Annales UMCS Informatica AI 2 (2004) 15-23

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

The inverse iteration with the WZ factorization used to the Markovian models

Beata Bylina*

Department of Computer Science, Maria Curie-Skłodowska University,
pl. M.Curie-Skłodowskiej 1, 20-031 Lublin, Poland

Abstract

In the article a new approach for solving Markovian linear systems is presented. It is the inverse iteration with the use of the WZ factorization. The paper contains the theoretical base of the method and numerical experiment results where the accuracy and the performance time were investigated. A modified (with the use of the SBLAS library and the Harwell-Boeing sparse matrix storage scheme) method was also presented. The experiments were carried out with the matrices generated for a Markovian model of a leaky bucket with tokens.

1. Introduction

Behaviour of a real system can often be described by specifying all the states in which a given system can be and by showing the ways the system changes its states in time. If future states of the system depend only on the present state (not on any past states) the behaviour of the system can be presented as a *Markov process* (a kind of a *stochastic process*). If a Markov process state space is discrete, the process is called a *Markov chain*.

A system modelled with a Markov process assumes exactly one state (from the whole set) in each time moment. The system evolution is described with the probabilities of transitions from one state to another. In such a model we are usually interested in probabilities of particular states in a given time moment t.

In this paper we are interested in *discrete-time Markov chains* (DTMC). For such a Markov chain the system state is probed in a discrete set T of time moments – the set T is usually identified with the set of natural numbers. Moreover, we assume our DTMC is *homogeneous*, that is the probabilities of transitions between states are independent of time.

* *E-mail address*: beatas@golem.umcs.lublin.pl

If this Markov chain is ergodic (i.e. irreductible and aperiodic), the probability vector $\pi(t)$ converges (as t grows) to a stationary one π, independent of the initial vector $\pi(0)$, so we have:

$$\pi = \pi \mathbf{P}\,, \tag{1}$$

where **P** is a matrix of transition probabilities between states.

To find π we can solve (1) with the constraints:

$$\pi \geq 0\,, \quad \pi e = 1\,,$$

where $\mathbf{e} = (1,1,...,)^T$.

The equation (1) can be transformed to $\pi \mathbf{P} - \pi = 0$ and farther to $\pi(\mathbf{P} - \mathbf{I}) = 0$. Now we can denote $\mathbf{P} - \mathbf{I}$ by **Q** (**Q** is called a *transition rate matrix* or an *infinitesimal generator matrix*) and we get a linear equation system with the constraints:

$$\pi \mathbf{Q} = 0\,, \quad \pi \geq 0\,, \quad \pi \mathbf{e} = 1\,. \tag{2}$$

The matrix **Q** is a square $n \times n$ matrix (usually a huge one – but we consider only finite state spaces), of the rank (n - 1) (in the cases interesting for us), with the dominant diagonal. It is singular and sparse, so we need appropriate approaches to the system (2).

There are indirect methods, – like iterative and projection methods – that are widely used. However, we sometimes need a direct method (when we need the most accurate results, for example). The most popular of the direct methods are the LU factorization method and its relatives. In this article we want to present another direct method for solving (2), namely the WZ factorization method.

Two approaches for solving (2) with the WZ factorization were presented in [1]. In those experiments, we applied a not-packed storage scheme for transition rate matrices, as for a case of dense matrices. The use of the BLAS libraries made a significant improvement in performance of those algorithms [2,3].

In this article there is presented an approach based on the WZ factorization (sections 2 and 3) and the inverse iteration (section 4). The Harwell-Boeing storage scheme and the BSLAS (Sparse BLAS) library (section 5) is used in experiments, numerical results are displayed (section 6).

2. The WZ factorization

In this section we present shortly the WZ factorization and the method for solving a linear system

$$\mathbf{Ax} = \mathbf{b} \quad \text{were} \quad \mathbf{A} \in R^{n \times n}\,, \quad \mathbf{B} \in R^n$$

with the WZ factorization.

The WZ factorization is described in [4-6]. $\mathbf{A} = \mathbf{WZ}$, where matrices $\mathbf{W}$ and $\mathbf{Z}$ are of the following forms (for the even size n of the matrix $\mathbf{A}$):

$$\mathbf{W} = \begin{bmatrix} 1 & & & & & & & & & 0 \\ w_{21} & 1 & & & & & & & 0 & w_{2n} \\ \cdots & \cdots & \cdots & & & & & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & & & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & 1 & 0 & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & 0 & 1 & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & & & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & & & & & \cdots & \cdots & \cdots \\ w_{n-1,1} & 0 & & & & & & & 1 & w_{n-1,n} \\ 0 & & & & & & & & & 1 \end{bmatrix}, \tag{3}$$

$$\mathbf{Z} = \begin{bmatrix} z_{11} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & z_{1,n} \\ & z_{22} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & z_{2,n} & \\ & & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & & \\ & & & \cdots & \cdots & \cdots & \cdots & & & \\ & & & & z_{pp} & z_{pq} & & & & \\ & & & & z_{qp} & z_{qq} & & & & \\ & & & \cdots & \cdots & \cdots & \cdots & & & \\ & & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & & \\ & z_{n-1,2} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & z_{n-1,n} & \\ z_{n1} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & z_{nn} \end{bmatrix}, \tag{4}$$

where

$$m = \lfloor (n-1)/2 \rfloor, \quad p = \lfloor (n+1)/2 \rfloor, \quad q = \lceil (n+1)/2 \rceil.$$

After factorization we can solve two linear systems:

$$\begin{aligned} \mathbf{Wc} &= \mathbf{b}, \\ \mathbf{Zx} &= \mathbf{c}, \end{aligned}, \tag{5}$$

(where $\mathbf{c}$ is an auxiliary vector) instead of one (2).

3. Sequential algorithm for WZ factorization

The WZ method algorithm for solving linear systems (after [4,6]) consists of two parts: reduction of the matrix $\mathbf{A}$ (and the vector $\mathbf{b}$) to the matrix $\mathbf{Z}$ (and the vector $\mathbf{c}$) and next solving equation (5).

The first part consists in partial zeroing of columns of the matrix $\mathbf{A}$. In the first step, we zero the elements from the 2nd to n-1st in the 1st and nth columns.

Next, we update the matrix **A** and the vector **b**. After $m + 1$ such steps we get the matrix $\mathbf{Z} = \mathbf{A}^{(m+1)}$, (as defined in (4)) and the vector $\mathbf{c} = \mathbf{b}^{(m+1)}$. We get

$$\mathbf{W}^{(\mathbf{m})} \cdot \ldots \cdot \mathbf{W}^{(0)} \cdot \mathbf{A} = \mathbf{Z}$$

so

$$\mathbf{A} = \left\{\mathbf{W}^{(m)}\right\}^{-1} \cdot \ldots \cdot \left\{\mathbf{W}^{(0)}\right\}^{-1} \cdot \mathbf{Z} = \mathbf{W}\mathbf{Z} .$$

The second part of the method is to solve a linear system (5). This part consists in solving a linear system with two unknown quantities x_p and x_q and next updating the vector **b**.

Formally we can write this with the use of a MATLAB-like notation:

```
% part I: the elimination loop -- the reduction
%         from the matrix A to the matrix Z
for k=0:m
   k2=n-k+1;
   det=A(k,k)*A(k2,k2)-A(k2,k)*A(k,k2);
% computing elements of the matrix W
   for i=k+1:k2-1
      wk1=(A(k2,k)*A(i,k2)-A(k2,k2)*A(i,k))/det;
      wk2=(A(k,k2)*A(i,k)-A(k,k)*A(i,k2))/det;
% updating the matrix A
      for j=k+1:k2-1
         A(i,j)=A(i,j)+wk1*A(k,j)+wk2*A(k2,j);
% updating the vector b
      b(i)=b(i)+wk1*b(k)+wk2*b(k2);

% part II: computing the solution vector x
for j=m:0
% solving an auxiliary 2x2 system
   j2=n-j+1;
   det=A(j,j)*A(j2,j2)-A(j2,j)*A(j,j2);
   x(j)=(b(j)*A(j2,j2)-b(j2)*A(j,j2))/det;
   x(j2)=(b(j2)*A(j,j)-b(j)*A(j2,j))/det;
% updating the vector b
   for i=j-1:0
      i2=n-i-1
      b(i)=b(i)-x(j)*A(i,j)-x(j2)*A(i,j2);
      b(i2)=b(i2)-x(j)*A(i2,j)-x(j2)*A(i2,j2);
```

4. The inverse iteration

As we noted above, we are interested in finding the vector π from (2). In our case we need to solve a homogenous linear system with n equations and n unknowns. Such a system has a non-zero solution if and only if the coefficient matrix is singular. The matrix **Q** has a zero eigenvalue [7], so it is singular. Let us assign $\pi = \mathbf{x}^T$ and transpose (2) to get:

$$\mathbf{Q}^T\mathbf{x}=0\,,\quad \mathbf{x}\geq 0\,,\quad \mathbf{e}^T\mathbf{x}=1\,.$$

We want to present an approach to solve such a linear system. We assume that the Markov chain represented by **Q** is irreductible (so **Q** is of rank (*n*-1)).

The inverse iteration was introduced in [8]. It is a method for finding eigenfunctions of linear operators. It is used here to find an eigenvector **x** of a matrix **A**:

$$\mathbf{A}\mathbf{x}=\lambda\mathbf{x}$$

where **A** is a real or complex square matrix and λ is an eigenvalue. The inverse iteration is a method to compute an eigenvector associated with a known eigenvalue. Given an approximation μ of the eigenvalue λ and a starting vector $\mathbf{x}_0$, the inverse iteration generates a sequence of vectors $(\mathbf{x}_k)$ being solutions of the systems:

$$\left(\mathbf{A}-\mu\mathbf{I}\right)\mathbf{x}_k=s_k\mathbf{x}_{k-1}\quad \text{for}\quad k\geq 1$$

where s_k is responsible for normalization of $\mathbf{x}_k$ – usually it is chosen to make $\|\mathbf{x}_k\|=1$ with respect to a suitable norm.

The sequence $(\mathbf{x}_k)$ usually converges to the eigenvector associated with the eigenvalue λ, especialy when the λ is the eiganvalue closest to μ and when the vector $\mathbf{x}_0$ is a linear combination of the eigenvectors (what is almost sure).

Let $\mathbf{A}=\mathbf{Q}^T$. Now $\lambda=0$ is an eigenvalue of **A** so we can write (with s_k = 1):

$$\left(\mathbf{Q}^T-0\mathbf{I}\right)\mathbf{x}_k=\mathbf{x}_{k-1}$$

and then:

$$\mathbf{Q}^T\mathbf{x}_k=\mathbf{x}_{k-1}\,.$$

For k = 1 we get:

$$\mathbf{Q}^T\mathbf{x}_1=\mathbf{x}_0\,.$$

As a starting vector we can choose $\mathbf{x}_0=\mathbf{e}_p$ and we are to solve a system $\mathbf{Q}^T\mathbf{x}_1=\mathbf{e}_p$ what we can do by the WZ factorization. Let $\mathbf{Q}^T=\mathbf{WZ}$, so we get $\mathbf{WZx}_1=\mathbf{e}_p$ and then we are to solve

$$\begin{cases}\mathbf{Wy}=\mathbf{e}_p\,,\\ \mathbf{Zx}_1=\mathbf{y}.\end{cases}$$

It is obvious (see (3)) that the solution of the equation $\mathbf{Wy}=\mathbf{e}_p$ is $\mathbf{y}=\mathbf{e}_p$, so we are only to solve

$$\mathbf{Zx}_1=\mathbf{e}_p\,. \tag{6}$$

First, let us consider an even n. The equation (6) takes the form:

$$\begin{bmatrix} z_{11} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & z_{1,n} \\ & z_{22} & \cdots & \cdots & \cdots & \cdots & z_{2,n} & \\ & & \cdots & \cdots & \cdots & \cdots & & \\ & & & z_{pp} & z_{pq} & & & \\ & & & z_{qp} & z_{qq} & & & \\ & & \cdots & \cdots & \cdots & \cdots & & \\ & z_{n-1,2} & \cdots & \cdots & \cdots & \cdots & z_{n-1,n} & \\ z_{n1} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & z_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \\ x_q \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}. \tag{7}$$

The determinant $z_{pp}z_{qq} - z_{pq}z_{qp} = 0$ (because $\mathbf{Q}^T$ was singular and diagonal dominant), so the linear system is contradictory. We can try to solve a system very similar to (7): with $z_{pp}z_{qq} - z_{pq}z_{qp}$ equal to the minimal positive real number on the given machine. Such an approach gives very accurate results which was explained in [9] and which is shown in the numerical experiments described in this paper.

For an odd n we have:

$$\begin{bmatrix} z_{11} & \cdots & \cdots & \cdots & \cdots & \cdots & z_{1,n} \\ & z_{22} & \cdots & \cdots & \cdots & z_{2,n} & \\ & & \cdots & \cdots & \cdots & & \\ & & & z_{pp} & & & \\ & & \cdots & \cdots & \cdots & & \\ & z_{n-1,2} & \cdots & \cdots & \cdots & z_{n-1,n} & \\ z_{n1} & \cdots & \cdots & \cdots & \cdots & \cdots & z_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

and $z_{pp} = 0$. Here we solve the system with z_{pp} replaced with the minimal positive real number.

For both cases we compute only $\mathbf{x}_1$ because next approximations do not give better results [7,9]. In the end we normalize the solution to satisfy the equation $\mathbf{e}^T\mathbf{x} = 1$.

5. Harwell-Boeing storage scheme and SBLAS library

In the Harwell-Boeing storage scheme [10] a sparse matrix is stored in three one-dimensional arrays. The first one consists of values of non-zero matrix elements by rows, the second one consists of column indices of those elements and the third one consists of starting points of matrix rows.

SBLAS (*Sparse BLAS*) is a library written in Fortran 77 as a BLAS (*Basic Linear Algebra Subprograms*) library improvement for sparse vectors and

matrices. In this library a sparse vector is represented by a triple (NZ, X, INDX), where

- NZ – an integer, the number of non-zero elements;
- X – a one-dimentional array of values of non-zero elements;
- INDX – a one-dimentional array of integer indices of non-zero elements.

For some BLAS operations there is no need for a new sparse variant, for example:

- _NRM2 – the Euklid norm $\|\mathbf{x}\|_2$;
- _ASUM – the 1-norm $\|\mathbf{x}\|_1$;
- _SCAL – the vector scaling $\alpha\mathbf{x}$.

Some operations have to be treated differently, for example:

- _DOT+ – the dot product of a sparse vector and a dense vector;
- _AXPYI – updating a dense vector **y** with a sparse vector $\mathbf{x}\left(\mathbf{y} \leftarrow \mathbf{y} + \alpha\mathbf{x}\right)$.

In procedure/function names above, the symbol _+ is used for one of the letters S, D, C, Z which denote vector elements type (single precision real, double precision real, single precision complex double precision complex, respectively).

In the our algorithm the SBLAS library was used in the WZ factorization. When the vector **b** is updated, four auxiliary sparse vectors are created which consist non-zero elements of the matrix **A**, namely A(i,j), A(i,j2), A(i2,j), A(i2,j2). Next the vector **b** is updated with the DAXPYI subroutine. Such a modified WZ factorization algorithm will be called SWZ.

6. The numerical experiment and its results

The algorithms described above were implemented with the use of the language C for computations with double precision real numbers (however, the input data were given in single precision). For the SBLAS version the SBLAS routines were imported. For the minimal positive real number, we used FLT_MIN from <float.h>. The programs were compiled with the gcc compiler with the compiler option -O3 on. Algorithms were tested for the matrices generated for a leaky bucket model with tokens [11] (with various parameters). The model described in [11] uses a DTMC.

The performance times of the algorithms are presented in Table 1 and the accuracy – in Table 2.

Table 1. The performance times (in seconds) of the inverse iteration algorithm with the WZ factorization with the use of the gcc compiler

	Pentium IV 2.8 GHz		Pentium III 733 MHz	
Matrix size	WZ	SWZ	WZ	SWZ
1166	123	119	788	477
2101	1211	1160	4824	4782
2850	3437	3233	19415	15153
4049	10462	10135	54213	50843
5225	24992	24170	118082	117592

Table 2. The Euclid norm of the results of the described algorithms with the use of the gcc compiler

	Pentium IV 2.8 GHz		Pentium III 733 MHz	
Matrix size	WZ	SWZ	WZ	SWZ
1166	2.84305e-08	2.84305e-08	2.84305e-08	2.84305e-08
2101	3.43647e-08	3.43647e-08	3.43647e-08	3.43647e-08
2850	4.77358e-08	4.77358e-08	4.77358e-08	4.77358e-08
4049	4.66344e-08	4.66344e-08	4.66344e-08	4.66344e-08
5225	5.48455e-08	5.48455e-08	5.48455e-08	5.48455e-08

7. Conclusion

The approach presented above is the alternative to the traditional LU factorization and to the iteration and projection methods. We showed that the inverse iteration with the WZ factorization is applicable to the Markovian modelling. The numerical experiment showed that the use of the SBLAS library speeds up the performance without changing the accuracy (which goes down with the rise of the matrix size).

References

[1] Bylina B., Bylina J., *Solving Markov chains with the WZ factorization for modelling networks*, Proceedings of 3rd International Conference Aplimat 2004, Bratislava, (2004) 307.

[2] Bylina B., *Solving linear systems with vectorized WZ factorization*, Annales UMCS Informatica, Lublin, 1 (2003) 5.

[3] Bylina B., Bylina J., *Rozwiazywanie układów równań metoda rozkładu WZ z wykorzystaniem bibliotek BLAS1 i BLAS2*, Studia Informatica, Gliwice, to appear, in Polish.

[4] Chandra Sekhara Rao S., *Existence and uniqueness of WZ factorization*, Parallel Computing, 23 (1997) 1129.

[5] Evans D.J., Hatzopoulos M., *The parallel solution of linear system*, Int. J. Comp. Math., 7 (1979) 227.

[6] Yalamov P., Evans D.J., *The WZ matrix factorization method*, Parallel Computing, 21 (1995) 1111.

[7] Stewart W., *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, Chichester, West Sussex, (1994).

[8] Wielandt H., *Beitraege zur mathematischen Behandlung komplexer Eigenwertprobleme, Teil V: Bestimmung hoeher Eigenwerte durch gebrochene Iteration*, Bericht B 44/J/37, Aerodynamische Versuchsanstalt Goettingen, Germany, (1944), in German.
[9] Wilkinson J.H., *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, England, (1965).
[10] Stabrowski M., *Kompresja niesymetrycznych macierzy rzadkich metodami iloczynu i sumy macierzy transponowanej*, Informatyka Stosowana S2/2002, VI Lubelskie Akademickie Forum Informatyczne, Kazimierz Dolny, (2002), 289, in Polish.
[11] Domańska J., Czachórski T., *Wpływ samopodobnej natury ruchu na zachowanie mechanizmu cieknącego wiadra*, Studia Informatica, 2A(53) (2003) 199, in Polish.