B_{21} & A_1 & A_0 & 0 & 0 & 0 & \cdots \\ B_{30} & B_{31} & A_2 & A_1 & A_0 & 0 & 0 & \cdots \\ B_{40} & B_{41} & A_3 & A_2 & A_1 & A_0 & 0 & \cdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \end{pmatrix}.$$

Markov chains with this structure occur when the states are grouped into levels, similar to those for QBD processes, but now transitions are no longer confined to interlevel and to adjacent neighboring levels; transitions are also permitted from any level to any *lower* level. In this particular case, we perform the analysis using two boundary columns (B_{i0} and $B_{i1}, i = 0, 1, 2, \ldots$). Certain applications give rise to more than two boundary columns, which may necessitate a restructuring of the matrix. This is considered at the end of this section.

As always, our objective is to compute the stationary probability vector π from the system of equations $\pi Q = 0$. As for QBD processes, let π be partitioned conformally with Q, i.e. $\pi = (\pi_0, \pi_1 \pi_2, \ldots)$ where $\pi_i = (\pi(i,1)\pi(i,2), \ldots \pi(i,K))$ for $i = 0, 1, \ldots$, and $\pi(i,k)$ is the probability of finding the Markov chain in state (i,k) at statistical equilibrium. Neuts has shown that there exists a matrix geometric solution to this problem and that it mirrors that of a QBD process, specifically that there exists a positive matrix R such that

$$\pi_i = \pi_{i-1} R \quad \text{for } i = 2, 3, \ldots,$$

i.e., that

$$\pi_i = \pi_1 R^{i-1} \quad \text{for } i = 2, 3, \ldots.$$

Observe that from $\pi Q = 0$ we have

$$\sum_{k=0}^{\infty} \pi_{k+j} A_k = 0, \qquad j = 1, 2, \ldots,$$

and, in particular,

$$\pi_1 A_0 + \pi_2 A_1 + \sum_{k=2}^{\infty} \pi_{k+1} A_k = 0.$$

Substituting $\pi_i = \pi_1 R^{i-1}$

$$\pi_1 A_0 + \pi_1 R A_1 + \sum_{k=2}^{\infty} \pi_1 R^k A_k = 0$$

or

$$\pi_1 \left(A_0 + R A_1 + \sum_{k=2}^{\infty} R^k A_k \right) = 0$$

provides the following relation from which the matrix R may be computed:

$$A_0 + R A_1 + \sum_{k=2}^{\infty} R^k A_k = 0. \tag{10.34}$$

Notice that Equation (10.34) reduces to Equation (10.29) when $A_k = 0$ for $k > 2$. Rearranging Equation (10.34), we find

$$R = -A_0 A_1^{-1} - \sum_{k=2}^{\infty} R^k A_k A_1^{-1},$$

which leads to the iterative procedure

$$R_{(0)} = 0, \quad R_{(l+1)} = -A_0 A_1^{-1} - \sum_{k=2}^{\infty} R_{(l)}^k A_k A_1^{-1}, \;\; l = 0, 1, 2, \ldots,$$

which is, as Neuts has shown, nondecreasing and converges to the matrix R. In many cases, the structure of the infinitesimal generator is such that the blocks A_i are zero for relatively small values of i, which limits the computational effort needed in each iteration. As before, the number of iterations needed for convergence is frequently large, and now the extremely efficient logarithmic reduction algorithm is no longer applicable—it is designed for QBD processes only. However, more efficient but also more complex algorithms have been developed and may be found in the current literature [3, 36, 43, 44].

We now turn to the derivation of the initial subvectors π_0 and π_1. From the first equation of $\pi Q = 0$, we have

$$\sum_{i=0}^{\infty} \pi_i B_{i0} = 0$$

and we may write

$$\pi_0 B_{00} + \sum_{i=1}^{\infty} \pi_i B_{i0} = \pi_0 B_{00} + \sum_{i=1}^{\infty} \pi_1 R^{i-1} B_{i0} = \pi_0 B_{00} + \pi_1 \left(\sum_{i=1}^{\infty} R^{i-1} B_{i0} \right) = 0, \qquad (10.35)$$

while from the second equation of $\pi Q = 0$,

$$\pi_0 B_{01} + \sum_{i=1}^{\infty} \pi_i B_{i1} = 0, \quad \text{i.e.,} \quad \pi_0 B_{01} + \pi_1 \sum_{i=1}^{\infty} R^{i-1} B_{i1} = 0. \qquad (10.36)$$

Putting Equations (10.35) and (10.36) together in matrix form, we see that we can compute π_0 and π_1 from

$$(\pi_0, \pi_1) \begin{pmatrix} B_{00} & B_{01} \\ \sum_{i=1}^{\infty} R^{i-1} B_{i0} & \sum_{i=1}^{\infty} R^{i-1} B_{i1} \end{pmatrix} = (0, 0).$$

The computed values of π_0 and π_1 must now be normalized by dividing them by

$$\alpha = \pi_0 e + \pi_1 \left(\sum_{i=1}^{\infty} R^{i-1} \right) e = \pi_0 e + \pi_1 (I - R)^{-1} e.$$

Example 10.28 We shall apply the matrix geometric method to the Markov chain of Example 10.27 now modified so that it incorporates additional transitions ($\xi_1 = .25$ and $\xi_2 = .75$) to lower non-neighboring states, Figure 10.2.

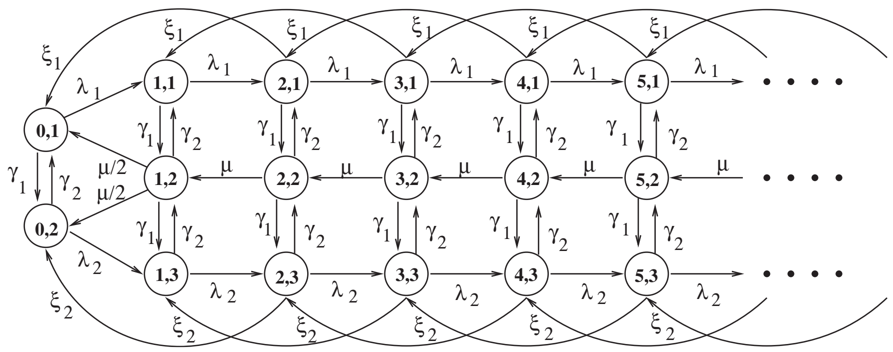

Figure 10.2. State transition diagram for a *G/M/*1-type process.

$$Q = \left(\begin{array}{cc|ccc|ccc|ccc|cc|c}
-6 & 5.0 & 1 & & & & & & & & & & & \\
3 & -3.5 & & & .5 & & & & & & & & & \\
\hline
 & & -6 & 5 & & 1 & & & & & & & & \\
2 & 2 & 3 & -12 & 5.0 & & & & & & & & & \\
 & & & 3 & -3.5 & & & .5 & & & & & & \\
\hline
.25 & & & & & -6.25 & 5 & & 1 & & & & & \\
 & & & 4 & & 3.00 & -12 & 5.00 & & & & & & \\
 & .75 & & & & & 3 & -4.25 & & & .5 & & & \\
\hline
 & & .25 & & & & & & -6.25 & 5 & & 1 & & \\
 & & & & & & 4 & & 3.00 & -12 & 5.00 & & & \\
 & & & & .75 & & & & & 3 & -4.25 & & .5 & \\
\hline
 & & & & & \ddots & & & & \ddots & & \ddots & & \ddots
\end{array}\right).$$

The computation of the matrix R proceeds as previously: The inverse of A_1 is now

$$A_1^{-1} = \begin{pmatrix} -.2233 & -.1318 & -.1550 \\ -.0791 & -.1647 & -.1938 \\ -.0558 & -.1163 & -.3721 \end{pmatrix},$$

which allows us to compute

$$A_0 A_1^{-1} = \begin{pmatrix} -.2233 & -.1318 & -.1550 \\ 0 & 0 & 0 \\ -.0279 & -.0581 & -.1860 \end{pmatrix}, \quad A_2 A_1^{-1} = \begin{pmatrix} 0 & 0 & 0 \\ -.3163 & -.6589 & -.7752 \\ 0 & 0 & 0 \end{pmatrix},$$

$$A_3 A_1^{-1} = \begin{pmatrix} -.0558 & -.0329 & -.0388 \\ 0 & 0 & 0 \\ -.0419 & -.0872 & -.2791 \end{pmatrix}.$$

The iterative process is

$$R_{(k+1)} = \begin{pmatrix} .2233 & .1318 & .1550 \\ 0 & 0 & 0 \\ .0279 & .0581 & .1860 \end{pmatrix} + R_{(k)}^2 \begin{pmatrix} 0 & 0 & 0 \\ .3163 & .6589 & .7752 \\ 0 & 0 & 0 \end{pmatrix} + R_{(k)}^3 \begin{pmatrix} .0558 & .0329 & .0388 \\ 0 & 0 & 0 \\ .0419 & .0872 & .2791 \end{pmatrix},$$

and iterating successively, beginning with $R_{(0)} = 0$, we find

$$R_{(1)} = \begin{pmatrix} .2233 & .1318 & .1550 \\ 0 & 0 & 0 \\ .0279 & .0581 & .1860 \end{pmatrix}, \quad R_{(2)} = \begin{pmatrix} .2370 & .1593 & .1910 \\ 0 & 0 & 0 \\ .0331 & .0686 & .1999 \end{pmatrix},$$

$$R_{(3)} = \begin{pmatrix} .2415 & .1684 & .2031 \\ 0 & 0 & 0 \\ .0347 & .0719 & .2043 \end{pmatrix}, \quad \ldots .$$

After 27 iterations, successive differences are smaller than 10^{-12}, at which point

$$R_{(27)} = \begin{pmatrix} .2440 & .1734 & .2100 \\ 0 & 0 & 0 \\ .0356 & .0736 & .1669 \end{pmatrix}.$$

The boundary conditions are now

$$(\pi_0, \pi_1) \begin{pmatrix} B_{00} & B_{01} \\ B_{10} + RB_{20} & B_{11} + RB_{21} + R^2 B_{31} \end{pmatrix} = (0,\ 0),$$

i.e.,

$$= (\pi_0, \pi_1) \left(\begin{array}{rr|rrr} -6.0 & 5.0 & 1 & 0 & 0 \\ 3.0 & -3.5 & 0 & 0 & .5 \\ \hline .0610 & .1575 & -5.9832 & 5.6938 & .0710 \\ 2.0000 & 2.000 & 3.000 & -12.0000 & 5.0000 \\ .0089 & .1555 & .0040 & 3.2945 & -3.4624 \end{array}\right) = (0,\ 0).$$

As before, we solve this by replacing the last equation with the equation $\pi_{0_1} = 1$. The system of equation becomes

$$(\pi_0, \pi_1) \left(\begin{array}{rr|rrr} -6.0 & 5.0 & 1 & 0 & 1 \\ 3.0 & -3.5 & 0 & 0 & 0 \\ \hline .0610 & .1575 & -5.9832 & 5.6938 & 0 \\ 2.0000 & 2.000 & 3.000 & -12.0000 & 0 \\ .0089 & .1555 & .0040 & 3.2945 & 0 \end{array}\right) = (0, 0 \mid 0, 0, 1)$$

with solution

$$(\pi_0, \pi_1) = (1.0,\ 1.7169 \mid .3730,\ .4095,\ .8470).$$

The normalization constant is

$$\begin{aligned}\alpha &= \pi_0 e + \pi_1 (I - R)^{-1} e \\ &= (1.0,\ 1.7169)e + (.3730,\ .4095,\ .8470) \begin{pmatrix} 1.3395 & .2584 & .3546 \\ 0 & 1 & 0 \\ .0600 & .1044 & 1.2764 \end{pmatrix} e \\ &= 2.7169 + 2.3582 = 5.0751,\end{aligned}$$

which allows us to compute

$$\pi_0/\alpha = (.1970,\ .3383) \quad \text{and} \quad \pi_1/\alpha = (.0735,\ .0807,\ .1669).$$

Successive subcomponents of the stationary distribution are now computed from $\pi_k = \pi_{k-1}R$. For example,

$$\pi_2 = \pi_1 R = (.0735,\ .0807,\ .1669) \begin{pmatrix} .2440 & .1734 & .2100 \\ 0 & 0 & 0 \\ .0356 & .0736 & .1669 \end{pmatrix} = (.0239,\ .0250,\ .0499)$$

and

$$\pi_3 = \pi_2 R = (.0239,\ .0250,\ .0499) \begin{pmatrix} .2440 & .1734 & .2100 \\ 0 & 0 & 0 \\ .0356 & .0736 & .1669 \end{pmatrix} = (.0076,\ .0078,\ .0135),$$

and so on.

Some simplifications occur when the initial B blocks have the same dimensions as the A blocks and when the infinitesimal generator can be written in the following commonly occurring form:

$$Q = \begin{pmatrix} B_{00} & A_0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ B_{10} & A_1 & A_0 & 0 & 0 & 0 & 0 & \cdots \\ B_{20} & A_2 & A_1 & A_0 & 0 & 0 & 0 & \cdots \\ B_{30} & A_3 & A_2 & A_1 & A_0 & 0 & 0 & \cdots \\ B_{40} & A_4 & A_3 & A_2 & A_1 & A_0 & 0 & \cdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots \end{pmatrix}.$$

In this case,

$$\pi_i = \pi_0 R^i \quad \text{for } i = 1, 2, \ldots.$$

Furthermore, $\sum_{i=0}^{\infty} R^i B_{i0}$ is an infinitesimal generator and the subvector π_0 is the stationary probability vector of $\sum_{i=0}^{\infty} R^i B_{i0}$ normalized so that $\pi_0(I - R)^{-1}e = 1$.

In some applications, such as queueing systems with bulk arrivals, more than two boundary columns can occur. Consider, for example the generator matrix

$$Q = \begin{pmatrix} B_{00} & B_{01} & B_{02} & A_0 & & & & & & & & \\ B_{10} & B_{11} & B_{12} & A_1 & A_0 & & & & & & & \\ B_{20} & B_{21} & B_{22} & A_2 & A_1 & A_0 & & & & & & \\ B_{30} & B_{31} & B_{32} & A_3 & A_2 & A_1 & A_0 & & & & & \\ B_{40} & B_{41} & B_{42} & A_4 & A_3 & A_2 & A_1 & A_0 & & & & \\ B_{50} & B_{51} & B_{52} & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & & & \\ B_{60} & B_{61} & B_{62} & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & & \\ B_{70} & B_{71} & B_{72} & A_7 & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & \\ B_{80} & B_{81} & B_{82} & A_8 & A_7 & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 \\ \vdots & \vdots & \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix}.$$

At present, this matrix is *not* block lower Hessenberg. However, if it is restructured into the form

$$\overline{Q} = \begin{pmatrix} \overline{B_{00}} & \overline{A_0} & 0 & 0 & \cdots \\ \overline{B_{10}} & \overline{A_1} & \overline{A_0} & 0 & \cdots \\ \overline{B_{20}} & \overline{A_2} & \overline{A_1} & \overline{A_0} & \\ \vdots & & \ddots & \ddots & \ddots \end{pmatrix}$$

$$= \left(\begin{array}{ccc|ccc|ccc|ccc|c} B_{00} & B_{01} & B_{02} & A_0 & & & & & & & & & \\ B_{10} & B_{11} & B_{12} & A_1 & A_0 & & & & & & & & \\ B_{20} & B_{21} & B_{22} & A_2 & A_1 & A_0 & & & & & & & \\ \hline B_{30} & B_{31} & B_{32} & A_3 & A_2 & A_1 & A_0 & & & & & & \\ B_{40} & B_{41} & B_{42} & A_4 & A_3 & A_2 & A_1 & A_0 & & & & & \\ B_{50} & B_{51} & B_{52} & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & & & & \\ \hline B_{60} & B_{61} & B_{62} & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & & & \\ B_{70} & B_{71} & B_{72} & A_7 & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & & \\ B_{80} & B_{81} & B_{82} & A_8 & A_7 & A_6 & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{array}\right)$$

with

$$\overline{A_0} = \begin{pmatrix} A_0 & & \\ A_1 & A_0 & \\ A_2 & A_1 & A_0 \end{pmatrix}, \quad \overline{A_1} = \begin{pmatrix} A_3 & A_2 & A_1 \\ A_4 & A_3 & A_2 \\ A_5 & A_4 & A_3 \end{pmatrix}, \quad \overline{B_{00}} = \begin{pmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{pmatrix}, \quad \ldots$$

then the matrix $\overline{Q}$ does have the desired block lower Hessenberg form and hence the techniques described in this section may be successfully applied to it.

In the case of discrete-time Markov chains, as opposed to the continuous-time case just outlined, it suffices to replace $-A_1^{-1}$ with $(I - A_1)^{-1}$, as we described for QBD processes.

10.6.3 Block Upper Hessenberg Markov Chains

We now move to block upper Hessenberg Markov chains, also called *M/G/*1-type processes and solved using the *matrix analytic method*. In the past two sections concerning QBD and *G/M/*1-type processes, we posed the problem in terms of continuous-time Markov chains, and mentioned that discrete-time Markov chains can be treated if the matrix inverse A_1^{-1} is replaced with the inverse

$(I - A_1)^{-1}$ where it is understood that A_1 is taken from a stochastic matrix. This time we shall consider the discrete-time case. Specifically, we consider the case when the stochastic transition probability matrix is irreducible and has the structure

$$P = \begin{pmatrix} B_{00} & B_{01} & B_{02} & B_{03} & \cdots & B_{0j} & \cdots \\ B_{10} & A_1 & A_2 & A_3 & \cdots & A_j & \cdots \\ 0 & A_0 & A_1 & A_2 & \cdots & A_{j-1} & \cdots \\ 0 & 0 & A_0 & A_1 & \cdots & A_{j-2} & \cdots \\ 0 & 0 & 0 & A_0 & \cdots & A_{j-3} & \cdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \end{pmatrix},$$

in which all submatrices A_j, $j = 0, 1, 2, \ldots$ are square and of order K, and B_{00} is square but not necessarily of order K. Be aware that whereas previously the block A_0 was the rightmost nonzero block of our global matrix, this time it is the leftmost nonzero block. Notice that the matrix $A = \sum_{i=0}^{\infty} A_i$ is a stochastic matrix. We shall further assume that A is irreducible, the commonly observed case in practice. Instances in which A is not irreducible are treated by Neuts.

$$\pi_A A = \pi_A \quad \text{and} \quad \pi_A e = 1.$$

The Markov chain P is known to be positive recurrent if the following condition holds:

$$\pi_A \left(\sum_{i=1}^{\infty} i A_i \, e \right) \equiv \pi_A \, b \ < 1. \tag{10.37}$$

Our objective is the computation of the stationary probability vector π from the system of equations $\pi P = \pi$. As before, we partition π conformally with P, i.e.,

$$\pi = (\pi_0, \pi_1, \pi_2, \ldots),$$

where

$$\pi_i = (\pi(i, 1), \pi(i, 2), \ldots, \pi(i, K))$$

for $i = 0, 1, \ldots$ and $\pi(i, k)$ is the probability of finding the system in state (i, k) at statistical equilibrium. The analysis of *M/G*/1-type processes is more complicated than that of QBD or *G/M*/1-type processes because the subvectors π_i no longer have a matrix geometric relationship with one another.

The key to solving upper block Hessenberg structured Markov chains is the computation of a certain matrix G which is stochastic if the Markov chain is recurrent, which we assume to be the case. This is the same matrix G that appears in the logarithmic reduction algorithm and has an important probabilistic interpretation. The element G_{ij} of this matrix is the conditional probability that starting in state i of any level $n \geq 2$, the process enters level $n - 1$ for the first time by arriving at state j of that level. This matrix satisfies the fixed point equation

$$G = \sum_{i=0}^{\infty} A_i G^i$$

and is indeed the minimal non-negative solution of

$$X = \sum_{i=0}^{\infty} A_i X^i.$$

It can be found by means of the iteration

$$G_{(0)} = 0, \quad G_{(k+1)} = \sum_{i=0}^{\infty} A_i G_{(k)}^i, \quad k = 0, 1, \ldots.$$

Once the matrix G has been computed, then successive components of π can be obtained from a relationship, called Ramaswami's formula, which we now develop. We follow the algebraic approach of Bini and Meini [3, 36], rather than the original probabilistic approach of Ramaswami. We begin by writing the system of equations $\pi P = \pi$ as $\pi(I - P) = 0$, i.e.,

$$(\pi_0,\ \pi_1,\ \ldots,\ \pi_j,\ \ldots)\left(\begin{array}{c|cccccc} I - B_{00} & -B_{01} & -B_{02} & -B_{03} & \cdots & -B_{0j} & \cdots \\ \hline -B_{10} & I - A_1 & -A_2 & -A_3 & \cdots & -A_j & \cdots \\ 0 & -A_0 & I - A_1 & -A_2 & \cdots & -A_{j-1} & \cdots \\ 0 & 0 & -A_0 & I - A_1 & \cdots & -A_{j-2} & \cdots \\ 0 & 0 & 0 & -A_0 & \cdots & -A_{j-3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array}\right)$$
$$= (0,\ 0,\ \ldots,\ 0,\ \ldots). \tag{10.38}$$

The submatrix in the lower right is block Toeplitz. Bini and Meini have shown that there exists a decomposition of this Toeplitz matrix into a block upper triangular matrix U and block lower triangular matrix L with

$$U = \begin{pmatrix} A_1^* & A_2^* & A_3^* & A_4^* & \cdots \\ 0 & A_1^* & A_2^* & A_3^* & \cdots \\ 0 & 0 & A_1^* & A_2^* & \cdots \\ 0 & 0 & 0 & A_1^* & \cdots \\ \vdots & \vdots & \vdots & & \ddots \end{pmatrix} \quad \text{and} \quad L = \begin{pmatrix} I & 0 & 0 & 0 & \cdots \\ -G & I & 0 & 0 & \cdots \\ 0 & -G & I & 0 & \cdots \\ 0 & 0 & -G & I & \cdots \\ \vdots & \vdots & & \ddots & \ddots \end{pmatrix}.$$

We denote the nonzero blocks of U as A_i^* rather than U_i since we shall see later that these blocks are formed using the A_i blocks of P. Once the matrix G has been formed then L is known. Observe that the inverse of L can be written in terms of the powers of G. For example,

$$\begin{pmatrix} I & 0 & 0 & 0 & \cdots \\ -G & I & 0 & 0 & \cdots \\ 0 & -G & I & 0 & \cdots \\ 0 & 0 & -G & I & \cdots \\ \vdots & \vdots & & \ddots & \ddots \end{pmatrix} \begin{pmatrix} I & 0 & 0 & 0 & \cdots \\ G & I & 0 & 0 & \cdots \\ G^2 & G & I & 0 & \cdots \\ G^3 & G^2 & G & I & \cdots \\ \vdots & \vdots & & \ddots & \ddots \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots \\ 0 & 1 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & \cdots \\ 0 & 0 & 0 & 1 & \cdots \\ \vdots & \vdots & \vdots & & \ddots \end{pmatrix}.$$

From Equation (10.38), we have

$$(\pi_0,\ \pi_1,\ \ldots,\ \pi_j,\ \ldots)\left(\begin{array}{c|c} I - B_{00} & -B_{01}\ -B_{02}\ -B_{03}\ \cdots\ -B_{0j}\ \cdots \\ \hline -B_{10} & \\ 0 & \\ 0 & UL \\ 0 & \\ \vdots & \end{array}\right) = (0,\ 0,\ \ldots,\ 0,\ \ldots),$$

which allows us to write

$$\pi_0(-B_{01},\ -B_{02},\ \ldots) + (\pi_1,\ \pi_2,\ \ldots)UL = 0$$

or

$$\pi_0(B_{01},\ B_{02},\ \ldots)L^{-1} = (\pi_1,\ \pi_2,\ \ldots)U,$$

i.e.,

$$\pi_0\,(B_{01},\ B_{02},\ \ldots)\begin{pmatrix} I & 0 & 0 & 0 & \cdots \\ G & I & 0 & 0 & \cdots \\ G^2 & G & I & 0 & \cdots \\ G^3 & G^2 & G & I & \cdots \\ \vdots & \vdots & & \ddots & \ddots \end{pmatrix} = (\pi_1,\ \pi_2,\ \ldots)\,U.$$

Forming the product of $(B_{01},\ B_{02},\ \ldots)$ and L^{-1} leads to the important result

$$\pi_0\left(B^*_{01},\ B^*_{02},\ \ldots\right) = (\pi_1,\ \pi_2,\ \ldots)\,U, \tag{10.39}$$

where

$$\begin{aligned} B^*_{01} &= B_{01} + B_{02}G + B_{03}G^2 + \cdots \ = \sum_{k=1}^{\infty} B_{0k}G^{k-1}, \\ B^*_{02} &= B_{02} + B_{03}G + B_{04}G^2 + \cdots \ = \sum_{k=2}^{\infty} B_{0k}G^{k-2}, \\ &\vdots \\ B^*_{0i} &= B_{0i} + B_{0,i+1}G + B_{0,i+2}G^2 + \cdots \ = \sum_{k=i}^{\infty} B_{0k}G^{k-i}. \end{aligned}$$

The system of equations (10.39) will allow us to compute the successive components of the vector π once the initial component π_0 and the matrix U are known. To see this, write Equation (10.39) as

$$\pi_0\left(B^*_{01},\ B^*_{02},\ \cdots\right) = (\pi_1,\ \pi_2,\ \cdots)\begin{pmatrix} A^*_1 & A^*_2 & A^*_3 & A^*_4 & \cdots \\ 0 & A^*_1 & A^*_2 & A^*_3 & \cdots \\ 0 & 0 & A^*_1 & A^*_2 & \cdots \\ 0 & 0 & 0 & A^*_1 & \cdots \\ \vdots & \vdots & \vdots & & \ddots \end{pmatrix}$$

and observe that

$$\begin{aligned} \pi_0 B^*_{01} &= \pi_1 A^*_1 &\implies&\ \pi_1 = \pi_0 B^*_{01} A^{*-1}_1, \\ \pi_0 B^*_{02} &= \pi_1 A^*_2 + \pi_2 A^*_1 &\implies&\ \pi_2 = \pi_0 B^*_{02} A^{*-1}_1 - \pi_1 A^*_2 A^{*-1}_1, \\ \pi_0 B^*_{03} &= \pi_1 A^*_3 + \pi_2 A^*_2 + \pi_3 A^*_1 &\implies&\ \pi_3 = \pi_0 B^*_{03} A^{*-1}_1 - \pi_1 A^*_3 A^{*-1}_1 - \pi_2 A^*_2 A^{*-1}_1, \\ & & \vdots & \end{aligned}$$

In general, we find

$$\begin{aligned} \pi_i &= \left(\pi_0 B^*_{0i} - \pi_1 A^*_i - \pi_2 A^*_{i-1} - \cdots - \pi_{i-1} A^*_2\right) A^{*-1}_1 \\ &= \left(\pi_0 B^*_{0i} - \sum_{k=1}^{i-1} \pi_k A^*_{i-k+1}\right) A^{*-1}_1, \quad i = 1, 2, \ldots . \end{aligned}$$

To compute the first subvector π_0 we return to

$$\pi_0\left(B^*_{01},\ B^*_{02},\ \ldots\right) = (\pi_1,\ \pi_2,\ \ldots)\,U$$

and write it as

$$(\pi_0,\ \pi_1,\ \ldots,\ \pi_j,\ \ldots)\left(\begin{array}{c|cccccc} I-B_{00} & -B_{01}^* & -B_{02}^* & -B_{03}^* & \cdots & -B_{0j}^* & \cdots \\ \hline -B_{10} & A_1^* & A_2^* & A_3^* & \cdots & A_j^* & \cdots \\ 0 & 0 & A_1^* & A_2^* & \cdots & A_{j-1}^* & \cdots \\ 0 & 0 & 0 & A_1^* & \cdots & A_{j-2}^* & \cdots \\ 0 & 0 & 0 & 0 & \cdots & \vdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array}\right) = (0,\ 0,\ \ldots,\ 0,\ \ldots).$$

From the first two equations, we have

$$\pi_0(I-B_{00}) - \pi_1 B_{10} = 0$$

and

$$-\pi_0 B_{01}^* + \pi_1 A_1^* = 0.$$

This latter gives

$$\pi_1 = \pi_0 B_{01}^* A_1^{*-1}$$

which when substituted into the first gives

$$\pi_0(I-B_{00}) - \pi_0 B_{01}^* A_1^{*-1} B_{10} = 0$$

or

$$\pi_0\left(I - B_{00} - B_{01}^* A_1^{*-1} B_{10}\right) = 0,$$

from which we may now compute π_0, but correct only to a multiplicative constant. It must be normalized so that $\sum_{i=0}^{\infty}\pi_i = 1$ which may be accomplished by enforcing the condition

$$\pi_0 e + \pi_0\left(\sum_{i=1}^{\infty} B_{0i}^*\right)\left(\sum_{i=1}^{\infty} A_i^*\right)^{-1} e = 1. \tag{10.40}$$

We now turn our attention to the computation of the matrix U. Since

$$UL = \begin{pmatrix} I-A_1 & -A_2 & -A_3 & \cdots & -A_j & \cdots \\ -A_0 & I-A_1 & -A_2 & \cdots & -A_{j-1} & \cdots \\ 0 & -A_0 & I-A_1 & \cdots & -A_{j-2} & \cdots \\ 0 & 0 & -A_0 & \cdots & -A_{j-3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix},$$

we have

$$\begin{pmatrix} A_1^* & A_2^* & A_3^* & A_4^* & \cdots \\ 0 & A_1^* & A_2^* & A_3^* & \cdots \\ 0 & 0 & A_1^* & A_2^* & \cdots \\ 0 & 0 & 0 & A_1^* & \cdots \\ \vdots & \vdots & \vdots & & \ddots \end{pmatrix} = \begin{pmatrix} I-A_1 & -A_2 & -A_3 & \cdots & -A_j & \cdots \\ -A_0 & I-A_1 & -A_2 & \cdots & -A_{j-1} & \cdots \\ 0 & -A_0 & I-A_1 & \cdots & -A_{j-2} & \cdots \\ 0 & 0 & -A_0 & \cdots & -A_{j-3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}\begin{pmatrix} I & 0 & 0 & 0 & \cdots \\ G & I & 0 & 0 & \cdots \\ G^2 & G & I & 0 & \cdots \\ G^3 & G^2 & G & I & \cdots \\ \vdots & \vdots & & \ddots & \ddots \end{pmatrix},$$

and it is now apparent that

$$A_1^* = I - A_1 - A_2G - A_3G^2 - A_4G^3 - \cdots = I - \sum_{k=1}^{\infty} A_k G^{k-1},$$

$$A_2^* = -A_2 - A_3G - A_4G^2 - A_5G^3 - \cdots = -\sum_{k=2}^{\infty} A_k G^{k-2},$$

$$A_3^* = -A_3 - A_4G - A_5G^2 - A_6G^3 - \cdots = -\sum_{k=3}^{\infty} A_k G^{k-3},$$

$$\vdots$$

$$A_i^* = -A_i - A_{i+1}G - A_{i+2}G^2 - A_{i+3}G^3 - \cdots = -\sum_{k=i}^{\infty} A_k G^{k-i}, \quad i \geq 2.$$

We now have all the results we need. The basic algorithm is

- Construct the matrix G.
- Obtain π_0 by solving the system of equations $\pi_0 \left(I - B_{00} - B_{01}^* A_1^{*-1} B_{10} \right) = 0$, subject to the normalizing condition, Equation (10.40).
- Compute π_1 from $\pi_1 = \pi_0 B_{01}^* A_1^{*-1}$.
- Find all other required π_i from $\pi_i = \left(\pi_0 B_{0i}^* - \sum_{k=1}^{i-1} \pi_k A_{i-k+1}^* \right) A_1^{*-1}$,

where

$$B_{0i}^* = \sum_{k=i}^{\infty} B_{0k} G^{k-i}, \quad i \geq 1; \quad A_1^* = I - \sum_{k=1}^{\infty} A_k G^{k-1} \quad \text{and} \quad A_i^* = -\sum_{k=i}^{\infty} A_k G^{k-i}, \quad i \geq 2.$$

This obviously gives rise to a number of computational questions. The first is the actual computation of the matrix G. We mentioned previously that it can be obtained from its defining equation by means of the iterative procedure

$$G_{(0)} = 0, \quad G_{(k+1)} = \sum_{i=0}^{\infty} A_i G_{(k)}^i, \quad k = 0, 1, \ldots.$$

However, this is rather slow. Neuts proposed a variant that converges faster, namely,

$$G_{(0)} = 0; \quad G_{(k+1)} = (I - A_1)^{-1} \left(A_0 + \sum_{i=2}^{\infty} A_i G_{(k)}^i \right), \quad k = 0, 1, \ldots$$

Among fixed point iterations such as these, the following suggested by Bini and Meini [3, 36], has the fastest convergence:

$$G_{(0)} = 0, \quad G_{(k+1)} = \left(I - \sum_{i=1}^{\infty} A_i G_{(k)}^{i-1} \right)^{-1} A_0, \quad k = 0, 1, \ldots.$$

Nevertheless, fixed point iterations can be very slow in certain instances. More advanced techniques based on cyclic reduction have been developed and converge much faster.

The second major problem is the computation of the infinite summations that appear in the formulae. Frequently the structure of the matrix is such that A_k and B_k are zero for relatively small (single-digit integer) values of k, so that forming these summations is not too onerous. In all cases, the fact that $\sum_{k=0}^{\infty} A_k$ and $\sum_{k=0}^{\infty} B_k$ are stochastic implies that the matrices A_k and B_k are negligibly small for large values of k and can be set to zero once k exceeds some threshold k_M. In this case, we take $\sum_{k=0}^{k_M} A_k$ and $\sum_{k=0}^{k_M} B_k$ to be stochastic. More precisely, a nonnegative matrix P is said to be *numerically stochastic* if $Pe < \mu e$ where μ is the precision of the computer. When k_M is not small, finite summations of the type $\sum_{k=i}^{k_M} A_k G^{k-i}$ should be evaluated using Horner's rule. For example, if $i = 1$ and $k_M = 5$:

$$A_1^* = \sum_{k=1}^{5} A_k G^{k-1} = A_1 + A_2 G + A_3 G^2 + A_4 G^3 + A_5 G^4$$

should be evaluated from the innermost parentheses outward as

$$A_1^* = A_1 + (A_2 + (A_3 + (A_4 + A_5 G)G)G)G.$$

Example 10.29 We shall apply the matrix analytic method to the Markov chain of Example 10.27, now modified so that it incorporates additional transitions ($\zeta_1 = 1/48$ and $\zeta_2 = 1/16$) to higher-numbered non-neighboring states. The state transition diagram is shown in Figure 10.3.

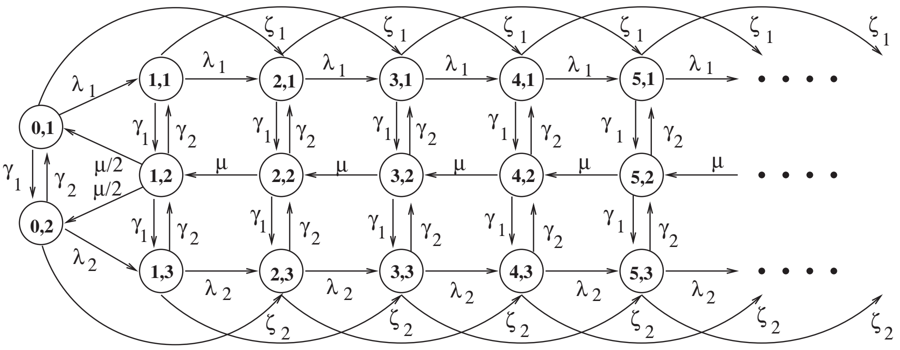

Figure 10.3. State transition diagram for an *M/G/*1-type process.

$$P = \left(\begin{array}{cc|ccc|ccc|ccc|c}
23/48 & 5/12 & 1/12 & & & 1/48 & & & & & & \\
1/4 & 31/48 & & & 1/24 & & & 1/16 & & & & \\ \hline
 & & 23/48 & 5/12 & & 1/12 & & & 1/48 & & & \\
1/3 & 1/3 & 1/4 & & 1/12 & & & & & & & \\
 & & & 1/4 & 31/48 & & & 1/24 & & & 1/16 & \\ \hline
 & & & & & 23/48 & 5/12 & & 1/12 & & & \\
 & & & 2/3 & & 1/4 & & 1/12 & & & & \\
 & & & & & & 1/4 & 31/48 & & & 1/24 & \\ \hline
 & & & & & & & & 23/48 & 5/12 & & \\
 & & & & & & 2/3 & & 1/4 & & 1/12 & \\
 & & & & & & & & & 1/4 & 31/48 & \\ \hline
 & & & & & \ddots & & & \ddots & & & \ddots
\end{array}\right).$$

We have the following block matrices:

$$A_0 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 2/3 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad A_1 = \begin{pmatrix} 23/48 & 5/12 & 0 \\ 1/4 & 0 & 1/12 \\ 0 & 1/4 & 31/48 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 1/12 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1/24 \end{pmatrix},$$

$$A_3 = \begin{pmatrix} 1/48 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1/16 \end{pmatrix}, \quad B_{00} = \begin{pmatrix} 23/48 & 5/12 \\ 1/4 & 31/48 \end{pmatrix}, \quad B_{01} = \begin{pmatrix} 1/12 & 0 & 0 \\ 0 & 0 & 1/24 \end{pmatrix},$$

$$B_{02} = \begin{pmatrix} 1/48 & 0 & 0 \\ 0 & 0 & 1/16 \end{pmatrix}, \quad \text{and} \quad B_{10} = \begin{pmatrix} 0 & 0 \\ 1/3 & 1/3 \\ 0 & 0 \end{pmatrix}.$$

First, using Equation (10.37), we verify that the Markov chain with transition probability matrix P is positive recurrent,

$$A = A_0 + A_1 + A_2 + A_3 = \begin{pmatrix} .583333 & .416667 & 0 \\ .250000 & .666667 & .083333 \\ 0 & .250000 & .750000 \end{pmatrix},$$

from which we can compute

$$\pi_A = (.310345,\ .517241,\ .172414).$$

Also

$$b = (A_1 + 2A_2 + 3A_3)e = \begin{pmatrix} .708333 & .416667 & 0 \\ .250000 & 0 & .083333 \\ 0 & .250000 & .916667 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.125000 \\ 0.333333 \\ 1.166667 \end{pmatrix}.$$

Since

$$\pi_A\, b = (.310345,\ .517241,\ .172414) \begin{pmatrix} 1.125000 \\ 0.333333 \\ 1.166667 \end{pmatrix} = .722701 < 1$$

the Markov chain is positive recurrent.

Let us consider the computation of the matrix G. As we noted previously, the ij element of G is the conditional probability that starting in state i of any level $n \geq 2$, the process enters level $n-1$ for the first time by arriving at state j of that level. What this means for this particular example is that the elements in column 2 of G must all be equal to 1 and all other elements must be zero, since the only transitions from any level n to level $n-1$ are from and to the second element. Nevertheless, it is interesting to see how each of the three different fixed point formula given for computing G actually perform on this example. The fact that we know the answer in advance will help in this comparison. We take the initial value $G_{(0)}$ to be zero.

Formula 1: $G_{(k+1)} = \sum_{i=0}^{\infty} A_i G_{(k)}^i, \quad k = 0, 1, \ldots,$

$$G_{(k+1)} = A_0 + A_1 G_{(k)} + A_2 G_{(k)}^2 + A_3 G_{(k)}^3.$$

After ten iterations, the computed matrix is

$$G_{(10)} = \begin{pmatrix} 0 & .867394 & 0 \\ 0 & .937152 & 0 \\ 0 & .766886 & 0 \end{pmatrix}.$$

Formula 2: $G_{(k+1)} = (I - A_1)^{-1} \left(A_0 + \sum_{i=2}^{\infty} A_i G_{(k)}^i \right), \quad k = 0, 1, \ldots,$

$$G_{(k+1)} = (I - A_1)^{-1} \left(A_0 + A_2 G_{(k)}^2 + A_3 G_{(k)}^3 \right).$$

In this case, after ten iterations, the computed matrix is

$$G_{(10)} = \begin{pmatrix} 0 & .999844 & 0 \\ 0 & .999934 & 0 \\ 0 & .999677 & 0 \end{pmatrix}.$$

Formula 3: $G_{(k+1)} = \left(I - \sum_{i=1}^{\infty} A_i G_{(k)}^{i-1} \right)^{-1} A_0, \quad k = 0, 1, \ldots,$

$$G_{(k+1)} = \left(I - A_1 - A_2 G_{(k)} - A_3 G_{(k)}^2 \right)^{-1} A_0.$$

This is the fastest of the three and after ten iterations we have

$$G_{(10)} = \begin{pmatrix} 0 & .999954 & 0 \\ 0 & .999979 & 0 \\ 0 & .999889 & 0 \end{pmatrix}.$$

We now continue with this example and shall use the exact value of G in the remaining parts of the algorithm. In preparation, we compute the following quantities, using the fact that $A_k = 0$ for $k > 3$ and $B_{0k} = 0$ for $k > 2$:

$$A_1^* = I - \sum_{k=1}^{\infty} A_k G^{k-1} = I - A_1 - A_2 G - A_3 G^2 = \begin{pmatrix} .520833 & -.520833 & 0 \\ -.250000 & 1 & -.083333 \\ 0 & -.354167 & .354167 \end{pmatrix},$$

$$A_2^* = -\sum_{k=2}^{\infty} A_k G^{k-2} = -(A_2 + A_3 G) = \begin{pmatrix} -.083333 & -.020833 & 0 \\ 0 & 0 & 0 \\ 0 & -.062500 & -.041667 \end{pmatrix},$$

$$A_3^* = -\sum_{k=3}^{\infty} A_k G^{k-3} = -A_3 = \begin{pmatrix} -.020833 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -.062500 \end{pmatrix},$$

$$B_{01}^* = \sum_{k=1}^{\infty} B_{0k} G^{k-1} = B_{01} + B_{02} G = \begin{pmatrix} .083333 & .020833 & 0 \\ 0 & .062500 & .041667 \end{pmatrix},$$

$$B_{02}^* = \sum_{k=2}^{\infty} B_{0k} G^{k-2} = B_{02} = \begin{pmatrix} .020833 & 0 & 0 \\ 0 & 0 & .062500 \end{pmatrix},$$

$$A_1^{*-1} = \begin{pmatrix} 2.640 & 1.50 & .352941 \\ .720 & 1.50 & .352941 \\ .720 & 1.50 & 3.176470 \end{pmatrix}.$$

We may now compute the initial subvector π_0 from

$$0 = \pi_0 \left(I - B_{00} - B_{01}^* A_1^{*-1} B_{10} \right) = \pi_0 \begin{pmatrix} .468750 & -.468750 \\ -.302083 & .302083 \end{pmatrix},$$

from which we find

$$\pi_0 = (.541701,\ .840571).$$

We now normalize this so that

$$\pi_0 e + \pi_0 \left(\sum_{i=1}^{\infty} B_{0i}^* \right) \left(\sum_{i=1}^{\infty} A_i^* \right)^{-1} e = 1,$$

i.e.,

$$\pi_0 e + \pi_0 \left(B_{01}^* + B_{02}^* \right) \left(A_1^* + A_2^* + A_3^* \right)^{-1} e = 1.$$

Evaluating,

$$\left(B_{01}^* + B_{02}^*\right)\left(A_1^* + A_2^* + A_3^*\right)^{-1} = \begin{pmatrix} .104167 & .020833 & 0 \\ 0 & .062500 & .104167 \end{pmatrix}$$

$$\times \begin{pmatrix} .416667 & -.541667 & 0 \\ -.250000 & 1 & -.083333 \\ 0 & -.416667 & .250000 \end{pmatrix}^{-1} = \begin{pmatrix} .424870 & .291451 & .097150 \\ .264249 & .440415 & .563472 \end{pmatrix}.$$

Thus

$$(.541701,\ .840571)\begin{pmatrix}1\\1\end{pmatrix} + (.541701,\ .840571)\begin{pmatrix} .424870 & .291451 & .097150 \\ .264249 & .440415 & .563472 \end{pmatrix}\begin{pmatrix}1\\1\\1\end{pmatrix} = 2.888888,$$

which leads to

$$\pi_0 = (.541701,\ .840571)/2.888888 = (.187512,\ .290967)$$

as the initial subvector.
We can now find π_1 from the relationship $\pi_1 = \pi_0 B_{01}^* A_1^{*-1}$. This gives

$$\pi_1 = (.187512,\ .290967)\begin{pmatrix} .083333 & .020833 & 0 \\ 0 & .062500 & .041667 \end{pmatrix}\begin{pmatrix} 2.640 & 1.50 & .352941 \\ .720 & 1.50 & .352941 \\ .720 & 1.50 & 3.176470 \end{pmatrix}$$

$$= (.065888,\ .074762,\ .0518225).$$

Finally, all needed remaining subcomponents of π can be found from

$$\pi_i = \left(\pi_0 B_{0i}^* - \sum_{k=1}^{i-1} \pi_k A_{i-k+1}^*\right) A_1^{*-1}.$$

For example, we have

$$\begin{aligned}
\pi_2 &= \left(\pi_0 B_{02}^* - \pi_1 A_2^*\right) A_1^{*-1} \\
&= (.042777,\ .051530,\ .069569), \\
\pi_3 &= \left(\pi_0 B_{03}^* - \pi_1 A_3^* - \pi_2 A_2^*\right) A_1^{*-1} = \left(-\pi_1 A_3^* - \pi_2 A_2^*\right) A_1^{*-1} \\
&= (.0212261,\ .024471,\ .023088), \\
\pi_4 &= \left(\pi_0 B_{04}^* - \pi_1 A_4^* - \pi_2 A_3^* - \pi_3 A_2^*\right) A_1^{*-1} = \left(-\pi_2 A_3^* - \pi_3 A_2^*\right) A_1^{*-1} \\
&= (.012203,\ .014783,\ .018471), \\
&\ \vdots
\end{aligned}$$

The probability that the Markov chain is in any level i is given by $\|\pi_i\|_1$. The probabilities of this Markov chain being in the first five levels are given as

$$\|\pi_0\|_1 = .478479,\ \|\pi_1\|_1 = .192473,\ \|\pi_2\|_1 = .163876,\ \|\pi_3\|_1 = .068785,\ \|\pi_4\|_1 = .045457.$$

The sum of these five probabilities is 0.949070.

10.7 Transient Distributions

So far, our concern has been with the computation of the stationary distribution π of discrete- and continuous-time Markov chains. We now turn our attention to the computation of state probability distributions at an arbitrary point of time. In the case of a discrete-time Markov chain, this means finding the distribution at some arbitrary time step n. This distribution is denoted $\pi^{(n)}$, a row vector whose ith component is the probability that the Markov chain is in state i at time step n. As we saw previously, it satisfies the relationship

$$\pi^{(n)} = \pi^{(n-1)}P(n-1) = \pi^{(0)}P(0)P(1)\cdots P(n-1),$$

where $P(i)$ is the transition probability matrix at step i. For a homogeneous discrete-time Markov chain, this reduces to

$$\pi^{(n)} = \pi^{(n-1)}P = \pi^{(0)}P^n,$$

where now $P(0) = P(1) = \cdots = P$. In this section, we shall consider only homogeneous Markov chains. For a continuous-time Markov chain with infinitesimal generator Q, we seek the distribution at any time t. Such a distribution is denoted $\pi(t)$, a row vector whose component $\pi_i(t)$ is the probability that the Markov chain is in state i at time t. We have previously seen that this vector satisfies the relationship

$$\pi(t) = \pi(0)e^{Qt},$$

where e^{Qt} is the matrix exponential defined by

$$e^{Qt} = \sum_{k=0}^{\infty}(Qt)^k/k!.$$

In both cases, what is usually required is seldom the probability distribution $\pi^{(n)}$ or $\pi(t)$ itself, but rather some linear combination of the components of these vectors, such as the probability that the Markov chain is in a single state i, ($\pi^{(n)}e_i$ or $\pi(t)\, e_i$, where e_i is a column vector whose elements are all zero except the ith, which is equal to 1) or the probability that the Markov chain is in a subset of states E_i $\left(\sum_{i\in E_i}\pi^{(n)}e_i \text{ or } \sum_{i\in E_i}\pi(t)e_i\right)$ or yet again a weighted sum of the probabilities wherein the unit component of e_i is replaced by an arbitrary scaler. Also, the evolution of this statistic from the initial time to the desired time step n or time t may be required rather than just its (single) value at the final time point.

The computation of transient distributions of discrete-time Markov chains rarely poses major problems. The procedure consists of repeatedly multiplying the probability distribution vector obtained at step $k-1$ with the stochastic transition probability matrix to obtain the probability distribution at step k, for $k = 1, 2, \ldots, n$. If n is large and the number of states in the Markov chain is small (not exceeding several hundreds), then some savings in computation time can be obtained by successively squaring the transition probability matrix j times, where j is the largest integer such that $2^j \leq n$. This gives the matrix P^{2^j} which can now be multiplied by P (and powers of P) until the value P^n is obtained. The distribution at time step n is now found from $\pi^{(n)} = \pi^{(0)}P^n$. If the matrix P is sparse, this sparsity is lost in the computation of P^n so this approach is not appropriate for large sparse Markov chains. Also, if a time trajectory of a statistic of the distribution is needed, this approach may be less than satisfactory, because only distributions at computed values of P^{2^j} will be available. One final point worth noting is that for large values of n, it may be beneficial to maintain a check on convergence to the stationary distribution, since this may occur, correct to some desired computational accuracy, prior to step n. Any additional vector–matrix multiplications after this point will not alter the distribution.

Example 10.30 Let us consider a discrete-time Markov chain with transition probability matrix P given by

$$P = \begin{pmatrix} .4 & 0 & .6 & 0 \\ .0002 & .3 & 0 & .6998 \\ .1999 & .0001 & .8 & 0 \\ 0 & .5 & 0 & .5 \end{pmatrix}.$$

Let us assume that this Markov chain starts in state 1, i.e., $\pi^{(0)} = (1, 0, 0, 0)$, and the value of being in the different states at some time step n is given by the vector $a = (0, 4, 0, 10)^T$. In other words, the Markov chain in states 1 and 3 is worthless, but worth 4 (arbitrary units) in state 2 and 10 in state 4. The value at time step n is $\pi^{(n)}a$ and we now wish to compute this statistic for various values of n.

For small values of $n = 1, 2, \ldots$, it suffices to compute $\pi^{(1)} = \pi^{(0)}P$, $\pi^{(2)} = \pi^{(1)}P$, $\pi^{(3)} = \pi^{(2)}P, \ldots$, and we find

$$\begin{aligned} \pi^{(1)} &= (.4,\ 0,\ .6,\ 0), \\ \pi^{(2)} &= (.27994,\ .00006,\ .7200,\ 0), \\ \pi^{(3)} &= (.255904,\ .00009,\ .743964,\ .000042), \\ \pi^{(4)} &= (.251080,\ .000122,\ .748714,\ .000084), \\ &\ \vdots, \end{aligned}$$

which gives the following values of the statistic $\pi^{(n)}a$:

$$0, \quad .00024, \quad .000780, \quad .001329, \ \ldots.$$

These values can be plotted to show the evolution of the statistic over a number of time steps.

If the transient distribution is required at much greater values of n, then advantage should be taken of the small size of the matrix to compute successive powers of the matrix. For example, if the distribution is required at time step 1000, then P^{1000} can be found from powering the matrix P. If, in addition, we are asked to plot the behavior of the statistic, until that time, we may wish to compute

$$\begin{aligned} \pi^{(100)} &= (.248093,\ .003099,\ .744507,\ .004251), \\ \pi^{(200)} &= (.246263,\ .006151,\ .739062,\ .008524), \\ \pi^{(300)} &= (.244461,\ .009156,\ .733652,\ .012731), \\ \pi^{(400)} &= (.242688,\ .012113,\ .728328,\ .016871), \\ &\ \vdots \\ \pi^{(1000)} &= (.232618,\ .028906,\ .698094,\ .040382), \end{aligned}$$

which gives the following values of the statistic $\pi^{(n)}a$:

$$.054900, \quad .109846, \quad .163928, \quad .217161, \quad \ldots, \quad .519446.$$

Finally, we point out that the distribution at time step $n = 1,000$ is a long way from the stationary distribution given by $\pi = (.131589,\ .197384,\ .394768,\ .276259)$, which has the statistic $\pi a = 3.552123$.

In the remainder of this section, we devote our attention to the computation of transient distributions of continuous-time Markov chains, i.e., the computation of $\pi(t)$ from

$$\pi(t) = \pi(0)e^{Qt},$$

where Q is the infinitesimal generator of an irreducible continuous-time Markov chain. Depending on the numerical approach adopted, $\pi(t)$ may be computed by first forming e^{Qt} and then premultiplying this with the initial probability vector $\pi(0)$. Such is the case with the matrix-scaling and -powering methods described in Section 10.7.1. They are most suitable when the transition rate matrix is small. The Matlab function *expm* falls into this category. In other cases, $\pi(t)$ may be computed directly without explicitly forming e^{Qt}. This is the approach taken by the uniformization method of Section 10.7.2 and the ordinary differential equation (ODE) solvers of Section 10.7.3. Both are suitable for large-scale Markov chains. Indeed, the uniformization method is a widely used and often extremely efficient approach that is applicable both to small transition rate matrices that may be either dense or sparse and to large sparse transition matrices. It is less suitable for stiff Markov chains. Methods for solving systems of ordinary differential equations have been actively researched by the numerical analysis community and provide an alternative to the uniformization method. In this section we suggest both single-step and multistep methods as possible ODE solution approaches.

10.7.1 Matrix Scaling and Powering Methods for Small State Spaces

Moler and Van Loan [37] discuss nineteen *dubious* ways to compute the exponential of a relatively small-order matrix. A major problem in all these methods is that the accuracy of the approximations depends heavily on the norm of the matrix. Thus, when the norm of Q is large or t is large, attempting to compute e^{Qt} directly is likely to yield unsatisfactory results. It becomes necessary to divide the interval $[0, t]$ into subintervals (called panels) $[0, t_0]$, $[t_0, t_1]$, $\ldots$, $[t_{m-1}, t_m = t]$ and to compute the transient solution at each time $t_j,\ j = 0, 1, \ldots, m$ using the solution at the beginning of the panel as the starting point. It often happens that this is exactly what is required by a user who can, as a result, see the evolution of certain system performance measures with time.

Matrix-scaling and -powering methods arise from a property that is unique to the exponential function, namely,

$$e^{Qt} = \left(e^{Qt/2}\right)^2. \tag{10.41}$$

The basic idea is to compute e^{Qt_0} for some small value t_0 such that $t = 2^m t_0$ and subsequently to form e^{Qt} by repeated application of the relation (10.41). It is in the computation of the initial e^{Qt_0} that methods differ, and we shall return to this point momentarily.

Let Q be the infinitesimal generator of an ergodic, continuous-time Markov chain, and let $\pi(0)$ be the probability distribution at time $t = 0$. We seek $\pi(t)$, the transient solution at time t. Let us introduce an integer m and a time $t_0 \neq 0$ for which $t = 2^m t_0$. Then

$$\pi(t) = \pi(2^m t_0).$$

Writing $t_j = 2t_{j-1}$, we shall compute the matrices e^{Qt_j} for $j = 0, 1, \ldots, m$ and consequently, by multiplication with $\pi(0)$, the transient solution at times $t_0, 2t_0, 2^2 t_0, \ldots, 2^m t_0 = t$. Note that $P(t_j) \equiv e^{Qt_j}$ is a stochastic matrix and that, from the Chapman–Kolmogorov equations,

$$P(t_j) = P(t_{j-1})P(t_{j-1}). \tag{10.42}$$

Thus, once $P(t_0)$ has been computed, each of the remaining $P(t_j)$ may be computed from Equation (10.42) by squaring the previous $P(t_{j-1})$. Thus matrix-powering methods, in the course of their computation, provide the transient solution at the intermediate times $t_0, 2t_0, 2^2 t_0, \ldots, 2^{m-1} t_0$. However, a disadvantage of matrix-powering methods, besides computational costs proportional to n^3 and memory requirements of n^2, is that repeated squaring may induce rounding error buildup, particularly in instances in which $m \gg 1$.

Example 10.31 Suppose we need to find the transition distribution at some time $t = 10$, of a Markov chain with infinitesimal generator

$$Q = \begin{pmatrix} -.6 & 0 & .6 & 0 \\ .1 & -.9 & .1 & .7 \\ .4 & .3 & -.8 & .1 \\ 0 & .5 & 0 & -.5 \end{pmatrix}.$$

To use the matrix-scaling and -powering method, we need to find a t_0 such that t_0 is small and there exists an integer m so that $t = 2^m t_0$. Setting $m = 5$ and solving for t_0 gives $t_0 = 10/32$ which we take to be sufficiently small for the purpose of this example. The reader may now wish to verify that

$$P(t_0) = e^{10Q/32} = \begin{pmatrix} .838640 & .007076 & .151351 & .002933 \\ .026528 & .769552 & .026528 & .177393 \\ .102080 & .074581 & .789369 & .033970 \\ .002075 & .126413 & .002075 & .869437 \end{pmatrix}$$

and that performing the operations $P(t_j) = P(t_{j-1})P(t_{j-1})$ for $j = 1, 2, \ldots, 5$ gives the result

$$P(t_5) = \begin{pmatrix} .175703 & .265518 & .175700 & .383082 \\ .136693 & .289526 & .136659 & .437156 \\ .161384 & .274324 & .161390 & .402903 \\ .129865 & .293702 & .129865 & .446568 \end{pmatrix} = e^{10Q}.$$

When premultiplied by an initial probability distribution, the distribution at time $t = 10$ can be found.

We now turn our attention to the computation of e^{Qt_0}. Since t_0 is small, methods based on approximations around zero are possible candidates. One good choice is that of rational Padé approximations around the origin. The (p, q) Padé approximant to the matrix exponential e^X is, by definition, the unique (p, q) rational function $R_{pq}(X)$,

$$R_{pq}(X) \equiv \frac{N_{pq}(X)}{D_{pq}(X)},$$

which matches the Taylor series expansion of e^X through terms to the power $p + q$. Its coefficients are determined by solving the algebraic equations

$$\sum_{j=0}^{\infty} \frac{X^j}{j!} - \frac{N_{pq}(X)}{D_{pq}(X)} = O\left(X^{p+q+1}\right),$$

which yields

$$N_{pq}(X) = \sum_{j=0}^{p} \frac{(p+q-j)!p!}{(p+q)!j!(p-j)!} X^j$$

and

$$D_{pq}(X) = \sum_{j=0}^{q} \frac{(p+q-j)!q!}{(p+q)!j!(q-j)!} (-X)^j.$$

For more detailed information on Padé approximants, the interested reader should consult the text by Baker [1]. A major disadvantage of Padé approximants is that they are accurate only near the origin and so should not be used when $\|X\|_2$ is large. However, since we shall be using them in the context of a matrix-scaling and -powering procedure, we may (and shall) choose t_0 so that $\|Qt_0\|_2$ is sufficiently small that the Padé approximant to e^{Qt_0} may be obtained with acceptable accuracy, even for relatively low-degree approximants.

When $p = q$, we obtain the *diagonal Padé approximants*, and there are two main reasons why this choice is to be recommended. First, they are more stable [37]. In Markov chain problems, all the eigenvalues of $X = Qt$ are to be found in the left half plane. In this case the computed approximants $R_{pq}(X)$ for $p \neq q$ have larger rounding errors, because either $p > q$ and cancellation problems may arise, or $p < q$ and $D_{pq}(X)$ may be badly conditioned. Second, we obtain a higher-order method with the same amount of computation. To compute $R_{pq}(X)$ with $p < q$ requires about qn^3 flops and yields an approximant that has order $p + q$. To compute $R_{qq}(X)$ requires essentially the same number of flops but produces an approximant of order $2q > p+q$. Similar statements may be made when $p > q$.

For diagonal Padé approximants we find

$$R_{pp}(X) = \frac{N_{pp}(X)}{N_{pp}(-X)}, \tag{10.43}$$

where

$$N_{pp}(X) = \sum_{j=0}^{p} \frac{(2p-j)!p!}{(2p)!j!(p-j)!} X^j \equiv \sum_{j=0}^{p} c_j X^j.$$

The coefficients c_j can be conveniently constructed by means of the recursion

$$c_0 = 1; \quad c_j = c_{j-1} \frac{p+1-j}{j(2p+1-j)}.$$

For actual implementation purposes, the following irreducible form offers considerable savings in computation time at the expense of additional memory locations:

$$R_{pp}(X) = \begin{cases} I + 2\dfrac{X\sum_{k=0}^{p/2-1} c_{2k+1}X^{2k}}{\sum_{k=0}^{p/2} c_{2k}X^{2k} - X\sum_{k=0}^{p/2-1} c_{2k+1}X^{2k}} & \text{if } p \text{ is even,} \\[2ex] -I - 2\dfrac{\sum_{k=0}^{(p-1)/2} c_{2k}X^{2k}}{X\sum_{k=0}^{(p-1)/2} c_{2k+1}X^{2k} - \sum_{k=0}^{(p-1)/2} c_{2k}X^{2k}} & \text{if } p \text{ is odd.} \end{cases} \tag{10.44}$$

Thus, for even values of p,

$$R_{pp}(X) = I + 2\frac{S_e}{T_e - S_e},$$

where

$$S_e = c_1 X + c_3 X^3 + \cdots + c_{p-1}X^{p-1} \quad \text{and} \quad T_e = c_0 + c_2X^2 + c_4X^4 + \cdots + c_pX^p,$$

while for odd values of p,

$$R_{pp}(X) = -\left(I + 2\frac{S_o}{T_o - S_o}\right),$$

where now

$$S_o = c_0 + c_2X^2 + c_4X^4 + \cdots + c_{p-1}X^{p-1} \quad \text{and} \quad T_o = c_1X + c_3X^3 + \cdots + c_pX^p.$$

These computations may be conveniently combined, and they cry out for a Horner-type evaluation procedure. Indeed, Horner evaluations of the numerator and the denominator in Equation (10.44) need only one-half the operations of a straightforward implementation of Equation (10.43).

The following four steps, adapted from that presented in [42], implements a Padé variant of the matrix-powering and -scaling approach for the computation of e^X. In this implementation, the integer m is chosen as $m = \lfloor \log \|X\|_\infty / \log 2 \rfloor + 1$. To compute the transient solution at time t of a Markov chain with generator Q and initial state $\pi(t_0)$, it suffices to apply this algorithm with $X = Qt$ and then to form $\pi(t_0)R$, where R is the approximation to e^X computed by the algorithm.

```
1. Find appropriate scaling factor:
   • Compute $m = \max(0, \lfloor \log \|X\|_\infty / \log 2 \rfloor + 1)$.
2. Compute coefficients and initialize:
   • Set $c_0 = 1$.
   • For $j = 1, 2, \ldots, p$ do
      ◦ Compute $c_j = c_{j-1} \times (p+1-j)/(j(2p+1-j))$.
   • Compute $X1 = 2^{-m}X;\ \ X2 = X1^2;\ \ T = c_p I;\ \ S = c_{p-1} I$.
3. Application of Horner scheme:
   • Set $odd = 1$.
   • For $j = p-1, \ldots, 2, 1$ do
      ◦ if $odd = 1$, then
         ∗ Compute $T = T \times X2 + c_{j-1} I$;
        else
         ∗ Compute $S = S \times X2 + c_{j-1} I$.
      ◦ Set $odd = 1 - odd$.
   • If $odd = 0$, then
      ◦ Compute $S = S \times X1;\ \ R = I + 2 \times (T-S)^{-1} \times S$;
     else
      ◦ Compute $T = T \times X1;\ \ R = -(I + 2 \times (T-S)^{-1} \times S)$.
4. Raise matrix to power $2^m$ by repeated squaring:
   • For $j = 1$ to $m$ do
      ◦ Compute $R = R \times R$.
```

This Padé approximation for e^X requires a total of approximately $(p + m + \frac{4}{3})n^3$ multiplications. It may be implemented with three double-precision arrays, each of size n^2, in addition to the storage required for the matrix itself.

This leaves us with the choice of p. In the appendix of [37], a backward error analysis of the Padé approximation is presented, in which it is shown that if $\|X\|_2/2^m \le \frac{1}{2}$, then

$$\left[R_{pp}(2^{-m}X)\right]^{2^m} = e^{X+E},$$

where

$$\frac{\|E\|_2}{\|X\|_2} \le \left(\frac{1}{2}\right)^{2p-3} \frac{(p!)^2}{(2p)!(2p+1)!} \approx \begin{cases} 0.77 \times 10^{-12}\ (p=5), \\ 0.34 \times 10^{-15}\ (p=6), \\ 0.11 \times 10^{-18}\ (p=7), \\ 0.27 \times 10^{-22}\ (p=8). \end{cases} \tag{10.45}$$

This suggests that relatively low-degree Padé approximants are adequate. However, the above analysis does not take rounding error into account. This aspect has been examined by Ward [54], who proposes certain criteria for selecting appropriate values for some computers. Additionally, a discussion on “the degree of best rational approximation to the exponential function” is provided by Saff [48]. Finally, numerical experiments on Markov chains by Philippe and Sidje [42] find that even values of p are better than odd values and that the value $p = 6$ is generally satisfactory.

10.7.2 The Uniformization Method for Large State Spaces

For large-scale Markov chains, methods currently used to obtain transient solutions are based either on readily available differential equation solvers such as Runge-Kutta methods or the Adams formulae and backward differentiation formulae (BDF) or on the method of uniformization (also called Jensen's method or the method of randomization). Most methods experience difficulty when both $\max_j |q_{jj}|$ (the largest exit rate from any state) and t (the time at which the solution is required) are large, and there appears to be little to recommend a single method for all situations. In this section we discuss the *uniformization method*. This method has attracted much attention, is extremely simple to program and often outperforms other methods, particularly when the solution is needed at a single time point close to the origin. If the solution is required at many points, or if plots need to be drawn to show the evolution of certain performance measures, then a method based on one of the differential equation solvers may be preferable. An extensive discussion on all these methods may be found in [50].

The uniformization method revolves around a discrete-time Markov chain that is embedded in the continuous-time process. The transition probability matrix of this discrete-time chain is constructed as

$$P = Q\Delta t + I$$

with $\Delta t \le 1/\max_i |q_{ii}|$. In this Markov chain all state transitions occur at a *uniform* rate equal to $1/\Delta t$—hence the name *uniformization*. Letting $\gamma = \max_i |q_{ii}|$, we may write

$$Q = \gamma(P - I)$$

and inserting this into the Kolmogorov forward differential equations we get

$$\pi(t) = \pi(0)e^{Qt} = \pi(0)e^{\gamma(P-I)t} = \pi(0)e^{-\gamma t}e^{\gamma Pt}.$$

Expanding $e^{\gamma Pt}$ in a Taylor series, we obtain

$$\pi(t) = \pi(0)e^{-\gamma t}\sum_{k=0}^{\infty}\frac{(\gamma t)^k P^k}{k!},$$

i.e.,

$$\pi(t) = \sum_{k=0}^{\infty}\pi(0)P^k\frac{(\gamma t)^k}{k!}e^{-\gamma t}. \tag{10.46}$$

Two observations may be made. First, the term $\pi(0)P^k$ may be recognized as the vector that provides the probability distribution after k steps of the discrete-time Markov chain whose stochastic transition probability matrix is P and with initial distribution $\pi(0)$. Second, the term $e^{-\gamma t}(\gamma t)^k/k!$ may be recognized from the Poisson process with rate γ as the probability of k events occurring in $[0, t)$. It is the probability that the discrete-time Markov chain makes k transition steps in the interval $[0, t)$. These probabilities may be interpreted as weights that when multiplied with the distribution of the discrete-time Markov chain after k steps and summed over all possible number of steps, (in effect, *unconditioning* the transient distribution which has been written in terms of power series of P) yields the transient distribution $\pi(t)$. The uniformization method computes the distribution $\pi(t)$ directly from Equation (10.46). Writing it in the form

$$\pi(t) = e^{-\gamma t}\sum_{k=0}^{\infty}\left(\pi(0)P^{k-1}\frac{(\gamma t)^{k-1}}{(k-1)!}\right)P\frac{\gamma t}{k} \tag{10.47}$$

exposes a convenient recursive formulation. Setting $y = \pi = \pi(0)$ and iterating sufficiently long with

$$y = y\left(P\frac{\gamma t}{k}\right), \quad \pi = \pi + y,$$

allows the transient distribution to be computed as $\pi(t) = e^{-\gamma t}\,\pi$.

The curious reader may wonder why we bother with Equation (10.46) when $\pi(t)$ might be thought to be more easily computed directly from the Chapman-Kolmogorov differential equations by means of the formula

$$\pi(t) = \pi(0)\sum_{k=0}^{\infty}\frac{(Qt)^k}{k!}.$$

However, the matrix Q contains both positive and negative elements and these may be greater than one which leads to a less stable algorithm than one based on the matrix P which, being a stochastic matrix, has all positive elements lying in the range [0, 1].

Among the numerical advantages of the uniformization technique is the ease with which it can be translated into computer code and the control it gives over the truncation error. Let us first discuss the truncation error. In implementing the uniformization method, we need to truncate the infinite series in (10.46). Let

$$\pi^*(t) = \sum_{k=0}^{K}\pi(0)P^k e^{-\gamma t}(\gamma t)^k/k! \tag{10.48}$$

and let $\delta(t) = \pi(t) - \pi^*(t)$. For any consistent vector norm, $||\delta(t)||$ is the truncation error. It is not difficult to numerically bound this error. If we choose K sufficiently large that

$$1 - \sum_{k=0}^{K} e^{-\gamma t}(\gamma t)^k/k! \le \epsilon,$$

or, equivalently, that

$$\sum_{k=0}^{K}\frac{(\gamma t)^k}{k!} \ge \frac{1-\epsilon}{e^{-\gamma t}} = (1-\epsilon)e^{\gamma t}, \tag{10.49}$$

where ϵ is some prespecified truncation criterion, then it follows that

$$\|\pi(t) - \pi^*(t)\|_\infty \le \epsilon.$$

To see this, observe that

$$\left\|\pi(t) - \pi^*(t)\right\|_\infty$$

$$= \left\|\sum_{k=0}^{\infty}\pi(0)P^k e^{-\gamma t}(\gamma t)^k/k! - \sum_{k=0}^{K}\pi(0)P^k e^{-\gamma t}(\gamma t)^k/k!\right\|_\infty$$

$$= \left\|\sum_{k=K+1}^{\infty}\pi(0)P^k e^{-\gamma t}(\gamma t)^k/k!\right\|_\infty \le \sum_{k=K+1}^{\infty} e^{-\gamma t}(\gamma t)^k/k!$$

$$= \sum_{k=0}^{\infty} e^{-\gamma t}(\gamma t)^k/k! - \sum_{k=0}^{K} e^{-\gamma t}(\gamma t)^k/k!$$

$$= 1 - \sum_{k=0}^{K} e^{-\gamma t}(\gamma t)^k / k! \ \le \ \epsilon.$$

Example 10.32 Consider a continuous-time Markov chain with infinitesimal generator

$$Q = \begin{pmatrix} -5 & 2 & 3 \\ 1 & -2 & 1 \\ 6 & 4 & -10 \end{pmatrix}.$$

Given $\pi(0) = (1, 0, 0)$ and $t = 1$, we wish to examine the behavior of Equation (10.46) as the number of terms in the summation increases. The uniformization process gives $\gamma = 10$ and

$$P = \begin{pmatrix} .5 & .2 & .3 \\ .1 & .8 & .1 \\ .6 & .4 & 0 \end{pmatrix}.$$

Suppose we require an accuracy of $\epsilon = 10^{-6}$. To find the value of K, the number of terms to be included in the summation, we proceed on a step-by-step basis, incrementing k until

$$\sigma_K = \sum_{k=0}^{K} \frac{(\gamma t)^k}{k!} \ \ge \ (1-\epsilon)e^{\gamma t} = (1 - 10^{-6})e^{10} = 22,026.4438.$$

Observe that successive terms in the summation satisfy

$$\xi_{K+1} = \xi_K \frac{\gamma t}{K+1} \quad \text{with } \xi_0 = 1$$

and that

$$\sigma_{K+1} = \sigma_K + \xi_{K+1} \quad \text{with } \sigma_0 = 1,$$

and so, beginning with $K = 0$, and using this recursion, we successively compute

$$\sigma_0 = 1, \ \sigma_1 = 11, \ \sigma_2 = 61, \ \sigma_3 = 227.6667, \sigma_4 = 644.33331, \ \ldots, \ \sigma_{28} = 22,026.4490.$$

Thus $K = 28$ terms are needed in the summation. To obtain our approximation to the transient distribution at time $t = 1$, we need to compute

$$\pi(t) \approx \sum_{k=0}^{28} \pi(0)P^k e^{-\gamma t} \frac{(\gamma t)^k}{k!} \tag{10.50}$$

for some initial distribution, $\pi(0) = (1, 0, 0)$, say. Using the recursion relation of Equation (10.47), we find

$$\begin{aligned}
k = 0: &\quad y = (1,\ 0,\ 0); \quad \pi = (1,\ 0,\ 0),\\
k = 1: &\quad y = (5,\ 2,\ 3); \quad \pi = (6,\ 2,\ 3),\\
k = 2: &\quad y = (22.5,\ 19,\ 8.5); \quad \pi = (28.5,\ 21,\ 11.5),\\
k = 3: &\quad y = (60.8333,\ 77,\ 28.8333); \quad \pi = (89.3333,\ 98,\ 40.3333),\\
&\quad \vdots\\
k = 28: &\quad y = (.009371,\ .018742,\ .004686); \quad \pi = (6,416.9883,\ 12,424.44968,\ 3,184.4922).
\end{aligned}$$

In this example, the elements of y increase until they reach $y = (793.8925,\ 1,566.1563,\ 395.6831)$ for $k = 10$ and then they begin to decrease. Multiplying the final π vector by e^{-10} produces the desired transient distribution:

$$\pi(1) = e^{-10}(6,416.9883,\ 12,424.44968,\ 3,184.4922) = (.291331,\ .564093,\ .144576).$$

In implementing the uniformization technique, we may code Equation (10.46) exactly as it appears, with the understanding that $\pi(0)P^k$ is computed iteratively, i.e., we do not construct the k^{th} power of P and premultiply it with $\pi(0)$, but instead form the sequence of vectors, $\psi(j+1) = \psi(j)P$, with $\psi(0) = \pi(0)$, so that $\pi(0)P^k$ is given by $\psi(k)$. Alternatively we may partition Equation (10.46) into time steps $0 = t_0, t_1, t_2, ..., t_m = t$ and write code to implement

$$\pi(t_{i+1}) = \sum_{k=0}^{\infty} \pi(t_i)P^k e^{-\gamma(t_{i+1}-t_i)}\gamma^k(t_{i+1}-t_i)^k/k!$$

recursively for $i = 0, 1, ..., m-1$. This second approach is the obvious way to perform the computation if the transient solution is required at various points $t_1, t_2, ...$ between the initial time t_0 and the final time t. It is computationally more expensive if the transient solution is required only at a single terminal point. However, it may prove useful when the numerical values of γ and t are such that the computer underflows when computing $e^{-\gamma t}$. Such instances can be detected a priori and appropriate action taken. For example, one may decide not to allow values of γt to exceed 100. When such a situation is detected, the time t may be divided into $d = 1 + \lfloor \gamma t/100 \rfloor$ equal intervals and the transient solution computed at times $t/d,\ 2t/d,\ 3t/d,\ \ldots, t$. Care must be taken in implementing such a procedure since the error in the computation of the intermediate values $\pi(t_i)$ may propagate along into later values, $\pi(t_j),\ j > i$.

An alternative to dividing a large interval γt into more manageable pieces may be to omit from the summation in Equation (10.48), terms for which the value of $e^{-\gamma t}(\gamma t)^k/k!$ is so small that it could cause numerical difficulties. This may be accomplished by choosing a *left* truncation point, l, as the largest value for which

$$\sum_{k=0}^{l-1} e^{-\gamma t}\frac{(\gamma t)^k}{k!} \le \epsilon_l$$

for some lower limit ϵ_l. The required transient distribution vector is now computed as

$$\pi^*(t) = \sum_{k=l}^{K} \psi(k)e^{-\gamma t}(\gamma t)^k/k!$$

with $\psi(k)$ computed recursively as before. The value of l is easily computed using the same procedure that was used to compute the upper limit of the summation, K. The amount of work involved in summing from $k = l$ is basically just the same as that in summing from $k = 0$, since $\psi(k)$ from $k = 0$ to $k = K$ must be computed in all cases. The only reason in summing from $k = l$ is that of computational stability.

One other wrinkle may be added to the uniformization method when it is used to compute transient distributions at large values of γt—the *stationary distribution* of the uniformized chain may be reached well before the last term in the summation (10.48). If this is the case, it means that, from the point at which steady state is reached, the values $\psi(k)$ no longer change. It is possible to monitor convergence of the uniformized chain and to determine the point at which it reaches steady state. Assume this occurs when k has the value k_s. The transient distribution at time t may then be computed more efficiently as

$$\pi^*(t) = \sum_{k=l}^{k_s} \psi(k)e^{-\gamma t}(\gamma t)^k/k! + \left(\sum_{k=k_s+1}^{K} e^{-\gamma t}(\gamma t)^k/k!\right)\psi(k_s).$$

The following two step implementation computes the transient solution $\pi(t)$ at time t given the probability distribution $\pi(0)$ at time $t = 0$; P, the stochastic transition probability matrix of the discrete-time Markov chain; γ, the parameter of the Poisson process; and ϵ, a tolerance criterion. It is designed for the case in which the number of states in the Markov chain is large. The only operation involving the matrix P is its multiplication with a vector. This implementation does not incorporate a lower limit nor a test for convergence to the steady-state.

```
1. Use Equation (10.49) to compute K, the number of terms in the summation:
   • Set $K = 0$; $\xi = 1$; $\sigma = 1$; $\eta = (1 - \epsilon)/e^{-\gamma t}$.
   • While $\sigma < \eta$ do
      ○ Compute $K = K + 1$; $\xi = \xi \times (\gamma t)/K$; $\sigma = \sigma + \xi$.
2. Approximate $\pi(t)$ from Equation (10.48):
   • Set $\pi = \pi(0)$; $y = \pi(0)$.
   • For $k = 1$ to $K$ do
      ○ Compute $y = yP \times (\gamma t)/k$; $\pi = \pi + y$.
   • Compute $\pi(t) = e^{-\gamma t}\pi$.
```

10.7.3 Ordinary Differential Equation Solvers

The transient distribution we seek to compute, $\pi(t) = \pi(0)e^{Qt}$, is the solution of the Chapman–Kolmogorov differential equations

$$\frac{d\pi(t)}{dt} = \pi(t)Q$$

with initial conditions $\pi(t = 0) = \pi(0)$. The solution of ordinary differential equations (ODEs) has been (and continues to be) a subject of extensive research, and there are numerous possibilities for applying ODE procedures to compute transient solutions of Markov chains. An immediate advantage of such an approach is that, unlike uniformization and matrix scaling and powering, numerical methods for the solution of ODEs are applicable to *nonhomogeneous* Markov chains, i.e., Markov chains whose infinitesimal generators are a function of time, $Q(t)$. However, since the subject of ODEs is so vast, and our objectives in this chapter rather modest, we shall be content to give only the briefest of introductions to this approach. Given a first-order differential equation $y' = f(t, y)$ and an initial condition $y(t_0) = y_0$, a solution is a differentiable function $y(t)$ such that

$$y(t_0) = y_0, \quad \frac{d}{dt}y(t) = f(t, y(t)). \tag{10.51}$$

In the context of Markov chains, the solution $y(t)$ is the row vector $\pi(t)$ and the function $f(t, y(t))$ is simply $\pi(t)Q$. We shall use the standard notation of Equation(10.51) for introducing general concepts, and the Markov chain notation when the occasion so warrants, such as in examples and Matlab code.

Numerical procedures to compute the solution of Equation (10.51) attempt to follow a unique solution curve from its value at an initially specified point to its value at some other prescribed point τ. This usually involves a discretization procedure on the interval $[0, \tau]$ and the computation of approximations to the solution at the intermediate points. Given a discrete set of points (or *mesh*) $\{0 = t_0, t_1, t_2, \ldots, t_\eta = \tau\}$ in $[0, \tau]$, we denote the exact solution of the differential Equation (10.51) at time t_i by $y(t_i)$. The *step size* or *panel width* at step i is defined as $h_i = t_i - t_{i-1}$. A numerical method generates a sequence $\{y_1, y_2, \ldots, y_\eta\}$ such that y_i is an approximation[3] to $y(t_i)$. In some instances the solution at time τ is all that is required, and in that case the computed solution is taken to be y_η. In other cases the value of $y(t)$ for all $t_0 \le t \le \tau$ is required, often in the form of a graph, and this is obtained by fitting a suitable curve to the values $y_0, y_1, y_2, \ldots, y_\eta$.

[3] The reader should be careful not to confuse y_i with $y(t_i)$. We use y_i for $i \neq 0$ to denote a *computed approximation* to the *exact value* $y(t_i)$. For $i = 0$, the initial condition gives $y_0 = y(t_0)$.

Single-Step Euler Method

If the solution is continuous and differentiable, then in a small neighborhood of the point (t_0, y_0) we can approximate the solution curve by its tangent y_0' at (t_0, y_0) and thereby move from (t_0, y_0) to the next point (t_1, y_1). Since this means that

$$y_0' = \frac{y_1 - y_0}{t_1 - t_0},$$

we obtain the formula

$$y_1 = y_0 + h_1 y_0' = y_0 + h_1 f(t_0, y_0), \tag{10.52}$$

where $h_1 = t_1 - t_0$. This process may be repeated from the computed approximation y_1 to obtain

$$y_2 = y_1 + h_2 f(t_1, y_1), \tag{10.53}$$

where $h_2 = t_2 - t_1$, and so on until the final destination is reached,

$$y_\eta = y_{\eta-1} + h_\eta f(t_{\eta-1}, y_{\eta-1}) \tag{10.54}$$

with $h_\eta = t_\eta - t_{\eta-1}$. This method is called the *forward Euler method (FEM)* or *explicit Euler method*. From Equations (10.52)–(10.54), the FEM method may be written as

$$y_{i+1} = y_i + h_{i+1} f(t_i, y_i), \quad i = 0, 1, \ldots, \eta - 1, \tag{10.55}$$

where $h_{i+1} = t_{i+1} - t_i$. In computing y_{i+1}, a method may incorporate the values of previously computed approximations y_j for $j = 0, 1, \ldots, i$, or even previous approximations to $y(t_{i+1})$. A method that uses only (t_i, y_i) to compute y_{i+1} is said to be an *explicit single-step method*. It is said to be a *multistep method* if it uses approximations at several previous steps to compute its new approximation. A method is said to be *implicit* if computation of y_{i+1} requires an approximation to $y(t_{i+1})$; otherwise, it is said to be *explicit*. An *implicit Euler method*, also called a *backward Euler method*, is defined by using the slope at the point (t_{i+1}, y_{i+1}). This gives

$$y_{i+1} = y_i + h_{i+1} f(t_{i+1}, y_{i+1}).$$

The *modified Euler method* incorporates the average of the slopes at both points under the assumption that this will provide a better average approximation of the slope over the entire panel $[t_i, t_{i+1}]$. The formula is given by

$$y_{i+1} = y_i + h_{i+1} \frac{f(t_i, y_i) + f(t_{i+1}, y_{i+1})}{2}. \tag{10.56}$$

This is also referred to as the *trapezoid rule*. A final formula in this category is the *implicit midpoint rule*, which involves the slope at the midpoint of the interval. The formula is given by

$$y_{i+1} = y_i + h_{i+1}\, f\left(t_i + \frac{h_i}{2}, \frac{y_i + y_{i+1}}{2}\right).$$

These are all single-step methods, since approximations preceding (t_i, y_i) are not used in the generation of the next approximation y_{i+1}.

In Markov chain problems the explicit Euler method (Equation10.55) is given by

$$\pi_{(i+1)} = \pi_{(i)} + h_{i+1} \pi_{(i)} Q,$$

i.e.,

$$\pi_{(i+1)} = \pi_{(i)} \left(I + h_{i+1} Q\right). \tag{10.57}$$

Note that $\pi_{(i)}$ is the *state vector* of probabilities at time t_i. We use this notation, rather than π_i, so as not to confuse the i^{th} component of the vector with the entire vector at time t_i. Thus, in the explicit

Euler method applied to Markov chains, moving from one time step to the next is accomplished by a scalar-matrix product and a vector-matrix product.

Example 10.33 Consider a continuous-time Markov chain with infinitesimal generator

$$Q = \begin{pmatrix} -2 & 1 & 1 \\ 3 & -8 & 5 \\ 1 & 2 & -3 \end{pmatrix}.$$

Suppose we wish to compute the transient distribution at time $t = 1$ given that the Markov chain begins in state 1. For illustrative purposes, we shall use a constant step size equal to 0.1 and compute $\pi_{(i)},\ i = 1, 2, \ldots, 10$. From the FEM formula, we obtain

$$\pi(.1) \approx \pi_{(1)} = \pi_{(0)}(I + .1Q) = (1, 0, 0)\left[\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + .1 \times \begin{pmatrix} -2 & 1 & 1 \\ 3 & -8 & 5 \\ 1 & 2 & -3 \end{pmatrix}\right]$$

$$= (1, 0, 0)\begin{pmatrix} .8 & .1 & .1 \\ .3 & .2 & .5 \\ .1 & .2 & .7 \end{pmatrix} = (.8, .1, .1).$$

$$\pi(.2) \approx \pi_{(2)} = (.8, .1, .1)\begin{pmatrix} .8 & .1 & .1 \\ .3 & .2 & .5 \\ .1 & .2 & .7 \end{pmatrix} = (.68, .12, .20).$$

Continuing in this fashion we finally compute

$$\pi(1.0) \approx \pi_{(10)} = (.452220,\ .154053,\ .393726)\begin{pmatrix} .8 & .1 & .1 \\ .3 & .2 & .5 \\ .1 & .2 & .7 \end{pmatrix} = (.447365,\ .154778,\ .397857).$$

The correct answer, obtained by a different method, is

$$\pi(1.0) = (.457446,\ .153269,\ .389285),$$

and a measure of the error is given as $\|\pi(1.0) - \pi_{(10)}\|_2 = .013319$.

The modified Euler or trapezoid rule, Equation (10.56), when applied to Markov chains, becomes

$$\pi_{(i+1)} = \pi_{(i)} + \frac{h_{i+1}}{2}\left(\pi_{(i)}Q + \pi_{(i+1)}Q\right),$$

i.e.,

$$\pi_{(i+1)}\left(I - \frac{h_{i+1}}{2}Q\right) = \pi_{(i)}\left(I + \frac{h_{i+1}}{2}Q\right), \tag{10.58}$$

which requires, in addition to the operations needed by the explicit Euler method, the solution of a system of equations at each step (plus a scalar-matrix product). These additional computations per step are offset to a certain extent by the better accuracy achieved with the trapezoid rule. If the size of the Markov chain is small and the step size is kept constant, the matrix

$$\left(I + \frac{h_{i+1}}{2}Q\right)\left(I - \frac{h_{i+1}}{2}Q\right)^{-1}$$

may be computed at the outset before beginning the stepping process, so that the computation per step required by the modified Euler method becomes identical to that of the explicit Euler. If the Markov chain is large, then the inverse should not be formed explicitly. Instead an LU decomposition should be computed and the inverse replaced with a backward and forward substitution process with U and L respectively. When the step size is not kept constant, each different value of h used requires that a system of linear equations be solved. Depending on the size and sparsity pattern of Q, the work required by the trapezoid rule to compute the solution to a specified precision may or may not be less than that required by the explicit Euler. The trade-off is between an implicit method requiring more computation per step but fewer steps and an explicit method requiring more steps but less work per step!

Example 10.34 Let us return to Example 10.33 and this time apply the trapezoid rule. Again, we shall use a step size of 0.1 and successively compute $\pi_{(i)},\ i = 1, 2, \ldots, 10$. From the Trapezoid formula, we obtain

$$\pi_{(i+1)} = \pi_{(i)}\left(I + \frac{h_{i+1}}{2}Q\right)\left(I - \frac{h_{i+1}}{2}Q\right)^{-1} = \pi_{(i)}\begin{pmatrix} .90 & .05 & .05 \\ .15 & .60 & .25 \\ .05 & .10 & .85 \end{pmatrix}\begin{pmatrix} 1.10 & -.05 & -.05 \\ -.15 & 1.40 & -.25 \\ -.05 & -.10 & 1.15 \end{pmatrix}^{-1},$$

i.e.,

$$\pi_{(i+1)} = \pi_{(i)}\begin{pmatrix} .832370 & .072254 & .095376 \\ .213873 & .459538 & .326590 \\ .098266 & .130058 & .771676 \end{pmatrix}.$$

Beginning with $\pi_{(0)} = (1, 0, 0)$, this allows us to compute

$$\begin{aligned} \pi(.1) &\approx \pi_{(1)} = (.832370,\ .072254,\ .095376), \\ \pi(.2) &\approx \pi_{(2)} = (.717665,\ .105750,\ .176585), \\ \pi(.3) &\approx \pi_{(3)} = (.637332,\ .123417,\ .239251), \\ &\ \vdots \\ \pi(1.0) &\approx \pi_{(10)} = (.456848,\ .153361,\ .389791). \end{aligned}$$

The correct answer is still

$$\pi(1.0) = (.457446,\ .153269,\ .389285),$$

and a measure of the error is given as $\|\pi(1.0) - \pi_{(10)}\|_2 = .0007889$ which is considerably better than that obtained with the FEM method.

Example 10.35 This time we examine the effect of decreasing the step size using the same infinitesimal generator, taking $\pi_{(0)} = (1, 0, 0)$ and the length of the interval of integration to be $\tau = 1$, as before.

$$Q = \begin{pmatrix} -2 & 1 & 1 \\ 3 & -8 & 5 \\ 1 & 2 & -3 \end{pmatrix}.$$

Table 10.2 shows the results obtained when Equations (10.57) and (10.58) are implemented in Matlab.[4] The first column gives the step size used throughout the range, the second is the 2-norm of the absolute error using the explicit Euler method, and the third column gives the 2-norm of the absolute error when the trapezoid method is used. Observe that the accuracy achieved with the explicit Euler method has a direct relationship with the step size. Observe also the much more accurate results obtained with the trapezoid method. Since equal step sizes were used in these examples, an essentially identical amount of computation is required by both. The trapezoid rule needs only an additional (3×3) matrix inversion and a matrix-matrix product.

[4] A listing of all the Matlab implementations of ODE methods used to generate results in this chapter is given at the end of this section.

Table 10.2. Euler and Trapezoid Method Results

Step size	Explicit Euler $\|y(t_1) - y_1\|_2$	Trapezoid $\|y(t_1) - y_1\|_2$
$h = .1$	$0.133193e{-}01$	$0.788907e{-}03$
$h = .01$	$0.142577e{-}02$	$0.787774e{-}05$
$h = .001$	$0.143269e{-}03$	$0.787763e{-}07$
$h = .0001$	$0.143336e{-}04$	$0.787519e{-}09$
$h = .00001$	$0.143343e{-}05$	$0.719906e{-}11$

Taylor Series Methods

We now turn our attention to *Taylor series methods*. The forward Euler method corresponds to taking the first two terms of the Taylor series expansion of $y(t)$ around the current approximation point. Expanding $y(t)$ as a power series in h, we have

$$y(t+h) = y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + \cdots . \qquad (10.59)$$

If we know $y(t)$ exactly, we may compute as many terms in the series as we like by repeated differentiation of $y' = f(t, y(t))$, the only constraint being the differentiability of the function $f(t, y(t))$. Different *Taylor series algorithms* are obtained by evaluating different numbers of terms in the series (10.59). For example, using the obvious notational correspondence with Equation (10.59), the Taylor's algorithm of order 2 may be written

$$y_{i+1} = y_i + hT_2(t_i, y_i),$$

where

$$T_2(t_i, y_i) = f(t_i, y_i) + \frac{h}{2}f'(t_i, y_i).$$

The order-1 Taylor series algorithm corresponds to the FEM method, since we have

$$y_{i+1} = y_i + hT_1(t_i, y_i) = y_i + hf(t_i, y_i).$$

Taylor series methods are not widely used because of the need to construct derivatives of the function $f(t, y(t))$. Their use is that they provide a mechanism to develop other methods which have an accuracy comparable to that of Taylor series of order p but which do not require derivatives of $f(t, y(t))$. This leads us introduce the concept of *order of convergence*.

Definition 10.7.1 (Order of convergence) *A numerical method is said to be convergent to order p if the sequence of approximations $\{y_0, y_1, y_2, \dots, y_\eta\}$ generated by the method (with constant step size $h = \tau/\eta$) satisfies*

$$\|y_i - y(t_i)\| = O(h^p), \quad i = 0, 1, \dots, \eta.$$

Example 10.36 Euler's method, described by Equation (10.55), is convergent to order 1; the error behaves like Mh, where M is a constant that depends on the problem and h is the maximum step size.

The order of convergence of a method concerns only the truncation error made in terminating the infinite series (10.59). In particular, it does not take into account any roundoff error that might occur. Indeed, as the truncation error tends to zero with decreasing h, the roundoff error has the opposite effect. The smaller the step size, the greater the number of steps that must be taken, and hence the greater the amount of computation that must be performed. With this greater amount of computation comes the inevitable increase in roundoff error. As h becomes smaller, the total error (truncation error plus rounding error) decreases, but invariably there comes a point at which it begins to increase again. Thus, it is important that h not be chosen too small.

Runge-Kutta Methods

An extremely important and effective class of single step methods is the class of Runge–Kutta methods. A Runge–Kutta algorithm of order p provides an accuracy comparable to a Taylor series algorithm of order p, but without the need to determine and evaluate the derivatives $f', f'', \ldots, f^{(p-1)}$, requiring instead the evaluation of $f(t, y)$ at selected points. The derivation of an order p Runge–Kutta method is obtained from a comparison with the terms through h^p in the Taylor series method for the first step, i.e., the computation of y_1 from the initial condition (t_0, y_0). This is because the analysis assumes that the value around which the expansion is performed is known exactly. A Runge–Kutta method is said to be of order p if the Taylor series for the exact solution $y(t_0 + h)$ and the Taylor series for the computed solution y_1 coincide up to and including the term h^p.

For example, a Runge–Kutta method of order 2 (RK2) is designed to give the same order of convergence as a Taylor series method of order 2, without the need for differentiation. It is written as

$$y_{i+1} = y_i + h(ak_1 + bk_2), \tag{10.60}$$

where a and b are constants,

$$\begin{aligned} k_1 &= f(t_i, y_i), \\ k_2 &= f(t_i + \alpha h, y_i + h\beta k_1), \end{aligned}$$

and α and β are constants. The constants are chosen so that the Taylor series approach agrees with Equation (10.60) at $i = 0$, through the term in h^2. Finding these constants requires computing the derivatives of Equation (10.60) at $i = 0$, and then comparing coefficients with those of the first and second derivatives obtained from the exact Taylor series expansion. We leave this as an exercise. Equating the coefficients imposes the following restrictions on the constants a, b, α, β:

$$a + b = 1, \quad \alpha b = \beta b = 1/2.$$

These are satisfied by the choice $a = b = \frac{1}{2}$ and $\alpha = \beta = 1$, so the resulting RK2 method is

$$y_{i+1} = y_i + \frac{h}{2} f(t_i, y_i) + \frac{h}{2} f(t_i + h, y_i + hf(t_i, y_i)).$$

Other choices for the constants are also possible.

The most widely used Runge–Kutta methods are of order 4. The standard RK4 method requires four function evaluations per step and is given by

$$y_{i+1} = y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{10.61}$$

where

$$\begin{aligned} k_1 &= f(t_i, y_i), \\ k_2 &= f(t_i + h/2, y_i + hk_1/2), \\ k_3 &= f(t_i + h/2, y_i + hk_2/2), \\ k_4 &= f(t_i + h, y_i + hk_3). \end{aligned}$$

Notice that each k_i corresponds to a point in the (t, y) plane at which the evaluation of f is required. Each is referred to as a *stage*. Thus, the method described by Equation (10.61) is a fourth-order, four-stage, explicit Runge–Kutta method.

When the standard explicit fourth-order Runge–Kutta method given by Equation (10.61) is applied to the Chapman–Kolmogorov equations $\pi' = \pi Q$, the sequence of operations to be performed to move from $\pi_{(i)}$ to the next time step $\pi_{(i+1)}$ is as follows:

$$\pi_{(i+1)} = \pi_{(i)} + h(k_1 + 2k_2 + 2k_3 + k_4)/6,$$

where

$$\begin{aligned} k_1 &= \pi_{(i)} Q, \\ k_2 &= (\pi_{(i)} + hk_1/2)Q, \\ k_3 &= (\pi_{(i)} + hk_2/2)Q, \\ k_4 &= (\pi_{(i)} + hk_3)Q. \end{aligned}$$

Example 10.37 We return to the continuous-time Markov chain of the previous examples and this time apply the RK4 method to obtain the same transient distribution. With $\pi_{(0)} = (1, 0, 0)$ and

$$Q = \begin{pmatrix} -2 & 1 & 1 \\ 3 & -8 & 5 \\ 1 & 2 & -3 \end{pmatrix},$$

we find, taking steps of size .1,

$$\begin{aligned} k_1 &= \pi_{(0)} Q = (-2,\ 1,\ 1), \\ k_2 &= (\pi_{(0)} + .05k_1)Q = (.9,\ .05,\ .05)Q = (-1.6,\ .6,\ 1), \\ k_3 &= (\pi_{(0)} + .05k_2)Q = (.92,\ .03,\ .05)Q = (-1.7,\ .78,\ .92), \\ k_4 &= (\pi_{(0)} + .1k_3)Q = (.830,\ .078,\ .092)Q = (-1.334,\ .390,\ .944), \end{aligned}$$

which allows us to compute

$$\pi(.1) \approx \pi_{(1)} = \pi_{(0)} + .1(k_1 + 2k_2 + 2k_3 + k_4)/6 = (.834433,\ .069167,\ .0964).$$

Continuing in this fashion we find

$$\begin{aligned} \pi(.2) \approx \pi_{(2)} &= (.720017,\ .103365,\ .176618), \\ \pi(.3) \approx \pi_{(3)} &= (.639530,\ .121971,\ .238499), \\ &\vdots \\ \pi(1.0) \approx \pi_{(10)} &= (.457455,\ .153267,\ .389278), \end{aligned}$$

and the final error is

$$\|\pi(1.0) - \pi_{(10)}\|_2 = .00001256.$$

An important characterization of initial-value ODEs has yet to be discussed—that of *stiffness*. Explicit methods have enormous difficulty solving stiff ODEs, indeed to such an extent that one definition of stiff equations is “problems for which explicit methods don’t work” [20]! Many factors contribute to stiffness, including the eigenvalues of the Jacobian $\partial f / \partial y$ and the length of the interval of integration. The problems stem from the fact that the solutions of stiff systems of differential equations contain rapidly decaying transient terms.

Example 10.38 As an example, consider the infinitesimal generator

$$Q = \begin{pmatrix} -1 & 1 \\ 100 & -100 \end{pmatrix}$$

with initial probability vector $\pi_{(0)} = (1, 0)$. Q has the decomposition $Q = S\Lambda S^{-1}$, where

$$S = \begin{pmatrix} 1 & -.01 \\ 1 & 1 \end{pmatrix} \quad \text{and} \quad \Lambda = \begin{pmatrix} 0 & 0 \\ 0 & -101 \end{pmatrix}.$$

Since $\pi_{(t)} = \pi_{(0)} e^{Qt} = \pi_{(0)} S e^{\Lambda t} S^{-1}$, we find that

$$\pi_{(t)} = \frac{1}{1.01}\left(e^0 + .01e^{-101t},\ .01e^0 - .01e^{-101t}\right).$$

The exponents e^{-101t} in this expression tend rapidly to zero, leaving the stationary probability distribution. In spite of this, however, when an explicit method is used, small step sizes must be used over the entire period of integration. It is not possible to increase the step size once the terms in e^{-101t} are effectively zero.

The classical definition asserts that a system of ODEs is stiff when certain eigenvalues of the Jacobian matrix (with elements $\partial f_i/\partial y_i$) have large negative real parts when compared to others. Using this definition, Markov chain problems (where the Jacobian is given by Q, the infinitesimal generator) are stiff when $\max_k |\text{Re}(\lambda_k)| \gg 0$. The Gerschgorin disk theorem is useful in bounding this quantity. From the special properties of infinitesimal generator matrices, this theorem implies that

$$\max_k |\text{Re}(\lambda_k)| \le 2 \max_j |q_{jj}|,$$

i.e., twice the largest total exit rate from any one state in the Markov chain.

Multistep BDF Methods

The so-called *BDF (backward differentiation formulae)* methods [16] are a class of multistep methods that are widely used for stiff systems. Only implicit versions are in current use, since low-order explicit versions correspond to the explicit Euler method ($k = 1$) and the midpoint rule ($k = 2$), and higher-order explicit versions ($k \ge 3$) are not stable. Implicit versions are constructed by generating an interpolating polynomial $z(t)$ through the points (t_j, y_j) for $j = i-k+1, \ldots, i+1$. However, now y_{i+1} is determined so that $z(t)$ satisfies the differential equation at t_{i+1}, i.e., in such a way that $z'(t_{i+1}) = f(t_{i+1}, y_{i+1})$. It is usual to express the interpolating polynomial in terms of backward differences

$$\nabla^0 f_i = f_i, \quad \nabla^{j+1} f_i = \nabla^j f_i - \nabla^j f_{i-1}.$$

With this notation, the general formula for implicit BDF methods is

$$\sum_{j=1}^{k} \frac{1}{j} \nabla^j y_{i+1} = h f_{i+1}$$

and the first three rules are

$$\begin{aligned} k = 1:&\quad y_{i+1} - y_i = h f_{i+1} && \text{(implicit Euler)},\\ k = 2:&\quad 3y_{i+1}/2 - 2y_i + y_{i-1}/2 = h f_{i+1},\\ k = 3:&\quad 11y_{i+1}/6 - 3y_i + 3y_{i-1}/2 - y_{i-2}/3 = h f_{i+1}. \end{aligned}$$

The BDF formulae are known to be stable for $k \le 6$ and to be unstable for other values of k. When applied to Markov chains, the implicit ($k = 2$) BDF formula is

$$3\pi_{(i+1)}/2 - 2\pi_{(i)} + \pi_{(i-1)}/2 = h\pi_{(i+1)}Q,$$

so that the system of equations to be solved at each step is

$$\pi_{(i+1)}(3I/2 - hQ) = 2\pi_{(i)} - \pi_{(i-1)}/2. \tag{10.62}$$

The solutions at two prior points $\pi_{(i-1)}$ and $\pi_{(i)}$ are used in the computation of $\pi_{(i+1)}$. To get the procedure started, the initial starting point $\pi_{(0)}$ is used to generate $\pi_{(1)}$ using the trapezoid rule, and then both $\pi_{(0)}$ and $\pi_{(1)}$ are used to generate $\pi_{(2)}$, using Equation (10.62).

Example 10.39 We shall apply the implicit ($k = 2$) BDF formula to the same continuous-time Markov chain that we have used in previous ODE examples. When applying the trapezoid rule in Example 10.34, we found $\pi_{(1)} = (.832370,\ .072254,\ .095376)$. We now have the two starting points $\pi_{(0)}$ and $\pi_{(1)}$ needed to launch the implicit BDF formula. We first compute

$$S = (1.5I - .1Q)^{-1} = \begin{pmatrix} 1.7 & -.1 & -.1 \\ -.3 & 2.3 & -.5 \\ -.1 & -.2 & 1.8 \end{pmatrix}^{-1} = \begin{pmatrix} .595870 & .029485 & .041298 \\ .087021 & .449853 & .129794 \\ .042773 & .051622 & .572271 \end{pmatrix}.$$

Had the Markov chains been large, we would have computed an LU factorization and applied this decomposition to solve the appropriate system of equations instead of forming the inverse. We now proceed by computing

$$\begin{aligned}
\pi(.2) \approx \pi_{(2)} &= \left(2\pi_{(1)} - .5\pi_{(0)}\right) S = (.714768,\ .109213,\ .176019),\\
\pi(.3) \approx \pi_{(3)} &= \left(2\pi_{(2)} - .5\pi_{(1)}\right) S = (.623707,\ .127611,\ .239682),\\
&\ \vdots\\
\pi(1.0) \approx \pi_{(10)} &= \left(2\pi_{(9)} - .5\pi_{(8)}\right) S = (.454655,\ .153688,\ .391657),
\end{aligned}$$

and the final error is

$$\|\pi(1.0) - \pi_{(10)}\|_2 = .003687.$$

Matlab Code for ODE Methods

The following code segments may be used to experiment with ODE methods for obtaining transient distributions of relatively small Markov chains. In each case, the computed result is checked against the results obtained from the Matlab matrix exponential function, *expm*. The input to each function is the transition rate matrix Q, the initial vector $y_0 = \pi(0)$, the time t at which the distribution is to be found and the step size, h. The output is an approximation to the transient distribution at time t, i.e., $\pi(t)$, and an estimate of the error, *err*. Two methods are included that are not discussed in the section itself, an implicit Runge-Kutta method and an implicit Adam's method, both of which may be used on stiff problems.

```
****************************************************************
function [y,err] = euler(Q,y0,t,h);   %%%%%%%%%%%%%%  Explicit Euler

[n,n] = size(Q);  I = eye(n);  eta = t/h;   y = y0;
yex = y0*expm(t*Q);                       %%%% Solution using Matlab fcn

R = I + h*Q;
```

```
for i=1:eta,
    y = y*R;
end
err = norm(y-yex,2);

*********************************************************************
function [y,err] = ieuler(Q,y0,t,h);  %%%%%%%%%%%%%%%%%  Implicit Euler

[n,n] = size(Q);  I = eye(n);  eta = t/h;  y = y0;
yex = y0*expm(t*Q);                     %%%%% Solution using Matlab fcn

R = inv(I-h*Q);
for i=1:eta,
    y = y*R;
end
err = norm(y-yex,2);

*********************************************************************
function [y,err] = trap(Q,y0,t,h);     %%%%%%%  Trapezoid/Modified Euler

[n,n] = size(Q);  I = eye(n);  eta = t/h;  y = y0;
yex = y0*expm(t*Q);                     %%%%% Solution using Matlab fcn

R1 = I + h*Q/2;  R2 = inv(I-h*Q/2);  R = R1*R2;
for i=1:eta,
    y = y*R;
end
err = norm(y-yex,2);

*********************************************************************
function [y,err] = rk4(Q,y0,t,h);   %  Explicit Runge-Kutta --- Order 4

[n,n] = size(Q);  I = eye(n);  eta = t/h;   y = y0;
yex = y0*expm(t*Q);                   % Solution using Matlab fcn

for i=1:eta,
    k1 = y*Q;
    k2 = (y + h*k1/2)*Q;
    k3 = (y + h*k2/2)*Q;
    k4 = (y + h*k3)*Q;
    y = y + h*(k1 + 2*k2 + 2*k3 + k4)/6;
end
err = norm(y-yex,2);

*********************************************************************
function [y,err] = irk(Q,y0,t,h);   %%%%%%%%%%%%%  Implicit Runge-Kutta

[n,n] = size(Q);  I = eye(n);  eta = t/h;  y = y0;
yex = y0*expm(t*Q);                   %%%%%%% Solution using Matlab fcn
```

```
R1 = I + h*Q/3;  R2 = inv(I-2*h*Q/3+ h*h*Q*Q/6);  R = R1*R2;
for i=1:eta,
    y = y*R;
end
err = norm(y-yex,2);

**********************************************************************


function [y,err] = adams(Q,y0,t,h);   %%%%%%   Adams --- Implicit: k=2

[n,n] = size(Q);  I = eye(n);  eta = t/h;
yex = y0*expm(t*Q);                    %%%% Solution using Matlab fcn

R1 = I + h*Q/2;  R2 = inv(I-h*Q/2);
y1 = y0*R1*R2;                         %%%%% Trapezoid rule for step 1

S1 = (I+2*h*Q/3);  S2 =  h*Q/12;  S3 = inv(I-5*h*Q/12);
for i=2:eta,                           %%%%%%% Adams for steps 2 to end
    y = (y1*S1 - y0*S2)*S3;
    y0 = y1; y1 = y;
end
err = norm(y-yex,2);

**********************************************************************
function [y,err] = bdf(Q,y0,t,h);   %%%%%%%%%%  BDF --- Implicit: k=2

[n,n] = size(Q);  I = eye(n);  eta = t/h;
yex = y0*expm(t*Q);                  %%%%% Solution using Matlab fcn

R1 = I + h*Q/2;  R2 = inv(I-h*Q/2);
y1 = y0*R1*R2;                       %%%%%%%% Trapezoid rule for step 1

S1 = inv(3*I/2 - h*Q);
for i=2:eta,                         %%%%%%%%%%% BDF for steps 2 to end
    y = (2*y1 - y0/2)*S1;
    y0 = y1; y1 = y;
end
err = norm(y-yex,2);

**********************************************************************
```

10.8 Exercises

Exercise 10.1.1 State Gerschgorin's theorem and use it to prove that the eigenvalues of a stochastic matrix cannot exceed 1 in magnitude.

Exercise 10.1.2 Show that if λ is an eigenvalue of P and x its associated left-hand eigenvector, then $1 - \alpha(1 - \lambda)$ is an eigenvalue of $P(\alpha) = I - \alpha(I - P)$ and has the same left-hand eigenvector x, where

$\alpha \in \Re' \equiv (-\infty, \infty)\backslash\{0\}$ (i.e., the real line with zero deleted). Consider the matrix

$$P(\alpha) = \begin{pmatrix} -1.2 & 2.0 & 0.2 \\ 0.6 & -0.4 & 0.8 \\ 0.0 & 2.0 & -1.0 \end{pmatrix}$$

obtained from the previous procedure. What range of the parameter α results in the matrix P being stochastic? Compute the stationary distribution of P.

Exercise 10.1.3 Look up the Perron-Frobenius theorem from one of the standard texts. Use this theorem to show that the transition probability matrix of an *irreducible* Markov chain possesses a *simple unit* eigenvalue.

Exercise 10.1.4 Part of the Perron-Frobenius theorem provides information concerning the eigenvalues of transition probability matrices derived from periodic Markov chains. State this part of the Perron-Frobenius theorem and use it to compute all the eigenvalues of the following two stochastic matrices given that the first has an eigenvalue equal to 0 and the second has an eigenvalue equal to -0.4:

$$P_1 = \begin{pmatrix} 0 & 0 & 0 & .5 & 0 & 0 & 0 & .5 \\ .5 & 0 & 0 & 0 & .5 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 & 0 & .5 & 0 & 0 \\ 0 & 0 & .5 & 0 & 0 & 0 & .5 & 0 \\ 0 & 0 & 0 & .5 & 0 & 0 & 0 & .5 \\ .5 & 0 & 0 & 0 & .5 & 0 & 0 & 0 \\ 0 & .5 & 0 & 0 & 0 & .5 & 0 & 0 \\ 0 & 0 & .5 & 0 & 0 & 0 & .5 & 0 \end{pmatrix}, \quad P_2 = \begin{pmatrix} 0 & 0 & 0 & .8 & 0 & 0 & 0 & .2 \\ .7 & 0 & 0 & 0 & .3 & 0 & 0 & 0 \\ 0 & .6 & 0 & 0 & 0 & .4 & 0 & 0 \\ 0 & 0 & .5 & 0 & 0 & 0 & .5 & 0 \\ 0 & 0 & 0 & .4 & 0 & 0 & 0 & .6 \\ .7 & 0 & 0 & 0 & .3 & 0 & 0 & 0 \\ 0 & .8 & 0 & 0 & 0 & .2 & 0 & 0 \\ 0 & 0 & .9 & 0 & 0 & 0 & .1 & 0 \end{pmatrix}.$$

Exercise 10.2.1 The state transition diagram for a discrete-time Markov chain is shown in Figure 10.4. Use Gaussian elimination to find its stationary distribution.

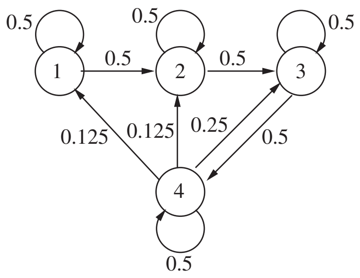

Figure 10.4. State transition diagram for Exercise 10.2.1.

Exercise 10.2.2 The state transition diagram for a continuous-time Markov chain is shown in Figure 10.5. Use Gaussian elimination to find its stationary distribution.

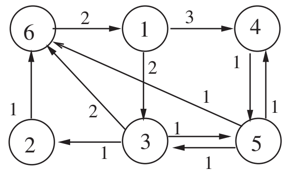

Figure 10.5. State transition diagram for Exercise 10.2.2.

Exercise 10.2.3 What is the Doolittle LU decomposition for the Markov chain of Exercise 10.2.2? Verify your answer by forming the product of L and U to obtain the infinitesimal generator matrix of this Markov chain.

Exercise 10.2.4 Use Gaussian elimination to find the stationary distribution of the continuous-time Markov chain whose transition rate matrix is

$$Q = \begin{pmatrix} -1 & 1 & & & & \\ & -2 & 2 & & & \\ & & -3 & 3 & & \\ & & & -4 & 4 & \\ & & & & -5 & 5 \\ 6 & & & & & -6 \end{pmatrix}.$$

Compare the number of multiplications/divisions needed if the singularity of the coefficient matrix is handled by

(a) replacing the first equation by $\pi e = 1$;
(b) replacing the last equation by $\pi e = 1$;
(c) replacing the row of zeros obtained during the final reduction step by $\pi_6 = 1$ and using a final normalization.

Exercise 10.2.5 Use Gaussian elimination to compute the stationary distribution of the discrete-time Markov chain whose transition probability matrix P is

$$P = \begin{pmatrix} 0 & 0 & .6 & 0 & 0 & .4 \\ 0 & .3 & 0 & 0 & 0 & .7 \\ 0 & 0 & 0 & .4 & .6 & 0 \\ 0 & .5 & 0 & .5 & 0 & 0 \\ .6 & .4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .8 & 0 & .2 \end{pmatrix}.$$

Is this stationary distribution a limiting distribution? Explain your answer. *Hint: Draw a picture.*

Exercise 10.2.6 Use Gaussian elimination to compute the stationary distribution of the continuous-time Markov chain whose infinitesimal generator Q is

$$Q = \begin{pmatrix} -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -5 & 0 & 3 & 0 & 2 & 0 \\ 0 & 0 & -3 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & -5 & 0 & 5 \\ 0 & 4 & 0 & 0 & 0 & -4 & 0 \\ 2 & 0 & 2 & 0 & 2 & 0 & -6 \end{pmatrix}.$$

Is the stationary distribution you compute unique? Explain your answer. *Hint: Draw a picture.*

Exercise 10.2.7 Use the Grassman-Taskar-Heyman (GTH) algorithm to compute the stationary distribution of the Markov chain whose transition rate matrix is

$$Q = \begin{pmatrix} -5 & 0 & 2 & 3 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 1 & -4 & 0 & 1 & 2 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 1 & 1 & -3 & 1 \\ 2 & 0 & 0 & 0 & 0 & -2 \end{pmatrix}.$$

Exercise 10.3.1 The stochastic transition probability matrix of a three-state Markov chain is given by

$$P = \begin{pmatrix} 0.0 & 0.8 & 0.2 \\ 0.0 & 0.1 & 0.9 \\ 0.6 & 0.0 & 0.4 \end{pmatrix}.$$

Carry out three iterations of the power method and the method of Jacobi using (0.5, 0.25, 0.25) as an initial approximation to the solution. Find the magnitude of the subdominant eigenvalue of the iteration matrix in both cases to see which will converge faster asymptotically. Approximately how many decimal places of accuracy would the subdominant eigenvalue suggest have been obtained after 100 iterations of both methods?

Exercise 10.3.2 Compute three iterations of the Gauss–Seidel method on the three-state Markov chain of Exercise 10.3.1 using the same initial approximation. Construct the Gauss–Seidel iteration matrix and compute the number of iterations needed to achieve ten decimal places of accuracy.

Exercise 10.3.3 The matrix P below is the stochastic transition probability matrix of a five-state Markov chain. Carry out three iterations of the power method, using an initial vector approximation whose components are all equal:

$$P = \begin{pmatrix} 0.2 & 0.0 & 0.005 & 0.795 & 0.0 \\ 0.0 & 0.0 & 0.998 & 0.002 & 0.0 \\ 0.002 & 0.0 & 0.0 & 0.0 & 0.998 \\ 0.8 & 0.001 & 0.0 & 0.198 & 0.001 \\ 0.0 & 0.998 & 0.0 & 0.002 & 0.0 \end{pmatrix}.$$

Exercise 10.3.4 Returning to the stochastic transition probability matrix of Exercise 10.3.3, and using the same initial starting approximation $x^{(0)}$ in which all components are equal, apply the standard (point) Gauss–Seidel method to obtain approximations $x^{(1)}$, $x^{(2)}$, and $x^{(3)}$.

Exercise 10.3.5 Given the stochastic transition probability matrix

$$P = \begin{pmatrix} 0.2 & 0.0 & 0.005 & 0.795 & 0.0 \\ 0.0 & 0.0 & 0.998 & 0.002 & 0.0 \\ 0.002 & 0.0 & 0.0 & 0.0 & 0.998 \\ 0.8 & 0.001 & 0.0 & 0.198 & 0.001 \\ 0.0 & 0.998 & 0.0 & 0.002 & 0.0 \end{pmatrix},$$

provide a Harwell-Boeing compact form for the corresponding transition rate matrix $Q^T = P^T - I$, as it might be used, for example, in the method of Gauss–Seidel.

Exercise 10.3.6 Consider the following stochastic transition probability matrix, provided in the Harwell-Boeing compact storage format.

.4	.6	.4	.4	.2	.2	.8
3	1	2	1	3	3	2
1	3	6	8			

Use the Gauss–Seidel iterative method and an initial approximation $x^{(0)} = (0, 1, 0)^T$ to compute $x^{(1)}$ and $x^{(2)}$.

Exercise 10.4.1 Consider the stochastic transition probability matrix of Exercise 10.3.3 which is in fact nearly completely decomposable. Permute this matrix to expose its NCD block structure and then perform two iterations of block Gauss–Seidel, beginning with an initial approximation in which all components are equal.

Exercise 10.5.1 Apply the NCD approximation procedure of Section 10.5 to the following stochastic transition probability matrix:

$$P = \begin{pmatrix} 0.2 & 0.0 & 0.005 & 0.795 & 0.0 \\ 0.0 & 0.0 & 0.998 & 0.002 & 0.0 \\ 0.002 & 0.0 & 0.0 & 0.0 & 0.998 \\ 0.8 & 0.001 & 0.0 & 0.198 & 0.001 \\ 0.0 & 0.998 & 0.0 & 0.002 & 0.0 \end{pmatrix}.$$

Exercise 10.5.2 Apply one complete step of the IAD method of Section 10.5 to the transition probability matrix of Exercise 10.5.1. Choose an equiprobable distribution as your starting approximation.

Exercise 10.6.1 Prove Equation (10.32), i.e., for any matrix R with spectral radius strictly less than 1 show that

$$\sum_{i=0}^{\infty} R^i = (I - R)^{-1}.$$

Exercise 10.6.2 Implement the logarithmic reduction algorithm outlined in the text. How many iterations are required to obtain the rate matrix R correct to ten decimal places, for a QBD process whose block matrices $A_0,\ A_1$, and A_2 are given below? Approximately how many iterations are required if the successive substitution algorithm is used?

$$A_0 = \begin{pmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 \end{pmatrix}, \quad A_1 = \begin{pmatrix} -16.0 & 12 & 0 & 0 \\ 0 & -16.0 & 12 & 0 \\ 0 & 0 & -16.0 & 12 \\ 0 & 0 & 0 & -16.0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 \end{pmatrix}.$$

Exercise 10.6.3 Consider a QBD process of the form

$$Q = \begin{pmatrix} B_{00} & A_2 & 0 & 0 & \cdots \\ A_0 & A_1 & A_2 & 0 & \cdots \\ 0 & A_0 & A_1 & A_2 & \\ \vdots & & \ddots & \ddots & \ddots \end{pmatrix},$$

where $A_0,\ A_1$, and A_2 are the same as in Exercise 10.6.2 and

$$B_{00} = \begin{pmatrix} -12 & 12 & 0 & 0 \\ 0 & -12 & 12 & 0 \\ 0 & 0 & -12 & 12 \\ 0 & 0 & 0 & -12 \end{pmatrix}.$$

Find the probability that this QBD process is in level 0 or 1.

Exercise 10.6.4 Consider a discrete-time Markov chain whose stochastic matrix P has the block structure

$$P = \begin{pmatrix} B_{00} & A_0 & & & & \\ B_{10} & A_1 & A_0 & & & \\ B_{20} & A_2 & A_1 & A_0 & & \\ & A_3 & A_2 & A_1 & A_0 & \\ \vdots & & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

with

$$A_0 = \begin{pmatrix} 1/8 & 1/32 \\ 1/16 & 1/16 \end{pmatrix}, \quad A_1 = \begin{pmatrix} 1/16 & 1/32 \\ 1/8 & 0 \end{pmatrix}, \quad A_1 = \begin{pmatrix} 0 & 1/4 \\ 1/4 & 0 \end{pmatrix}, \quad A_0 = \begin{pmatrix} 1/4 & 1/4 \\ 0 & 1/2 \end{pmatrix}$$

$$B_{00} = \begin{pmatrix} 3/4 & 3/32 \\ 3/8 & 1/2 \end{pmatrix}, \quad B_{10} = \begin{pmatrix} 1/2 & 1/4 \\ 1/4 & 1/2 \end{pmatrix}, \quad B_{20} = \begin{pmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{pmatrix}.$$

Beginning with $R_{(0)} = 0$, compute $R_{(1)}$ and $R_{(2)}$ using the successive substitution algorithm. What is the magnitude of the difference in successive iterations after 10 steps? Finally, compute the probability of the Markov chain being in a state of level 1.

Exercise 10.6.5 The transition rate matrix of a continuous-time Markov chain has the form

$$Q = \begin{pmatrix} B_{00} & B_{01} & & & & & & & \\ B_{10} & B_{11} & A_0 & & & & & & \\ B_{20} & 0 & A_1 & A_0 & & & & & \\ B_{30} & 0 & 0 & A_1 & A_0 & & & & \\ 0 & B_{41} & 0 & 0 & A_1 & A_0 & & & \\ 0 & 0 & A_4 & 0 & 0 & A_1 & A_0 & & \\ 0 & 0 & 0 & A_4 & 0 & 0 & A_1 & A_0 & \\ \vdots & \vdots & \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

with

$$A_0 = \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}, \quad A_1 = \begin{pmatrix} -12 & 0 \\ 0 & -10 \end{pmatrix}, \quad A_4 = \begin{pmatrix} 4 & 4 \\ 3 & 3 \end{pmatrix},$$

$$B_{00} = \begin{pmatrix} -8 & 6 \\ 2 & -4 \end{pmatrix}, \quad B_{01} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \quad B_{10} = \begin{pmatrix} 1 & 3 \\ 2 & 0 \end{pmatrix}, \quad B_{11} = \begin{pmatrix} -8 & 0 \\ 0 & -6 \end{pmatrix},$$

$$B_{20} = \begin{pmatrix} 0 & 8 \\ 6 & 0 \end{pmatrix}, \quad B_{30} = \begin{pmatrix} 4 & 4 \\ 1 & 5 \end{pmatrix}, \quad B_{41} = \begin{pmatrix} 6 & 2 \\ 3 & 3 \end{pmatrix}.$$

Construct Neuts' rate matrix R and the probability distribution of the states that constitute level 2.

Exercise 10.6.6 Which of the following three sequences of A blocks are taken from a positive-recurrent Markov chain of *M/G*/1 type? For those that are positive recurrent, compute the approximation to G that is obtained after 20 iterations using one of the three successive substitution strategies. Multiply your computed value of G on the right by the vector e and estimate the accuracy obtained.

(a)

$$A_0 = \begin{pmatrix} 1/4 & 1/4 & 0 \\ 0 & 0 & 0 \\ 0 & 1/4 & 1/4 \end{pmatrix}, \quad A_1 = \begin{pmatrix} 0 & 1/4 & 0 \\ 0 & 0 & 1/2 \\ 1/4 & 0 & 0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 1/8 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1/8 \end{pmatrix},$$

$$A_3 = \begin{pmatrix} 1/8 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1/8 \end{pmatrix}.$$

(b)

$$A_0 = \begin{pmatrix} 1/4 & 1/4 & 0 \\ 1/4 & 0 & 1/4 \\ 0 & 1/4 & 1/4 \end{pmatrix}, \quad A_1 = \begin{pmatrix} 0 & 1/4 & 0 \\ 0 & 0 & 1/2 \\ 1/4 & 0 & 0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 3/16 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 3/16 \end{pmatrix},$$

$$A_3 = \begin{pmatrix} 1/16 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1/16 \end{pmatrix}.$$

(c)

$$A_0 = \begin{pmatrix} 1/4 & 1/4 & 0 \\ 1/4 & 0 & 1/4 \\ 0 & 1/4 & 1/4 \end{pmatrix}, \quad A_1 = \begin{pmatrix} 0 & 1/4 & 0 \\ 0 & 0 & 1/4 \\ 1/4 & 0 & 0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 3/16 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 3/16 \end{pmatrix},$$

$$A_3 = \begin{pmatrix} 1/16 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1/16 \end{pmatrix}, \quad A_4 = \begin{pmatrix} 0 & 0 & 0 \\ 1/8 & 0 & 1/8 \\ 0 & 0 & 0 \end{pmatrix}.$$

Exercise 10.6.7 Consider a Markov chain whose infinitesimal generator is given by

$$Q = \left(\begin{array}{ccc|ccc|ccc|ccc|ccc|c}
-8 & 4 & 0 & 3 & 0 & 0 & 1 & 0 & 0 & & & & & & & \\
0 & -4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & & & & & & & \\
4 & 0 & -8 & 0 & 0 & 3 & 0 & 0 & 1 & & & & & & & \\
\hline
5 & 5 & 0 & -18 & 4 & 0 & 3 & 0 & 0 & 1 & 0 & 0 & & & & \\
5 & 0 & 5 & 0 & -14 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & & & & \\
0 & 5 & 5 & 4 & 0 & -18 & 0 & 0 & 3 & 0 & 0 & 1 & & & & \\
\hline
 & & & 5 & 5 & 0 & -18 & 4 & 0 & 3 & 0 & 0 & 1 & 0 & 0 & \\
 & & & 5 & 0 & 5 & 0 & -14 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & \\
 & & & 0 & 5 & 5 & 4 & 0 & -18 & 0 & 0 & 3 & 0 & 0 & 1 & \\
\hline
 & & & & & & & \ddots & & & \ddots & & & \ddots & & \ddots
\end{array}\right).$$

Verify that the states of this Markov chain are positive recurrent and find the probability that this Markov chain is in level 2.

Exercise 10.7.1 Given that it starts in state 1, find the transient distribution at time steps $n = 10$, 20, and 30 of the discrete-time Markov chain whose transition probability matrix is given by

$$P = \begin{pmatrix} .4 & 0 & .6 & 0 \\ .1 & .1 & .1 & .7 \\ .4 & .3 & .2 & .1 \\ 0 & .5 & 0 & .5 \end{pmatrix}.$$

Exercise 10.7.2 When the stationary distribution π of a discrete-time Markov chain with stochastic transition probability matrix P is required, we can find it from either $\pi P = \pi$ or $\pi Q = 0$, where $Q = P - I$. Can the same be said for computing transient distributions? Compare the transient distribution at time step $n = 10$ obtained in the previous question (using the same initial distribution $\pi^{(0)} = (1, 0, 0, 0)$) and the distribution obtained by forming $\pi^{(0)} e^{10Q}$.

Exercise 10.7.3 Write Matlab code to implement the Padé algorithm for computing e^{Qt}. Check your algorithm by testing it on the infinitesimal generator

$$Q = \left(\begin{array}{ccc|cc|ccc}
-.15 & .0 & .149 & .0009 & .0 & .00005 & .0 & .00005 \\
.1 & -.35 & .249 & .0 & .0009 & .00005 & .0 & .00005 \\
.1 & .8 & -.9004 & .0003 & .0 & .0 & .0001 & .0 \\
\hline
.0 & .0004 & .0 & -.3 & .2995 & .0 & .0001 & .0 \\
.0005 & .0 & .0004 & .399 & -.4 & .0001 & .0 & .0 \\
\hline
.0 & .00005 & .0 & .0 & .00005 & -.4 & .2499 & .15 \\
.00003 & .0 & .00003 & .00004 & .0 & .1 & -.2 & .0999 \\
.0 & .00005 & .0 & .0 & .00005 & .1999 & .25 & -.45
\end{array}\right)$$

and comparing it with the answer obtained by the Matlab function expm. Choose a value of $t = 10$.

Exercise 10.7.4 In the uniformization method, what value should be assigned to the integer K in Equation (10.48) so that an accuracy of 10^{-6} is achieved when computing transient distributions for the Markov chains with infinitesimal generators:

(a)

$$Q = \begin{pmatrix} -4 & 2 & 1 & 1 \\ 1 & -4 & 2 & 1 \\ 1 & 1 & -4 & 2 \\ 2 & 1 & 1 & -4 \end{pmatrix} \quad \text{at time } t = 3.0;$$

(b)

$$Q = \begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ .5 & .5 & -1.5 & .5 \\ 0 & 0 & 1 & -1 \end{pmatrix} \quad \text{at time } t = 6.0.$$

Exercise 10.7.5 Returning to the previous exercise, find the distribution at the specified times if each Markov chain begins in state 4.

Exercise 10.7.6 The infinitesimal generator of a continuous-time Markov chain is

$$Q = \begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ .5 & .5 & -1.5 & .5 \\ 0 & 0 & 1 & -1 \end{pmatrix}.$$

If this Markov chain begins in state 4, what is the transient distribution at time $t = 6$ computed by (a) the explicit Euler method and (b) the trapezoid method? Use a step size of $h = .25$ in both methods.

Exercise 10.7.7 In the text, a Runge-Kutta method of order 2 was described as having the form

$$y_{n+1} = y_n + ak_1 + bk_2$$

with

$$k_1 = hf(x_n, y_n), \quad k_2 = hf(x_n + \alpha h, y_n + \beta k_1).$$

Show that equating this with the Taylor series expansion of order 2 leads to the conditions

$$a + b = 1 \quad \text{and} \quad \alpha b = \beta b = 1/2.$$

Exercise 10.7.8 Write Matlab code to implement the implicit ($k = 3$) BDF method. Compare the results obtained with this method and the implicit ($k = 2$) BDF method when applied to a continuous-time Markov chain with infinitesimal generator given by

$$Q = \begin{pmatrix} -0.800 & 0.0 & 0.005 & 0.795 & 0.0 \\ 0.0 & -1.000 & 0.998 & 0.002 & 0.0 \\ 0.002 & 0.0 & -1.000 & 0.0 & 0.998 \\ 0.800 & 0.001 & 0.0 & -0.802 & 0.001 \\ 0.0 & 0.998 & 0.0 & 0.002 & -1.000 \end{pmatrix}.$$

Use a step size of $h = .25$, initial starting distribution of (1,0,0,0,0) and compute the transient distribution at time $t = 12$.