Nig. J. Pure & Appl. Sci. Vol. 35 (Issue 1, 2022)
e-ISSN 2756-4045
(C) 2022 Faculty of Physical Sciences and Faculty of Life Sciences, Univ. of Ilorin, Nigeria
www.njpas.com.ng

On the Application of Successive Over-relaxation Algorithmic and Block Numerical Iterative Solutions for the Stationary Distribution in Markov Chain

*[1]Agboola Sunday O. and [2]Ayinde Semiu A.

[1]Department of Mathematical Sciences, Faculty of Natural and Applied Sciences, Nigerian Army University Biu, No. 1 Biu – Gombe Road, PMB 1500 Biu, Borno State, Nigeria.
[2]Department of Basic Sciences (Mathematics Unit), Babcock University, Ilisan Remo, Ogun State

Date Received: 29-01-2022
Date Accepted: 13-03-2022
DOI: https://doi.org/10.48198/NJPAS/22.A02

ABSTRACT

The evolution of a system is represented by transitions from one state to the next, and the system's physical or mathematical behavior can also be depicted by defining all of the numerous states it can be in and demonstrating how it moves between them. In this study, the iterative solution methods for the stationary distribution of Markov chains were investigated, which start with an initial estimate of the solution vector and then alter it in such a way that it gets closer and closer to the genuine solution with each step or iteration., and also involved matrices operations such as multiplication with one or more vectors, which leaves the transition matrices unchanged and saves time. Our goal is to use Successive Overrelaxation Algorithmic and Block Numerical Iterative Solution Methods to compute the solutions. With the help of some existing Markov chain laws, theorems, and formulas, the normalization principle and matric operations such as lower, upper, and diagonal matrices are used. The stationary distribution vector's $\pi^{(k+1)} = \{\pi_1^{(k+1)} \quad \pi_2^{(k+1)} \quad \pi_3^{(k+1)} \quad \pi_4^{(k+1)} \quad \pi_5^{(k+1)}, \ k = 0, 1, 2, \dots, n\}$ are obtained for the illustrative examples, taken the initial stationary solution to be $\pi^{(0)} = (0.2 \quad 0.2 \quad 0.2 \quad 0.2 \quad 0.2)^T$ and it was observed that all subsequent iterations yield exactly the same result as $\pi^{(1)}$, and this shows that, the block iterative method requires only a single iteration to obtain the solution to full machine precision.

Keywords: Gauss-Seidel, infinitesimal generator, block iterative, compact storage, Successive over-relaxation

Introduction

In the discipline of numerical analysis, Iterative and direct solution methods are the two types of solution methods available. Iterative techniques begin with a rough estimate of the solution vector, which is subsequently tweaked until it gets closer to the true solution with each step or iteration. It eventually converges on the true solution. If there is no known initial approximation, a guess is performed or an arbitrary initial vector is used instead. The solution must be computed when a specified number of well-defined stages have been completed Stewart (2009). The most widely utilized methods for deriving the stationary probability vector from either the stochastic

Corresponding Author: **Agboola, Sunday .O.**
Department of Mathematical Sciences, Faculty of Natural and Applied Sciences, Nigerian Army University Biu, No. 1 Biu – Gombe Road, PMB 1500 Biu, Borno State, Nigeria.
Phone: +2348058222949; Email: agboolasunday70@gmail.com

transition probability matrix or the infinitesimal generator are iterative methods of one form or another. This decision was made for a variety of reasons. A look at the conventional iterative approaches reveals that the matrices are only involved in one operation: multiplication with one or more vectors, which leaves the transition matrices unchanged. When the transition matrix is large and not banded, direct techniques are generally not preferred due to the volume of fill-in that can quickly overwhelm available storage capacity. Romanovsky (1970) established the application and simulation of discrete Markov Chains while Ramaswami (1980, 1988) demonstrated stable recursion for the steady state vector in M/G/1 type Markov chains, which was followed by Stewart (1994, 2009) with the development of Numerical Solutions of Markov Chains, and Pesch *et al*. (2015) demonstrated the applicability of the Markov chain technique in Germany's wind feed. Uzun and Kiral (2017) utilized the Markov chain model of fuzzy state to forecast gold price movement and calculate the probabilistic transition matrix of gold price closing returns, whereas Aziza *et al.* (2019) used the Markov chain model of fuzzy state to forecast monthly rainfall data. Clemence (2019) demonstrated the application of Markov chain to the spread of disease infection, demonstrating that Hepatitis B became more infectious over time than tuberculosis and HIV, while Vermeer and Trilling (2020) demonstrated the application of Markov chain to journalism. Agboola (2021) introduced direct equation solving algorithms compositions of lower -upper triangular matrix and Grassmann–Taksar–Heyman for the stationary distribution of Markov chains while Agboola, and Ayoade (2021) analysed the matrix geometric and analytical block numerical iterative methods for stationary distribution in the structured Markov chains. Agboola and Ayinde (2021) demonstrated the performance measure analysis on the states classification in Markov chain while Agboola and Badmus (2021) established the application of renewal reward processes in homogeneous discrete Markov chain and, Agboola (2022) discussed the decomposition and aggregation algorithmic numerical iterative solution methods for the stationary distribution of Markov chain. Agboola, and Ayoade (2022) Analysed the block lower Hessenberg numerical iterative methods for stationary distribution in the structured Markov chains However, in this study, the successive overrelaxation method (SOR) and block numerical iterative solution methods and algorithms for computing the stationary distribution of Markov chain is considered.

Notation

H_ω Iteration matrix for successive overrelaxation method (SOR)

ω varied constant to detect various iterative method from the formulae

$\pi_i^{(k+1)}$ i^{th} component of the $(k+1)^{th}$ iteration for stationary distribution vector's

Materials and Methods

The study area consisted of the Application of Successive Overrelaxation Algorithmic and Block Numerical Iterative Solutions for the Stationary Distribution in Markov Chain. The Gauss–Seidel method looks similar to the successive overrelaxation method (SOR). The i^{th} component of the $(k+1)^{th}$iteration is obtained from $AY = B$, a linear system of *n* equations in *n* unknowns.

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

It is observed that the expression within the big parentheses on the right-hand side completely represents the Gauss–Seidel method, and that when ω is equal to 1, the successive overrelaxation method (SOR) simplifies to Gauss–Seidel. It's also possible to write a backward SOR relaxation. Overrelaxation is described as a process for $\omega > 1$; underrelaxation is described as a process for $\omega < 1$. As a result, the SOR technique converges only

if $0 < \omega < 1$ is true. Stewart (1994). This is a necessary condition for convergence, but it is not sufficient. Much research has gone into determining an optimal, or even a fair, value for ω, particularly for issues emerging in the numerical solution of partial differential equations. Although some findings have been established for specific classes of matrices, little is currently known regarding the best choice of ω for arbitrary nonsymmetric linear systems. If a series of similar experiments is to be undertaken, it may well be worthwhile to carry out some numerical tests to try to determine an appropriate value; some sort of adaptive technique might be added into the algorithm. For example, it is possible to begin iterating with a value of ω = 1 and, after numerous iterations have been carried out, to estimate the rate of convergence from the computed approximations. After a several more iterations, the value of ω may be increased by 1.1, and a new estimate of the rate of convergence can be produced. If this is better than previously, it should be increased again, perhaps to 1.2, and the same method followed. The value of ω should be reduced if the rate of convergence is not as good Stewart (2009) when applied to the homogeneous system

$$Q^T y = (E - L - U)y = 0, \qquad (2)$$

The SOR method becomes

$$y_i^{(k+1)} = a_{ii}\,(1-\omega)y_i^{(k)} + \omega\left\{\frac{1}{d_{ii}} - \sum_{j=1}^{i-1} l_{ij}y_j^{(k+1)} - \sum_{j=1+1}^{n} U_{ij}y_j^{(k)}\right\}, \quad i = 1, 2, \dots, n. \qquad (3)$$

or in matrix form

$$y^{(k+1)} = (1-\omega)y^{(k)} + \omega\left\{E^{-1}\left(Ly^{(k+1)} - Uy^{(k)}\right)\right\}$$

Re-arranging, we find

$$(E - \omega L)y^{(k+1)} = [(1-\omega)E + \omega U]y^{(k)} \qquad (4)$$

Or

$$y^{(k+1)} = (E - \omega L)^{-1}[(1-\omega)E + \omega U]y^{(k)} \qquad (5)$$

and thus, the iteration matrix for the SOR method is

$$H_\omega = (E - \omega L)^{-1}[(1-\omega)E + \omega U] \qquad (6)$$

It corresponds to the splitting

$$M = \omega^{-1}(E - \omega L) \quad \text{and} \quad N = \omega^{-1}[(1-\omega)E + \omega U]$$

The stationary probability vector is the eigenvector corresponding to a unit eigenvalue of the SOR iteration matrix, as shown in Equation (3). However, because the eigenvalues are dependent on the relaxation parameter ω, it is not always true that this unit eigenvalue is the dominant eigenvalue when using the SOR approach. There's a chance that H_ω has eigenvalues that are strictly bigger than 1. The SOR approach is identical to the power method applied to H_ω when the unit eigenvalue is the dominating eigenvalue. The relaxation parameter value that maximizes the difference between this unit eigenvalue and the subdominant eigenvalue of H_ω is the best choice, and the convergence rate achieved with this value of can be significantly better than Gauss–Seidel.

Data Structures for Large Sparse Matrices

This section concentrates on various algorithmic features that must be considered when using iterative approaches to solve large-scale Markov chains. When the transition matrix is larger than a few hundred, keeping it in a two-dimensional array, which is how we're used to seeing matrices, becomes impractical. Because each state can only reach a small number of states in a single step, the transition matrix is sparse in most cases, with only a few nonzero elements in each row. We'll look at ways to make use of the sparsity of this matrix to store it efficiently. One of the most significant advantages of iterative approaches over direct methods is that no changes to the elements of the transition matrix occur during the algorithm's execution. As a result, the matrix can be stored once and for all in a compact format without the need for additional processes to deal with insertions (due to nonzero entries becoming nonzero) and deletions (due to the elimination of nonzero elements). The

storage technique should not obstruct the numerical operations that must be performed on the matrix as a constraint. The pre- and post-multiplication of the matrix by a vector is the main numerical operation performed by the iterative methods we investigate, in fact, the only numerical operation performed on the matrix.

$$z = Ay \quad \text{and} \quad z = A^T y \tag{7}$$

One easy method is to store the nonzero components of the matrix in a real (double-precision) one-dimensional array aa, with two integer arrays ia and ja indicating the row and column positions of these items, respectively. We have $ia(k) = i$ and $ja(k) = j$ if the nonzero element a_{ij} is stored in position k of aa, i.e. $aa(k) = a_{ij}$.

(8)

Block Iterative Methods

To distinguish them from their block counterparts, the iterative methods we've looked at so far are commonly referred to as point iterative approaches. Block iterative methods are generalizations of point iterative methods, and they can be especially useful in Markov chain situations when the state space can be divided into subsets. In general, block iterative algorithms need more processing per iteration, but this is compensated by a higher convergence rate. Let's divide the defining homogeneous system of equations $\pi Q = 0$ into three parts.

$$(\pi_1 \quad \pi_2 \quad \pi_3 \quad \pi_4)\begin{pmatrix} Q_{11} & Q_{12} & \cdots & Q_{1n} \\ Q_{21} & Q_{22} & \cdots & Q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ Q_{n1} & Q_{n1} & \cdots & Q_{nn} \end{pmatrix} = 0.$$

We now introduce the block splitting

$$Q^T = E_n - (L_n - U_n) \tag{9}$$

We have E_n, a block diagonal matrix, L_n and U_n strictly lower and upper triangular block matrices, respectively.

$$E_n = \begin{pmatrix} E_{22} & 0 & \cdots & 0 \\ 0 & E_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & E_{nn} \end{pmatrix},$$

$$L_n = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ L_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{n1} & L_{n1} & \cdots & 0 \end{pmatrix},$$

$$U_n = \begin{pmatrix} 0 & U_{12} & \cdots & U_{1n} \\ 0 & 0 & \cdots & U_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix},$$

the block Gauss–Seidel method is given by

$$(E_n - L_n)\, y^{(k+1)} = \left(U_n y^{(k)}\right) \tag{10}$$

and corresponds to the splitting $M = (E_n - L_n);\ N = U_n$. The i^{th} block equation is given by

$$E_{ii} y_i^{(k+1)} = \left\{\sum_{j=1}^{i-1} L_{ij} y_j^{(k+1)} + \sum_{j=1+1}^{n} U_{ij} y_j^{(k)}\right\},\ \ i = 1, 2, \ldots, n. \tag{11}$$

where the sub-vectors y_i are partitioned conformally with $E_{ii},\ i = 1, 2, \ldots, n,$ This implies that at each iteration we must now solve *N* systems of linear equations

$$E_{ii} y_i^{(k+1)} = z_i,\ \ i = 1, 2, \ldots, n \tag{12}$$

$$z_i = \left\{\sum_{j=1}^{i-1} L_{ij} y_j^{(k+1)} + \sum_{j=1+1}^{n} U_{ij} y_j^{(k)}\right\},\ \ i = 1, 2, \ldots, n. \tag{13}$$

Before solving the i^{th} system, the right-hand side z_i can always be computed. The N systems of equations (13) are nonhomogeneous and have nonsingular coefficient matrices if our Markov chain is irreducible. To solve them, we can utilize either direct or iterative methods. Naturally, there is no obligation to solve all of the diagonal blocks using the same way. Instead, methods can be customized to fit specific block structures.

If a direct technique is utilized, a LU decomposition of block E_{ii} can be created once and for all before starting the iteration, reducing $E_{ii} y_i^{(k+1)} = z_i,\ \ i = 1, 2, \ldots, n,$ to a forward and backward substitution in each global iteration. This strategy may be

particularly efficient due to the nonzero structure of the blocks. If the diagonal blocks are themselves diagonal matrices, upper or lower triangular matrices, or even tridiagonal matrices, obtaining their LU decomposition is quite simple, and a block iterative technique becomes very appealing.

If the diagonal blocks are large and lack a suitable nonzero structure, it may be appropriate to solve these block equations using matrix iterative methods (such as point Gauss–Seidel), in which case we can have multiple inner (or local) iterative methods (one for each block thus analyzed) within an outer (or global) iteration. There are a few strategies that can be utilized to speed up the procedure. First, the solution computed at iteration k utilizing any block E ii should be utilized as an initial approximation to the solution computed at iteration k + 1. Second, obtaining a highly accurate answer in early (outer) iterations is rarely profitable. Until the global process begins to converge, we should only demand a modest number of digits of precision. One simple approach to accomplish this is to perform only a certain number of iterations for each inner solution. This will not provide much accuracy at initially, but when combined with the first recommendation, the accuracy will improve as you progress through the outer iterations. Intuitively, the greater the block sizes (and consequently the lower the number of blocks) for a given transition rate matrix Q, the fewer the number of (outer) iterations required for convergence.

The approach degenerates to a regular direct method in the exceptional case of only one block, and we compute the answer in a single "iteration." The decrease in the number of iterations associated with larger blocks is partially countered by an increase in the number of operations that must be completed at each iteration. However, it may be demonstrated that there is no growth in some significant cases. When the matrix is block tridiagonal (as in quasi-birth-death processes) and the diagonal blocks are similarly tridiagonal, it can be proven that both point and block iterative approaches have the same computational effort each iteration. In this scenario, the block approaches are extremely efficient because to the reduced amount of iterations. We can define a block Jacobi method in the same way that we can define a block Gauss–Seidel method.

$$E_{ii}y_i^{(k+1)} = \left\{\sum_{j=1}^{i-1} L_{ij}y_j^{(k+1)} + \sum_{j=1+1}^{n} U_{ij}y_j^{(k)}\right\}, \quad i = 1, 2, \dots, n. \tag{14}$$

and a block SOR method

$$y_i^{(k+1)} = (1-\omega)y_i^{(k)} + \omega\left\{E_{ii}^{-1}\left(\sum_{j=1}^{i-1} L_{ij}y_j^{(k+1)} + \sum_{j=1+1}^{n} U_{ij}y_j^{(k)}\right)\right\}, \quad i = 1, 2, \dots, n. \tag{15}$$

Results and Discussion

This section discusses the application of formulae for performance measures as well as composition of algorithms for Successive Overrelaxation Algorithmic and Block Numerical Iterative Solutions for the Stationary Distribution in Markov Chain which are demonstrated with illustrative examples.

Illustrative example 1: Considering the (4 × 4) matrix A given by

$$A = \begin{pmatrix} -3.2 & 0.0 & 1.5 & 0.6 \\ 0.6 & -0.9 & 0.0 & 0.0 \\ 0.4 & 1.8 & -1.5 & 0.0 \\ 0.0 & 0.5 & 0.7 & -0.6 \end{pmatrix}$$

It may be stored as

aa:	−3.2	1.5	0.6	−0.9	0.6	−1.5	0.4	1.8	−0.6	0.5	0.7
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:	−3.2	0.6	0.4	−0.9	1.8	0.5	1.5	−1.5	0.7	0.6	−0.6
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:	−3.2	−0.9	−1.5	−0.6	1.5	0.6	0.6	0.7	0.5	1.8
ia:	1	2	3	4	1	2	1	4	4	3
ja:	1	2	3	4	3	1	4	3	2	2

and so forth. The matrix is stored by rows in the first case, by columns in the second case, and in a random form in the third situation (although diagonal elements are given first). The following algorithm, in which n z specifies the number of nonzero elements in A, computes the product $z = Ay$. regardless of the order in which the components of the matrix A are entered into *aa*.

Algorithm 1: 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 $z(nrow) = z(nrow) + aa(next) \times y(ncol)$.

It's as simple as swapping the arrays ia and ja to get the product $z = A^T y$. 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 by $y_j$and forms one term of the inner product $\sum_{j=1}^{n} a_{ij} y_j = z_j$, 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. If a partial ordering is imposed on the positions of the nonzero elements in the array aa, a more efficient storing technique can be developed. Consider the scenario when the matrix's nonzero components are kept in rows; elements from row i precede those from row $i + 1$, but elements within a row may or may not be in order. This is common with Markov chains, because it's common to generate all the states that can be reached in a single step from a given state i before generating the states that can be reached from the next state, $i + 1$. As a result, the matrix is created row by row. It is feasible to eliminate the integer array ia and replace it with a smaller array when the nonzero elements are stored by rows in this manner. The items of ia are utilized as pointers into the arrays *aa* and *ja* in the most typical compact storage technique. The k^{th} element of ia indicates where the first element of row k is kept in aa and *ja*. As a result, $ia(1) = 1$. is always true. In addition, the first empty position of aa and ja is usually stored in position (n + 1) of ia. This usually means that $ia(n + 1) = n_z + 1$. The number of nonzero elements in row i is then given by $ia(i + 1) - ia(i)$. Even though the matrix is kept in a compact form, this makes it straightforward to jump to any row—the $ia(i + 1) - ia(i)$ nonzero components of row I begin at $aa[ia(i)]$. The Harwell-Boeing format is a row-wise packing system that is sometimes used.

Illustrative Example 2: Consider, once again, the same 4 × 4 matrix in Example 1

$$A = \begin{pmatrix} -3.2 & 0.0 & 1.5 & 0.6 \\ 0.6 & -0.9 & 0.0 & 0.0 \\ 0.4 & 1.8 & -1.5 & 0.0 \\ 0.0 & 0.5 & 0.7 & -0.6 \end{pmatrix}$$

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

aa:	−3.2	1.5	0.6	−0.9	0.6	−1.5	0.4	1.8	−0.6	0.5	0.7
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 $z = Ay$ may be computed by

Algorithm 2: 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 $sum = sum + aa(j) \times y(ja(j))$.

• Set $z(i) = sum$.

It may have seemed that the SOR technique was numerically more difficult than the simple power method or Gauss–Seidel, and that including a sparse storage data structure would be more difficult. This is not the case, however. The SOR method, like the power method and the Jacobi or Gauss–Seidel methods, requires simply a matrix-vector multiplication per iteration. the formula is utilized when programming SOR.

$$y_i^{(k+1)} = (1-\omega)y_i^{(k)} + \frac{\omega}{a_{ii}}\left\{b_i - \sum_{j=1}^{i-1} a_{ij}y_j^{(k+1)} - \sum_{j=1+1}^{n} a_{ij}y_j^{(k)}\right\}, \quad i = 1, 2, \dots, n. \qquad (16)$$

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

$$y_i^{(k+1)} = (1-\omega)y_i^{(k)} - \omega\left\{\sum_{j=1}^{i-1} a_{ij}y_j^{(k+1)} + \sum_{j=1+1}^{n} a_{ij}y_j^{(k)}\right\}, \quad i = 1, 2, \dots, n. \qquad (17)$$

$$y_i^{(k+1)} = y_i^{(k)} - \omega\left\{\sum_{j=1}^{i-1} a_{ij}y_j^{(k+1)} + a_{ii}y_j^{(k)} + \sum_{j=1+1}^{n} a_{ij}y_j^{(k)}\right\}, \qquad (18)$$

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

Algorithm 3: 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 $sum = sum + aa(j) \times y(ja(j))$.

• Compute $y(i) = y(i) - \omega \times sum$.

The only difference between this and Algorithm 2, which is a simple matrix-vector multiply algorithm, is in the very last line. The elements of y are generated sequentially in the SOR method, allowing them to be overwritten with fresh values as soon as they are computed. After the new value of $y(i)$ has been computed, the computation of element $y(i+1)$ begins. The matrix A above must be replaced with Q^T, the transpose of the infinitesimal generator, when utilizing the SOR procedure to derive the stationary distribution of a Markov chain. If Q is formed by rows, as is typically the case, this could be an issue. If it is not possible to create Q by columns (that is, for each state, we must locate the states that may access that state in a single step), the matrix must be transposed without being expanded into a full two-dimensional format. If there is enough storage for the compacted matrix and its compacted transpose, the transposition operation can be done in $O(n_z)$ operations, where n_z is the number of nonzero items stored. If there isn't enough room for a second compressed copy, the transposition can be done in situ using a conventional sorting technique in $O(n_z \log n_z)$operations. Obviously, the moral of the story is to try to store the matrix Q as columns. Unfortunately, in many Markov chain applications, determining all destination states that arise from a given source state (row-wise generation) is far more convenient than determining all source states that lead to a particular destination state (column-wise generation).

Illustrative example 3: We apply the block Gauss–Seidel method to find the stationary distribution of the continuous-time Markov chain with infinitesimal generator given by

$$Q = \begin{pmatrix} -4 & 2.0 & 1.0 & 0.5 & 0.5 \\ 0.0 & -3 & 3.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -1 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 & -5 & 4.0 \\ 1.0 & 0.0 & 0.0 & 1.0 & -2 \end{pmatrix}$$

We put the first three states in the first subset and the remaining two states into a second subset.

Transposing Q and writing out the system of equations, we have

$$Q^T Y = \begin{pmatrix} D_{11} & -U_{12} \\ -L_{21} & D_{22} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$
$$= \begin{pmatrix} -4 & 0.0 & 0.0 & 1.0 & 1.0 \\ 2.0 & -3 & 0.0 & 0.0 & 0.0 \\ 1.0 & 3.0 & -1 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.0 & -5 & 1.0 \\ 0.5 & 0.0 & 1.0 & 4.0 & -2 \end{pmatrix} \begin{pmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \\ \pi_4 \\ \pi_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Equation (15) becomes

$$E_{ii} y_i^{(k+1)} = \left\{ \sum_{j=1}^{i-1} L_{ij} y_j^{(k+1)} + \sum_{j=i+1}^{2} U_{ij} y_j^{(k)} \right\}, \quad i = 1, 2, \ldots, n. \tag{19}$$

and leads to the two block equations

$$i = 1: \quad E_{11} y_1^{(k+1)} = \left\{ \sum_{j=1}^{0} L_{1j} y_j^{(k+1)} + \sum_{j=2}^{2} U_{1j} y_j^{(k)} \right\} = U_{12} y_2^{(k)}, \tag{20}$$

$$i = 1: \quad E_{22} y_2^{(k+1)} = \left\{ \sum_{j=1}^{1} L_{2j} y_j^{(k+1)} + \sum_{j=3}^{2} U_{2j} y_j^{(k)} \right\} = L_{21} y_1^{(k+1)}. \tag{21}$$

Writing these block equations in full, they become

$$\begin{pmatrix} -4 & 0 & 0 \\ 2 & -3 & 0 \\ 1 & 3 & -1 \end{pmatrix} \begin{pmatrix} \pi_1^{(k+1)} \\ \pi_2^{(k+1)} \\ \pi_3^{(k+1)} \end{pmatrix} = - \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \pi_4^{(k)} \\ \pi_5^{(k)} \end{pmatrix}$$

and

$$\begin{pmatrix} -5 & 1 \\ 4 & -2 \end{pmatrix} \begin{pmatrix} \pi_4^{(k+1)} \\ \pi_5^{(k+1)} \end{pmatrix} = - \begin{pmatrix} 0.5 & 0 & 0 \\ 0.5 & 0 & 0 \end{pmatrix} \begin{pmatrix} \pi_1^{(k+1)} \\ \pi_2^{(k+1)} \\ \pi_3^{(k+1)} \end{pmatrix}.$$

We'll use LU decompositions to solve these block equations. The first subsystem can be solved with only forward substitution because E_{11} is lower triangular:

$$E_{11} = \begin{pmatrix} -4 & 0 & 0 \\ 2 & -3 & 0 \\ 1 & 3 & -1 \end{pmatrix} = L \times I.$$

Forming an LU decomposition of the second subsystem, we have

$$E_{22} = \begin{pmatrix} -5 & 1 \\ 4 & -2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -0.8 & 1 \end{pmatrix} \begin{pmatrix} -5 & 1 \\ 0 & -1.2 \end{pmatrix} = L \times U.$$

Taking the initial distribution to be

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

and substituting the first block equation, we find

$$\begin{pmatrix} -4 & 0 & 0 \\ 2 & -3 & 0 \\ 1 & 3 & -1 \end{pmatrix} \begin{pmatrix} \pi_1^{(1)} \\ \pi_2^{(1)} \\ \pi_3^{(1)} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0.2 \\ 0.2 \end{pmatrix} = \begin{pmatrix} 0.4 \\ 0.0 \\ 0.0 \end{pmatrix}.$$

Forward substitution successively gives

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

The second block system now becomes

$$\begin{pmatrix} 1 & 0 \\ -0.8 & 1 \end{pmatrix} \begin{pmatrix} -5 & 1 \\ 0 & -1.2 \end{pmatrix} \begin{pmatrix} \pi_4^{(1)} \\ \pi_5^{(1)} \end{pmatrix} = - \begin{pmatrix} 0.5 & 0 & 0 \\ 0.5 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.100 \\ 0.0667 \\ 0.300 \end{pmatrix} = - \begin{pmatrix} 0.05 \\ 0.35 \end{pmatrix}.$$

The answer is computed by first obtaining z from Lz = B and then Y from UY = z in the conventional form LUY = B. From

$$\begin{pmatrix} 1 & 0 \\ -0.8 & 1 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = - \begin{pmatrix} 0.05 \\ 0.35 \end{pmatrix},$$

we obtain

$$z_1 = -0.05 \quad \text{and} \quad z_2 = -0.39.$$

Now, solving

$$\begin{pmatrix} -5 & 1 \\ 0 & -1.2 \end{pmatrix} \begin{pmatrix} \pi_4^{(1)} \\ \pi_5^{(1)} \end{pmatrix} = \begin{pmatrix} -0.05 \\ -0.39 \end{pmatrix},$$

We find

$$\pi_4^{(1)} = 0.075 \quad \text{and} \quad \pi_5^{(1)} = 0.325.$$

We now have

$$\pi^{(1)} = (0.100 \quad 0.0667 \quad 0.300 \quad 0.075 \quad 0.325).$$

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

$$\pi^{(1)} = (0.1154 \quad 0.0769 \quad 0.3462 \quad 0.0865 \quad 0.375),$$

$$\pi^{(2)} = (0.1154 \quad 0.0769 \quad 0.3462 \quad 0.0865 \quad 0.375),$$

$$\vdots$$

$$\pi^{(20)} = (0.1154 \quad 0.0769 \quad 0.3462 \quad 0.0865 \quad 0.375)$$

$$\vdots$$

$$\pi^{(50)} = (0.1154 \quad 0.0769 \quad 0.3462 \quad 0.0865 \quad 0.375)$$

The result is the same in all subsequent iterations. Indeed, the block iterative technique just requires a single iteration to find the solution to full machine accuracy in this case. It is possible to conclude that the iteration matrix has an eigenvalue of 1 and four equals 0, which explains why convergence occurs in a single iteration.

Conclusions

The Successive Overrelaxation Algorithmic and Block Numerical Iterative Solution Methods for the stationary distribution of Markov chains which start with an initial estimate of the solution vector and then alter it in such a way that it gets closer and closer to the genuine solution with each step or iteration, has been investigated, in order to provide some insight into the solutions of stationary distribution of Markov chain. Normalization principle, Matrix operations such as Lower, upper and diagonal matrices are used with the help of some existing laws, theorems and formulas of Markov chain, while the stationary distribution vector's

$\pi^{(k+1)} = \{\pi_1^{(k+1)} \quad \pi_2^{(k+1)} \quad \pi_3^{(k+1)} \quad \pi_4^{(k+1)} \quad \pi_5^{(k+1)}, k = 0, 1, 2, \dots, n\}$ are obtained for the illustrative examples, taken the initial stationary solution to be $\pi^{(0)} = (0.2 \quad 0.2 \quad 0.2 \quad 0.2 \quad 0.2)^T$, and it was observed that all subsequent iterations yield exactly the same result as $\pi^{(1)}$. Which shows that, the block iterative method requires only a single iteration to obtain the solution to full machine precision.

References

Agboola, S. O. (2016). Repairman problem with multiple batch deterministic repairs, Unpublished Ph.D. Thesis, Obafemi Awolowo University, Ile-Ife, Nigeria, 256pp.

Agboola, S. O. (2021). Direct Equation Solving methods Algorithms Compositions of Lower -Upper Triangular Matrix and Grassmann–Taksar–Heyman for the stationary Distribution of Markov chains, *International Journal of Applied Science and Mathematics (IJASM),* 8(6): 87 – 96. www.ijasm.org.

Agboola, S. O. (2022). The Decomposition and Aggregation Algorithmic Numerical Iterative Solution Methods for the Stationary Distribution of Markov Chain, *Journal of Scientific and Engineering, 9(1): 116 - 123. CODEN (USA): JSERBR.* www.jsaer.com.

Agboola, S. O. and Ayoade, A. A. (2021). On the Analysis of Matrix Geometric and Analytical Block Numerical Iterative Methods for Stationary Distribution in the Structured Markov Chains, *International Journal of Contemporary Applied Researches (IJCAR),* 8(11): 51 – 65, Turkey, http://www.ijcar.net

Agboola, Sunday O. and Ayoade, Abayomi A. (2022). On the Analysis of Block Lower Hess Enberg Numerical Iterative Methods for Stationary Distribution in the Structured Markov Chains, *International Journal of Engineering Research and Applications, 12(1): 07 – 14.* www.ijera.com

Agboola, S. O. and Ayinde, S. A. (2021). The Performance Measure Analysis on the States Classification in Markov Chain, *Dutse Journal of Pure and Applied Sciences (DUJOPAS), Faculty of Science Journal, Federal University Dutse, Jigawa State.*7(4b): 19-29. https://fud.edu.ng/dujopas.

Agboola, S. O. and Badmus, N. I. (2021). Application of Renewal Reward Processes in Homogeneous Discrete Markov Chain,

FUDMA Journal of Sciences (FJS), Faculties of Earth Science and Physical Science Journal, Federal University DutsinMa, 5(4): 210 – 215. https://fjs.fudutsinma.edu.ng

Azizah, A., Welastica, R., Nur, F., Ruchjana, B. and Abdullah, A. (2019). An application of Markov chain for predicting rainfall data at West Java using data mining approach, *Earth and Environmental Science*, 303(1): 203 – 216.

Clemence. T. (2019). Markov chain modelling of HIV, Tuberculosis, and Hepatitis B transmission in Ghana, *Hindawi, Interdisciplinary Perspective on Infectious Disease*, 27(1)**:** 204 – 214.

Dayar., T. (1998). Permuting Markov chains to nearly completely decomposable form. *Technical Report BU-CEIS-9808*, Department of Computer Engineering and Information Science, Bilkent University, Ankara, Turkey. P 18 – 31.

Pesch, T., Schroder, S., Allelein, H. and Hake, J. (2015). A new Markov chain related statistical approach for modelling synthetic wind power time series, *New Journal of Physics, Dentsche Physikalishe,* 35(2)**:** 64 – 85.

Philippe, B and Sidje, B. (1993) *Transient Solution of Markov Processes by Krylov Subspaces*, Technical Report, *IRISA*— Campus de Beaulieu, Rennes, France. P 11 - 24.

Ramaswami, V. (1988). A Stable Recursion for the Steady State Vector in Markov chains of M/G/1 type. *Communication in Statist. Stochastic Models*, 4(1)**:** 183–188.

Ramaswami, V and Neuts, M. F. (1980). Some explicit formulas and computational methods for infinite server queues with phase type arrivals. *Journal of Applied Probability,* 17(1)**:** 498–514.

Romanovsky, V.I. (1970). *Discrete Markov Chains*, Wolters-Noord off, Groningen, Netherlands. P 23 – 44.

Stewart, W. J. (1994). *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, N.J. P 310 – 328.

Stewart, W. J. (2009). *Probability, Markov Chain, Queues and Simulation*, Princeton University Press, United Kingdom. P 1- 42.

Uzun, B. and Kiral, E. (2017). Application of Markov chain-fuzzy states to gold price, *Science Direct. ELSEVIER,* 120(1): 365 – 371.

Vermeer, S. and Trilling, D. (2020): Toward a better understanding of a new user journeys: A Markov chain approach. *Journalism Journal,* 21(1): 879 – 894.

Zakaria, N. N., Mahmod, O., Rajalmgan, S., Hamita, D., Lazim, A. and Evizal, A. (2019). Markov chain model development for forecasting air pollution index of Miri, Sarawak, *Sustainability* 11(1)**:** 5190 -5202.