Online Companion to the Paper ‘Aggregation of Markovian Models - An Alternating Least Squares Approach -’[*]

Peter Buchholz and Jan Kriege
Department of Computer Science
TU Dortmund
Dortmund, Germany
Email: {peter.buchholz,jan.kriege}@udo.edu

June 13, 2012

Abstract

This includes some additional details as an extension to the paper “Aggregation of Markovian Models - An Alternating Least Squares Approach -” (QEST’12).

1 Introduction

The online companion is, of course, written to add information which could not be included in the paper due to space restrictions. In so far it is only understandable with the knowledge of the original paper [1]. It includes 3 sections which describe some basic steps to transform the used non negative least squares problems in standard form, the basic octave programs realizing the approach and some more example results. This online companion will be modified and extended from time to time. Therefore take a look on the date of the current version.

2 Transformation of the LLS Problems in Standard Form

We show how to transform the problems LLSEI (12) and NNLSE (15) into the form NNLS $\min_{\mathbf{x}:\mathbf{x}\geq\mathbf{0}}(\|\mathbf{Cx}\|)$. Equation numbers point to the numbers of the equations in the paper. Observe that the basis for the following steps can be found in the standard literature [2, 3] which is adopted to our specific needs here.

In the first step LLSEI is transformed into a problem of the type NNLSE. We consider general equalities of the form $\mathbf{Ex} = \mathbf{f}$, $\mathbf{Fx} \geq \mathbf{0}$ and assume that $\mathbf{C}$ is an $r \times c$ matrix, $\mathbf{F}$ is an $r_i \times c$ and $\mathbf{E}$ an $r_e \times c$ matrix. Consequently, $\mathbf{x}$ is a vector of length c and $\mathbf{f}$ is of length r_e.

2.1 Elimination of Inequalities

Define a new vector $\mathbf{0} \leq \mathbf{w} = \mathbf{Fx}$. For matrix $\mathbf{F}$ the following orthogonal factorization exists

$$\mathbf{H}^T\mathbf{FK} = \begin{pmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

[*]To appear at 9th Int. Conf. on Quantitative Analysis of Systems (QEST’12).

where $\mathbf{H}$, $\mathbf{K}$ are orthogonal matrices and $\mathbf{S}$ is a non-singular lower triangular matrix which can be computed with the algorithm proposed in [3, chap. 14] or any other algorithm for orthogonal matrix factorization [4]. $\mathbf{S}$ is an $c_1 \times c_1$ matrix and, of course, c_1 is also the rank of $\mathbf{F}$ and $c_1 \leq c$ since otherwise no solution for the inequalities exists. Now use the substitution $\mathbf{x} = \mathbf{K}\mathbf{y} \Rightarrow \mathbf{y} = \mathbf{K}^T\mathbf{x}$. With

$$\mathbf{y} = \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} \text{ and } \mathbf{H} = (\mathbf{H}1\ \mathbf{H}_2)$$

where $\mathbf{y}_i$ is of length c_i ($c_2 = c - c_1$). We have

$$\mathbf{F}\mathbf{x} = \mathbf{w} \Rightarrow (\mathbf{H}1\ \mathbf{H}_2) \begin{pmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = \mathbf{w}$$

and $\mathbf{S}\mathbf{y}_1 = \mathbf{H}_1^T\mathbf{w} \Rightarrow \mathbf{y}_1 = \mathbf{S}^{-1}\mathbf{H}_1^T\mathbf{w}$ and $\mathbf{0} = \mathbf{H}_2^T\mathbf{w}$ with $\mathbf{w} \geq \mathbf{0}$. This relations can be substituted in (12) yielding

$$\mathbf{C}\mathbf{x} = \mathbf{C}\mathbf{K} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = \mathbf{C}\mathbf{K} \begin{pmatrix} \mathbf{S}^{-1}\mathbf{H}_1^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{w} \\ \mathbf{y}_2 \end{pmatrix}.$$

Now define

$$\hat{\mathbf{C}} = \mathbf{C}\mathbf{K} \begin{pmatrix} \mathbf{S}^{-1}\mathbf{H}_1^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \text{ and } \hat{\mathbf{x}} = \begin{pmatrix} \mathbf{w} \\ \mathbf{y}_2 \end{pmatrix}$$

where $\hat{\mathbf{C}}$ is an $r \times (r_i + c_2)$ matrix. The inequalities are integrated into the equality constraints.

$$\mathbf{E}\mathbf{x} = \mathbf{f} \Rightarrow \mathbf{E}\mathbf{K} \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} = \mathbf{f} \Rightarrow$$
$$\mathbf{E}\mathbf{K} \begin{pmatrix} \mathbf{S}^{-1}\mathbf{H}_1^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{w} \\ \mathbf{y}_2 \end{pmatrix}.$$

Additionally, $\mathbf{H}_2^T\mathbf{w} = \mathbf{0}$ and $\mathbf{w} \geq \mathbf{0}$ has to hold. Then define

$$\hat{\mathbf{E}} = \begin{pmatrix} \mathbf{E}\mathbf{K} \begin{pmatrix} \mathbf{H}_1^T \\ \mathbf{0} \end{pmatrix} & \mathbf{E}\mathbf{K} \begin{pmatrix} \mathbf{0} \\ \mathbf{I} \end{pmatrix} \\ \mathbf{H}_2^T & \mathbf{0} \end{pmatrix} \text{ and } \hat{\mathbf{f}} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0} \end{pmatrix}$$

where $\hat{\mathbf{E}}$ is a $(r_e + r_i - c_1) \times (r_e + c_2)$ matrix.

Then $\min_{\hat{\mathbf{x}}} \left(\|\hat{\mathbf{C}}\hat{\mathbf{x}}\|\right)$ with the equality constraints $\hat{\mathbf{E}}\hat{\mathbf{x}} = \hat{\mathbf{f}}$ and $\hat{\mathbf{x}} = (\mathbf{w}, \mathbf{y}_2)^T$ where $\mathbf{w} \geq \mathbf{0}$ is an NNLSE where only some but not all variables need to be non-negative.

2.2 Elimination of Unconstrained Variables

The remaining problem equals $\min_{\mathbf{x}:\mathbf{x}\geq\mathbf{0}} (\|\mathbf{C}\mathbf{x}\|_2)$ with the equality constraints $\mathbf{E}\mathbf{x} = \mathbf{f}$, $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)^T$ and $\mathbf{x}_1 \geq \mathbf{0}$. Let $r \times c$ be the dimension of $\mathbf{C}$ and $r_e \times c$ the dimension of $\mathbf{E}$. We assume that $\mathbf{x}_1$ is of length c_1 and $\mathbf{x}_2$ is of length $c_2 = c_e - c_1$.

According to the decomposition of vector $\mathbf{x}$ the matrices $\mathbf{C}$ and $\mathbf{E}$ by considering the first c_1 columns and the remaining c_2 columns resulting in

$$(\mathbf{E}_1\ \mathbf{E}_2) \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \mathbf{f} \text{ and } \min \left\| (\mathbf{C}_1\ \mathbf{C}_2) \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} \right\|.$$

This implies $\mathbf{E}_2\mathbf{x}_2 = \mathbf{f} - \mathbf{E}_1\mathbf{x}_1$. As for the elimination of inequalities an orthogonal decomposition of $\mathbf{E}_2$ can be computed such that

$$\mathbf{H}^T\mathbf{E}_2\mathbf{K} = \begin{pmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

where $\mathbf{H}$ and $\mathbf{K}$ are orthogonal matrices and $\mathbf{S}$ is an $c_{21} \times c_{21}$ non-singular lower triangular matrix. Matrices $\mathbf{H}^T$ and K^T are written as

$$\mathbf{H}^T = \begin{pmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \end{pmatrix} \text{ and } \mathbf{K}^T = \begin{pmatrix} \mathbf{K}_1 \\ \mathbf{K}_2 \end{pmatrix}$$

where $\mathbf{K}_1$ and $\mathbf{H}_1$ involve the first c_{21} rows of $\mathbf{H}^T$ and $\mathbf{K}^T$, respectively. Observe that $\mathbf{K}_1$ and $\mathbf{H}_1$ are orthogonal matrices too. $\mathbf{E}_2\mathbf{x}_2 = \mathbf{f} - \mathbf{E}_1\mathbf{x}_1$ can be rewritten as

$$\begin{array}{ll}
\begin{pmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{K}_1 \\ \mathbf{K}_2 \end{pmatrix} \mathbf{x}_2 = \begin{pmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \end{pmatrix} (\mathbf{f} - \mathbf{E}_1\mathbf{x}_1) & \Leftrightarrow \\
\begin{pmatrix} \mathbf{S}\mathbf{K}_1 \\ \mathbf{0} \end{pmatrix} \mathbf{x}_2 = \begin{pmatrix} \mathbf{H}_1\mathbf{f} \\ \mathbf{H}_2\mathbf{f} \end{pmatrix} - \begin{pmatrix} \mathbf{H}_1\mathbf{E}_1 \\ \mathbf{H}_2\mathbf{E}_1 \end{pmatrix} \mathbf{x}_1 & \Rightarrow \\
\mathbf{S}\mathbf{K}_1\mathbf{x}_2 = \mathbf{H}_1\mathbf{f} - \mathbf{H}_1\mathbf{E}_1\mathbf{x}_1 & \Leftrightarrow \\
\mathbf{x}_2 = \underbrace{\mathbf{K}_1^T\mathbf{S}^{-1}\mathbf{H}_1\mathbf{f}}_{\mathbf{f}_2} - \underbrace{\mathbf{K}_1^T\mathbf{S}^{-1}\mathbf{H}_1\mathbf{E}_1\mathbf{x}_1}_{-\mathbf{E}_{12}\mathbf{x}_1} &
\end{array}$$

Substituting this representation of $\mathbf{x}_2$ into the optimization problem we obtain

$$(\mathbf{E}_1 + \mathbf{E}_2\mathbf{E}_{12})\,\mathbf{x}_1 = \mathbf{f} - \mathbf{f}_2 \text{ and } \min \|(\mathbf{C}_1 + \mathbf{C}_2\mathbf{E}_{12})\,\mathbf{x}_1\|$$

which has to be solved for $\mathbf{x}_1 \geq \mathbf{0}$. Renaming of the variables such that $\mathbf{E} = \mathbf{E}_1 + \mathbf{E}_2\mathbf{E}_{12}$, $\mathbf{f} = \mathbf{f} - \mathbf{f}_2$, $\mathbf{C} = \mathbf{C}_1 + \mathbf{C}_2\mathbf{E}_{12}$ and $\mathbf{x} = \mathbf{x}_1$ results in a NNLSE problem with c_1 variables.

2.3 Integration of the Equalities

The remaining problem is to solve $\min(\|\mathbf{C}\mathbf{x}\|)$ under the constraints $\mathbf{E}\mathbf{x} = \mathbf{f}$ and $\mathbf{x} \geq \mathbf{0}$ (problem NNLSE). The commonly used approach for the solution of the problem is the use of weighting to integrate the equalities in the minimization problem and the subsequent solution of the problem NNLS. This approach is also proposed in [2]. The idea of weighting is to substitute the constrained problem by an unconstrained problem with a penalty function of the equality constraints. Thus, we consider the unconstrained NNLS

$$\min_{\mathbf{x}:\mathbf{x}\geq\mathbf{0}} \|\mathbf{C}_\epsilon\mathbf{x} - \mathbf{f}_\epsilon\|$$

where $\mathbf{C}_\epsilon = \begin{pmatrix} \mathbf{E} \\ \epsilon\mathbf{C} \end{pmatrix}$ and $\mathbf{f}_{epsilon} = \begin{pmatrix} \mathbf{f} \\ \mathbf{0} \end{pmatrix}$.

As proved in [2] the solution of the unconstrained problem converges to the solution of the constrained problem for $\epsilon \to \mathbf{0}$. The solution can be computed with the algorithm proposed in [3] or its variant [5]. Further details about the structure of the problem and the choice of ϵ can be found in [2]. It should be noted that if $\mathbf{E}\mathbf{x} = \mathbf{f}$ has no non-negative solution, then the algorithm computes a vector $\mathbf{x}$ that minimizes $\|\mathbf{E}\mathbf{x} - \mathbf{f}\|$.

3 Programs in *octave* to perform the minimization

In the previous section the mathematical basis of the optimization problem is given which allows one to implement a solver based on the solution of unconstrained least squares problems. Here we present another approach to realize the approach, namely the use of the standard optimization solver for quadratic problems in the software package *octave qt*. For details of *octave* and the solver we refer to [6, 7].

3.1 Algorithm 1 in *octave*

```
function
        [B0, B1, B2, W, err] = agg_iter(iniv, A0, A1, A2, B0, B1, B2, V, max_eps);
% AGG_ITER
% $iniv = 1xm$ initial vector
% matrices of the original component
% $A0 = m \times m$ matrix with internal transition probabilities
% $A1 = m \times m$ matrix with output event transition probabilities
% $A2 = m \times m$ matrix with input event transition probabilities
% $max\_eps$ allowed difference between elements
% $n$ size of the aggregate ($< m$)
% OUTPUT:
% matrices for the aggregate
% $B0 = n \times n$ matrix with internal transition probabilities
% $B1 = n \times n$ matrix with output event transition probabilities
% $B2 = n \times n$ matrix with input event transition probabilities
% $V = m \times n$ transformation matrix
% DESCRIPTION:
% The function computes iteratively the aggregate and the corresponding
% transformation matrix
%
%
% set the dimensions
  $m = rows(A0);$
  $n = columns(V);$
  if $n > m$
    error('Number of states in orginal smaller than number of states for the aggregate') ;
  end
  if $rows(A0)! = m||rows(A1)! = m||rows(A2)! = m$
    error('Wrong matrix dimension for $A$ matrix') ;
    return ;
  endif
  if $rows(B0)! = n||rows(B1)! = n||rows(B2)! = n$
    $B0 = B1 = B2 = zeros(n);$
  endif
  $iter = 1;$
  do
    $max\_diff = 0.0;$
    $[C0, C1, C2] = agg\_comp\_b(A0, A1, A2, B0, B1, B2, V);$
    $W = agg\_comp\_v(iniv, A0, A1, A2, C0, C1, C2, V);$
    $iter = iter + 1;$
    $diff = norm(V - W, inf);$
    if $diff > max\_diff$
      $max\_diff = diff$ ;
    end
    $diff = norm(B0 - C0, inf);$
    if $diff > max\_diff$
      $max\_diff = diff;$
    end
    $diff = norm(B1 - C1, inf);$
    if $diff > max\_diff$
      $max\_diff = diff;$
    end
    $diff = norm(B2 - C2, inf);$
    if $diff > max\_diff$
      $max\_diff = diff;$
    end
    $V = W;$
    $B0 = C0;$
    $B1 = C1;$
    $B2 = C2;$
  until $max\_diff < max\_eps || iter > 10;$
end
```

The function can be used to compute approximate aggregates with $K \leq 2$. It uses two subfunction *agg_comp_b* to compute a new component $\mathcal{B}$ (see equation (14) and (15)) and *agg_comp_v* to compute a new matrix $\mathbf{V}$ (see equation (12)). Both functions use the octave standard solver *qp*.

```
function [C0, C1, C2] = agg_comp_b(A0, A1, A2, B0, B1, B2, V);
% INPUT:
% matrices of the original component
% A0 = m x m matrix with internal transition probabilities
% A1 = m x m matrix with output event transition probabilities
% A2 = m x m matrix with input event transition probabilities
% matrices of the aggregate
% B0 = n x n matrix with internal transition probabilities
% B1 = n x n matrix with output event transition probabilities
% B2 = n x n matrix with input event transition probabilities
% V = m x n transformation matrix V >= 0 and V*1 = 1
% it is assumed that all matrices are non-negative and
% A2*1 = 1 as well as (A0 + A1)*1 = 1
% m is the order of the original component
% n (<= m) is the order of the aggregate
% OUTPUT:
% improved matrices for the aggregate
% C0 = n x n matrix with internal transition probabilities
% C1 = n x n matrix with output event transition probabilities
% C2 = n x n matrix with input event transition probabilities
% DESCRIPTION:
% The function computes improved matrices for the aggregate from the matrices
% of the original component and an available transformation matrix V
%
% set the dimensions
  m = rows(V);
  n = columns(V);
  % first the row sums for B0 and B1 are computed
  % generate matrix H, q, and E
  H = kron(eye(2), V' * V);
  c = [(A0 * ones(m, 1))', (A1 * ones(m, 1))'];
  c = -c';
  q = kron(eye(2), V') * c;
  E = kron(ones(1, 2), eye(n));
  % Solve the optimization problem for b0 and b1
  [x, obj, info, lambda] = qp([], H, q, E, ones(n, 1), zeros(2 * n, 1), ones(2 * n, 1));
  if info.info > 0
    printf("QLP for (b0, b1)' not successful info %i\n", info.info) ;
  end
  b0 = x(1 : n);
  b1 = x(n + 1 : 2 * n);
  % Now the matrices C0, C1 and C2 have to be computed
  E = kron(ones(1, n), eye(n));
  H = kron(eye(n), V') * kron(eye(n), V);
  c = -1 * reshape(A0 * V, n * m, 1);
  q = kron(eye(n), V') * c;
  x = reshape(B0, n * n, 1);
  [x, obj, info, lambda] = qp(x, H, q, E, b0, zeros(n * n, 1), ones(n * n, 1));
  if info.info > 0
    printf("QLP for C0 not successful info %i\n", info.info) ;
  end
  C0 = reshape(x, n, n);
  c = -1.0 * reshape(A1 * V, n * m, 1);
  q = kron(eye(n), V') * c;
  x = reshape(B1, n * n, 1);
  [x, obj, info, lambda] = qp(x, H, q, E, b1, zeros(n * n, 1), ones(n * n, 1));
  if info.info > 0
    printf("QLP for C1 not successful info %i\n", info.info) ;
  end
  C1 = reshape(x, n, n);
  c = −1.0 * reshape(A2 * V, n * m, 1);
  q = kron(eye(n), V') * c;
  f = ones(n, 1);
  x = reshape(B2, n * n, 1);
  [x, obj, info, lambda] = qp(x, H, q, E, f, zeros(n * n, 1), ones(n * n, 1));
  if info.info > 0
    printf("QLP for C2 not successful info %i\n", info.info) ;
  end
  C2 = reshape(x, n, n);
end

function W = agg_comp_v(iniv, A0, A1, A2, B0, B1, B2, V);
% INPUT:
% matrices of the original component
% pi = 1 x m initial vector of the original component
% A0 = m x m matrix with internal transition probabilities
% A1 = m x m matrix with output event transition probabilities
% A2 = m x m matrix with input event transition probabilities
% matrices of the aggregate
% B0 = n x n matrix with internal transition probabilities
% B1 = n x n matrix with output event transition probabilities
% B2 = n x n matrix with input event transition probabilities
% V = m x n transformation matrix V >= 0 and V*1 = 1
% it is assumed that all matrices are non-negative and
% A2*1 = 1 as well as (A0 + A1)*1 = 1
% m is the order of the original component
% n (<= m) is the order of the aggregate
% OUTPUT:
% improved transformation matrix W
% DESCRIPTION:
% The function computes an improved transformation matrix
%
% initialize the dimensions
```

```
  m = rows(V);
  n = columns(V);
  % first the row sums for B0 and B1 are computed
  % generate matrices F, E and H
  % F is only needed, if negative elements are allowed for V
  % F = kron(eye(n), iniv);
  E = kron(ones(1, n), eye(m));
  H = kron(eye(n), A0' * A0) + kron(B0 * B0', eye(m)) - kron(B0, A0) - kron(B0', A0');
  H = H + kron(eye(n), A1' * A1) + kron(B1 * B1', eye(m)) - kron(B1, A1) - kron(B1', A1');
  H = H + kron(eye(n), A2' * A2) + kron(B2 * B2', eye(m)) - kron(B2, A2) - kron(B2', A2');
  q = zeros(m * n, 1);
  f = ones(m, 1);
  x = reshape(V, m * n, 1);
  % Solve the optimization problem for b0 and b1
  % without negative elements
  [x, obj, info, lambda] = qp(x, H, q, E, f, zeros(m * n, 1), ones(m * n, 1));
  % with negative elements
  % [x, obj, info, lambda] = qp(x, H, q, E, f, -10*ones(m*n, 1), 10*ones(m*n, 1).zeros(n, 1), F, ones(n, 1));
  if info.info > 0
    printf("QLP for W not successful info %i\n", info.info) ;
  end
  W = reshape(x, m, n);
end
```

3.2 Algorithm 2 in *octave*

The function *hcl* in the following procedure realizes the hierarchical clustering approach.

```
function [B0, B1, B2, W] = agg_overall(A0, A1, A2, n);
% INPUT:
% matrices of the original component
% A0 = m x m matrix with internal transition probabilities
% A1 = m x m matrix with output event transition probabilities
% A2 = m x m matrix with input event transition probabilities
% n size of the aggregate
% OUTPUT:
% matrices for the aggregate
% B0 = n x n matrix with internal transition probabilities
% B1 = n x n matrix with output event transition probabilities
% B2 = n x n matrix with input event transition probabilities
% V = m x n transformation matrix
% DESCRIPTION:
% The function computes iteratively the aggregate and the corresponding
% transformation matrix
%
% set the dimensions
 $m = rows(A0);$
 if $n > m$
   error('Number of states in orginal smaller than number of staes for the aggregate') ;
 end
 if $rows(A0)! = m||rows(A1)! = m||rows(A2)! = m$
   error('Wrong matrix dimension for A matrix') ;
   return ;
 endif
 $B0 = B1 = B2 = zeros(n);$
 $V = agg\_new\_v(m, n, 0);$
 $iniv = zeros(1, m);$
 for $i = 1 : m$
   $iniv(i) = 1.0/m;$
 endfor
 $iter = 1;$
 $min\_err = 1.0e + 6;$
 while $iter < 3$
   $Z = V;$
   $[B0, B1, B2, V, err] = agg\_iter(iniv, A0, A1, A2, B0, B1, B2, V, 1.0e - 3);$
   if $err < min\_err$
     $C0 = B0;$
     $C1 = B1;$
     $C2 = B2;$
     $W = V;$
     $min\_err = err;$
     if $min\_err < 1.0e - 12$
       break ;
     endif
   endif
   $[M, cl] = hcl(V, n);$
   % Check the number of clusters
   for $i = 1 : n$
     $count(i) = 0;$
   endfor
   $no\_cluster = 0;$
```

```
    for $i = 1 : m$
      $count(cl(i)) + +;$
      if $count(cl(i)) == 1$
        $no\_cluster + +;$
      endif
    endfor
    if $no\_cluster! = columns(V)$
      printf("Reset the number of aggregated states from %i to %i\n", $columns(V)$, $no\_cluster$) ;
    endif
    $V = zeros(m, n);$
    for $i = 1 : m$
      $V(i, cl(i)) = 1;$
    endfor
    if $norm(V - Z) < 1.0e - 6||norm(V - W) < 1.0e - 6$
      printf("Norm(V-Z) %.3e norm(V-W) %.3e\n", $norm(V - Z)$, $norm(V - W)$) ;
      break ;
    endif
    $iter + +;$
  endwhile
end
```

4 Some more Example Results

References

[1] P. Buchholz and J. Kriege, "Aggregation of markovian models - an alternating least squares approach -," in *Proc. 9th Int. Conf. on Quantitative Evaluation of Systems (QEST).* IEEE Press, 2012.

[2] K. H. Haskell and R. J. Hanson, "An algorithm for linear least squares problems with equality and nonnegativity constraints," *Mathematical Porgramming*, vol. 21, pp. 98–118, 1981.

[3] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, ser. Classics in Applied Mathematics. SIAM, 1995.

[4] G. H. Golub and C. F. van Loan, *Matrix Computations.* John Hopkins University Press, 1996.

[5] J. Cantarella and M. Piatek, "tsnnls: A solver for large sparse least squares problems with non-negative variables," http://www.michaelpiatek.com/papers/tsnnls.pdf.

[6] "Gnu octave," http://www.gnu.org/software/octave/.

[7] "Quadratic programming in octave," http://www.gnu.org/software/octave/doc/interpreter/Quadratic-Programming.html.