Inexact Sparse Matrix-Vector Products in the Calculation of Passage Time Distributions in Large Semi-Markov Models

Dingle, Nicholas J.

2013

MIMS EPrint: 2013.8

Manchester Institute for Mathematical Sciences
School of Mathematics
The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097
Inexact Sparse Matrix–Vector Products in the Calculation of Passage Time Distributions in Large Semi-Markov Models

Nicholas J. Dingle*

Abstract

We have previously presented an iterative algorithm based on repeated sparse matrix–vector multiplication for the calculation of passage time distributions in large semi-Markov models. We showed that the required number of operations can be reduced without affecting the accuracy of the final result if we do not perform multiplications with vector elements that are small in magnitude. Our earlier evaluation was limited, however, to a small number of test cases and no general error bound was derived. This paper addresses our prior work’s limitations. We present an error analysis of inexact matrix–vector products in our iterative algorithm that leads to a bound on the overall error compared with the exactly-computed solution. We support this analysis with numerical results from a range of semi-Markov models that demonstrate that the bound is valid in practice and that reducing the number of multiplications leads to a reduction in run-time of over 50% in the best case.

1 Introduction

We have previously presented [2] an iterative algorithm for the calculation of passage time distributions in large semi-Markov models, the core of which is repeated sparse matrix–vector multiplication. In prior work [7] we investigated reducing the number of operations by not performing multiplications with vector elements that were small in magnitude; this approach is

*This work was supported by Engineering and Physical Sciences Research Council grant EP/I006702/1 “Novel Asynchronous Algorithms and Software for Large Sparse Systems” Author’s address: School of Mathematics, The University of Manchester, Manchester, M13 9PL, UK; email: nicholas.dingle@manchester.ac.uk.
referred to as *inexact or approximate* matrix–vector products in the literature [1, 13, 6, 10, 11, 12, 14]. Our earlier evaluation was limited, however, to experiments involving a small number of test cases. Although we showed that run-time was reduced and accuracy was not affected in the examples considered, we did not derive a general error bound.

Sidje *et. al.* [10, 11] derive error bounds for inexact matrix–vector multiplications, but the result applies only to inexact uniformisation for the calculation of transient probability distributions in continuous time Markov chains (CTMCs). In this paper we show that their analysis can also be applied to inexact calculation of passage time distributions in semi-Markov chains. In doing so we overcome our prior work’s limitations. We also provide numerical results that confirm that our bound is valid in practice and that demonstrate reductions in computational effort and run-times.

We denote computed quantities by placing a tilde above the quantity’s name; e.g. \( \tilde{v} \) is the vector that results from an inexact matrix–vector multiplication.

## 2 Iterative Passage Time Algorithm of Semi-Markov Models

In this section we summarise our iterative algorithm [2] that is used to calculate first passage time densities and distributions in semi-Markov processes. Our notation is simplified from the original representation, in order to focus on the key aspects.\(^1\)

Semi-Markov processes (SMPs) are stochastic processes where the choice of the next state depends only on the current state and state holding times have general distributions. The state transition behaviour of an SMP can therefore be characterised by two matrices \( P \) and \( H(t) \), where \( p_{ij} \) is the state transition probability between states \( i \) and \( j \) and \( h_{ij}(t) \) is the sojourn time distribution in state \( i \) when the next state is \( j \).

The iterative algorithm computes the distribution of time taken to transit from a set of source states to a set of target states by calculating and then numerically inverting its Laplace transform. This process requires the calculation of a number (typically several hundred) of samples of the Laplace transform at a range of values of a complex parameter \( s \), which we denote

\(^1\)In particular we use \( \ell(s) \) here in place of \( L^{(s)}_{ij}(s) \).
by \( \ell(s) \). This is accomplished by an iterative calculation of the form:

\[
\ell^T(s) = (a^T U + a^T U^2 + a^T U^3 + \cdots + a^T U^r) = \sum_{k=1}^{r} a^T U^k,
\]

where \( a^T \in \mathbb{R}^n \) is a probability vector (its elements sum to 1), \( U \in \mathbb{C}^{n \times n} \) has elements \( u_{ij} = p_{ij} h^*_ij(s) \), and \( h^*_ij(s) \) is the Laplace transform of \( h_{ij}(t) \).

The value of \( r \) is chosen so that the difference between the two final terms of the sum is less than some suitably small value \( \varepsilon_{tol} \).

### 3 Error Bounds for Inexact Matrix–Vector Multiplication

The central operation in (1) is repeated sparse matrix–vector multiplication; we begin with the vector \( v_0^T = a^T \) and then repeatedly multiply with the matrix \( U \), according to \( v_k^T = v_{k-1}^T U \ (k \geq 1) \). In the analysis that follows we will make use of the following lemma.

**Lemma 3.1.** \( ||U||_{\infty} \leq 1 \), where \( U \in \mathbb{C}^{n \times n} \) is defined above and \( || \cdot ||_{\infty} \) is the matrix \( \infty \)-norm (maximum absolute row sum).

**Proof.** Recall that \( u_{ij} = p_{ij} h^*_ij(s) \). As \( p_{ij} \) is a state transition probability between two states, \( \sum_{k=1}^{n} p_{ik} = 1 \), \( 1 \leq i \leq n \), and by Lemma 1 from [3] \( |h^*_ij(s)| \leq 1 \), \( 1 \leq i, j \leq n \), for all values of \( s \) we encounter. Thus \( \sum_{k=1}^{n} |p_{ik} h^*_ik(s)| \leq 1 \) for all \( i \) and therefore \( ||U||_{\infty} \leq 1 \). \( \square \)

#### 3.1 Bound on the error in the \( k \)th iteration

We want to bound the error introduced into the matrix–vector product \( v_{k-1}^T U \) that arises from ignoring elements of \( v_{k-1}^T \) that are smaller than some threshold value \( \varepsilon \). An exact matrix–vector product can be written as

\[
v_k = v_{k-1} U = \begin{bmatrix} v_1^{(k-1)} & v_2^{(k-1)} & \cdots & v_n^{(k-1)} \end{bmatrix} \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_n^T \end{bmatrix} = \sum_{j=1}^{n} v_j^{(k-1)} u_j^T
\]

where \( v_j^{(k-1)} \) is the \( j \)th element of \( v_{k-1}^T \) and \( u_j^T \) is the \( j \)th row of \( U \).

---

2The original presentation of the algorithm used two matrices \( U \) and \( U' \), but as \( U' \) is \( U \) with one or more rows zeroed out we can frame our analysis here purely in terms of \( U \) without affecting the bounds presented in the next section.
When we compute the inexact matrix–vector product we only multiply rows of \( \mathbf{U} \) with those elements of \( \mathbf{v}_{k-1}^T \) that are larger than \( \varepsilon \); as in [11] we denote the set of these elements as 

\[
\text{Supp}_\varepsilon (\mathbf{v}_{k-1}^T) = \{ 1 \leq i \leq n \mid |v_i^{(k-1)}| > \varepsilon \}
\]

The inexact matrix–vector product can therefore be expressed as

\[
\mathbf{v}_{k-1}^T \mathbf{U} \approx \sum_{j \in \text{Supp}_\varepsilon (\mathbf{v}_{k-1}^T)} v_j^{(k-1)} \mathbf{u}_j^T
\]

with associated error vector

\[
\mathbf{\varepsilon}_k^T = \sum_{j \notin \text{Supp}_\varepsilon (\mathbf{v}_{k-1}^T)} v_j^{(k-1)} \mathbf{u}_j^T
\]

Lemma 3.1 tells us that \( ||\mathbf{u}_j||_1 \leq 1 \), for all \( j \), while the largest value of \( |v_j^{(k-1)}| \) for \( j \notin \text{Supp}_\varepsilon (\mathbf{v}_{k-1}^T) \) will be bounded by \( \varepsilon \) by definition. This allows us to bound the norm of the error vector of each inexact iteration:

\[
||\mathbf{\varepsilon}_k||_1 \leq \sum_{j \notin \text{Supp}_\varepsilon (\mathbf{v}_{k-1}^T)} v_j^{(k-1)} ||\mathbf{u}_j||_1 \leq \sum_{j \notin \text{Supp}_\varepsilon (\mathbf{v}_{k-1}^T)} \varepsilon = \varepsilon n_{k-1} \tag{2}
\]

where \( n_{k-1} = n - |\text{Supp}_\varepsilon (\mathbf{v}_{k-1}^T)| \); that is, \( n_{k-1} \) is the number of elements of \( \mathbf{v}_{k-1}^T \) with modulus less than or equal to \( \varepsilon \).

### 3.2 Bound on the overall error

We now relate the overall error in the inexactly-computed Laplace transform, \( \tilde{\ell}(s) \), to the errors in each iteration. The statement of the theorem and its proof are adaptations of Theorem 3.1 in [11].

**Theorem 3.1.** The overall error in the computed Laplace transform of the passage time density satisfies

\[
||\ell(s) - \tilde{\ell}(s)||_1 \leq \varepsilon \sum_{k=1}^{r} \sum_{j=1}^{k} n_{j-1} \tag{3}
\]

An upper bound on (3) is \( \frac{1}{2}r(r+1)\varepsilon n \). This has the advantage of not requiring us to record \( n_{j-1} \) throughout the computation in order to evaluate the bound, but can potentially be much larger.
Proof. Let $\mathbf{v}_0^T = \mathbf{a}^T$. For $k = 1, \ldots, r$, let $\mathbf{v}_k^T$ be the vector computed at the $k$th iteration by inexact matrix–vector multiplication and let $\mathbf{e}_k^T$ be the associated error vector:

$$ \mathbf{v}_{k-1}^T \mathbf{U} = \mathbf{v}_k^T + \mathbf{e}_k^T, \quad k = 1, \ldots, r. $$

We have

$$ \ell^T(s) = \sum_{k=1}^r \mathbf{v}_k^T = \sum_{k=1}^r (\mathbf{v}_{k-1}^T \mathbf{U} - \mathbf{e}_k^T) = \sum_{k=1}^r \left( \left( \mathbf{v}_0^T \mathbf{U}^k \right) - \sum_{j=1}^k \mathbf{e}_j^T \mathbf{U}^{k-j} \right) $$

and hence, by the definition of $\ell(s)$ in (1):

$$ \ell(s) - \hat{\ell}(s) = \sum_{k=1}^r \sum_{j=1}^k (\mathbf{U}^T)^{k-j} \mathbf{e}_j. $$

Using Lemma 3.1, (2), and the facts that the matrix $\infty$-norm is sub-multiplicative and $||\mathbf{U}^T||_1 = ||\mathbf{U}||_\infty$ [8], we have

$$ ||\ell(s) - \hat{\ell}(s)||_1 \leq \sum_{k=1}^r \sum_{j=1}^k ||\mathbf{e}_j||_1 \leq \sum_{k=1}^r \sum_{j=1}^k \varepsilon n_j \leq \frac{1}{2} r(r + 1) \varepsilon n $$

where $n_j = n - |\text{Supp}_\varepsilon(\mathbf{v}_j^{T-1})|$ and thus $n_j \leq n$ for all $j$. 

4 Numerical Results

In this section we evaluate the tightness of the bound given in (3). We first investigate it in the context of small random matrices, before going on to assessing its validity in calculations arising from large semi-Markov models. In the second set of evaluations we also investigate the number of multiplications performed and the overall solution time.

4.1 Random Matrices

Here, $\mathbf{U}$ is a dense matrix of random complex numbers with $||\mathbf{U}||_\infty \leq 1$ (in accordance with Lemma 3.1), and $\mathbf{v}$ is initialised with $v_i = \mu 10^{-i}$, where $\mu$ is a randomly generated number between 0 and 1. This ensures that there are elements of $\mathbf{v}$ smaller than $\varepsilon$. Table 1 shows $||\ell(s) - \hat{\ell}(s)||_1$ for a range of sizes of $\mathbf{U}$. For each $n$ and $\varepsilon$ we generated 20 different $\mathbf{U}$, and we report the largest $||\ell(s) - \hat{\ell}(s)||_1$ and corresponding $r$ observed in each case. In all cases the observed error is much lower than the bound given in (3).
In this section we base our experiments on the Courier communication protocol model [15] and the Voting and Web-server system models [5]. Table 2 displays the largest values of $||\ell(s) - \tilde{\ell}(s)||_1$ observed for each model across all $\ell(s)$ calculated in each case – recall that $\ell(s)$ is calculated for a range of values of $s$ to enable the Laplace transform to be numerically inverted. We observe that in all cases the observed errors are smaller than the bound given in (3).

It is noticeable in Table 1 that as $n$ increases $r$ decreases and the observed errors remain the same, while in Table 2 $r$ and the errors increase with $n$. This is because $U$ in Section 4.1 is dense and as $n$ grows the magnitudes of the individual elements decrease. In contrast, $U$ in Section 4.2 is sparse and contains approximately the same number of elements, whose values do not depend on $n$, for all $n$.

Our motivation for inexact matrix–vector multiplication was to save
Table 3: Runtime results for $\varepsilon_{tol} = 1e^{-06}, \varepsilon = 1e^{-12}$

<table>
<thead>
<tr>
<th>Name</th>
<th>$n$</th>
<th>Exact</th>
<th></th>
<th></th>
<th>Inexact</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mults</td>
<td>Time</td>
<td>Mults</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Courier</td>
<td>29 010</td>
<td>1 299 510</td>
<td>8.0</td>
<td>255 372</td>
<td>7.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>228 420</td>
<td>13 937 256</td>
<td>103.4</td>
<td>3 243 225</td>
<td>89.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 179 390</td>
<td>74 809 515</td>
<td>577.7</td>
<td>18 820 414</td>
<td>510.8</td>
<td></td>
</tr>
<tr>
<td>Voting</td>
<td>10 300</td>
<td>1 006 266</td>
<td>4.1</td>
<td>63 945</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>106 540</td>
<td>35 330 865</td>
<td>114.0</td>
<td>797 349</td>
<td>58.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 140 050</td>
<td>1 272 051 095</td>
<td>3 439.4</td>
<td>8 895 354</td>
<td>1 623.4</td>
<td></td>
</tr>
<tr>
<td>Web-server</td>
<td>10 995</td>
<td>3 351 719</td>
<td>8.7</td>
<td>650 718</td>
<td>8.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>107 289</td>
<td>72 305 111</td>
<td>242.0</td>
<td>5 508 497</td>
<td>140.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 044 540</td>
<td>1 812 711 509</td>
<td>5 718.2</td>
<td>41 841 224</td>
<td>3 003.1</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Runtime results for $\varepsilon_{tol} = 1e^{-16}, \varepsilon = 1e^{-32}$

<table>
<thead>
<tr>
<th>Name</th>
<th>$n$</th>
<th>Exact</th>
<th></th>
<th></th>
<th>Inexact</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Mults</td>
<td>Time</td>
<td>Mults</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Courier</td>
<td>29 010</td>
<td>2 647 465</td>
<td>11.4</td>
<td>1 316 970</td>
<td>13.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>228 420</td>
<td>29 399 657</td>
<td>170.1</td>
<td>15 368 882</td>
<td>177.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 179 390</td>
<td>185 057 400</td>
<td>1 099.2</td>
<td>101 348 672</td>
<td>1 163.2</td>
<td></td>
</tr>
<tr>
<td>Voting</td>
<td>10 300</td>
<td>2 175 257</td>
<td>6.0</td>
<td>423 811</td>
<td>5.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>106 540</td>
<td>67 301 777</td>
<td>189.2</td>
<td>7 920 521</td>
<td>104.8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 140 050</td>
<td>***</td>
<td>5 773.4</td>
<td>83 350 661</td>
<td>2 807.8</td>
<td></td>
</tr>
<tr>
<td>Web-server</td>
<td>10 995</td>
<td>5 654 019</td>
<td>13.6</td>
<td>2 079 948</td>
<td>16.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>107 289</td>
<td>128 129 802</td>
<td>409.6</td>
<td>33 930 250</td>
<td>295.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 044 540</td>
<td>***</td>
<td>10 391.5</td>
<td>476 132 866</td>
<td>6 469.3</td>
<td></td>
</tr>
</tbody>
</table>

7
time by not carrying out operations involving small vector elements. Tables 3 and 4 present operation-count and run-time figures for the three semi-Markov models for two values of $\varepsilon$. The *** entries in Table 4 denote those cases for which the number of operations was too large to be represented by the C++ unsigned long datatype. We observe that in all cases fewer multiplications were performed in the inexact case than in the exact, and that in most cases this led to a reduction in run-time. Only for small models did the overhead of checking outweigh doing fewer multiplications; this is case, for example, for the Courier model when $\varepsilon = 1e-32$.

![Figure 1: Percentage of vector elements larger that $\varepsilon = 1e-12$](image)

We also observe that the reduction in multiplications and in run-times varies between the different model sizes. Figs. 1 and 2 plot the percentage of vector elements larger than $\varepsilon$ during the computation of $\ell(s)$ for three medium-sized models. It is obvious that the proportion of elements larger than $1e-32$ is greater than those larger than $1e-12$, and this explains why we observe a much larger reduction in the number of operations in the experiments where $\varepsilon = 1e-12$ than when $\varepsilon = 1e-32$. The reduction in operations is greatest for the Voting model, and it is this that has the smallest percentage of non-excluded elements. Likewise, the Courier model had the highest percentage of vector elements not excluded in both cases, and it was this model which demonstrated the smallest saving in operations and run-time.
5 Conclusion

By using an approach that has previously been applied to uniformisation, we derived a bound on the error that occurs when inexact matrix–vector products are used in our iterative passage time algorithm. Our experiments demonstrate that this bound is valid in practice, and that the reduction in multiplications translates into a reduction in run-time of over 50% in the best case.

In future work we will investigate more efficient methods for excluding vector elements to further improve the reductions in run-times. We currently use bitvectors to index which elements of $v$ are larger than $\epsilon$, but we still loop over these bitvectors when deciding which vector elements to use. One possibility might be to use de Bruijn sequences [4] to improve the efficiency of checking for set bits in bitvectors [9]. We would also like to tighten our overall error bound as our results show that it can be pessimistic, but to do so will require more knowledge of the properties of $U$. 

Figure 2: Percentage of vector elements larger that $\epsilon=1e^{-32}$
References


