MARKOV MODELLING

W.K. Grassmann
Department of Computational Science
University of Saskatchewan
Saskatoon, Saskatchewan
Canada

Markov modelling deals with discrete event systems in which events happen completely at random.

The resulting Markovian systems can often be analyzed analytically. In this session, we show how to use the available packages, and what methods can be employed to find transient and steady state results. It turns out that Markov modelling is extremely convenient to analyze systems with very few state variables.

I. DEFINITION OF MARKOV MODELLING

To define Markov modelling, it is best to first consider discrete event simulation. In discrete event simulation, one has

1. states (e.g., lengths of different lines);
2. events (e.g., arrivals, change of line);
3. a schedule (time at which events occur including the time when simulation ends);
4. an initial state;
5. output statistics (e.g. number in system at time T, length of a queue averaged from 0 to T).

Markov modelling deals with discrete event systems as described above, except that events happen completely at random, at a rate which depends on the state. No schedule is necessary, which simplifies the system considerably. Moreover, in this session, we consider only analytical solutions to find the expectations of output statistics.

II. AN EXAMPLE

The following example is introduced to clarify the above terms. Suppose there are two parallel lines, and arrivals always join the shorter line. If both lines are of the same length, they join line 1. The customers stay in line until they are served by their respective server. The arrival rate is 1, and each server works at a service rate of 1.5. Arrivals are Poisson, and service times are exponential. At time zero, the system is empty. We want to find the expected number in the system at time 0.5 as well as expectation of the number in the system, averaged over the interval from 0 to 0.5. In this case, one has the following:

1. The state consists of 2 state variables, X1 and X2, which represent the length of line 1, respectively, line 2. Both X1 and X2 are non-negative integers. Hence

   X1, X2 $\geq$ 0

2. The events and their rates are given by the following table:

Table 1: Event Table for Parallel-Queues

Event Name	Effect	Rate	Condition
Arrival 1	X1 + 1, X2	1	X1 $\leq$ X2
Arrival 2	X1, X2 + 1	1	X1 > X2
Departure 1	X1 -1, X2	1.5	
Departure 2	X1, X2 - 1	1.5	

This table should be essentially self-explanatory. It indicates that the event called "Arrival 1", which is an arrival in line 1, increases X1 by 1, leaving X2 unchanged. The event occurs at a rate of 1, and it can only take place if X1 $\leq$ X2. The other events can be interpreted in a similar way. In addition to the conditions stated, each event must satisfy the following:

The state after the event takes place (the target state) must be in the prescribed state space. For instance, departure 1 requires $X1 \geq 0$ because X1 would be negative otherwise. This condition is implied, however, and is not stated in Table 1.

3. The <u>initial</u> state is:

   $X1 = 0, X2 = 0$

4. The output statistics are:

   (a) the number in the system at time $T = 0.5$ or $X1(0.5) + X2(0.5)$

   (b) The number in the system, averaged from 0 to 1, or

$$\frac{1}{0.5}\int_0^{0.5} X1(t) + X2(t)\, dt.$$

Our aim is to calculate the <u>expectations</u> of these output statistics by <u>analytical methods</u>.

III. GENERAL FORMULATIONS

In general one has:

1. A set of <u>states</u> S. For our purposes, we assume that each state $X \in S$ is a d-tupel

   $\{X1, X2, \ldots, Xd\}$.

2. u <u>events</u>, numbered from 1 to u, their <u>effects</u> $f_\ell(X)$, $\ell = 1, 2, \ldots, u$, their <u>rates</u> $\lambda_\ell(X)$, and a set C_ℓ of states at which the events are allowed to occur. This information is given in Table 2.

<u>Table 2</u>: Event Table for General Markovian System

Event #	Effect	Rate	Conditions
1	$f_1(X)$	$\lambda_1(X)$	C_1
2	$f_2(X)$	$\lambda_2(X)$	C_2
.	.	.	.
.	.	.	.
.	.	.	.
u	$f_u(X)$	$\lambda_u(X)$	C_u

We note that the rates may vary with the states. We should also mention that infinite rates are allowed, even though they will not be discussed here in any detail.

3. An initial condition X(0). This initial condition may be random.

4. A set of output statistics. In general, output statistics are based on descriptors, where a descriptor is a function of the state. The number in the system, in our example, for instance, is given by the descriptor function $h(X) = X1 + X2$. If $H(t)$ is the value of such a descriptor at time t, we consider the following output statistics:

   (i) $H(t)$, $0 \leq t \leq T$

   (ii) $\overline{H}(T) = \frac{1}{T}\int_0^T H(t)\, dt$.

   Maxima and minima of descriptors can also be handled, although they require additional state variables. As mentioned, only the expectations of output statistics will be calculated. We also note that probabilities are expectations of output statistics. Their descriptor are their indicator functions.

IV. PROGRAMS AND INPUT SPECIFICATIONS

The following software packages to do Markov modelling are available:

1. The RQA - analyzer of Wallace and Rosenberg [16];
2. The QUE - package [5];
3. The EVA - package;
4. Many ad-hoc programs to solve specific problems. We should mention here in particular the efforts of Gross and Miller [10], Melamed and Yadin [13], Hillier and Boling [12], Odoni and Roth [14], Goldberg [2], De Santis [1], and others.

In this session, we only discuss the QUE-package and the EVA-package. The QUE-package is interactive, and Table 3 gives a typical dialog. In fact, it shows how to describe the event "Arrival 1" of Table 1. The user answers are always underlined.

<u>Table 3</u>: Typical Dialog to Define an Event in the QUE-Package

```
EVENT #1: ARRIVAL 1
THIS EVENT INCREASES
   STATE VARIABLE1 BY ?  1
   STATE VARIABLE2 BY ?  0
ARE THERE ANY OTHER RESTRICTIONS TO THIS
EVENT?  YES
MINIMUM(1) MAXIMUM(2) OR LINEAR(3) RESTRICTIONS?  3
GIVE COEFFICIENTS OF LINEAR RESTRICTION
   STATE VAR 1 * 1
   STATE VAR 2 * -1
   LESS THAN OR EQUAL 0
IS THE RATE CONSTANT(1), VARIABLE(2)
   OR INFINITE(3)?  1
THE RATE EQUALS?  1
```

This table should essentially be self-explanatory. In general, the program is written such that modelling can be done with a minimum effort. On the other hand, it slightly restricts the shape of the event functions $f_\ell(X)$ and the rate functions $\lambda_\ell(X)$.

The second package, EVA, does not have these restrictions. In EVA, the user has to provide a FORTRAN subroutine, called EVENTS, which has to provide all the information about the events and their rates. For the example of Table 1, this subroutine is given in Table 4. The program should be essentially self-explanatory. For each event, the target state XTAR(1), XTAR(2) is calculated, given the state X(1), X(2). Once a target state is found, the subroutine INSERT is called to record the state in an appropriate way. At the same time, a check is performed to see whether or not the target state is in the state space. If it is outside, the event is not applied to this particular state.

Table 4: Example of EVENTS - Routine of EVA

```
      SUBROUTINE EVENTS (X)
      INTEGER X(2), XTAR(2)
      LAMDA = 1
      MU = 1.5
C
C     ARRIVAL 1
C
      IF (X(1) . GT. X(2)) GO TO 20
      XTAR(1) = X(1) + 1
      XTAR(2) = X(2)
      CALL INSERT (XTAR, LAMDA)
      GO TO 30
10
C
C     ARRIVAL 2
C
20    XTAR(1) = X(1)
      XTAR(2) = X(2) + 1
      CALL INSERT (XTAR, LAMDA)
C
C     DEPARTURE 1
30    XTAR(1) = X(1) -1
      XTAR(2) = X(2)
      CALL INSERT (XTAR, MU)
C
C     DEPARTURE 2
C
      XTAR(1) = X(1)
      XTAR(2) = X(2) - 1
      CALL INSERT (XTAR, MU)
C
C     RETURN
C
      RETURN
      END
```

We note that even though the EVA-package allows complete freedom to formulate the events and their rates, it does take longer to program and debug the subroutine "EVENTS" than it does to conduct a dialog in the QUE-package. There is thus a price to pay for the increased flexibility.

V. THE CALCULATION OF THE PROBABILITY TO BE IN STATE X AT TIME t.

The later sections are based on the calculation of P(X;t), which is the probability to be in state X at time t. In this section, we thus explore the methods that are available to calculate P(X;t). For convenience, we also define $\underline{P}(t)$ to be the probability vector for time t, that is:

$$\underline{P}(t) = [P(X;t) \mid X \in S].$$

The problem thus consists of finding $\underline{P}(t)$, given $\underline{P}(0)$, using analytical methods.

In the following, we give a number of equations for $\underline{P}(t)$.

1. According to the Chapman-Kolmorgorov equation, one has:

$$\frac{d}{dt} P(Y;t) = \sum_{X \in S} a_{xy} \; P(X;t) \tag{1}$$

Here, a_{xy} is the rate of going from X to Y, and:

$$a_{xx} = -\sum_{Y \in X} a_{xy}.$$

2. Written in matrix form, the Chapman-Kolmorgorov equation becomes:

$$\frac{d}{dt} \underline{P}(t) = \underline{P}(t) \; A, \tag{2}$$

with:

$$A = [a_{xy}].$$

3. A formal solution of (2) is given by

$$\underline{P}(t) = \underline{P}(0) \exp (At) = \sum_{n=0}^{\infty} \underline{P}(0)(At)^n/n!. \tag{3}$$

The right-hand side of this equation is the Taylor-expansion of the matrix exponential.

4. Using the event-notation given in Table 2, the Chapman-Kolmorgorov equation can be written as:

$$\frac{d}{dt} P(Y;t) = \sum_{e=1}^{u} \lambda_e [f_e^{-1}(Y)] \; P[f_e^{-1}(Y);t] - \lambda(Y) \; P(Y;t). \tag{4}$$

with:

$$\lambda(X) = \sum_{e=1}^{u} \lambda_e(X) = -a_{xx}.$$

λ(X) will also be referred to as the leaving rate of state X.

We now consider the numerical solution of the Chapman-Kolmorgorov equation. For such a solution, numerical stability is of prime importance. Fortunately, it can be shown (Grassmann [3]) that any algorithm which only deals with positive numbers, and which does not contain subtractions is always numerically stable. For such algorithms, one can even calculate tight bounds for the rounding error.

The first algorithm to be considered is the method of Liou [8], which essentially uses the Taylor-expansion given in (3). This algorithm is numerically unstable.

The second algorithm is Runge-Kutta, applied to either equation (1) or (4). It turns out that Runge-Kutta is identical to Liou's algorithm, applied in a step-wise fashion (see [7]). In other words, one first calculates $\underline{P}(h)$, given $\underline{P}(0)$, then $\underline{P}(2h)$, given $\underline{P}(h)$, then $\underline{P}(3h)$, given $\underline{P}(2h)$, and so on, until $\underline{P}(t)$ is found. While doing this, one normally only takes the first 4 terms of Taylor expansion given by (3), which reduces the numerical instabilities to an acceptable level. Runge-Kutta has been used, among others, by Odoni and Roth [12].

The third algorithm is the randomization method. This method works as follows: First, observe that any event with no effect (that is, with $f_e(X) = X$)) will have no influence on any output statistic. We can thus introduce as many of these null-events as we like. In particular, we can introduce null-events for each state X in such a way that $\lambda(X)$ has the same value λ for each $X \epsilon S$. Thus, let $\lambda_0(X)$ be the rate of the null-events. Whenever an event or a null-event takes place, we say that a jump has occurred. Obviously, the probability that a jump is caused by event e, $e = 0, 1, \ldots, u$ is given by:

$$P_e\ (X) = \lambda_e\ (X)/\lambda.$$

The probability that after n jumps, the system is in state Y can be calculated recursively as:

$$P^n(Y) = \sum_{e=0}^{u} P_e\ [f^{-1}(Y)]\ P^{n-1}(f^{-1}(Y)). \quad (6)$$

Here, $P^n(Y)$ is the desired probability. The initial probabilities are obviously given by:

$$P^0(Y) = P(Y;0).$$

Once the $P^n(Y)$ are calculated, $P(Y;t)$ is simply (see e.g. [7], [8], [10] or [11]):

$$P(Y;t) = \sum_{n=0}^{\infty} P^n(Y)\ p(n;\lambda t). \quad (7)$$

Here, $p(n;\lambda t)$ is the probability of having n jumps during time t. It can be shown that $p(n;\lambda t)$ is Poisson, that is:

$$p(n;\lambda t) = e^{-\lambda t}\ (\lambda t)^n/n!$$

Equations (5) and (6) describe the randominzation method. Besides De Santis [1], Grassmann [5, 8], Gross and Miller [10], it has also been applied by Melamed and Yadin [13] and others. The next section will further demonstrate and elaborate this method. Here, we note the following:

(a) Randomization works exclusively on positive numbers, and it contains no subtraction which makes it numerically extremely stable (see [3]);

(b) Randomization is equivalent to Liou's method, provided this method is applied to find $P(Y;t)e^{\lambda t}$ instead of $P(Y;t)$, as one can easily verify algebraically. The stepwise application of randomization is thus equivalent to Runge-Kutta, applied to $P(Y;t)\ \exp\ (\lambda t)$. It has been shown [4] that this adjustment improves the performance of Runge - Kutta, at least in the worst case. In short, randomization seems to be the best method available.

VI. THE APPLICATION OF RANDOMIZATION

In this section, we first show how to calculate the $P^n(X)$ for the example given in Table 1. We then consider several manipulations of equation (7) which allow us to find expectations of output statistics directly.

In the example, the leaving rate $\lambda(X)$ obviously has its maximum if both servers are busy. Using the rates given in Table 1, $\lambda(X)$ is then $1 + 1.5 + 1.5 = 4$. If one or both of the servers are idle, we have to inroduce null-events to compensate. For simplicity, we consider these null-events as departures, except that their effect is nil. We thus have:

P_a = Probability of an arrival = 1/4 = 0.25

P_{d1} = Probability of a departure 1 = 1.5/4 = 0.375

P_{d2} = Probability of a departure 2 = 1.5/4 = 0.375.

Since the events "Arrival 1" and "Arrival 2" are mutually exclusive we can treat them like 1 event, except that we manipulate the effect correspondingly.

We assume that the initial state is X1 = X2 = 0 or which means:

$$P^0(0,0) = P(0,0;0) = 1$$

All other initial probabilities are zero. Now, let us consider the possibilities for the first jump. Essentially, we have an arrival with probability 0.25, and a null-event with probability 0.75. Hence, after the jump n=1, we are in state 0,0 with probability 0.75 and in state 1,0 with probability 0.25. Using a probability tree, we can then calculate the probabilities for the difference possible states after n jumps, given we know the probabilities after n-1 jumps. For the system in question, this can easily be done manually, and indeed, I recommend to the reader to do these calculations himself. The results are given in Table 5.

Table 5: The Calculation of

	jump (n)					E(H -	$E(\overline{H}$
State	0	1	2	3	4	(0.25))	(0.25))
0,0	1	0.75	0.656	0.598	0.567		
1,0		0.25	0.282	0.293	0.286		
1,1			0.062	0.070	0.085		
0,1				0.023	0.035		
2,1				0.016	0.018		
2,0					0.006		
2,2					0.004		
$E(H_n)$	0	0.25	0.406	0.504	0.573		
p(n;1)	0.368	0.368	0.184	0.061	0.019	0.208	
q(n;1)	0.632	0.264	0.080	0.019	0.004		0.136

According to equation (7), we can now obtain the probability vector for time t by multiplying the entries of Table 5 (if necessary expanded for n=5, 6, ...,) by $p(n;\lambda t)$. However, if only the expectations of certain output statistics are required, one can use the following equations which can easily be derived from equation (7)

$$E\,[H(t)] = \sum_{n=0}^{\infty} E(H_n)\; p(n;\lambda t) \qquad (8)$$

$$E\,[\bar{H}(T)] = \sum_{n=0}^{\infty} E(H_n)\; q(n;\lambda T) \qquad (9)$$

Here, $H(t)$ and H_n are the values of a descriptor H at time t, respectively, after n jumps. $\bar{H}(T)$ is the value of $H(t)$, averaged over the interval from 0 to T. $q(n;\lambda T)$ is finally given by

$$q(n;\lambda t) = \frac{1}{T}\int_0^T p(n;\lambda t)\,dt = \frac{1}{\lambda T}\sum_{m=n+1}^{\infty} p(m;\lambda T).$$

If H is the number in the system, $H(t)$ is thus the number in the system at time t, and $\bar{H}(T)$ is the number in the system, averaged from 0 to T.

In Table 5, the expectations of these two output statistics are calculated for t=0.25, respectively T=0.25, using equation (8) and (9). The computational effort to do this is minimal.

VII. EQUILIBRIUM SOLUTIONS

In this section, we investigate the long run behaviour of Markovian systems, a topic of interest both in simulation and in queueing. For this purpose, let P(X) be the value of P(X;t) as t converges toward infinity. Methods to calculate P(X) can be found in Steward [15], Goldberg [2] and Grassmann and Taksar [9]. Here, we discuss three of these methods, namely the method of Wallace [17], the method of Gauss-Seidel [12], and Gaussian elimination [2, 9].

1. The method of Wallace

Wallace observes that $P^n(X)$ as given by (6) converges toward P(X), and that for large enough n, $P^n(X)$ can thus be used as an approximation for P(X). The main question, which is by no means trivial, is how one can assure that n is indeed large enough to give P(X) with the required precision. Moreover, if care is not taken that at least one recurrent state has null-events, the chain X_n may be periodic, and equilibrium is never reached. Nevertheless, Wallace's method is a useful and numerically stable method.

2. The Method of Gauss-Seidel

To apply the method of Gauss-Seidel, the states must be numbered. Thus, let X_1 be state number 1, X_2 state number 2 etc., and assume that there are $N = \|S\|$ states. Also, let b_{ij} be the rate of going from state X_i to state X_j, and let b_i be the rate of leaving X_i. Finally, let p_i be the equilibrium probability for state X_i. The equilibrium equations can then be written as:

$$p_j b_j = \sum_{i=1}^{j-1} p_i b_{ij} + \sum_{i=j+1}^{N} p_i b_{ij}. \qquad (10)$$

This suggests the following recursive scheme:

$$p_j^n = \sum_{i=1}^{j-1} \frac{b_{ij}}{b_j}\, p_i^n + \sum_{i=j+1}^{N} \frac{b_{ij}}{b_j}\, p_i^{n-1}\,.$$

Here, p_j^n is the nth approximation to the final probability p_j, given certain initial values p_j^0. After each iteration, the p_j^n must be divided by an appropriate constant such that their sum becomes 1.

The method of Gauss-Seidel is very similar to the method of Wallace. The stopping criterium may again pose a problem. Moreover, in the method of Gauss-Seidel, the convergence is effected by the ordering of the states. According to my experience, the method of Gauss-Seidel converges faster than the method of Wallace, although more research to corroborate this finding would be desirable. I also could not find any example in which the method cycles, as it does sometimes in Wallace's method, unless precautions are taken.

3. The Method of Gauss-Jordan

The problem of the elimination of the p_j from the steady state equation by the method of Gauss-Jordan is its numerical stability. However, in [9], we described a pivoting strategy which avoids subtractions and operations with negative numbers, ensuring thus numerical stability. As a starting point, one uses equation (10). One now pivots, starting with $b_{N,N}$, and continuing with $b_{N-1,N-1}$, ... If this is done, all off-diagonal b_{ij} will always stay positive. Moreover, the pivot element in step n can be found as:

$$b_{n,n} = \sum_{j=1}^{n-1} b_{n,j}.$$

For the details of this method, we refer to the paper mentioned above. Here, we only note that the method is ideally suited for banded matrices, that is, matrices that have all $b_{ij} = 0$ unless $i - g \le j \le i + h$. As it turns out, this is normally the case in Markovian systems.

VIII. COMPUTATIONAL CONSIDERATIONS

Mathematicians usually think that a problem is solved when all equations are written down. In Markov Modelling, this is not true. Indeed, the housekeeping chores and related matter are both more difficult to program and often more time-consuming to execute than the straightforward mathematics. This begins with the generation of matrices, which we now discuss.

The simplist method to find the transition matrix is the following. First, one restricts the state space to some manageable size. In the example of Table 1, one could, for instance, restrict both X1 and X2 to be at or below some limit, such as 2. Next, one can enumerate all states, X, and apply all possible events to each state X. For our example, this is done in Table 6 (see also [6]). For instance, in state 0, 0, the only event that can occur is an arrival at line 1, which brings us to state 1, 0, and which happens at a rate of 1. For the next state in lexigraphical order, namely 01, we either have an arrival at line 1, which brings us to state 1, 1, or a departure from line 2, which brings us to state 0, 0. The rates of these events are, respectively, 1 and 1.5. In this way, one goes on combining all events with all the possible states.

Table 6: Fixed Matrix Generation

State	Arrival 1		Arrival 2		Departure 1		Departure 2	
	New State	Rate	New State	Rate	New State	Rate	New State	Rate
0,0	1,0	1	-		-		-	
0,1	1,1	1	-		-		0,0	1.5
0,2	1,2	1	-		-		0,1	1.5
1,0	-		1,1	1	0,0	1.5	-	
1,1	2,1	1	-		0,1	1.5	1,0	1.5
1,2	2,2	1	-		0,2	1.5	1,1	1.5
2,0	-		2,1	1	1,0	1.5	-	
2,1	-		2,2	1	1,1	1.5	2,0	1.5
2,2	-		-		1,2	1.5	2,1	1.5

The entries of this Table can now be stored in arrays. In the QUE-package, for instance, the original state is stored in an array ORIG, the corresponding new or target state in an array TARG, and the rate in an array RATE. Thus, in the first 2 rows of Table 6 would be stored as:

```
ORIG(1) = 0.0  TARG(1) = 1,0 , RATE(1) = 1
ORIG(2) = 1,0  TARG(2) = 1,1 , RATE(2) = 1
ORIG(3) = 1,0  TARG(3) = 0,0 , RATE(3) = 1.5
```

The states are then numbered consecutively for easier reference. In algorithmic terms, one thus has:

```
1. NST = 0
     For all $X \in S$, do following
        For all events $e=1,2,\dots, u$, do following
           If event $e$ is possible
              NST = NST + 1
              ORIG(NST) = X
              TARG(NST) = $f_e(X)$
              RATE(NST) = $\lambda_e(X)$.

2. Replace all state descriptors by a unique
   reference number between 1 and N.
```

After the above algorithm has been completed, the elements are stored in a form suitable for doing randomization and the method of Wallace. They are not stored suitably to apply the method of Gauss-Seidel or Gauss-Jordan. The reason is that the entries are generated by rows, whereas both, Gauss-Seidel and Gauss-Jordan require the entries by columns, as evidenced by equation (10). For the latter two methods, the states have to be re-ordered. This re-ordering may seem a trivial task, but it makes such things as changing the transition matrix very difficult. Yet changing the transition matrix is very important, especially for doing sensitivity analysis and analyzing systems with changing rates. The methods of Gauss-Seidel and Gauss-Jordan have thus a definite disadvantage here.

When constructing Table 5, we suggested to use decision trees to find the new states and their probabilities. In this method, we do not generate a state before its probability is different from zero. This adaptive method has several advantages over the fixed method discussed earlier. In particular, one need not limit the state space in advance. Moreover, one can easily program this method in such a way that new states are only introduced if their probability exceeds a certain lower limit, removing thus unlikely states from consideration. Finally, since states with a probability of zero are ignored, there is a possible reduction in computer time.

On the other hand, this method has two disadvantages. First, one must find an efficient method to identify a state as new or old, because the method works too slowly otherwise. In the EVA-package, this state identification is done through hashing, but multiple linked lists are also possible. The second disadvantage is that the method generates the new states row-wise. This means that the adoptive method is not applicable for Gauss-Seidel or Gausss-Jordan. It is applicable for Wallace's method though.

A major consideration for all methods is the computer time. In the iterative methods, this time depends essentially on two variables, namely the number of iterations needed, and the number of non-zero entries in the transition matrix (we consider this matrix as fixed for the moment). Specifically, the number of operations is given for all three iterative methods as:

$$2rm + d$$

Here, r is the number of iterations, and m is the number of non-zero entries in the transition matrix, and d is a number small when compared to m. To illustrate how this formula would work out consider a Markovian system with u events and d state variables. Each of these d state variables can assume exactly 3 different values, and all events are possible for each possible states. The number of non-zero entries becomes then $z^d u$ and one has:

$$2rz^d u$$

operations. If there are, for instance, d=3 state variables, each being able to assume 10 different values, and if one has 4 events, which would be typical for such a system, one has 8000 operations per iteration, which is not much for a modern computer. However, each additional state variable increases the computational effort by a factor, and this exponential growth will soon become intolerable. In short, for small systems, the methods discussed here are extremely convenient and fast, especially when using packages. For systems with many state variables, however, they are not recommended.

REFERENCES

1. M. De Santis, Markovian Models for Time-Sharing Systems, M.Sc. Thesis, University of Saskatchewan, Saskatoon, April 1977.

2. H.M. Goldberg, Computation of State Probabilities for M/M/s Priority Queues with Customer Classes Having Different Service Rates, INFOR, 19 (1) Feb. 1981, pp. 48-58.

3. W. Grassmann, Rounding Errors in Some Recursive Methods Used in Computational Probability, Dept. of O.R., Stanford University Tech Report 73, April 1983.

4. W. Grassmann, The GI/PH/1 Queue, INFOR, 20 (2), pp. 144-156, May 1982.

5. W. Grassmann, The QUE-Package, Session 82 of the Canadian Information Processing Society, pp. 104-108, Saskatoon, May 1982.

6. W. Grassmann, Stochastic Systems for Management, American Elsevier, 1981.

7. W. Grassmann, Transient Solutions in Markovian Queues, EJOR 1, pp. 396-402, 1977.

8. W. Grassmann, Transient Solutions in Markovian Queueing Systems, Comput & Ops. Res., 4, pp. 47-53, 1977.

9. W. Grassmann, M. Taksar, Applications of Semi-Regenerative Theory to Computations of Stationary Distributions of Markov Chains, Dept. O.R., Stanford University, Tech. Report 77, May 1983.

10. D. Gross, R.D.R. Miller, The Randomization Technique as a Modelling Tool and Solution Procedure for Transient Markov Processes, Operations Research, to appear.

11. A.T. Kohlas, Stochastic Methods of Operations Research, Cambridge University Press, 1982.

12. F.S. Hillier, R.W. Boling, Finite Queues in Series with Exponential Erlang Service Times, Operations Research 15, pp. 268-303, 1967.

13. B. Melamed, M. Yadin, Numerical Computation of Sojourn Time Distributions in Queueing Networks, preprint, 1981.

14. A.R. Odoni, E. Roth, An Empirical Investigation of Transient Behavior of Stationary Queueing Systems, Operations Research 31, (3), pp. 432-455, 1983.

15. W.T. Steward, Comparison of Numerical Techniques in Markov Modelling, Com. of the ACM, 21, pp. 144-152, 1978.

16. V.L. Wallace, R.S. Rosenberg, RQA-1, The Recursive Queue Analyzer, Systems Eng. Lab, University of Michigan, Ann Arbor, Tech. Report 2, 1966.

17. V.L. Wallace, Markovian Models and Numerical Analysis of Computer System Behavior, Proc. AFIPS, Spring Joint Computer Conference, 28, pp. 141-148, AFIPS-Press, 1966.