SIAM J. SCI. COMPUT.
Vol. 29, No. 6, pp. 2562–2580

INEXACT UNIFORMIZATION METHOD FOR COMPUTING TRANSIENT DISTRIBUTIONS OF MARKOV CHAINS*

ROGER B. SIDJE†, KEVIN BURRAGE†, AND SHEV MACNAMARA†

Abstract. The uniformization method (also known as randomization) is a numerically stable algorithm for computing transient distributions of a continuous time Markov chain. When the solution is needed after a long run or when the convergence is slow, the uniformization method involves a large number of matrix-vector products. Despite this, the method remains very popular due to its ease of implementation and its reliability in many practical circumstances. Because calculating the matrix-vector product is the most time-consuming part of the method, overall efficiency in solving large-scale problems can be significantly enhanced if the matrix-vector product is made more economical. In this paper, we incorporate a new relaxation strategy into the uniformization method to compute the matrix-vector products only approximately. We analyze the error introduced by these inexact matrix-vector products and discuss strategies for refining the accuracy of the relaxation while reducing the execution cost. Numerical experiments drawn from computer systems and biological systems are given to show that significant computational savings are achieved in practical applications.

Key words. Markov chains, uniformization, inexact methods, relaxed matrix-vector

AMS subject classifications. 65C40, 65F50

DOI. 10.1137/060662629

1. Introduction.

Computing transient probabilities is an important problem in Markov chain modeling. Numerical techniques are based on solving the Chapman–Kolmogorov system of differential equations:

$$\text{(1)} \qquad \begin{cases} \dfrac{d\boldsymbol{w}(t)}{dt} = \boldsymbol{A}\boldsymbol{w}(t), & t \in [0,\ T], \\ \boldsymbol{w}(0) = \boldsymbol{v}, & \text{an initial probability distribution.} \end{cases}$$

The coefficient matrix $\boldsymbol{A}$ is an *infinitesimal generator* of order n, where n is the number of states in the Markov chain. Thus $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, with elements $a_{ij} \geq 0$ when $i \neq j$, and $a_{jj} = -\sum_{i=1, i\neq j}^{n} a_{ij}$, which means that $\boldsymbol{A}$ is a minus Z-matrix [2] with zero column sum (our notation uses column vectors and transposes the transition matrix; see, e.g., [18, section 8.5.2]). Of interest is the *transient solution*, $\boldsymbol{w}(t)$, which is given by the solution of (1) and is known to be

$$\text{(2)} \qquad \boldsymbol{w}(t) = e^{t\boldsymbol{A}}\boldsymbol{v}.$$

Since the matrix exponential can be full even when the original matrix is sparse, the practical computation of $e^{t\boldsymbol{A}}$ in full remains possible only when $\boldsymbol{A}$ is relatively small, i.e., when the number of states in the Markov chain does not exceed a few hundreds. Examples of such methods can be found in Moler and Van Loan [11]. In general, aspects such as size, stiffness, and accuracy are the main concerns when solving

*Received by the editors June 9, 2006; accepted for publication (in revised form) March 26, 2007; published electronically October 24, 2007. This work was supported by the National Facility of the Australian Partnership for Advanced Computing (APAC).

http://www.siam.org/journals/sisc/29-6/66262.html

†Department of Mathematics, Advanced Computational Modelling Centre, University of Queensland, Brisbane QLD 4072, Australia (rbs@maths.uq.edu.au, kb@maths.uq.edu.au, shev@maths.uq.edu.au). The second author thanks the Australian Research Council (ARC) for supporting him via a Federation Fellowship.

Markov chains numerically. Solution techniques for large-scale problems include general ODE solvers and Krylov-based methods [13, 14, 18]. However, Markovian analysts have traditionally used the uniformization method, which still remains very popular. Evidence suggests that it can work very well, especially on nonstiff problems [15], and this is one of the reasons why it remains one of the most widely used methods for computing transient solutions.

Because the matrix-vector product is the most time-consuming part of the uniformization method, the overall efficiency of the method in solving large-scale problems can be significantly enhanced if the matrix-vector product is made more economical. In this paper, we incorporate a new relaxation strategy into the method so as to perform matrix-vector products only approximately. We analyze the error introduced by these inexact matrix-vector products, and discuss strategies for refining the accuracy of the relaxation while reducing the execution cost. Numerical experiments drawn from computer systems and biological systems show that significant computational savings are achieved in practical applications.

The paper is structured as follows. Section 2 provides a brief overview of the uniformization method. Section 3 describes our proposed variant of the method based on matrix-vector products that are inexact. An analysis of the error is given, and strategies to refine the accuracy while reducing the execution cost are discussed. Section 4 presents some numerical results. Section 5 provides some concluding remarks.

2. The uniformization method. The uniformization (or randomization) algorithm is based on the evaluation of the ℓth partial Taylor series expansion of the matrix exponential [6, 7]. The length ℓ is determined so that the prescribed tolerance on the approximation is fulfilled. Since $\boldsymbol{A}$ is a minus Z-matrix (i.e., the diagonal elements of $\boldsymbol{A}$ are negative and the off-diagonal elements are nonnegative), a naive use of the expression

$$\boldsymbol{w}(t) = e^{t\boldsymbol{A}}v \approx \boldsymbol{v} + \frac{t\boldsymbol{A}}{1!}\boldsymbol{v} + \cdots + \frac{(t\boldsymbol{A})^{\ell}}{\ell!}\boldsymbol{v} + \cdots$$

is subject to severe roundoff errors due to terms of alternating signs. Thus the uniformization technique instead uses the modified formulation

$$\boldsymbol{w}(t) = e^{\alpha t(\boldsymbol{P}-\boldsymbol{I})}\boldsymbol{v} = e^{-\alpha t}e^{\alpha t\boldsymbol{P}}\boldsymbol{v}, \quad \boldsymbol{P} = \boldsymbol{I} + \frac{1}{\alpha}\boldsymbol{A}, \quad \alpha = \max_i |a_{ii}|,$$

where $0 \le \boldsymbol{P} \le 1$ componentwise and $\|\boldsymbol{P}\|_1 = 1$. The resulting truncated approximation

$$\overline{\boldsymbol{w}}(t) = \sum_{k=0}^{\ell} e^{-\alpha t}\frac{(\alpha t)^k}{k!}\boldsymbol{P}^k\boldsymbol{v} \tag{3}$$

involves only nonnegative terms and becomes numerically stable. If ε_{tol} denotes the prescribed error tolerance, the condition

$$\|\boldsymbol{w}(t) - \overline{\boldsymbol{w}}(t)\|_1 \le \varepsilon_{\text{tol}}$$

leads to a choice of ℓ such that

$$\sum_{k=0}^{\ell} e^{-\alpha t}\frac{(\alpha t)^k}{k!} \ge 1 - \varepsilon_{\text{tol}}. \tag{4}$$

The length ℓ may therefore be determined simply by adding up the above series until the inequality is met. In fact, practical implementations will subdivide the integration domain to prevent overflow issues. Specifically, choose θ in advance (say, $\theta = 100$), and set

$$m = \left\lceil \frac{\alpha t}{\theta} \right\rceil, \quad \bar{t} = \frac{t}{m};$$

then we have

$$\overline{\boldsymbol{w}}(t) = \left(e^{-\alpha \bar{t}} e^{\alpha \bar{t} \boldsymbol{P}}\right)^m \boldsymbol{v} = e^{-\alpha \bar{t}} e^{\alpha \bar{t} \boldsymbol{P}} \cdots e^{-\alpha \bar{t}} e^{\alpha \bar{t} \boldsymbol{P}} \boldsymbol{v}, \tag{5}$$

which is evaluated from left to right, and each step amounts essentially to our earlier discussion with different $\boldsymbol{v}$'s and a reduced $\bar{t}$ such that $\alpha \bar{t} \le \theta$. The overall scheme is summarized in the pseudocode below.

ALGORITHM 1: Uniformization$(t, A, v, \varepsilon_{\text{tol}})$.

{Compute $\boldsymbol{w} \approx \exp(t\boldsymbol{A})\boldsymbol{v}$}

1. $\alpha = \max_i |a_{ii}|$; $\boldsymbol{P} := \boldsymbol{I} + \boldsymbol{A}/\alpha$;
2. $\theta := 100$; $m := \lceil \alpha t/\theta \rceil$; $t := t/m$;
3. $s := \alpha t$; $r := e^{-s}$;
4. Choose ℓ to satisfy (4) ;
5. $\boldsymbol{w} := \boldsymbol{v}$;
6. **for** $i := 1 : m$ **do**
7. $\quad \boldsymbol{f} := \boldsymbol{w}$;
8. $\quad$ **for** $k := 1 : \ell$ **do**
9. $\quad\quad \boldsymbol{f} := \frac{s}{k}\boldsymbol{P}\boldsymbol{f}$;
10. $\quad\quad \boldsymbol{w} := \boldsymbol{w} + \boldsymbol{f}$;
11. $\quad$ **end**
12. $\quad \boldsymbol{w} := r\boldsymbol{w}$;
13. **end**

The popularity of uniformization is related to three main facts. First, its simplicity and malleability facilitate its implementation—only a matrix-vector product is needed per Horner-like iteration, with absolutely minimal extra storage (the matrix $\boldsymbol{P}$ need not be formed explicitly). Second, it works surprisingly well in a great variety of circumstances. Third, and perhaps most significantly, the transformation from $\boldsymbol{A}$ to $\boldsymbol{P}$ has a concrete interpretation. The matrix $\boldsymbol{P}$ is stochastic and is the transition matrix of a discrete time Markov chain (DTMC) that emulates the behavior of the continuous time Markov chain (CTMC), whose infinitesimal generator is $\boldsymbol{A}$. As described in Gross and Miller [7], this probabilistic interpretation allows deriving other parameters of interest. A shortcoming of uniformization is that ℓ is likely to be large, and thus (3) involves many matrix-vector multiplications, for large values of αt. Therefore, making the matrix-vector product economical, as we are proposing, can substantially decrease the execution time.

3. Inexact uniformization. Over recent years, there has been a growing interest in using inexact (or relaxed) matrix-products that are computed only to sufficient accuracy as needed [3, 16, 19]. Such interest has been motivated by applications where the matrix is not known exactly or is too expensive to apply. Most of the studies have focused, however, on Krylov subspace methods because by construction they are matrix-free and heavily dependent on matrix-vector products. The results obtained there have so far been quite promising [4, 10].

3.1. The inexact uniformization algorithm. Based on this evidence, we are interested in developing an inexact uniformization, where the matrix in statement 9 of Algorithm 1 is allowed to be $\boldsymbol{P} + \boldsymbol{E}_k$, where $\boldsymbol{E}_k$ is an error matrix at the kth step, and can change at each iteration. Thus, at each step, rather than computing $\boldsymbol{f}_k = \frac{\alpha t}{k}\boldsymbol{P}\boldsymbol{f}_{k-1}$, we compute an approximation $\tilde{\boldsymbol{f}}_k = \frac{\alpha t}{k}(\boldsymbol{P}+\boldsymbol{E}_k)\tilde{\boldsymbol{f}}_{k-1}$. Directly doing such a substitution is, however, fraught with the risk of corrupting the computations. To understand why, observe that if we are to compute $\tilde{\boldsymbol{f}}_k = \frac{\alpha t}{k}(\boldsymbol{P} + \boldsymbol{E}_k)\tilde{\boldsymbol{f}}_{k-1}$, the sequence of $\|\tilde{\boldsymbol{f}}_k\|$ grows before it decays, amplifying any error introduced and/or making it difficult to monitor the error. In general, this illustrates a possible pitfall when converting an existing method to using an inexact matrix-vector product. Thus we draw the attention of the reader to the fact that it may sometimes be necessary to recast an algorithm for it to fully benefit from the inexact matrix-vector idea.

In our present case, a small yet vital change is needed to make the approach successful. In this formulation, we split the computations using

$$\gamma_k = \frac{(\alpha t)^k}{k!}$$

and

$$\widetilde{\boldsymbol{f}}_k = (\boldsymbol{P} + \boldsymbol{E}_k)\widetilde{\boldsymbol{f}}_{k-1},$$

where $\boldsymbol{E}_k$ is the error matrix at step k. The updated algorithm is given below, showing the change (statement 11 of Algorithm 2).

ALGORITHM 2: InexactUniformization$(t, A, v, \varepsilon_{\text{tol}})$.

{Compute $\boldsymbol{w} \approx \exp(t\boldsymbol{A})\boldsymbol{v}$}

```
 1.  α = max_i |a_ii| ;   P := I + A/α ;
 2.  θ := 100 ;   m := ⌈αt/θ⌉ ;   t := t/m ;
 3.  s := αt ;   r := e^{-s} ;
 4.  Choose ℓ to satisfy (4) ;
 5.  w := v ;
 6.  for i := 1 : m do
 7.      γ := 1 ;
 8.      f := w ;
 9.      for k := 1 : ℓ do
10.          γ := γ s/k ;
11.          f :≈ Pf ;
12.          w := w + γf ;
13.      end
14.      w := rw ;
15.  end
```

If we were using *exact* arithmetic with $\boldsymbol{E}_k = 0$, we would generate the sequence of vectors $\boldsymbol{f}_k = \boldsymbol{P}\boldsymbol{f}_{k-1}$, which themselves would remain probability vectors, but with our relaxation approach, we compute

$$e^{\alpha t}\widetilde{\boldsymbol{w}} = \boldsymbol{v} + \sum_{k=1}^{\ell} \frac{(\alpha t)^k}{k!}(\boldsymbol{P} + \boldsymbol{E}_k)\cdots(\boldsymbol{P} + \boldsymbol{E}_1)\boldsymbol{v}$$

$$= \boldsymbol{v} + \sum_{k=1}^{\ell} \frac{(\alpha t)^k}{k!}(\boldsymbol{P}^k + \boldsymbol{\mathcal{E}}_k)\boldsymbol{v},$$

where

$$\boldsymbol{\mathcal{E}}_k = (\boldsymbol{P} + \boldsymbol{E}_k) \cdots (\boldsymbol{P} + \boldsymbol{E}_1) - \boldsymbol{P}^k.$$

Therefore the global error due to the inexact matrix-vector products is

$$\overline{\boldsymbol{w}} - \widetilde{\boldsymbol{w}} = -\sum_{k=1}^{\ell} e^{-\alpha t} \frac{(\alpha t)^k}{k!} \boldsymbol{\mathcal{E}}_k \boldsymbol{v}, \tag{6}$$

and the total error is

$$\|\boldsymbol{w} - \widetilde{\boldsymbol{w}}\|_1 = \|(\boldsymbol{w} - \overline{\boldsymbol{w}}) + (\overline{\boldsymbol{w}} - \widetilde{\boldsymbol{w}})\|_1 \le \varepsilon_{\text{tol}} + \|\overline{\boldsymbol{w}} - \widetilde{\boldsymbol{w}}\|_1. \tag{7}$$

In order to show that our inexact scheme is a viable approach, we have to show that the gap between the true solution and the computed solution, $\|\overline{\boldsymbol{w}} - \widetilde{\boldsymbol{w}}\|_1$, can be bounded in a practically useful way. This correlates with [16, 19], where a bound is established for the gap between the true residual and the computed residual in the context of linear systems. To obtain a meaningful bound in our context, it is helpful to consider the relationship between the global error and the local errors. This relationship is derived in the following result.

THEOREM 3.1. *Let $\widetilde{\boldsymbol{f}}_0 \equiv \boldsymbol{v}$ be the given starting vector, and for $k = 1, \ldots, \ell$ let $\widetilde{\boldsymbol{f}}_k$ be the approximate vector computed by the inexact matrix-vector product at step k, and let $\boldsymbol{\varepsilon}_k$ be the corresponding local error, that is,*

$$\boldsymbol{P}\widetilde{\boldsymbol{f}}_{k-1} = \widetilde{\boldsymbol{f}}_k + \boldsymbol{\varepsilon}_k, \quad k = 1, \ldots, \ell.$$

Then the global error of the inexact uniformization method satisfies

$$\overline{\boldsymbol{w}} - \widetilde{\boldsymbol{w}} = \sum_{k=1}^{\ell} e^{-\alpha t} \frac{(\alpha t)^k}{k!} \sum_{j=1}^{k} \boldsymbol{P}^{k-j} \boldsymbol{\varepsilon}_j \tag{8}$$

and therefore

$$\|\overline{\boldsymbol{w}} - \widetilde{\boldsymbol{w}}\|_1 \le \sum_{k=1}^{\ell} \|\boldsymbol{\varepsilon}_k\|_1. \tag{9}$$

Proof. We have

$$\begin{aligned} e^{\alpha t}\widetilde{\boldsymbol{w}} &= \widetilde{\boldsymbol{f}}_0 + \sum_{k=1}^{\ell} \frac{(\alpha t)^k}{k!} \widetilde{\boldsymbol{f}}_k \\ &= \boldsymbol{v} + \sum_{k=1}^{\ell} \frac{(\alpha t)^k}{k!} (\boldsymbol{P}\widetilde{\boldsymbol{f}}_{k-1} - \boldsymbol{\varepsilon}_k) \\ &= \boldsymbol{v} + \sum_{k=1}^{\ell} \frac{(\alpha t)^k}{k!} (\boldsymbol{P}^k \boldsymbol{v} - \boldsymbol{P}^{k-1}\boldsymbol{\varepsilon}_1 - \boldsymbol{P}^{k-2}\boldsymbol{\varepsilon}_2 - \cdots - \boldsymbol{\varepsilon}_k). \end{aligned}$$

Hence (8) follows immediately. And, using the fact that $\|\boldsymbol{P}\|_1 = 1$, we obtain

$$\|\overline{\boldsymbol{w}} - \widetilde{\boldsymbol{w}}\|_1 \le \sum_{k=1}^{\ell} e^{-\alpha t} \frac{(\alpha t)^k}{k!} \sum_{j=1}^{k} \|\boldsymbol{\varepsilon}_j\|_1 \le \left(\sum_{k=1}^{\ell} e^{-\alpha t} \frac{(\alpha t)^k}{k!} \right) \sum_{k=1}^{\ell} \|\boldsymbol{\varepsilon}_k\|_1 \le \sum_{k=1}^{\ell} \|\boldsymbol{\varepsilon}_k\|_1. \qquad \square$$

COROLLARY 3.2. *With the same notation as in Theoreom* 3.1, *the accumulated error due to the inexact matrix-vector products satisfies*

$$\boldsymbol{f}_k - \widetilde{\boldsymbol{f}}_k = -\boldsymbol{\mathcal{E}}_k \boldsymbol{v} = -\sum_{j=1}^{k} \boldsymbol{P}^{k-j} \boldsymbol{\varepsilon}_j$$

and therefore

$$\|\boldsymbol{f}_k - \widetilde{\boldsymbol{f}}_k\|_1 = \|\boldsymbol{\mathcal{E}}_k \boldsymbol{v}\|_1 \leq \sum_{j=1}^{k} \|\boldsymbol{\varepsilon}_j\|_1.$$

Proof. The result can be inferred from (6) and (8) and the uniqueness of the Taylor expansion. Another way is to expand $\boldsymbol{f}_k - \widetilde{\boldsymbol{f}}_k = \boldsymbol{P}(\boldsymbol{f}_{k-1} - \widetilde{\boldsymbol{f}}_{k-1}) - \boldsymbol{\varepsilon}_k$. ∎

The above analysis shows that the local errors in the matrix-vector products provide a useful mechanism for estimating the accumulated errors. And more significantly, (9) shows that the global error grows only linearly with the local errors, which implies that the algorithm is robust. When the domain is split as shown in (5), we do not have $\widetilde{\boldsymbol{f}}_0 = \boldsymbol{v}$ anymore, but the analysis can be easily extended to this case by setting $\widetilde{\boldsymbol{f}}_0 = \boldsymbol{v} + \boldsymbol{\varepsilon}_0$, with $\boldsymbol{v}$ understood as the current operand and $\boldsymbol{\varepsilon}_0$ as the error so far. We now discuss some ways of relaxing the matrix-vector product while controlling the local error.

3.2. Monitoring the inexact matrix-vector product. For very large problems, the probability sum condition $\|\boldsymbol{f}\|_1 = \mathbb{1}^T \boldsymbol{f} = 1$ implies that some of the components of $\boldsymbol{f}$ must necessarily be zero or at least very small. In other words, the most significant part of the support of the distribution at a given time is limited to a few components corresponding to the subset of most likely reachable states at that time. This subset changes over time, and hence the most significant part of the support changes over time too, but the size of the current subset remains small compared to the size of the full state space. While this may not always be the case, for example, if the distribution is uniform, there is a large class of important problems that exhibit this property, reflecting the fact that not all states are reached instantaneously. This observation is the cornerstone of our strategy for reducing the cost of the matrix-vector product. Let

$$\mathrm{Supp}(\boldsymbol{f}) = \{1 \leq j \leq n \mid f_j > 0\};$$

we can write the matrix-vector product in a column-oriented manner as

$$\boldsymbol{P}\boldsymbol{f} = \Big(\boldsymbol{I} + \frac{1}{\alpha}\boldsymbol{A}\Big)\boldsymbol{f} = \boldsymbol{f} + \frac{1}{\alpha}\begin{bmatrix}\boldsymbol{a}_1 & \boldsymbol{a}_2 & \dots & \boldsymbol{a}_n\end{bmatrix}\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} = \boldsymbol{f} + \sum_{j \in \mathrm{Supp}(\boldsymbol{f})} \frac{f_j}{\alpha}\boldsymbol{a}_j.$$

The key to efficiency resides therefore in tracking only a suitable approximation of this support. To this end, define the ε-support of $\boldsymbol{f}$ as

$$\mathrm{Supp}_\varepsilon(\boldsymbol{f}) = \{1 \leq j \leq n \mid f_j > \varepsilon\},$$

and let $\mathrm{Supp}_\varepsilon^C(\boldsymbol{f})$ be its complement, that is,

$$\mathrm{Supp}(\boldsymbol{f}) = \mathrm{Supp}_\varepsilon(\boldsymbol{f}) \cup \mathrm{Supp}_\varepsilon^C(\boldsymbol{f}).$$

We now use

$$\boldsymbol{P}\boldsymbol{f} \approx \boldsymbol{f} + \sum_{j \in \mathrm{Supp}_\varepsilon(\boldsymbol{f})} \frac{f_j}{\alpha} \boldsymbol{a}_j.$$

Since the essential part of the support is captured in this way, the inexact matrix-vector product may be quite a good approximation. Furthermore, the larger the problem size, the fewer (comparatively) the number of components of significance in the support, making the approach even more effective. The corresponding local error vector is therefore

$$\boldsymbol{\varepsilon} = \sum_{j \notin \mathrm{Supp}_\varepsilon(\boldsymbol{f})} \frac{f_j}{\alpha} \boldsymbol{a}_j = \sum_{j \in \mathrm{Supp}_\varepsilon^C(\boldsymbol{f})} \frac{f_j}{\alpha} \boldsymbol{a}_j.$$

Noting that $\|\boldsymbol{a}_j\|_1 = 2|a_{jj}|$ and $\|\boldsymbol{a}_j\|_\infty = |a_{jj}|$ since $a_{jj} = -\sum_{i=1, i\neq j}^{n} a_{ij}$, we have

$$\|\boldsymbol{\varepsilon}\|_1 \leq \sum_{j \in \mathrm{Supp}_\varepsilon^C(\boldsymbol{f})} \frac{2\varepsilon}{\alpha} |a_{jj}|, \qquad \|\boldsymbol{\varepsilon}\|_\infty \leq \sum_{j \in \mathrm{Supp}_\varepsilon^C(\boldsymbol{f})} \frac{\varepsilon}{\alpha} |a_{jj}|,$$

and since $\alpha \geq |a_{jj}|$, we get

$$\|\boldsymbol{\varepsilon}\|_1 \leq 2\varepsilon(|\,\mathrm{Supp}(\boldsymbol{f})| - |\,\mathrm{Supp}_\varepsilon(\boldsymbol{f})|), \qquad \|\boldsymbol{\varepsilon}\|_\infty \leq \varepsilon(|\,\mathrm{Supp}(\boldsymbol{f})| - |\,\mathrm{Supp}_\varepsilon(\boldsymbol{f})|),$$

where $|S|$ denotes the cardinality of the set S. Note that $|\,\mathrm{Supp}(\boldsymbol{f})| - |\,\mathrm{Supp}_\varepsilon(\boldsymbol{f})|$ is anticipated to be less than n, although we shall later use n as an upper bound for simplicity. Combining this analysis with Theorem 3.1, we obtain the following result.

THEOREM 3.3. *For $k = 1, \ldots, \ell$ let $\widetilde{\boldsymbol{f}}_k$ be the approximate vector computed by the inexact matrix-vector product at step k using the ε-support criteria. Then the global error of the inexact uniformization method satisfies*

$$\|\overline{\boldsymbol{w}} - \widetilde{\boldsymbol{w}}\|_1 \leq 2\varepsilon \sum_{k=1}^{\ell} (|\,\mathrm{Supp}(\widetilde{\boldsymbol{f}}_k)| - |\,\mathrm{Supp}_\varepsilon(\widetilde{\boldsymbol{f}}_k)|) \leq 2\varepsilon \sum_{k=1}^{\ell} (n - |\,\mathrm{Supp}_\varepsilon(\widetilde{\boldsymbol{f}}_k)|) \leq 2n\ell\varepsilon.$$

Remark 3.4. Our description has focused on applying a threshold on the support of a probability vector $\boldsymbol{f}$. It has the severity of not weighting the components with the actual matrix. Indeed since $|a_{jj}|/\alpha \leq 1$, it is possible to have an $f_j > \varepsilon$, and yet $f_j|a_{jj}|/\alpha \leq \varepsilon$, in which case we may ignore the column $\boldsymbol{a}_j$ as well. Thus we can trade the conciseness of the earlier formulation for a little extra saving in the matrix-vector product. In fact, in our experiments, our column-oriented matrix-vector product $\boldsymbol{A}\boldsymbol{f}/\alpha$ accumulates columns only where $f_j|a_{jj}|/\alpha > \varepsilon$. Thus our inexact matrix-vector product is effectively

$$\boldsymbol{P}\boldsymbol{f} \approx \boldsymbol{f} + \sum_{j \text{ s.t. } f_j \frac{|a_{jj}|}{\alpha} > \varepsilon} \frac{f_j}{\alpha} \boldsymbol{a}_j, \tag{10}$$

for which the corresponding local error vector is

$$\boldsymbol{\varepsilon} = \sum_{j \text{ s.t. } f_j \frac{|a_{jj}|}{\alpha} \leq \varepsilon} \frac{f_j}{\alpha} \boldsymbol{a}_j \leq \varepsilon \left| \left\{ j \text{ s.t. } f_j \frac{|a_{jj}|}{\alpha} \leq \varepsilon \right\} \right|.$$

We shall refer to this scheme as the *scaled* ε-support.

Remark 3.5. Another way to identify the most relevant part of the support is to determine the components of $\boldsymbol{f}$ that quickly sum to one, by, for example, sorting $\boldsymbol{f}$ in decreasing order and selecting the first few probabilities that quickly sum to one. However, for large problems, the sort, in $O(n \log n)$ operations, is potentially more expensive than the matrix-vector product, in $O(nz)$ operations, where nz is the number of nonzero entries of the matrix. A partial, heuristic sort is possible, but this is more involved.

Remark 3.6. Sparse matrices are usually stored using compact storage formats, of which a great variety exist [1]. Since our approach is column-oriented, it is crucial to use a format that fits the approach; otherwise the overhead of looking up and accessing the relevant columns of the matrix can outweigh any benefit of the relaxation algorithm. We recommend therefore the compressed column storage (CCS) format (cf. [1]), as it is ideally suited to our approach. Unfortunately, if the algorithm is implemented in MATLAB, which has its native sparse data type [5], referencing the relevant columns at each matrix-vector product hampers the vectorization and introduces a severe overhead. We observed that when performing the matrix-vector product $\boldsymbol{f} \leftarrow \boldsymbol{f} + \boldsymbol{A}\boldsymbol{f}/\alpha$, with $\boldsymbol{A}$ stored as a native sparse matrix in MATLAB, it is actually more efficient to zero the nonrelevant components of $\boldsymbol{f}$ than to index the corresponding columns of $\boldsymbol{A}$. That is, letting `diagA = abs(diag(A))`, it is more efficient in MATLAB to execute

```
support = find(diagA.*f > alpha*tol);
g(1:n) = 0;  g(support) = (1/alpha)*f(support);
f = f + A*g;
```

rather than

```
support = find(diagA.*f > alpha*tol);
f = f + A(:,support)*((1/alpha)*f(support));
```

Thus the advantage of the inexact matrix-vector product is lost in this environment. The ideal is of course to completely avoid arithmetic operations involving zeros or the negligible part of the support. The reader wishing to experiment with our approach in MATLAB should bear in mind these issues. Our numerical experiments were conducted using the CCS format in FORTRAN, which offers a finer granularity on controlling the operations without an excessive tradeoff. We anticipate that such a procedural language is the most appropriate for very large problems.

3.3. Detecting the stationary probability distribution. A particular feature of a Markov chain is that its transition matrix $\boldsymbol{A}$ has 0 as a simple eigenvalue, and for any other nonzero eigenvalue λ of $\boldsymbol{A}$, $\Re e(\lambda) < 0$. Hence the differential system (1) is stable and has a steady-state solution. It follows from this property that if $\tilde{\boldsymbol{w}}$ denotes the eigenvector associated with the eigenvalue 0, that is, $\boldsymbol{A}\tilde{\boldsymbol{w}} = 0$, then we have

$$\lim_{t \to \infty} e^{t\boldsymbol{A}} \boldsymbol{v} = \tilde{\boldsymbol{w}}.$$

The vector $\tilde{\boldsymbol{w}}$ is known as the *stationary* probability vector. When integrating therefore over a very long time t, it is possible for the uniformization method to tend to $\tilde{\boldsymbol{w}}$. However, implementations often shy away from attempting to detect this situation not only because it is not obvious how to do so cheaply and reliably, but also because of the unwarranted overhead of testing for an event that is more often than not unlikely to happen. However, we observed that when the stationary distribution

is reached (to within the error tolerance), a vast amount of computation is wasted unnecessarily. We can afford to check this in our approach. Indeed if in the course of our algorithm the stationary probability distribution is reached, then (10) implies that

$$\boldsymbol{P}\boldsymbol{f} \approx \boldsymbol{f}.$$

Detecting this situation enables further computational savings by avoiding the addition of unnecessary terms in the series. We implemented this check only in between the steps of (5). In other words, there are at most $m-1$ checkpoints in Algorithm 2, with the steady solution assumed if $\|\boldsymbol{P}\boldsymbol{f} - \boldsymbol{f}\|_\infty \leq \varepsilon_{\text{tol}}$, where $\boldsymbol{P}\boldsymbol{f}$ is itself approximated with the inexact scheme.

4. Numerical experiments. Experiments were undertaken at the National Facility of the Australian Partnership for Advanced Computing (APAC). We used its SGI Altix supercomputer infrastructure, but in a single node execution (Intel Itanium2 with 1.6 GHz). We implemented the code in FORTRAN using double precision (machine precision $= 0.2\,10^{-15}$), and used the Intel FORTRAN compiler (ifort) with a compilation switch to request its highest level of optimization (`-O3`). For comparison, we include the exact uniformization where we added the provision of detecting the steady solution as described in section 3.3. We also include the Krylov code (DGEXPV) from the EXPOKIT package [14], which implements Arnoldi's full orthogonalization method (FOM), where we set 30 as the dimension of the Krylov basis everywhere. This is labeled as Krylov(30) in the tables. The same ε_{tol} is used for the error control criteria. Refer to [14] for more details.

We compute the transient probability vector, $\boldsymbol{w}(t) = \exp(t\boldsymbol{A})\boldsymbol{v}$, for $t = 1,\ 10,\ 100$ over nine different data sets with $\boldsymbol{v} = (1, 0, \ldots, 0)^T$ of appropriate length. The choice of e_1 as the starting vector may seem too special, but it reflects the fact that we order the state space by reachability and that the Markov chain starts in the first state before evolving into other states. We repeated some of the calculations with a random starting vector. In general the computation time for the inexact uniformization increased from the case where e_1 is the starting vector, but it was still notably faster than the exact uniformization, although the differences in performance were less marked the more stringent the error tolerance became.

Recall from (7) that, in order to achieve a desired accuracy on the final result, we should set a comparable (or higher) accuracy for the inexact matrix-vector product, leading evidently to the exact uniformization if the matrix-vector product is computed to full accuracy. In the experiments, we set the same ε_{tol} value to control the inexact matrix-vector product (10). We run the experiments with the accuracy tolerance parameter ranging from $\varepsilon_{\text{tol}} = 10^{-5}$ to $\varepsilon_{\text{tol}} = 10^{-10}$. Note that the graphical charts use a logarithmic scale.

4.1. Mutual exclusion (MUTEX) problem. Mutual exclusion is a situation commonly encountered in computer systems. In this model, N distinguishable processes compete for a shared 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 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 experiments, we set the pair (N, P) to be $(16, 4)$, $(20, 8)$, and $(20, 16)$, with the rates chosen as $\lambda^{(i)} = 1/i$ and $\mu^{(i)} = i$, for $i = 1, \ldots, N$. The largest matrix in this series is of order $n = 1,047,225$ with $nz = 21,972,345$ nonzero elements, and $\|A\|_\infty = 232.65$. A modified version of the MARCA Markov chain modeling package [17] was used for the generation.

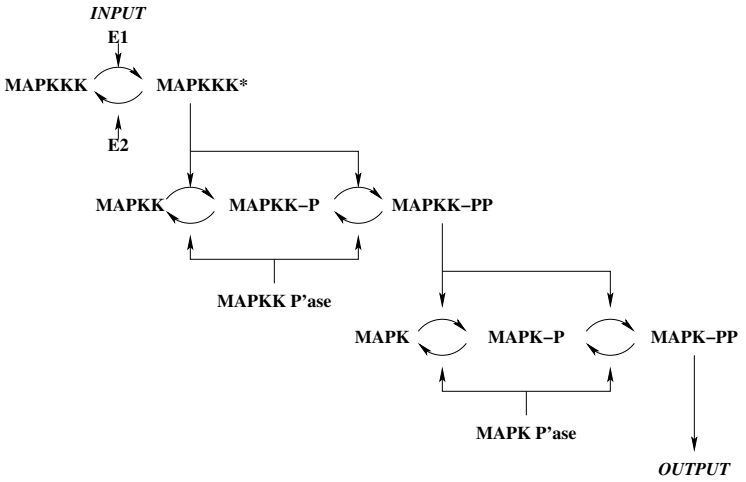
4.2. Nearly completely decomposable (NCD) queueing network. This model represents a multiuser interactive computer environment in which the system architecture is a time-shared, multiprogrammed, paged, virtual memory computer. The system consists of a set of N terminals, from which N users generate commands; 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 that 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 [12]). The matrices that are obtained are NCD, 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 were chosen to generate three matrices, the largest of which is of order $n = 91,881$ with $nz = 623,241$ nonzero elements (see Table 1).

4.3. Mitogen-activated protein kinase (MAPK) cascade. The mitogen-activated protein kinase (MAPK) cascade is implicated in a variety of signaling processes governing transitions in a cell's phenotype [9]. The MAPK cascade consists of 22 chemical species and 30 chemical reaction channels. A schematic of the cascade and the reactions, with their associated rate constants, is given in Figure 1. By simulating the dynamics of the MAPK cascade using the chemical master equation (CME), we obtain a Markov chain of very large dimension [4]. This example shows that it is possible to solve the CME for the MAPK cascade, at least for small numbers of molecules.

TABLE 1

Problem characteristics. The table shows n, the order of the transition rate matrix; its number of nonzero entries, $nz(\boldsymbol{A})$; the minimum and maximum number of nonzero entries on its columns, $\min nz(\boldsymbol{a}_j)$ and $\max nz(\boldsymbol{a}_j)$; and its L-$\infty$ norm (maximum row sum) $\|\boldsymbol{A}\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$. The MAPK and MUTEX problems are described in sections 4.3 *and* 4.1*, respectively, while the NCD problems are described in* 4.2.

	n	$nz(\boldsymbol{A})$	$\min nz(\boldsymbol{a}_j)$	$\max nz(\boldsymbol{a}_j)$	$\|.\|_\infty$
MAPK1	3,505	27,583	2	13	26.80
MAPK2	484,770	5,820,720	2	21	95.10
MAPK3	1,566,390	20,190,096	2	22	129.70
MUTEX1	2,517	20,949	5	17	154.44
MUTEX2	263,950	4,031,310	9	21	232.65
MUTEX3	1,047,225	21,972,345	17	21	232.65
NCD1	5,456	35,216	2	7	23.26
NCD2	12,341	81,221	2	7	35.51
NCD3	91,881	623,241	2	7	99.39



1	KKK + E1 $\xrightarrow{1.0}$ KKK.E1	16	KKPP + KKP'ase $\xrightarrow{1.0}$ KKPP.KKP'ase
2	KKK + E1 $\xleftarrow{1.0}$ KKK.E1	17	KKPP + KKP'ase $\xleftarrow{1.0}$ KKPP.KKP'ase
3	KKK.E1 $\xrightarrow{0.1}$ KKK* + E1	18	KKPP.KKP'ase $\xrightarrow{0.1}$ KKP + KKP'ase
4	KKK* + E2 $\xrightarrow{1.0}$ KKK.E2	19	KKPP + K $\xrightarrow{1.0}$ KKPP.K
5	KKK* + E2 $\xleftarrow{1.0}$ KKK.E2	20	KKPP + K $\xleftarrow{1.0}$ KKPP.K
6	KKK.E2 $\xrightarrow{0.1}$ KKK + E2	21	KKPP.K $\xrightarrow{0.1}$ KKPP + KP
7	KK + KKK* $\xrightarrow{1.0}$ KK.KKK*	22	KP + KP'ase $\xrightarrow{1.0}$ KP.KP'ase
8	KK + KKK* $\xleftarrow{1.0}$ KK.KKK*	23	KP + KP'ase $\xleftarrow{1.0}$ KP.KP'ase
9	KK.KKK* $\xrightarrow{0.1}$KKP+KKKx*	24	KP.KP'ase $\xrightarrow{0.1}$ K+KP'ase
10	KKP + KKP'ase $\xrightarrow{1.0}$ KKP.KKP'ase	25	KP + KKPP $\xrightarrow{1.0}$ KP.KKPP
11	KKP + KKP'ase $\xleftarrow{1.0}$ KKP.KKP'ase	26	KP + KKPP $\xleftarrow{1.0}$ KP.KKPP
12	KKP.KKP'ase $\xrightarrow{0.1}$ KK + KKP'ase	27	KP.KKPP $\xrightarrow{0.1}$ KPP + KKPP
13	KKP + KKK* $\xrightarrow{1.0}$ KKP.KKK*	28	KPP + KP'ase $\xrightarrow{1.0}$ KPP.KP'ase
14	KKP + KKK* $\xleftarrow{1.0}$ KKP.KKK*	29	KPP + KP'ase $\xleftarrow{1.0}$ KPP.KP'ase
15	KKP.KKK* $\xrightarrow{0.1}$ KKPP + KKK*	30	KPP.KP'ase $\xrightarrow{0.1}$ KP+KP'ase

Reactions and their rate constants

FIG. 1. *Schematic and detailed reactions in a mitogen-activated protein kinase (MAPK) cascade consisting of* 10 *coupled Michaelis–Menten enzyme kinetic models (adapted from* [9]*).*

In the largest experiment here, there can be anywhere from 0 to 5 molecules of each of the 22 species, so that an upper bound on the dimension of the full state space is 6^{22}. This reflects the so-called *curse of dimensionality*, which motivates the need for efficient solution techniques in systems biology. Since not all of these states are reachable in one step, the matrix is sparse and is generated based on the reachability from seven key species: the MAPK, MAPK kinase, MAPK kinase-kinase (KKK), and enzymes denoted E1, E2, MAPK phosphatase, and MAPKK phosphatase.

4.4. Discussion. The results of our extensive simulations are presented in the appendix. These detailed tables show how the results are affected by varying the choice of ε_{tol} (which determines the ε-support) from $\varepsilon_{\text{tol}} = 10^{-5}$ to $\varepsilon_{\text{tol}} = 10^{-10}$. The reported computation times (in seconds) over such a range give an indication of the extra work required per decimal place of accuracy. For each test problem, we present $w_1(t)$ and $w_n(t)$, the first and last components of $\boldsymbol{w}(t)$, as computed by the three different

methods. The results show that the inexact method produces results accurate to the specified tolerance.

In order to encapsulate the simulation results given in the appendix we present a number of figures. Figure 2 shows how the probability vector and the support evolve over time for each of the three classes of problems. Clearly the three problems have very different characteristics.

In Figure 3, we present timing results for the three classes of problems with $\varepsilon_{\text{tol}} = 10^{-7}$ and $\varepsilon_{\text{tol}} = 10^{-10}$. We see that the inexact uniformization (inexactUnif) outperforms the exact uniformization (exactUnif) in nearly all cases (except one where the exactUnif detects equilibrium early). Furthermore, for integration intervals that are not too large, the inexactUnif is very competitive with the Krylov approach. However, as the integration interval increases, the inexactUnif becomes less competitive. One of the reasons for this is that the Krylov method is very efficient in detecting equilibrium.

In Figure 4, we compare the numbers of uMATVEC between the three methods.

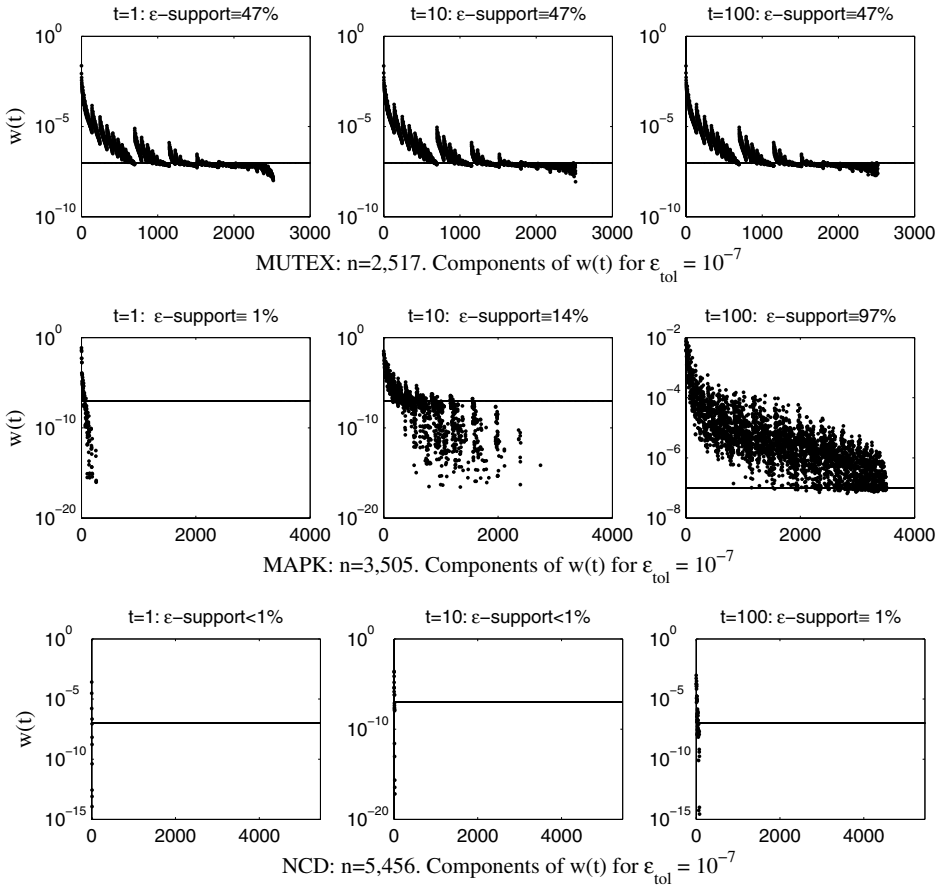


FIG. 2. *Evolution of the probability vector over time. The figure shows the* scaled *components as explained in Remark* 3.4. *Components above the straight line are those in the scaled ε-support, and thus they are those involved in the inexact matrix-vector product. The reported percentage is their proportion with respect to the number of components. It is seen that the ε-support changes over time, and various situations can happen depending on the matrix.*

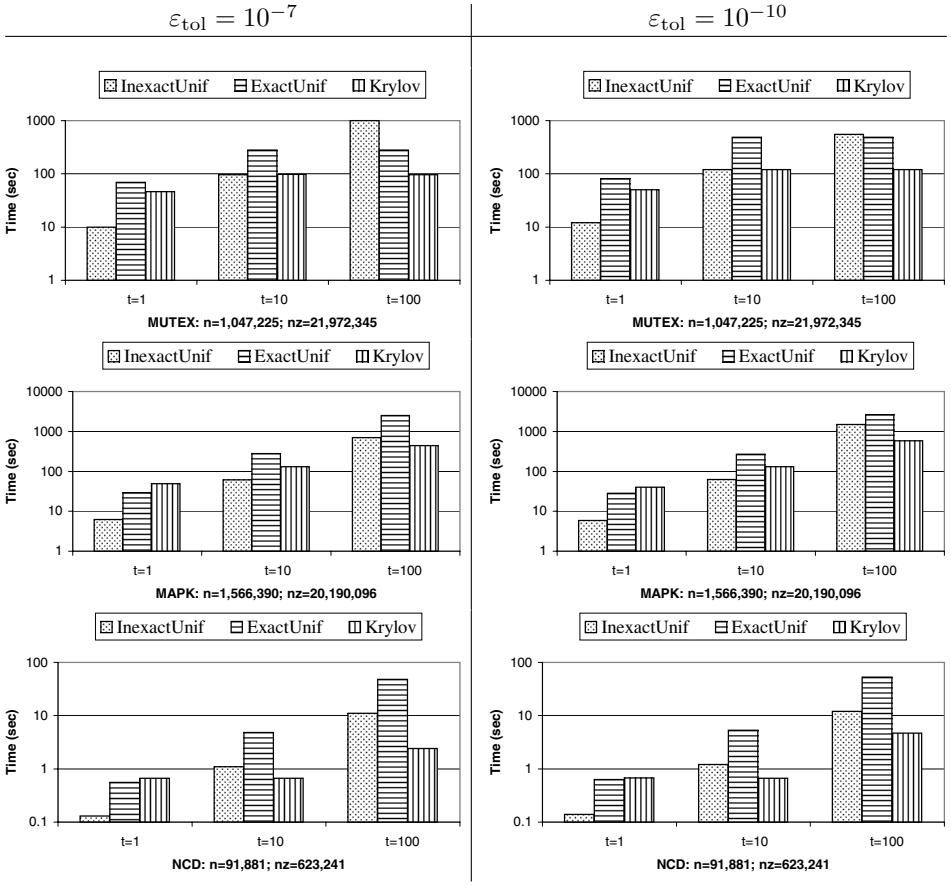


FIG. 3. *Timings for the largest matrices in the series when* $\varepsilon_{\text{tol}} = 10^{-7}$ *and* $\varepsilon_{\text{tol}} = 10^{-10}$. *At short integration intervals,* $t = 1$ *and* $t = 10$, *the times are all small in the NCD plots, but on the other problems it is seen that the inexactUnif is effective. Note in the MUTEX plots that the steady state occurs early at* $t = 10$. *The Krylov method is able to detect this almost immediately (due to an invariant subspace in its Arnoldi loop) and keeps a constant execution time as soon as* $t \geq 10$. *The exactUnif detects equilibrium later than the Krylov method but sooner than the inexact method.*

uMATVEC measures one unit of a matrix-vector product. This amounts to the standard matrix-vector product for the normal matrix-free methods, exactUnif and Krylov. However, in the case of inexactUnif, since not all columns participate in its matrix-vector operation, the number of uMATVEC is obtained by counting the number of participating columns over the entire run and dividing this by the order n.

In all cases, we see that there are substantially fewer uMATVEC operations with the inexactUnif technique. However, in this case, a single uMATVEC becomes increasingly more expensive as more and more conditional checks need to be processed. Indeed, this is one of the reasons why the inexactUnif becomes less competitive with the Krylov technique as the integration domain increases. It should be recalled that inexactUnif still uses almost the same number of iterations as exactUnif. Thus uMATVEC is a simple metric that gives a unified way of contrasting the methods in terms of matrix-vector products while giving a sense of the number of columns discarded by inexactUnif. Since it does not capture the fact that we still have to loop over the entire matrix using conditional checks to decide whether to skip over columns

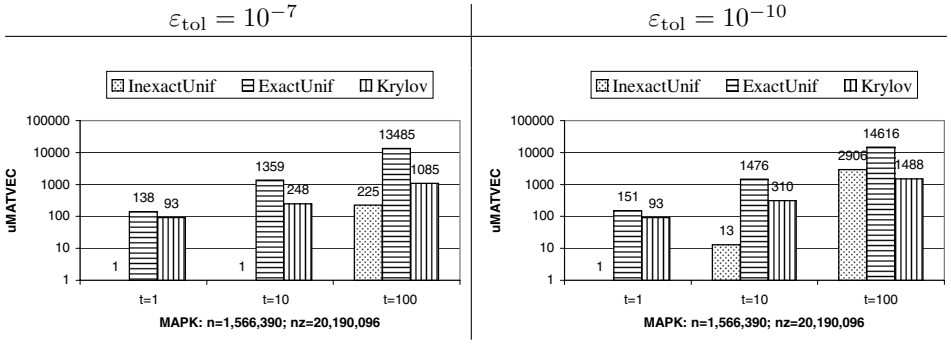


FIG. 4. *Numbers of uMATVEC operations for the largest MAPK matrix when* $\varepsilon_{\text{tol}} = 10^{-7}$ *and* $\varepsilon_{\text{tol}} = 10^{-10}$. *uMATVEC measures one unit of a matrix-vector product. This amounts to the standard matrix-vector product for the normal matrix-free methods, exactUnif and Krylov. However, in the case of inexactUnif, since not all columns participate in the matrix-vector operation, the reported number of uMATVEC operations is obtained by summing the number of participating columns over the entire run and dividing this by the order* n.

TABLE 2

*Distribution of the numbers of nonzeros in the columns. For example, the MAPK*1 *matrix has one column with two nonzero elements,* 11 *columns with three nonzero elements, and so forth. The location of the nonzero elements in these columns is not relevant.*

$nz(\boldsymbol{a}_j)$	Numbers of columns that have the indicated numbers of nonzeros, $nz(\boldsymbol{a}_j)$								
	MAPK1	MAPK2	MAPK3	MUTEX1	MUTEX2	MUTEX3	NCD1	NCD2	NCD3
2	1	1	1	0	0	0	3	3	3
3	11	17	17	0	0	0	87	117	237
4	61	140	151	0	0	0	407	742	3,082
5	179	668	880	1,820	0	0	87	117	237
6	426	2,617	3,708	0	0	0	1,218	2,223	9,243
7	712	7,664	12,162	0	0	0	3,654	9,139	79,079
8	866	18,460	32,120	0	0	0			
9	717	35,511	69,410	0	125,970	0			
10	386	57,179	125,404	0	0	0			
11	125	75,710	189,459	0	0	0			
12	20	84,437	242,549	0	0	0			
13	1	77,521	261,368	0	0	0			
14		59,439	238,129	0	0	0			
15		36,983	181,494	0	0	0			
16		18,630	115,291	0	0	0			
17		7,234	59,765	697	0	4,845			
18		2,100	24,713		0	0			
19		410	7,800		0	0			
20		47	1,724		0	0			
21		2	235		137,980	1,042,380			
22			10						

(together with scaling the components as indicated before), the reader should look at the value of inexactUnif in conjunction with that of exactUnif, keeping in mind the increasing number of conditional checks for matrices of large dimension over thousands of steps. This explains why the execution time of inexactUnif, seen in Figure 3, is not directly proportionate to its number of uMATVEC operations. Rather, the pay-off of the inexact matrix-vector routine is worthwhile if the discarded columns have sufficient nonzero elements so that the savings in floating point operations exceeds the conditional checks. Refer to Table 2 for the distribution of the numbers of nonzeros in the columns.

We can draw some other conclusions based on our extensive numerical tests.

- The inexactUnif is a worthy challenger not only to the exactUnif but also to the Krylov method in a number of cases. In general the inexactUnif method brings about a 10-fold improvement over the uniformization method on problems of significant dimension and a wide range of the accuracy tolerance parameter ε_{tol}, to the point of even becoming faster than the Krylov method at short integration domains, $t = 1$ and $t = 10$.
- Only when $t = 100$ does the Krylov method become the fastest due to its inherent higher convergence rate [8]. This comes at the price of the extra storage for the Krylov basis (which may not be possible on all computer systems for problems this large).
- For the MUTEX problems the steady state occurs early at $t = 10$. Since the Krylov method is able to detect this almost immediately (due to an invariant subspace in its Arnoldi loop), this explains why its uMATVEC and execution time are constant as soon as $t \geq 10$. The exactUnif detects equilibrium later than the Krylov method, but sooner than the inexact method. Hence the inexactness tends to delay the detection of the steady state. This can be clearly seen on the MUTEX plots in Figure 3. The inexact method took many more iterations, while the other methods finished early, thus keeping their constant execution time.
- The inexact method is able to track the ε-support and exclude impressive numbers of columns, making the method worthwhile and very efficient on problems with a large number of nonzero elements per column. Understandably, it is disadvantaged on problems with very few nonzero elements per column (such as a bidiagonal or tridiagonal matrix, for example). Determining whether to avoid a column may not be worth the effort on a nearly empty column (and testing this for all columns of a large matrix over thousands of steps would incur a substantial overhead too).

5. Conclusion. We have presented a new approach to substantially reducing the cost of the uniformization method. Our new algorithm consists of relaxing the matrix-vector products so as to perform them only to sufficient accuracy. Our approach is of practical value as it tells precisely how this relaxation can be performed, with the added benefit of retaining the minimalism and malleability that make the method so popular with Markovian analysts. We characterized the ensuing global error and showed that it grows at most linearly with the local errors introduced by the inexact matrix-vector products. We discussed strategies for refining the accuracy while reducing the execution cost. It has been observed that using our technique in an environment such as MATLAB does not reap the full benefits because of the overhead of operating on its native sparse data structure from the scripting level. However, using the appropriate sparse storage format in FORTRAN, numerical experiments showed that significant computational savings can be achieved in practical applications without a marked deterioration of accuracy.

Appendix A. MAPK results. Detailed results with the accuracy tolerance parameter ranging from $\varepsilon_{\text{tol}} = 10^{-5}$ to $\varepsilon_{\text{tol}} = 10^{-10}$, and the first and last components of $\boldsymbol{w}(t)$ as computed by the different methods at various times. See section 4.4 for how to interpret the results.

A.1. MAPK1: $n = 3,505$; $nz = 27,583$; $||.||_\infty = 26.80$.

	$\varepsilon_{\text{tol}} = 10^{-5}$		$\varepsilon_{\text{tol}} = 10^{-6}$		$\varepsilon_{\text{tol}} = 10^{-7}$		$\varepsilon_{\text{tol}} = 10^{-8}$		$\varepsilon_{\text{tol}} = 10^{-9}$		$\varepsilon_{\text{tol}} = 10^{-10}$	
	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time
$t = 1$												
inexactUnif	1	2.0E−03	1	2.0E−03	1	2.0E−03	1	2.9E−03	2	2.9E−03	3	2.9E−03
exactUnif	36	8.8E−03	38	9.8E−03	41	8.8E−03	43	9.8E−03	45	9.8E−03	47	1.1E−02
Krylov(30)	62	2.1E−02	62	2.0E−02	62	2.0E−02	62	2.0E−02	62	2.0E−02	62	1.9E−02
$t = 10$												
inexactUnif	6	1.4E−02	12	1.5E−02	23	1.9E−02	46	2.3E−02	82	3.2E−02	126	4.1E−02
exactUnif	242	5.7E−02	252	6.0E−02	262	6.2E−02	270	6.3E−02	278	6.5E−02	286	6.6E−02
Krylov(30)	124	3.9E−02	124	3.8E−02	124	3.8E−02	124	3.9E−02	155	4.8E−02	155	4.7E−02
$t = 100$												
inexactUnif	473	2.3E−01	1195	3.9E−01	1823	5.2E−01	2186	6.0E−01	2417	6.5E−01	2572	6.8E−01
exactUnif	1795	4.2E−01	2448	5.7E−01	2533	5.9E−01	2601	6.1E−01	2686	6.3E−01	2754	6.4E−01
Krylov(30)	310	9.8E−02	341	1.1E−01	372	1.2E−01	403	1.3E−01	434	1.4E−01	496	1.5E−01

	$w_1(t)$			$w_n(t)$		
	$t = 1$	$t = 10$	$t = 100$	$t = 1$	$t = 10$	$t = 100$
$\varepsilon_{\text{tol}} = 10^{-7}$						
inexactUnif	1.552574100E−01	6.088158784E−02	1.532050930E−02	0.000000000E+00	0.000000000E+00	2.690948138E−07
exactUnif	1.552574114E−01	6.088159971E−02	1.532115373E−02	1.996756526E−28	1.968854041E−13	1.495909999E−07
Krylov(30)	1.552574169E−01	6.088160594E−02	1.532117162E−02	2.154610103E−28	1.968857655E−13	1.495910210E−07
$\varepsilon_{\text{tol}} = 10^{-10}$						
inexactUnif	1.552574168E−01	6.088160591E−02	1.532117147E−02	0.000000000E+00	0.000000000E+00	1.495911835E−07
exactUnif	1.552574168E−01	6.088160593E−02	1.532117150E−02	2.150201066E−28	1.968857862E−13	1.495911800E−07
Krylov(30)	1.552574169E−01	6.088160594E−02	1.532117152E−02	2.154612807E−28	1.968857878E−13	1.495911803E−07

A.2. MAPK2: $n = 484,770$; $nz = 5,820,720$; $||.||_\infty = 95.10$.

	$\varepsilon_{\text{tol}} = 10^{-5}$		$\varepsilon_{\text{tol}} = 10^{-6}$		$\varepsilon_{\text{tol}} = 10^{-7}$		$\varepsilon_{\text{tol}} = 10^{-8}$		$\varepsilon_{\text{tol}} = 10^{-9}$		$\varepsilon_{\text{tol}} = 10^{-10}$	
	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time
$t = 1$												
inexactUnif	1	1.2E+00	1	1.3E+00	1	1.4E+00	1	1.4E+00	1	1.4E+00	1	1.6E+00
exactUnif	101	5.5E+00	105	5.7E+00	110	6.2E+00	114	6.2E+00	117	6.4E+00	121	7.0E+00
Krylov(30)	62	8.2E+00	93	1.2E+01	93	1.3E+01	93	1.2E+01	93	1.2E+01	93	1.3E+01
$t = 10$												
inexactUnif	1	1.1E+01	1	1.2E+01	2	1.3E+01	5	1.3E+01	11	1.5E+01	21	1.6E+01
exactUnif	945	5.2E+01	980	5.3E+01	1015	5.7E+01	1050	6.2E+01	1078	6.5E+01	1106	6.4E+01
Krylov(30)	186	2.4E+01	217	2.9E+01	217	2.8E+01	248	3.5E+01	248	3.4E+01	279	4.0E+01
$t = 100$												
inexactUnif	5	4.0E+01	119	1.4E+02	400	1.7E+02	1079	2.7E+02	2340	3.7E+02	4127	4.5E+02
exactUnif	3169	1.7E+02	7153	3.9E+02	10010	5.5E+02	10335	6.1E+02	10660	6.6E+02	10920	6.3E+02
Krylov(30)	682	8.8E+01	775	1.0E+02	899	1.1E+02	992	1.3E+02	1085	1.5E+02	1178	1.7E+02

	$w_1(t)$			$w_n(t)$		
	$t = 1$	$t = 10$	$t = 100$	$t = 1$	$t = 10$	$t = 100$
$\varepsilon_{\text{tol}} = 10^{-7}$						
inexactUnif	4.488055377E−03	6.724395696E−04	5.309184144E−05	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	4.488055750E−03	6.724409029E−04	5.365710324E−05	5.578122275E−49	5.323320342E−23	1.485858527E−13
Krylov(30)	4.488055970E−03	6.724412733E−04	5.365741171E−05	1.195373611E−51	5.293177404E−23	1.485866141E−13
$\varepsilon_{\text{tol}} = 10^{-10}$						
inexactUnif	4.488055970E−03	6.724412698E−04	5.365621479E−05	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	4.488055970E−03	6.724412730E−04	5.365740680E−05	6.017255413E−49	5.323334687E−23	1.485867037E−13
Krylov(30)	4.488055970E−03	6.724412734E−04	5.365740705E−05	2.005569704E−49	5.322076447E−23	1.485867036E−13

A.3. MAPK3: $n = 1,566,390$; $nz = 20,190,096$; $||.||_\infty = 129.70$.

	$\varepsilon_{\text{tol}} = 10^{-5}$		$\varepsilon_{\text{tol}} = 10^{-6}$		$\varepsilon_{\text{tol}} = 10^{-7}$		$\varepsilon_{\text{tol}} = 10^{-8}$		$\varepsilon_{\text{tol}} = 10^{-9}$		$\varepsilon_{\text{tol}} = 10^{-10}$	
	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time
$t = 1$												
inexactUnif	1	5.8E+00	1	5.6E+00	1	6.2E+00	1	5.5E+00	1	5.3E+00	1	5.9E+00
exactUnif	128	2.6E+01	134	2.6E+01	138	2.9E+01	143	2.6E+01	147	2.5E+01	151	2.8E+01
Krylov(30)	93	4.9E+01	93	4.8E+01	93	4.9E+01	93	3.9E+01	93	3.6E+01	93	4.0E+01
$t = 10$												
inexactUnif	1	5.7E+01	1	5.9E+01	1	6.1E+01	3	5.4E+01	6	5.7E+01	13	6.2E+01
exactUnif	1260	2.7E+02	1305	2.6E+02	1359	2.8E+02	1395	2.5E+02	1440	2.6E+02	1476	2.7E+02
Krylov(30)	217	1.2E+02	248	1.3E+02	248	1.3E+02	279	1.2E+02	310	1.3E+02	310	1.3E+02
$t = 100$												
inexactUnif	1	1.5E+02	40	4.9E+02	225	7.0E+02	642	8.1E+02	1486	1.1E+03	2906	1.5E+03
exactUnif	3169	6.2E+02	7951	1.5E+03	13485	2.5E+03	13920	2.5E+03	14268	2.6E+03	14616	2.6E+03
Krylov(30)	837	3.9E+02	930	4.8E+02	1085	4.4E+02	1240	5.0E+02	1395	5.9E+02	1488	5.8E+02

	$w_1(t)$			$w_n(t)$		
	$t = 1$	$t = 10$	$t = 100$	$t = 1$	$t = 10$	$t = 100$
$\varepsilon_{\text{tol}} = 10^{-7}$						
inexactUnif	5.603577847E−04	5.356397548E−05	3.056917426E−06	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	5.603579078E−04	5.356416637E−05	3.115903889E−06	2.870698209E−55	3.641162019E−26	1.912229657E−15
Krylov(30)	5.603579480E−04	5.356419530E−05	3.115922178E−06	3.717505224E−56	3.582836703E−26	1.912244107E−15
$\varepsilon_{\text{tol}} = 10^{-10}$						
inexactUnif	5.603579480E−04	5.356419257E−05	3.115704160E−06	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	5.603579480E−04	5.356419305E−05	3.115922477E−06	3.096836990E−55	3.641169451E−26	1.912241195E−15
Krylov(30)	5.603579480E−04	5.356419311E−05	3.115922502E−06	1.672589706E−55	3.640413645E−26	1.912241197E−15

Appendix B. MUTEX results. Parameters as for Appendix A.

B.1. MUTEX1: $n = 2,517$; $nz = 20,949$; $||.||_\infty = 154.44$.

	$\varepsilon_{\text{tol}} = 10^{-5}$		$\varepsilon_{\text{tol}} = 10^{-6}$		$\varepsilon_{\text{tol}} = 10^{-7}$		$\varepsilon_{\text{tol}} = 10^{-8}$		$\varepsilon_{\text{tol}} = 10^{-9}$		$\varepsilon_{\text{tol}} = 10^{-10}$	
	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time
$t = 1$												
inexactUnif	9	5.9E−03	23	1.1E−02	50	1.4E−02	83	1.9E−02	107	2.1E−02	114	2.2E−02
exactUnif	98	1.7E−02	103	1.8E−02	107	1.9E−02	111	1.9E−02	115	2.0E−02	118	2.1E−02
Krylov(30)	93	2.2E−02	93	2.1E−02	93	2.2E−02	93	2.2E−02	93	2.1E−02	93	2.1E−02
$t = 10$												
inexactUnif	99	5.8E−02	245	9.1E−02	508	1.3E−01	463	9.9E−02	733	1.4E−01	927	1.8E−01
exactUnif	263	4.6E−02	412	7.0E−02	569	9.8E−02	585	1.0E−01	756	1.3E−01	931	1.6E−01
Krylov(30)	156	3.7E−02	156	3.6E−02	156	3.8E−02	186	4.3E−02	186	4.4E−02	187	4.4E−02
$t = 100$												
inexactUnif	994	5.8E−01	2438	8.9E−01	5004	1.3E+00	504	1.1E−01	632	1.2E−01	837	1.6E−01
exactUnif	287	5.1E−02	448	7.7E−02	463	7.9E−02	637	1.1E−01	653	1.1E−01	841	1.4E−01
Krylov(30)	156	3.7E−02	156	3.6E−02	156	3.6E−02	187	4.4E−02	187	4.4E−02	187	4.4E−02

	$w_1(t)$			$w_n(t)$		
	$t = 1$	$t = 10$	$t = 100$	$t = 1$	$t = 10$	$t = 100$
$\varepsilon_{\text{tol}} = 10^{-7}$						
inexactUnif	5.908162476E−01	5.759677948E−01	5.759646421E−01	1.025981203E−08	8.623696382E−09	2.452064837E−08
exactUnif	5.908914429E−01	5.760429646E−01	5.760438606E−01	1.813281322E−10	1.765543744E−10	1.765546611E−10
Krylov(30)	5.908914877E−01	5.760430075E−01	5.760423408E−01	1.813281459E−10	1.765543864E−10	1.765541820E−10
$\varepsilon_{\text{tol}} = 10^{-10}$						
inexactUnif	5.908914876E−01	5.760430259E−01	5.760430262E−01	1.813281459E−10	1.765543919E−10	1.765543920E−10
exactUnif	5.908914876E−01	5.760430259E−01	5.760430262E−01	1.813281459E−10	1.765543919E−10	1.765543920E−10
Krylov(30)	5.908914877E−01	5.760430251E−01	5.760429693E−01	1.813281459E−10	1.765543917E−10	1.765543746E−10

B.2. MUTEX2: $n = 263,950$; $nz = 4,031,310$; $||.||_\infty = 232.65$.

	$\varepsilon_{\text{tol}} = 10^{-5}$		$\varepsilon_{\text{tol}} = 10^{-6}$		$\varepsilon_{\text{tol}} = 10^{-7}$		$\varepsilon_{\text{tol}} = 10^{-8}$		$\varepsilon_{\text{tol}} = 10^{-9}$		$\varepsilon_{\text{tol}} = 10^{-10}$	
	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time
$t = 1$												
inexactUnif	1	1.1E+00	1	1.3E+00	1	1.5E+00	3	1.5E+00	7	1.8E+00	13	2.4E+00
exactUnif	216	7.1E+00	226	7.9E+00	236	9.1E+00	244	8.5E+00	252	8.8E+00	258	9.5E+00
Krylov(30)	93	4.9E+00	93	4.8E+00	124	7.0E+00	124	6.4E+00	124	6.5E+00	124	6.7E+00
$t = 10$												
inexactUnif	2	1.2E+01	6	1.2E+01	14	1.3E+01	33	1.5E+01	71	1.8E+01	138	2.4E+01
exactUnif	549	2.0E+01	716	2.5E+01	1037	3.6E+01	1217	4.3E+01	1414	5.0E+01	1772	6.4E+01
Krylov(30)	187	9.7E+00	218	1.1E+01	218	1.2E+01	218	1.1E+01	249	1.4E+01	249	1.3E+01
$t = 100$												
inexactUnif	19	1.2E+02	57	1.2E+02	146	1.4E+02	337	1.5E+02	711	1.9E+02	1386	2.3E+02
exactUnif	581	2.0E+01	751	2.9E+01	931	3.4E+01	1281	4.5E+01	1486	5.7E+01	1691	6.0E+01
Krylov(30)	187	9.7E+00	218	1.2E+01	218	1.1E+01	218	1.1E+01	249	1.4E+01	249	1.3E+01

	$w_1(t)$			$w_n(t)$		
	$t = 1$	$t = 10$	$t = 100$	$t = 1$	$t = 10$	$t = 100$
$\varepsilon_{\text{tol}} = 10^{-7}$						
inexactUnif	5.841439919E−01	5.683451353E−01	5.669922628E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	5.846449101E−01	5.699475028E−01	5.699496995E−01	8.689328648E−21	8.466573484E−21	8.466606732E−21
Krylov(30)	5.846449817E−01	5.699465427E−01	5.699465011E−01	1.057141134E−19	3.344706750E−18	3.344706506E−18
$\varepsilon_{\text{tol}} = 10^{-10}$						
inexactUnif	5.846430475E−01	5.699412545E−01	5.699381226E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	5.846449816E−01	5.699465444E−01	5.699465457E−01	8.689329709E−21	8.466558908E−21	8.466558928E−21
Krylov(30)	5.846449817E−01	5.699465222E−01	5.699457741E−01	8.705266675E−21	-2.082401649E−20	-2.082398916E−20

B.3. MUTEX3: $n = 1,047,225$; $nz = 21,972,345$; $||.||_\infty = 232.65$.

	$\varepsilon_{\text{tol}} = 10^{-5}$		$\varepsilon_{\text{tol}} = 10^{-6}$		$\varepsilon_{\text{tol}} = 10^{-7}$		$\varepsilon_{\text{tol}} = 10^{-8}$		$\varepsilon_{\text{tol}} = 10^{-9}$		$\varepsilon_{\text{tol}} = 10^{-10}$	
	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time
$t = 1$												
inexactUnif	1	9.3E+00	1	1.1E+01	1	1.0E+01	1	1.1E+01	2	1.1E+01	4	1.2E+01
exactUnif	333	6.4E+01	348	7.4E+01	360	6.9E+01	372	7.3E+01	384	7.5E+01	396	8.0E+01
Krylov(30)	93	3.5E+01	93	3.6E+01	124	4.6E+01	124	4.8E+01	124	4.8E+01	124	5.0E+01
$t = 10$												
inexactUnif	1	8.7E+01	2	9.1E+01	5	9.6E+01	11	1.0E+02	24	1.1E+02	48	1.2E+02
exactUnif	716	1.5E+02	1044	2.1E+02	1387	2.8E+02	1750	3.8E+02	2120	4.2E+02	2506	4.9E+02
Krylov(30)	218	8.5E+01	249	1.0E+02	249	9.8E+01	249	1.1E+02	280	1.1E+02	311	1.2E+02
$t = 100$												
inexactUnif	6	8.9E+02	18	9.4E+02	49	1.0E+03	114	1.1E+03	244	1.2E+03	224	5.5E+02
exactUnif	726	1.6E+02	1051	2.0E+02	1405	2.8E+02	1761	3.5E+02	2146	4.6E+02	2536	4.9E+02
Krylov(30)	218	8.6E+01	249	9.2E+01	249	9.6E+01	249	9.7E+01	280	1.2E+02	311	1.2E+02

	$w_1(t)$			$w_n(t)$		
	$t = 1$	$t = 10$	$t = 100$	$t = 1$	$t = 10$	$t = 100$
$\varepsilon_{\text{tol}} = 10^{-7}$						
inexactUnif	5.839384439E−01	5.676057995E−01	5.655566443E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	5.846448305E−01	5.699513949E−01	5.699507492E−01	3.215221919E−36	3.144217184E−36	3.144213576E−36
Krylov(30)	5.846449817E−01	5.699465374E−01	5.699462861E−01	8.454329836E−24	-1.173634626E−17	-1.173634108E−17
$\varepsilon_{\text{tol}} = 10^{-10}$						
inexactUnif	5.846420671E−01	5.699383602E−01	5.699350419E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	5.846449815E−01	5.699465481E−01	5.699465470E−01	3.215222751E−36	3.144190105E−36	3.144190099E−36
Krylov(30)	5.846449817E−01	5.699465393E−01	5.699463596E−01	1.026507866E−20	2.463669522E−21	2.463668745E−21

Appendix C. NCD results. Parameters as for Appendix A.

C.1. NCD1: $n = 5,456$; $nz = 35,216$; $||.||_\infty = 23.26$.

	$\varepsilon_{\text{tol}} = 10^{-5}$		$\varepsilon_{\text{tol}} = 10^{-6}$		$\varepsilon_{\text{tol}} = 10^{-7}$		$\varepsilon_{\text{tol}} = 10^{-8}$		$\varepsilon_{\text{tol}} = 10^{-9}$		$\varepsilon_{\text{tol}} = 10^{-10}$	
	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time
$t = 1$												
inexactUnif	1	2.0E−03	1	2.9E−03	1	2.9E−03	1	2.9E−03	1	2.9E−03	1	2.9E−03
exactUnif	29	8.8E−03	31	9.8E−03	33	1.1E−02	35	1.2E−02	37	1.3E−02	39	1.3E−02
Krylov(30)	62	2.9E−02	62	2.7E−02	62	2.7E−02	62	2.6E−02	62	2.5E−02	62	2.6E−02
$t = 10$												
inexactUnif	1	1.4E−02	1	1.5E−02	1	1.5E−02	1	1.5E−02	1	1.5E−02	1	1.8E−02
exactUnif	186	6.2E−02	196	6.4E−02	204	6.7E−02	212	7.0E−02	218	7.2E−02	226	7.4E−02
Krylov(30)	62	2.6E−02	62	2.6E−02	62	2.7E−02	62	2.6E−02	62	2.7E−02	62	2.6E−02
$t = 100$												
inexactUnif	3	1.3E−01	5	1.3E−01	8	1.4E−01	11	1.4E−01	15	1.5E−01	19	1.5E−01
exactUnif	1704	5.6E−01	1764	5.9E−01	1824	6.0E−01	1884	6.2E−01	1944	6.4E−01	1992	6.5E−01
Krylov(30)	124	5.4E−02	124	5.3E−02	124	5.4E−02	155	6.6E−02	155	6.7E−02	186	8.0E−02

	$w_1(t)$			$w_n(t)$		
	$t = 1$	$t = 10$	$t = 100$	$t = 1$	$t = 10$	$t = 100$
$\varepsilon_{\text{tol}} = 10^{-7}$						
inexactUnif	9.970072970E−01	9.706652267E−01	7.506102592E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	9.970072973E−01	9.706652300E−01	7.506102657E−01	0.000000000E+00	1.48178429E−146	9.62797897E−105
Krylov(30)	9.970073730E−01	9.706653663E−01	7.506110649E−01	1.27652477E−197	7.52708063E−167	4.377266630E−83
$\varepsilon_{\text{tol}} = 10^{-10}$						
inexactUnif	9.970073730E−01	9.706653662E−01	7.506110643E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	9.970073730E−01	9.706653662E−01	7.506110643E−01	0.000000000E+00	1.49205224E−146	9.62805426E−105
Krylov(30)	9.970073730E−01	9.706653663E−01	7.506110649E−01	1.38760040E−199	8.47997046E−170	3.164209892E−64

C.2. NCD2: $n = 12,341$; $nz = 81,221$; $||.||_\infty = 35.51$.

	$\varepsilon_{\text{tol}} = 10^{-5}$		$\varepsilon_{\text{tol}} = 10^{-6}$		$\varepsilon_{\text{tol}} = 10^{-7}$		$\varepsilon_{\text{tol}} = 10^{-8}$		$\varepsilon_{\text{tol}} = 10^{-9}$		$\varepsilon_{\text{tol}} = 10^{-10}$	
	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time
$t = 1$												
inexactUnif	1	6.8E−03	1	7.8E−03	1	7.8E−03	1	8.8E−03	1	8.8E−03	1	9.8E−03
exactUnif	38	2.8E−02	41	3.1E−02	44	3.4E−02	46	3.5E−02	48	3.7E−02	50	3.8E−02
Krylov(30)	62	6.7E−02	62	6.3E−02	62	6.3E−02	62	6.4E−02	62	6.3E−02	62	6.3E−02
$t = 10$												
inexactUnif	1	4.8E−02	1	4.9E−02	1	5.2E−02	1	5.3E−02	1	5.5E−02	1	5.7E−02
exactUnif	264	2.0E−01	274	2.1E−01	284	2.2E−01	292	2.2E−01	302	2.3E−01	310	2.4E−01
Krylov(30)	62	6.3E−02	62	6.3E−02	62	6.4E−02	62	6.4E−02	62	6.4E−02	62	6.4E−02
$t = 100$												
inexactUnif	2	4.7E−01	4	4.9E−01	6	5.0E−01	8	5.2E−01	11	5.4E−01	15	5.6E−01
exactUnif	2592	2.0E+00	2682	2.0E+00	2772	2.1E+00	2862	2.2E+00	2952	2.2E+00	3024	2.3E+00
Krylov(30)	124	1.3E−01	124	1.3E−01	124	1.3E−01	155	1.6E−01	186	1.9E−01	248	2.6E−01

	$w_1(t)$			$w_n(t)$		
	$t = 1$	$t = 10$	$t = 100$	$t = 1$	$t = 10$	$t = 100$
$\varepsilon_{\text{tol}} = 10^{-7}$						
inexactUnif	9.960117786E−01	9.610794944E−01	6.821540892E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	9.960117791E−01	9.610794990E−01	6.821540968E−01	0.000000000E+00	1.73272207E−201	5.34734533E−147
Krylov(30)	9.960118222E−01	9.610796409E−01	6.821552696E−01	0.000000000E+00	0.000000000E+00	1.78287456E−142
$\varepsilon_{\text{tol}} = 10^{-10}$						
inexactUnif	9.960118221E−01	9.610796408E−01	6.821552686E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	9.960118221E−01	9.610796408E−01	6.821552686E−01	0.000000000E+00	1.75235449E−201	5.34739831E−147
Krylov(30)	9.960118222E−01	9.610796409E−01	6.821552696E−01	0.000000000E+00	0.000000000E+00	3.603659264E−94

C.3. NCD3: $n = 91,881$; $nz = 623,241$; $||.||_\infty = 99.39$.

	$\varepsilon_{\text{tol}} = 10^{-5}$		$\varepsilon_{\text{tol}} = 10^{-6}$		$\varepsilon_{\text{tol}} = 10^{-7}$		$\varepsilon_{\text{tol}} = 10^{-8}$		$\varepsilon_{\text{tol}} = 10^{-9}$		$\varepsilon_{\text{tol}} = 10^{-10}$	
	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time	uMATVEC	Time
$t = 1$												
inexactUnif	1	1.2E−01	1	1.2E−01	1	1.3E−01	1	1.3E−01	1	1.4E−01	1	1.4E−01
exactUnif	82	5.1E−01	87	5.4E−01	90	5.6E−01	94	5.8E−01	97	6.0E−01	101	6.3E−01
Krylov(30)	62	6.8E−01	62	6.7E−01	62	6.7E−01	62	6.7E−01	62	6.8E−01	62	6.8E−01
$t = 10$												
inexactUnif	1	1.0E+00	1	1.1E+00	1	1.1E+00	1	1.1E+00	1	1.2E+00	1	1.2E+00
exactUnif	725	4.5E+00	750	4.7E+00	775	4.8E+00	800	5.0E+00	825	5.1E+00	845	5.3E+00
Krylov(30)	62	6.7E−01	62	6.8E−01	62	6.7E−01	62	6.7E−01	62	6.7E−01	62	6.7E−01
$t = 100$												
inexactUnif	1	1.0E+01	2	1.1E+01	3	1.1E+01	4	1.1E+01	6	1.2E+01	8	1.2E+01
exactUnif	7250	4.5E+01	7500	4.7E+01	7750	4.8E+01	8000	5.0E+01	8250	5.1E+01	8450	5.3E+01
Krylov(30)	155	1.7E+00	155	1.7E+00	217	2.4E+00	279	3.0E+00	341	3.7E+00	434	4.7E+00

	$w_1(t)$			$w_n(t)$		
	$t = 1$	$t = 10$	$t = 100$	$t = 1$	$t = 10$	$t = 100$
$\varepsilon_{\text{tol}} = 10^{-7}$						
inexactUnif	9.920394538E−01	9.236736066E−01	4.653047977E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	9.920394553E−01	9.236736193E−01	4.653048069E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
Krylov(30)	9.920395500E−01	9.236740522E−01	4.653069901E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
$\varepsilon_{\text{tol}} = 10^{-10}$						
inexactUnif	9.920395499E−01	9.236740518E−01	4.653069870E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
exactUnif	9.920395499E−01	9.236740518E−01	4.653069870E−01	0.000000000E+00	0.000000000E+00	0.000000000E+00
Krylov(30)	9.920395500E−01	9.236740522E−01	4.653069888E−01	0.000000000E+00	0.000000000E+00	3.17331408E−220

Acknowledgments. The comments of the referees contributed to improving the paper and are gratefully acknowledged.

REFERENCES

[1] R. Barrett, M. W. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1993.

[2] A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, New York, 1979.

[3] A. Bouras and V. Frayssé, *Inexact matrix-vector products in Krylov methods for solving linear systems: A relaxation strategy*, SIAM J. Matrix Anal. Appl., 26 (2005), pp. 660–678.

[4] K. Burrage, M. Hegland, S. MacNamara, and R. B. Sidje, *A Krylov-based finite state projection algorithm for solving the chemical master equation arising in the discrete modelling of biological systems*, in Proceedings of the 150$^{\text{th}}$ Markov Anniversary Meeting, A. N. Langville and W. J. Stewart, eds., Boson Books, Raleigh, NC, 2006, pp. 21–38.

[5] J. R. Gilbert, C. Moler, and R. Schreiber, *Sparse matrices in MATLAB: Design and implementation*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 333–356.

[6] W. Grassmann, *Transient solutions in Markovian queueing systems*, Comput. Oper. Res., 4 (1977), pp. 47–56.

[7] D. Gross and D. R. Miller, *The randomization technique as modelling tool and solution procedure for transient Markov processes*, Oper. Res., 32 (1984), pp. 343–361.

[8] M. Hochbruck and Ch. Lubich, *On Krylov subspace approximations to the matrix exponential operator*, SIAM J. Numer. Anal., 34 (1997), pp. 1911–1925.

[9] C.-Y. F. Huang and J. E. Ferrell, Jr., *Ultrasensitivity in the mitogen-activated protein kinase cascade*, Proc. Natl. Acad. Sci. USA, 93 (1996), pp. 10078–10083.

[10] S. MacNamara, R. B. Sidje, and K. Burrage, *An improved dynamic finite state projection algorithm for the numerical solution of the chemical master equation with applications*, in Proceedings of the 13th Biennial Computational Techniques and Applications Conference (CTAC-2006), W. Read and A. J. Roberts, eds., Anziam J., 48 (2007), pp. C397–C419.

[11] C. Moler and C. Van Loan, *Nineteen dubious ways to compute the exponential of a matrix, Twenty-five years later*, SIAM Rev., 45 (2003), pp. 3–49.

[12] B. Philippe, Y. Saad, and W. J. Stewart, *Numerical methods in Markov chains modeling*, Oper. Res., 40 (1992), pp. 1156–1179.

[13] B. Philippe and R. B. Sidje, *Transient solutions of Markov processes by Krylov subspaces*, in Computations with Markov Chains: 2nd International Workshop on the Numerical Solution of Markov Chains, W. J. Stewart, ed., Kluwer International Publishers, Boston, MA, 1995, pp. 95–119.

[14] R. B. Sidje, *Expokit: A software package for computing matrix exponentials*, ACM Trans. Math. Software, 24 (1998), pp. 130–156; available online at http://www.expokit.org.

[15] R. B. Sidje and W. J. Stewart, *A numerical study of large sparse matrix exponentials arising in Markov chains*, Comput. Statist. Data Anal., 29 (1999), pp. 345–368.

[16] V. Simoncini and D. B. Szyld, *Theory of inexact Krylov subspaces methods and applications to scientific computing*, SIAM J. Sci. Comput., 25 (2003), pp. 454–477.

[17] W. J. Stewart, *Marca: Markov Chain Analyser. A Software Package for Markov Modelling*, Technical report, Department of Computer Science, North Carolina State University, Raleigh, NC, 1988; also IEEE Computer Repository report R76 232, 1976, and IRISA Publication Interne 45, Université de Rennes, France.

[18] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, Princeton, NJ, 1994.

[19] J. van den Eshof and G. L. G. Sleijpen, *Inexact Krylov subspace methods for linear systems*, SIAM J. Matrix Anal. Appl., 26 (2004), pp. 125–153.