ELSEVIER

Chemometrics and Intelligent Laboratory Systems 42 (1998) 93–103

Chemometrics and intelligent laboratory systems

Improving the speed of multi-way algorithms: Part I. Tucker3

Claus A. Andersson *, Rasmus Bro [1]

*Chemometrics Group, Food Technology, Department of Dairy and Food Science, Royal Veterinary and Agricultural University, Roligheds*v*ej 30, iii, DK-1958 Frederiksberg C, Denmark*

Abstract

In an attempt to improve the speed of multi-way algorithms, this paper investigates several different implementations of the Tucker3 algorithm. The interest is specifically aimed at developing a fast algorithm in the MATLAB™ environment that is suitable for large data arrays. Nine different implementations are developed and tested on real and simulated data. In a subsequent paper, it will be demonstrated that a fast algorithm for the Tucker3 model provides a perfect basis for improving the speed of other multi-way algorithms. From the Internet address http:\\newton.mli.kvl.dk\foodtech.html, the developed algorithms can be downloaded.

Keywords: Tucker3; Three-mode factor analysis; 3-MFA; Three-way principal component analysis

1. Introduction

The Tucker3 model, or *N*-way PCA, is one of the most basic multi-way models used in chemometrics. It originates from psychometrics from the pioneering work of Tucker [1], and the algorithmic solution for estimating the model was later substantially improved by Kroonenberg and de Leeuw [2] and ten Berge et al. [3]. Several successful applications have been demonstrated in quite different areas such as chromatography [4], environmental analysis [5] and person perception analysis [6]. Having an efficient algorithm especially for large data sets is therefore of utmost importance. Several different algorithms have been described in the literature. Almost all are based on least squares regression, singular value decomposition (SVD), Gram–Schmidt (GS) orthogonalization, or a modified Bauer–Rutishauer (BR) estimation. In this paper, all steps in the Tucker3 algorithm will be optimized with respect to speed. The focus will be on three-way models, but all results are equally applicable on models of higher orders [7]. In the sequel, nine different algorithms will be compared as they have been implemented in MATLAB, and it will also shortly be described how the algorithms can be modified to handle missing values and data with different uncertainties.

The sizes of the arrays considered are such that the computer has physical memory to hold the array and intermediate working arrays. If the array size exceeds what the physical computer memory can hold, other problems arise and other algorithms may be better (see Refs. [8–10]). These algorithms do not work with exact least-squares solutions, but rather try to approximate the solution by finding suitable bases

* Corresponding author. E-mail: ca@kvl.dk.
[1] E-mail: rb@kvl.dk.

0169-7439/98/$19.00
PII: S0169-7439(98)00010-0

for the different modes. An efficient algorithm for the case of one large mode has also been proposed [11]. For arrays whose size does not exceed the potential computer power, it is not necessary to compress the array prior to modelling as most Tucker3 algorithms are quite fast. The purpose of this paper is to provide the fastest way of estimating the Tucker3 model, and implicitly providing suitable bases for the modes of large arrays.

2. Theory

In the following, scalars are indicated by italics, vectors by bold lower-case characters, bold capitals are used for two-way matrices, and underlined bold capitals for three-way arrays. The letters I, J, and K are reserved for indicating the dimension of the three different modes. The ijkth element of $\mathbf{X}$ is called x_{ijk} and is the element in the ith row, jth column, and kth tube of $\mathbf{X}$. When three-way arrays are unfolded to matrices, the following notation will be used: If $\mathbf{X}$ is an $I \times J \times K$ array and is unfolded to an $I \times JK$ matrix, the order of J and K indicates which indices are running fastest. In this case, the indices of J are running fastest, meaning that the first J columns of $\mathbf{X}$ contain all variables for $k = 1$ and for $j = 1$ to $j = J$. In short, we will term the $I \times JK$ matrix $\mathbf{X}^{(1)}$, where the superscript indicates that it is the *first* mode that is preserved. Likewise $\mathbf{X}^{(2)}$ is a $J \times IK$ matrix and $\mathbf{X}^{(3)}$ a $K \times IJ$ matrix. If the arrangement of the array is clear from the context, the superscript will not be shown.

An $I \times J \times K$ array $\mathbf{X}$ is given and a Tucker3 model of ranks R^{A}, R^{B}, and R^{C} respectively is sought. Written in matrix notation letting $\mathbf{X}^{(1)}$ be the $I \times JK$ unfolded array, and $\otimes$ denoting the Kronecker product, the Tucker3 model reads

$$\mathbf{X}^{(1)} = \mathbf{A}\mathbf{G}^{(1)}(\mathbf{C}^{\mathrm{T}} \otimes \mathbf{B}^{\mathrm{T}}) + \mathbf{E}^{(1)}, \tag{1}$$

where $\mathbf{A}\mathbf{G}^{(1)}(\mathbf{C}^{\mathrm{T}} \otimes \mathbf{B}^{\mathrm{T}})$ is the model of $\mathbf{X}^{(1)}$, $\mathbf{E}^{(1)}$ is the unmodelled part, i.e., the residuals of the model, and $\mathbf{G}^{(1)}$ is the core array $\mathbf{G}$ arranged as an $R^{\mathrm{A}} \times R^{\mathrm{B}}R^{\mathrm{C}}$ matrix. In Eq. (1), $\mathbf{A}$ has size $I \times R^{\mathrm{A}}$, $\mathbf{B}$ has size $J \times R^{\mathrm{B}}$, and $\mathbf{C}$ has size $K \times R^{\mathrm{C}}$ and the matrices hold the loadings in the first, second, and third mode, respectively. In the following, we will omit the residual part for simplicity. We restrict ourselves to estimate the Tucker3 model with orthonormal $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. We further restrict ourselves to algorithms based on iteratively estimating one of the four sets of parameters $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{G}$ conditionally on the remaining parameters. In most cases, such an algorithm will be a so-called alternating least squares (ALS) algorithm.

The core array $\mathbf{G}$ can be found conditional on $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ by a simple projection of $\mathbf{X}$ onto $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. In matrix notation this reads

$$\mathbf{G}^{(1)} = \mathbf{A}^{\mathrm{T}}\mathbf{X}^{(1)}(\mathbf{C} \otimes \mathbf{B}) \tag{2}$$

If the model is perfect, then $\mathbf{G}$ will express all variation of $\mathbf{X}$. For completeness, note that $\mathbf{G}$ can also be computed from $\mathbf{X}$ arranged as $\mathbf{J} \times \mathbf{IK}$ or a $\mathbf{K} \times \mathbf{IJ}$ matrices:

$$\mathbf{G}^{(2)} = \mathbf{B}^{\mathrm{T}}\mathbf{X}^{(2)}(\mathbf{C} \otimes \mathbf{A}), \text{and}\, \mathbf{G}^{(3)} = \mathbf{C}^{\mathrm{T}}\mathbf{X}^{(3)}(\mathbf{B} \otimes \mathbf{A}), \tag{3}$$

From the definition of $\mathbf{G}$ it follows that the Tucker3 model of $\mathbf{X}$ can be stated

$$\begin{aligned}\mathbf{A}\mathbf{G}^{(1)}(\mathbf{C}^{\mathrm{T}} \otimes \mathbf{B}^{\mathrm{T}}) &= \mathbf{A}\mathbf{A}^{T}\mathbf{X}^{(1)}(\mathbf{C} \otimes \mathbf{B})(\mathbf{C}^{\mathrm{T}} \otimes \mathbf{B}^{\mathrm{T}}) \\ &= \mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{X}^{(1)}(\mathbf{C}\mathbf{C}^{\mathrm{T}} \otimes \mathbf{B}\mathbf{B}^{\mathrm{T}})\end{aligned} \tag{4}$$

For $\mathbf{B}$ and $\mathbf{C}$ fixed it follows that finding the optimal $\mathbf{A}$ is equal to minimizing the norm of $(\mathbf{X} - \mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{M})$, where $\mathbf{M} = \mathbf{X}(\mathbf{C}\mathbf{C}^{\mathrm{T}} \otimes \mathbf{B}\mathbf{B}^{\mathrm{T}})$. Using that

$$\begin{aligned}&(\mathbf{C}\mathbf{C}^{\mathrm{T}} \otimes \mathbf{B}\mathbf{B}^{T})(\mathbf{C}\mathbf{C}^{T} \otimes \mathbf{B}\mathbf{B}^{T})^{T} \\ &\quad = (\mathbf{C}\mathbf{C}^{T}\mathbf{C}\mathbf{C}^{T} \otimes \mathbf{B}\mathbf{B}^{T}\mathbf{B}\mathbf{B}^{T}) = (\mathbf{C}\mathbf{C}^{T} \otimes \mathbf{B}\mathbf{B}^{T}),\end{aligned} \tag{5}$$

and by tr denoting the trace of the square argument matrix, the sought norm is

$$\begin{aligned}&\mathrm{tr}\left((\mathbf{X} - \mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{M})(\mathbf{X} - \mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{M})^{\mathrm{T}}\right) \\ &\quad = \mathrm{tr}(\mathbf{X}\mathbf{X}^{\mathrm{T}}) - 2\mathrm{tr}(\mathbf{A}\mathbf{A}^{T}\mathbf{M}\mathbf{X}^{\mathrm{T}}) \\ &\qquad + tr(\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{M}^{\mathrm{T}}\mathbf{A}\mathbf{A}^{\mathrm{T}}) \\ &\quad = \mathrm{tr}(\mathbf{X}\mathbf{X}^{\mathrm{T}}) - 2\mathrm{tr}(\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{X}^{\mathrm{T}}) \\ &\qquad + \mathrm{tr}(\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{A}^{\mathrm{T}}).\end{aligned} \tag{6}$$

As $\mathrm{tr}(\mathbf{X}\mathbf{X}^{\mathrm{T}})$ is fixed, minimizing this expression is equal to minimizing

$$\begin{aligned}&-2\mathrm{tr}(\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{X}^{\mathrm{T}}) + \mathrm{tr}(\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{X}^{\mathrm{T}}\mathbf{A}\mathbf{A}^{\mathrm{T}}) \\ &\quad - 2\mathrm{tr}(\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{X}^{\mathrm{T}}\mathrm{A}) + \mathrm{tr}(\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{X}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}) \\ &\quad - \mathrm{tr}(\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{X}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}})\end{aligned} \tag{7}$$

and hence the optimal **A** is found by maximizing

$$\mathrm{tr}(\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{X}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}) = \mathrm{tr}(\mathbf{A}^{\mathrm{T}}\mathbf{M}\mathbf{M}^{\mathrm{T}}\mathbf{A}) \quad (8)$$

which shows that **A** is the R^{A} largest eigenvectors of $\mathbf{M}\mathbf{M}^{\mathrm{T}}$ or equivalently the first R^{A} left singular vectors of a singular value decomposition of **M**.

For estimating **B** and **C** similar relations hold, and these relations form the basis for an algorithm for estimating the Tucker3 model. The essentials of such an algorithm are outlined in the generic algorithm below:

1. Initialize **B** and **C**.
2. Calculate $\mathbf{M}^{(1)}$ from **B**, **C** and $\mathbf{X}^{(1)}$. Calculate **A**.
3. Calculate $\mathbf{M}^{(2)}$ from **C**, **A** and $\mathbf{X}^{(2)}$. Calculate **B**.
4. Calculate $\mathbf{M}^{(3)}$ from **A**, **B** and $\mathbf{X}^{(3)}$. Calculate **C**.
5. Goto step one until convergence
6. Calculate the core **G**

Before going into the details of the algorithm, it is appropriate to elaborate on the computation of **M** and $\mathbf{M}\mathbf{M}^{\mathrm{T}}$. As **A** is a basis for the column space of the best fitted rank R^{A} approximation of $\mathbf{M} = \mathbf{X}(\mathbf{C}\mathbf{C}^{\mathrm{T}} \otimes \mathbf{B}\mathbf{B}^{\mathrm{T}})$, it follows that **A** can also be determined from the much smaller matrix $\mathbf{X}(\mathbf{C} \otimes \mathbf{B})$. The cross-product of **M** is derived from

$$\begin{aligned}&(\mathbf{X}(\mathbf{C}\mathbf{C}^{\mathrm{T}} \otimes \mathbf{B}\mathbf{B}^{\mathrm{T}})(\mathbf{X}(\mathbf{C}\mathbf{C}^{\mathrm{T}} \otimes \mathbf{B}\mathbf{B}^{\mathrm{T}}))^{\mathrm{T}}\\ &\quad = \mathbf{X}(\mathbf{C}\mathbf{C}^{\mathrm{T}} \otimes \mathbf{B}\mathbf{B}^{\mathrm{T}})\mathbf{X}^{\mathrm{T}}\\ &\quad = (\mathbf{X}(\mathbf{C} \otimes \mathbf{B})(\mathbf{X}(\mathbf{C} \otimes \mathbf{B}))^{\mathrm{T}} \quad (9)\end{aligned}$$

and can hence also be computed from the smaller matrix $\mathbf{X}(\mathbf{C} \otimes \mathbf{B})$.

There are several important steps in actually implementing the Tucker3 algorithm for large problems: (i) avoiding the use of Kronecker products and unnecessarily large working matrices, (ii) a good initialisation method, (iii) if possible, avoiding intermediate estimation of the core array, which is algorithmically unnecessary, and (iv) a fast method for estimating an F-component orthonormal basis for a matrix. In the following, we will use the update of **A** as an example.

Ad (i) It is very common to express the Tucker3 model and algorithm using Kronecker products. While intuitively appealing for providing simple matrix expressions for array models, this approach should not be adopted in the actual implementation, as it leads to very large intermediate arrays and excessively many elementary operations. Instead, one should rearrange the arrays continuously as exemplified below. This approach is justified by the fact that rearranging a matrix is very fast, as it only requires changes in indices, not real computations.

The projections $\mathbf{X}(\mathbf{C} \otimes \mathbf{B})$ can be written in matrix notation:

$$\mathbf{W}^{(2)} = \mathbf{B}^{\mathrm{T}}\mathbf{X}^{(2)}$$
$$\mathbf{V}^{(3)} = \mathbf{C}^{\mathrm{T}}\mathbf{W}^{(3)}$$
$$\mathbf{X}^{(1)}(\mathbf{C} \otimes \mathbf{B}) = \mathrm{V}^{(1)}$$

Though complicated to look at, this way of computing the projections is much faster than directly using the Kronecker products—in particular for large arrays. From version 5.0 of MATLAB, general arrays are supported, thus eliminating the need to specifically program these rearrangements.

Ad (ii) There is no need for initializing the first mode, i.e., **A**, as this is given by $\mathbf{X}^{(1)}$, **B** and **C** in the first iteration according to the algorithm in question. The most straightforward method for initializing **B** and **C** is to use the R^{B} and R^{C} left singular vectors from an SVD of $\mathbf{X}^{(2)}$ and $\mathbf{X}^{(3)}$. A slight change is suggested here. As above, matrix **B** is the first R^{C} left singular vectors from an SVD of the $J \times IK$ matrix $\mathbf{X}^{(2)}$. Subsequently, **C** is obtained as the R^{C} first left singular vectors from an SVD of $(\mathbf{B}^{\mathrm{T}}\mathbf{X}^{(2)})^{(3)}$. In this way, the initial **B** and **C** are likely to be closer to the solution than results from the SVDs on the separated modes would be. In addition, **C** is derived from a matrix of size $K \times R^{\mathrm{B}}I$. As such, this initialization scheme requires fewer computations than if the separate SVDs should be calculated. The order in which the component matrices **B** and **C** are calculated is of no importance, and one should choose the smallest of the two first.

Ad (iii) As the core array of the model is implicitly given by **A**, **B** and **C**, one can simply calculate it once after convergence. But, instead of estimating the full model of **X** to determine the error after each iteration, the sum of the squared core entries provides a robust and monotonically increasing parameter that may be used to detect convergence. During iterations, the sum of the squared residuals, **E**, is mini-

mized. Denoting by $\|\ \|_2^2$, the square of the 2-norm of the argument, we formulate this as $\min \|\mathbf{E}\|_2^2 = \min\|\mathbf{X} - \mathbf{M}\|_2^2 = \min\|\mathbf{X}\|_2^2 - \|\mathbf{M}\|_2^2$. Since $\|\mathbf{M}\|_2^2 = \|\mathbf{G}\|_2^2$ for orthonormal factors, this corresponds to maximizing $\|\mathbf{G}\|_2^2$. Thus, in the implementations under discussion, we calculate the core to use the sum of the squared core elements to detect convergence.

Ad (iv) The very essential part of the Tucker3 algorithm is the derivation of orthonormal loading matrices. Using **M**, the size of the matrix from which **A** is calculated is $I \times JK$. Using $\mathbf{X(C \otimes B)}$ the size is only $I \times R^{\mathrm{A}}R^{\mathrm{B}}$. In addition, the computation of $\mathbf{X(C \otimes B)}$ is much faster than the computation of $\mathbf{X(CC}^{\mathrm{T}} \otimes \mathbf{BB}^{\mathrm{T}})$. The following procedures have been tested for determining **A** given the matrix $\mathbf{X(C \otimes B)}$:

- SVD on $\mathbf{X(C \otimes B)}$
- Approximate Bauer–Rutishauser on $\mathbf{X}(\mathrm{C} \otimes \mathbf{B(X(C \otimes B))}^{\mathrm{T}}$
- Exact Bauer–Rutishauser on $\mathbf{X(C \otimes B)(X(C \otimes B))}^{\mathrm{T}}$
- Gram–Schmidt orthogonalization of $\mathbf{X(C \otimes B)(X(C \otimes B))}^{\mathrm{T}}$
- NIPALS on $\mathbf{X(C \otimes B)(X(C \otimes B))}^{\mathrm{T}}$

Preliminary investigations did include QR factorization of $\mathbf{X(C \otimes B(X(C \otimes B))}^{\mathrm{T}}$, but since this approach invariably gives results similar to GS, we chose to leave this approach out of the discussion.

2.1. Algorithms

We will shortly describe the implemented variations of the Tucker3 algorithm by showing the update of the first mode loadings in pseudo-code. It is assumed that only the first R^{A} principal vectors are used in SVD and NIPALS. By baurut we mean an algorithm that estimates eigenvectors according to the principle of Bauer–Rutishauser and by gsm we mean an algorithm that orthonormalizes according to the Gram–Schmidt procedure. It should be noted that the calls to the baurut and nipals algorithms use the previous iterates of the factors as initial guesses in order to save computing time. Kroonenberg et al. [11] have compared the Gram–Schmidt orthogonalization with the method of Bauer–Rutishauser.

T1: SVD-based algorithm

$\mathbf{M} = \mathbf{X(C \otimes B)}$

$\mathbf{[A,S,V]} = \mathrm{svd}(\mathbf{M})$

T2: Bauer-Rutishauser I algorithm. One-step update in each mode

$\mathbf{M} = \mathbf{(X(C \otimes B))(X(C \otimes B))}^{\mathrm{T}}$

$\mathbf{[U,S,V]} = \mathrm{svd}(\mathbf{A}^{\mathrm{T}}\mathbf{M}^2\mathbf{A})$

$\mathbf{A} = \mathbf{MAUS}^{-1/2}$

T3: Bauer–Rutishauser II algorithm. Three-step update in each mode

$\mathbf{M} = \mathbf{(X(C \otimes B))(X(C \otimes B))}^{\mathrm{T}}$

for $i = 1$ to 3

$\mathbf{[U,S,V]} = \mathrm{svd}(\mathbf{A}^{\mathrm{T}}\mathbf{M}^2\mathbf{A})$

$\mathbf{A} = \mathbf{MAUS}^{-1/2}$

end

T4: Bauer–Rutishauser III algorithm. Repeated update in each mode until convergence of **A**

$\mathbf{M} = \mathbf{(X(C \otimes B))(X(C \otimes B))}^{\mathrm{T}}$

while **A** has not converged

$\mathbf{[U,S,V]} = \mathrm{svd}(\mathbf{A}^{\mathrm{T}}\mathbf{M}^2\mathbf{A})$

$\mathbf{A} = \mathbf{MAUS}^{-1/2}$

endwhile

T5: Bauer–Rutishauser IV algorithm. Advanced BR algorithm

$\mathbf{M} = \mathbf{(X(C \otimes B))(X(C \otimes B))}^{\mathrm{T}}$

$\mathbf{A} = \mathrm{baurut}(\mathbf{M},\mathbf{A})$

T6: Gram–Schmidt I algorithm. One-step update in each mode

$\mathbf{M} = \mathbf{(X(C \otimes B))(X(C \otimes B))}^{\mathrm{T}}$

$\mathbf{A} = \mathrm{gsm}(\mathbf{MA})$

T7: Gram–Schmidt II algorithm. Three-step update in each mode

$\mathbf{M} = (\mathbf{X}(\mathbf{C} \otimes \mathbf{B}))(\mathbf{X}(\mathbf{C} \otimes \mathbf{B}))^{\mathrm{T}}$

for $i = 1$ to 3

$\mathbf{A} = \mathrm{gsm}(\mathbf{MA})$

end

T8: Gram–Schmidt III algorithm. Repeated update in each mode until convergence of **A**

$\mathbf{M} = (\mathbf{X}(\mathbf{C} \otimes \mathbf{B}))(\mathbf{X}(\mathbf{C} \otimes \mathbf{B}))^{\mathrm{T}}$

while **A** has not converged

$\mathbf{A} = \mathrm{gsm}(\mathbf{MA})$

endwhile

T9: NIPALS-based algorithm.

$\mathbf{M} = \mathbf{X}(\mathbf{C} \otimes \mathbf{B})$

$[\mathbf{A},\mathbf{P}] = \mathrm{nipals}(\mathbf{M},\mathbf{A})$

As the principles of SVD (Algorithm 1) and NIPALS (Algorithm 9) are well known and widely used in chemometrics, we will elaborate on the Bauer–Rutishauser algorithm and the Gram–Schmidt orthogonalization. For the Bauer–Rutishauser algorithm, we investigate four different methods: A simple one-step BR update (Algorithm 2) as suggested by Kroonenberg et al. [12], an approach repeating the simple update three times (Algorithm 3) and an approach, where the simple BR update is repeated until convergence of the eigenvector estimates is reached (Algorithm 4). In addition, we have implemented an advanced algorithm, which is referred to as the full Bauer–Rutishauser algorithm (Algorithm 5) (see Rutishauser [13]). To explore the continuity between the two extremes, i.e., the simple Algorithm 2 and the advanced Algorithm 5, we have added three-step and convergence-based implementations of Algorithm 2. The simple Algorithms 2 and 3 may be regarded as being equal to Algorithm 5, merely with a looser convergence criterion. Three implementations of the Gram–Schmidt orthogonalization for estimating eigenvectors are investigated. Algorithm 6 is a simple one-step GS update as suggested by Kroonenberg [14], Algorithm 7 repeats the simple update three times and Algorithm 8 repeats the simple GS update until convergence is reached. By using repeated iterations better estimates of the true eigensolutions are obtained with a small computational effort, since the working matrices are present in directly accessible forms.

2.2. The Bauer–Rutishauser approach

In [15], Rutishauser proposes an algorithm that improves the convergence order of the bi-iteration method for estimating eigenvectors of matrices suggested by Bauer [16]. Using that $\mathbf{Y}$ $(I \times I)$, e.g., obtained as $\mathbf{XX}^{\mathbf{T}}$, is positive definite and symmetric, the aim of Rutishauser's algorithm is to achieve the good numerical features offered by Bauer's approach with a high convergence order. Rutishauser sets forth several suggestions to improve convergence as well as robustness of the algorithm. For the present purpose, we shall take a less general approach, since we do not require extreme accuracy of the obtained eigensolutions, and we desire to keep the computational requirements at a minimum. Thus, the implementation is kept simple and efficient in Algorithms 2, 3 and 4. Rutishauser's strongest suggestions are implemented in the somewhat more advanced Algorithm 5 for comparative reasons. In Algorithms 2, 3 and 4, new orthogonal iterates of A are provided through one, three or more Ritz-iterations such that the eigendirections, represented by matrix $\mathbf{A}_n$, are defined by the projected eigenvalue problem

$$\mathbf{A}_n^{\mathrm{T}}\mathbf{Y}^{-2}\mathbf{A}_n = \mathbf{D}_n^{-2} \quad \mathbf{A}_n^{\mathrm{T}}\mathbf{A}_n = \mathbf{I} \tag{10}$$

where $\mathbf{D}_n$ $(R^{\mathrm{A}} \times R^{\mathrm{A}})$ is diagonal and holds the eigenvalues of $\mathbf{Y}$ on the diagonal. $\mathbf{A}_n$ is found as

$$\mathbf{A}_n = \mathbf{Y}\mathbf{A}_{n-1}\mathbf{Q}_n\mathbf{D}_n^{-1} \quad \mathbf{Q}_n^{\mathrm{T}}\mathbf{Q}_n = \mathbf{I} \tag{11}$$

$\mathbf{Q}_{\mathrm{n}}$ $(R^{\mathrm{A}} \times R^{\mathrm{A}})$ and $\mathbf{D}_{\mathrm{n}}$ are found by, e.g. an SVD, according to

$$\mathbf{Q}_n\mathbf{D}_n^2\mathbf{Q}_n = \mathbf{A}_{n-1}^{\mathrm{T}}\mathbf{Y}^{\mathrm{T}}\mathbf{Y}\mathbf{A}_{n-1} = \mathbf{A}_{n-1}^{\mathrm{T}}\mathbf{Y}^{2}\mathbf{A}_{n-1} \tag{12}$$

This approach gives results with improved numerical stability and higher convergence rate than the trivial rule $\mathbf{A}_n = \mathbf{Y}\mathbf{A}_{n-1}$ $(n = 1,2,\ldots)$. The reader is referred to Refs. [13,16,17] for details and proofs. The

method may be seen as an extension of the method proposed by Bauer where

$$\mathbf{A}_n = \mathbf{Y}\mathbf{A}_{n-1}\mathbf{R}_n^{-1} \quad (13)$$

and $\mathbf{R}_n$ ($R^A \times R^A$) is an upper triangular matrix with positive diagonal elements which may be derived directly from an extended Gram–Schmidt orthogonalization of $\mathbf{Y}\mathbf{A}_{n-1}$ (see [17] for advanced algorithmic approaches in this direction). One may argue that an algorithm based solely on Eqs. (10)–(12) is overly simplified. Hence, we also implemented a more complete Bauer–Rutishauser algorithm according to some of Rutishauser's many suggestions [13]. In the implementation used here, a series of eigenprojections are calculated with the significant termination by a single Ritz iteration to estimate the orthogonal eigenvectors according to the projected eigenvalue problem.

2.3. The Gram–Schmidt approach

The Gram–Schmidt algorithm may be used for finding an orthonormal basis of any matrix. The orthogonalization is very cheap in terms of operations and it is non-iterative. For the present purpose we have applied a very simple GS algorithm with re-orthogonalization [13,17]. By the repeated eigenprojection of $\mathbf{Y}$ onto $\mathbf{A}_{n-1}$ the enforced response is returned in the new iterate $\mathbf{A}_n$ according to

$$\mathbf{A}_n = \mathbf{Y}\mathbf{A}_{n-1} \qquad (n = 2,3\ldots) \quad (14)$$

However, after applying the eigenprojection several times, the columns of $\mathbf{A}_n$ tend to become correlated, thereby compromising orthogonality. To ensure the condition of the estimated base and to avoid an uncontrolled increase in correlation between the columns of $\mathbf{A}$ during iterations, we suggest to apply the GS orthogonalization continuously. Thus, the resulting sequence takes the form of

$$\mathbf{A}_n = \mathrm{gsm}(\mathbf{Y}\mathbf{A}_{n-1}) \quad (n = 2,3\ldots) \quad (15)$$

where gsm represents the orthogonalization of the matrix argument. To orthogonalize the columns of $\mathbf{Z}$ ($I \times R^A$), assuming that $\mathbf{Z}$ is non-singular, the GS algorithm will return an orthonormal basis in $\mathbf{V}$ according to the following pseudo MATLAB code, in which $\mathbf{V}(:,i)$ designates the ith column of matrix $\mathbf{V}$,

$$\mathbf{V}(:,1) = \mathbf{Z}(:,1)/\|\mathbf{Z}(:,1)\|_2$$

For $i = 2$ to R^A

$$\mathbf{V}(:,i) = \mathbf{Z}(:,i) - \mathbf{V}(:,1{:}(i-1))\mathbf{V}(:,1{:}(i-1))^T \times \mathbf{Z}(:,i)$$

$$\mathbf{V}(:,i) = \mathbf{V}(:,i)/\|\mathbf{V}(:,i)\|_2 \quad (16)$$

end

Some important special variations of the Tucker3 model are: How to incorporate different uncertainties for different elements and how to handle missing elements. We will shortly discuss different ways to approach these special cases.

2.4. Incorporating uncertainties

If the uncertainties of the individual data elements are known, it can be feasible to use these in the decomposition. If the uncertainties are almost equal for all elements, there is no need to change the algorithm, but otherwise at least two different possibilities exist. If the uncertainty of a given variable remains almost the same over all modes, it will suffice to scale the array accordingly, keeping in mind the 'rules' for scaling multi-way arrays (see Kroonenberg [14]). After scaling, an unconstrained model is estimated from the scaled array. If the uncertainties vary also over variables, or if an iteratively reweighted solution is sought for robustness, then one cannot estimate the model using eigenvector-based methods, but has to use regression-based methods or the weighted least squares approach suggested by Kiers [18].

2.5. Missing elements

Missing elements can be effectively handled by the current algorithm by iteratively replacing missing elements with model estimates of the elements. The model is thus estimated from an array with no missing elements, and after each iteration the model of $\mathbf{X}$ is estimated from the parameters. All elements that are missing are replaced with model estimates, and the algorithm is repeated until the convergence criterion is fulfilled *and* the estimates of the missing elements do not change significantly. That way, the missing elements do not directly influence the outcome of the model.

3. Experimental

The aim of this investigation is to find the fastest algorithm among the nine under discussion. The algorithms are compared on the time needed to obtain similar fits to (i) one measured and (ii) several synthesized data sets. For each data set, a model with many factors is estimated in order to ensure that all systematic information is modelled. To facilitate a discussion of the efficiency of the algorithms, we have shown the number of FLOPS (floating operations) required to obtain the solutions. The matrices **A**, **B** and **C** are initiated as previously suggested.

Due to the huge amounts of data handled during iterations, there is a lot of so-called *dead time*. The *dead time* of the algorithms has been estimated by removing the code specifically related to the updating schemes and only keeping the data management operations. By running 20 iterations of this void algorithm, an estimate on the average dead time per iteration, t_0, caused by the size of the data array in question is obtained. After the various algorithms have been applied, the number of iterations, N, is known and the dead time may be subtracted from the total time, T, to give a clearer picture of the time used specifically by the updating schemes. The time used on handling the data is of course required to obtain any solution, but by removing the dead time, the common background is subtracted, thereby allowing us to discuss the sheer time differences caused by the specific updating schemes.

Given the array **X** the error f to minimize is given by

$$f(\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{G};\mathbf{X}) = \|\mathbf{X}^{(1)} - \mathbf{A}\mathbf{G}^{(1)}(\mathbf{B}^{\mathrm{T}} \otimes \mathbf{C}^{\mathrm{T}})\|_2^2 \quad (17)$$

where $\mathbf{X}^{(1)}$ is the frontal slice-wise unfolding **X** and component matrices **A**, **B** and **C** contain the orthogonal factors in their columns. All iterative procedures require a criterion to indicate if a sufficiently accurate model has been estimated. Given the dimensionality of the model, the desire of the analyst is to obtain the lowest possible value of f in the shortest possible time. As stated earlier, we do not calculate f explicitly, but we use the sum of the squared core entries instead, designated by g, as this is obtained with much fewer computations. We seek to maximize the value of g, since the variance (of **X**) described by the model and g will be at maximum for the same set of **A**, **B** and **C** (rotation disregarded).

For checking convergence, we have taken one approach for the real data set and another for the 200 synthesized data sets. For the real data set, a minimum value of the sum of squared core entries, referred to as g^*, will be used as stopping criterion. The value of g^* has been set slightly below the asymptotic value of g, which was found by inspection. With regards to the numerous synthesized data sets, an unsupervised criterion to detect convergence is required, and for the present application, this is formulated as the maximum difference of g between two successive iterations. We will return to this later.

3.1. Data

The measured data set originates from spectrofluorometric measurements on 65 samples, and has dimensions $65 \times 40 \times 311$ representing approximately 6.5 MB of data. The values range from zero to approximately 1295. Investigations not reported here have revealed that the rank is in the range 6 to 8. Hence, a model of order $8 \times 8 \times 8$ is estimated. By inspection and evaluation of different solutions, the value of g^* for this data set was set to $3.720345874925 \cdot 10^{10}$. Using this value of g^* in all algorithms applied to this data set, the fit of the models will be comparable from the viewpoint of the analyst.

A number of 200 synthesized data sets of dimensions $120 \times 120 \times 120$ with PARAFAC rank not less than 8 are produced by synthesizing factors from Gaussian peaks in each of the three modes with randomly distributed peak centres and peak widths, and subsequently applying 5% homoscedastic (additive) and 5% heteroscedastic (multiplicative) normally distributed noise. With regards to spectral data, the resulting level of noise may be regarded as being high. To ensure the rank, the peak centres were forced to differ in locations within modes and the synthesized cores consisted of random values between 0 and 1, where the diagonal elements (1,1,1), (2,2,2),...,(8,8,8) were forced to be 1. The convergence criterion was estimated for each synthesized data array in the following way: The SVD-based algorithm (Algorithm 1) was applied with the criterion that convergence was reached when two successive fits differed by less than 0.0005. If the algorithm

honoured this convergence criterion within 19 steps, the convergence criterion was divided by two, and the SVD-algorithm was restarted; this was repeated until the number of required steps exceeded 20. This feasible convergence criterion was then used for the remaining 8 algorithms. We estimate that at least 20 iterations are required to get accurate measurements on the computational time. Since very slow convergence of the most approximate algorithms (i.e., small changes between iterations in Algorithms 2 and 6) could erroneously cause the algorithms to exit too early, a subsequent evaluation of the errors of the final models was performed to reject models that did not fit data satisfactorily.

4. Results and discussion

The results from the models of the measured data set are listed in Table 1. Standard deviations are negligible and are not listed. It is readily seen that Algorithm 9 stands out with the lowest time consumption, T, and the highest efficiency in terms of FLOPS. This is likely due to the simplicity of the NIPALS algorithm, which is used to estimate eigenvectors in each of the three subproblems in this algorithm. Another important observation is the high number of required main iterations, N, for Algorithms 2 and 6. These two algorithms have a high degree of simplicity in common, but whereas this was intended to decrease the time consumption of the subproblems, the increase in the overall number of main iterations renders these approaches infeasible. Algorithms 2 and 6 provide too inaccurate approximations to the eigensolutions; hence, they require more iterations to reach the exact eigensolutions. If the eigensolutions are inaccurately determined in one iteration, then the next iteration will suffer from this suboptimality in the posed problem. This is in contrast to the experiences reported by Kroonenberg [14] (p. 87), where he argues that it is not worthwhile to solve for highly accurate eigenvectors since the resulting algorithm will obtain an iteration-in-iteration structure requiring too much computational effort, since after all, during iterations, the eigenproblems posed are only formulated in terms of intermediate factors. Whereas this may be true for smaller sized problems, the compromise between spending computational time on estimating accurate eigensolutions and the overhead introduced by handling the large data arrays, appears to favour the updating schemes that are more accurate. From the viewpoint of the analyst Algorithms 2 and 6 are suboptimal in terms of FLOPS as well as time; thus, we will leave them out of the remaining discussion. With regards to time consumption, T, we see that Algorithms 1 and 4 use markedly more time on the same number of iterations. By inspection of T_c, we conclude that this is due to the time used in the updating schemes. Algorithm 1 includes an SVD which is stable and accurate, but very time consuming. So, in addition to the conclusions drawn from the very simple algorithms (i.e., that the eigenvectors must be accurate) it is indicated that there *is* an upper limit to the effort that should be used on improving the accuracy of the eigensolutions. When compared to Algo-

Table 1
Results from models of the measured data set

Algorithm number	Flops (10^9)	Number of iterations (N)	Time, T (s)	Corr. time ($T = T - N\,t_0$) (s)	Final value of $g\,g(N)\,10^{10}$
1	1.46	21	63.54	28.03	3.72034587493
2	1.75	35	74.40	15.22	3.72034587493
3	1.33	22	52.56	15.36	3.72034587493
4	1.41	21	54.02	18.51	3.72034587493
5	1.15	21	49.18	13.67	3.72034587493
6	1.64	35	71.48	12.30	3.72034587493
7	1.10	22	46.96	9.76	3.72034587493
8	1.10	21	47.78	12.27	3.72034587493
9	0.81	21	42.54	7.03	3.72034587493

All values of required number of FLOPS, number of iterations (N), computation time (T), and corrected time (T_c) are averages of 20 model runs.
$t_0 = 1.69$ s it^{-1}.

rithm 3, Algorithm 4 appears to iterate too many times in the substeps judged from the value of T_c. This may be corrected by reducing the number of inner iterations, e.g., by adjusting the convergence criteria for the subiterations. Algorithms 5, 7 and 8 offers almost similar performance in terms of FLOPS and time consumption. In line with the findings of Kroonenberg et al. [12], the efficiency of the GS approaches, especially Algorithms 7 and 8, are certainly of interest. With regards to repeating the substeps, it holds for BR and GS that the three-step approaches significantly reduces the *overall* time needed to reach a solution. Since repeated application of the GS update improves the accuracy of the estimated eigensolutions, we attribute the decrease of iterations to the increased adequacy of the subsequently posed eigenproblems. Comparing columns four and three clearly shows that for all nine algorithms, most time is spent on handling data and *not* on solving the eigenproblems. Hence, improving the speed of data handling will contribute significantly in reducing the time consumption. The sum of squared core elements in the last iterations, $g(N)$, in Table 1 verifies that the final models are actually comparable in fit. Thus, the results from the measured data set suggest that Algorithms 5, 7, 8 or 9 are fastest, with Algorithm 9 being fastest for this data set.

The findings from the analysis of the 200 synthesized data sets are listed in Table 2. Since the synthesized data arrays are very different, we have listed the standard deviation next to the parameters. It is immediately recognized that the observed standard deviations are high, thereby rendering interpretation difficult. This is mainly due to the very different properties of the data sets and not a matter of great concern. However, the pattern found in the mean values for T are verified by the fact that Algorithms 3, 7 and 9 are fastest in 41 (21%), 63 (32%) and 84 (42%) of the 200 models. Since the number of required inner iterations for all updates depends strongly on the size and the characteristics of the data under investigation, we investigated the correlation coefficients and the condition number of the synthesized factor matrices. Over all three modes, the synthesized factor matrices had absolute correlation coefficients ranging from 0.1319 to 0.8235 with a mean value of 0.5471 and a S.D. at 0.2852. Ranging from $2.01 \cdot 10^1$ to $3.19 \cdot 10^6$ the condition numbers (i.e., the ratio between the largest eigenvalue and the lowest) was found to have a mean value at $3.69 \cdot 10^4$ with a S.D. at $1.80 \cdot 10^5$. Based on these findings, we may conclude that the synthesized data sets were constructed from factors that were somewhat correlated, thereby introducing ill-posed subproblems. With regards to the time spent on the updating schemes, T_c, Algorithms 2 and 6 were fastest, but the eigensolutions provided within iterations were too simple and too inaccurate. This is in accordance with the findings from the measured data set. Thus, they (consistently) required the highest number of iterations. We consolidate the findings from the analysis of the measured data set; algorithms for analysis of large data arrays

Table 2
Mean values and standard deviations of required number of FLOPS, iterations (N), computation time (T), and corrected time (T_c) from 200 synthesized data sets

Algorithm number	FLOPS (10^9)		Iterations (N)		Time, T (sec)		Corr. time, T_c (s)	
	Mean	S.D.	Mean	S.D.	Mean	S.D.	Mean	S.D.
1	2.48	0.80	28.80	9.29	107.54	35.59	24.45	7.89
2	2.43	0.81	35.33	11.66	107.08	35.62	5.15	1.82
3	2.21	0.65	30.20	8.88	93.99	27.82	6.86	2.09
4	2.86	0.86	28.80	9.30	103.68	32.13	20.59	5.59
5	2.24	0.67	29.02	8.82	100.41	30.03	16.70	4.62
6	2.38	0.79	35.33	11.70	106.61	35.46	4.68	1.67
7	2.07	0.61	30.20	8.91	92.94	27.49	5.81	1.78
8	2.23	0.70	28.80	9.34	98.93	30.90	15.84	4.34
9	1.92	0.61	28.81	9.22	90.75	28.58	7.65	2.28

Compare with Fig. 1.
For the synthesized data sets t_0 is 2.89 s it^{-1}.

must be based on fast, but accurate, algorithms for estimating eigensolutions of the involved subproblems. It should be noted that Algorithm 4 (convergence-based BR) and 8 (convergence-based GS) required the same number of outer iterations as Algorithm 1 (based on SVD) for all data sets, verifying that these three algorithms provide the same accurate eigensolutions. In accordance with Table 1, Algorithms 7 and 9 require less time and less FLOPS to reach a solution. To conclude, we find that Algorithm 9 offer the best combination of simplicity and accuracy of the eigensolutions of the synthesized data arrays.

Measurements of number of FLOPS, iterations N, total computing time T, and corrected time T_c, were arranged as matrices of dimensions 200 (data sets) × 9 (algorithms), one matrix for each parameter. To illustrate the significant covariations of the parameters, we have extracted one principal component from each of the four matrices. The superposed factors for the nine algorithms in Fig. 1 are scaled such that the largest element in each factor has a value of one. The figure illustrates the essence of this investigation. The efficiency of the updating schemes in Algorithms 3, 7 and 9 are evident from the simultaneous low levels of FLOPS, iterations (N) and the total time required

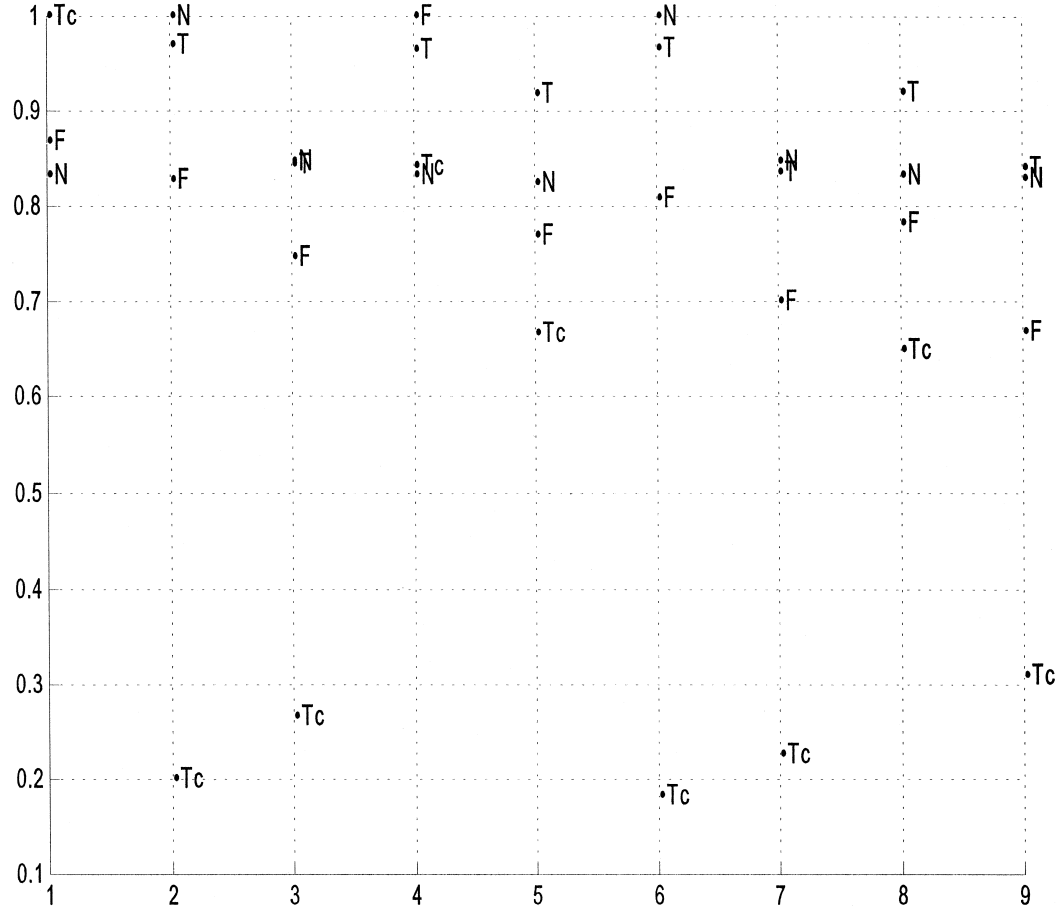

Fig. 1. Results from the models of 200 synthesized data sets with respect to the nine applied algorithms. The four superposed principal components represent the number of FLOPS (F), iterations (N), the total computing time (T), and corrected time (T_c). Compare with Table 2. The three factors explain 76.57%, 76.50%, 76.47%, and 80.38% of the variation in the matrices. The factors are scaled such that the largest element has a value of one.

(T) as seen from Fig. 1. We notice that the simple NIPALS algorithm (Algorithm 9), well known and widely used in chemometrics, substantiates itself as an excellent compromise between speed and accuracy. The huge gain in convergence when comparing single-step implementations (Algorithms 2 and 6) to the three-step implementations (Algorithms 3 and 7) is substantiated. The convergence-based iterations (Algorithms 4 and 8) are sensitive towards the threshold of the convergence criteria, and the optimal convergence criterion may depend on the data at hand.

5. Conclusion

We have compared nine algorithms for solving the Tucker3 model on very large data arrays. Through modelling of one measured and several synthesized data sets especially the NIPALS-based implementation appears to be feasible with regards to time consumption and FLOPS. The implementations based on three repeated simple Gram–Schmidt updates are suggested as alternative algorithms. Furthermore, we have found that accuracy, perhaps more than speed, is required in implementations of Tucker3 models of large data arrays to yield results in the shortest possible time.

Acknowledgements

Prof. Lars Munck is thanked for providing psychological support for what must be the most exploratory industry-related research group. The authors further wish to thank Professor Lars Munck for financial support through the Nordic Industry Foundation project P93149 and the FØTEK foundation. We also wish to thank Henk Kiers, Sijmen de Jong and Age Smilde and two anonymous referees for good comments on this paper.

References

[1] L. Tucker, Some mathematical notes on three-mode factor analysis, Psychometrika 31 (1966) 279–311.
[2] P.M. Kroonenberg, J. de Leeuw, Principal components analysis of three-mode data by means of alternating least squares algorithms, Psychometrika 45 (1980) 69–97.
[3] J.M.F. ten Berge, J. de Leeuw, P.M. Kroonenberg, Some additional results on principal components analysis of three-mode data by means of alternating least squares, Psychometrika 52 (1987) 183–191.
[4] C.L. de Ligny, M. Spanjer, J.C. van Houwelingen, H.M. Weesie, Three-mode factor analysis of data on retention in normal-phase high-performance liquid chromatography, J. Chromatogr. 301 (1984) 311–324.
[5] P.J. Gemperline, K.H. Miller, T. West, E. Weinstein, J.C. Hamilton, J.T. Bray, Principal component analysis, trace elements and blue crab shell disease, Anal. Chem. 64 (1992) 523–532.
[6] W.A. van der Kloot, P.M. Kroonenberg, External analysis with three-mode principal component analysis, Psychometrika 50 (1985) 479–494.
[7] A. Kapteyn, H. Neudecker, T. Wansbeek, An approach to *n*-mode components analysis, Psychometrika 91 (1986) 269–275.
[8] B.K. Alsberg, O.M. Kvalheim, Speed improvement of multivariate algorithms by the method of postponed basis matrix multiplications: Part I. Principal component analysis, Chemometr. Intell. Lab. Syst. 24 (1994) 31–42.
[9] B.K. Alsberg, O.M. Kvalheim, Speed improvement of multivariate algorithms by the method of postponed basis matrix multiplications: Part II. Three-mode principal component analysis, Chemometr. Intell. Lab. Syst. 24 (1994) 43–54.
[10] H. Kiers, R.A. Harshman, Relating two proposed methods for speedup of algorithms for fitting two- and three-way principal component and related multilinear methods, Chemometr. Intell. Lab. Syst. 39 (1997) 31–40.
[11] H.A.L. Kiers, P.M. Kroonenberg, J.M.T. ten Berge, An efficient algorithm for TUCKALS on data with large number of observation units, Psychometrika 33 (1992) 415–422.
[12] P.M. Kroonenberg, J.M.F. ten Berge, P. Brouwer, H. Kiers, Gram-Schmidt versus Bauer–Rutishauser in alternating least squares algorithms for three-mode principal component analysis, Comp. Stat. Q. 2 (1989) 81–87.
[13] H. Rutishauser, F.L. Computational aspects of, Bauer's simultaneous iteration method, Numer. Math. 13 (1969) 4–13.
[14] P.M. Kroonenberg, Three-mode principal component analysis, DSWO Press, Leiden, 1983.
[15] R. Bro, C.A. Andersson, Improving the speed of multi-way algorithms Part II: Compression, Chemometr. Intell. Lab. Syst. 42 (1998) 103–111.
[16] F.L. Bauer, Das Verfahren der Treppeniterationen und verwandte Verfahren zur Lösung algebraischer Eigenwertprobleme, ZAMP 8 (1957) 214–235.
[17] J.W. Longley, Least squares computations using orthogonalization methods, Lecture Notes in Pure and Applied Mathematics, Marcel Dekker, New York, 1984.
[18] H.A.L. Kiers, Weighted least squares using ordinary least squares algorithms, Psychometrika 62 (1997) in press.