

## A review of numerical methods for solving large Markov chains

BEATA BYLINA<sup>a</sup>

JAROSŁAW BYLINA<sup>b</sup>

<sup>a</sup>Department of Computer Science, Marie Curie-Skłodowska University, Pl. M. Curie-Skłodowskiej 1,  
20-031 Lublin, Poland, e-mail: beatas@golem.umcs.lublin.pl

<sup>b</sup>Department of Computer Science, Marie Curie-Skłodowska University, Pl. M. Curie-Skłodowskiej 1,  
20-031 Lublin, Poland, e-mail: jmbylina@hektor.umcs.lublin.pl

### 1. Introduction

In this paper we would like to make a review of chosen numerical algorithms used for numerical solving of Markov chains.

We are interested in stationary solutions of homogeneous, irreducible Markov chain. Such a chain can be described with an infinitesimal generator matrix  $\mathbf{Q}$  defined for continuous-time Markov chains (CTMCs) as following:

$$\mathbf{Q} = (q_{ij})_{1 \leq i \leq n, 1 \leq j \leq n},$$
$$q_{ij} = \lim_{\Delta t \rightarrow 0} \frac{p_{ij}(\Delta t)}{\Delta t} \quad \text{for } i \neq j,$$
$$q_{ii} = - \sum_{j \neq i} q_{ij},$$

where  $p_{ij}(\Delta t)$  is the probability that if the chain is in the state  $i$  it will be in the state  $j$  after the time  $\Delta t$ .

For a discrete-time Markov chain (DTMC) we can define the matrix  $\mathbf{Q}$  as  $\mathbf{Q} = \mathbf{P} - \mathbf{I}$ , where  $\mathbf{P}$  is a stochastic matrix of the DTMC and it is defined:

$$\mathbf{P} = (p_{ij})_{1 \leq i \leq n, 1 \leq j \leq n},$$

where  $p_{ij}$  is the probability that if the chain is in the state  $i$  it will be in the state  $j$  in the next time moment.

Stationary solutions can be obtained (both for CTMCs and for DTMCs) from a linear system:

$$\boldsymbol{\pi} \mathbf{Q} = \mathbf{0},$$

where  $\boldsymbol{\pi} = (\pi_1, \dots, \pi_n)$  is a vector of probabilities of particular states of the Markov chain (so  $\boldsymbol{\pi} \geq \mathbf{0}$  and  $\sum_{i=1}^n \pi_i = 1$ ) which are to be found.

The same probability vector can be obtained as an eigenvector of the stochastic matrix  $\mathbf{P}$  from:

$$\boldsymbol{\pi} \mathbf{P} = \boldsymbol{\pi}$$

For CTMCs we can define  $\mathbf{P}$  as:

$$\mathbf{P} = \mathbf{I} + \alpha \mathbf{Q}, \quad (1)$$

where  $0 < \alpha < 1/(\max_{i=1, \dots, n} |q_{ii}|)$ .

For a convenient notation we assume  $\boldsymbol{\pi} = \mathbf{x}^T$  and our problem is to solve:

$$\mathbf{Q}^T \mathbf{x} = \mathbf{0} \quad (2)$$

(or as an eigenvector problem:  $\mathbf{P}^T \mathbf{x} = \mathbf{x}$ ) with the constraints:

$$\mathbf{x} \geq \mathbf{0}, \quad \mathbf{e}^T \mathbf{x} = 1. \quad (3)$$

Despite its familiar form the equation is rather special. The matrix  $\mathbf{Q}$  is singular so the equation (2) has solutions and it can be proven [13] that – if  $\text{rank} \mathbf{Q} = n - 1$  (it is true in interesting for us cases) – there exists exactly one solution satisfying (3). Moreover, the matrix  $\mathbf{Q}$  is huge (sometimes millions of states or even more), very sparse and ill-conditioned. We have to choose a suitable algorithm for solving our problem (depending on our aims to achieve – accuracy, time or size).

There are following approaches to solve the equation (2) [13]:

- direct methods (variations of the Gaussian elimination – see section 2);
- iterative methods (section 3);
- projection methods (section 4);
- decompositional methods (not covered in this paper – they are the material for the separate article on their own).

## 2. Direct Methods

Methods which would give us an exact solution in a finite number of steps – if the machine accuracy were infinite – are called *direct methods* (or sometimes: *exact methods*).

The traits of the direct methods are:

- constant execution time (or rather a constant number of computation steps for a given matrix size) known in advance;
- modification (or rather complete reconstruction) of the given matrix;
- the *fill-in*;
- rather good accuracy.

The fill-in is a very troublesome phenomenon. It consists in appearing nonzero elements in the output matrices in places of zero elements in the input matrix. It is very unconvient if we want to store matrices in a compact manner (i.e. without zero entries) – what is very efficient and indeed necessary for such huge and sparse matrices. In compact storage schemes we must implement some routines to insert new nonzero elements – or provide some space for such entries. However, amount of this space must be estimated in advance what is not a trivial problem. Sometimes we simply have not the space needed for the fill-in.

### 2.1. The Gaussian Elimination and the LU Factorization

In the Gaussian elimination the equation

$$\mathbf{Ax} = \mathbf{b}$$

is transformed into:

$$\mathbf{Ux} = \mathbf{c}$$

where the matrix  $\mathbf{U} = (u_{ij})$  is an upper triangular matrix. From such an equation we can easily obtain the vector  $\mathbf{x}$  with the back-substitution:

$$x_n = c_n/u_{nn}; \quad x_i = (c_i - \sum_{k=i+1}^n u_{ik}x_k)/u_{ii} \quad \text{for } i = n-1, \dots, 2, 1.$$

The Gaussian elimination transforming the matrix  $\mathbf{U}$  into the matrix  $\mathbf{A}$  (and the vector  $\mathbf{b}$  into the vector  $\mathbf{c}$ ) consists in  $(n-1)$  steps. In the step number  $i$  all the elements from the column  $i$  being below the row  $i$  are replaced with zeroes by adding the row  $i$  scaled to every row below the row  $i$  (with analogous changes in the vector  $\mathbf{b}$ ).

The scaling factors are the elements of a lower triangular matrix  $\mathbf{L}$  (which have a unity diagonal) and the LU factorization is given by the equality  $\mathbf{A} = \mathbf{L}\mathbf{U}$  (so  $\mathbf{U} = \mathbf{L}^{-1}\mathbf{A}$  and  $\mathbf{c} = \mathbf{L}^{-1}\mathbf{b}$ ).

However, our equation (2) is a homogeneous equation with a singular matrix. To solve such an equation with the LU factorization we present four approaches after [13].

**Replacing an equation.** It is the most intuitive approach. Instead of solving (2) we can solve an equation:

$$\tilde{\mathbf{Q}}_n^T \mathbf{x} = \mathbf{e}_n$$

where  $\mathbf{e}_n = (\underbrace{0, \dots, 0}_{n-1}, 1)^T$  and  $\tilde{\mathbf{Q}}_n$  is the matrix  $\mathbf{Q}$  with the column  $n$  replaced with the vector  $\mathbf{e} = (\underbrace{1, \dots, 1}_n)^T$ . In other words we solve the linear system (2) with the last equations replaced with normalization equation

$$\mathbf{e}^T \mathbf{x} = 1. \quad (4)$$

This approach is the least accurate from the presented in this section. Moreover, it is the slowest because the matrix  $\tilde{\mathbf{Q}}_n^T$  is not diagonally dominant any longer and for its LU factorization the pivoting is necessary.

**The zero pivot.** In this approach (and in the next two approaches) we apply the LU factorization directly to the matrix  $\mathbf{Q}$  (or to its submatrix, as in “Removing an equation”) – so there is no need for the pivoting (the matrix is diagonally dominant).

The matrix is both diagonally dominant and singular therefore in the last step of the Gaussian elimination we get a zero as a diagonal entry ( $u_{nn} = 0$ ). So in the back-substitution we start with the equation  $0 \cdot x_n = 0$  where  $x_n$  can be an arbitrary real number  $\eta$ . The remaining equations are solved normally and eventually we get  $x_i = \xi_i \eta$  with  $\xi_n = 1$ . The last step is getting rid of  $\eta$  – it is done from the normalization equation (4).

**Removing an equation.** In this approach the matrix  $\mathbf{Q}^T$  is divided in blocks:

$$\mathbf{Q}^T = \begin{pmatrix} \mathbf{B} & \mathbf{d} \\ \mathbf{c}^T & f \end{pmatrix}.$$

Here the matrix  $\mathbf{B}$  is a nonsingular square matrix of the size  $(n-1) \times (n-1)$ ,  $\mathbf{c}$  and  $\mathbf{d}$  are vectors of the size  $(n-1)$  and  $f$  is a real number. We also assume  $\mathbf{x}^T = (\hat{\mathbf{x}}^T, 1)$ . Now, our equation (2) has the form:

$$\begin{cases} \mathbf{B}\hat{\mathbf{x}} + \mathbf{d} = \mathbf{0}, \\ \mathbf{c}^T \hat{\mathbf{x}} + f = 0. \end{cases}$$

Only the first equation,  $\mathbf{B}\hat{\mathbf{x}} = -\mathbf{d}$ , is solved by the LU factorization (the second one is ignored, ‘removed’) and after that the vector  $\mathbf{x}^T = (\hat{\mathbf{x}}^T, 1)$  is normalized.

**The inverse iteration.** We solve the equation

$$\mathbf{Q}^T \mathbf{x}^{(1)} = \mathbf{x}^{(0)},$$

which arises when we assume  $\mu = 0$ ,  $\mathbf{A} = \mathbf{Q}^T$  and  $k = 1$  in an iterative formula serving for finding an eigenvector  $x$  associated with an approximated eigenvalue  $\mu$ :

$$\mathbf{x}^{(k)} = (\mathbf{A} - \mu \mathbf{I})^{(-1)} \mathbf{x}^{(k-1)}.$$

As a starting vector we choose  $\mathbf{x}^{(0)} = \mathbf{e}_n$ . The consecutive steps are performed analogically as they are in “The zero pivot”.

## 2.2. The GTH Algorithm

For solving the equation (2) the GTH algorithm (given by Grassmann, Taskar and Heyman in 1985 [8]) is recommended. It is an LU factorization variant that appears to be more stable because of exploitation of the matrix  $\mathbf{Q}$  property:

$$q_{ii} < 0, \quad q_{ij} \geq 0, \quad \sum_{j=1}^n q_{ij} = 0.$$

Its disadvantage is more complicated requirements for accessing the matrix items and therefore more complicated storage scheme and more space for storing the matrix.

We present the GTH version of the Gaussian elimination for the matrix  $\mathbf{Q}^T$  on the figure 1.

## 3. Iterative Methods

All the iterative methods have the similar scheme. They start with a starting vector  $\mathbf{x}_0$  and then they generate a sequence  $(\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \dots)$  which – hopefully – converges to the solution vector  $\mathbf{x}$ . A very general scheme of an iterative method applied to the equation  $\mathbf{A}\mathbf{x} = \mathbf{b}$  is shown on the figure 2.

The advantages of the iterative methods:

- they need no modification of the given matrix (so no fill-in is generated and we do not need any additional space for new elements and we spend no additional time on inserting these elements into a complicated storage structure);
- they need very little additional memory;

```

for  $i = 1, 2, \dots, n - 1$ :
  1.  $s \leftarrow 0.0$ 
  2. for  $j = i + 1, i + 2, \dots, n$ :
    (a)  $r \leftarrow q_{ji}/q_{ii}$ 
    (b) for  $k = i + 1, i + 2, \dots, n$ :
      i.  $q_{jk} \leftarrow q_{jk} + r q_{ik}$ 
    (c) if  $j > i + 1$  then  $s \leftarrow s + q_{j,i+1}$ 
  3.  $q_{i+1,i+1} \leftarrow s$ 

```

Fig. 1: The GTH version of the Gaussian elimination for the transposed infinitesimal generator matrix

```

1. choose  $\mathbf{x}^{(0)}$ 
2.  $i \leftarrow 0$ 
3. while  $\mathbf{Q}^T \mathbf{x}^{(i)}$  is too far from  $\mathbf{0}$  do:
  (a)  $\mathbf{x}^{(i+1)} \leftarrow \mathbf{F}(\mathbf{x}^{(i)})$ 
  (b)  $i \leftarrow i + 1$ 

```

Fig. 2. A general scheme of an iterative method

- they are usually faster than direct methods – especially when we do not need very good accuracy offered by the direct methods;
- they are easy to implement efficiently and easy to vectorize and to parallelize.

However, the iterative methods have some disadvantages too. We do not know the time needed to achieve required accuracy. Moreover, sometimes we can have even troubles with convergency and we can achieve a solution not satisfying us – especially when the required accuracy is high (what can be an issue in our applications).

In classical iterative methods (covered in this section) if we are to solve an equation of the form  $\mathbf{Ax} = \mathbf{b}$  the function  $\mathbf{F}$  has a form:

$$\mathbf{F}(\mathbf{v}) = \mathbf{H}\mathbf{v} + \mathbf{c},$$

where the matrix  $\mathbf{H}$  depends on the matrix  $\mathbf{A}$  and the vector  $\mathbf{c}$  depends on the matrix  $\mathbf{A}$  and the vector  $\mathbf{b}$ .

We can obtain some general formulas from such an (informal) deduction: Let  $\mathbf{A} = \mathbf{M} - \mathbf{N}$ , where  $\mathbf{M}$  is not singular. Now we have  $(\mathbf{M} - \mathbf{N})\mathbf{x} = \mathbf{b}$  and after some conversions:

$$\mathbf{x} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x} + \mathbf{M}^{-1}\mathbf{b}.$$

Assigning

$$\mathbf{F}(\mathbf{v}) = \mathbf{M}^{-1}\mathbf{N}\mathbf{v} + \mathbf{M}^{-1}\mathbf{b} \quad (5)$$

we get

$$\mathbf{H} = \mathbf{M}^{-1}\mathbf{N} \quad (6)$$

and

$$\mathbf{c} = \mathbf{M}^{-1}\mathbf{b}$$

(in our applications  $\mathbf{c} = \mathbf{0}$ , because  $\mathbf{b} = \mathbf{0}$ ). For certain matrices  $\mathbf{M}$  and  $\mathbf{N}$  the function  $\mathbf{F}$  from (5) applied to the scheme from the figure 2 gives a convergent method.

### 3.1. The Power Method

The power method is the most simple iterative approach. It exploits the fact that the unknown vector is an eigenvector of the matrix  $\mathbf{P}^T$  associated with the eigenvalue 1 – so it is its a fixed point of a function  $\mathbf{F}(\mathbf{v}) = \mathbf{P}^T\mathbf{v}$ . Such fixed points sometimes can be obtained from iterations of  $\mathbf{F}$  – that is by applying a scheme from the figure 2 – and so it is in our case.

The step 3a from the figure 2 takes the form:

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{P}^T \mathbf{x}^{(k)}$$

for a DTMC or the form:

$$\mathbf{x}^{(k+1)} \leftarrow (\mathbf{I} + \alpha \mathbf{Q}^T) \mathbf{x}^{(k)}$$

for a CTMC (where  $0 < \alpha < 1/(\max_{i=1,\dots,n} |q_{ii}|)$ , see (1)).

The power method is convergent but its convergence is very slow.

### 3.2. The Method of Jacobi

The method of Jacobi is a classical iterative methods with the coefficient matrix  $\mathbf{Q}^T$  split as following:

$$\mathbf{Q}^T = \mathbf{D} - (\mathbf{L} + \mathbf{U})$$

what corresponds to assigning:

$$\mathbf{M} = \mathbf{D}, \quad \mathbf{N} = \mathbf{L} + \mathbf{U}$$

in (6). The matrix  $\mathbf{D} = (d_{ij})$  is a diagonal matrix, the matrix  $\mathbf{L} = (l_{ij})$  is a strictly lower triangular matrix (with zeroes on its diagonal) and the matrix  $\mathbf{U} = (u_{ij})$  is a strictly upper triangular matrix (with zeroes on its diagonal). So in this method the step 3a from the figure 2 looks as following:

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{(k)}$$

and in scalar form:

$$x_i^{(k+1)} \leftarrow \frac{1}{d_{ii}} \left( \sum_{j \neq i} (l_{ij} + u_{ij}) x_j^{(k)} \right), \quad \text{for } i = 1, \dots, n.$$

Of course, there is no need for computing  $\mathbf{D}^{-1}$ , because the matrix  $\mathbf{D}$  is a diagonal one.

The method of Jacobi is much better than the power method and it is very convenient to vectorize and to parallelize [14].

### 3.3. The Method of Gauss-Seidel

In this method we have the same splitting of the matrix  $\mathbf{Q}^T$  but with a different grouping of components:

$$\mathbf{Q}^T = (\mathbf{D} - \mathbf{L}) - \mathbf{U}$$

(the matrices  $\mathbf{D}$ ,  $\mathbf{L}$ ,  $\mathbf{U}$  are defined as in the previous section). Here we have  $\mathbf{M} = (\mathbf{D} - \mathbf{L})$  and  $\mathbf{N} = \mathbf{U}$ , so the step 3a from the figure 2 is:

$$\mathbf{x}^{(k+1)} \leftarrow (\mathbf{D} - \mathbf{L})^{-1} \mathbf{U} \mathbf{x}^{(k)}. \quad (7)$$

In this case there is no need for computing  $(\mathbf{D} - \mathbf{L})^{-1}$  either, because the matrix  $(\mathbf{D} - \mathbf{L})$  is a lower triangular one and a version of the back-substitution (known from the LU factorization) can be performed. The method of Gauss-Seidel can be interpreted as the method of Jacobi in which we use just computed items of the vector  $\mathbf{x}^{(k)}$  for computing next items of the same vector in the same iteration. In scalar form (7) looks as following:

$$x_i^{(k+1)} \leftarrow \frac{1}{d_{ii}} \left( \sum_{j=1}^{i-1} l_{ij} x_j^{(k+1)} + \sum_{j=i+1}^n u_{ij} x_j^{(k)} \right), \quad \text{for } i = 1, \dots, n.$$

and its scalar form:

However, this is the cause for which the method of Gauss-Seidel is less convenient for vectorization and parallelization. On the other hand, the convergence of this method is usually better than the one of the method of Jacobi.

### 3.4. The Successive Overrelaxation (SOR) Method

The SOR method is a modification of the method of Gauss-Seidel. Here we split the matrix  $\mathbf{Q}^T$  as follows:

$$\mathbf{Q}^T = (\mathbf{D} - \omega\mathbf{L})/\omega - ((1 - \omega)\mathbf{D} + \omega\mathbf{U})/\omega,$$

where  $0 < \omega < 2$  [15] (for  $\omega = 1$  we get the method of Gauss-Seidel). The vector form of the iteration step 3a is:

$$\mathbf{x}^{(k+1)} \leftarrow (\mathbf{D} - \omega\mathbf{L})^{-1} ((1 - \omega)\mathbf{D} + \omega\mathbf{U})\mathbf{x}^{(k)}$$

and the scalar form is:

$$x_i^{(k+1)} \leftarrow (1 - \omega)x_i^{(k)} + \frac{\omega}{d_{ii}} \left( \sum_{j=1}^{i-1} l_{ij} x_j^{(k+1)} + \sum_{j=i+1}^n u_{ij} x_j^{(k)} \right), \quad \text{for } i = 1, \dots, n.$$

The parameter  $\omega$  controls how the previous approximation vector will be taken as a part of the new approximation vector. A good value of the parameter  $\omega$  can speed up the convergence very much but its choice is – in general – still an open problem.

### 3.5. The Preconditioned Power Method

The original power method (section 3.1) can be very slow for some matrices  $\mathbf{P}$ . However, its convergence can be enhanced by the preconditioning. The idea behind the

preconditioning is to change the given linear system to get the better convergency but without changing its solutions. The original system  $\mathbf{Ax} = \mathbf{b}$  is replaced by

$$\mathbf{M}^{-1}\mathbf{Ax} = \mathbf{M}^{-1}\mathbf{b},$$

where  $\mathbf{M}^{-1}$  is called a preconditioning matrix. The matrix  $\mathbf{M}$  should be chosen so that the matrix  $\mathbf{M}^{-1}$  is easy to compute (for example by the Gaussian elimination) and approximates  $\mathbf{A}^{-1}$ .

Of course, our equation is  $\mathbf{Q}^T\mathbf{x} = \mathbf{0}$  with the singular coefficient matrix  $\mathbf{Q}^T$ , so the matrix  $(\mathbf{Q}^T)^{-1}$  cannot be approximated, because it does not exist. Instead, we can choose the matrix  $\mathbf{M}^{-1}$  to approximate the matrix  $(\mathbf{Q}^T)^\#$  – the group inverse of the matrix  $\mathbf{Q}^T$ . Now we can solve a new system:

$$\mathbf{M}^{-1}\mathbf{Q}^T\mathbf{x} = \mathbf{0}$$

with the power method (which now has a much better convergency) and our step 3a from the figure 2 is:

$$\mathbf{x}^{(k+1)} \leftarrow (\mathbf{I} + \mathbf{M}^{-1}\mathbf{Q}^T)\mathbf{x}^{(k)}.$$

How to find a suitable matrix  $\mathbf{M}$ ? The most effective methods are incomplete LU factorizations. Such an incomplete LU factorization consists in an usual factorization of  $\alpha\mathbf{Q}^T$  ( $0 < \alpha < 1/(\max_{i=1,\dots,n} |q_{ii}|)$ ) as in (1) but some entries in the output matrices are omitted and replaced by zeroes. In such an approach we get an invertible matrix  $\mathbf{M}$  which differs from  $\mathbf{Q}^T$  by a remainder matrix  $\mathbf{E}$ , being small in some sense:

$$\mathbf{M} = \tilde{\mathbf{L}}\tilde{\mathbf{U}} = \alpha\mathbf{Q}^T + \mathbf{E}$$

and the fill-in is very little (or controlled at least) or even none.

All the methods of incomplete factorizations described below can give good results but they are not sufficiently investigated, especially in the applications to Markov chains.

**ILU(0).** This is the most straightforward manner of the incomplete factorization. It generates no fill-in, because all the nonzero elements arising on the places of zero elements in the input matrix during the factorization are discarded. In other words,  $\tilde{\mathbf{L}} + \tilde{\mathbf{U}}$  has the same zero structure as the matrix  $\alpha\mathbf{Q}^T$ .

**ILUTH.** The incomplete LU factorization with a threshold is a method where the LU factorization is performed in a usual row-by-row manner but only result items with their absolute values greater than (or equal to) the given threshold (and all the diagonal elements) are remembered – values less than the threshold are replaced with zeroes.

**ILUK.** In this type of the incomplete LU factorization we determine the amount of memory available for the output matrices in advance. We do this by defining a number  $K$ . After transforming each row only  $K$  largest elements from this row (and the diagonal element) are remembered.

#### 4. Projection Methods

The projection methods consists in approximating the solution vector with a vector from a small-dimension subspace. Such approximations are repeated until our approximation is sufficiently close to the solution – in some sense the projection methods are iterative methods.

The projection methods need more space than iterative methods (because they have to store huge basis vectors of subspaces), but can converge faster than classical iterative methods – although the convergence rate is much better for the matrices ‘more beautiful’ in their structure than the ones arising in solving Markov chains.

##### 4.1. The Projection Step

To solve a linear system  $\mathbf{Ax} = \mathbf{b}$  by a projection method first we have to choose two subspaces of dimension  $m$  from the  $n$ -dimensional space:

- $\mathcal{K}$  which is a subspace containing the approximation;
- $\mathcal{L}$  which is a subspace defining constraints for selection of approximation from  $\mathcal{K}$ .

Let the subspace  $\mathcal{K}$  be spanned by  $\mathbf{V} = (\mathbf{v}_1, \dots, \mathbf{v}_m)$ . The approximated solution is in  $\mathcal{K}$  so it can be written

$$\mathbf{x} = \mathbf{V}\mathbf{y}$$

where  $\mathbf{y}$  is an  $m$ -dimensional unknown vector. To find  $\mathbf{y}$  we require that the residual vector  $\mathbf{b} - \mathbf{Ax} = \mathbf{b} - \mathbf{AV}\mathbf{y}$  be orthogonal to the subspace  $\mathcal{L}$  spanned by  $\mathbf{W} = (\mathbf{w}_1, \dots, \mathbf{w}_m)$ , that is:

$$\mathbf{W}^T(\mathbf{b} - \mathbf{AV}\mathbf{y}) = 0$$

and then (if the matrix  $\mathbf{W}^T\mathbf{AV}$  is nonsingular):

$$\mathbf{y} = (\mathbf{W}^T\mathbf{AV})^{-1}\mathbf{W}^T\mathbf{b}.$$

If we know an initial approximation  $\mathbf{x}^{(0)}$  we will rather seek a difference  $\mathbf{d}$  between the exact solution  $\mathbf{x}$  and  $\mathbf{x}^{(0)}$ :  $\mathbf{x} = \mathbf{x}^{(0)} + \mathbf{d}$ . Setting  $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{Ax}^{(0)}$  we are to solve the equation

$$\mathbf{Ad} = \mathbf{r}^{(0)}$$

what can be done with the described above projection step.

1. choose:
    - an initial approximation  $\mathbf{x}^{(0)}$
    - a subspace  $\mathcal{K}$  spanned by  $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_m]$
    - a subspace  $\mathcal{L}$  spanned by  $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_m]$
  2.  $\mathbf{r}^{(0)} \leftarrow -\mathbf{Q}^T \mathbf{x}^{(0)}$
  3.  $\mathbf{y} \leftarrow (\mathbf{W}^T \mathbf{A} \mathbf{V})^{-1} \mathbf{W}^T \mathbf{r}^{(0)}$
  4.  $\mathbf{x}^{(0)} \leftarrow \mathbf{x}^{(0)} + \mathbf{V} \mathbf{y}$

Fig. 3. A basic projection step for the equation  $\mathbf{Q}^T \mathbf{x} = \mathbf{0}$

A basic projection step for our equation 2 (where  $\mathbf{A} = \mathbf{Q}^T$  and  $\mathbf{b} = \mathbf{0}$ ) is shown on the figure 3.

The most efficient method for general, non-symmetric coefficient matrices (as  $\mathbf{Q}^T$ ) are methods based on Krylov subspaces. A Krylov subspace is defined by its dimension  $m$ , a matrix  $\mathbf{A}$  and a vector  $\mathbf{v}$ :

$$\mathcal{K}_m(\mathbf{A}, \mathbf{v}) = \text{span}\{\mathbf{v}, \mathbf{A}\mathbf{v}, \mathbf{A}^2\mathbf{v}, \dots, \mathbf{A}^{m-1}\mathbf{v}\}.$$

Many of such methods require that an orthonormal basis be found for the Krylov subspace. Unfortunately, classical Gram-Schmidt orthogonalization is numerically poor. To deal with it there are two main kinds of methods: Arnoldi process (which is a modified Gram-Schmidt orthogonalization) and Lanczos methods (originally for symmetric coefficient matrices but generalized in some ways).

#### 4.2. The Arnoldi Process

The Arnoldi process [1] on its own (see the figure 4) generates the orthonormal basis  $\mathbf{V} = (\mathbf{v}_1, \dots, \mathbf{v}_m)$  for the subspace  $\mathcal{K}_m(\mathbf{A}, \mathbf{v})$  and an upper Hessenberg matrix  $\mathbf{H} =$

1.  $\mathbf{v}_1 \leftarrow \mathbf{v} / \|\mathbf{v}\|_2$
2. for  $j = 1, 2, \dots, m$ :
  - (a)  $\mathbf{w} \leftarrow \mathbf{A} \mathbf{v}_j$
  - (b) for  $i = 1, 2, \dots, j$ :
    - i.  $h_{ij} \leftarrow \mathbf{v}_i^T \mathbf{w}$
    - ii.  $\mathbf{w} \leftarrow \mathbf{w} - h_{ij} \mathbf{v}_i$
  - (c)  $h_{j+1,j} \leftarrow \|\mathbf{w}\|_2$
  - (d)  $\mathbf{v}_{j+1} \leftarrow \mathbf{w} / h_{j+1,j}$

Fig. 4. The basic Arnoldi process for a subspace  $\mathcal{K}_m(\mathbf{A}, \mathbf{v})$

$(h_{ij})$ :

$$\mathbf{H} = \begin{pmatrix} h_{11} & h_{12} & h_{13} & \cdots & h_{1,m-2} & h_{1,m-1} & h_{1m} \\ h_{21} & h_{22} & h_{23} & \cdots & h_{2,m-2} & h_{2,m-1} & h_{2m} \\ 0 & h_{32} & h_{33} & \cdots & h_{3,m-2} & h_{3,m-1} & h_{3m} \\ 0 & 0 & h_{43} & \cdots & h_{4,m-2} & h_{4,m-1} & h_{4m} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & h_{m-1,m-2} & h_{m-1,m-1} & h_{m-1,m} \\ 0 & 0 & 0 & \cdots & 0 & h_{m,m-1} & h_{mm} \end{pmatrix}$$

which represents the linear transformation  $\mathbf{A}$  restricted to  $\mathcal{K}_m(\mathbf{A}, \mathbf{v})$  with respect to the basis  $\mathbf{V}$ , that is  $\mathbf{H} = \mathbf{V}^T \mathbf{A} \mathbf{V}$ .

The original Arnoldi process applied to a linear system  $\mathbf{A} \mathbf{x} = \mathbf{b}$  is called *the full orthogonalization method* (FOM) [10] but a better approach is *the generalized minimum residual* algorithm (GMRES) [11]. Both the methods are shown on the figure 5. They differ only in one step – how to find the vector  $\mathbf{y}$  (but both the procedures are projections [11]).

The GMRES algorithm is very popular in its iterative form. In the iterative GMRES after computing the new vector  $\mathbf{x}^{(0)}$ , the new residual  $-\mathbf{Q}^T \mathbf{x}^{(0)}$  is checked if it is sufficiently close to  $\mathbf{0}$ . If not, the whole algorithm is repeated with the new  $\mathbf{x}^{(0)}$  as an initial guess.

1. choose  $\mathbf{x}^{(0)}$  and  $m$
2.  $\mathbf{r}^{(0)} \leftarrow -\mathbf{Q}^T \mathbf{x}^{(0)}$
3.  $\beta \leftarrow \|\mathbf{r}^{(0)}\|_2$
4.  $\mathbf{v}_1 \leftarrow \mathbf{r}^{(0)}/\beta$
5. for  $j = 1, \dots, m$ :
  - (a)  $\mathbf{w} \leftarrow \mathbf{Q}^T \mathbf{v}_j$
  - (b) for  $i = 1, \dots, j$ :
    - i.  $h_{ij} \leftarrow \mathbf{v}_i^T \mathbf{w}$
    - ii.  $\mathbf{w} \leftarrow \mathbf{w} - h_{ij} \mathbf{v}_i$
  - (c)  $h_{j+1,j} \leftarrow \|\mathbf{w}\|_2$
  - (d)  $\mathbf{v}_{j+1} \leftarrow \mathbf{w}/h_{j+1,j}$
6. FOM only:  
find  $\mathbf{y} = (y_1, \dots, y_m)$  from the  $m \times m$  Hessenberg system  $\mathbf{H}\mathbf{y} = \beta \mathbf{e}_1$
7. GMRES only:  
find  $\mathbf{y} = (y_1, \dots, y_m)$  minimizing  $\|\beta \mathbf{e}_1 - \bar{\mathbf{H}}\mathbf{y}\|_2$   
where  $\bar{\mathbf{H}} = (h_{ij})$  is an  $(m+1) \times m$  upper Hessenberg matrix
8.  $\mathbf{x}^{(0)} \leftarrow \mathbf{x}^{(0)} + \sum_{i=1}^m \mathbf{v}_i y_i$

Fig. 5. The FOM and GMRES methods for the equation  $\mathbf{Q}^T \mathbf{x} = \mathbf{0}$

### 4.3. The Methods Related to Conjugate Gradients

The original symmetric Lanczos algorithm is used for finding approximations of eigenvalues of a symmetric matrix  $\mathbf{A}$ . For such a matrix a tridiagonal, symmetric matrix  $\mathbf{T}$  is generated, eigenvalues of which are approximations of the subset of the eigenvalues of a given matrix. When we perform the Arnoldi process on a symmetric matrix we get  $\mathbf{H} = \mathbf{T}$  – that is Arnoldi's matrix  $\mathbf{H}$  becomes:

$$\mathbf{T} = \begin{pmatrix} \alpha_1 & \beta_2 & & & & & \\ \beta_2 & \alpha_2 & \beta_3 & & & & \\ & \beta_3 & \alpha_3 & \beta_4 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \beta_{m-1} & \alpha_{m-1} & \beta_m & \\ & & & & \beta_m & \alpha_m & \end{pmatrix}.$$

For the  $m = n$  the eigenvalues of  $\mathbf{T}$  are the same as the eigenvalues of  $\mathbf{A}$  (if the arithmetic is exact).

There are some method for solving  $\mathbf{Ax} = \mathbf{b}$  that are connected to the Lanczos method – first the steepest descent method and the conjugate gradient (CG) method (the latter is shown on the figure 6). Unfortunately, these methods are usable to symmetric, positive-definite matrices. However there exist some modifications of the CG method suitable for us.

**CGNR.** The conjugate gradient method for the normal equations consists in solving a normal equations:

$$\mathbf{A}^T \mathbf{Ax} = \mathbf{A}^T \mathbf{b}$$

or:

$$\mathbf{AA}^T \mathbf{z} = \mathbf{b} \quad \text{and then} \quad \mathbf{x} = \mathbf{A}^T \mathbf{z}$$

with the CG method, instead of  $\mathbf{Ax} = \mathbf{b}$ . The matrix  $\mathbf{A}^T \mathbf{A}$  (and also  $\mathbf{AA}^T$ ) is a symmetric, positive-definite matrix, so the CG method can be used. Unfortunately such an approach has more computations to do and has a worse condition.

**BCG and CGS.** The more suitable algorithms are the *biconjugate gradient* algorithm (BCG) [7] and the *conjugate gradient squared* algorithm (CGS) [12]. The latter (shown on the figure 7) converges faster than the former and requires less computation per iteration. Moreover, both the algorithms requires less work and memory per iterations than the GMRES algorithm.

However, the approximations from BCG and CGS do not satisfy an optimal property (as GMRES approximations does), so it implies that the GMRES will do not more

1. choose  $\mathbf{x}^{(0)}$
2.  $\mathbf{r}^{(0)} \leftarrow \mathbf{b} - \mathbf{A}\mathbf{x}^{(0)}$
3.  $\mathbf{v}^{(0)} \leftarrow \mathbf{0}$
4.  $\beta \leftarrow 0$
5. for  $j = 1, 2, \dots$ :
  - (a)  $\mathbf{v}^{(j)} \leftarrow \mathbf{r}^{(j-1)} + \beta\mathbf{v}^{(j-1)}$
  - (b)  $\alpha \leftarrow \frac{(\mathbf{r}^{(j-1)})^T \mathbf{r}^{(j-1)}}{(\mathbf{v}^{(j)})^T \mathbf{A}\mathbf{v}^{(j)}}$
  - (c)  $\mathbf{x}^{(j)} \leftarrow \mathbf{x}^{(j-1)} + \alpha\mathbf{v}^{(j)}$
  - (d)  $\mathbf{r}^{(j)} \leftarrow \mathbf{r}^{(j-1)} - \alpha\mathbf{A}\mathbf{v}^{(j)}$
  - (e)  $\beta \leftarrow \frac{(\mathbf{r}^{(j)})^T \mathbf{r}^{(j)}}{(\mathbf{r}^{(j-1)})^T \mathbf{r}^{(j-1)}}$

Fig. 6. The conjugate gradient algorithm

iterations than the algorithms here described. Moreover, GMRES approximation errors decrease monotonically and in BCG and CGS it does not, so we have no guarantee that the algorithms described in this section give us a solutions at all.

#### 4.4. The Preconditioned Iterative Methods

For an ill-conditioned matrix (as  $\mathbf{Q}^T$  is) projective methods can converge slowly. To make the condition better we can use a preconditioner in the similar manner as it was described for the power method in the section 3.5.

## 5. Conclusion

In this paper we described traditional methods for solving:

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

that can be used to solve the equation:

$$\mathbf{Q}^T \mathbf{x} = \mathbf{b}$$

arising during modelling with Markov chains.

1. choose  $\mathbf{x}^{(0)}$
2.  $\mathbf{r}^{(0)} \leftarrow -\mathbf{Q}^T \mathbf{x}^{(0)}$
3.  $\hat{\mathbf{r}}^{(0)} \leftarrow \mathbf{r}^{(0)}$
4.  $\mathbf{p} \leftarrow \mathbf{0}$
5.  $\mathbf{q} \leftarrow \mathbf{0}$
6.  $\rho_0 \leftarrow 1$
7.  $k \leftarrow 0$
8. do until  $\mathbf{r}^{(k)}$  is sufficiently near to zero:
  - (a)  $k \leftarrow k + 1$
  - (b)  $\rho_k \leftarrow (\hat{\mathbf{r}}^{(0)})^T \mathbf{r}^{(k-1)}$
  - (c)  $\beta \leftarrow \rho_k / \rho_{k-1}$
  - (d)  $\mathbf{u} \leftarrow \mathbf{r}^{(k-1)} + \beta \mathbf{q}$
  - (e)  $\mathbf{p} \leftarrow \mathbf{u} + \beta(\mathbf{q} + \beta \mathbf{p})$
  - (f)  $\mathbf{v} \leftarrow \mathbf{Q}^T \mathbf{p}$
  - (g)  $\alpha \leftarrow \rho_k / (\mathbf{v}^T \hat{\mathbf{r}}^{(0)})$
  - (h)  $\mathbf{q} \leftarrow \mathbf{u} - \alpha \mathbf{v}$
  - (i)  $\mathbf{u} = \mathbf{u} + \mathbf{q}$
  - (j)  $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha \mathbf{u}$
  - (k)  $\mathbf{r}^{(k)} = -\mathbf{Q}^T \mathbf{x}^{(k)}$
9. normalize  $\mathbf{x}^{(k)}$

Fig. 7. The conjugate gradient squared algorithm applied to  $\mathbf{Q}^T \mathbf{x} = \mathbf{0}$

The methods are still developed. Some traditional methods are adapted to Markov chains – like the WZ factorization [4]. The vectorized, paralelized and distributed implementations on various of described here (and many more) methods are worked on [5, 6, 9]. So the applications of supercomputers are investigated [14]. There are much hope in some new methods as decompositional methods and methods based on Kroncker algebra [3]. The combinations of the various traditional methods are investigated.

Selection of a suitable solution method is by no means easy. There are papers on automatic proper choice of the solving method for a given model [2]. The choice depends on many questions:

- the matrix structure and its degree of decomposability (e.g. is it NCD?);
- the matrix closeness to a suitable structure (and possibility to convert it);
- the matrix sparseness;
- the matrix size (and our storage possibilities);
- time to find the solution;
- desired accuracy;
- the matrix conditioning.

### References

- [1] W.E. Arnoldi: The principle of minimized iteration in the solution of the matrix eigenvalue problem, *Quarterly for Applied Mathematics* 9, 1951, pp. 17–29.
- [2] W. Barge, W. Stewart: Autonomous Solution Methods for Large-Scale Markov Chains (to be printed).
- [3] P. Buchholz, M. Fischer, P. Kemper: Distributed Steady State Analysis Using Kronecker Algebra, *Numerical Solution of Markov Chains*, Zargoza, Spain, 1999, pp. 76–95.
- [4] B. Bylina, J. Bylina: Solving Markov chains with the WZ factorization for modelling networks, *Proceedings of 3rd International Conference Aplimat 2004*, Bratislava 2004, pp. 307–312.
- [5] J. Bylina: Distributed solving of Markov chains for computer network models, *Annales UMCS Informatica* 1 (2003), Lublin 2003, pp. 15–20.
- [6] J. Bylina, B. Bylina: GMRES dla rozwiązywania łańcuchów Markowa na komputerze wektorowym CRAY SV1, *Algorytmy, metody i programy naukowe*, Polskie Towarzystwo Informatyczne, Lublin 2004, pp. 19–24 (in Polish).

- [7] R. Fletcher: Conjugate gradient methods for indefinite systems, *Lecture Notes in Mathematica* 506, Springer-Verlag Berlin Heidelberg 1976, pp. 73–89.
- [8] W.K. Grassmann, M.I. Taskar, D.P. Heyman: Regenerative analysis and steady state distribution for Markov chains, *Operations Research* 33 (5), 1985, pp. 1107–1116.
- [9] W. Knottenbelt, P. Harrison: Distributed disk-based solution techniques for large Markov models, *Numerical Solution of Markov Chains*, Zaragoza, Spain, 1999, pp. 58–75.
- [10] Y. Saad: Krylov subspace methods for solving unsymmetric linear systems, *Mathematics of Computation* 37, 1981, pp. 105-126.
- [11] Y. Saad, M.H. Schultz: GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems, *SIAM Journal of Scientific and Statistical Computing* 7, 1986, pp. 856–869.
- [12] P. Sonneveld: CGS, a fast Lanczos-type solver for nonsymmetric linear systems, *SIAM Journal of Scientific and Statistical Computing* 10 (1), 1989, pp. 36–52.
- [13] W. Stewart: *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, Chichester, West Sussex, 1994.
- [14] Z. Szczerbiski: Parallel computing applied to solving large Markov chains. A feasibility study, *Studia Informatica*, Vol. 24, Number 4 (56), 2003, pp. 7-28.
- [15] D.M. Young: *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.