The 7th Balkan Conference on Operational Research
"BACOR 05"
Constanta, May 2005, Romania

STEADY-STATE SOLUTIONS OF MARKOV CHAINS

DIMITAR RADEV
Department of Communication Technique & Technologies, University of Rousse, Bulgaria

VLADIMIR DENCHEV
ELENA RASHKOVA
Department of Communication Technique & Technologies, University of Rousse, Bulgaria

Abstract
The paper is devoted on methods and algorithms for steady-state analysis of Markov chains. Basic, direct and iterative methods for steady-state analysis of Markov chains are concerned, where Gaussian Elimination method and Grassman method, as well as Power, Jacobi's and Gauss-Seidel's methods are implemented. Algorithms for computation of steady-state probability vector for finite Markov chains are developed. Comparison of numerical solutions to exact equilibrium solution for local-balance equation of Discrete-Time Markov Chain is given. Example and numerical results for feedback networks of Markovian queues are shown.

Keywords*: Steady-State Probabilities, Queuing Theory, Discrete-Time Markov Chains, Numerical Methods, Approximation Techniques*

1. INTRODUCTION

Markov processes provide very flexible, powerful, and efficient means for description and analysis of dynamic (communication, computer) system properties. Performance and dependability measures for communication networks can be derived and evaluated with steady-state analysis of Discrete-Time Markov Chains (DTMC) and Continuous-Time Markov Chains (CTMC). Direct methods and iterative methods can be used for numerical solution steady-state analysis of Markov chains [1]. Direct methods operate and modify the parameter matrix, and use a fixed amount of computation time independent of the parameter values [4], but are subject to accumulation of round-off errors and have difficulties with sparse storage [2].

Iterative methods are based on the property of successive convergence to the desired solution. The evaluation can be terminated if iterates are sufficiently close to the exact value. The main advantage of iterative methods, compared with direct methods is that they preserve the sparsity of the parameter matrix [3], because efficient sparse storage schemes and efficient sparsity-preserving algorithms can be used. Other disadvantage of iterative methods is that convergence is not always guaranteed and depends on the method. The rate of convergence is highly sensitive to the values of entries in the parameter matrix [5].

One of important tasks here is receiving numerical solutions to exact equilibrium solution for local-balance equation of discrete-time Markov chain. This is very interesting for modeling of computer and communication networks, especially with heavy tailed traffic. These traffic processes are describing with discrete-time Markov chains, continuous-time Markov chains and ergodic Markov chains. That's why in this research is working out algorithms for numerical solution of equilibrium for local-balance equation of discrete-time Markov chain. On the base of numerical solution methods is suggesting a procedure for steady state probability vector.

2. STEADY-STATE ANALYSIS OF MARKOV CHAINS

For computation of steady-state probability vector of *ergodic* Markov chains most often is using the following model. Setting $\mathbf{v} = \mathbf{vP}$, and $\mathbf{0} = \boldsymbol{\pi}\mathbf{Q}$, can be written (2.1).

$$\mathbf{0} = \mathbf{v}(\mathbf{P} - \mathbf{I}) \tag{2.1}$$

Therefore, both for discrete-time and continuous-time Markov chains, a linear system (2.2) need to be solved:

$$\mathbf{0} = \mathbf{xA} \tag{2.2}$$

Due to its type of entries representing the parameters of a Markov chain, matrix $\mathbf{A}$ is singular and it can be shown that $\mathbf{A}$ is of rank n-1 for any Markov chain of size $|S| = n$. It follows immediately that the resulting set of equations is not linearly independent and that one of the equations is redundant. To yield a unique, positive solution, a normalization condition have to be applied on the solution $\mathbf{x}$ of equation $\mathbf{0} = \mathbf{xA}$. We directly impose the normalization condition into the (2.2) with (2.3).

$$\mathbf{x1} = 1 \tag{2.3}$$

This can be regarded as substituting one of the columns (say, the last column) of matrix $\mathbf{A}$ by the unit vector. The resulting linear system of non-homogeneous equations is shown in (2.4).

$$\mathbf{b} = \mathbf{xA}, \qquad \mathbf{b} = [0,0,...,0,1] \tag{2.4}$$

For any given ergodic continuous-time Markov chains, a discrete-time Markov chains can be constructed, which yields an identical steady-state probability vector as for the CTMC. Consider the generator matrix $\mathbf{Q} - [q_{ij}]$ of a continuous-time Markov chains, where is formulated (2.5),

$$\mathbf{P} = \mathbf{Q} / q + \mathbf{I} \tag{2.5}$$

where q is chosen such that $\mathbf{q} > \max_{i,j \in S} |q_{ij}|$. Setting $\mathbf{q} = \max_{i,j \in S} |q_{ij}|$ should be avoided in order to assure aperiodicity of the resulting DTMC [2]. The resulting matrix $\mathbf{P}$ can be used to determine the steady-state probability vector $\boldsymbol{\pi} = \mathbf{v}$, by solving $\mathbf{v} = \mathbf{vP} \quad and \quad \mathbf{v1} = 1$. This method, is used to reduce a CTMC to a DTMC, and is called randomization or sometimes uniformization in the literature [3]. On the other hand, a transition probability matrix $\mathbf{P}$ of an ergodic DTMC is given, and generator matrix $\mathbf{Q}$ of a CTMC can be defined according to (2.6).

$$\mathbf{Q} = \mathbf{P} - \mathbf{I} \tag{2.6}$$

By solving $\mathbf{0} = \boldsymbol{\pi}\mathbf{Q}$ under the condition $\boldsymbol{\pi}\mathbf{1} = 1$, the desired steady-state probability vector $\boldsymbol{\pi} = \mathbf{v}$ can be obtained.

To determine the steady-state probabilities of finite Markov chains, different approaches for the solution of a linear system of the form $\mathbf{0} = \mathbf{xA}$ are used. In this case both direct and iterative numerical methods and techniques can lead to closed-form results. While direct methods yield exact results, iterative methods are generally more efficient, both in time and space. Disadvantages of iterative methods are that for some of them no guarantee convergence given in general. Since iterative methods are considerably more efficient in solving Markov chains, they are commonly used for larger models. For smaller models with less than a few thousand states, direct methods are reliable and accurate. Though closed-form results are highly desirable, they can be obtained for only a small class of models that have some structure in their matrix.

3. DIRECT METHODS FOR NUMERICAL SOLUTION

The closed-form solution methods are applicable when Markov chains possess special structures. For Markov chains with a more general structure, we need to refer to numerical methods. There are two broad classes of numerical methods to solve the linear systems of equations: direct methods and iterative methods. Direct methods operate and modify the parameter matrix. They use a fixed amount of computation time independent of the parameter values and we don’t aim to reach convergence. The use of sparse storage is difficult since original zero entries can become non-zeros. Direct methods are also subject to accumulation of round-off errors.

There are many direct methods for the solution of a system of linear equations. Some of them are restricted to certain regular structures of the parameter matrix that are of less importance for Markov chains, since these structures generally cannot be assumed in the case of a Markov chain. Among the most commonly applied techniques are the Gaussian elimination algorithm and a derivative of it - Grassmann's algorithm. The original version of the algorithm is usually referred to algorithms of Grassmann, Taksar, and Heyman (GTH), which are based on a renewal argument [5]. There is a newer variant where a simple relation to the Gaussian elimination algorithm is done. The Gaussian elimination algorithm suffers sometimes from numerical difficulties created by subtractions of nearly equal numbers. It is exactly this property that is avoided by the GTH algorithms and its variant through reformulations relying on regenerative properties of Markov chains. Cancellation errors are conveniently avoided in this way.

3.1 GAUSSIAN ELIMINATION

The idea of the algorithm is to transform the system of equations (3.1), into an equivalent one by applying elementary operations on the parameter matrix that preserve the rank of the matrix.

$$\begin{aligned} a_{0,0}x_0 + a_{1,0}x_1 + \cdots + a_{n-1,0}x_{n-1} &= b_0, \\ a_{0,1}x_0 + a_{1,1}x_1 + \cdots + a_{n-1,1}x_{n-1} &= b_1, \\ &\vdots \\ a_{0,n-1}x_0 + a_{1,n-1}x_1 + \cdots + a_{n-1,n-1}x_{n-1} &= b_{n-1}. \end{aligned} \qquad (3.1)$$

As a result, an equivalent system of linear equations specified by (3.2) with a triangular matrix structure is derived, from which the desired solution **x**, which is identical to the solution of the original system can be obtained:

$$\begin{aligned} a_{0,0}^{(n-1)}x_0 &= b_0^{(n-1)}, \\ a_{0,1}^{(n-2)}x_0 + a_{1,1}^{(n-2)}x_1 &= b_1^{(n-2)}, \\ &\vdots \\ a_{0,n-1}^{(0)}x_0 + a_{1,n-1}^{(0)}x_1 + \cdots + a_{n-1,n-1}^{(0)}x_{n-1} &= b_{n-1}^{0}. \end{aligned} \qquad (3.2)$$

If the system of linear equations has been transformed into a triangular structure, the final results can be obtained by means of a straightforward substitution process.

To arrive at system (3.2), an elimination procedure first needs to be performed on the original system (3.1). Informally, the algorithm can be described as follows; first the n-th equation of (3.1) is solved for x_{n-1}, and then x_{n-1} is eliminated from all other *n*-1 equations. Next, the (*n*-1)th equation is used to solve for x_{n-2}, and, again, x_{n-2} is eliminated from the remaining *n*-2 equations, and so forth. Finally, (3.2) results, where $a_{i,j}^{(k)}$ denotes the coefficient of x_i in the $(j+1)$-th equation, obtained after the k-th elimination step.

The Gaussian elimination procedure takes advantage of elementary matrix operations that preserve the rank of the matrix. Such elementary operations correspond to interchanging of equations, multiplication of equations by a real-valued constant, and addition of a multiple of an equation to another equation. In matrix terms, the essential part of Gaussian elimination is provided by the factorization of the parameter matrix **A** into the components of an upper triangular matrix **U** and a lower triangular matrix **L**.
As a result of the factorization of the parameter matrix **A,** the computation of the result vector **x** can split into two simpler steps:

$$\mathbf{b} = \mathbf{xA} = \mathbf{xUL} = \mathbf{yL}. \qquad (3.3)$$

Now the Gaussian elimination algorithm can be summarized as follows from Fig. 3.1.

STEP 1*: Construct the parameter matrix* **A** *and the right-side vector* **b** *according to:*

$$\mathbf{b} = \mathbf{x}\mathbf{A}, \qquad \mathbf{b} = [0,0,...,0,1];$$

STEP 2*: Carry out elimination steps or, apply the standard algorithm to split the parameter matrix* **A** *into upper triangular matrix* **U** *and lower triangular matrix* **L** *such that* $\mathbf{A} = \mathbf{UL}$ *holds. Note that the parameters of* **U** *can be computed with:*

$$a_{ij}^{(k)} = \begin{cases} 0, & \begin{array}{l} j = n-k-1, n-k-2, ..., 0 \\ i = n-1, n-2, ..., n-k, \end{array} \\ a_{ij}^{(k-1)} - a_{n-k}^{(k-1)} \dfrac{a_{n-k,j}^{(k-1)}}{a_{n-k,n-k}^{(k-1)}}, & otherwise \end{cases},$$

and the computation of **L** *can be deliberately avoided.*

STEP 3*: Compute the intermediate results* **y** *according to* $\mathbf{yL} = \mathbf{b}$ *or, compute the intermediate results with the result from* $\mathbf{xU} = (b_0^{(n-1)}, b_1^{(n-2)}, ..., b_{n-1})$ *according to:*

$$b_j^{(k)} = b_j^{(k-1)} - b_{n-k}^{(k-1)} \frac{a_{n-k,j}^{(k-1)}}{a_{n-k,n-k}^{(k-1)}},$$

where $j = n-k-1, n-k, ..., 0.$

STEP 4: *Perform the substitution to yield the final result* **x** *according to* $\mathbf{xU} = \mathbf{y}$ *by applying the formulae:*

$$x_0 = \frac{b_0^{(n-1)}}{a_{0,0}^{(n-1)}},$$

$$x_j = \frac{b_j^{(n-j)}}{a_{j,j}^{(n-j)}} - \sum_{k=0}^{j-1} \frac{a_{k,j}^{(n-j)}}{a_{j,j}^{(n-j)}} x_k, \; j = 1,2,..., n-1.$$

Fig. 3.1 Algorithm for Gaussian Elimination

3.2 THE GRASSMANN ALGORITHM

Grassmann's algorithm is a numerically stable variant of the Gaussian elimination procedure. The algorithm completely avoids subtractions and it is therefore less sensitive to rounding and cancellation errors caused by the subtraction of nearly equal numbers. Grassmann's algorithm was originally introduced for the analysis of ergodic, discrete-time Markov chains *X* = {*Xn; n* = 0, 1,...} and was based on arguments from the theory of regenerative processes [3].

The transition rates of a new Markov chain, having one state less than the original one, are defined. This elimination step, i.e., the computation of $\overline{q}_{j,i}$ is achieved merely by adding non-negative quantities to originally non-negative values $q_{j,i}, j \neq i$. Only the diagonal elements $q_{i,i}$ and $\overline{q}_{i,i}$ are negative.

The elimination procedure is iteratively applied to the generator matrix with entries $q_{j,i}^{(k)}$ of stepwise reduced state spaces until an upper triangular matrix results, where $q_{j,i}^{(k)}$ denotes the matrix entries after having applied elimination step $k, 1 \leq k \leq n-1$. Finally, each element $q_{i,i}^{(n-1)}$ on the main diagonal is equal to -1.

The elimination is followed by a substitution process to express the relations of the state probabilities to each other. To yield the final state probability vector the normalization condition must be applied. Grassmann's algorithm is presented in terms of a CTMC generator matrix and the parameter matrix must initially be properly defined:

STEP 1: $\mathbf{A} = \begin{cases} \mathbf{Q}, & \mathit{forCTMC} \\ \mathbf{P} - \mathbf{I}, & \mathit{forDTMC} \end{cases}$

STEP 2: *For* $l = n-1, n-2, \ldots, 1$:

$$a_{j,i}^{(n-l)} = \begin{cases} \dfrac{a_{j,i}^{(n-l-1)}}{\sum_{m=0}^{l-1} a_{l,m}^{(n-l-1)}}, & j < l, \\ & i = l \\ a_{j,i}^{(n-l-1)} + \dfrac{a_{j,l}^{(n-l-1)} a_{l,i}^{(n-l-1)}}{\sum_{m=0}^{l-1} a_{l,m}^{(n-l-1)}}, & j \neq i, \\ & 1 \leq j, \\ & i \leq l-1 \\ -1, \quad j = i = l & \\ 0 \quad j = l, i < l. & \end{cases}$$

STEP 3: *For* $l = 1, 2, \ldots, n-1$:

$$x_l = \sum_{i=0}^{l-1} x_i a_{il}^{(n-l)}.$$

STEP 4: *For* $l = 0, 1, \ldots, n-1$:

$$\left.\begin{matrix} \pi_i \\ \nu_i \end{matrix}\right\} = \frac{x_i}{\sum_{j=0}^{n-1} x_j}.$$

Fig. 3.2 The Grassmann algorithm

In matrix notation, the parameter matrix **A** is decomposed into factors of an upper triangular matrix **U** and a lower triangular matrix **L** such that the following equations hold.

$$\mathbf{0} = \mathbf{xA} = \mathbf{xUL}. \tag{3.4}$$

Of course, any solution of $\mathbf{0} = \mathbf{xU}$ is also a solution of the original equation $\mathbf{0} = \mathbf{xA}$. Therefore, there is no need to represent **L** explicitly. Although cancellation errors are being avoided with Grassmann's algorithm, rounding errors can still occur, propagate, and accumulate during the computation. Therefore, applicability of the algorithm is also limited to medium size (around 500 states) Markov models.

4. ITERATIVE METHODS FOR NUMERICAL SOLUTION

The main advantage of iterative methods over direct methods is that they preserve the sparsity of the parameter matrix and efficient sparsity-preserving algorithms and sparse storage schemes can be used. A good initial estimate can speed up the computation considerably. The evaluation can be terminated if the iterates are sufficiently close to the exact value, i.e., a pre-specified tolerance is reached. Finally, because the parameter matrix is not changed in the iteration process, iterative methods are not subject to accumulation of round-off errors. The main disadvantage of iterative methods is that convergence is not always guaranteed and depending on the method, the rate of convergence is highly sensitive to the values of entries in the parameter matrix.

4.1. CONVERGENCE OF ITERATIVE METHODS

Convergence is a very important issue for iterative methods that must be dealt consciously. A heuristic approach can be applied for choosing appropriate techniques for decisions on convergence, but there are no general algorithms for the selection of such a technique. Because the desired solution vector is not known, an estimate of the error must be used to determine convergence. A tolerance level ε must be specified to provide a measure of how close the current iteration vector $x^{(k)}$ is to the desired solution vector **x**. New York, Some distance measures are often used to evaluate the current iteration vector $x^{(k)}$ in relation to some earlier iteration vectors $x^{(l)}, l < k$. If the current iteration vector is "close enough" to earlier ones with respect to ε, then this condition is taken as an indicator of convergence to the final result. If ε is too small, convergence could become very slow or not take place at all. If ε is too large, accuracy requirements could be violated or, worse, convergence could be wrongly assumed. Some appropriate norm functions have to be applied in order to compare different iteration vectors. Size and type of the parameter matrix should be taken into consideration for the right choice of such a norm function. Concerning the right choice of ε and the norm function, we can say that components x_i of the solution vector can differ significantly from each other.

4.2 POWER METHOD

The Power method is a reliable iterative method for the computation of the steady-state probability vector of finite ergodic Markov chains. It sometimes tends to converge slowly and the solely condition needed for convergence is the transition probability matrix **P** to be aperiodic, and then irreducibility is not necessary. The power method follows the transient behavior of the underlying discrete-time Markov chains until some stationary, not necessarily steady-state, convergence is reached. Therefore, it can also be used as a method for computing the transient state probability vector $\mathbf{v}(n)$ of a DTMC.

Equation $\mathbf{v} = \mathbf{vP}$ suggests starting with an initial guess of some probability vector $\mathbf{v}^{(0)}$ and repeatedly multiplying it by the transition probability matrix **P** until convergence to ***v*** is reached, with $\lim_{i\to\infty} \mathbf{v}^{(i)} = \mathbf{v}$. Since ergodicity, or at least aperiodicity of the underlying Markov chain are assumed, this procedure is guaranteed to converge to the desired fixed point of the unique steady-state probability vector. A single iteration step is as follows from (4.1).

$$\mathbf{v}^{(i+1)} = \mathbf{v}^{(i)}\mathbf{P}, \qquad i \geq 0 . \tag{4.1}$$

The relation between the iteration vector at step *i* and the initial probability vector can be presented as (4.2).

$$\mathbf{v}^{(i)} = \mathbf{v}^{(0)}\mathbf{P}^{i}, \qquad i \geq 0 . \tag{4.2}$$

To yield the final result of the steady-state probability vector ***v*** only a renormalization remains to be performed. The speed of convergence of the power method depends on the relative sizes of the eigenvalues. The closer non-dominant eigenvalues are equals to 1, which slower the convergence. The algorithm of the power method is shown on Fig. 3.1.

***STEP 1**: Select q appropriately:*

$$\mathbf{A} = \begin{cases} \mathbf{P}, \\ \mathbf{Q}/q + \mathbf{I}; \end{cases}$$

$$\mathbf{v}^{(0)} = \left(\mathbf{v}_0^{(0)}, \mathbf{v}_1^{(0)}, \ldots, \mathbf{v}_{n-1}^{(0)}\right).$$

Select convergence criterion ε, and let $n = 0$. Define some vector norm function $f\left(\left\|\mathbf{v}^{(n)}, \mathbf{v}^{(l)}\right\|\right), n \geq l$.

Set convergence = false.

***STEP 2**:Repeat until convergence:*

STEP 2.1: $\mathbf{v}^{(n+1)} = \mathbf{v}^{(n)}\mathbf{A}$ ***;***

STEP 2.2: If $f\left(\left\|\mathbf{v}^{(n+1)}, \mathbf{v}^{(l+1)}\right\|\right) < \varepsilon, l \leq n$

THEN convergence = true;

STEP 2.3: $n = n+1, l = l+1$.

***STEP 3**:* $\left.\begin{matrix}\boldsymbol{\pi} \\ \mathbf{v}\end{matrix}\right\} \approx \mathbf{v}^{(n)}$.

Fig. 4.1 The power method algorithm

4.3. JACOBI'S METHOD

Let define the system of linear equations (4.3).

$$\mathbf{b} = \mathbf{xA} . \tag{4.3}$$

The normalization condition may or may not be incorporated in (4.3). The parameters of both DTMC and CTMC are given by the entries of the matrix $\mathbf{A} = [a_{ij}]$. The solution vector **x** will contain the unconditional state probabilities. If the normalization is incorporated, we have $\mathbf{b} = [0,0,...,0,1]$, and $\mathbf{b} = 0$ otherwise. Consider the j-th equation from the system (4.3) as (4.4).

$$b_j = \sum_{i \in S} a_{ij} x_i \; . \tag{4.4}$$

Solving (4.4) for x_j leads to (4.5).

$$x_j = \frac{b_j - \sum_{i, i \neq j} a_{ij} x_i}{a_{jj}} . \tag{4.5}$$

Any given approximate solution $\hat{\mathbf{x}} = [\hat{x}_0, \hat{x}_1, ..., \hat{x}_{n-1}]$ can be inserted for the variables $x_i, i \neq j$, on the right side of (4.5). From these intermediate values, better estimates of the x_j on the left side of the equation may be obtained. The iterative method requires applying this procedure repeatedly and in parallel for all *n* equations. The values x^k of the k-th iteration step are computed from values obtained from the (*k* - l)-st step for each equation independently as (4.6).

$$x_j^{(k)} = \frac{b_j - \sum_{i, i \neq j} a_{ij} x_i^{(k-1)}}{a_{jj}} \qquad \forall j \in S \, . \tag{4.6}$$

The iteration may be started with an arbitrary initial vector $\mathbf{x}^0$. Note that the equations can be evaluated in parallel, a fact that can be used as a means for computational speed-up. The method is called method of simultaneous displacement or, simply, the Jacobi method. Since the method is quite simple, it suffers from poor convergence and hence is rarely applied in its raw form. The algorithm is presented on Fig. 4.2.

STEP 1: *Define parameter matrix* **A** *and* **b** *properly from generator matrix* **Q** *or transition probability matrix* **P**.

- ✓ *Choose initial vector* $\mathbf{x}^{(0)}$;
- ✓ *Choose convergence criterion* ε.
- ✓ *Choose some norm function* $f\left(\left\|\mathbf{x}^{(k)}, \mathbf{x}^{(l)}\right\|\right), k \geq l$.
- ✓ *Split parameter matrix* **A=D-L-U**.
- ✓ *convergence=false, and* $k = l = 1$.

STEP 2: *Repeat until convergence:*

STEP 2.1: $\mathbf{x}^{(k)} = \left(\mathbf{b} + \mathbf{x}^{(k-1)}(\mathbf{U} + \mathbf{L})\right)\mathbf{D}^{-1}$;

STEP 2.2: If $f\left(\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-l)}\right\|\right) < \varepsilon$

Then convergence =true;

Else $k = k + 1$ *and* $l \in \{1,...,k\}$.

STEP 3: $\left.\begin{matrix}\boldsymbol{\pi} \\ \mathbf{v}\end{matrix}\right\} \approx \frac{x^{(k)}}{\sum_{j=0}^{n-1} x_j^{(k)}}$.

Fig. 4.2 The Jacobi's method algorithm

Splitting the matrix **A=D-L-U** into its constituents of the diagonal matrix **D,** the strictly lower-triangular matrix **L**, and the strictly upper-triangular matrix **U** provide a way to present the main computation step of the Jacobi's method in matrix notation, as is shown in (4.7).

$$\mathbf{x}^{(k)} = \left(\mathbf{b} + \mathbf{x}^{(k-1)}(\mathbf{U} + \mathbf{L})\right)\mathbf{D}^{-1} . \tag{4.7}$$

The Jacobi's method is of less practical importance due to its slow pattern of convergence. But techniques have been derived to speed up its convergence, resulting in well-known algorithms such as Gauss-Seidel iteration.

4.4 GAUSS-SEIDEL METHOD

To improve convergence, a given method often needs to be changed only slightly. We can serialize the procedure from (4.6) and take advantage of the already updated new estimates in each step. Assuming the computations to be arranged in order $0,1,...,n-1,$, where $|S| = n$, it immediately follows, that for calculation of the estimates $x_j^{(k)}$, all j previously computed estimates $x_i^{(k)}, i < j$, can be used in the computation. Taking advantage of the more up-to-date information, we can significantly speed up the convergence. The resulting method is called the *Gauss-Seidel* iteration and it main principle is presented in (4.8).

$$x_j^{(k)} = \frac{b_j - \left(\sum_{i=0}^{j-1} a_{ij} x_i^{(k)} + \sum_{i=j+1}^{n-1} a_{ij} x_i^{(k-1)} + \right)}{a_{jj}}, \qquad \forall j \in S \, . \tag{4.8}$$

Note that the order in which the estimates $x_j^{(k)}$ are calculated in each iteration step can have a crucial impact on the speed of convergence. Most often, the matrices of Markov chains are sparse and the interdependencies between the equations are limited to a certain degree, and parallel evaluation might still be possible, even if the most up-to-date information is incorporated in each computation step. The equations can deliberately be arranged so that the interdependencies become more or less effective for the convergence process. Apparently, a trade-off exists between the pattern of convergence and possible speedup due to parallelism. In matrix notation, the Gauss-Seidel iteration step is written as (4.9).

$$\mathbf{x}^{(k)} = \left(\mathbf{b} + \mathbf{x}^{(k-1)}\mathbf{L}\right)(\mathbf{D} - \mathbf{U})^{-1}, \qquad k \geq 1 . \tag{4.9}$$

Reflecting the Gauss-Seidel step more obviously, we can rewrite his approach as (4.10).

$$\mathbf{x}^{(k)} = \left(\mathbf{b} + \mathbf{x}^{(k)}\mathbf{U} + \mathbf{x}^{(k-1)}\mathbf{L}\right)\mathbf{D}^{-1}, \qquad k \geq 1 . \tag{4.10}$$

5. COMPARISON OF NUMERICAL SOLUTION METHODS

5.1. EXAMPLE 1

We consider an arbitrary connected three-node network with four customers. The state transition rate diagram is shown in Fig. 5.1. In this diagram are represented possible transitions between nodes of Markov chain. The transition rates between the states are taken equal to $\mu_1 = 0.5 \quad \mu_2 = 0.4 \quad \mu_3 = 0.1$. The numeration of the states represents the total number of customers in each node. Consider receiving of local balance for two examples.

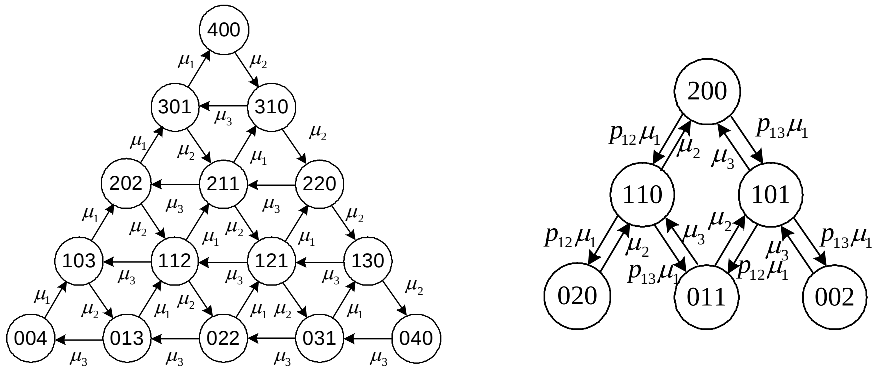

Fig. 5.1 State-transition rate diagram showing local balance for a) example 1 and b) example2

Firstly we write down the local balance equations, and then we find the solution, by a substitution process, from where we get the exact steady-state probabilities, as indicated in Table 5.1. Next we follow the algorithm in Fig 3.1 and achieve the results for the steady state probabilities from the Gaussian elimination algorithm. Comparing the results with the exact ones we may say that the Gaussian elimination algorithm gives precise results (to the eight decimal) and only in states 9 and 11 mistakes are found.

State	Number	Exact Value	Computed Value	Error
1	2	3	4	5
(0,0,4)	1	0,000962837	0,000962837	0
(0,1,3)	2	0,004814184	0,004814184	0
(0,2,2)	3	0,024070919	0,024070919	0
(0,3,1)	4	0,120354595	0,120354595	0
(0,4,0)	5	0,601772974	0,601772974	0
(1,3,0)	6	0,150446243	0,150446243	0
(2,2,0)	7	0,037510811	0,037510811	0
(3,1,0)	8	0,009402703	0,009402703	0
(4,0,0)	9	0,002350674	0,002350674	0.000000002
(3,0,1)	10	0,00188054	0,00188054	0
(2,0,2)	11	0,001504433	0,001504433	0.000000001
(1,0,3)	12	0,001203546	0,001203546	0
(1,1,2)	13	0,00601773	0,00601773	0
(1,2,1)	14	0,030088649	0,030088649	0
(2,1,1)	15	0,007522162	0,007522162	0

Table 5.1 Comparison of numerical results for calculated steady-state probabilities using exact method and Gaussian elimination

5.2 EXAMPLE 2

Consider two customers circulating among three nodes. When a customer has received service of mean duration $1/\mu_1$ at the first station, it queues with probability p_{12} at station two for service of mean duration $1/\mu_2$, or with p_{13} at station three with a mean service duration $1/\mu_3$. After completion of services at stations two or three, customers return with probability 1 back to station one. The state transition rate diagram is shown in Fig. 5.1 (b). In this diagram we represent a continuous-time Markov chain with possible transitions between nodes. The transition rates between the states are $\mu_1 = 1$ $\mu_2 = 2$ $\mu_3 = 3$ $p_{12} = 0.4$ $p_{13} = 0.6$. The numeration of the states represents the total number of customers in each node.

Firstly is computed the exact values for the steady state probabilities, using a substitution process. These values can be for comparison with the iteration vector. Next were used the power method algorithm to compute the steady state probabilities, reaching 45 iterations form where was received accuracy to the sixth decimal, as is shown in Table 5.2. Afterward from algorithm on Fig. 4.2 were computed probabilities according to Jacobi's method. The obtained results were better, but for their receiving it is necessary to provide much more iteration steps – 300. The power method converges faster for this network and gives results which are enough precise.

State	Exact	Power		Jacobi	
		$\nu = 45$	Error	$\nu = 300$	Error
(2,0,0)	0.6578947368	0.6578940535	-0,0000006833	0.6578947398	0,0000000030
(1,1,0)	0.1315789474	0.131579464	0,0000005166	0.1315789172	-0,0000000302
(0,2,0)	0.1315789474	0.131578764	-0,0000001834	0.1315789478	0,0000000004
(1,0,1)	0.02631578947	0.0263160561	0,0000002666	0.02631578619	-0,0000000033
(0,1,1)	0.02631578947	0.0263157728	-0,0000000167	0.02631578846	-0,0000000010
(0,0,2)	0.02631578947	0.0263159728	0,0000001833	0.02631582059	0,0000000311

Table 5.2 Comparison of numerical results for calculated steady-state probabilities using exact method, Power method and Jacobi's method

CONCLUSIONS

Basic direct and iterative methods for steady-state analysis of Markov chains are examined by Gaussian Elimination method and Grassman method, as well as Power, Jacobi's and Gauss-Seidel's method. Numerical results for two networks are compared, using exact methods, Gaussian elimination, Power and Jacobi's method. The Jacobi's method is of less practical importance due to its slow pattern of convergence. The Power method is a reliable iterative method, though it sometimes tends to converge slowly. It can also be used as a method for computing the transient state probability vector of a DTMC. Applicability of the Gaussian elimination algorithm is limited to medium size (around 500 states) Markov models and it suffers from round-off and cancellation errors, as well as numerical difficulties created by subtractions of nearly equal numbers, but gives good results.

BIBLIOGRAPHY

[1] Bhalai, S. (2002), Markov Decision Processes: the control of high-dimensional systems, Universal Press, Amsterdam, The Netherlands, 142 pp;
[2] Bolch, G., Greiner, S., Meer, H, Trivedi, K., (1998), Queueing Networks and Markov Chains: Modelling and Performance Evaluation with Computer Sciene Applications, New York, John Wiley& Sons, 726pp;
[3] Boxma, O., Koole, G., Liu, Z. (1994), "Queueing-theoretic solution methods for models of parallel and distributed systems", Performance Evaluation of Parallel and Distributed Systems - Solution Methods, CWI Tract 105 & 106, CWI, Amsterdam, The Netherlands, pp. 1-24;
[4] Radev, D., Denchev, V., Rashkova, E. (2005), "Approximations Algorithms for Steady-State Solutions of Markov Chains", Proceedings of the International Conference on Computer Systems and Technologies CompSysTech'2005, Varna, Bulgaria;
[5] Trivedi, K., (2001), Probability and Statistics with Reliability, Queuing, and Computer Science Applications, New York, John Willey & Sons, 830pp.