Research Article

JOURNAL OF CHEMOMETRICS

Received: 1 August 2007, Revised: 17 December 2007, Accepted: 28 December 2007, Published online in Wiley InterScience: 2008

(www.interscience.wiley.com) DOI: 10.1002/cem.1130

Tucker1 model algorithms for fast solutions to large PARAFAC problems†

Mark H. Van Benthem[a]* and Michael R. Keenan[a]

We describe a method of performing trilinear analysis on large data sets using a modification of the PARAFAC-ALS algorithm. Our method iteratively decomposes the data matrix into a core matrix and three loading matrices based on the Tucker1 model. The algorithm is particularly useful for data sets that are too large to upload into a computer's main memory. While the performance advantage in utilizing our algorithm is dependent on the number of data elements and dimensions of the data array, we have seen a significant performance improvement over operating PARAFAC-ALS on the full data set. In one case of data comprising hyperspectral images from a confocal microscope, our method of analysis was approximately 60 times faster than operating on the full data set, while obtaining essentially equivalent results. Published in 2008 by John Wiley & Sons, Ltd.

Keywords: PARAFAC; Tucker model; multivariate factor analysis; trilinear

1. INTRODUCTION

Very large data sets are becoming commonplace in the analytical chemistry laboratory [1]. So-called hyphenated techniques have driven this data richness in the past [2] and now, the phenomenon is being driven ever faster by methods that employ multiple sources and detectors such as hyperspectral imaging [3–5]. The trilinear nature of some of these data makes analysis by PARAFAC [6] very attractive because of its special properties which can produce rotationally unique results. Unfortunately, large data sets make data handling and processing difficult. The hyperspectral imaging confocal microscope (HSI-CM) at Sandia National Laboratories [4] collects images that are on the order of 200 × 200 pixels by 512 emission wavelength elements. In order to examine the interaction of different fluorophores, we will collect hyperspectral images as a function of 18 to 20 time-sequenced photobleaching events. The size of these data sets exceeds 1 Gigabyte in *single precision*. The challenges to analyzing data of this extent, with commonly available computational equipment and techniques, are many and self-evident.

Analyzing very large data sets with alternating least squares based methods is especially difficult. For small data sets three copies of the data are usually retained in memory, one arranged for each mode of the three-way data. This is prohibitive with very large data sets, as only one complete set of data can be retained in RAM, if the full data can be loaded in memory at all. In principle, one can operate with a single three-way array and merely permute it to cycle through the data modes as needed. Unfortunately, alternating least squares methods require transposition of the data or factor matrices. In MATLAB® [7], transposition is effectively accomplished by making a copy of the matrix while permuting its elements. In-place transposition algorithms exist, but they can be slow and do not exist in the native MATLAB® language. This problem, in particular, makes operating PARAFAC-ALS problematic with large data since repeated transpositions (or higher order permutations) are necessary. Fortunately, Tucker1 and Tucker3 [8,9] PCA models exist which allow data compression prior to trilinear analysis.

Tucker1 models are particularly attractive because there are well known, robust methods for performing PCA on two-way data, that is, matrices. Unfortunately, Tucker1 models do not provide a least squares approximation to the data as do Tucker3 methods.[9] However, Tucker3 models usually involve many iterations of least squares on the full data set. Given that each method has it drawbacks, in this paper we will explore several ways of exploiting Tucker1 models as preliminary compression tools and actively use them during PARAFAC-ALS. Our novel algorithms permit a fast, convenient way to estimate the trilinear factors from very large three-way data.

In this paper we describe a method of approximating a Tucker1 model that runs very quickly on large data sets. We accomplish this by performing a factorization of an unfolded three-way array in one dimension and then performing subsequent factorizations of the resulting scores matrix rearranged in an appropriate fashion to yield the other two dimensions. As a tutorial, we will discuss the use of Tucker1 models in PARAFAC as a way to compress the data and to increase the speed of calculations. Finally, we demonstrate the advantages of our method over the PARAFAC on uncompressed data.

Notation is as follows: (1) scalars are lowercase italics, (2) vectors are lowercase bold and are all presented as column vectors, (3) matrices are uppercase bold, and (4) three-way arrays are indicated by fancy bold letters. We will utilize the notation of Tucker [9] for arrays reorganized as matrices and for matrices as well. This format will become obvious below.

* *Sandia National Laboratories, MS0895, Albuquerque, NM 87185-0895, USA. E-mail: mhvanbe@sandia.gov*

a *M. H. Van Benthem, M. R. Keenan*
Sandia National Laboratories, MS0895, Albuquerque, NM 87185-0895, USA

2. HYPERSPECTRAL IMAGING FLUORESCENCE DATA

In its typical data acquisition mode, the HSI-CM collects notional two-way data. That is, for an r-component chemical system, the data follow the model

$$f_{ik} = \sum_{p=1}^{r} s_{ip} c_{kp} + e_{ik} \tag{1}$$

where f_{ik} is the fluorescence intensity at wavelength i and pixel k, s_{ip} is the unit-concentration fluorescence response for the pth fluorophore at the ith wavelength, c_{kp} is the concentration, or relative amount of the pth fluorophore, at kth pixel, and e_{ik} is the error in the model fit of f_{ik}. Rewriting this relationship in matrix terms yields

$$\mathbf{F} = \mathbf{S}\mathbf{C}^{\mathrm{T}} + \mathbf{E} \tag{2}$$

where $\mathbf{F}$ is the $i \times k$ matrix of fluorescence intensities, $\mathbf{S}$ is the $i \times r$ matrix of fluorescence responses and $\mathbf{C}$ is the $k \times r$ matrix of concentrations. A typical method used to analyze the two-way data in $\mathbf{F}$ is multivariate curve resolution (MCR) [10–12]. Briefly, the matrices $\mathbf{S}$ and $\mathbf{C}$ are alternately estimated using multivariate least squares while employing constraints. Using Tucker's notation [9], our indices above are used to represent the mode of the data, when associated with matrices. Equation (2) can be written as

$${}_{i}\mathbf{F}_{k} \cong {}_{i}\mathbf{S}_{r}({}_{k}\mathbf{C}_{r})^{\mathrm{T}} \equiv {}_{i}\mathbf{S}_{r}\mathbf{C}_{k} \tag{3}$$

where, ${}_{i}\mathbf{F}_{k}$ is oriented as wavelength (i-mode) by concentration (k-mode) with dimensions $i \times k$. Clearly, the transpose operator can now be represented as a change in subscripts. When we add an additional acquisition mode, temporal photobleaching, in which the fluorescence intensity has a linear dependence as do the concentration and wavelength modes, the HSI-CM can collect notional three-way data.

Three-way data that follow the trilinear model can be, expressed as

$$f_{ijk} = \sum_{p=1}^{r} s_{ip} t_{jp} c_{kp} + e_{ijk} \tag{4}$$

where f_{ijk} is f_{ik} measured at the jth time interval and t_{jp} is the concentration-independent photodecomposition rate for the pth fluorophore evaluated at the jth time interval. This triple product is sometimes represented as [13]

$$\mathbb{F} = \otimes\left({}_{i}\mathbf{S}_{r}, {}_{j}\mathbf{T}_{r}, {}_{k}\mathbf{C}_{r}\right) + \mathbb{E} \tag{5}$$

where the fancy $\mathbb{F}$ is now a three-way array with error $\mathbb{E}$. Three-way data collected as concentration by wavelength matrices for a set of j photobleaching events, ${}_{i}\mathbf{F}_{k}(1)$ through ${}_{i}\mathbf{F}_{k}(j)$, they would be organized conceptually as a deck of cards into the array,

$$\mathbb{F} = \{\, {}_{i}\mathbf{F}_{k}(1) \quad {}_{i}\mathbf{F}_{k}(2) \quad \cdots \quad {}_{i}\mathbf{F}_{k}(j) \,\} \tag{6}$$

It is fairly straightforward to convert $\mathbb{F}$ into a matrix and this may be done in a number of ways. For convenience, we perform this transformation by simply adjoining the matrices next to one another to create an i by (jk) matrix

$${}_{i}\mathbf{F}_{(jk)} = [\, {}_{i}\mathbf{F}_{k}(1) \quad {}_{i}\mathbf{F}_{k}(2) \quad \cdots \quad {}_{i}\mathbf{F}_{k}(j) \,] \tag{7}$$

Now, the Tucker notation [9] has the data oriented as i by (jk), with j-outer loop and k-inner loop. The other two orientations of the data are easily generated using a cyclic permutation of ${}_{i}\mathbf{F}_{(jk)}$, involving only a transpose operation and matrix reshape. Specifically, they are

$${}_{k}\mathbf{F}_{(ij)} = {}_{k}({}_{(jk)}\mathbf{F}_{i})_{(ij)} \tag{8}$$

$${}_{j}\mathbf{F}_{(ki)} = {}_{j}({}_{(ij)}\mathbf{F}_{k})_{(ki)} = {}_{j}({}_{(ij)}({}_{k}({}_{(jk)}\mathbf{F}_{i})_{(ij)})_{k})_{(ki)} \tag{9}$$

The identity on the right side of Equation (9) illustrates the functionality of the Tucker notation as it is used in this work. It indicates from inside-out of the parentheses the operations on the matrix ${}_{i}\mathbf{F}_{(jk)}$: (1) transpose to ${}_{(jk)}\mathbf{F}_{i}$, (2) reshape to ${}_{k}\mathbf{F}_{(ij)}$, (3) transpose to ${}_{(ij)}\mathbf{F}_{k}$, and finally, (4) reshape to ${}_{j}\mathbf{F}_{(ki)}$. An additional cycle of the matrix in Equation (9) results in our original data in Equation (6),

$${}_{i}\mathbf{F}_{(jk)} = {}_{i}({}_{(ki)}\mathbf{F}_{j})_{(jk)} \tag{10}$$

Note how the indices rotate across the parentheses and the bold letter from the left and move from the right hand side to the far left position. In terms of computational demand, the most costly parts of the permutations above are the transpositions. Those familiar with MATLAB® may recognize the *reshape* operation, which requires no movement of data. Since matrices in MATLAB® are organized as column-major, notionally a column vector with integers (indices) that specify how that linear sequence of numbers is interpreted as a matrix; reshaping matrices is a simple matter of rewriting these indices. Having constructed the matrices in Equations (7), (8) and (9), we will now construct the conformable matrices needed to make the least squares estimates of ${}_{i}\mathbf{S}_{r}$, ${}_{k}\mathbf{C}_{r}$ and ${}_{j}\mathbf{T}_{r}$, respectively.

Similar to the way the data in the array $\mathbb{F}$ are rearranged to create the matrices in Equations (7)–(9), we need to arrange our model estimates in ${}_{i}\mathbf{S}_{r}$, ${}_{k}\mathbf{C}_{r}$ and ${}_{j}\mathbf{T}_{r}$ to make our least squares projections. In the case of the estimate for ${}_{i}\mathbf{S}_{r}$, we generate the conformable matrix, ${}_{r}\mathbf{X}_{(jk)}$, into which we project the data ${}_{i}\mathbf{F}_{(jk)}$ using the *Khatri–Rao* operator [14]. The *Khatri–Rao* operator, denoted by $\odot$, essentially performs the *vec* operation [15,16] on the outer product of *each* column of the pair of matrices on which it operates. The following equation provides an example:

$$\begin{aligned} {}_{(jk)}\mathbf{X}_{r} &= \left({}_{j}\mathbf{T}_{r} \odot {}_{k}\mathbf{C}_{r}\right) \\ &= \left[\, vec\left({}_{k}\mathbf{c}(1)\mathbf{t}_{j}(1)\right) \quad vec\left({}_{k}\mathbf{c}(2)\mathbf{t}_{j}(2)\right) \quad \cdots \quad vec\left({}_{k}\mathbf{c}(r)\mathbf{t}_{j}(r)\right) \,\right] \end{aligned} \tag{11}$$

With Equation (11) we can solve for ${}_{i}\hat{\mathbf{S}}_{r}$ using least squares, *viz*,

$${}_{i}\hat{\mathbf{S}}_{r} = \left({}_{i}\mathbf{F}_{(jk)}\mathbf{X}_{r}\right)\left({}_{r}\mathbf{X}_{(jk)}\mathbf{X}_{r}\right)^{-1} \tag{12}$$

which is the normal equations formulation of the least squares problem. However, explicitly constructing ${}_{r}\mathbf{X}_{(jk)}$ and forming its cross-product is costly. One can avoid the expense of the calculations through better computational approaches. Specifically, the cross-product ${}_{r}\mathbf{X}_{(jk)}\mathbf{X}_{r}$ can be directly computed using the Hadamard product (element-by-element product indicated

by $*$) of the cross-products of the individual matrices, ${}_k\mathbf{C}_r$ and ${}_j\mathbf{T}_r$, that is, [15,17]

$$ {}_r\boldsymbol{\Omega}_r \equiv ({}_r\mathbf{C}_k\mathbf{C}_r) * ({}_r\mathbf{T}_j\mathbf{T}_r) = ({}_r\mathbf{X}_{(jk)}\mathbf{X}_r) \tag{13} $$

Efficiently computing the cross-product ${}_i\mathbf{F}_{(jk)}\mathbf{X}_r$ poses a greater challenge. The strategy here involves computing part of the cross-product for the solution of one mode and using that solution to also solve the following mode. For example, to compute ${}_i\mathbf{F}_{(jk)}\mathbf{X}_r$ for the solution of ${}_i\hat{\mathbf{S}}_r$ we first compute

$$ {}_{(ki)}\mathbf{Y}_r = {}_{(ki)}({}_i\mathbf{F}_{(jk)})_j\mathbf{T}_r \tag{14} $$

where ${}_{(ki)}({}_i\mathbf{F}_{(jk)})_j$ is merely a low-cost reshape of ${}_i\mathbf{F}_{(jk)}$. The next step is to complete the cross-product

$$ \forall p \in \{1,2,\ldots,r\}, {}_i\mathbf{q}_p = {}_i({}_{(ki)}\mathbf{Y}_p)_k\mathbf{c}_p \tag{15} $$

where ${}_k\mathbf{c}_p$ is the pth column of ${}_k\mathbf{C}_r$, ${}_i({}_{(ki)}\mathbf{Y}_p)_k$ is the $i \times k$-reshaped pth column of ${}_{(ki)}\mathbf{Y}_r$, and ${}_i\mathbf{q}_p$ is the pth column of ${}_i\mathbf{Q}_r$, which is

$$ {}_i\mathbf{Q}_r = {}_i\mathbf{F}_{(jk)}\mathbf{X}_r = {}_i\mathbf{F}_{(jk)}({}_j\mathbf{T}_r \odot {}_k\mathbf{C}_r)_r \tag{16} $$

Thus, the least squares estimate for ${}_i\hat{\mathbf{S}}_r$ is

$$ {}_i\hat{\mathbf{S}}_r = ({}_i\mathbf{Q}_r)({}_r\boldsymbol{\Omega}_r)^{-1} \tag{17} $$

We can now recycle ${}_{(ki)}\mathbf{Y}_r$ toward obtaining the least squares estimate of ${}_k\hat{\mathbf{C}}_r$ using

$$ \forall p \in \{1,2,\ldots,r\}, {}_k\mathbf{p}_p = {}_k({}_{(ki)}\mathbf{Y}_p){}_i\hat{\mathbf{s}}_p \tag{18} $$

where ${}_k\mathbf{p}_p$ is the pth column of ${}_k\mathbf{P}_r = {}_k\mathbf{F}_{(ij)}({}_i\hat{\mathbf{S}}_r \odot {}_j\mathbf{T}_r)_r$. The estimate of ${}_k\hat{\mathbf{C}}_r$ is computed in the next step using

$$ {}_k\hat{\mathbf{C}}_r = {}_k\mathbf{P}_r({}_r\boldsymbol{\Xi}_r)^{-1} \tag{19} $$

where

$$ {}_r\boldsymbol{\Xi}_r \equiv ({}_r\mathbf{T}_j\mathbf{T}_r) * ({}_r\hat{\mathbf{S}}_i\hat{\mathbf{S}}_r) = ({}_i\hat{\mathbf{S}}_r \odot_j \mathbf{T}_r)^{\mathrm{T}}({}_i\hat{\mathbf{S}}_r \odot_j \mathbf{T}_r) \tag{20} $$

The third step is the least squares solution of ${}_j\hat{\mathbf{T}}_r$. First compute

$$ {}_r\mathbf{Z}_{(jk)} = {}_r\hat{\mathbf{S}}_i\mathbf{F}_{(jk)} \tag{21} $$

Completing the cross-product with each column of ${}_k\hat{\mathbf{C}}_r$

$$ \forall p \in \{1,2,\ldots,r\}, {}_j\mathbf{r}_p = {}_k({}_p\mathbf{Z}_{jk})_j\hat{\mathbf{c}}_p \tag{22} $$

where ${}_j\mathbf{r}_p$ is the pth column of ${}_j\mathbf{R}_r = {}_j\mathbf{F}_{(ki)}({}_k\hat{\mathbf{C}}_r \odot {}_i\hat{\mathbf{S}}_r)_r$. Finally the solution for ${}_j\hat{\mathbf{T}}_r$ is

$$ {}_j\hat{\mathbf{T}}_r = {}_j\mathbf{R}_r({}_r\boldsymbol{\Pi}_r)^{-1} \tag{23} $$

where

$$ {}_r\boldsymbol{\Pi}_r \equiv ({}_r\hat{\mathbf{S}}_i\hat{\mathbf{S}}_r) * ({}_r\hat{\mathbf{C}}_k\hat{\mathbf{C}}_r) = ({}_k\hat{\mathbf{C}}_r \odot_i \hat{\mathbf{S}}_r)^{\mathrm{T}}({}_k\hat{\mathbf{C}}_r \odot_i \hat{\mathbf{S}}_r) \tag{24} $$

This approach is similar in principle to that of Tomasi [18]. We have found a modest savings from computing cross-products by efficiently computing the *Khatri–Rao* product, however, the extent of the benefits is dependent on the relative sizes of i, j and k.

3. USING THE TUCKER1 MODEL IN PARAFAC

For the moment, assume that the dimensions of the three-way array are unique (i.e., $i \neq j \neq k$); and that the product of any two dimensions is greater than the third (i.e., $i \times j > k$, $k \times i > j$ and $j \times k > i$). Further assume that the data in the three-way array follow the trilinear model of Equation (4). One can estimate the noise-free or chemical rank (pseudo-rank) of each matrix orientation of the data by examining the eigenvalues of the cross-product of that orientation. For example, to estimate the number of components in the wavelength mode, i, one would examine the eigenvalues of ${}_i\mathbf{F}_{(jk)}\mathbf{F}_i$. The maximum pseudo-rank one could estimate in this case is i, the dimension of left-hand side of ${}_i\mathbf{F}_{(jk)}$, since that is maximum possible rank of the *matrix* ${}_i\mathbf{F}_{(jk)}$. Of course, this is not necessarily the rank of the *array* $\mathbb{F}$ represented by ${}_i\mathbf{F}_{(jk)}$. Now, let us further assume that the number of components in the array, r, is less than the rank of each dimension (i.e., $r < \min\{i, j, k\}$). Under these conditions, the pseudo-rank for each matrix orientation of the array should exhibit the same pseudo-rank, r, and one can estimate the number of components by examining the pseudo-ranks of the three orientations. In this case, one could decompose the data in each mode into a reduced basis set of r-principal component scores and loadings and use these in a PARAFAC-ALS algorithm.

The assumptions made above are highly restrictive in a mathematical sense. However, for many spectroscopic purposes, they are not terribly restrictive. In the case of HSI-CM data, the array is $512 \times 40\,000 \times 18$. Is it reasonable to ever expect the pseudo-rank of the data to exceed 40 000 or even 512? Our answer is, 'Probably not'. If it does, the time required to decompose these data would be prohibitive. More to the point, the results are not likely interpretable. However, the pseudo-rank could easily exceed 18 for a complex image-system. In the case where $\max\{i, j, k\} > r > \min\{i, j, k\}$, each mode can still be modeled by its independent principal component scores and loadings, except in the case where r is greater than or equal to one or more i, j, or k, in which case the decomposition of the corresponding mode(s) will be full rank.

The Tucker1 model essentially finds principal-component models for each mode of data in a three-way array [19–21]. Thus for each mode, the method seeks a dimension reduction sufficient to account for all of the systematic variability in the data. Working with a reduced dimension model can significantly diminish the time required to perform PARAFAC-ALS. One can represent the Tucker1 model with the following three equations:

$$ {}_i\mathbf{V}_q\mathbf{D}_{(jk)} \cong {}_i\mathbf{F}_{(jk)} \tag{25} $$

$$ {}_j\mathbf{W}_m\mathbf{B}_{(ki)} \cong {}_j\mathbf{F}_{(ki)} \tag{26} $$

$$ {}_k\mathbf{U}_n\mathbf{A}_{(ij)} \cong {}_k\mathbf{F}_{(ij)} \tag{27} $$

In each case, **D**, **B** and **A** are the rank-q, -m or -n scores matrices, respectively, and **V**, **W** and **U** are the respective rank-q, -m, or -n matrices whose columns are mutually orthonormal. Ideally, the ranks of the decompositions are chosen such that $q = \min\{i, r\}$, $m = \min\{j, r\}$ and $n = \min\{k, r\}$. However, in practice it is often necessary to choose values of q, m, or n that are larger than r [20] because of effects of non-uniform noise, components that fall below noise levels or other data complexities. So, a better choice

for the rank of the decompositions is $q = \min\{i, r+\delta_i\}$, $m = \min\{j, r+\delta_j\}$ and $n = \min\{k, r+\delta_k\}$, where δ_i, δ_j, δ_k, are integer values chosen to ensure the data is appropriately modeled by the basis set.

Finally, the data can be approximated as

$$ {}_i\mathbf{F}_{(jk)} \cong {}_i\mathbf{V}_q\mathbf{G}_{(mn)}({}_m\mathbf{W}_j \otimes {}_n\mathbf{U}_k)_{(jk)} \tag{28} $$

where ${}_q\mathbf{G}_{(mn)}$ is the core matrix and $\otimes$ is the Kronecker tensor product [16]. Estimation of the core matrix is simply accomplished using

$$ {}_q\mathbf{G}_{(mn)} = {}_q\mathbf{V}_i\mathbf{F}_{(jk)}({}_j\mathbf{W}_m \otimes {}_k\mathbf{U}_n)_{(mn)} \tag{29} $$

For large data sets, it is more efficient to compute the ${}_q\mathbf{G}_{(mn)}$ using the following technique: [22]

$$ {}_q\mathbf{G}''_{(jk)} = {}_q\mathbf{V}_i\mathbf{F}_{(jk)} \tag{30} $$

$$ {}_n\mathbf{G}'_{(qj)} = {}_n\mathbf{U}_k\mathbf{G}''_{(qj)} \tag{31} $$

$$ {}_m\mathbf{G}_{(nq)} = {}_m\mathbf{W}_j\mathbf{G}'_{(nq)} \tag{32} $$

The core matrix ${}_q\mathbf{G}_{(mn)}$ can be substituted for the data matrix ${}_i\mathbf{F}_{(jk)}$ in a PARAFAC-ALS algorithm to solve for a set of transformation matrices: ${}_q\tilde{\mathbf{S}}_r$, ${}_n\tilde{\mathbf{C}}_r$ and ${}_m\tilde{\mathbf{T}}_r$. The core matrix PARAFAC-ALS solution is obtained in the same fashion as the full data set illustrated in Equations (12)–(24) substituting the transformation matrices for the letter-associated data-space solution matrices. The core matrix method implementation for the solution of ${}_q\tilde{\mathbf{S}}_r$ is

$$ {}_q\hat{\tilde{\mathbf{S}}}_r = {}_q\mathbf{G}_{(mn)}({}_m\tilde{\mathbf{T}}_r \odot {}_n\tilde{\mathbf{C}}_r)\left({}_r\tilde{\Omega}_r\right)^{-1} = {}_q\tilde{\mathbf{Q}}_r\left({}_r\tilde{\Omega}_r\right)^{-1} \tag{33} $$

where ${}_q\tilde{\mathbf{Q}}_r$ and ${}_r\tilde{\Omega}_r$ are computed in the same fashion as ${}_q\mathbf{Q}_r$ and ${}_r\Omega_r$, using ${}_n\tilde{\mathbf{C}}_r$ and ${}_m\tilde{\mathbf{T}}_r$ in place of ${}_n\mathbf{C}_r$ and ${}_m\mathbf{T}_r$, respectively.

${}_n\tilde{\mathbf{C}}_r$ is estimated in a complementary fashion using

$$ {}_n\hat{\tilde{\mathbf{C}}}_r = {}_n\mathbf{G}_{(qm)}\left({}_q\hat{\tilde{\mathbf{S}}}_r \odot {}_m\tilde{\mathbf{T}}_r\right)\left({}_r\tilde{\Xi}_r\right)^{-1} = {}_n\tilde{\mathbf{P}}_r\left({}_r\tilde{\Xi}_r\right)^{-1} \tag{34} $$

Finally, ${}_m\hat{\tilde{\mathbf{T}}}_r$ is estimated with

$$ {}_m\hat{\tilde{\mathbf{T}}}_r = {}_m\mathbf{G}_{(nq)}\left({}_n\hat{\tilde{\mathbf{C}}}_r \odot {}_q\hat{\tilde{\mathbf{S}}}_r\right)\left({}_r\tilde{\Pi}_r\right)^{-1} = {}_m\tilde{\mathbf{R}}_r\left({}_r\tilde{\Pi}_r\right)^{-1} \tag{35} $$

After convergence, the data-space solution matrices are computed using

$$ {}_i\hat{\mathbf{S}}_r = {}_i\mathbf{V}_q\hat{\tilde{\mathbf{S}}}_r \tag{36} $$

$$ {}_k\hat{\mathbf{C}}_r = {}_k\mathbf{U}_n\hat{\tilde{\mathbf{C}}}_r \tag{37} $$

$$ {}_j\hat{\mathbf{T}}_r = {}_j\mathbf{W}_m\hat{\tilde{\mathbf{T}}}_r \tag{38} $$

A drawback of the core-based method, as presented, is that it cannot easily employ constraints such as nonnegativity [20]. Fortunately, a minor modification to the core-matrix method will allow the use of fast nonnegative least squares algorithms [23,24]. Using the relationships in Equations (36), (37) and (38), we substitute ${}_q\mathbf{V}_i\mathbf{S}_r$, ${}_n\mathbf{U}_k\mathbf{C}_r$ and ${}_m\mathbf{W}_j\mathbf{T}_r$ for ${}_q\tilde{\mathbf{S}}_r$, ${}_n\tilde{\mathbf{C}}_r$, and ${}_m\tilde{\mathbf{T}}_r$, respectively, and rewrite Equations (33), (34) and (35) as

$$ {}_i\hat{\mathbf{S}}_r = \left({}_i\mathbf{V}_q\left({}_q\mathbf{G}_{(mn)}\left({}_m\mathbf{W}_j\mathbf{T}_r \odot {}_n\mathbf{U}_k\mathbf{C}_r\right)\right)\right)\left({}_r\Omega_r\right)^{-1} \tag{39} $$

$$ {}_k\hat{\mathbf{C}}_r = \left({}_k\mathbf{U}_n\left({}_n\mathbf{G}_{(qm)}\left({}_q\mathbf{V}_i\hat{\mathbf{S}}_r \odot {}_m\mathbf{W}_j\mathbf{T}_r\right)\right)\right)\left({}_r\Xi_r\right)^{-1} \tag{40} $$

$$ {}_j\hat{\mathbf{T}}_r = \left({}_j\mathbf{W}_m\left({}_m\mathbf{G}_{(nq)}\left({}_n\mathbf{U}_k\hat{\mathbf{C}}_r \odot {}_q\mathbf{V}_i\hat{\mathbf{S}}_r\right)\right)\right)\left({}_r\Pi_r\right)^{-1} \tag{41} $$

and solve them in a constrained least squares sense. While this is equivalent to reconstructing the model-estimated data from the core matrix, the estimate is not explicitly formed so memory demands are minimal and computation is fast.

This method substantially reduces the time required to perform PARAFAC-ALS. It is also amenable to different ranks, q, m and n, for the PCA decompositions and it is easy to impose nonnegativity and other constraints on the solution.

4. FAST TUCKER1 MODEL FOR LARGE DATA SETS

When data sets get very large, the ability to manipulate them becomes limited. Making multiple copies of arrays for the purpose of solving for the various modes becomes impossible in the volatile memory. Under this constraint, we need to find a more efficient method to perform PARAFAC-ALS. Key to this is to obtain the characteristic matrices, **V**, **U** and **W**, and the core matrix, **G**. At times, data sets are so large that they cannot even be loaded into the volatile memory. Under these circumstances, an out-of-core-memory PCA scheme is of greatest utility.

The out-of-core-memory PCA method involves loading the data in chunks sufficient to create a cross-product matrix on which eigenanalysis can be performed. Consider the $i \times k \times j$, $512 \times 40\,000 \times 18$, HSI-CM data, stored in a file format that has sequential spectra (i-mode), then sequential images (k-mode) and then sequential temporal profiles (j-mode). So the first 512 elements in the file contain the first spectrum (first pixel) of the first image of the first photobleach event, and the first 20 480 000 elements contains all spectra (all pixels) of the first image, of the first photobleach event. In order to obtain the 512×512 cross-product matrix ${}_i\mathbf{F}_{(jk)}\mathbf{F}_i$, we would need to load the first spectrum, or as many spectra as suitable, form a temporary cross-product matrix, ${}_i\mathbf{H}_i$, then load another set of spectra, form the cross-product of that set of spectra and add it to ${}_i\mathbf{H}_i$. When all of the data are loaded, cross-products formed, and summed, the result will be ${}_i\mathbf{F}_{(jk)}\mathbf{F}_i$; thus,

$$ {}_i\mathbf{F}_{(jk)}\mathbf{F}_i = {}_i\mathbf{H}_i = \sum_{l=1}^{jk} {}_i\boldsymbol{\sigma}(l)\boldsymbol{\sigma}_i(l) \tag{42} $$

is the desired product, where ${}_i\boldsymbol{\sigma}(l)$ is the lth spectrum in the data set. With ${}_i\mathbf{F}_{(jk)}\mathbf{F}_i$, we can compute ${}_i\mathbf{V}_q$ using eigenanalysis, solving the symmetrical eigenvalue problem

$$ {}_i\mathbf{V}_q\boldsymbol{\Sigma}_q\mathbf{V}_i \cong {}_i\mathbf{H}_i \tag{43} $$

where ${}_q\Sigma_q$ is the diagonal rank-q eigenvalue matrix of ${}_i\mathbf{H}_i$.

Ideally now one would compute ${}_k\mathbf{U}_n$ and ${}_j\mathbf{W}_m$ using the same methodology as for ${}_i\mathbf{V}_q$. This could work for ${}_j\mathbf{W}_m$, the temporal mode data; after all, its cross-product matrix is only 18×18.

Unfortunately, it requires reading the file elements at non-sequential positions and the intervals are potentially very large, which can be costly. Of course, the prospect of computing ${}_k\mathbf{U}_n$ from a $40\,000 \times 40\,000$ ${}_k\mathbf{F}_{(ij)}\mathbf{F}_k$ is not realistic with most desktop computers today. An alternative solution to this problem involves decompositions of reorientations of ${}_q\mathbf{D}_{(jk)}$. To accomplish this, we must now compute ${}_q\mathbf{D}_{(jk)}$.

Start by reloading the data in parts as described above to compute ${}_q\mathbf{D}_{(jk)}$. Load individual spectra or image chunks and multiply each ${}_q\mathbf{D}_{(jk)}$ loaded spectrum by ${}_i\mathbf{V}_q$ to sequentially build ${}_q\mathbf{D}_{(jk)}$, specifically,

$$\forall p \in \{1, 2, \ldots, jk\}_q\mathbf{d}_p = {}_q\mathbf{V}_i\sigma_p \tag{44}$$

With ${}_q\mathbf{D}_{(jk)}$, we will obtain the loadings for the other modes

$${}_k\mathbf{U}_n\mathbf{\Sigma}_n\mathbf{U}_k \cong {}_k\left({}_{(jk)}\mathbf{D}_q\right)_{(qj)}\left({}_q\mathbf{D}_{(jk)}\right)_k \cong {}_k\mathbf{F}_{(ij)}\mathbf{F}_k \tag{45}$$

$${}_j\mathbf{W}_m\mathbf{\Sigma}_m\mathbf{W}_j \cong {}_j\left({}_q\mathbf{D}_{(jk)}\right)_{(kq)}\left({}_{(jk)}\mathbf{D}_q\right)_j \cong {}_j\mathbf{F}_{(ki)}\mathbf{F}_j \tag{46}$$

where ${}_n\mathbf{\Sigma}_n$ and ${}_m\mathbf{\Sigma}_m$ are the diagonal rank-*n* and -*m* eigenvalue matrices of ${}_k\mathbf{F}_{(ij)}\mathbf{F}_k$ and ${}_j\mathbf{F}_{(ki)}\mathbf{F}_j$, respectively.

We use these relationships because ${}_i\mathbf{V}_q$ are the eigenvectors of ${}_i\mathbf{F}_{(jk)}\mathbf{F}_i$, so its product ${}_q\mathbf{V}_i\mathbf{V}_q$ is the identity, $\mathbf{I}_q$. The rearrangements of ${}_i\mathbf{V}_q\mathbf{D}_{(jk)}$ that produce the approximations to the products ${}_k\mathbf{F}_{(ij)}\mathbf{F}_k$ and ${}_j\mathbf{F}_{(kj)}\mathbf{F}_j$ result in identity matrices, $\mathbf{I}_{(qj)}$ and $\mathbf{I}_{(kq)}$, that are conformable with the rearranged scores matrices, ${}_k\mathbf{D}_{(qj)}$ and ${}_j\mathbf{D}_{(kq)}$, respectively. If the size of a single mode is greater than the product of the other mode and the rank of the factor analysis then the eigenanalysis can be performed on the shorter side, which can then be used to estimate the eigenvectors. For example, in the HSI-CM data $k > j \times q$, and so choose to compute the eigenvectors of the (*qj*) side of the matrix, then multiply ${}_k\mathbf{D}_{(qj)}$ by those eigenvectors and normalize the results to obtain ${}_k\mathbf{U}_n$.

Specifically, let

$${}_{(qj)}\mathbf{U}'{}_n\mathbf{\Sigma}_n\mathbf{U}'{}_{(qj)} \cong {}_{(qj)}\left({}_q\mathbf{D}_{(jk)}\right)_k\left({}_{(jk)}\mathbf{D}_q\right)_{(qj)} \tag{47}$$

$${}_k\mathbf{U}_n \cong {}_k\mathbf{D}_{(qj)}\mathbf{U}'{}_n\mathbf{\Sigma}_n^{-1/2} \tag{48}$$

Finally, the core matrix is easily computed as

$${}_n\mathbf{G}'_{(qj)} = {}_n\mathbf{U}_k\left({}_q\mathbf{D}_{(jk)}\right)_{(qj)} \tag{49}$$

$${}_m\mathbf{G}_{(nq)} = {}_m\mathbf{W}_j\left({}_n\mathbf{G}'_{(qj)}\right)_{(nq)} \tag{50}$$

Note that the intermediate core matrix, ${}_n\mathbf{G}'_{(qj)}$ in Equation (49) can be reshaped and substituted into Equation (46) for ${}_j\mathbf{D}_{(kq)}$. Thus,

$${}_j\mathbf{W}_m\Sigma_m\mathbf{W}_j \cong {}_j\mathbf{G}'_{(nq)}\mathbf{G}'_j$$

is another route to the solution of ${}_j\mathbf{W}_m$.

Admittedly, this technique, combined with the core matrix method described above, is a bookkeeping nightmare; but it is very fast, employable with large data sets and provides accurate results. An exemplary implementation in the MATLAB[a] language attached in Appendix 1 for the reader's perusal and elucidation. As in the method for using all three Tucker1 models, this technique is readily amenable to constraint imposition.

5. PERFORMANCE OF THE ALGORITHM ON SYNTHETIC DATA

In a first comparison of algorithms, we only compared the computational speed for given number of iterations for different array sizes and ranks. We constructed synthetic three-way data representing three arbitrary modes using triads of ranks five and fifteen. We chose dimensions of $10 \times 25 \times 30$, $20 \times 1000 \times 1500$, $20 \times 20 \times 20$ and $350 \times 350 \times 350$ elements to represent small and large asymmetric and symmetric problems. The data contained no noise or very low noise levels (<1% max signal) and the pure components in each mode had no well-defined structure, as they were chosen from a uniform random distribution. Each simulation comprised 200 iterations.

We computed the trilinear solution for each set of simulated data using the full-data PARAFAC-ALS method described in Equations (12–24) and the Fast Tucker1 method described above. The time to perform the initial PCA involved in the Fast Tucker1 method is included in the elapsed time in the simulation. The simulation was run employing using MATLAB® version 7.3.0.267 (R2006b) running on Microsoft Windows XP on a Dell Computer, using a dual core, 3.2 GHz Pentium IV processor, with 1.0 Gbyte RAM. Table I contains the results of the simulation. Rank indicated in the table is the rank of the array; however, the rank of the PCA used to form the core matrix is data rank plus three. These are also indicated in the table.

Table I illustrates the performance advantages and disadvantages of the Fast Tucker1 algorithm. When faced with a small problem, our new method can actually impose a penalty on performance, as is evidenced in the first column. Although in this case the overall time involved is so small, it would hardly be inconvenient for the user. However, for large problems our new method has a sizeable advantage over the full-data method, in both the symmetric and asymmetric cases.

6. PERFORMANCE OF THE ALGORITHM ON HSI-CM DATA

We next tested the new algorithm on actual data collected with the HSI-CM. Our collaborators at the University of Texas Medical Branch provided slides containing fixed HEK 293 cells, transiently transfected with green fluorescent protein (GFP) and yellow fluorescent protein (YFP). We imaged these slides using the Sandia National Laboratories HSI-CM. Data were collected with 488 nm laser excitation, employing a 20× objective, and using *x*- and *y*-translational step sizes of 0.5 μm and 0.48 μm, respectively. The images had original dimensions of 199×204 image pixels by 512 wavelength elements by 18 photobleaching steps for a total of 374 132 736 elements. However, due to some detector saturation, we eliminated 40 rows of the image plane reducing it to 159×204 image pixels for a total of 298 930 176 elements.

Data pretreatment included removing cosmic spikes, subtracting a simple offset added by the EMCCD software and prior to conducting PCA and PARAFAC, scaling the data by the inverse of the square-root of the mean in each of the three modes to accommodate Poisson-like noise [25]. The comparison was performed with MATLAB® version 7.3.0.267 (R2006b) running on Red Hat Linux, 64 bit Enterprise 4 version, on a Dell Computer, using a dual core, 3.2 GHz Xeon processor and 6.0 Gbyte RAM. This calculation was not possible with 32-bit Windows XP, given the size of the data and Windows memory limitations. The

Table I. Performance comparison for simulated data. Time units are seconds and speed units are iterations per second. Ratio is the PARAFAC-ALS time divided by the Fast Tucker1 Method time

	Trilinear array dimensions							
Mode A	10	20	20	350	10	20	20	350
Mode B	25	1000	20	350	25	1000	20	350
Mode C	30	1500	20	350	30	1500	20	350
Data rank	5	5	5	5	15	15	15	15
PCA rank	8	8	8	8	18	18	18	18
PARAFAC-ALS								
Time	0.23	164.29	0.18	239.83	0.48	174.54	0.33	293.59
Speed	885.27	1.22	1106.58	0.83	412.70	1.15	601.72	0.68
Fast Tucker1 Method								
Time	0.37	5.73	0.16	15.32	0.29	11.52	0.32	6.28
Speed	540.32	34.93	1273.43	13.06	691.45	17.36	624.31	31.85
Ratio	0.61	28.69	1.15	15.66	1.68	15.15	1.04	46.75

computational limits only applied to the full-data PARAFAC algorithm; however, our new code was able to perform the analysis on a 32-bit Windows XP computer using PCA scores and loadings computed using an out-of-core-memory computational scheme.

The side-by-side analyses included 1000 iterations, the first 100 of which were rigorously nonnegativity constrained to avert two-factor degeneracies, for a four factor, trilinear model. We chose a four-factor model based upon our knowledge of the chemical system and rank analysis of the three modes. For our Fast Tucker1 method, we chose to use double the number of our expected rank, or to keep eight factors. This choice was based in part on several analyses of the data and the reproducibility of the results.

The times required to complete the analysis using the full data PARAFAC and the Fast Tucker1 methods were 4702 and 79 s, respectively. Thus, we see a factor of 60 advantage in this case. Results for the comparison are shown in Figures 1–4.

While we reserve judgment on the overall quality of the results in terms of their physical meaning, we find that the two methods produced highly correlated, comparable results. In fact, the smallest angle cosine, also called the uncorrected correlation coefficient [13] is 0.9970 between Factor 1 of the full data method and the Fast Tucker1 Method. Thus, although there are very small differences in the results, they both provide the same decomposition of the data and, more importantly, the same interpretation.

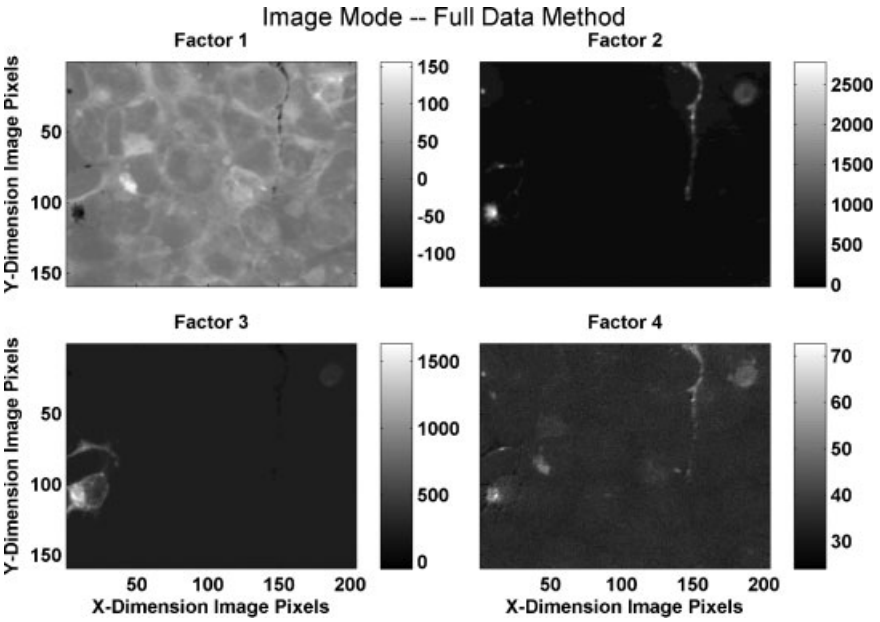

Figure 1. Image mode results for four factor model of HSI-CM data estimated using the full-data method. Calculation time for 1000 iterations, the first 100 of which rigorously nonnegativity constrained, was 4702 s.

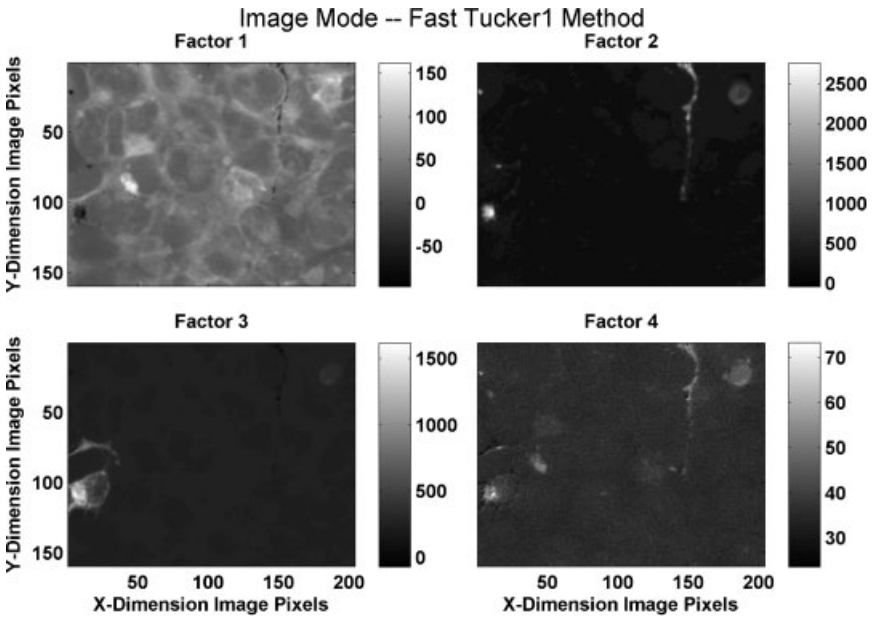

Figure 2. Image mode results for four factor model of HSI-CM data estimated using the Fast Tucker1 Method. Calculation time for 1000 iterations, the first 100 of which rigorously nonnegativity constrained, was 79 s.

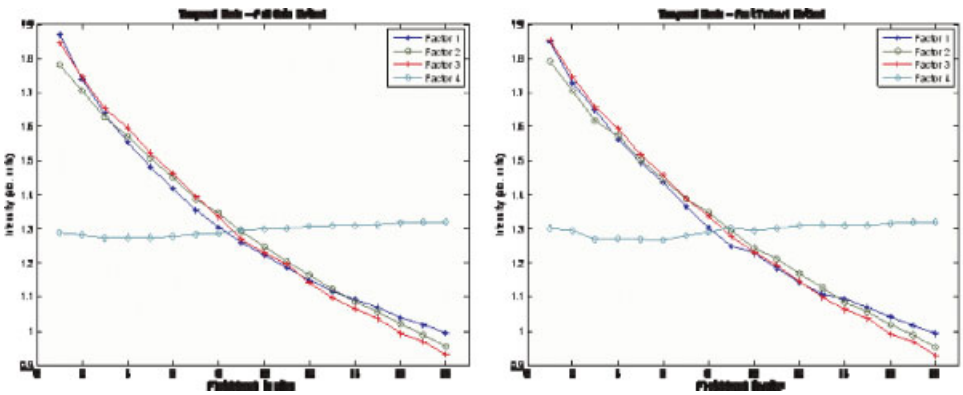

Figure 3. Temporal mode results for four factor model of HSI-CM data estimated using the full-data method (left) and the Fast Tucker1 Method (right). Each step on the *x*-axis represents a subsequent photobleach event. This figure is available in colour online at www.interscience.wiley.com/journal/cem

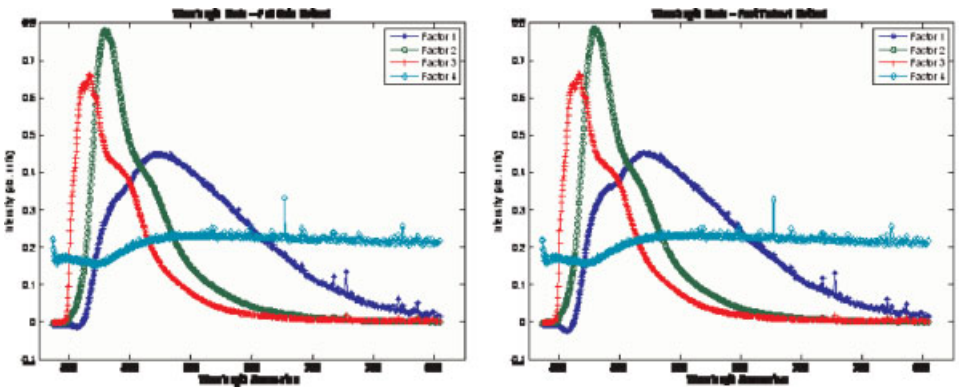

Figure 4. Wavelength mode results for four factor model of HSI-CM data estimated using the full-data method (left) and the Fast Tucker1 Method (right). The four components are interpreted for each factor (1) cell autofluorescence, (2) YFP, (3) GFP and (4) characteristic readout 'dark' shape from EMCCD detector. This figure is available in colour online at www.interscience.wiley.com/journal/cem

7. CONCLUSIONS

We have presented a fast, convenient PARAFAC algorithm based on the core matrix method that can be used with nonnegativity constraints. The method allows trilinear decomposition of very large data sets without employing extraordinary computational resources. We have demonstrated a substantial performance advantage for large data sets and an insignificant overhead penalty for small data sets.

Acknowledgements

The authors thank Dr Allan R. Brasier and Ping Liu of the University of Texas Medical Branch for the fixed cells that were imaged and analyzed in this work, Dr Michael Sinclair of Sandia National Laboratories for assistance with the HSI-CM, and Howland D.T. Jones of Sandia National Laboratories for use of data preprocessing software. We also thank Brett Bader of Sandia National Laboratories for his valuable comments on the original manuscript. This work was supported by Sandia National Laboratories and the Department of Energy. Sandia is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-ACO4-94AL85000.

REFERENCES

1. Wold S, Berglund A, Kettaneh N. New and old trends in chemometrics. How to deal with the increasing data volumes in R&D&P (research, development and production)—with examples from pharmaceutical research and process modeling. *J. Chemom.* 2002; **16**: 377–386.
2. Kowalski BR, Seasholtz MB. Recent developments in multivariate calibration. *J. Chemom.* 1991; **5**: 129–145.
3. Keenan MR, Timlin JA, Van Benthem MH, Haaland DM. Algorithms for constrained linear unmixing with application to the hyperspectral analysis of fluorophore mixtures. *Proc. SPIE Int. Soc. Opt. Eng.* 2002; **4816**: 193–202.
4. Sinclair MB, Haaland DM, Timlin JA, Jones HDT. Hyperspectral confocal microscope. *Appl. Opt.* 2006; **45**: 6283–6291.
5. Grahn H, Geladi P (eds). Techniques and Applications of Hyperspectral Image Analysis. John Wiley & Sons Ltd: Chichester, West Sussex, England, 2007.
6. Harshman RA. Foundations of the PARAFAC procedure: model and conditions for an 'explanatory' multi-mode factor analysis. *UCLA Working Papers Phonetics* 1970; **16**: 1–84.
7. *MATLAB Ver. 7.3.0.267 (R2006b)*, The MathWorks, Inc., Natick, MA, 2006.
8. Kroonenberg PM, de Leeuw J. Principal component analysis of three-mode data by means of alternating least squares algorithms. *Psychometrika* 1980; **45**: 69–97.
9. Tucker LR. Some mathematical notes on three-mode factor analysis. *Psychometrika* 1966; **31**: 279–311.
10. Tauler R, Barcelo D. Multivariate curve resolution applied to liquid chromatography-diode array detection. *Trends Anal. Chem.* 1993; **12**: 319–327.
11. Van Benthem MH, Keenan MR, Haaland DM. Application of equality constraints on variables during alternating least squares procedures. *J. Chemom.* 2002; **16**: 613–622.
12. Tauler R. Multivariate curve resolution applied to 2nd-order data. *Chemom. Intell. Lab. Syst.* 1995; **30**: 133–146.
13. Mitchell BC, Burdick DS. Slowly converging PARAFAC sequences: swamps and two-factor degeneracies. *J. Chemom.* 1994; **8**: 155–168.
14. Bro R. *Dissertation*, University of Amsterdam, 1998.
15. Kiers HAL. Towards a standardized notation and terminology in multiway analysis. *J. Chemom.* 2000; **14**: 105–1122.
16. Van Loan CF. The ubiquitous Kronecker product. *J. Comput. Appl. Math.* 2000; **123**: 85–100.
17. Smilde A, Bro R, Geladi P. Multi-way Analysis: Applications in the Chemical Sciences. John Wiley and Sons: New York, 2004.
18. Tomasi G. *In practical and computational aspects in chemometric data analysis. Thesis*, Frederiksberg, 2006, 195–224.
19. DeLathauwer L, DeMoor B, Vandewalle J. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl.* 2000; **21**: 1253–1278.
20. Bro R, Andersson CA. Improving the speed of multiway algorithms part II: compression. *Chemom. Intell. Lab. Syst.* 1998; **42**: 105–113.
21. Appellof CJ, Davidson ER. Strategies for analyzing data from video fluorometric monitoring of liquid chromatographic effluents. *Anal. Chem.* 1981; **53**: 2053–2056.
22. Andersson CA, Bro R. Improving the speed of multi-way algorithms part I: Tucker3. *Chemom. Intell. Lab. Syst.* 1998; **42**: 93–103.
23. Van Benthem MH, Keenan MR. Fast algorithm for the solution of large-scale non-negativity-constrained least squares problems. *J. Chemom.* 2004; **18**: 441–450.
24. Bro R, DeJong S. A fast non-negativity-constrained least squares algorithm. *J. Chemom.* 1997; **11**: 393–401.
25. Keenan MR, Kotula PG. Accounting for Poisson noise in the multivariate analysis of ToF-SIMS spectrum images. *Surf. Interface Anal.* 2004; **36**: 203–212.

APPENDIX

```
function [iSr,jTr,kCr,RMSE,iter] = parafac_t1(F,q,iSr,jTr,kCr,thresh,maxit)
  % PARAFAC_T1 - Three mode decomposition using Tucker1 (T1) PARAFAC-ALS.
  % Demonstration code best if modes are of increasing size, i < k < j
  % If needed data can be reorganized using Matlab function "permute"
  % INPUT ARGUMENTS
  % F:      Data entered as an (i)F(jk) with i < k < j
  % q:      Rank of the T1 PCA decomposition of the data
  % iSr, jTr and kCr:      Initial estimates for iSr, jTr and kCr modes
  % thresh:      Termination on RMSE difference between consecutive iterations
  % maxiter:      Termination on maximum number of iterations
  % OUTPUT ARGUMENTS
  % iSr,jTr,kCr:      PARAFAC-ALS estimates for iSr, jTr and kCr modes
  % RMSE:      RMS sum of squares of residuals
  % iter:      iteration count
  % M. H. Van Benthem
  % Sandia National Laboratories
  [i,r] = size(iSr);                  % dimension S/i-mode and obtain PARAFAC rank
  [j] = size(jTr,1);                  % dimension T/j-mode
  [k] = size(kCr,1);                  % dimension C/k-mode
  n = min(q,k);                       % obtain PCA rank for k-mode, q <= k
  m = min(q,j);                       % obtain PCA rank for j-mode, q <= j
  q = min(q,i);                       % obtain PCA rank for i-mode, q <= i
  % Compute PCA factors for each mode in turn, while computing core matrix
  for ii = 1:3
    if ii == 1                        % i-mode
      Eind = 1:q;                     % indexing of i-mode PCA loadings
    elseif ii == 2                    % k-mode
      Eind = 1:n;                     % indexing of k-mode PCA loadings
      F = reshape(F',k,q*j);          % reorient core" as kGqj in Equation 31
    else                              % j-mode
      Eind = 1:m;                     % indexing of j-mode PCA loadings
      F = reshape(F',j,q*n);          % reorient core' as jGnq in Equation 32
    end
    [nrows,ncols] = size(F);          % dimensions of data/core
    xpr = nrows > ncols;              % pick the smaller size matrix for xp
    if xpr                            % if fewer columns than rows in F
      FXP = F'*F;                     % form the cross product of data/core (FtF)
    else                              % otherwise fewer rows than columns
      FXP = F*F';                     % form the cross product of data/core (FFt)
    end
    if ii == 1                        % get sum of squares for RMSE calculation
      SSz = trace(FXP);
    end
    [Px,E] = eig(FXP);                % perform eigenanalysis on data/scores
    [E,ee] = sort(diag(E),'descend'); % order by maximum variance (EV)
    Px = Px(:,ee(Eind));              % Choose loadings that maximize variance
    if xpr                            % if used FtF rather than FFt, get the loadings
      Px = (F*Px)*diag(1./sqrt(E(Eind))); % scaled to unit length
    end
    if ii == 1
      iVq = Px;                       % loadings resulting from Equation 25
      F = iVq'*F;                     % perform calculation of Equation 30
    elseif ii == 2
      kUn = Px;                       % loadings resulting from Equation 26
      F = kUn'*F;                     % perform calculation of Equation 31
    else
      jWm = Px;                       % loadings resulting from Equation 25
      F = jWm'*F;                     % perform calculation of Equation 32
      F = reshape(F',q,m*n);          % reorient core as qGmn (now F)
    end
```

```
end
RMSEo = 2; RMSE = 1; iter = 0;          % set up preliminaries for PARAFAC-ALS
CtC = (kCr'*kCr);                       % Compute cross-product of C
TtT = (jTr'*jTr);                       % Compute cross-product of T
nCr = kUn'*kCr;                         % Compute initial estimate of nCr
mTr = jWm'*jTr;                         % Compute initial estimate of mTr
while abs(RMSEo-RMSE) > thresh && iter < maxit
  iter = iter + 1; RMSEo = RMSE;
  % Solve for S-hat
  rOMr = CtC.*TtT;                      % T1 core version of Khatri–Rao in Equation 13
  nqYr = reshape(F,n*q,m)*mTr;          % T1 core version of Equation 14
  qQr = zeros(q,r);                     % T1 core version of Equation 15 (performed in loop)
  for p = 1:r
    qQr(:,p) = reshape(nqYr(:,p),q,n)*nCr(:,p);
  end
  qSr = qQr/rOMr;                       % Compute estimate of qSr as in Equation 33
  qSr = qSr * sparse(diag(1./sqrt(sum(qSr.^2,1)))); % set to unit length
  StS = qSr'*qSr;                       % Compute cross-product of S-hat
  % Solve for C-hat
  rXIr = TtT.*StS;                      % T1 core version of Khatri–Rao in Equation 20
  nPr = zeros(n,r);                     % T1 core version of Equation 18 (performed in loop)
  for p = 1:r
    nPr(:,p) = qSr(:,p)'*reshape(nqYr(:,p),q,n);
  end
  nCr = nPr/rXIr;                       % Compute estimate of nCr as in Equation 34
  nCr = nCr * sparse(diag(1./sqrt(sum(nCr.^2,1)))); % set to unit length
  CtC = nCr'*nCr;                       % Compute cross-product of C-hat
  % Solve for T-hat
  rPIr = (StS).*(CtC);                  % T1 core version of Khatri-Rao in Equation 24
  rZmn = qSr'*F;                        % T1 core version of Equation 21
  mRr = zeros(m,r);                     % T1 core version of Equation 22 (performed in loop)
  for p = 1:r
    mRr(:,p) = nCr(:,p)'*reshape(rZmn(p,:),n,m);
  end
  mTr = mRr/rPIr;                       % Compute estimate of nCr as in Equation 35
  TtT = mTr'*mTr;                       % Compute cross-product of T-hat
  % Compute RMSE for termination
  RMSE = sqrt(SSz-sum(sum(jWm*mTr.*(2*(jWm*mRr)-jWm*mTr*rPIr))));
end
iSr = iVq*qSr;                          % Rotate loadings for estimate of iSr
kCr = kUn*nCr;                          % Rotate loadings for estimate of kCr
jTr = jWm*mTr;                          % Rotate loadings for estimate of jTr
```