Generating of the Coefficient Matrix of the System of Homogeneous Differential Equations

Kirill S. Shardakov
Department of Information Systems and Technologies, Emperor Alexander I St. Petersburg State Transport University
St. Petersburg, Russia
k.shardakov@gmail.com

Vladimir P. Bubnov
Department of Information Systems and Technologies, Emperor Alexander I St. Petersburg State Transport University
St. Petersburg, Russia
bubnov1950@yandex.ru

Alexander N. Pavlov
Laboratory of Information Technologies in System Analysis and Modeling, St. Petersburg Institute for Informatics and Automation of the RAS,
St. Petersburg, Russia
pavlov62@list.ru

Abstract

In this paper the sequential algorithm for generating a matrix of coefficients for a system of homogeneous differential equations describing a model of a non-stationary queueing system is proposed. Its comparison with the recursive algorithm is given. The optimal storage structure of the list of states for a sequential algorithm is given. A decrease of the performing time for the algorithm compared with the recursive one was noted due to no need to sort the list of states and matrix of coefficients.

1 Introduction

Automated monitoring systems play an important role in the life of modern society. Monitoring is a continuous process of observing and recording the parameters of an object in comparison with the specified criteria. In some industries data is collected and accumulated very intensively. The mathematical basis for simulation of monitoring systems is queuing theory. Most authors use models of theory on the assumption that the task queue is infinite, there is a stationary mode, and the load factor does not exceed unity [Zeg12, Oso13, Upa16].

Great practical and theoretical interest are non-stationary queueing system (nSQS). There are few works devoted to this topic in comparison with works devoted to the stationary regime. Examples of such works are [Bub11, Bub10, Bub99]. The various models are discussed in detail in [Bub99]. The disadvantage of the work [Bub11, Bub10] is that they consider only the classical numerical method for solving systems of ordinary differential equations (ODE) - the Runge-Kutta method.

Later in [7–9] a numerical-analytical method is presented, the speed and accuracy of which, when solving an ODE system describing nSQS, are superior to the most common Runge-Kutta method for solving this kind of problems. One of the advantages of this method is the recursive algorithm for generating the matrix of coefficients of an ODE system without deriving the general equation of the ODE system. But this algorithm also has significant disadvantages: (1) the output of the algorithm provides an unordered list of states, the states in it are in the order of their recursive generation; (2) the output of the algorithm provides an array of the desired dimension instead of the lower triangular matrix; (3) based on the first two points, it follows the necessity of sorting the list of states and the matrix of coefficients to bring them into a ready for further use. When replacing the values in the list of states, it is necessary to swap rows and columns for the same state numbers in the matrix, and for large matrix sizes, this is a resource-intensive operation. To eliminate these disadvantages, a sequential algorithm for generating a matrix of coefficients is proposed, this algorithm is using a completely different approach.

In: B. V. Sokolov, A. D. Khomonenko, A. A. Bliudov (eds.): Selected Papers of the Workshop Computer Science and Engineering in the framework of the 5 th International Scientific-Methodical Conference "Problems of Mathematical and Natural-Scientific Training in Engineering Education", St.-Petersburg, Russia, 8–9 November, 2018, published at http://ceur-ws.org

2 Essence of a sequential algorithm

The essence of the sequential algorithm is to divide the list of states into subgroups with the same properties and further use these properties. For example, consider the simplest single-channel queueing system (QS) characterized by the number of tasks i (i=¯(0,N)) in it and the number of tasks j (j=¯(0,N-i)), that have already been completed, where N is the total number of tasks that can enter into the system. The input is sequentially received N tasks with intensities {λ1, λ2, … , λN}, which depends on the task number, and they are served with intensities{μ1, μ2, … , μN}, which also depends on the task number.

We divide states into groups, in a way that in each group with constant j, the value of i will grow to N-j. Note some important facts: (1) the number of groups will always be N+1; (2) the length of each next group will always be 1 less than the current; (3) within one group, states can transit into each other sequentially and only with intensity λ; (4) the transition with intensity μ can occur only from state i of group j to the state of the next group j + 1, while the number of the new state inside the group will always be i-1, the intensity of such a transition will always be μj.

The next step is generating a list of states with length Ns = (N+1)*(N+2)/2. The list is generated by simply iterating j in the outer loop, i in the inner loop, until i+j <= N. When this threshold is reached, j increases by one. At the output, the algorithm provides an ordered list of states and does it in a single loop pass.

An important component is the data structure for storing this list. The main list of states contains several groups, each of them is a separate list. Inside the group there are states, each of them is stored as a list containing a state number and a list with its description. This storage structure simplifies further filling of the matrix of coefficients and makes it possible not to go through the list of states when searching for the desired state, but to immediately go to the right place in the list. For example:

```
Main list of states [
        Group [
                State [
                        State number,
                        [tasks in the system,
                         completed tasks]
                ]
        ]
].
```

The next step is filling of the coefficient matrix. Under initial conditions, this is a square zero matrix with dimension Ns. We must sequentially take states one by one from the list of states and sequentially apply 3 boundary conditions for each state, due to it all necessary changes of the matrix of coefficients that describe this state occur. Denote the matrix of coefficients as A, the number of the current state as Num, μ and λ – the intensities of arriving and maintenance of the task with the number Num.

1. If this is **not** the first state within group:

$$A_{Num,Num} = A_{Num,Num} - \mu;$$

$$A_{Num,Num-1} = A_{Num,Num-1} + \lambda;$$

2. If this is **not** the last state within group:

$$A_{Num,Num} = A_{Num,Num} - \lambda;$$

3. If this is **not** the first group:

$$A_{Num,\Pr ev\_Num} = A_{Num,\Pr ev\_Num} + \mu,$$

where Prev_Num is the state number from which you can transit to the state with the Num state number after completed the task, in the state list it is always in group j-1 and has a number in its group i + 1.

After processing all states, such algorithm at the output provides a ready-made lower triangular matrix of coefficients that eliminates the need to sort it.

The flowchart of the entire algorithm is shown in Figure 1.

3 Comparative analysis of recursive and sequential algorithms

Both algorithms for generating a matrix of coefficients of an ODE system without deriving the general system equation were implemented in Python3. All measurements were performed when the algorithms are running on the same device, it allows us to consider the obtained values as valid for comparison. The performing time of the algorithms may differ up or down depending on the power of the platform on which the algorithms are running, but the general trends will remain correct. The results of the comparative analysis are presented in the table 1.The flowchart of the entire algorithm is shown in Figure 1.

As we can see from Table 1, the number of possible states of the system, and consequently, the number of equations in the system of ODE grows exponentially from the number of tasks that can enter the nSQS. A graph of dependence between number of tasks and number of states is presented in Figure 2.

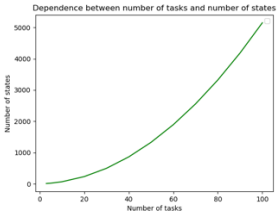

Figure 2: Dependence between number of tasks and number of states

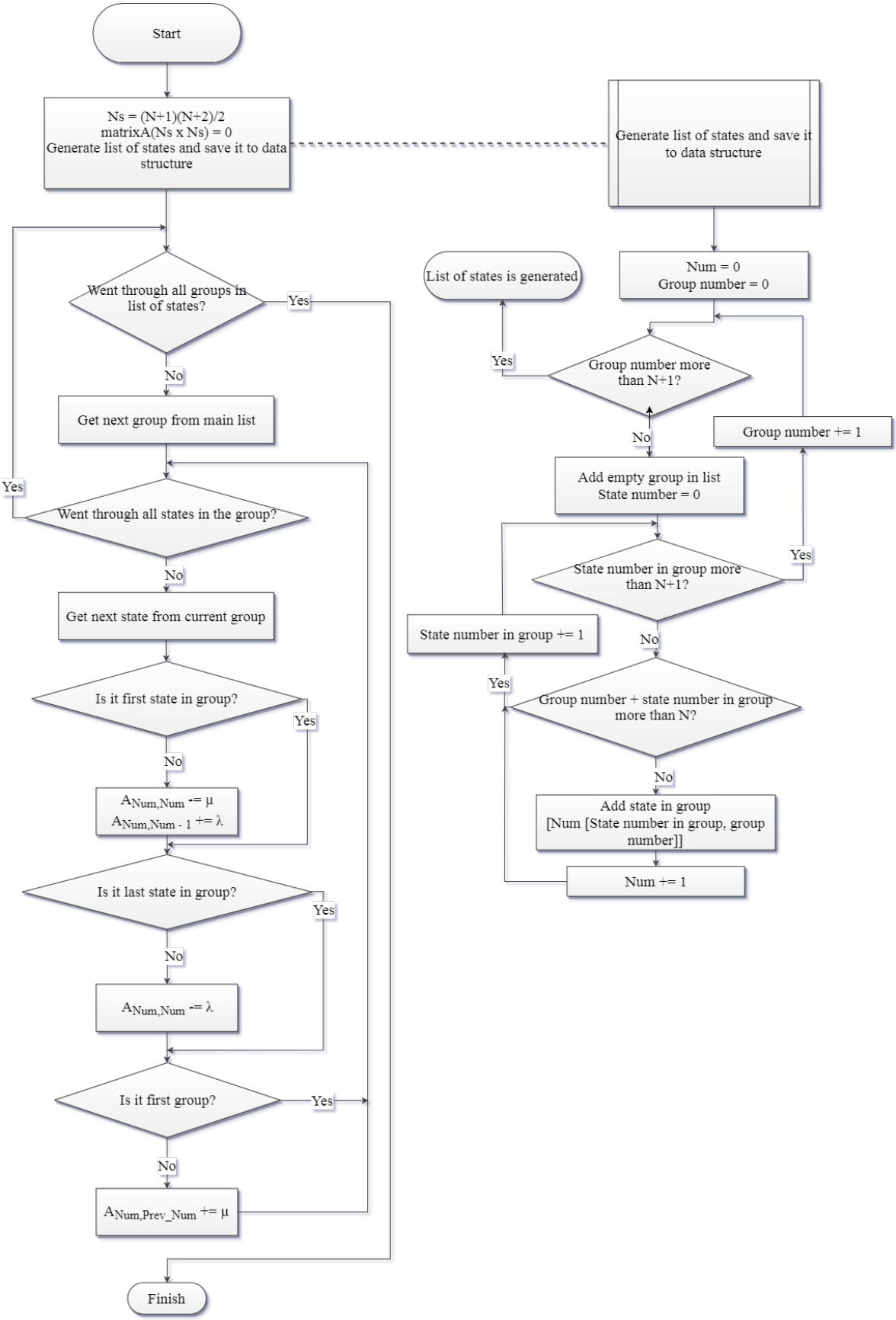

Figure 1: The flowchart of the sequential algorithm

Table 1: Comparison of performing time of algorithms

Tasks	States	Performing time of sequential algorithm, seconds	Performing time of recursive algorithm, seconds	Generation without sorting (for recursive), seconds
3	10	0.0027289390564	0.000235080718994	0.000102996826172
5	21	0. 000874042510986	0.000907897949219	0.000251054763794
10	66	0. 000315189361572	0.00605607032776	0.000638008117676
20	231	0. 00100684165955	0.129016876221	0.00578188896179
30	496	0. 00470900535583	0.592235088348	0.0215079784393
40	861	0. 00546598434448	2.48418688774	0.0802609920502
50	1326	0.0105609893799	12.4112920761	0.151191949844
60	1891	0.0128128528595	83.0276899338	0.318285942078
70	2556	0.0225200653076	224.342078924	0.576465129852
80	3321	0.0305080413818	568.904677153	0.847626924515
90	4186	0.0393660068512	1163.46502709	1.33484387398
100	5151	0.105370044708	2274.09017706	1.87042212486

Figure 3 shows a comparison of two parts of the recursive algorithm: generating a list of states and a matrix of coefficients and their sorting. As we can see from Figure 3, with an increase the number of states, most of the time the recursive algorithm is busy sorting the results.

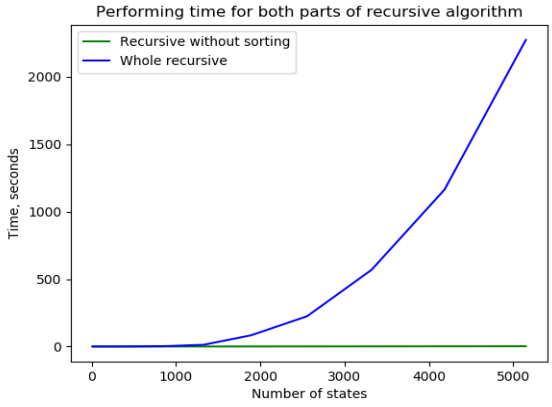

Figure 3: Performing time for both parts of recursive algorithm

Figure 4 presents a comparison of the total time of the recursive and sequential algorithms for generating the coefficient matrix. The execution time of the recursive algorithm grows exponentially with an increase the number of states, from Figure 3 it can be concluded that most of this time is spent to sorting.

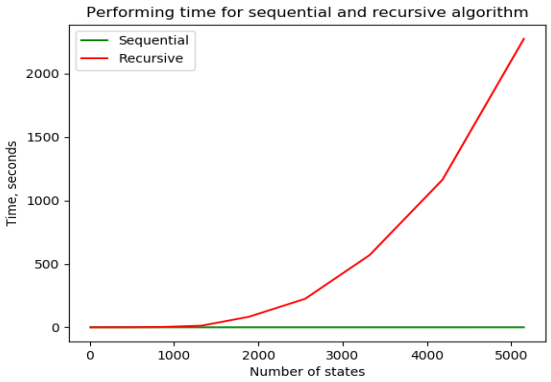

Figure 4: Performing time for sequential and recursive algorithms

Figure 5 shows the execution time of the sequential algorithm and the execution time of the recursive algorithm without the time-consuming sorting step. As you can see, even in this case, the recursive algorithm is inferior to the sequential algorithm in speed.

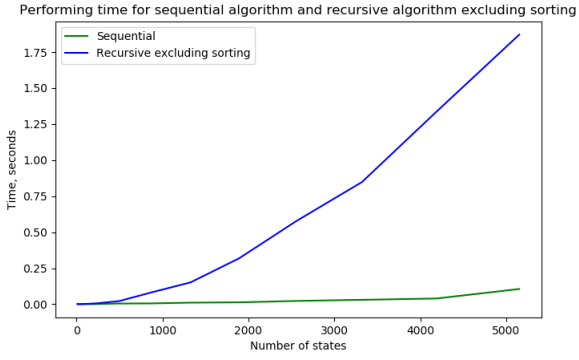

Figure 5: Performing time for sequential algorithm and recursive algorithm excluding sorting

Figure 6 shows a graph of dependence between number of states and performing time of sequential

algorithm, 100 values are used with the number of incoming applications from 1 to 100.

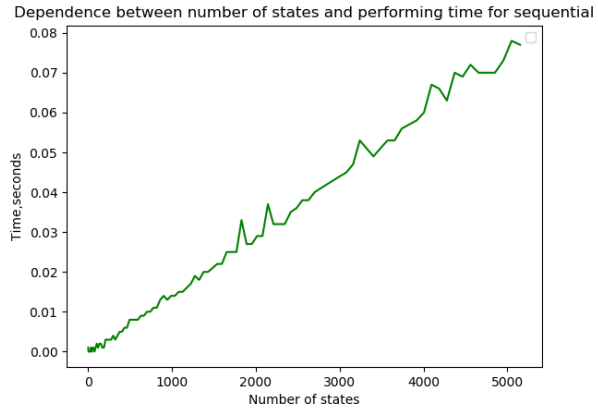

Figure 6: Dependence between number of states and performing time of sequential algorithm

As we can see from Figure 6, the performing time of this algorithm has an ordinary linear dependence on the number of tasks coming into the system and the number of states of this system.

4 Conclusion

The proposed sequential algorithm has undeniable advantages compared with the recursive algorithm: the performing time of it is significantly less and has a linear dependence on the number of tasks coming into the system, in contrast to the exponential dependence in the recursive algorithm. A sequential output algorithm provides a sorted list of states and a lower triangular matrix of coefficients, which eliminates the need for sorting used in the recursive algorithm.

As the system parameters increase, the number of states may increase, in some individual cases, the number of rules and conditions for transitions, but the overall complexity remains at the same level. It is recommended to use a sequential algorithm for implementations of the numerical-analytical method, instead of a recursive algorithm.

References

[Zeg12] P.D. Zegzhda, D.P. Zegzhda, A.V. Nikolskiy (2012). Using graph theory for cloud system security modeling. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Pp. 309–318.

[Oso13] T. Osogami, R. Raymond (2013). Analysis of transient queues with semi definite optimization. Queueing Systems, vol. 73. Pp. 195–234.

[Upa16] S. Upadhyaya (2016). Queueing systems with vacation: an overview. International journal of mathematics in operational research, vol. 9, issue 2. Pp. 167–213.

[Bub11] V.P. Bubnov, A.D. Khomonenko, A.V. Tyrva Software reliability model with coxian distribution of length of intervals between errors detection and fixing moments // International Computer Software and Applications Conference. 2011. Pp. 310-314.

[Bub10] V.P. Bubnov, A.V. Tyrva, A.D. Khomonenko Model of reliability of the software with coxian distribution of length of intervals between the moments of detection of errors // International Computer Software and Applications Conference. 34th Annual IEEE International Computer Software and Applications Conference, COMPSAC 2010. Seoul, 2010. Pp. 238-243.

[Bub99] V.P. Bubnov, V.I. Safonov Razrabotka dinamicheskih modelej nestacionarnyh system obsluzhivaniya. [Developing dynamic modeling of non-stationary systems.] / V.P. Bubnov, V.I. Safonov. – Saint-Petersburg, 1999, 65 p.

[Bub15] V.P. Bubnov, A.S. Eremin, S.A. Sergeev Osobennosti programmnoj realizacii chislenno analiticheskogo metoda raschyota modelej nes-tacionranyh sistem obsluzhivaniya: Trudy SPIIRAN. [Features of the software implementation of numerical-analytical method of calculation models non-stationary service systems: SPIIRAS Proceedings.] / V.P. Bubnov, A.S. Eremin, S.A. Sergeev. 2015. №1. Pp. 218-232.

[Bub15] V.P. Bubnov, A.D. Khomonenko, S.A. Sergeev Recursive method for generating the coefficient matrix of the system of homogeneous differential equations describing nonstationary system maintenance: Proceedings of International Conference on Soft Computing and Measurements, SCM 2015 18. 2015. Pp. 75-77.

[Ser15] S.A. Sergeev Method for compilation of the system of homogeneous differential equations for calculation probability-time characteristics which describing non stationary systems. // Intellectual Technologies on Transport.. 2015. №2. Pp. 32-42.

[Wol14] R.W. Wolff, Y.-C. Yao Little's law when the average waiting time is infinite. Queueing Systems, 2014. vol. 76. Pp. 267–281.

[Sud13] R. Sudhesh, K.V. Vijayashree Stationary and transient analysis of M/M/1 G-queues. Int. J. of Mathematics in Operational Research, 2013. vol. 5. no 2. Pp. 282–299.

[Sud13] R. Sudhesh, L. Francis Raj Stationary and transient solution of Markovian queues — an alternate approach. Int. J. of Mathematics in Operational Research, 2013. vol. 5. no. 3. Pp. 407–421.

[Bub14] V.P. Bubnov, A.V. Tyrva, A.S. Eremin [A set of non-stationary queuing system models with phase-type distributions]. Trudy SPIIRAN – SPIIRAS Proceedings, 2014. vol. 6(37). Pp. 61–71.

[Feh69] E. Fehlberg Low-order classical Runge—Kutta formulas with step size control and their application to some heat transfer problems. NASA Technical Report 315 (1969), extract published in Computing vol. 6, no. 1–2, 1970. Pp. 61–71.

[Bub11] V.P. Bubnov Algoritm analiticheskogo raschyota veroyatnostej sostoyanij nestacionarnyh system obsluzhivani-ya: Izvestiya Peterburgskogo universiteta putej soobshcheni-ya. [Algorithm of analytical calculation of non-stationary state probabilities service systems: News from the St. Petersburg University of communication.] / V.P. Bubnov. 2011. № 4. 90-97 p.