Web appendix: Numerical evaluation of quasi-stationary distributions

Erik A. van Doorn[a] and Philip K. Pollett[b]

[a] Department of Applied Mathematics, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
E-mail: e.a.vandoorn@utwente.nl

[b] Department of Mathematics, The University of Queensland
Qld 4072, Australia
E-mail: pkp@maths.uq.edu.au

13 January 2013

We consider here the numerical evaluation of quasi-stationary distributions, discussing a range of methods and giving guidance on how to implement these in MATLAB®, perhaps the most widely used package for scientific computing. MATLAB's numerical linear algebra features are built on LAPACK, a Fortran library developed for high-performance computers, and it is worth pointing out that other interfaces exist, including the open source Scilab[1] and Octave[2], which are gaining in popularity. LAPACK itself is in the public domain, and available to aficionados from netlib[3].

Restricting our attention to the case where $S = \{1, 2, \ldots, n\}$ is irreducible, we seek to evaluate the left eigenvector $\boldsymbol{u} = (u_i,\, i \in S)$ of Q corresponding to the eigenvalue, $-\alpha$, with maximal real part (being simple, real and strictly negative). Once normalized so that $\boldsymbol{u}\mathbf{1}^T = 1$, $\boldsymbol{u}$ is the unique quasi-stationary (and then limiting conditional) distribution. If S is reducible, then we would address an eigenvector problem within a restricted set of states (typically, when $-\alpha$ has geometric multiplicity one, we would determine $\boldsymbol{u}$ over states that are accessible from the minimal class $S_{\min}$, and then put $u_j = 0$ whenever j is not accessible from $S_{\min}$); refer to Subsection 3.2 of the paper (General state space).

MATLAB provides two routines for evaluating the eigenvalues and eigenvectors of a square matrix, namely `eig` and `eigs`. The first is, in principle, suitable for any square matrix, its utility limited by the availability of memory and processing power. The second is suitable for large sparse matrices (matrices populated primarily with zeros). We will explain how both routines are used to evaluate quasi-stationary distributions.

According to Cleve Moler[4] there are 16 different "code paths" for the `eig` function. The one trod in our case would be the QR algorithm preceded by a reduction of Q to Hessenberg form (for details see Golub and van Loan [1]), unless, exceptionally, `eig` identified some special structure that it could exploit. The following sequence of commands will usually suffice if Q is not too large:

```
[V,D]=eig(transpose(Q));
[mu,position]=max(real(diag(D)));
u=V(:,position); u=u/sum(u);
alpha=-mu;
```
(1)

The first step includes, importantly, transposing Q (MATLAB evaluates *right* eigenvectors). The result

[1] `http://www.scilab.org/`

[2] `http://www.gnu.org/software/octave/`

[3] `http://www.netlib.org/lapack/`

[4] MATLAB News & Notes, Winter 2000.

is a diagonal matrix D of all eigenvalues of Q and a matrix V whose columns are the corresponding eigenvectors ($Q^TV = Q^TD$, equivalently, $V^TQ = DQ$). The second identifies the eigenvalue with maximum real part (which, for our Q, is real) and records its position. The third step evaluates the quasi-stationary distribution by first extracting the relevant eigenvector and then normalizing it. The final step evaluates the decay parameter as the negative of the dominant eigenvalue. We mention here, and for later reference, that were Q to be *conservative* over S (and hence positive recurrent) the above algorithm would return the unique *stationary* distribution, namely the unique solution to the system ($\boldsymbol{\pi}Q = \mathbf{0}\,;\ \boldsymbol{\pi}\mathbf{1}^T = 1$), and α would be returned as 0 (or very close to 0). However, this is not how one would evaluate a stationary distribution. Rather, since we are solving a system of linear equations, a standard factorize-and-solve method such as Gaussian elimination should be used; in MATLAB, the matrix right-divide command

```
u=[zeros(1,length(Q)) 1]/[Q ones(length(Q),1)];
```

will achieve this. Better still, we might use the GTH algorithm (Grassmann et al. [2], but see also [6]), a version of Gaussian elimination, which is regarded as the gold standard for Markov chains; its superior properties are detailed in O'Cinneide [4].

The above method assumes, of course, that Q is in the MATLAB workspace. But, setting up Q might not be a trivial matter, particularly when the state space is multi-dimensional. In these cases a bijection $f : S \to S'$, where $S' = \{1, 2, \ldots, |S|\}$, is needed to render Q as a square matrix over S'; the rate of transition from $\boldsymbol{x}$ to $\boldsymbol{y}$ in S is assigned to $q_{f(\boldsymbol{x}),f(\boldsymbol{y})}$. If the number x of occupied patches in the metapopulation model described in Example 1 of the paper were constrained by another variable y, being the number of patches *suitable* for occupancy, then the extant states would form a triangular array $S = \{(x, y) : 1 \le x \le y \le n\}$ (see Example 2 of the paper). An appropriate bijection from S to $S' = \{1, 2, \ldots, \frac{1}{2}n(n+1)\}$ would be $f(x, y) := y + \frac{1}{2}(x-1)(2n-x)$. For more complex state spaces, a hash table might provide a more efficient implementation of f, although there would be some setup costs. Perhaps surprisingly, the inverse map is seldom required; typically we would be estimating quantities such as $\Pr(X(t) \in A | X(t) \in S)$ for $A \subset S$ (summing $u_{f(\boldsymbol{x})}$ over $\boldsymbol{x} \in A$), but even when identifying quantities such as the mode of $\boldsymbol{u}$, a simple search may suffice.

Notice that if, for example, $n = 1000$ in our metapopulation model, Q would have $250,500,250,000$ elements, and thus, stored as dense matrix, would require at least $2,000$ gigabytes of main memory. Yet, with only nearest neighbour transitions, typical of this sort of model, only 3 million (0.0003%) of these entries will be non-zero. For such problems, sparse matrix technology should be used. MATLAB provides the full range of sparse matrix operations. The `eigs` command implements Arnoldi's

algorithm, which evaluates (typically a selection of) eigenvalues and eigenvectors of a sparse matrix. The algorithm is iterative. On each iteration, the "basic" Arnoldi method is used. Starting with a "seed vector" $\boldsymbol{v}$, an $m \times m$ upper-Hessenberg matrix H and an $n \times m$ matrix V is constructed in such a way that $V^T Q V = H$, with m fixed to be much smaller than n. The eigenvectors of H are determined by some efficient dense-matrix method and these are used to provide estimates of the extremal eigenvectors of Q. The idea is that if z is an eigenvector of H, then Vz should be close to an eigenvector of Q. Implementations differ in the way $\boldsymbol{v}$ is updated ready for the next iteration. MATLAB's `eigs` implements (through LAPACK) a (random) restart method due to Lehoucq and Sorensen [3]. An alternative restart method, one that is particularly suited to the present problem, is described in Pollett and Stewart [5], but presently not available in MATLAB. For further details, see [1, Chapter 9].

For large problems with sparse transition structure, we must first set up Q as a sparse matrix. The simplest way is to begin with the command `Q=sparse([])`, which initializes Q as an empty sparse matrix (replacing the usual step of setting Q to be the zero matrix: `Q=zeros(n,n)`). Then, we simply enter the non-zero elements as we would normally. (A more complicated method, but one which can markedly reduce execution time, involves setting up vectors of row indices and column indices of non-zero entries and a vector of their values.) Replacing the first step in the earlier procedure by

```
[V,D]=eigs(transpose(Q));
```

will achieve the desired effect, but, as we require the eigenvector corresponding to the eigenvalue with maximum real part, it is significantly more efficient proceed as follows:

```
[u,mu]=eigs(transpose(Q),1,'lr');
u=u/sum(u);                                    (2)
alpha=-mu;
```

(The incantation `[u,mu]=eigs(A,k,'lr')` yields the k eigenvalues of A with largest real part and the corresponding right eigenvectors.) It is quite remarkable that our dense-matrix code can be tweaked so simply.

MATLAB permits us a great deal of control over the way `eigs` is used. For example, the value of the Arnoldi parameter m can be changed from the default $m = 20$. If m is chosen too large or too small, the algorithm will be slow; if too large the time taken to evaluate the eigenvectors of H will be predominant, while if too small the number of outer iterations might be prohibitively large.

Another useful feature of `eigs`, which is facilitated by LAPACK's remarkable "reverse communication" interface, is the ability to pass Q to `eigs` as a function (function handle) that evaluates $Q^T x$:

```
[u,mu]=eigs(@Qfun,1,'lr');
```

where

```
function y = Qfun(x)
  .....
end
```

declared elsewhere in our code, effects the operation $Q^T x$ as an efficient elementwise calculation. So, Q *does not need to be stored at all*, and in principle very much larger problems can be tackled. However, evaluation of $Q^T x$ can be time consuming and, because evaluation is frequent, the resulting code can be very slow.

We mention a final approach to evaluating the quasi-stationary distribution $\boldsymbol{u}$, which exploits the return map $\boldsymbol{m} \mapsto \pi^{\boldsymbol{m}}$ (recall that $\pi^{\boldsymbol{m}}$ is the stationary distribution of the process instantaneously returned to S, on departure, according to the measure $\boldsymbol{m}$, that is, $\pi^{\boldsymbol{m}}(Q + \boldsymbol{a}^T \boldsymbol{m}) = \mathbf{0}$). In the present finite state-space setting, the return map is contractive, and thus iteration leads us to $\boldsymbol{u}$. So, we start with an estimate of $\boldsymbol{u}$, which might for example be suggested by an analytical approximation such as a diffusion approximation, and then iterate until the desired accuracy is achieved, at each step using a Gaussian elimination algorithm to evaluate $\pi^{\boldsymbol{m}}$. If MATLAB's matrix right-divide command is used, sparsity or bandedness in the modified Q will be detected automatically and exploited. However, after at most two iterations the modified Q will have n more non-zero entries than Q.

To illustrate the methods described above consider the simple metapopulation model (Example 1 of the paper), which is a birth-death process with $S = \{1, 2, \ldots, n\}$ and with birth and death rates $\lambda_i = (c/n)i(n - i)$ and $\mu_i = ei$. The quasi-stationary distribution is easily evaluated using code sequence (1) after setting up Q as follows:

```
Q=zeros(n,n);
i=1; lambda=(c/n)*i*(n-i); mu=e*i;
Q(i,i+1)=lambda; Q(i,i)=-(lambda + mu);
for i=2:n-1
   lambda=(c/n)*i*(n-i); mu=e*i;
   Q(i,i+1)=lambda; Q(i,i-1)=mu; Q(i,i)=-(lambda + mu);
```

```
  end
  i=n; mu=e*i; Q(i,i-1)=mu; Q(i,i)=-mu;
```

The result is illustrated in Figures 1 and 2 of the paper.

For the elaboration in which number of patches available for occupancy varies (Example 2 of the paper), we have a two-dimensional state space consisting of a set $S = \{(x, y) : 1 \leq x \leq y \leq n\}$ (irreducible) of transient states and a set $A = \{(0, y) : 0 \leq y \leq n\}$ (irreducible) in which the process is eventually trapped. Recall that the non-zero transition rates are

$$q((x, y); (x, y + 1)) = r(n - y),$$
$$q((x, y); (x, y - 1)) = d(y - x),$$
$$q((x, y); (x - 1, y - 1) = dx,$$
$$q((x, y); (x + 1, y)) = (c/n)x(y - x),$$
$$q((x, y); (x - 1, y)) = ex.$$

Thus, to evaluate the quasi-stationary distribution we would first set up Q rendered as a square matrix over $S' = \{1, 2, \ldots, \frac{1}{2}n(n + 1)\}$ using the bijection $f(x, y) := y + \frac{1}{2}(x - 1)(2n - x)$. For example, for the colonization transition we would have

```
for y=2:n
  for x=1:(y-1)
    i=index([x,y,n]); j=index([x+1,y,n]);
    Q(i,j)=(c/n)*x*(y-x);
  end
end
```

with the bijection defined elsewhere as

```
function i=index(state)
  x=state(1); y=state(2); n=state(3);
  i=x+(y-1)*(2*n-y)/2;
end
```

We could then use code sequence (1), as before. If n is large we would use sparse matrix code, preceding the above with `Q=sparse([])` and then using code sequence (2), or, better, setting up the rows and columns "manually":

```
Q_size=n*(n+1)/2;
Qnz=0;
   .....
for y=2:n
  for x=1:(y-1)
    Qnz=Qnz+1;
    Q_row(Qnz)=index([x,y,n]);
    Q_col(Qnz)=index([x+1,y,n]);
    Q_val(Qnz)=(c/n)*x*(y-x);
  end
end
   .....
Q=sparse(Q_row,Q_col,Q_val,Q_size,Q_size,Qnz);
```

We performed numerical experiments evaluating the quasi-stationary distribution on a PC equipped with an Intel® Xeon® 6-core 3.33 GHz processor, using parameters $e = 0.1$, $c = 0.6$, $d = 0.1$ and $r = 0.5$. Table 1 compares the execution time of `eig` versus `eigs` (with the default of $m = 20$ for the Arnoldi parameter) for the n-patch metapopulation model with different values of n. It is clear that sparse methods are considerably more efficient when the number of states is large. Table 2 compares the time needed to set up Q as a sparse matrix and the time to evaluate the quasi-stationary distribution using `eigs` ($m = 20$) for the n-patch metapopulation model with various values of n. Listed also is the corresponding size of the state space and the number of non-zero elements of Q ($= n(3n-2)$). It is clear that the execution time is dominated by the transition matrix set-up time. Figure 1 displays the average time, each average taken over 10 runs, needed to evaluate the quasi-stationary distribution of the 100-patch metapopulation model using `eigs` for different values of m. Recall that when m is large the time taken to evaluate the eigenvectors of H will be predominant, while if too small the number of outer iterations might be prohibitively large. Notice that MATLAB's default value of $m = 20$ is close to optimal for our problem. The quasi-stationary distribution of the 100-patch metapopulation model evaluated using `eigs` with $m = 20$ is illustrated in Figure 3 of the paper.

n	$\lvert S\rvert$	Execution time	
		`eig`	`eigs`
20	400	0.056	0.024
30	900	0.281	0.042
50	2500	2.618	0.094
100	10000	118.294	0.300
150	22500	1120.634	0.702

Table 1. Time (in seconds) to evaluate the quasi-stationary distribution using `eig` and `eigs` (with $m = 20$) in the n-patch metapopulation model for various values of n. Listed also is the corresponding size of the state space.

n	$\lvert S\rvert$	nnz(Q)	Execution time	
			Q setup	qsd $\boldsymbol{u}$
20	400	1,160	0.012	0.024
30	900	2,640	0.033	0.042
50	2500	7,400	0.157	0.094
100	10,000	29,800	1.977	0.300
150	22,500	67,200	16.013	0.702
200	40,000	119,600	59.810	1.715
300	90,000	269,400	340.285	5.720
500	250,000	749,000	2687.983	8.584

Table 2. Time (in seconds) to set up Q as a sparse matrix and the time to evaluate the quasi-stationary distribution using `eigs` (with $m = 20$) in the n-patch metapopulation model for various values of n. Listed also is the corresponding size of the state space and the number of non-zero elements of Q.

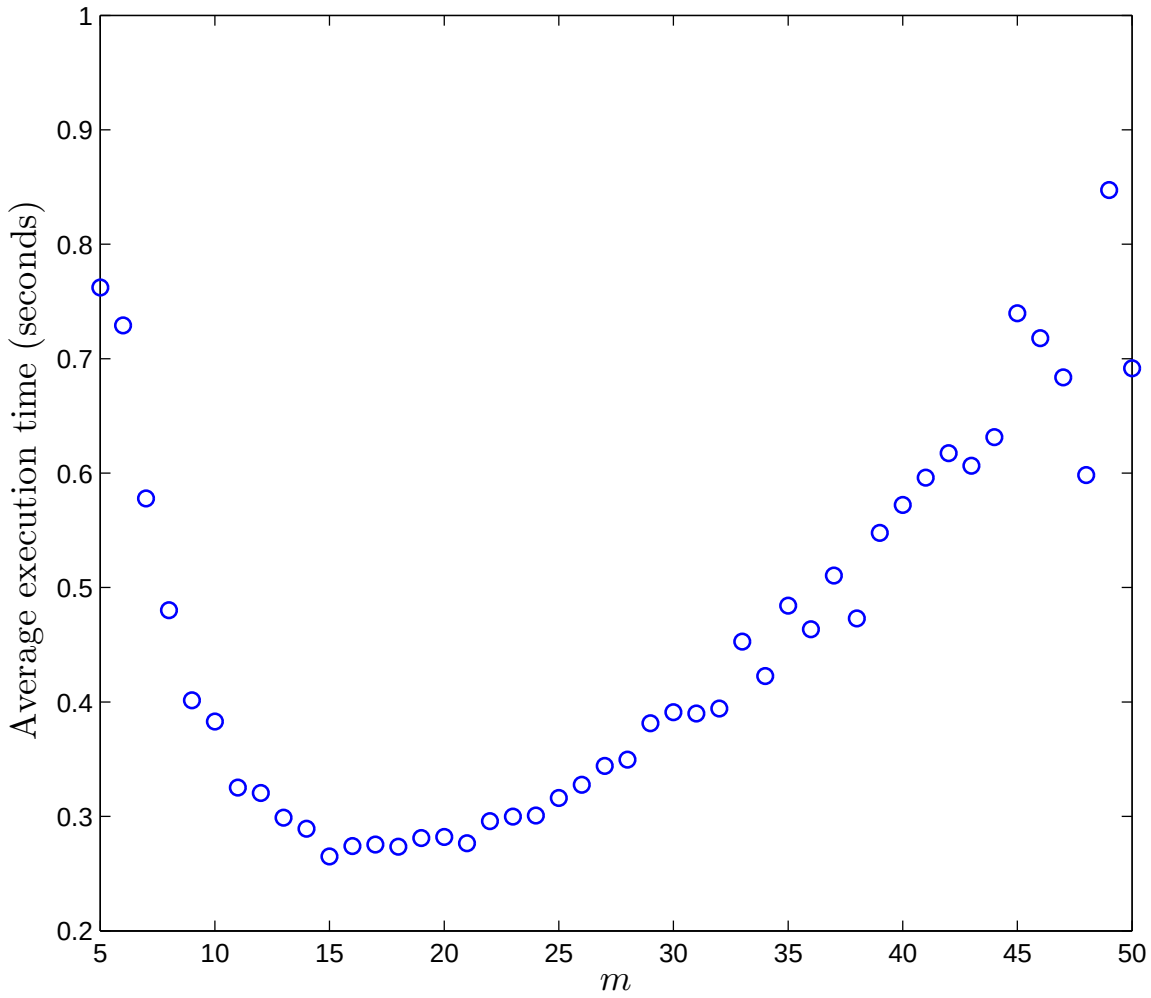

Fig 1. Execution times (each averaged over 10 runs) for evaluating the quasi-stationary distribution of the 100-patch metapopulation model using `eigs` for values of m.

References

[1] Golub, H.G. and van Loan, C.F. (1996). *Matrix Computations*. 3rd ed. Baltimore: Johns Hopkins University Press.

[2] Grassmann, W.K. Taksar, M.I. and Heyman, D.P. (1985). Regenerative analysis and steady state distributions for Markov chains. *Oper. Res.* **33**, 1107-1116.

[3] Lehoucq, R.B. and Sorensen, D.C. (1996). Deflation techniques for an implicitly restarted Arnoldi iteration *SIAM. J. Matrix Anal. & Appl.* **17**, 789-821.

[4] O'Cinneide, C.A. (1993). Entrywise perturbation theory and error analysis for Markov chains. *Numer. Math.* **65**, 109-120.

[5] Pollett, P.K. and Stewart, D.E. (1994). An efficient procedure for computing quasistationary distributions of Markov chains with sparse transition structure. *Adv. Appl. Probab.* **26**, 68-79.

[6] Seneta, E. (1998). Complementation in stochastic matrices and the GTH algorithm. *SIAM J. Matrix Anal. Appl.* **19**, 556-563.