ELSEVIER

Chemometrics and Intelligent Laboratory Systems 38 (1997) 223–242

Chemometrics and intelligent laboratory systems

# A weighted non-negative least squares algorithm for three-way 'PARAFAC' factor analysis

Pentti Paatero *

*Department of Physics, University of Helsinki, P.O. Box 9, FIN-00014 Helsinki, Finland*

Received 29 July 1996; revised 14 September 1996; accepted 8 March 1997

**Abstract**

A time-efficient algorithm PMF3 is presented for solving the three-way PARAFAC (CANDECOMP) factor analytic model. In contrast to the usual alternating least squares, the PMF3 algorithm computes changes to all three modes simultaneously. This typically leads to convergence in 40–100 iteration steps. The equations of the weighted multilinear least squares fit are given. The optional non-negativity is achieved by imposing a logarithmic penalty function. The algorithm contains a possibility for dynamical reweighting of the data during the iteration, allowing a robust analysis of outlier-containing data. The problems typical of PARAFAC models are discussed (but not solved): multiple local solutions, degenerate solutions, non-identifiable solutions. The question of how to verify the solution is discussed at length. The program PMF3 is available for 486-Pentium based PC computers.

*Keywords:* Positive matrix factorization; Multilinear models; PARAFAC; CANDECOMP; Multiple solutions; Degeneracy

## 1. Introduction

Three-way data arrays need to be analyzed in most diverse areas of research, ranging from physical sciences, such as chemistry or astronomy, to social sciences like psychology. Much of this analysis is based on multilinear models: the measured array is to be approximated by an array functional $Y(A, B, \dots)$. The arguments of $Y(\dots)$ are matrices (or vectors). The dependence of $Y$ on its arguments is multilinear, meaning that when considering the dependence of $Y$ on a single argument, this dependence is linear.

A good coverage of current research problems in the multilinear field is presented by Coppi and Bolasco [1]. The most basic models are called 'multiway factor analysis', meaning that each element of $Y$ is represented by a (multiple) sum of products of elements of the matrices $A, B, \dots$; these matrices are called *factor matrices.* In different applications different properties of the factor matrices are emphasized: in physical sciences they are often required to be non-negative in order to achieve direct correspondence with the measurable non-negative quantities. The other extreme is that the columns of the factor matrices (= *the factors*) may be required to be orthogonal to each other.

* E-mail: pentti.paatero@helsinki.fi.

Most of the research in multiway models is performed with three-way data arrays. A serious problem has been the lack of efficient algorithms. Quoting from the last page of the mentioned book (Wansbeek and Verhees [2]) "*Deriving succinct formulas is one thing, but programming these is something else. Programming the formulas at face value is of course possible but extremely tedious, and the resulting programs are time and space inefficient.*" Because of such inefficiencies, much of the research of 3-way models has adopted roundabout techniques: the 3-way problem is reduced to a set of two-way problems. This makes computations fast but may obscure the essential properties of the model in question and prevent straightforward interpretation of the results. The present work attempts to alleviate the inefficiency problem by presenting a time-efficient algorithm which enables the research to be performed directly within the 3-way framework.

The 3-way $p$-component PARAFAC model is defined by

$$\mathbf{X}_{ijk} = \sum_{h=1}^{p} \mathbf{A}_{ih}\mathbf{B}_{jh}\mathbf{C}_{kh} + \mathbf{E}_{ijk} = \mathbf{Y}_{ijk} + \mathbf{E}_{ijk} \quad (i=1, \ldots, m, j=1, \ldots, n, k=1, \ldots, o). \tag{1}$$

Here $\mathbf{X}$ is the observed 3-way data array and $\mathbf{E}$ the corresponding array of residual values. $\mathbf{Y}$ is the fitted array. The matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are the unknown 'factor matrices'. In the original definition of PARAFAC by Harshman and Lundy [3], the object function

$$Q(\mathbf{E}) = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{o} \mathbf{E}_{ijk}^2 \tag{2}$$

was minimized without regard to non-negativity of the unknown variables $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$. In this work an efficient algorithm is described for solving the weighted PARAFAC problem. The weighted object function $Q$ is defined by

$$Q(\mathbf{E}) = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{o} w_{ijk}\mathbf{E}_{ijk}^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{o} (\mathbf{E}_{ijk}/\sigma_{ijk})^2. \tag{3}$$

The weighted object function $Q$ is minimized under the constraint that all elements of the three factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ must be non-negative. This model has originally been solved by Ross and Leurgans [4]. However, the present algorithm is such that the non-negativity criterion may also be dropped for such variables which may need to assume negative values.

The current approach is possible both for 3-way models and for 2-way models, i.e. 'factor analysis'. The name 'positive matrix factorization' (PMF) has been used to denote the non-negatively constrained weighted least squares approach by Paatero and Tapper [5]. The existing programs for 2-way and 3-way models are called PMF2 and PMF3, respectively.

### *1.1. Connections with previous work*

Solving the PARAFAC model may be viewed as a general optimization task or as a general curve fitting task. In the first case a general optimization program could be used so that the program just repeatedly calls a black box routine for computing the object function $Q$. In the second case the program 'knows' that the function $Q$ has the form of a sum-of-squares of differences between measured and fitted data; the routine for computing the fitted values (and possibly their derivatives) is again called in the black box fashion. The performance of a general curve fitting routine is expected to be better than that of a general optimizing routine when performing curve fitting tasks. It has been mentioned in conference discussions that it would be useful to apply general optimization/curve fitting tools to PARAFAC problems. When developing the alternating least squares (ALS) solution of the PARAFAC problem, Harshman also experimented with an optimization approach but at that time this approach was too slow and was abandoned [6].

From the curve fitting viewpoint all the factor values $\mathbf{A}_{ih}$, $\mathbf{B}_{jh}$, and $\mathbf{C}_{kh}$ are the parameters of the model while the $\mathbf{X}_{ijk}$ form the 'curve' to be fitted with $\mathbf{Y}_{ijk}$. The PARAFAC model has the special property that each fitted value $\mathbf{Y}_{ijk}$ depends only on a small number $(3p)$ of the parameters (see Eq. (1)). It seems likely that any general-purpose curve fitter which presumes the dependence of each fitted value on each parameter is bound to be quite inefficient. The present work may be viewed as a specialized curve fitting algorithm which applies the well-known principles of curve fitting while utilizing the special structure of the PARAFAC model.

Some optimizing/curve fitting algorithms minimize the object function first in one fixed subspace, then in another, and so on until all directions in the total space of the parameters have been examined; this scanning of subspaces is repeated until a convergence has been found. The simplest variant is that each parameter (each coordinate direction) is one search subspace. It is well known that such techniques usually converge slowly; although the algorithm is simple, they are not often used. ALS belongs to this class of algorithms. In one scan of the parameter space, it minimizes $Q$ first in each one of the $m$ subspaces spanned by the $m$ rows of the matrix $\mathbf{A}$, then in the $n$ subspaces spanned by $\mathbf{B}$ and finally in the $o$ subspaces spanned by $\mathbf{C}$; altogether $(m+n+o)$ minimizations. (Alternatively one may view it as performing minimizations in only 3 subspaces, the first is spanned by all elements of $\mathbf{A}$, the second and third by elements of $\mathbf{B}$ and $\mathbf{C}$.) The special advantage of ALS is that in each subspace fit, the fitted values $\mathbf{Y}_{ijk}$ are *linear* functions of the parameters, hence the fit is easy to compute as a weighted or unweighted regression task. In accordance with the general optimization experience, the 3-way ALS often converges slowly, requiring up to tens of thousands of steps for good convergence. Appendix A presents a simple example which clearly demonstrates the slowness of the ALS algorithm.

### 1.2. The weights

If the standard deviations $\sigma_{ijk}$ of data values are known, then optimum least squares (LS) weights for each value $\mathbf{X}_{ijk}$ are $w_{ijk} = 1/\sigma_{ijk}^2$, as already indicated in Eq. (3). There are important connections between physical reality and the values $\sigma_{ijk}$, especially in environmental models. These connections have been discussed earlier by Paatero and Tapper [5] and this discussion is not repeated. A robust approach is possible so that the weights are adjusted dynamically during the iterative computation of the LS fit. This technique has been described for two-way factor models by Paatero [7]. The same technique has also been applied successfully for 3-way models. In this work it is simply noted that depending on the application, the weights may be user-specified, they may be computed from known standard deviations according to Eq. (3), or they may be dynamically recomputed during computations.

### 1.3. Preprocessing the array: Centering and scaling

Preprocessing has been reviewed by ten Berge [8]. It appears that centering is really a part of the description of the problem to be solved. It is part of setting up the model. Thus it is not discussed in the present work which concentrates on how to solve the model. Scaling the array means multiplying the slices of the array by suitable values; the slices may be taken in all three directions. In contrast to centering, scaling may affect both the model and the way of solving it. In the original unweighted PARAFAC model, there is the implicit assumption that all the values $\sigma_{ijk}$ are equal. Performing the row/column/page scaling of the array in the unweighted model is equivalent to specifying a rank-1 array $\sigma_{ijk}$ of the standard deviations in the weighted model. This question has been discussed by Paatero and Tapper [9] for the two-way model but the results are also valid for $n$-ways.

When solving the unweighted PARAFAC model one obtains different results if the array is differently scaled. As an example, consider an array which represents concentrations of different elements in environmental samples. One may express all concentrations in percent and obtain a fit with the unweighted PARAFAC. A different unweighted fit is obtained, however, if trace element concentrations are expressed in ppm. In contrast, in the weighted Eq. (3) different scalings only effect proportional changes in the numerator and in the denominator; these changes cancel each other. The same weighted fit would be obtained in the example irrespective of the units used for trace element concentrations. Thus scaling of the array is irrelevant for the basic weighted PARAFAC model. However, later it will be seen that scaling has to be taken into account when specifying the penalty functions.

## 2. The goals and the methods in this work

The following list enumerates problems that may occur when a PARAFAC model is solved with the ALS algorithm:

1. Local minima may occur.
2. It is difficult to know when the process has fully converged.
3. There is no satisfying way of quantifying the goodness of the fitted model.
4. There are occasions ('swamps') when the ALS algorithm advances extremely slowly (see Mitchell and Burdick [10]).
5. The results may depend on the starting point of the iteration.
6. The number of iterations needed for convergence may depend strongly on the starting point of the iteration; in the majority of all computed cases the ALS algorithm is slow.

As presented in Section 1, this work introduces a new algorithm for solving essentially the same PARAFAC model which has been solved by earlier researchers. The emphasis is on getting the same solutions as before but getting them much faster than before. The PARAFAC model defines *what* is to be computed: a certain expression is to be minimized with respect to the unknown factor values. All algorithms that solve the model correctly should in principle produce the same results. The algorithm specifies *how* the result is computed; different algorithms may differ in speed, memory requirements, and ease of use but they should produce the same results. Sometimes these concepts are not kept separated. Confusion may result if e.g. the term ALS is used in the meaning of 'a multilinear model' when in fact ALS is just one technique for solving a multilinear model.

Problem 3 in the list above concerns the fitted model. Thus no progress is expected with problem 3 when changing the technique of solving the model. Problem 1 describes a property of the mathematical PARAFAC model. Both ALS and PMF3 are local techniques that attempt to find a direction in the many-dimensional space so that the current solution is improved when proceeding in this direction. Both techniques minimize the object function in the local sense: the three factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ define a (local) minimum of $Q$ if there are no such increment matrices $\Delta\mathbf{A}$, $\Delta\mathbf{B}$, and $\Delta\mathbf{C}$ that the expression $Q(X, \mathbf{A} + \lambda\Delta\mathbf{A}, \mathbf{B} + \lambda\Delta\mathbf{B}, \mathbf{C} + \lambda\Delta\mathbf{C})$ decreases when $\lambda$ increases from zero. Introduction of PMF3 will not influence the question of local minima. It appears that currently no suitable algorithm exists that would be certain to find the global minimum in a reasonable time.

Problem 5 is two-sided, it will be discussed further subsequently. If there are local solutions or if the model is indeterminate (defined later), then it is natural that the results depend on the starting point. Both ALS and PMF3 have this dependence. However, if there is no such indeterminacy in the model then there should not be any dependence on the initial point and none is observed with PMF3. If it is observed with ALS, then it probably depends on the slow convergence which causes that the iteration is interrupted prematurely.

Problems 2, 4, and 6 are related to the algorithm and improvement is possible here. Indeed, with PMF3 there have been no problems in convergence testing. In Appendix B a simple test is given which has proved quite adequate. The question of swamps has not been studied yet in connection with PMF3. The variability of the number of iteration steps as a function of the starting point is also seen with PMF3, the same problem may need 20 or 80 steps depending on the starting point. The main goal of improving on the slowness of ALS has been achieved, however. The number of iteration steps is typically 40, and hardly ever exceeds 150.

The PARAFAC model being solved with PMF3 is in fact not exactly the same as in earlier research: there is a regularization term in the object function (see below). This term influences the results obtained for indeterminate models. Thus PMF3 with regularization may produce different results than ALS without the regularization term (the regularization may be easily added to PARAFAC in connection with ALS if desired). The regularization may also be the reason that swamps have not been seen with PMF3, but this is an open question.

The PMF3 algorithm produces an additional result which is not possible with ALS: the joint covariance matrix of all factor elements. Future research may perhaps use this tool for addressing problem 3.

### *2.1. On the optimality of the choices made when developing the PMF3*

This paper reports on the current algorithm PMF3. Many choices had to be made when developing the algorithm. The reader would like to have proofs or at least heuristic arguments for supporting these choices. In some cases no such information is available. It is quite possible that the choices are not optimal or that there are several equally good choices. An attempt is made in the following presentation to indicate the arguments supporting the choices. It should be understood that these arguments cannot be conclusive. The merits of alternative choices can only be ascertained by writing a program based on the alternative choices.

### *2.2. How to publish an algorithm*

The scientific paradigm is to publish so that the reader is able to replicate the published work and verify the results. This is feasible when dealing with simple algorithms such as matrix inversion. The present work, however, is an example of a specialized large-scale optimizing software. It is not possible to publish such software in the 'cookbook recipe' fashion so that the reader would be able to type the code in his/her computer and be ready to run. When implementing the algorithm, a huge number of small but crucial practical details must be considered and it is not possible to include all that detail in a scientific paper. Experience in writing or maintaining optimizing software is necessary in such work. The scientific paper may only provide guidance for the reader in how to write his/her own program. Most important might be that the paper provides motivation as to why to use the new approach and what is better in the new technique. The question of verifying the results will be discussed later in this paper.

In Appendix B a pseudocode version of the program PMF3 is given. It is intended to give a coherent view of what is going on in the program and contains some simplifications in comparison to the real code.

### *2.3. The workload*

Coarse estimates of workload are given for parts of the algorithm. These values are asymptotic, they are only meaningful for large problems. One unit of workload (= one 'operation') comprises one multiplication, one addition and the necessary indexing computations. The actual computing time varies according to programming techniques, compiler, size of data array, etc. For a Pentium-based PC computer, one work unit may correspond to anything between 0.1 and 2 $\mu$s.

## 3. Summary of the Gauss–Newton approach for solving a least squares problem with penalty terms

In this section a concise summary of the Gauss–Newton approach is given so that the main presentation remains more coherent. For simplicity, it is assumed here that $\mathbf{X}_i$ $(i = 1, \ldots, m)$ is a vector of measured values. The vector of unknowns, to be determined, is denoted by $\boldsymbol{\Theta}_j$ $(j = 1, \ldots, n)$. There is a specified model which defines each one of the fitted values as a known function of all of the unknowns: $\mathbf{Y}_i = \mathbf{Y}_i(\boldsymbol{\Theta})$ $(i = 1, \ldots, m)$. This model need not be linear. In practical applications, it is possible that each fitted value $\mathbf{Y}_i$ only depends on a few of the parameters $\boldsymbol{\Theta}_j$ but the conceptual model assumes a full dependence.

The LS curve fitting task is defined by the object function $Q$, to be minimized, which is quadratic with respect to the residuals:

$$Q = \sum_{i=1}^{m} w_i (\mathbf{Y}_i(\boldsymbol{\Theta}) - \mathbf{X}_i)^2. \tag{4}$$

The current approximation of $\boldsymbol{\Theta}$ is denoted by $\boldsymbol{\Theta}^0$ and the corresponding fitted values by $\mathbf{Y}_i^0$. In the Gauss–Newton curve fitting technique the exact $Q$ defined in Eq. (4) is replaced by an approximate expression where $\mathbf{Y}$ has been linearized around the current point $\boldsymbol{\Theta}^0$:

$$\begin{aligned} Q^{\mathrm{GN}} &= \sum_{i=1}^{m} w_i \left( \mathbf{Y}_i^0 + \sum_{j=1}^{n} \frac{\partial \mathbf{Y}_i}{\partial \boldsymbol{\Theta}_j} (\boldsymbol{\Theta}_j - \boldsymbol{\Theta}_j^0) - \mathbf{X}_i \right)^2 = \sum_{i=1}^{m} w_i \left( \mathbf{Y}_i^0 + \sum_{j=1}^{n} \mathbf{J}_{ij} (\boldsymbol{\Theta}_j - \boldsymbol{\Theta}_j^0) - \mathbf{X}_i \right)^2 \\ &= (\mathbf{Y}^0 + \mathbf{J}(\boldsymbol{\Theta} - \boldsymbol{\Theta}^0) - \mathbf{X})^{\mathrm{T}} \mathbf{W} (\mathbf{Y}^0 + \mathbf{J}(\boldsymbol{\Theta} - \boldsymbol{\Theta}^0) - \mathbf{X}) = (\mathbf{J}\Delta\boldsymbol{\Theta} - \mathbf{E})^{\mathrm{T}} \mathbf{W} (\mathbf{J}\Delta\boldsymbol{\Theta} - \mathbf{E}). \end{aligned} \tag{5}$$

Here, $\mathbf{W}$ denotes the diagonal matrix of weights $w_i$ and $\mathbf{J}$ is the Jacobian matrix, consisting of the partial derivatives of the fitted values $\mathbf{Y}_i$ with respect to the parameters $\boldsymbol{\Theta}_j$. The current residual is $\mathbf{E} = \mathbf{X} - \mathbf{Y}^0$. The Jacobian contains one row corresponding to each data point (corresponding to each equation to be satisfied) and one column corresponding to each element in the vector of unknowns.

Minimizing $Q^{\mathrm{GN}}$ with respect to $\Delta\boldsymbol{\Theta}$ leads to the system of 'normal equations'

$$\mathbf{J}^{\mathrm{T}} \mathbf{W} \mathbf{J} \Delta\boldsymbol{\Theta} = \mathbf{J}^{\mathrm{T}} \mathbf{W} \mathbf{E}. \tag{6}$$

The unknowns $\Delta\boldsymbol{\Theta}$ may be solved e.g. by Cholesky decomposition and back substitution; usually it is better not to compute the inverse matrix $(\mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J})^{-1}$ unless it is specifically needed.

Often one wishes to influence the fit by adding a penalty part $Q^{\mathrm{P}}$ into the object function. If this penalty is formulated in the form of a sum-of-squares, then the Gauss–Newton approach can be applied to the enhanced object function $Q = Q^{\mathrm{GN}} + Q^{\mathrm{P}}$. One simple example for the penalty is

$$Q^P = \sum_{j=1}^{n} \lambda_j (\boldsymbol{\Theta}_j - \boldsymbol{\Theta}_j^1)^2. \tag{7}$$

The penalty of Eq. (7) presents the a priori information that the value of each unknown parameter $\boldsymbol{\Theta}_j$ should be close to the corresponding target value $\boldsymbol{\Theta}_j^1$. Often the target values are equal to zero. Each weight $\lambda_j$ corresponds to the importance assigned to this a priori information. The penalty may be written in a similar form as Eq. (5):

$$Q^P = (\Delta\boldsymbol{\Theta} - \boldsymbol{\Theta}^*)^{\mathrm{T}} \boldsymbol{\Lambda} (\Delta\boldsymbol{\Theta} - \boldsymbol{\Theta}^*). \tag{8}$$

The difference between the target value and the current value of $\boldsymbol{\Theta}$ is the vector $\boldsymbol{\Theta}^* = \boldsymbol{\Theta}^1 - \boldsymbol{\Theta}^0$. The diagonal matrix $\boldsymbol{\Lambda}$ consists of the penalty weights $\lambda_j$. The enhanced $Q$ is thus

$$Q = (\mathbf{J}\Delta\boldsymbol{\Theta} - \mathbf{E})^{\mathrm{T}} \mathbf{W} (\mathbf{J}\Delta\boldsymbol{\Theta} - \mathbf{E}) + (\Delta\boldsymbol{\Theta} - \boldsymbol{\Theta}^*)^{\mathrm{T}} \boldsymbol{\Lambda} (\Delta\boldsymbol{\Theta} - \boldsymbol{\Theta}^*). \tag{9}$$

The normal equations of the enhanced $Q$ are obtained when this expression is minimized with respect to the unknowns $\Delta\boldsymbol{\Theta}$:

$$(\mathbf{J}^{\mathrm{T}} \mathbf{W} \mathbf{J} + \boldsymbol{\Lambda}) \Delta\boldsymbol{\Theta} = \mathbf{J}^{\mathrm{T}} \mathbf{W} \mathbf{E} + \boldsymbol{\Lambda} \boldsymbol{\Theta}^*. \tag{10}$$

It is seen that the addition of penalty terms into $Q$ adds to the diagonal of the matrix of the normal equations and also adds to the right-side vector. (If regularization is desired, then the target vector is zero and the right-side addition is $= -\boldsymbol{\Lambda}\boldsymbol{\Theta}^0$.) If there are several penalty and/or regularization terms in the enhanced object function, then there are correspondingly more terms in the two sums in Eq. (10).

Crucial for the application of the Gauss–Newton approach is that the penalty be expressed in the form of a sum-of-squares. Other forms of penalty (e.g. sums of other powers of the parameters $\Theta_j$) would need to be approximated by a quadratic functional around the current solution point.

## 4. The enhanced object function Q of the PARAFAC model

Three well-known indeterminacies are connected with the PARAFAC model. First there is the permutation indeterminacy: the columns of all three factor matrices may be similarly permuted without influencing the fit. It is necessary that the starting point of an iterative least squares PARAFAC algorithm is non-degenerate so that all factors differ from each other. Then the permutation is already fixed and the permutational indeterminacy does not influence the computation of the increment matrices.

The second indeterminacy concerns factor scaling: two triples such as (**A**, **B**, **C**) and ($r$**A**, **B**$/r$, **C**) offer the same LS fit. Hence the problem of determining the unknown matrices **A**, **B**, and **C** cannot have a unique solution. Although this may seem a trivial problem, the algorithm must cope with it. The natural solution would be to impose a fixed norm for two of the modes. In the early phases of this work the constraints

$$\sum_{i=1}^{n} |\mathbf{B}_{ih}| = 1, \quad \sum_{i=1}^{o} |\mathbf{C}_{ih}| = 1 \qquad (h = 1, \ldots, p) \tag{11}$$

were imposed. However, it was not possible to implement these constraints reliably so that they would not slow the iteration in some cases. For this reason the idea of fixing the norms of the factors was abandoned. In hindsight it appears that the choice of the norm was an unfortunate one: probably the sum-of-squares norm would be easier to implement without sacrificing the convergence properties of the algorithm.

The third indeterminacy is the rotational non-identifiability. If an array **X** does not fulfill the conditions specified by Kruskal [11,12], then the factorization is not unique. It is not possible to know beforehand if an array is identifiable or not. Thus the algorithm should be tolerant of the rotational non-identifiability as well as the factor scaling indeterminacy. This is achieved by including quadratic 'regularization' terms $R(\mathbf{A})$, $R(\mathbf{B})$, and $R(\mathbf{C})$ in the object function, as shown below.

Both the factor scaling indeterminacy and the rotational non-identifiability are significant in the present approach because several modes are computed together. In the usual ALS algorithm only one mode is computed in any given instance. Then the other two modes fix the scaling and the rotational status of the solution. Thus the computation of the ALS algorithm does not suffer from these indeterminacies. But of course, the indeterminacy is present in the ALS solution so that in indeterminate cases the ALS algorithm converges to different solutions when started from different initial values.

There are basically two techniques for implementing the non-negativity constraints. In small problems it is efficient to eliminate such unknowns from the problem which are about to become negative. This is called the active set strategy. In factor analytic problems there may be thousands of unknowns. It would be very slow to eliminate hundreds of variables one at a time in a program which changes all variables at a time. It would require that the model is recomputed once for each elimination because eliminating several variables at once may easily lead to an unstable algorithm. On the other hand, Bro remarked [13] that the utilization of the active set algorithm 'NNLS' by Lawson and Hanson [14] is very efficient in the 3-way ALS algorithm. These two statements are not in real conflict. In the ALS algorithm, only $p$ variables are changed at a time during one elementary step; there are $m + n + o$ such elementary steps in one major step. Eliminating one variable from the active set costs only one extra elementary step, not a full step. Such a cost is negligible even if thousands of variables are to be eliminated. This is in contrast with PMF3 where eliminating one single variable according to NNLS would cost one extra full step.

In this work the other option was adopted, the use of a penalty function which keeps all of the constrained variables strictly positive. (Sometimes the term *barrier function* is used when the variables are prevented from ever reaching the boundary.) Including the iterative penalty function technique to the iterative LS fit only causes a modest increase of the workload. Logarithmic penalty terms $P(\mathbf{A})$, $P(\mathbf{B})$, and $P(\mathbf{C})$ were included in the enhanced object function for all constrained factor elements. The expression for the enhanced object function is thus

$$\begin{aligned}\tilde{Q}(\mathbf{E},\mathbf{A},\mathbf{B},\mathbf{C}) &= Q(\mathbf{E}) + P(\mathbf{A}) + P(\mathbf{B}) + P(\mathbf{C}) + R(\mathbf{A}) + R(\mathbf{B}) + R(\mathbf{C}) \\ &= \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{o} w_{ijk}\mathbf{E}_{ijk}^2 - \alpha\left(\sum_{i=1}^{m}\sum_{h=1}^{p}\log\mathbf{A}_{ih} + \sum_{j=1}^{n}\sum_{h=1}^{p}\log\mathbf{B}_{jh} + \sum_{k=1}^{o}\sum_{h=1}^{p}\log\mathbf{C}_{kh}\right) \\ &\quad + \gamma\left(\sum_{i=1}^{m}\sum_{h=1}^{p}\mathbf{A}_{ih}^2 + \sum_{j=1}^{n}\sum_{h=1}^{p}\mathbf{B}_{jh}^2 + \sum_{k=1}^{o}\sum_{h=1}^{p}\mathbf{C}_{kh}^2\right). \qquad (12)\end{aligned}$$

The coefficient $\alpha$ controls the strength of the penalty terms which prevent the factors **A**, **B**, and **C** from becoming negative. Similarly, $\gamma$ controls the strength of the regularization terms. Both the coefficients $\alpha$ and $\gamma$ are given smaller values during the iteration so that their final values are 'negligible but not zero'. A smaller $\alpha$ lets the variables approach closer to the value zero. Similarly, a smaller $\gamma$ decreases the distortion caused by regularization. It appears that initial larger values of these coefficients allow the early part of the iteration to proceed more quickly.

Other functional forms could be used for the regularization instead of the quadratic functional. The sum of absolute values was suggested by Paatero [7] as a means for achieving desired rotations. Other forms could perhaps be tailored to particular problems. No alternatives were tried in the present work; the quadratic form was chosen because it is easy to implement and also because it has been used much e.g. in deconvolution and in ridge regression. Similarly, there may be alternatives for the logarithmic form of the penalty function. Negative powers have the desired behavior of growing without limit when the argument approaches zero. They were not tried in the present work and thus it is impossible to know how well they would work. The heuristic reason for choosing the logarithmic form was that the logarithm is scale-invariant.

Eq. (12) is in fact only suitable if there are no essential scale differences between the slices of the array **X**. Otherwise the regularization terms $R(\mathbf{A})$, $R(\mathbf{B})$, and $R(\mathbf{C})$ must be modified so that they influence different slices of the array in correct proportions:

$$R(\mathbf{A}) = \gamma\sum_{i=1}^{m}\sum_{h=1}^{p}\gamma_i^{\mathrm{A}}\mathbf{A}_{ih}^2 \qquad (13)$$

and similarly for $R(\mathbf{B})$ and $R(\mathbf{C})$. In the current program the coefficients $\gamma_i^{\mathrm{A}}$, $\gamma_j^{\mathrm{B}}$, and $\gamma_k^{\mathrm{C}}$ are computed by a simple heuristic algorithm which attempts to give equal significance to all slices; no theoretical basis is available for this algorithm. However, note that usually the overall penalty scaling coefficients $\alpha$ and $\gamma$ are given small enough values towards the end of the iteration so that the exact values of the coefficients $\gamma_i^{A}$, $\gamma_j^{B}$, and $\gamma_k^{C}$ are not important if the model is identifiable.

### 4.1. The non-negativity constraints and the logarithmic penalty function

The true logarithmic penalty function prevents any of the factor values from ever reaching the exact value zero. However, by specifying a sufficiently small coefficient $\alpha$ one may allow that the factor values may approach arbitrarily close to zero. On the other hand, moving to/from zero is a slow process. Up to ten iterations may be required when a factor component descends down to near-zero or climbs up from near-zero. Thus getting near to zero should not be allowed in the early phases of the iteration when the model has not yet been stabilized, i.e. it is not yet known which values will finally be near-zero. The current program is run so that three user-specified pairs of values $\alpha$, $\gamma$, are applied. First the largest values are in use while the solution 'takes shape'. When the convergence starts to slow down with the largest pair, the medium-sized values are chosen. They allow the solution to settle down. Finally the smallest values allow that the solution approaches close to the exact zeros.

The logarithmic penalty function is approximated by a quadratic expression around the current point. Such an approximated penalty function is not strong enough in order to prevent negative values from being occasionally computed for the next point. This approach may seem inefficient as it necessitates cutting the computed step short. However, it would be very slow indeed to implement the true logarithmic form of the penalty separately at each step. The overall strategy here is to *postpone all precision calculations to as late as possible*, to a time when the model is a better approximation of the final result (''never do today something which may become unnecessary by tomorrow''). It suffices that the quadratic approximation is a true local representation of the logarithmic shape when the iteration has converged with the current coefficient $\alpha$. In other words, the penalty shape converges simultaneously with the solution, both reach their final values simultaneously.

It was stated above that the logarithmic penalty function prevents any of the factor values from ever reaching an exact value of zero. On the other hand it was also said that the approximating quadratic expression occasionally allows zero or negative values to be computed. It has been suggested that perhaps the approximation is crucial in allowing the computed factor values to become small enough. Closer inspection reveals that this is not the case: the converged solution is the same when the approximation is used and when true logarithms are used for computing the steps. Essential for obtaining sufficiently good approximations of zero values for factor elements is that the last value assigned to the coefficient $\alpha$ be small enough.

### 4.2. *Enumerating the unknowns and the data points*

The current point is denoted by **A**, **B**, and **C**. At each step it is necessary to determine three increment matrices $\Delta\mathbf{A}$, $\Delta\mathbf{B}$, and $\Delta\mathbf{C}$ so that the new point defined by the matrices $\mathbf{A}+\Delta\mathbf{A}$, $\mathbf{B}+\Delta\mathbf{B}$, and $\mathbf{C}+\Delta\mathbf{C}$ would minimize Eq. (12). The three unknown increment matrices $\Delta\mathbf{A}$, $\Delta\mathbf{B}$, and $\Delta\mathbf{C}$ contain $(m+n+o)p$ unknown values. The standard technique for solving least squares (LS) problems represents all the unknown values as elements of one vector. Thus, the three factor matrices must be mapped onto a vector of dimension $(m+n+o)p$. It would be possible to name the unknowns according to their index in the vector. This would be quite unclear in the equations, however. Instead, the elements of the vector shall be designated with a triple notation $(r, j, x)$. The first index of a triple may have the values $r=1, \dots, p$. The last index $x$ is one of the three non-numerical symbols $a$, $b$, or $c$, pointing to one of the matrices $\Delta\mathbf{A}$, $\Delta\mathbf{B}$, or $\Delta\mathbf{C}$. The second index $j$ has values from 1 up to $m$, $n$, or $o$, depending on whether $x$ is $a$, $b$, or $c$. The ordering of the elements in the vector is $\Delta\mathbf{A}_{11}$ $\Delta\mathbf{A}_{12}$ $\dots$ $\Delta\mathbf{A}_{1p}$ $\Delta\mathbf{A}_{21}$ $\dots$ $\Delta\mathbf{C}_{1o}$ $\Delta\mathbf{C}_{2o}$ $\dots$ $\Delta\mathbf{C}_{po}$. The corresponding index triples are written as $(1, 1, a)$ $(2, 1, a)$ $\dots$ $(p, 1, a)$ $(1, 2, a)$ $\dots$ $(1, o, c)$ $(2, o, c)$ $\dots$ $(p, o, c)$.

The matrix of the LS normal equations has dimensions $(m+n+o)p$ by $(m+n+o)p$. The elements of this matrix are identified by two triples. The first triple indicates the row, the second the column. Only the lower triangle of this symmetric matrix is actually needed in the program.

In a LS model the data to be fitted is also represented by a vector. Thus the arrays **X**, **Y**, and **E** must also be arranged as vectors. The ordering of points is chosen so that the first index $(i=1, 2, \dots, m)$ changes most rapidly, the third $(k=1, \dots, o)$ most slowly. The notation vect(**E**) means the values of the array **E**, arranged as a vector.

## 5. The iterative LS fit

The iteration consists of repeatedly solving for the increment matrices $\Delta\mathbf{A}$, $\Delta\mathbf{B}$, and $\Delta\mathbf{C}$, and 'making the step', i.e. setting **A**, **B**, and **C** equal to the sums $\mathbf{A}+\Delta\mathbf{A}$, $\mathbf{B}+\Delta\mathbf{B}$, and $\mathbf{C}+\Delta\mathbf{C}$. The customary technique of solving the PARAFAC model is 'alternating least squares' (ALS). It consists of iteratively solving the three elementary linear LS tasks

$$\begin{aligned}&\min_{\Delta \mathbf{A}} Q(\mathbf{X}, \mathbf{A}+\Delta \mathbf{A}, \mathbf{B}, \mathbf{C})\\&\min_{\Delta \mathbf{B}} Q(\mathbf{X}, \mathbf{A}, \mathbf{B}+\Delta \mathbf{B}, \mathbf{C})\\&\min_{\Delta \mathbf{C}} Q(\mathbf{X}, \mathbf{A}, \mathbf{B}, \mathbf{C}+\Delta \mathbf{C}).\end{aligned} \tag{14}$$

At each elementary step the current point (**A**, **B**, **C**) is considered known. Thus the elementary problem is linear and may be solved exactly. The computed increment is immediately added to the factor matrix, the updated value is then used in the following elementary steps. Ross and Leurgans [4] have enhanced the convergence rate of ALS by performing a long continuation step (**A**, **B**, and **C** change together) after the three elementary steps. The combined workload of the three elementary ALS steps is approximately $3mnop(p+2)$. In the unweighted case (all $\mathbf{w}_{ijk}=1$) the workload drops to approximately $6mnop$ operations. (The exact values depend on how the computations are organized.)

The new PMF approach is based on simultaneous determination of increments to all three factor matrices. It consists of repeatedly solving the non-linear LS task

$$\min_{\Delta \mathbf{A}, \Delta \mathbf{B}, \Delta \mathbf{C}} Q(\mathbf{X}, \mathbf{A}+\Delta \mathbf{A}, \mathbf{B}+\Delta \mathbf{B}, \mathbf{C}+\Delta \mathbf{C}). \tag{15}$$

In contrast to Eq. (14), Eq. (15) is a non-linear LS problem. Only an approximate solution is computed at each step. The mathematics of the step is first described using informal intuitive notation, where the product notation means the outer matrix product. Then the detailed equations are given based on the description of the Gauss–Newton approach previously presented.

The increment matrices $\Delta\mathbf{A}$, $\Delta\mathbf{B}$, and $\Delta\mathbf{C}$ are required to solve in the least squares sense the overdetermined system of equations $\mathbf{X}=(\mathbf{A}+\Delta\mathbf{A})(\mathbf{B}+\Delta\mathbf{B})(\mathbf{C}+\Delta\mathbf{C})$. Using the current residual array $\mathbf{E}=\mathbf{X}-\mathbf{ABC}$, the equations to be solved may be written as

$$\mathbf{AB}\Delta\mathbf{C}+\mathbf{A}\Delta\mathbf{BC}+\Delta\mathbf{ABC}+\mathbf{A}\Delta\mathbf{B}\Delta\mathbf{C}+\Delta\mathbf{AB}\Delta\mathbf{C}+\Delta\mathbf{A}\Delta\mathbf{BC}+\Delta\mathbf{A}\Delta\mathbf{B}\Delta\mathbf{C}=\mathbf{E}. \tag{16}$$

The Gauss–Newton approximation is obtained by dropping all higher-order terms and solving the simple equation

$$\mathbf{AB}\Delta\mathbf{C}+\mathbf{A}\Delta\mathbf{BC}+\Delta\mathbf{ABC}=\mathbf{E}. \tag{17}$$

The detailed equations are obtained by applying Eq. (10) to the enhanced object function defined in Eq. (12). The Jacobian matrix **J** contains the partial derivatives of each $\mathbf{Y}_{ijk}$ with respect to each one of the unknown increment values $\Delta\mathbf{A}_{ih}$, $\Delta\mathbf{B}_{jh}$, and $\Delta\mathbf{C}_{kh}$. The Jacobian is sparse and contains repetitions of same values. Its sparsity structure is shown in Fig. 1. In reality this matrix is not present in the program at all. The vector $\Delta\boldsymbol{\Theta}$ consists of all the $(m+n+o)p$ unknowns $\Delta\mathbf{A}$, $\Delta\mathbf{B}$, and $\Delta\mathbf{C}$, as already explained.

The regularization terms are represented by a diagonal matrix $\boldsymbol{\Gamma}$, consisting of the coefficients $\gamma_i^{\mathrm{A}}$, $\gamma_j^{\mathrm{B}}$, and $\gamma_k^{\mathrm{C}}$, multiplied by the strength parameter $\gamma$ (see Eq. (13)). The logarithmic penalty function is approximated by a diagonal matrix $\boldsymbol{\Lambda}$ and vector $\boldsymbol{\Theta}^*$, according to the technique shown in Eq. (7).

The 'normal equations' (Eq. (20)) of the Gauss–Newton approach are obtained from the equations

$$\boldsymbol{\Psi}^{\mathrm{E}}=\boldsymbol{\Psi}+\boldsymbol{\Gamma}+\boldsymbol{\Lambda}=\mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J}+\boldsymbol{\Gamma}+\boldsymbol{\Lambda}, \tag{18}$$

$$\boldsymbol{\Xi}^{\mathrm{E}}=\boldsymbol{\Xi}-\boldsymbol{\Gamma}\boldsymbol{\Theta}^0+\boldsymbol{\Lambda}\boldsymbol{\Theta}^*=\mathbf{J}^{\mathrm{T}}\mathbf{W}\,\mathrm{vect}(\mathrm{E})-\boldsymbol{\Gamma}\boldsymbol{\Theta}^0+\boldsymbol{\Lambda}\boldsymbol{\Theta}^*, \tag{19}$$

$$\boldsymbol{\Psi}^{\mathrm{E}}\Delta\boldsymbol{\Theta}=\boldsymbol{\Xi}^{E}, \tag{20}$$

where **W** is the diagonal matrix of the weights $\mathbf{w}_{ijk}$. It is best to compute matrix $\boldsymbol{\Psi}$ by using explicit formulae for its elements, not by straightforward matrix multiplication as suggested by the appearance of Eq. (18). The

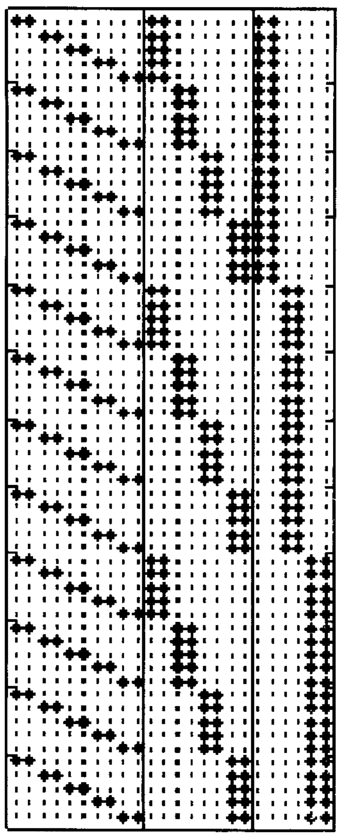

Fig. 1. A schematic presentation of the Jacobian matrix **J** for a model where $m=5$, $n=4$, $o=3$, and $p=2$. Large dots indicate non-zero elements of the matrix. Each row of **J** corresponds to one element in the 3-way array **X**. The ordering of rows is so that the first index runs most rapidly from 1 to 5. The columns correspond to the unknowns. The ordering of the columns is the same as the ordering of factor matrix elements in the vector of unknowns. The vertical lines delineate the columns corresponding to the unknown matrices $\Delta\mathbf{A}$, $\Delta\mathbf{B}$, and $\Delta\mathbf{C}$.

sparsity structure of matrix $\boldsymbol{\Psi}$ is shown in Fig. 2. Eq. (20) is best solved by Cholesky decomposition of the matrix $\boldsymbol{\Psi}^{\mathrm{E}}$. The zeros in the topmost diagonal block are conserved in the Cholesky decomposition whereas in the other blocks the zero elements are filled in. This utilization of sparsity is useful if the dimensions obey the condition $m \gg n+o$. Computing the Cholesky decomposition amounts to approximately $p^3(3m+n+o)(n+o)^2/6$ operations. This step is usually the largest workload of the new algorithm.

If the weights obey $w_{ijk}=1/\sigma_{ijk}^2$ then the inverse matrix $(\boldsymbol{\Psi}^{\mathrm{E}})^{-1}$ is the joint covariance matrix for all the increments $\Delta\mathbf{A}$, $\Delta\mathbf{B}$, and $\Delta\mathbf{C}$. The significance or usefulness of this covariance matrix will be explored in the future.

## *5.1. Component presentation of the normal equations*

The elements of matrix $\boldsymbol{\Psi}$ in Eq. (20) are given in Eqs. (21)–(27). In all these equations, the indices $q$ and $r$ independently have the values $1, \ldots, p$.

$$\boldsymbol{\Psi}(q,i,a;r,j,a)=0 \quad \text{if } i\neq j; \quad \text{similarly for } b \text{ and } c, \tag{21}$$

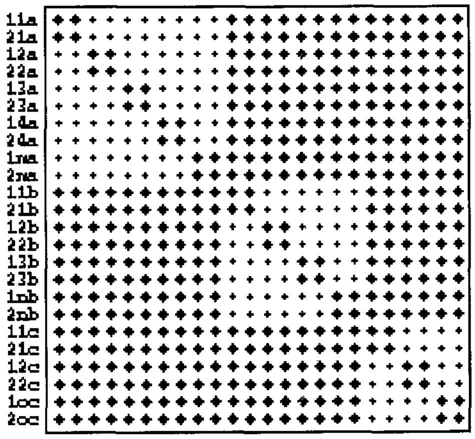

Fig. 2. A schematic presentation of the matrix $\boldsymbol{\Psi}$ for a model where $m=5$, $n=4$, $o=3$, and $p=2$. Large dots indicate non-zero elements of the matrix. Index triples mark the variables corresponding to each row of $\boldsymbol{\Psi}$. The correspondence to columns is the same as to rows.

$$\boldsymbol{\Psi}(q,i,a;r,i,a)=\sum_{h=1}^{n}\sum_{k=1}^{o}w_{ihk}\mathbf{B}_{hq}\mathbf{B}_{hr}\mathbf{C}_{kq}\mathbf{C}_{kr}\quad(i=1,\ldots,m),\tag{22}$$

$$\boldsymbol{\Psi}(q,j,b;r,j,b)=\sum_{i=1}^{m}\sum_{h=1}^{o}w_{ijh}\mathbf{A}_{iq}\mathbf{A}_{ir}\mathbf{C}_{hq}\mathbf{C}_{hr}\quad(j=1,\ldots,n),\tag{23}$$

$$\boldsymbol{\Psi}(q,k,c;r,k,c)=\sum_{i=1}^{m}\sum_{h=1}^{n}w_{ihk}\mathbf{A}_{iq}\mathbf{A}_{ir}\mathbf{B}_{hq}\mathbf{B}_{hr}\quad(k=1,\ldots,o).\tag{24}$$

There are six off-diagonal blocks: *ab*, *ac*, *bc*, *ba*, *ca*, *cb*. Because the matrix is symmetric, the last three blocks are transposes of the first three, e.g. $\boldsymbol{\Psi}(r,k,c;q,i,a)=\boldsymbol{\Psi}(q,i,a;r,k,c)$. Thus their equations are not given separately.

$$\boldsymbol{\Psi}(q,i,a;r,j,b)=\mathbf{A}_{ir}\mathbf{B}_{jq}\sum_{k=1}^{o}w_{ijk}\mathbf{C}_{kq}\mathbf{C}_{kr}\quad(i=1,\ldots,m;\ j=1,\ldots,n),\tag{25}$$

$$\boldsymbol{\Psi}(q,i,a;r,k,c)=\mathbf{A}_{ir}\mathbf{C}_{kq}\sum_{j=1}^{n}w_{ijk}\mathbf{B}_{jq}\mathbf{B}_{jr}\quad(i=1,\ldots,m;\ k=1,\ldots,o),\tag{26}$$

$$\boldsymbol{\Psi}(q,j,b;r,k,c)=\mathbf{B}_{jr}\mathbf{C}_{kq}\sum_{i=1}^{m}w_{ijk}\mathbf{A}_{iq}\mathbf{A}_{ir}\quad(j=1,\ldots,n;\ k=1,\ldots,o).\tag{27}$$

The workload in forming matrix $\boldsymbol{\Psi}$ is $1.5mnop(p+1)$ operations.

The right-side vector $\boldsymbol{\Xi}$ of the normal equation (Eq. (20)) is given by

$$\begin{aligned}\boldsymbol{\Xi}(q,i,a)&=\sum_{j=1}^{n}\sum_{k=1}^{o}w_{ijk}\mathbf{E}_{ijk}\mathbf{B}_{jq}\mathbf{C}_{kq}\quad(i=1,\ldots,m)\\ \boldsymbol{\Xi}(q,j,b)&=\sum_{i=1}^{m}\sum_{k=1}^{o}w_{ijk}\mathbf{E}_{ijk}\mathbf{A}_{iq}\mathbf{C}_{kq}\quad(j=1,\ldots,n)\\ \boldsymbol{\Xi}(q,k,c)&=\sum_{i=1}^{m}\sum_{j=1}^{n}w_{ijk}\mathbf{E}_{ijk}\mathbf{A}_{iq}\mathbf{B}_{jq}\quad(k=1,\ldots,o),\end{aligned}\tag{28}$$

where $\mathbf{E}$ is the current residual matrix, $\mathbf{E}_{ijk}=\mathbf{X}_{ijk}-\sum_{h=1}^{p}\mathbf{A}_{ih}\mathbf{B}_{jh}\mathbf{C}_{kh}$.

The workload in forming the vector $\Xi$ is $mno(2p+1)$ operations.

### 5.2. Step length control

In principle the solution of Eq. (20) gives both the step direction and the step length. This step is based on the local linearization of the model. However, the non-linearities of the model sometimes cause the computed step to produce a worsening of the fit, i.e. an increased value of $Q$. Allowing such steps would lead to a divergent algorithm. It is necessary to check each step for a decreasing $Q$. If an increase is found then the step is shortened and again the $Q$ is computed. If repeated shortening of the step does not lead to a decrease of $Q$ then the tentative step is abandoned altogether and instead an initial-style step (see below) is made. This happens very seldom except when the minimum has already been found.

The proposed step must also be checked with respect to the constraints. The step is not computed with a true logarithmic penalty function, as implied by Eq. (12). Instead a quadratic approximation of the logarithmic part is used. The longer the step the worse is this approximation so that the computed step may well violate the constraints. Then it is necessary to shorten the step and update the quadratic approximation of the logarithmic penalty function. In fact this updating is performed after each step, violating or not, because it presents a relatively small computational workload.

The final step length thus fulfills two criteria: it does not violate the constraints and it guarantees a decrease of the object function.

### 5.3. Startup of the algorithm

A starting point is needed for the algorithm. Two options have been used in the present work: a set of values specified by the user or a set of pseudorandom values. In both cases, it is necessary to check that the point is feasible, i.e. it does not violate any of the non-negativity constraints.

The program must cope with the situation that the initial approximation is poor requiring large :hanges to the initial values. The steps computed by the main algorithm would need to be shortened to a very small fraction of the computed length. This would be slow and costly. Marquardt-style step length control (described below) would help somewhat. However, it is better to use simpler means for the initial steps because the sophistication of the main algorithm would be wasted in any case.

The present program performs the initial steps so that it adjusts a single value of any of the matrices **A**, **B**, or **C** at a time, performing the constraint check at the same time. The workload for one round of such co-ordinate steps (all elements of **A**, **B**, and **C**) is approximately $9mnop$ work units. When three rounds of these initial steps have been made so that no constraint violations were imminent, then the program shifts to the main algorithm. An alternative possibility for startup would be the usual ALS steps; this alternative has not been explored. The heuristic reason for choosing single variable steps was the simplicity of implementing non-negativity.

### 5.4. Enhancements of the algorithm

The algorithm of the previous section is fully functional, in fact such an algorithm has been used routinely since 1994. In some situations the algorithm appears slow, making short zig–zag steps so that the total number of steps may approach one hundred or so. The following enhancements may speed up the convergence by a factor of two or three.

#### 5.4.1. Marquardt-type step length control

In the early phases of the iteration (after performing the initial steps) the computed increments may still be large, even exceeding the current factor values. Because of non-linear effects such steps get truncated to a small fraction of the original length, this makes the progress slow. It was found useful to control the step length using a technique which is often called 'Levenberg–Marquardt' step control (see Dahlqvist and Björk, p. 442 [15]): all the diagonal entries of matrix $\boldsymbol{\Psi}^{E}$ are multiplied by the value $1 + \lambda$ before solving Eq. (20). The constant $\lambda$ is maintained dynamically so that $\lambda$ is increased whenever there is need to limit the length of computed steps, and decreased when it is desirable to compute longer steps: whenever the computed steps can be made of full length, the value of $\lambda$ is decreased by a fixed factor (e.g. 0.2) towards a lower limit, typically 0.01. When the tentative step has to be shortened, then $\lambda$ is increased by a certain factor, e.g. 5.

### 5.4.2. *Newton–Raphson algorithm*

The basic Gauss–Newton algorithm utilizes only the Jacobian matrix, consisting of the partial derivatives of $\mathbf{Y}_{ijk}$ with respect to the unknowns. All higher-order terms are rejected in Eq. (17). If all the second derivatives of $Q$ with respect to the unknowns are used, then the algorithm is called Newton–Raphson (see Dahlqvist and Björk, pp. 442–444 [15]). The definition of the matrix $\boldsymbol{\Psi}$ in Eq. (20) is then

$$\boldsymbol{\Psi}_{ij} = \frac{\partial^2 Q}{\partial \Phi_i \partial \Phi_j} \tag{29}$$

the right-side vector $\boldsymbol{\Xi}$ remains the same as in Gauss–Newton. This technique has been tried for solving the 3-way factor analytic model.

The Newton–Raphson approach leads to the same equations as given above for the Gauss–Newton algorithm, except for one change: an additional term appears in the off-diagonal blocks of matrix $\boldsymbol{\Psi}$. Eq. (25) becomes

$$\left.\begin{aligned} \boldsymbol{\Psi}(q, i, a; r, j, b) &= \mathbf{A}_{ir}\mathbf{B}_{jq}\sum_{k=1}^{o} w_{ijk}\mathbf{C}_{kq}\mathbf{C}_{kr} \quad (q \neq r) \\ \boldsymbol{\Psi}(q, i, a; q, j, b) &= \mathbf{A}_{iq}\mathbf{B}_{jq}\sum_{k=1}^{o} w_{ijk}\mathbf{C}_{kq}^2 - \sum_{k=1}^{o} w_{ijk}\mathbf{E}_{ijk}\mathbf{C}_{kq} \end{aligned}\right\} \quad i = 1, \ldots, m;\ j = 1, \ldots, n \tag{30}$$

and similarly Eqs. (26) and (27) are changed accordingly. In some tests the Newton–Raphson variant performed well. However, the matrix defined by Eq. (30) need not be positive definite and in many trials this result caused problems. Eq. (20) cannot be used as such if the matrix is indefinite. Attempts to circumvent this problem did not lead to a reliable and efficient algorithm based on Eq. (30).

### 5.4.3. *Newton–Raphson correction to the Gauss–Newton algorithm*

An alternative approach to utilizing the 2nd derivatives of the object function might be to correct the Gauss–Newton step so that the corrected step approximates the true Newton–Raphson step. After laborious analysis, it was found that this is indeed possible. The N–R-corrected right-side vector $\boldsymbol{\Xi}^{\mathrm{NR}}$ is defined by

$$\begin{aligned} \boldsymbol{\Xi}^{\mathrm{NR}}(q, i, a) &= \sum_{j=1}^{n}\sum_{k=1}^{o} w_{ijk}\mathbf{E}_{ijk}(\mathbf{B}_{jq} + \lambda\Delta\mathbf{B}_{jq})(\mathbf{C}_{kq} + \lambda\Delta\mathbf{C}_{kq}) \quad (i = 1, \ldots, m) \\ \boldsymbol{\Xi}^{\mathrm{NR}}(q, j, b) &= \sum_{i=1}^{m}\sum_{k=1}^{o} w_{ijk}\mathbf{E}_{ijk}(\mathbf{A}_{iq} + \lambda\Delta\mathbf{A}_{iq})(\mathbf{C}_{kq} + \lambda\Delta\mathbf{C}_{kq}) \quad (j = 1, \ldots, n) \\ \boldsymbol{\Xi}^{\mathrm{NR}}(q, k, c) &= \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ijk}\mathbf{E}_{ijk}(\mathbf{A}_{iq} + \lambda\Delta\mathbf{A}_{iq})(\mathbf{B}_{jq} + \lambda\Delta\mathbf{B}_{jq}) \quad (k = 1, \ldots, o), \end{aligned} \tag{31}$$

cf. Eq. (28) for $\boldsymbol{\Xi}$. When Eq. (20) has been solved, the corrected $\boldsymbol{\Xi}^{\mathrm{NR}}$ is computed by inserting the newly computed increments $\Delta\mathbf{A}$, $\Delta\mathbf{B}$, and $\Delta\mathbf{C}$ into Eq. (31). The N–R-corrected solution $\Delta\boldsymbol{\Theta}^{\mathrm{NR}}$ (= improved values for increments $\Delta\mathbf{A}$, $\Delta\mathbf{B}$, and $\Delta\mathbf{C}$) is then obtained by solving

$$\boldsymbol{\Psi}^{\mathrm{E}}\Delta\boldsymbol{\Theta}^{\mathrm{NR}} = \boldsymbol{\Xi}^{\mathrm{NR,E}}. \tag{32}$$

Solving this equation is fast because the Cholesky decomposition of matrix $\boldsymbol{\Psi}^{\mathrm{E}}$, already computed when solving Eq. (20), is also valid now. In theory there should be no parameter $\lambda$ in Eq. (31). However, practical experience has shown that the best results are obtained with $\lambda = 0.5$. The reason for this is not clear.

Eqs. (31) and (32) define an iterative scheme which only produces the true N–R solution if iterated until convergence. In the problematic cases mentioned above, this iteration would diverge. Practical experience indicates that it is best to be satisfied with just a partial N–R correction: Eqs. (31) and (32) are either not iterated at all, or are iterated only once.

## 6. Problems: Multiple local solutions, non-identifiable solutions, degenerate models

All two-way factorizations are in principle non-identifiable unless some additional constraints (such as non-negativity, orthogonality) remove all rotational freedom from the solution. In contrast, many 3-way models are fully unique by themselves so that there is no rotational ambiguity at all. Criteria for identifying these unique factorizations are given by Kruskal [11,12]. Verifying the correctness of 3-way programs (discussed below) is easiest when dealing with these good identifiable cases.

The solutions for 3-way models applied to real data often seem to have multiple local solutions. Depending on the starting point of the iteration, the program finds one or another of these alternative solutions. This behavior is typical for all existing PARAFAC algorithms which are based on the least squares definition of the problem: there is no known technique which would allow the identification of the global minimum in reasonable time except for running the iteration repeatedly from different starting points. One soon notices that repeated runs find the same local solutions over and over again, each of them typically having its own unique value of $Q$. Then one may be fairly confident that the 'best' solution is among those found so far. It is tentatively suggested that all local solutions should be inspected and considered. It is not safe to assume that the solution with the smallest $Q$ would necessarily be the 'best' or 'correct' solution.

Typical for the non-identifiable cases is that the same minimum value of the object function $Q$ may be reached with different factor values. If a non-identifiable case is run repeatedly, the same value is obtained for $Q$ but some of the factors may be different in each run. However, regularization may have an influence here. Minimizing the enhanced function $Q$ tends to make the case pseudo-identifiable and selects among all the possible solutions favoring the solution(s) with smallest possible regularization terms. This is a by-product of the regularization which may not be useful in itself. The regularization is needed in order to obtain any solution at all; without regularization the matrix $\boldsymbol{\Psi}$ of a non-identifiable case would be singular and no solution could be obtained. One could make arbitrary changes in the coefficients of the individual terms in the regularization expressions $R(\mathbf{A})$, $R(\mathbf{B})$, and $R(\mathbf{C})$. This would change the solution of an non-identifiable case and serve as a warning for non-identifiability. However, it is expected that the covariance matrix $(\boldsymbol{\Psi}^{\mathrm{E}})^{-1}$ of the computed factor elements will act as a better warning of non-identifiability. In a non-identifiable case there should be some very large elements in the covariance matrix. This question is a subject of further study.

Degeneracy is a surprising property of the PARAFAC model, it has no analogy in the 2-way factor analysis. This question is discussed at length by Kruskal et al. [16]; a degenerate demonstration array of dimensions 2 by 2 by 2 is given there. When a degenerate case is solved, better and better approximations (smaller and smaller values of $Q$) are obtained when some of the factors approach infinity in such a way that large negative and positive values mostly cancel each other. This phenomenon is sometimes called 'divergence' which may be misleading. It would be more correct to say that the solution approaches the infinity point while the fitted array $\mathbf{Y}$

approaches a limiting value. When the $2 \times 2 \times 2$ demonstration matrix is analyzed by PMF3, one may observe how the absolute values of the solution grow larger and larger. Of course, requiring non-negativity will prevent the degenerate solution from diverging to negative values, but even then the solution may be strange, approaching zero without any apparent reason. It is necessary to restate the results from Kruskal et al. [16]: degenerate solutions are not always due to a problem in any given algorithm. They may be the property of the array $\mathbf{X}$, and it appears that degenerate arrays $\mathbf{X}$ are not at all seldom in practice. All unconstrained least squares algorithms will produce a sequence of ever-growing solutions for a degenerate case, provided they are run long enough (perhaps tens of thousands of iteration steps). Getting meaningful solutions for degenerate cases requires that the model be somehow modified so that the undesired freedom is removed from the model. Adding some extra constraints might perhaps prevent large negative and positive values from appearing. Solving the degenerate cases in a useful way will be the subject of another future paper; the problem is mentioned here in order to help the users diagnose this special difficulty when evaluating 3-way programs.

## 7. Discussion

The new algorithm has good convergence properties. Typically it needs 40 steps and hardly ever more than 150 steps for full convergence. The following results have been obtained in a comparison of ALS vs. PMF3 made by Harshman and Lundy [17]:

''In easy cases with orthogonal factors both programs needed typically 30 iterations. In difficult cases (near collinearity of factors, different magnitudes of factors) less than 100 iterations was almost always sufficient for PMF and usually less than 50 iterations. In contrast PARAFAC needed 60 to 200 times as many iterations, or even more.''

''Also, when dealing with such 'difficult' datasets PARAFAC was much more likely to have problems with 'swamps' (degenerate-form solutions which still fail to converge after very many iterations, even after 20,000 iterations, although presumably would converge eventually since the data had true trilinear structure). Sometimes for 5 or 6 out of 6 random starts this would occur, while PMF3 would quickly recover the trilinear structure for 4 to 6 out of 6 random starts.''

Comparisons of four different algorithms for solving the PARAFAC model have been made by a group led by Hopke and these results will be published soon [18]. Although one step made by the new algorithm consumes more time than the steps made by other algorithms, the rate of convergence is sufficiently good so that its overall speed is clearly better for most problems.

The major weakness of the new algorithm could be the behavior with extremely large models. The main computational workload of one step is proportional to the expression $p^3(3m+n+o)(n+o)^2/6$, cf. $3mnop(p+2)$ for the ALS. If two of the dimensions are large then the workload may grow too much. Assuming $m=n=100$, $o=p=5$ would give approximately $10^8$ operations/step which is still possible. On the other hand, already $m=n=300$, $o=p=10$ would exceed $10^{10}$ operations/step which is on the limit of being impossible on a PC computer.

### 7.1. Verification of results

In the introduction it was indicated that verification of results requires special attention. The end user of the algorithm PMF3 cannot verify correctness of the program by just recompiling the algorithm, in so doing the duplication would be too close, possible errors or problems in the original algorithm would also be duplicated. Neither is it feasible to re-code the algorithm because of the large amount of work and the need of special experience. The following approaches are available in order to verify the results:

In the current case the formulation of the mathematical model being solved is particularly simple, it is the minimization of a well-defined function under a number of inequality constraints. General-purpose programs exist for such minimization. Thus small 3-way problems could probably be solved successfully with these function minimizers. This would perform a check of the algorithm PMF3. This has not been attempted in the present work.

Comparison with other 3-way programs offers another possibility of checking. Such work is underway and will be published soon. The main result seems to be that PMF3 and the other programs agree when solving most of the suite of test cases.

Analyzing synthetic error-free ($\mathbf{E} = 0$) test cases offers another possibility for testing. Then the correct answer is known, it is those matrices **A**, **B**, and **C** that were used for synthesizing the array **X**. PMF3 has performed well in these tests. However, success in these error-free tests does not guarantee that the program would perform well in real-life tests where **E** is significantly non-zero. Synthetic tests may be constructed so that the model is identifiable by observing that the true factor matrices fulfill the Kruskal conditions [11,12]. Degeneracy should not be a problem with such synthetic test data. However, local solutions may occur.

Repeated runs of the same problem, starting from different initial values for the matrices **A**, **B**, and **C** is another means for testing. The same result should be achieved regardless of the starting values. If small changes of the starting point result in similar changes in the results, then the program is not converging properly. PMF3 has also performed well in these tests; these and the error-free synthetic test cases have been the main lines of testing it.

Another possible approach for testing might be to evaluate the gradient of $Q$ at the proposed minimum. If the gradient is significantly non-zero then a minimum has not been found. However, numerical accuracy causes that all gradient components are non-zero; the problem is to decide what is 'significantly non-zero'. This approach has not been tried in the present work.

In all testing one must observe the possibility of the problem cases, outlined in the previous section. If two runs or two algorithms give two different answers (different factors and different $Q$) then there is the possibility that they have found different local minima. It is then necessary to repeat the runs with yet different starting points and observe the results. If the two algorithms produce the same set of solutions (not necessarily in the same order) then they agree. If two algorithms produce different factor solutions having essentially the same $Q$, then it is possible that the case is non-identifiable and that different forms of regularization select different solutions from the infinite set of possible solutions. And if only one algorithm is being tested it is probably good if it repeatedly produces a finite number of different results (some of the results may appear more often than the others). If, on the other hand, an infinite number of different solutions seems to be appearing then either the algorithm is not converging properly or else the test case is either degenerate or non-identifiable, not fulfilling the Kruskal conditions [11,12,16]. Degeneracy is indicated if the solutions contain extremely large negative and positive values.

### 7.2. Standard deviations not known for data?

Sometimes the objection is raised that in practice the standard deviations of data are not known, hence such weighted models as PMF cannot be used. Here it is necessary to recall that the customary factor analytic techniques are also implicitly based on assumptions of standard deviations of data values: PCA is optimal if the standard deviations of all elements in the (possibly centered and column-scaled) matrix **X** are equal. If columns have been centered and normalized then the optimality criterion of PCA for the original matrix is that the standard deviations in each column of the original matrix are constant and proportional to the norm of the centered column, see Paatero and Tapper [9]. If the standard deviations are in fact something different then PCA can still be used but it is no longer optimal. If absolutely nothing is known of the standard deviations of the data, then the best non-robust approach would probably be to assume equal std-dev for all data, and thus accept the use of unscaled PCA. However, the same applies also to the weighted models such as PMF: If nothing is known of the standard deviations of the data, then one may assume that all values $\sigma_{ijk} = 1$ and one is doing equally well as with PCA. (It would probably be better to use PMF in the robust mode, as described by Paatero [7] but this question goes outside of the present work.) On the other hand, some information of the standard deviations of the data is almost always available. Typically one knows the laboratory error estimates, they give the lower limits for standard deviations of data values. Sometimes repeated measurements give indication of data reproducibility, and so on.

It is seen that the weighted methods are never worse than the unweighted ones: it is possible to use constant weighting if nothing better is available. Almost always there is some information about the standard deviations although complete knowledge may well be limited to the realm of simulation studies.

### *7.3. Availability of the programs*

The programs PMF2 and PMF3 have been written in Fortran90. Normally they are only distributed in compiled form, as .exe files. Contact the author about the availability of the programs for various platforms. For PC computers (486-Pentium) free evaluation licenses for the programs are available for trying out the programs during a period of six months.

## Acknowledgements

Financial support from the Vilho, Yrjö, and Kalle Väisälä foundation is gratefully acknowledged. The help of Richard Harshman was crucial for understanding the nature of the degenerate solutions. I wish to thank Philip K. Hopke for hospitality and support while preparing this manuscript. The enumeration of the outstanding problems with PARAFAC was suggested by an anonymous referee. This work was supported in part by the U.S. Environmental Protection Agency under Grant No. R822482-01-0 to Clarkson University.

## Appendix A. Demonstration of the slowness of the ALS algorithm

In unconstrained unweighted 2-way factor analysis the ALS algorithm usually converges reasonably fast. A slow convergence is only observed if the singular values $\mathbf{S}_{pp}$ and $\mathbf{S}_{p+1,p+1}$ are of almost equal magnitude. However, weighting of data points or imposing non-negativity constraints may slow ALS dramatically. Similarly extending the model to three ways may slow ALS. It is well known that extremely slow convergence of ALS is observed in connection with degenerate arrays and so-called swamps. However, here the emphasis is on the slowness of the ordinary non-degenerate no-swamp convergence of ALS. It is hoped that the degenerate cases may be analyzed in future studies.

The following $2 \times 2 \times 1$ array is analyzed with a 1-factor model ($p = 1$):

| observed matrix **X** | | matrix of std. deviations $\boldsymbol{\sigma}$ | |
|---|---|---|---|
| 1 | 10 | 1 | 1 |
| 10 | 70 | 1 | 30 |

The following initial values were used: $\mathbf{A}^T = \mathbf{B}^T = (2 \quad 5)$, $\mathbf{C} = 1$.

When this task is solved with the ALS algorithm, approximately 55 steps are needed in order that all the factor elements converge to the relative precision of $10^{-4}$. The final convergence is geometric, with each step the distance to the true solution decreases by 15%.

When the same task is solved with PMF3, the first five steps are initial one-variable steps. In this case ($p = 1$) they are comparable to five ALS steps. Then the PMF3 algorithm converges in four main steps so that the relative error of the factor elements drops well below $10^{-4}$.

This simple example is not suffering from numerical instabilities and there cannot be any degeneracies in a one-factor model. Thus the observed difference of convergence rates is simply due to the difference in the optimization principles. In this case the efficiency of the subspace steps of ALS is on the average less than one tenth of the efficiency of the full-space steps of PMF3.

## Appendix B. Pseudocode version of the algorithm PMF3

```
0 Input the matrices $\mathbf{X}$ and $\boldsymbol{\sigma}$, initiate the factor matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ to pseudorandom or user-specified values. Check for feasibility. Turn full-step mode off.
1. Initial operations when starting a new step.
   1.1. Perform reweighting if needed (e.g. if doing a robust computation).
   1.2. Compute the value $Q_1$ of the object function $Q$ when starting the step.
   1.3. If previous step(s) have been performed without step cutting (at paragraphs 2.2 and 5.1) and full step mode is off, switch the full step mode on.
   1.4. If full-step mode is on, perform a full step (paragraphs 3 to 6), otherwise, perform one-variable-at-a-time step (paragraph 2).
2. One-variable-at-a-time step: For each factor element in turn, do 2.1 to 2.3.
   2.1. Determine an increment of that variable so that the object function $Q$ (including penalty) is minimized with respect to this increment.
   2.2. Prevent the factor value from becoming negative by cutting the step short if necessary.
   2.3. Update the approximation of the logarithmic penalty function for this element.
   2.4. Continue with the convergence test, paragraph 7.
3. $\Psi$ and $\Xi$
   3.1. Compute the matrix $\Psi$ and the vector $\Xi$ according to Eqs. (21)–(28).
   3.2. Adjust the diagonal of $\Psi$ according to the current values of the penalty functions and Marquardt step-length control.
   3.3. Adjust the vector $\Xi$ according to the current values of the penalty functions.
4. Solve the step $\Delta\Theta$ from Eq. (20)
   4.1. Compute the Cholesky decomposition of $\Psi^{E}$.
   4.2. Solve for $\Delta\Theta$ by back-substitution.
5. Process $\Delta\Theta$ to find the final step vector.
   5.1. Cut the step if necessary so that no constrained component becomes negative.
   5.2. Compute $Q = Q_2$. If $Q_2 > Q_1$, cut the step and iterate 5.2 until $Q_2 < Q_1$ or until 5.2 has been iterated 10 times. If the latter, continue at paragraph 2.
   5.3. Attempt another cut to see if an even smaller value of $Q$ can be obtained.
   5.4. Compute the N–R corrected $\Xi$ from Eq. (31). Solve for corrected $\Delta\Theta$ by back-substituting according to Eq. (32). Process the corrected solution through the checks in 5.1, 5.2, and 5.3. If they fail, accept the uncorrected solution as it was after step 5.3.
6. Now a new improved feasible solution has been secured.
   6.1. Compute the new current point = old current point + step.
   6.2. Update the quadratic approximations of log penalty functions
   6.3. Update the Marquardt parameter $\lambda$ according to step cutting at 5.1 and 5.2.
   6.4. Compute $Q = Q_3$, write information about the current solution.
7. Convergence testing.
   7.1. Is this the $\upsilon$th consecutive step fulfilling $|Q_1 - Q_3| < \Delta$?
   7.2. If it is, choose the next set of penalty coefficients ($\alpha$ and $\gamma$) and continue with paragraph 1, or continue with paragraph 8 (results) if last pair of ($\alpha$ and $\gamma$) has already been applied.
   7.3. If it is not, continue the iteration with paragraph 1.
```

8. Output of results.
   8.1. Write $Q$, factors, residuals, etc., as specified by the user.
   8.2. Compute and write the covariance matrix $(\Psi^{E})^{-1}$ if so requested by the user.
   8.3. End the run or continue with another task.

Notes.

The step cutting in paragraphs 2.2 and 5.1 is done so that no constrained component is allowed to decrease more than by $xx\%$ in a single step (typically $xx = 60$). This keeps the points at a safe distance from the constraint boundaries and allows a stable gradual decrease towards zero.

In 7.1, the step count $\upsilon$ and convergence limit $\Delta$ are specified by the user separately corresponding to each of the different pairs ($\alpha$ and $\gamma$). Typical values: $\upsilon = 3$, $\Delta = 5$ (first pair of $\alpha$ and $\gamma$), $\Delta = 0.01$ (last pair of $\alpha$ and $\gamma$). These values are not critical.

## References

[1] R. Coppi, S. Bolasco (Eds.), Multiway Data Analysis, Elsevier, Amsterdam, 1989.

[2] T. Wansbeek, J. Verhees, Models for multidimensional matrices in econometrics and psychometrics, in: R. Coppi, S. Bolasco (Eds.), Multiway Data Analysis, Elsevier, Amsterdam, 1989, pp. 543–552.

[3] R.A. Harshman, M.E. Lundy, The PARAFAC model for three-way factor analysis and multidimensional scaling, in: Law et al. (Eds.), Research Methods for Multimode Data Analysis, Praeger, New York, 1984, pp. 122–215.

[4] R.T. Ross, S. Leurgans, Component resolution using multilinear models, Meth. Enzymol. 246 (1995) 679–700.

[5] P. Paatero, U. Tapper, Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values, Environmetrics 5 (1994) 111–126.

[6] R.A. Harshman, Foundations of the PARAFAC procedure, UCLA working papers in phonetics, Vol. 16, 1970, pp. 1–84, University Microfilms International No. 10,085.

[7] P. Paatero, Least squares formulation of robust non-negative factor analysis, Chemometrics Intell. Lab. Syst. 37 (1997) 23–35.

[8] J.M.F. ten Berge, Convergence of PARAFAC preprocessing procedures and the Deming–Stephan method of iterative proportional fitting, in: R. Coppi, S. Bolasco (Eds.), Multiway Data Analysis, Elsevier, Amsterdam, 1989, pp. 53–63.

[9] P. Paatero, U. Tapper, Analysis of different modes of factor analysis as least squares fit problems, Chemometrics Intell. Lab. Syst. 18 (1993) 183–194.

[10] B.C. Mitchell, D.S. Burdick, Slowly converging PARAFAC sequences: Swamps and two-factor degeneracies, J. Chemometrics 8 (1994) 155–168.

[11] J.B. Kruskal, Three-way arrays: Rank and Uniqueness of trilinear decompositions, with applications to arithmetic complexity and statistics, Linear Algebra Appl. 18 (1977) 95–138.

[12] J.B. Kruskal, Rank, decomposition, and uniqueness for 3-way and $N$-way arrays, in: R. Coppi, S. Bolasco (Eds.), Multiway Data Analysis, Elsevier, Amsterdam, 1989, pp. 7–18.

[13] R. Bro, Comment to this paper in the INCINC96 conference, 1996.

[14] C.L. Lawson, R.J. Hanson, Solving Least Squares Problems, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[15] G. Dahlqvist, Å. Björk, Numerical Methods, Prentice-Hall, Englewood Cliffs, NJ, 1974.

[16] J.B. Kruskal, R.A. Harshman, M.E. Lundy, How 3-MFA data can cause degenerate parafac solutions, among other relationships, in: R. Coppi, S. Bolasco (Eds.), Multiway Data Analysis, Elsevier, Amsterdam, 1989, pp. 115–130.

[17] R.A. Harshman, M.E. Lundy, Private communication, 1996.

[18] P.K. Hopke, P. Paatero, H. Jia, R.T. Ross, R.A. Harshman, A comparative review of four different PARAFAC programs, Chemometrics Intell. Lab. Syst. (1997), in preparation.