# A Survey of Methods for Computing Large Sparse Matrix Exponentials Arising in Markov Chains*

Roger B. SIDJE† William J. STEWART‡

March 28, 1996

**Abstract**

Krylov subspace techniques have been shown to yield robust methods for the numerical computation of large sparse matrix exponentials and especially the transient solutions of Markov Chains. The attractiveness of these methods results from the fact that they allow us to compute the action of a matrix exponential operator on an operand vector without having to compute, explicitly, the matrix exponential in isolation. In this paper we compare a Krylov-based method with some of the current approaches used for computing transient solutions of Markov chains. After a brief synthesis of the features of the methods used, numerical comparisons are performed on a POWER CHALLENGEarray supercomputer on three different models.

**Keywords.** Matrix exponential, Markov chains, Krylov methods, Ordinary differential equations.

**AMS Subject Classification.** 65F99, 65L05, 65U05.

## 1 Introduction

There exists several numerical techniques for obtaining transient solutions of homogeneous, irreducible Markov chains. These techniques are based either on computing matrix exponentials or integrating the Chapman-Kolmogorov system of differential equations:

$$\begin{cases} \dfrac{dw(t)}{dt} &= Aw(t), \quad t \in [0,\ T] \\ w(0) &= v, \qquad \text{initial probability distribution.} \end{cases} \tag{1}$$

The coefficient matrix $A$ is an *infinitesimal generator* of order $n$, where $n$ is the number of states in the Markov chain. Thus $A \in \mathbf{R}^{n\times n}$, with elements $a_{ij} \geq 0$ when $i \neq j$, and $a_{ij} = -\sum_{i\neq j}^{n} a_{ij}$. From an algebraic standpoint, a homogeneous Markov chain leads to a time-independant matrix $A$, and irreducibility means that the original problem can not be 'decoupled' in the sense that there does not exist a permutation matrix $P$ for which

$$PAP^T = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}.$$

*This work has benefit from the support of NSF/INRIA agreement INT-9205155.

†Formerly at INRIA/IRISA, Rennes, France. Now at Department of Mathematics, University of Queensland, Australia (rbs@maths.uq.oz.au).

‡Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206, USA. (billy@csc.ncsu.edu). Research supported in part by NSF grants CCR-9413309 and INT-9205155.

Of interest to us in this paper is the transient solution, $w(t)$. It is given by the solution of (1) and is known to be:

$$w(t) = e^{tA}v. \tag{2}$$

Since the matrix exponential is full even when the original matrix is sparse, the practical computation of $e^{tA}$ in full remains possible only when $A$ is relatively small, i.e., when the number of states in the Markov chain does not exceed a few hundreds. In [15], Moler and Van Loan provide an instructive review of possible methods applicable in this context. Although this review shows that none of the methods is unconditionally acceptable for all classes of problems, methods such as those of the Padé-type or matrix decompositions, with careful implementation, can be satisfactory in many contexts. These methods involve matrix-matrix operations.

To address large problems, the family of series methods, in which matrix-vector operations are paramount, appears to be a reasonable choice. In this paper we will consider one of this class, the so-called *uniformization* method. The use of this method is particularly widespread for reasons that we shall see later. Also, the class of ordinary differential equations solvers is appealing because of the high availability of ready-to-use efficient library routines for solving initial value problems in ODEs. In addition to being multiple- or single-step, ODE solvers can be explicit or implicit, yielding four possible categories. Each category yields new classes of methods in their own right – depending on their derivation, their analytic and numerical properties, or on implementation aspects. For example, some of the particularities of a method can be: multistage or otherwise, order, stability region, matrix-free or otherwise, stiff or non-stiff, computational cost, etc. In general, implicit methods — which are more costly, appear suitable for stiff-problems while cheap explicit methods are satisfactory only on non-stiff problems. It is neither realistic, nor the ambition of this paper to include all the methods with their variants. Rather, we shall consider representatives of each of the well-known major classes namely, Runge-Kutta, Adams, and Backward-Differentiation Formulas (BDF). The choice of a particular implementation (for instance the VODPK package) was motivated by criteria such as availability, portability, and popularity – all of these understandably backed by robustness and efficiency.

Along with these general methods, we include a Krylov projection method which we shall use as a reference. As we shall see, this method appears to be the most versatile and although not always the fastest, it was the only method to complete all the test problems assigned within the allotted computation interval.

The organization of the paper is as follows. In Sections 2, 3, and 4, we review the methods used in this study. In Section 5 we describe the supercomputer used in our experiments. Finally, in Section 6 we describe the test models and compare the methods on these models.

## 2 The Krylov-based method

The Krylov-based algorithm that we consider is presented below. It generates an approximation to $w(t) = \exp(tA)v$ and computes the matrix exponential times a vector rather than the matrix exponential in isolation.

ALGORITHM 1: $w(t) = \exp(tA)v$

**1.** $w := v$; $t_k := 0$;
$\overline{H}_{m+2} := \text{zeros}[m+2, m+2]$;
**2.** **while** $t_k < t$ **do**
  $v := w$; $\beta := \|v\|_2$;
  $v_1 := v/\beta$;
  **for** $j := 1 : m$ **do**
    $p := Av_j$;
    **for** $i := 1 : j$ **do**
      $h_{ij} := v_i^T p$;
      $p := p - h_{ij} v_i$;
    **end**
    $h_{j+1,j} := \|p\|_2$;
    $v_{j+1} := p/h_{j+1,j}$;
  **end**
  $\overline{H}(m+2, m+1) := 1$;
  **repeat**
    $\tau :=$ step-size;
    $F_{m+2} := \exp(\tau \overline{H}_{m+2})$;
    $w := \beta V_{m+1} F(1 : m+1, 1)$;
    $err\_loc :=$ local error estimate;
  **until** $err\_loc \leq 1.2 tol$;
  $t_k := t_k + \tau$;
**end**

The underlying principle is to approximate

$$w(t) = e^{tA}v = v + \frac{(tA)}{1!}v + \frac{(tA)^2}{2!}v + \cdots \tag{3}$$

by an element of the Krylov subspace

$$\mathcal{K}_m(tA, v) = \text{Span}\{v, (tA)v, \ldots, (tA)^{m-1}v\}, \tag{4}$$

where $m$, the dimension of the Krylov subspace, is small compared to $n$, the order of the coefficient matrix (usually $m \leq 50$ whilst $n$ can exceed many thousands). The approximation used is

$$\widetilde{w}(t) = \beta V_{m+1} \exp(t\overline{H}_{m+1}) e_1 \tag{5}$$

where $\beta = \|v\|_2$; $V_{m+1} = [v_1, \ldots, v_{m+1}]$ and $\overline{H}_{m+1} = [h_{ij}]$ are, respectively, the orthonormal basis and the upper Hessenberg matrix resulting from the well-known Arnoldi process

(see, e.g., [8, 20]); $e_1$ is the first unit basis vector. The distinctive feature is that the original large problem (3) is converted to the small problem (5) which is more desirable. In reality however, due to stability and accuracy considerations, $w(t)$, is not computed in one go. On the contrary, a time-stepping strategy along with error estimation is embedded within the process. In other words, there is an integration scheme similar to that of a standard ODE solver. Typically, the algorithm evolves with the integration scheme

$$\left\{ \begin{array}{rcl} w(0) & = & v \\ w(t_{k+1}) & = & e^{(t_k+\tau_k)A}v = e^{\tau_k A}w(t_k), \quad k = 0, 1, \ldots, s \end{array} \right. \tag{6}$$

where

$$\tau_k = t_{k+1} - t_k, \qquad 0 = t_0 < t_1 < \cdots < t_s < t_{s+1} = t.$$

It is clear from (6) that the crux of the problem remains an operation of the form $e^{\tau A}v$, albeit with different $v$'s. The selection of a specific step-size $\tau$ is made so that $e^{\tau A}v$ is now effectively approximated by $\beta V_{m+1} \exp(\tau \overline{H}_{m+1})e_1$. Following the procedures of ODEs solvers, an a *posteriori* error control is carried out to ensure that the intermediate approximation is acceptable with respect to expectations on the global error. More information may be obtained from [23].

This technique seems to have emanated from Chemical Physics and it is now gaining wide acceptance thanks to the theoretical studies of Gallopoulos and Saad [7], Saad [19], as well as Hochbruck and Lubich [13]. Within the framework of Markov chains, relevant studies are those of Philippe and Sidje [17], and Sidje [22, 23]. The Markovian context brings other considerations. We need to compute $w(t) = \exp(tA)v$ subject to the constraint that the resulting vector is a probability vector and thus with components in the range [0, 1] and with sum equal to 1. Since the analytic solution of the Chapman-Kolmogorov system of differential equations is $w(t)$, its computation can be addressed totally in the perspective of ODEs. But ODEs solvers do not bring any guarantees either. Practice shows that these methods may produce results with negative components!

Two results of interest were obtained in [17, 22] by exploiting the fact that the matrix $A$ satisfies some inherent properties when it originates from a Markov chain model. Firstly the computed Krylov approximation is mathematically guaranteed to be a probability vector for small enough step-sizes and secondly, the global error in the approximation grows at most linearly. Additionally, it is possible to detect and cope with excessive roundoff errors. The ensuing code referred to as **EXPOMK**$(m)$ implementing this customization is a component of the EXPOKIT package, a full description of which may be found in [23].

# 3 Uniformization

The Uniformization (or randomization) technique is based on the evaluation of the $p$th partial Taylor series expansion of the matrix exponential [9, 10]. The length $p$ is fixed so that the prescribed tolerance on the approximation is fulfilled. Since $A$ is essentially nonnegative (i.e., the diagonal elements of $A$ are negative and the off-diagonal elements are nonnegative), a naive use of the expression $w(t) = e^{tA}v \approx \sum_{k=0}^{p}(1/k!)(tA)^k v$ is subject to severe roundoff errors due to terms of alternating signs. Uniformization uses the modified formulation $w(t) = e^{\alpha t(P-I)}v = e^{-\alpha t}e^{\alpha tP}v$ where $\alpha \equiv \max_i |a_{ii}|$ and $P \equiv \frac{1}{\alpha}A + I$ is

nonnegative with $\|P\|_1 = 1$. The resulting truncated approximation

$$\widetilde{w}(t) = \sum_{k=0}^{p} e^{-\alpha t} \frac{(\alpha t)^k}{k!} P^k v \tag{7}$$

involves only nonnegative terms and becomes numerically stable. If $\epsilon$ denotes the prescribed error tolerance, the condition $\|w(t) - \widetilde{w}(t)\|_1 \leq \epsilon$ leads to a choice of $p$ such that

$$1 - \sum_{k=0}^{p} e^{-\alpha t} \frac{(\alpha t)^k}{k!} \leq \epsilon. \tag{8}$$

The rank $p$ may be determined simply by adding-up the above series until satisfaction. The popularity of Unifomization is related to three main facts. Firstly, its handiness and malleability facilitate its implementation – only a matrix-vector product is needed per iteration. Secondly, the transformation from $A$ to $P$ has a concrete interpretation. The matrix $P$ is stochastic and is the transition matrix of a Discrete Time Markov Chain (DTMC) which emulates the behavior of the Continuous Time Markov Chain (CTMC) whose infinitesimal generator is $A$. Thirdly and perhaps most important, it works surprizingly well in a great variety of circumstances. The shortcoming of Uniformization is that $p$ is likely to be large (and thus (7) involves many matrix-vector mutiplications) for large values of $t$.

A variant of Unformization, referred to as Power Uniformization was proposed by Abdallah and Marie [1]. It combines the standard Uniformization with scaling and squaring as follows. We may write

$$e^{tA} = \left(e^{\frac{tA}{2^s}}\right)^{2^s} \tag{9}$$

where $s$ is chosen such that $\|\frac{tA}{2^s}\| < 1$. The technique then uses a truncated Taylor series (modified as in the Uniformization method) to approximate $e^{\frac{tA}{2^s}}$ and repeated squaring to recover $e^{tA}$. This method is numerically stable [1, 6]. However, in contrast to the standard Uniformization which uses only matrix-vector operations, the scaling-squaring phase of Power Uniformization uses matrix-matrix operations and thus is suitable for small-sized problems only.

# 4 ODE Solvers

The remainder of our overview, presented in § 4.1 – § 4.3, deals with ODE solution-techniques. This summary is provided primarily for the sake of completeness and readers may wish to consult the textbooks [4, 11, 12] for more details. The specific aspects of a particular implementation can be found in the associated references.

## 4.1 Runge-Kutta (RK)

Runge-Kutta methods form a class of multistage, one-step integration methods. They can be either explicit or implicit but only an explicit variant has been included in this paper. Consider the problem of solving the autonomous linear system of first order differential equations

$$\begin{cases} y'(t) &= Ay(t) \equiv f(t, y(t)), & t \in [0,\ T] \\ y(0) &= y_0, & \text{initial condition.} \end{cases} \tag{10}$$

A given $s$-stage RK-method is defined by its Butcher-tableau,

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array}, \quad A = [a_{ij}] \in \mathbf{R}^{s\times s}, \quad b = (b_1, \ldots, b_s)^T, \quad c = (c_1, \ldots, c_s)^T = A\mathbb{1}, \quad \mathbb{1} = (1, \ldots, 1),$$

in which, for an explicit RK-method, $A$ is strictly lower triangular and $c_1 = 0$. At each step of the integration scheme, the next iterate is computed as

$$y_{k+1} \quad = \quad y_k + h_k \sum_{j=1}^{s} b_j f(x_k + h_k c_j, Y_j), \tag{11}$$

where the intermediate stage-evaluations are given by

$$Y_i = y_k + h_k \sum_{j=1}^{i-1} a_{ij} f(x_k + h_k c_j, Y_j), \qquad i = 1, \ldots, s. \tag{12}$$

In practice, to enable step-size and error control, the s-stage scheme is embedded within another (s+1)-stage scheme, in which case the conjoint schemes are referred to as a RK($s$,$s+1$) pair. The method is of order $s$ (i.e., the local truncation error is of order $\mathcal{O}(h^{s+1})$) and $s+1$ function evaluations (matrix-vector multiplies in our case) are used per step. The code used here is the RKSUITE available on netlib and is due to Brankin et al. [2]. We selected the **RK78** pair which is actually one of the most reputed explicit RK-method.

### 4.2 Adams

Adams methods belong to the class of linear multi-step methods. In its general formulation, a $r$-step method combines the $r$ previous computed values $y_k, y_{k-1}, \ldots, y_{k-r+1}$, which are approximations to the exact solution at the $r$ points $t_k, t_{k-1}, \ldots, t_{k-r+1}$, into a *multistep formula* to obtain the next iterate $y_{k+1}$. Adams methods are derived from the integral representation

$$y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} f(t, y(t))dt \tag{13}$$

which is replaced by the approximation

$$y_{k+1} = y_k + \int_{t_k}^{t_{k+1}} P_k(t)dt \tag{14}$$

where $P_k(t)$ is a polynomial approximating $f(t, y(t)) = y'(t)$ over the range $[t_k, t_{k+1}]$. Adams schemes (explicit Adams-Bashforth and implicit Adams-Moulton) are recovered by taking $P_k(t)$ to be the Lagrange interpolation polynomial in the ways described below.

#### 4.2.1 Adams-Bashforth (AB)

Since the approximations $y_k, y_{k-1}, \ldots, y_{k-r+1}$ are already known, the approximated values $f_k = f(t_k, y_k)$, $f_{k-1} = f(t_{k-1}, y_{k-1})$, $\ldots$, $f_{k-r+1} = f(t_{k-r+1}, y_{k-r+1})$ are also available and

the polynomial $P_k(t)$ is taken to be the Lagrange interpolation polynomial at the distinct nodes $t_k, t_{k-1}, \ldots, t_{k-r+1}$. This polynomial is

$$P_k^{AB}(t) = \sum_{i=0}^{r-1} f_{k-i} L_i^{AB}(t) \tag{15}$$

where

$$L_i^{AB}(t) = \prod_{\substack{j=0 \\ j \neq i}}^{r-1} (t - t_{k-j}) \Bigg/ \prod_{\substack{j=0 \\ j \neq i}}^{r-1} (t_{k-i} - t_{k-j}) \, . \tag{16}$$

Substituting (15) into (14) yields the formulation

$$\left\{ \begin{array}{lcl} y_{k+1} & = & y_k + h_k(\beta_{r-1}^{AB} f_k + \beta_{r-2}^{AB} f_{k-1} + \cdots + \beta_0^{AB} f_{k-r+1}) \\ f_{k+1} & = & f(t_{k+1}, y_{k+1}) \end{array} \right. \tag{17}$$

in which the $\{\beta_i^{AB}\}_{i=0}^{r-1}$ are given analytically by

$$\beta_i^{AB} = \frac{1}{h_k} \int_{t_k}^{t_{k+1}} L_i^{AB}(t) dt \tag{18}$$

and can be computed by recurrence. This formulation provides an explicit representation which allows us to immediately compute the approximation. The method is said to be *explicit* and its order is shown to be $r$.

### 4.2.2 Adams-Moulton (AM)

This scheme is obtained by choosing $P_k(t)$ to be the Lagrange interpolation polynomial not only at the nodes $(y_k, t_k), (y_{k-1}, t_{k-1}), \ldots, (y_{k-r+1}, t_{k-r+1})$ but also at the additional and *still unknown* node $(y_{k+1}, t_{k+1})$. In this case the polynomial obtained is

$$P_k^{AM}(t) = \sum_{i=0}^{r} f_{k+1-i} L_i^{AM}(t) \tag{19}$$

where

$$L_i^{AM}(t) = \prod_{\substack{j=0 \\ j \neq i}}^{r} (t - t_{k+1-j}) \Bigg/ \prod_{\substack{j=0 \\ j \neq i}}^{r} (t_{k+1-i} - t_{k+1-j}) \, . \tag{20}$$

This yields the formulation

$$\left\{ \begin{array}{lcl} y_{k+1} & = & y_k + h_k(\beta_r^{AM} f_{k+1} + \beta_{r-1}^{AM} f_k + \cdots + \beta_0^{AM} f_{k-r+1}) \\ f_{k+1} & = & f(t_{k+1}, y_{k+1}) \end{array} \right. \tag{21}$$

in which the $\{\beta_i^{AM}\}_{i=0}^{r}$ are given analytically by

$$\beta_i^{AM} = \frac{1}{h_k} \int_{t_k}^{t_{k+1}} L_i^{AM}(t) dt \tag{22}$$

and can be computed by recurrence. The unknown $y_{k+1}$ is involved in both sides of (21) and therefore at each step, a non-trivial equation of the form

$$y_{k+1} - h_k \beta_r^{AM} f(t_{k+1}, y_{k+1}) \quad = \quad g_k \tag{23}$$

where

$$g_k = y_k + h_k \sum_{i=0}^{r-1} \beta_i^{AM} f_{k+1-i} \tag{24}$$

must be solved for $y_{k+1}$. The method is said to be *implicit*. The usual approaches to solving (23) are root-finding techniques and fixed-point iterations. With the fixed-point approach, one proceeds by an iterative scheme of the form

$$\begin{cases} y_{k+1}^{[0]} = [\text{initial-estimate}] \\ y_{k+1}^{[i]} = g_k + h_k \beta_r^{AM} f(t_{k+1}, y_{k+1}^{[i-1]}), \quad i = 1, 2, \ldots \end{cases} \tag{25}$$

whose convergence is guaranteed when $h_k < 1/|L\beta_r^{AM}|$, where $L$ is the Lipschitz constant for $f$ (the upper limit on the magnitude of the Jacobian $f_y$). The initial estimate $y_{k+1}^{[0]}$ is usually the approximate solution computed by another explicit ODE solver (e.g., AB) and we said that the explicit and the implicit solvers form a *predictor-corrector* (PC) pair. The order of Adams-Moulton's method is $r+1$ and the number of function evaluations (E) depends on the number of steps before convergence of (25). However, it is also possible to deliberately interrupt the process after a given number iterations, say $s$, yielding another class of methods referred to as Adams–$P(EC)^sE$.

Adams methods (especially those of high order) are satisfactory from an accuracy point of view but they are not advocated for stiff problems since they are not $A(\theta)$-stable. This study includes two variants of variable step-size, variable order, variable coefficients Adams methods:

1. An implicit **ADAMS-PECE** scheme – the implementation is due to Shampine and Gordon [21] and is available on netlib. It is called *ode.f*.

2. An implicit Adams-Moulton scheme in which functional iteration is used in the non-linear phase. This variant is referred to as **VODPK/ADAMS**, for it comes from the VODPK package due to Byrne et al. [3]. It also is available on netlib.

### 4.3 Backward-Differentiation Formula (BDF)

BDF methods belong to a class of linear multi-step methods which combine the $r$ previous computed approximations into a multistep formula satisfying a *differential constraint*. Assume that we know the $y_k, y_{k-1}, \ldots, y_{k-r+1}$ which are approximations to the exact solution at the $r$ points $t_k, t_{k-1}, \ldots, t_{k-r+1}$. Let us consider the interpolation polynomial $P_k^{BDF}$ at the distinct nodes $(t_k, y_k), (t_{k-1}, y_{k-1}), \ldots, (t_{k-r+1}, y_{k-r+1})$ and also $(t_{k+1}, y_{k+1})$. The next approximation $y_{k+1}$ in BDF methods is determined by imposing the differential constraint

$$\frac{d}{dt} P_k^{BDF}(t_{k+1}) = f(t_{k+1}, y_{k+1}). \tag{26}$$

It can be shown that this yields the formulation

$$y_{k+1} - h_k \beta_r^{BDF} f(t_{k+1}, y_{k+1}) \quad = \quad \alpha_{r-1}^{BDF} f_k + \cdots + \alpha_0^{BDF} f_{k-r+1} \tag{27}$$

where $\beta_r^{BDF}$ and $\{\alpha_i^{BDF}\}_{i=0}^{r-1}$ are analytically known. The value of $y_{k+1}$ is retrieved after solving the non-trivial equation (27) with root-finding or fixed-point iteration techniques.

BDF methods are *implicit* with order $r$. It has been shown that their stability behaviors are satisfactorily only when $r = 1, 2, \ldots, 6$. These methods are suitable for the integration of stiff ODEs.

In our study we used the **VODPK/BDF** implementation which is variable step-size, variable order, variable coefficients, and which embodies several options in the management of the Jacobian. The variants considered are:

1. **VODPK/BDF_FI** — the non-linear phase is solved using functional iteration.
2. **VODPK/BDF_GM** — the non-linear problem is solved using a Newton root-finding scheme and each inversion of the Jacobian required during the Newton scheme was done *iteratively* via SPIGMR (Scaled Preconditioned Incomplete GMRES). The preconditioner chosen within GMRES was the incomplete ILU0 preconditioner (in all instances).
3. **VODPK/BDF_LU0** — the linear systems arising during the Newton scheme were solved *directly* using the *incomplete* ILU0 decomposition of the Jacobian.
4. **VODPK/BDF_LU** — the linear systems were solved *directly* using the *exact* (sparse) LU decomposition of the Jacobian.

# 5 The POWER CHALLENGEarray

All experiments were carried on a Silicon Graphics POWER CHALLENGEarray supercomputer. The SGI POWER CHALLENGE is a shared-memory multiprocessor machine, i.e., it consists of a collection of homogeneous processors (equivalent capabilities) which can execute distinct instruction streams in parallel. The collection is usually referred to as a rack (or box, frame, chassis) which reflects the physical structuring. Several racks of the SGI POWER CHALLENGE can be connected together and for clarity, Silicon Graphics, Inc., refers to one rack as a POWER CHALLENGE, whilst a cluster of racks is referred to as a POWER CHALLENGEarray.

The POWER CHALLENGEarray on which we performed our experiments is housed in the University of Queensland, Brisbane. It is composed of two racks referred to as `moreton` and `fraser` which have been configured so as to operate almost as two independent shared-memory systems. The user can request a job to be executed on either of the systems or can simply hand over to the operating system which will then automatically select the more suitable system depending on the workload and the availability of resources required by the job. Table 2 gives the particularities of their composition. Features not indicated on Table 2 are identical to the default settings as given on Table 1.

Each microprocessor chip is a MIPS R8000 superscalar RISC 90MHz 64-bit arithmetic chip. The 64-bit design encourages working in double precision (64 bit). In this way accuracy is improved and the maximum potential of the hardware is achieved. With the 4-way superscalar feature, the processor is capable of scheduling up to four instructions per cycle. Moreover, the floating-point unit can perform the dual operation consisting of a *floating-point multiply-add* (madd) in one cycle so that, cumulatively, the superscalar feature allows for executing up to six operations per clock cycle. A noteworthy innovation introduced by SGI has been to implant an index addressing mechanism (base-register plus

index-register) within the MIPS R8000 chip. Hence the extra calculations usually needed for pointer and index increments do not have a detrimental effect on performance. On aggregate, the peak performance of the University of Queensland's system is 7.2 Gflops.

| | | |
|---|---|---|
| CPU | Microprocessor | MIPS R8000 |
| | Architecture | 64-bit, 4-way superscalar, 6 operations |
| | Doubleword (64-bit) registers | 32 general purpose & 32 floating-point |
| | Data cache (D-cache) | 16 Kb (on-chip) |
| | Instruction cache (I-cache) | 16 Kb or 32 Kb (on-chip) |
| | Mutual (D&I) cache | 4 Mb (off-chip) |
| | Frequency | 90 MHz |
| | Peak performance | 360 Mflops |
| | Number of procs per rack | 1 – 18 |
| Memory | Operating system bulk | 32 Mb |
| | Memory size | 64 Mb – 16 Gb |
| | Interleaving | 1-, 2-, 4- or 8-way |
| | Memory bus | 256-bit (data) & 40-bit (address) |
| | Number of buses | 1 – 4 |
| | Memory bandwidth | 1.2 Gb/s |
| I/O | asynchronous I/O | yes |
| | memory-mapped I/O | yes |
| | direct I/O | yes |
| | Disk capacity | up to 3.2 Tb |
| | I/O bandwidth | 320 Mb/s |

Table 1: General technical features.

| | 2 racks | | |
|---|---|---|---|
| | moreton | fraser | Total |
| Processors | 16 procs | 4 procs | 20 procs |
| Instruction cache | 16 Kb | 16 Kb | |
| Peak performance | 5.76 Gflops | 1.44 Gflops | 7.2 Gflops |
| Memory size | 6 Gb | 2 Gb | 8 Gb |
| Interleaving | 8-way | 4-way | |
| Disk capacity | 94 Gb | 11 Gb | 105 Gb |
| I/O bandwidth | 180 Mb/s | | |

Table 2: Configuration at The University of Queensland.

For the development of numerically intensive applications, SGI provides high-performance scientific libraries collating a large amount of (serial and parallel) routines that have been optimised for its architecture. These include: BLAS, LAPACK, NAG, FFT, etc. Compilers are provided that support pivotal programming languages: f77, f90, High Performance Fortran (HPF), C, C++. The environment is equipped with (semi-)automatic tools that can generate a parallel version of sequential code automatically: PFA (Power Fortran Accelerator) and PCA (Power C Accelerator). The operating system is IRIX which is a derivative of UNIX System V. All experiments were undertaken in serial mode and the various codes were compiled with the optimizing switch at level 3 (i.e., -O3).

# 6 Numerical Comparisons

## 6.1 The Models

Infinitesimal generators (i.e., transition rate matrices) were generated for three different system models. Parameter values in each model were varied to provide matrices having different dimension and number of nonzero elements. A modified version of the MARCA Markov chain modelling package [24] was used for the generation. The complete suite of models may be obtained by anonymous ftp from N. Carolina State University. Access is obtained through *ftp.csc.ncsu.edu* and the files are in the directory *MARCA_Models*. This directory not only contains the models used in this study but others models as well. Its purpose is to provide a collection of matrices arising in Markov chain modelling on which comparative testing such as that performed in this study may be undertaken. Each model consists of an individual file containing Fortran77 code (e.g., *mutex.f*), a file containing input data, (e.g., *mutex_in*) and a makefile (e.g., *mutex_makefile*). In addition, a file containing Fortran77 code, *generate.f* is used in all models. To produce the infinitesimal generators corresponding to the input data, it suffices to execute *make -f mutex_makefile* (or copy the makefile into *Makefile* and execute *make*) and to run the resultant executable module, (e.g., *mutex*). Each of the input files contains information concerning the matrices that are generated, including size and number of nonzero elements, as well as information on how the models may be modified. The directory *MARCA_Models* also contains a README file containing further information. In the next three subsections we provide only brief information on each of the three models used, since the matrices themselves are easily retrievable.

**An NCD Queueing Network Example**

This model represents a multi-user interactive computer environment in which the system architecture is a time-shared, multiprogrammed, paged, virtual memory computer. The system consists of a set of $N$ terminals from which $N$ users generate commands; a central processing unit, (CPU); a secondary memory device, (SM); and a filing device, (FD). A queue of requests is associated with each device and the scheduling is assumed to be FCFS (First Come First Served). When a command is generated, the user at the terminal remains inactive until the system responds. Symbolically, a user having generated a command enters the CPU queue. The behavior of the process in the system is characterized by a compute time followed either by a page fault, after which the process enters the SM queue, or an input/output (file request) in which case the process enters the FD queue. Processes which terminate their service at the SM or FD queue return to the CPU queue. Symbolically, completion of a command is represented by a departure of the process from the CPU to the terminals. This model has been often used in previous studies, (see [16]). The matrices that are obtained are nearly-completely-decomposable, a factor that makes computation of stationary distribution by certain methods rather difficult. Our interest is in seeing how it affects methods for obtaining transient solutions. Parameter values of $N$ from 10 to 80 were chosen, giving rise to matrices ranging in size from 1,771 to 91,881 with corresponding numbers of nonzero elements of 47,381 and 623,241 respectively, (see Table 3).

**Mutual Exclusion Example**

In this model, $N$ distinguishable processes share a certain resource. Each of these processes alternates between a *sleeping* state and a resource *using* state. However, the number of processes that may concurrently use the resource is limited to $P$ where $1 \leq P \leq N$ so that when a process wishing to move from the sleeping state to the resource using state finds $P$ processes already using the resource, that process fails to access the resource and returns to the sleeping state. Notice that when $P = 1$ this model reduces to the usual mutual exclusion problem. When $P = N$ all of the the processes are independent. Let $\lambda^{(i)}$ be the rate at which process $i$ awakes from the sleeping state wishing to access the resource, and let $\mu^{(i)}$ be the rate at which this same process releases the resource when it has possession of it. In the examples chosen, we choose $N$ to have the values 12 and 16 and $P$ to have values 4, 8, 11 and 12. As indicated in Table 3, the matrices obtained range from order 2,517 with 20,949 nonzero elements to order 64,839 with 1,094,983 nonzero elements. The rates were chosen as $\lambda^{(i)} = 1/i$ and $\mu^{(i)} = i$, for $i = 1, \ldots, N$.

**A Leaky Bucket Example**

This is a model with applications to telecommunication modelling and in particular to modelling the dimensioning of intermediate buffer sizes that are simple, robust and rapid. The "Leaky Bucket" model is of this type. It consists of a buffer that receives packets at some probabilistic rate but sends them on at a rate that is deterministic. This model examines the effect of external arrivals to such a queue. The external flow consist of several sources from which a single one is isolated. The others are modeled as a single Poisson process. The buffer can accommodate a maximum of $C$ packets and is served by a single server with constant service rate of $D$. To simpliify the analysis $D$ is taken to be the unit of time. The arrival flow of the observed source is taken to be $T \times D$, where $T$ is an integer. The remainder of the arrival sources are modeled by a Poisson process of rate $\lambda$. Packets that arrive according to this Poisson process have priority over those of the constant source (Non-preemptive priority). The model was used with different values of $T$ and $C$ as indicated in Table 3.

## 6.2 Observations

The tables containing the results obtained with the different methods on the three sample models are presented below. Certain features of these tables contribute to a better analysis and appreciation of the results. The presentation of the methods in Sections 2 through 4 outlined the different basic operations that arise in each method. One operation common to all methods is the matrix-vector multiplication and so it is instructive to quantify the extent at which this operation permeates any given method. For this purpose, the number of matrix-vector multiplies performed in each method is recorded and is indicated in the tables under the label '*nmult*'. However, this number is of limited value if considered in isolation, i.e., without taking into account the context in which the operation arises. The UNIF method, for example, is designed to make use of (and only of) straight matrix-vector multiplication, while in EXPOMK, it is a sub-operation within the Arnoldi procedure where each matrix-vector product is associated with a costly Gram-Schmidt orthogonalization sweep. In general, in explicit methods, the operation is involved at one

level (when assembling the current approximation) whereas in implicit methods, the operation is involved at two levels (when assembling the current approximation and when solving the non-linear problem through functional iteration or Newton-GMRES for instance). In this respect, *nmult* helps us to gain an insight into the evolution of the overall execution. For example, a large *nmult* in RK78 may reflect that the step-sizes were too small (yielding too many integration steps) and in VODPK/BDF_GM, it may reflect either that the step-sizes where too small or that GMRES did not converge quickly. Similar interpretations apply for the other methods.

Another attribute of interest is the norm of the infinitesimal generator, for the computation of the matrix exponential is difficult when the norm is large. This difficulty can also be understood from an ODE standpoint. Indeed, with $A$ being an infinitesimal generator, $\|A\|$ corresponds to the Lipschitz constant for the ODE problem, so that when $t\|A\|$ is large, the problem is 'stiff' in the usual ODE sense (see, e.g., [4, 12]).

The integration domain $t$ in our experiments ranges from 1 to 1000 and thus, a certain degree of stiffness is gradually introduced. It is observed that some methods regularly failed when $t = 100$ and/or $t = 1000$. ('Failure' means that they did not succeed after the maximum allowable number of matrix-vector multiplications). These methods include RK78, ADAMS-PECE, VODPK/ADAMS, VODPK/BDF_FI. Their failure is somewhat in accordance with what theory predicts. Even though RK78 is by definition the only explicit method among them, all of these methods behave like explicit methods and therefore are more suited for non-stiff or moderately stiff problems. On the whole, although these method completed when $t = 1$ or $t = 10$, their performance is not really competitive with that of the other methods except in occasional instances. Notice the curious behavior of VODPK/BDF_FI which appears in the LEAKY2 table. We see that the method failed when $t = 10$ but succeeded when $t = 100$. This suggests that the functional iteration process was impeded around $t = 10$ (perhaps due to an irregularity) and that the enlargement of the integration domain induced a selection of a step-size which enabled the method to 'leap' out of the critical point.

It is interesting to contrast VODPK/BDF_LU0 and VODPK/BDF_LU. In principle the incomplete LU0 decomposition (in which one drops non-null elements whose locations are not within the sparsity pattern of $A$) is cheaper and has less information, while the complete (sparse) LU decomposition is costly but has more information. When repeated inversions of the Jacobian are done, the use of LU0 is cheaper by far than the use of LU and if the information within LU0 is sufficient, one may wonder if better performance will be obtained. It turns out this is exactly what happens. The performance of the LU0 variant is better in all cases. To understand why, observe that these matrices are sparse with small norms and so the growth factor within a normal LU decomposition is small. Thus the elements that will be created through a normal LU decomposition will be small (in magnitude) and although they add further calculations, their impact is negligible. When so many of them are created, the VODPK/BDF_LU is hampered by an excessive fill-in which is practically useless.

It now becomes meaningful to contrast VODPK/BDF_LU0 and VODPK/BDF_GM, given that the LU0 decomposition is a component of the Newton-GMRES phase – where it is used only as a preconditioner. Not surprisingly, we see now that VODPK/BDF_LU0 performs better than VODPK/BDF_GM except in a few cases, e.g., MUTEX1 when $t = 1000$, MUTEX3 when $t = 1000$, and MUTEX4 when $t = 100$. (Notice that the norms

of these particular cases are greater than the other norms in the series.) To summarize, the Jacobian of most of these problems can be inverted almost exactly using the LU0 decomposition without the need to resort to a complete LU decomposition or the Newton-GMRES root-finding process. Further overhead is not necessary.

The remaining methods are UNIF, EXPOMK and VODPK/BDF_LU0. In the LEAKY tables, we observe that UNIF performs best everywhere for $t = 1$, $t = 10$, and $t = 100$, and is followed by EXPOMK and VODPK/BDF_LU0. When $t = 1000$, EXPOMK performs best and is followed by UNIF (except in LEAKY3 where VODPK/BDF_LU0 is faster than UNIF). In the MUTEX tables, it may be seen that UNIF is best everywhere for $t = 1$ followed by EXPOMK, except in MUTEX5 where it is outperformed by EXPOMK. When $t = 10$, $t = 100$, and $t = 1000$, EXPOMK is fastest, followed by VODPK/BDF_LU0, save in the case of MUTEX1 in which VODPK/BDF_LU0 failed. In the NCD tables, the ranking is more irregular. However, it may be observed that VODPK/BDF_LU0 is best everywhere for $t = 1000$ and is followed by EXPOMK.

From all these observations, one may draw some general remarks. UNIF performs best on most of the cases in which $t\|A\|$ does not exceed 500 (e.g., in the LEAKY cases). However, as soon as $t\|A\|$ exceeds this threshold, EXPOMK finishes first (e.g., the MUTEX cases) but is challenged by VODPK/BDF_LU0 (e.g., the NCD cases). It should be noted that the NCD matrices are nearly block-diagonal matrices, so that the LU0 decomposition, while maintaining its low cost, has very rich information. With regard to reliabity and robustness specifically, EXPOMK was the only method to complete in all problems.

### 6.3 Numerical Results

Table 3 below provides the characteristics of the three models used in the experiments. The results themselves are presented in Tables 4 through 6.

| MODEL | Parameters | | Size | | | Norm |
|---|---|---|---|---|---|---|
| NCD | N | | $n$ | $nz$ | density(%) | $\|.\|_\infty$ |
| NCD1 | 20 | | 1,771 | 11,011 | 0.35 | 12.9 |
| NCD2 | 40 | | 12,341 | 81,221 | 0.05 | 35.5 |
| NCD3 | 50 | | 23,426 | 156,026 | 0.03 | 49.4 |
| NCD4 | 60 | | 39,711 | 266,631 | 0.02 | 64.8 |
| NCD5 | 80 | | 91,881 | 623,241 | 0.01 | 99.4 |
| MUTEX | N | P | $n$ | $nz$ | density(%) | $\|.\|_\infty$ |
| Mutex1 | 12 | 8 | 3,797 | 47,381 | 0.33 | 92.2 |
| Mutex2 | 12 | 11 | 4,095 | 53,223 | 0.32 | 92.2 |
| Mutex3 | 16 | 4 | 2,517 | 20,949 | 0.33 | 154.4 |
| Mutex4 | 16 | 8 | 39,203 | 563,491 | 0.04 | 154.4 |
| Mutex5 | 16 | 12 | 64,839 | 1,094,983 | 0.03 | 154.4 |
| LEAKY | T | C | $n$ | $nz$ | density(%) | $\|.\|_\infty$ |
| Leaky1 | 8 | 64 | 16,578 | 397,150 | 0.14 | 4.6 |
| Leaky2 | 12 | 64 | 24,898 | 596,826 | 0.10 | 4.6 |
| Leaky3 | 16 | 64 | 33,218 | 796,502 | 0.07 | 4.6 |
| Leaky4 | 24 | 64 | 49,858 | 1,195,854 | 0.05 | 4.7 |
| Leaky5 | 32 | 64 | 66,498 | 1,595,206 | 0.04 | 4.7 |

Table 3: Model Characteristics.

In all the experiments, the initial vector is $(1, 0, \ldots, 0)$ and the requested error tolerance on the computed solution is $10^{-10}$. The following exit indicators are used in the tables:

| | |
|---|---|
| $*$ | implies that a 'happy breakdown' occurred within EXPOMK. In other words, the method automatically detected that the steady-state solution had been reached and to continue the integration process was meaningless. |
| Failed | means the corresponding method failed (i.e., the method did not succeed after the maximum allowable number of matrix-vector multiplications and 'stiffness' was notified on exit). |
| $\infty$-Time | means the CPU time quota was exceeded. |
| $\infty$-Fill-in | means VODPK/BDF_LU could not proceed due to excessive fill-in generated by the sparse LU decomposition. |

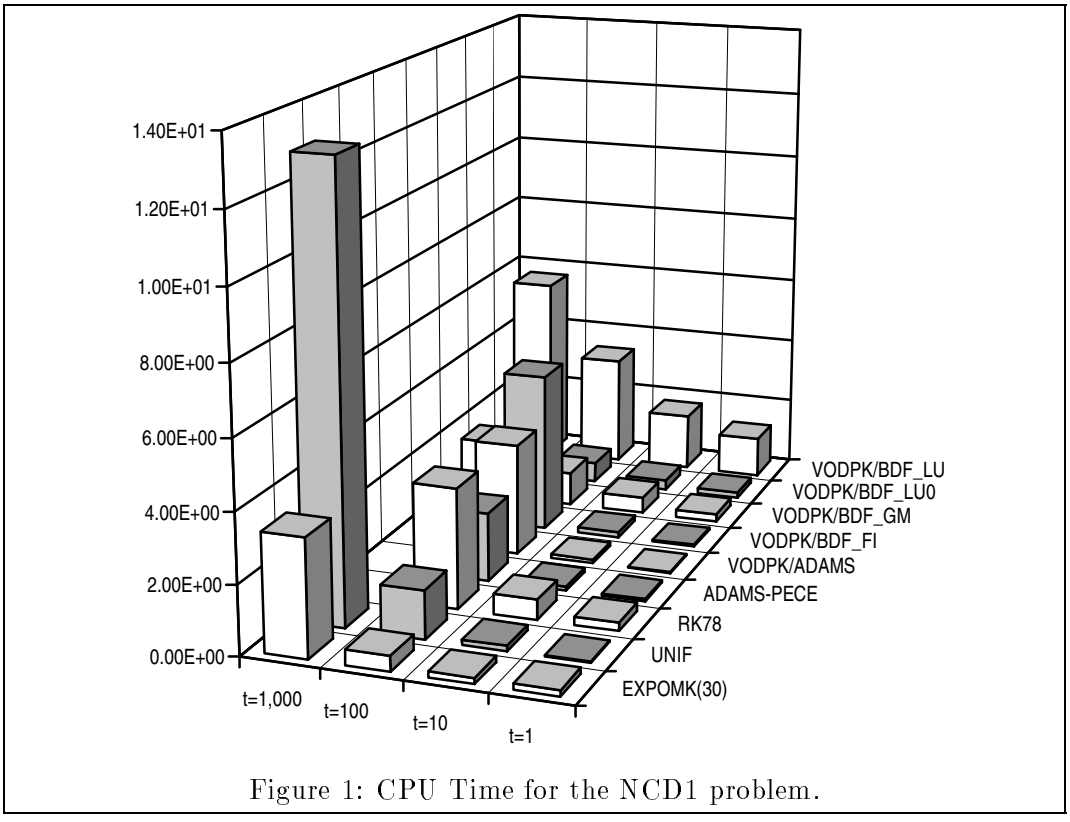


Figure 1: CPU Time for the NCD1 problem.

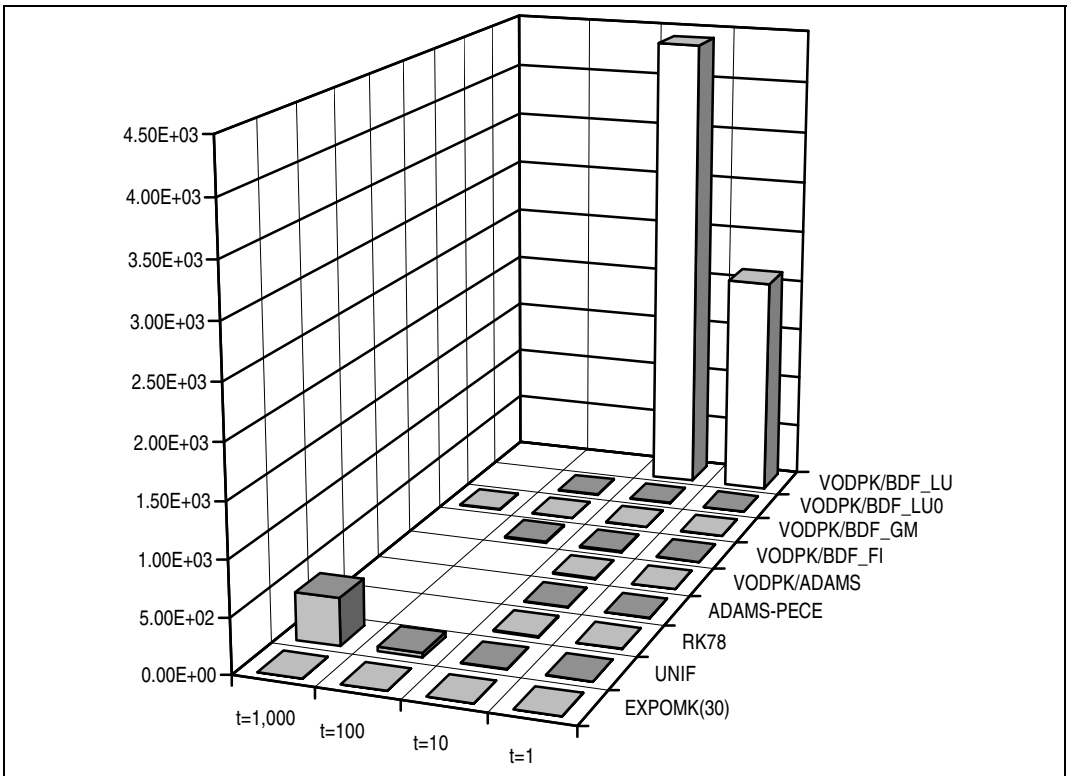


Figure 2: CPU Time for the MUTEX1 problem.

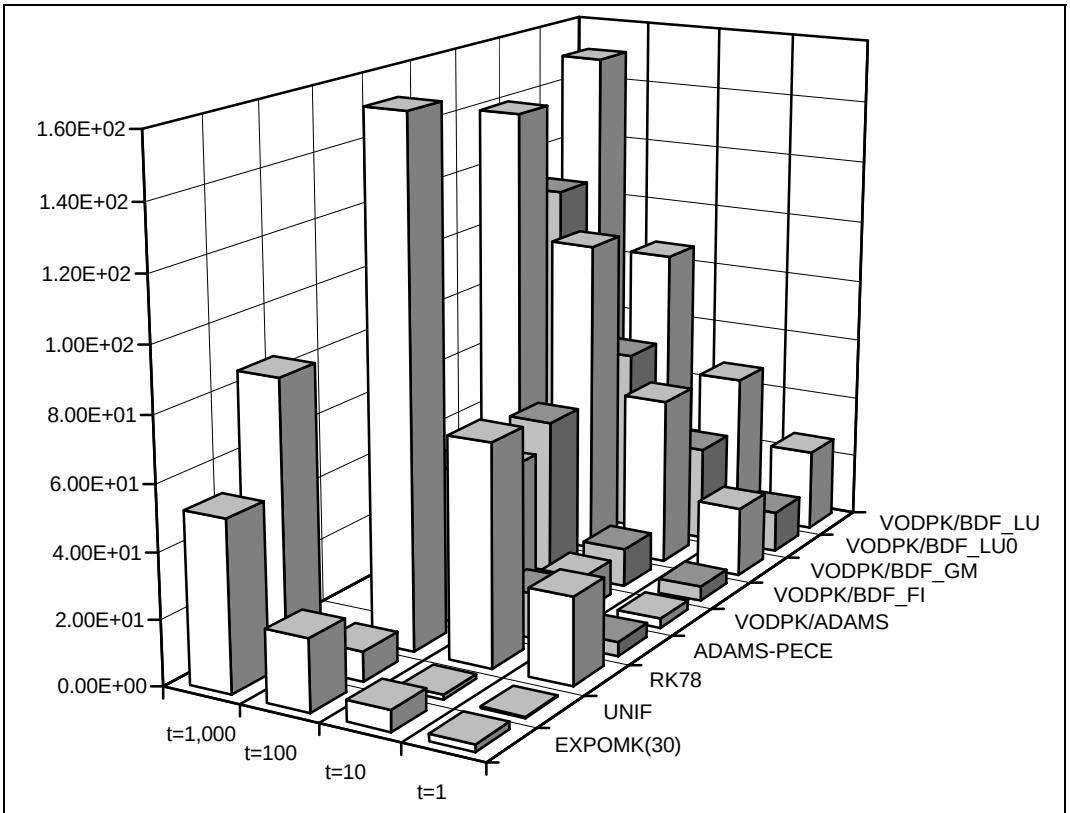


Figure 3: CPU Time for the LEAKY1 problem.

| **NCD1** $\|.\|_\infty = 12.9$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 1,771;\ nz = 11,011$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 62 | .18E+00 | 62 | .17E+00 | 155 | .44E+00 | 1178 | .34E+01 |
| UNIF | 28 | .34E-01 | 122 | .15E+00 | 1120 | .14E+01 | 10985 | .13E+02 |
| RK78 | 146 | .24E+00 | 368 | .59E+00 | 2225 | .35E+01 | Failed | |
| ADAMS-PECE | 33 | .64E-01 | 67 | .13E+00 | 1040 | .20E+01 | Failed | |
| VODPK/ADAMS | 15 | .34E-01 | 38 | .11E+00 | 1487 | .33E+01 | Failed | |
| VODPK/BDF_FI | 25 | .56E-01 | 60 | .15E+00 | 2096 | .47E+01 | Failed | |
| VODPK/BDF_GM | 34 | .22E+00 | 83 | .47E+00 | 174 | .10E+01 | 311 | .17E+01 |
| VODPK/BDF_LU0 | 22 | .16E+00 | 48 | .30E+00 | 97 | .59E+00 | 173 | .10E+01 |
| VODPK/BDF_LU | 22 | .12E+01 | 48 | .17E+01 | 97 | .33E+01 | 157 | .56E+01 |

| **NCD2** $\|.\|_\infty = 35.9$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 12,341;\ nz = 81,221$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 62 | .13E+01 | 62 | .12E+01 | 248 | .51E+01 | 2666 | .55E+02 |
| UNIF | 50 | .47E+00 | 310 | .28E+01 | 3024 | .27E+02 | 30260 | .27E+03 |
| RK78 | 172 | .22E+01 | 407 | .48E+01 | Failed | | Failed | |
| ADAMS-PECE | 37 | .54E+00 | 77 | .11E+01 | Failed | | Failed | |
| VODPK/ADAMS | 13 | .22E+00 | 27 | .53E+00 | Failed | | Failed | |
| VODPK/BDF_FI | 22 | .39E+00 | 55 | .11E+01 | Failed | | Failed | |
| VODPK/BDF_GM | 29 | .17E+01 | 71 | .35E+01 | 142 | .72E+01 | 286 | .15E+02 |
| VODPK/BDF_LU0 | 20 | .13E+01 | 42 | .26E+01 | 80 | .46E+01 | 157 | .93E+01 |
| VODPK/BDF_LU | 20 | .22E+02 | 42 | .32E+02 | 81 | .52E+02 | 141 | .83E+02 |

| **NCD3** $\|.\|_\infty = 49.4$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 23,426;\ nz = 156,026$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 62 | .34E+01 | 62 | .33E+01 | 279 | .15E+02 | 3689 | .20E+03 |
| UNIF | 62 | .10E+01 | 438 | .77E+01 | 4200 | .73E+02 | 41990 | .73E+03 |
| RK78 | 172 | .51E+01 | 433 | .12E+02 | Failed | | Failed | |
| ADAMS-PECE | 33 | .11E+01 | 75 | .26E+01 | Failed | | Failed | |
| VODPK/ADAMS | 12 | .46E+00 | 27 | .11E+01 | Failed | | Failed | |
| VODPK/BDF_FI | 19 | .70E+00 | 44 | .20E+01 | Failed | | Failed | |
| VODPK/BDF_GM | 25 | .39E+01 | 62 | .78E+01 | 142 | .18E+02 | 270 | .33E+02 |
| VODPK/BDF_LU0 | 17 | .31E+01 | 36 | .50E+01 | 79 | .11E+02 | 150 | .20E+02 |
| VODPK/BDF_LU | 17 | .44E+02 | 36 | .65E+02 | 78 | .13E+03 | 126 | .18E+03 |

| **NCD4** $\|.\|_\infty = 64.8$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 39,711;\ nz = 266,631$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 62 | .71E+01 | 62 | .72E+01 | 341 | .40E+02 | 4588 | .54E+03 |
| UNIF | 75 | .26E+01 | 576 | .20E+02 | 5511 | .20E+03 | 55080 | .20E+04 |
| RK78 | 185 | .13E+02 | 446 | .33E+02 | Failed | | Failed | |
| ADAMS-PECE | 33 | .27E+01 | 79 | .69E+01 | Failed | | Failed | |
| VODPK/ADAMS | 12 | .98E+00 | 26 | .24E+01 | Failed | | Failed | |
| VODPK/BDF_FI | 18 | .15E+01 | 41 | .38E+01 | Failed | | Failed | |
| VODPK/BDF_GM | 23 | .66E+01 | 60 | .14E+02 | 121 | .29E+02 | 242 | .58E+02 |
| VODPK/BDF_LU0 | 16 | .51E+01 | 36 | .95E+01 | 69 | .19E+02 | 135 | .36E+02 |
| VODPK/BDF_LU | 16 | .98E+02 | 36 | .15E+03 | 69 | .24E+03 | 119 | .35E+03 |

| **NCD5** $\|.\|_\infty = 99.4$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 91,881;\ nz = 623,241$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 62 | .21E+02 | 62 | .20E+02 | 465 | .15E+03 | 6665 | .18E+04 |
| UNIF | 101 | .14E+02 | 845 | .12E+03 | 8450 | .12E+04 | $\infty$-Time | |
| RK78 | 185 | .40E+02 | 459 | .10E+03 | Failed | | Failed | |
| ADAMS-PECE | 39 | .11E+02 | 87 | .25E+02 | Failed | | Failed | |
| VODPK/ADAMS | 12 | .33E+01 | 25 | .83E+01 | Failed | | Failed | |
| VODPK/BDF_FI | 17 | .48E+01 | 43 | .12E+02 | Failed | | Failed | |
| VODPK/BDF_GM | 21 | .16E+02 | 51 | .35E+02 | 119 | .75E+02 | 237 | .15E+03 |
| VODPK/BDF_LU0 | 15 | .13E+02 | 34 | .23E+02 | 69 | .49E+02 | 129 | .10E+03 |
| VODPK/BDF_LU | 15 | .36E+03 | 34 | .49E+03 | 69 | .76E+03 | 112 | .11E+04 |

Table 4: Results for Nearly-Completely-Decomposable Example.

| **MUTEX1** $\|.\|_\infty = 92.2$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 3,797$; $nz = 47,381$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 93 | .75E+00 | 217 | .16E+01 | *218 | .18E+01 | *218 | .15E+01 |
| UNIF | 138 | .46E+00 | 1304 | .44E+01 | 12936 | .43E+02 | 129370 | .43E+03 |
| RK78 | 667 | .33E+01 | 2521 | .12E+02 | Failed | | Failed | |
| ADAMS-PECE | 232 | .15E+01 | 1313 | .73E+01 | Failed | | Failed | |
| VODPK/ADAMS | 179 | .14E+01 | 1819 | .11E+02 | Failed | | Failed | |
| VODPK/BDF_FI | 171 | .11E+01 | 1895 | .95E+01 | 2770 | .13E+02 | Failed | |
| VODPK/BDF_GM | 142 | .24E+01 | 262 | .43E+01 | 299 | .49E+01 | 370 | .62E+01 |
| VODPK/BDF_LU0 | 80 | .16E+01 | 140 | .27E+01 | 314 | .57E+01 | Failed | |
| VODPK/BDF_LU | 80 | .21E+04 | 141 | .44E+04 | $\infty$-Time | | $\infty$-Time | |

| **MUTEX2** $\|.\|_\infty = 92.2$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 4,095$; $nz = 53,223$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 93 | .60E+00 | 217 | .14E+01 | *218 | .14E+01 | *218 | .14E+01 |
| UNIF | 154 | .50E+00 | 1503 | .49E+01 | 14952 | .49E+02 | 149770 | .49E+03 |
| RK78 | 667 | .27E+01 | 2826 | .12E+02 | Failed | | Failed | |
| ADAMS-PECE | 240 | .12E+01 | 1445 | .71E+01 | Failed | | Failed | |
| VODPK/ADAMS | 162 | .10E+01 | 2130 | .11E+02 | Failed | | Failed | |
| VODPK/BDF_FI | 176 | .12E+01 | 2199 | .12E+02 | Failed | | Failed | |
| VODPK/BDF_GM | 141 | .26E+01 | 249 | .44E+01 | 318 | .57E+01 | 464 | .79E+01 |
| VODPK/BDF_LU0 | 80 | .17E+01 | 131 | .27E+01 | 162 | .38E+01 | 451 | .87E+01 |
| VODPK/BDF_LU | 80 | .18E+04 | 131 | .36E+04 | 149 | .53E+04 | 149 | .71E+04 |

| **MUTEX3** $\|.\|_\infty = 154.4$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 2,517$; $nz = 20,949$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 93 | .40E+00 | *187 | .81E+00 | *187 | .85E+00 | *187 | .84E+00 |
| UNIF | 118 | .21E+00 | 1085 | .19E+01 | 10584 | .19E+02 | 105570 | .19E+03 |
| RK78 | 680 | .17E+01 | 2106 | .51E+01 | Failed | | Failed | |
| ADAMS-PECE | 238 | .72E+00 | 1139 | .33E+01 | Failed | | Failed | |
| VODPK/ADAMS | 135 | .54E+00 | 1389 | .44E+01 | Failed | | Failed | |
| VODPK/BDF_FI | 167 | .65E+00 | 1575 | .48E+01 | Failed | | Failed | |
| VODPK/BDF_GM | 150 | .15E+01 | 265 | .25E+01 | 289 | .26E+01 | 343 | .31E+01 |
| VODPK/BDF_LU0 | 84 | .98E+00 | 143 | .16E+01 | 311 | .29E+01 | 1575 | .14E+02 |
| VODPK/BDF_LU | $\infty$-Time | | $\infty$-Time | | $\infty$-Time | | $\infty$-Time | |

| **MUTEX4** $\|.\|_\infty = 154.4$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 39,203$; $nz = 563,491$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 124 | .18E+02 | 248 | .35E+02 | *249 | .35E+02 | *249 | .35E+02 |
| UNIF | 214 | .16E+02 | 1837 | .14E+03 | 18421 | .14E+04 | $\infty$-Time | |
| RK78 | 680 | .71E+02 | 3265 | .34E+03 | Failed | | $\infty$-Time | |
| ADAMS-PECE | 287 | .34E+02 | 1722 | .19E+03 | Failed | | Failed | |
| VODPK/ADAMS | 187 | .23E+02 | 2491 | .27E+03 | Failed | | Failed | |
| VODPK/BDF_FI | 217 | .27E+02 | 2670 | .30E+03 | Failed | | Failed | |
| VODPK/BDF_GM | 126 | .49E+02 | 222 | .81E+02 | 256 | .95E+02 | 404 | .14E+03 |
| VODPK/BDF_LU0 | 74 | .34E+02 | 120 | .51E+02 | 299 | .11E+03 | 401 | .17E+03 |
| VODPK/BDF_LU | $\infty$-Fill-in | | $\infty$-Fill-in | | $\infty$-Fill-in | | $\infty$-Fill-in | |

| **MUTEX5** $\|.\|_\infty = 154.4$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 64,839$; $nz = 1,094,983$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 124 | .34E+02 | *280 | .77E+02 | *280 | .78E+02 | *280 | .79E+02 |
| UNIF | 256 | .43E+02 | 2352 | .39E+03 | 23491 | .39E+04 | 234770 | .40E+05 |
| RK78 | 680 | .14E+03 | 4010 | .85E+03 | Failed | | Failed | |
| ADAMS-PECE | 316 | .78E+02 | 2114 | .49E+03 | Failed | | Failed | |
| VODPK/ADAMS | 225 | .59E+02 | Failed | | Failed | | Failed | |
| VODPK/BDF_FI | 253 | .64E+02 | Failed | | Failed | | Failed | |
| VODPK/BDF_GM | 123 | .92E+02 | 220 | .16E+03 | 255 | .18E+03 | 284 | .20E+03 |
| VODPK/BDF_LU0 | 72 | .63E+02 | 122 | .10E+03 | 363 | .26E+03 | 211 | .17E+03 |
| VODPK/BDF_LU | $\infty$-Fill-in | | $\infty$-Fill-in | | $\infty$-Fill-in | | $\infty$-Fill-in | |

Table 5: Results for Mutual Exclusion Example.

| **LEAKY1** $\|.\|_\infty = 4.6$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 16,578;\ nz = 397,150$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 31 | .22E+01 | 93 | .65E+01 | 310 | .22E+02 | *745 | .52E+02 |
| UNIF | 14 | .48E+00 | 45 | .15E+01 | 270 | .91E+01 | 2520 | .86E+02 |
| RK78 | 511 | .27E+02 | 1317 | .68E+02 | 3150 | .16E+03 | Failed | |
| ADAMS-PECE | 79 | .44E+01 | 229 | .14E+02 | 831 | .48E+02 | Failed | |
| VODPK/ADAMS | 47 | .29E+01 | 161 | .11E+02 | 664 | .41E+02 | Failed | |
| VODPK/BDF_FI | 76 | .46E+01 | 183 | .12E+02 | 804 | .49E+02 | Failed | |
| VODPK/BDF_GM | 109 | .22E+02 | 294 | .53E+02 | 583 | .10E+03 | 785 | .14E+03 |
| VODPK/BDF_LU0 | 54 | .13E+02 | 146 | .30E+02 | 293 | .58E+02 | 558 | .11E+03 |
| VODPK/BDF_LU | 54 | .26E+02 | 146 | .47E+02 | 297 | .86E+02 | 510 | .15E+03 |

| **LEAKY2** $\|.\|_\infty = 4.6$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 24,898;\ nz = 596,826$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 31 | .37E+01 | 93 | .11E+02 | 757 | .86E+02 | *838 | .10E+03 |
| UNIF | 14 | .97E+00 | 45 | .31E+01 | 268 | .19E+02 | 2505 | .17E+03 |
| RK78 | 550 | .51E+02 | 1395 | .13E+03 | 3384 | .32E+03 | Failed | |
| ADAMS-PECE | 79 | .80E+01 | 221 | .24E+02 | 938 | .10E+03 | Failed | |
| VODPK/ADAMS | 45 | .49E+01 | 155 | .19E+02 | 310 | .37E+02 | Failed | |
| VODPK/BDF_FI | 81 | .85E+01 | Failed | | 799 | .92E+02 | Failed | |
| VODPK/BDF_GM | 105 | .34E+02 | 785 | .14E+03 | 671 | .18E+03 | 1313 | .35E+03 |
| VODPK/BDF_LU0 | 61 | .23E+02 | 558 | .11E+03 | 348 | .11E+03 | 923 | .27E+03 |
| VODPK/BDF_LU | 61 | .56E+02 | 146 | .88E+02 | 348 | .18E+03 | 547 | .30E+03 |

| **LEAKY3** $\|.\|_\infty = 4.6$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 33,218;\ nz = 796,502$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 31 | .52E+01 | 93 | .16E+02 | 310 | .52E+02 | *900 | .15E+03 |
| UNIF | 14 | .14E+01 | 45 | .45E+01 | 266 | .27E+02 | 2490 | .25E+03 |
| RK78 | 550 | .73E+02 | 1447 | .19E+03 | 3514 | .47E+03 | Failed | |
| ADAMS-PECE | 79 | .11E+02 | 221 | .33E+02 | 990 | .15E+03 | Failed | |
| VODPK/ADAMS | 45 | .70E+01 | 151 | .26E+02 | 941 | .15E+03 | Failed | |
| VODPK/BDF_FI | 79 | .12E+02 | 177 | .29E+02 | 775 | .13E+03 | 775 | .13E+03 |
| VODPK/BDF_GM | 103 | .45E+02 | 272 | .10E+03 | 723 | .27E+03 | 723 | .27E+03 |
| VODPK/BDF_LU0 | 61 | .31E+02 | 147 | .65E+02 | 373 | .16E+03 | 373 | .16E+03 |
| VODPK/BDF_LU | 61 | .96E+02 | 147 | .14E+03 | 373 | .29E+03 | 834 | .64E+03 |

| **LEAKY4** $\|.\|_\infty = 4.7$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 49,858;\ nz = 1,195,854$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 31 | .72E+01 | 93 | .22E+02 | 341 | .80E+02 | *993 | .23E+03 |
| UNIF | 14 | .22E+01 | 45 | .71E+01 | 266 | .42E+02 | 2475 | .39E+03 |
| RK78 | 550 | .10E+03 | 1525 | .29E+03 | 3683 | .70E+03 | Failed | |
| ADAMS-PECE | 79 | .16E+02 | 221 | .47E+02 | 997 | .21E+03 | Failed | |
| VODPK/ADAMS | 44 | .97E+01 | 163 | .41E+02 | 1493 | .37E+03 | Failed | |
| VODPK/BDF_FI | 79 | .19E+02 | 174 | .44E+02 | 761 | .19E+03 | Failed | |
| VODPK/BDF_GM | 103 | .67E+02 | 259 | .15E+03 | 699 | .40E+03 | 1530 | .86E+03 |
| VODPK/BDF_LU0 | 61 | .45E+02 | 141 | .93E+02 | 362 | .23E+03 | 752 | .47E+03 |
| VODPK/BDF_LU | 61 | .17E+03 | 141 | .26E+03 | 362 | .52E+03 | $\infty$-Time | |

| **LEAKY5** $\|.\|_\infty = 4.7$ | $t = 1$ | | $t = 10$ | | $t = 100$ | | $t = 1000$ | |
|---|---|---|---|---|---|---|---|---|
| $n = 66,498;\ nz = 1,595,206$ | nmult | Time | nmult | Time | nmult | Time | nmult | Time |
| EXPOMK(30) | 31 | .10E+02 | 93 | .30E+02 | 310 | .98E+02 | *1458 | .46E+03 |
| UNIF | 14 | .30E+01 | 44 | .94E+01 | 264 | .57E+02 | 2475 | .53E+03 |
| RK78 | 550 | .14E+03 | 1590 | .41E+03 | Failed | | Failed | |
| ADAMS-PECE | 79 | .22E+02 | 221 | .66E+02 | 997 | .29E+03 | Failed | |
| VODPK/ADAMS | 44 | .14E+02 | 157 | .55E+02 | Failed | | Failed | |
| VODPK/BDF_FI | 68 | .21E+02 | 159 | .53E+02 | Failed | | Failed | |
| VODPK/BDF_GM | 103 | .84E+02 | 249 | .19E+03 | 679 | .49E+03 | 1936 | .13E+04 |
| VODPK/BDF_LU0 | 60 | .57E+02 | 135 | .12E+03 | 351 | .28E+03 | 909 | .71E+03 |
| VODPK/BDF_LU | 60 | .29E+03 | 135 | .42E+03 | 351 | .79E+03 | 823 | .17E+04 |

Table 6: Results for Leaky-Bucket Example.

# References


[1] H. Abdallah and R. Marie. The Uniformized Power Method for Transient Solutions of Markov Processes. *Computers Ops Res.*, 20(5):515–526, 1993.

[2] R.W. Brankin, I. Gladwell, and L. F. Shampine. RKSUITE: A Suite of Runge-Kutta Codes for the Initial Value Problem for ODEs, softreport 91-1. Technical report, Math. Dept., Southern Methodist University, Dallas, Texas, U.S.A, 1991.

[3] P. N. Brown, G. D. Byrne, and A. C. Hindmarsh. VODE: A Variable-coefficient ODE Solver. *SIAM J. Sci. and Stat. Comp.*, 20:1038–1051, 1989.

[4] K. Burrage. *Parallel and Sequential Methods for Ordinary Differential Equations.* Oxford University Press, Oxford, 1995.

[5] C. A. Clarotti. The Markov Approach to Calculating System Reliability : Computational Problems. In A. Serra and R. E. Barlow, editors, *International School of Physics «Enrico Fermi», Varenna, Italy*, pages 54–66, Amsterdam, 1984. North-Holland Physics Publishing.

[6] C. Fassino. *Computation of Matrix Functions.* PhD thesis, Università di Pisa, Genova - Udine, Italia, March 1993.

[7] E. Gallopoulos and Y. Saad. Efficient Solution of Parabolic Equations by Krylov Approximation Methods. *SIAM J. Sci. Stat. Comput.*, 13(5):1236–1264, 1992.

[8] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, second edition, 1989.

[9] W. Grassmann. Transient Solutions in Markovian Queueing Systems. *Comput. Opns. Res.*, 4:47–56, 1977.

[10] D. Gross and D. R. Miller. The Randomization Technique as Modelling Tool and Solution Procedure for Transient Markov Processes. *Operations Research*, 32(2):343–361, 1984.

[11] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I, Nonstiff Problems.* Springrer-Verlag, Berlin, 1987.

[12] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II, Stiff and Differential-algebraic Problems.* Springrer-Verlag, Berlin, 1991.

[13] M. Hochbruck and C. Lubich. On Krylov Subspace Approximation to the Matrix Exponential Operator. *submitted to SIAM J. Numer. Anal.*, 1995.

[14] M. Malhotra and K. S. Trivedi. Higher-order Methods for Transient Analysis of Stiff Markov Chains. Submitted, June 1991.

[15] C. B. Moler and C. F. Van Loan. Nineteen Dubious Ways to Compute the Exponential of a Matrix. *SIAM Review.*, 20(4):801–836, October 1978.

[16] B. Philippe, Y. Saad, and W. J. Stewart. Numerical Methods in Markov Chains Modeling. *Operations Research*, 40(6), 1992.

[17] B. Philippe and R. B. Sidje. Transient Solutions of Markov Processes by Krylov Subspaces. In W. J. Stewart, editor, *2nd International Workshop on the Numerical Solution of Markov Chains*, Raleigh, NC, USA, 1995.

[18] A. Reibman and K. Trivedi. Transient Analysis of Cumulative Measures of Markov Model Behavior. *Commu. Statist.-Stochastic Models*, 5(4):683–710, 1989.

[19] Y. Saad. Analysis of some Krylov Subspace Approximations to the Matrix Exponential Operator. *SIAM J. Numer. Anal.*, 29(1):208–227, 1992.

[20] Y. Saad. *Numerical Methods for Large Eigenvalue Problems.* John Wiley & Sons, Manchester Univ. Press, 1992.

[21] L. F. Shampine and M. K Gordon. *Solution of Ordinary Differential Equations – The Initial Value Problem.* W. H. Freeman and Co., San Francisco, 1975.

[22] R. B. Sidje. *Parallel Algorithms for Large Sparse Matrix Exponentials: Application to Numerical Transient Analysis of Markov Processes.* PhD thesis, University of Rennes 1, July 1994.

[23] R. B. Sidje. EXPOKIT. Software for Computing Large Sparse Matrix Exponentials. *in preparation*, 1996.

[24] W. J. Stewart. MARCA: Markov Chain Analyser. A Software Package for Markov Modelling. Department of Computer Science, N. Carolina State University, Raleigh, NC 27695-8206, 1988. Also: IEEE Computer Repository No R76 232, 1976. and IRISA Publication Interne No. 45, Université de Rennes, France.