Chemometrics and Intelligent Laboratory Systems 24 (1994) 31–42

Chemometrics and intelligent laboratory systems

Speed improvement of multivariate algorithms by the method of postponed basis matrix multiplication Part I. Principal component analysis

Bjørn K. Alsberg *, Olav M. Kvalheim

Department of Chemistry, University of Bergen, Allégt. 41, 5007 Bergen, Norway

(Received 28 October 1993; accepted 14 February 1994)

Abstract

Compression is one way of making analysis of large data matrices faster. Compression is here defined as the case when a large matrix **X** is replaced by a smaller *coefficient* matrix **C**. The coefficients are obtained by least squares fitting to some compression basis. When performing, e.g., principal component analysis (PCA) of **C**, the results are comparable but not equal to the results from analyzing **X**. In this paper we suggest a solution to this problem by rewriting the PCA algorithm in terms of **C** and the compression basis matrices. This has been accomplished by applying a method where speed improvement is achieved by postponing basis matrix calculations in key steps of the PCA algorithm. The method suggested can also be applied to other (but not all kinds of) multivariate algorithms.

1. Introduction

The analysis of large data arrays is emerging as a problem in analytical chemistry. New instruments produce huge quantities of data which need some kind of reduction in order to be practical to analyze. One approach to this problem is *compression*. In the scheme suggested by our laboratory [1,2] the original data table **X** (which may be an *N*-mode data table) is compressed to produce a matrix **C** using B-splines [3,4] or any other suitable compression basis. **C** has a much lower number of elements than **X** and is used instead of the original representation in numerical analyses. Of course, the compression must be such that the loss of significant information is minimized. So far principal component analysis (PCA) [5–7] has been studied and will be the subject of this study also. A drawback of using the **C** matrix instead of **X** is that scores and loading vectors cannot be perfectly transferred to the original domain. This problem can be demonstrated by observing that **X** is assumed to be well described by the smaller coefficient matrix **C** and two B-spline basis sets $\mathbf{B}_1$ and $\mathbf{B}_2$:

$$\tilde{\mathbf{X}} = \mathbf{B}_1\mathbf{C}\mathbf{B}_2^{\mathrm{T}} \qquad (1)$$

In practice we have that $\tilde{\mathbf{X}} \neq \mathbf{X}$, but it is assumed that $\tilde{\mathbf{X}}$ retains the important aspects of the

* Corresponding author. e-mail: alsberg@kj.uib.no.

structure in **X**. The estimation of **C** can be formulated using least squares approximation as:

$$\mathbf{C} = (\mathbf{B}_1^T\mathbf{B}_1)^{-1}\mathbf{B}_1^T\mathbf{X}\mathbf{B}_2(\mathbf{B}_2^T\mathbf{B}_2)^{-1} \tag{2}$$

By decomposing **C** with PCA the following is obtained (the subscript c is used to designate matrices and vectors associated with the compressed representation **C**):

$$\mathbf{C} = \mathbf{T}_c\mathbf{P}_c^T + \mathbf{E}_c \tag{3}$$

which is comparable to:

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathbf{E} \tag{4}$$

The dimensions of the matrices are: Dim(**X**) = $[N \times M]$, Dim(**C**) = $[n \times m]$, Dim(**T**) = $[N \times A]$, Dim(**P**) = $[M \times A]$, Dim($\mathbf{T}_c$) = $[n \times A]$, Dim($\mathbf{P}_c$) = $[m \times A]$, Dim($\mathbf{B}_1$) = $[N \times n]$ and Dim($\mathbf{B}_2$) = $[M \times m]$, $n < N$, $m < M$ and A is the total number of extracted PCA factors. It is tempting to use the basis sets $\mathbf{B}_1$ and $\mathbf{B}_2$ to get estimates of the scores and loading matrices in the original domain as follows:

$$\mathbf{P}_b^T = \mathbf{P}_c^T\mathbf{B}_2^T \tag{5}$$

and

$$\mathbf{T}_b = \mathbf{B}_1\mathbf{T}_c \tag{6}$$

Fig. 1 gives an illustration of the compression process and the different matrices involved.

Eqs. 5 and 6 *do not*, however, produce the true scores and loadings. Neither $\mathbf{T}_b$ nor $\mathbf{P}_b^T$ are orthogonal as required for the true scores and loadings. This can be demonstrated by writing:

$$\mathbf{J}_T = \mathbf{T}_b^T\mathbf{T}_b = \mathbf{T}_c^T(\mathbf{B}_1^T\mathbf{B}_1)\mathbf{T}_c \tag{7}$$

$$\mathbf{J}_P = \mathbf{P}_b^T\mathbf{P}_b = \mathbf{P}_c^T(\mathbf{B}_2^T\mathbf{B}_2)\mathbf{P}_c \tag{8}$$

$\mathbf{J}_T$ will be a diagonal matrix containing the eigenvalues if $\mathbf{B}_1^T\mathbf{B}_1$ is the identity matrix. $\mathbf{J}_P$ will similarly be the identity matrix if $\mathbf{B}_2^T\mathbf{B}_2$ is the

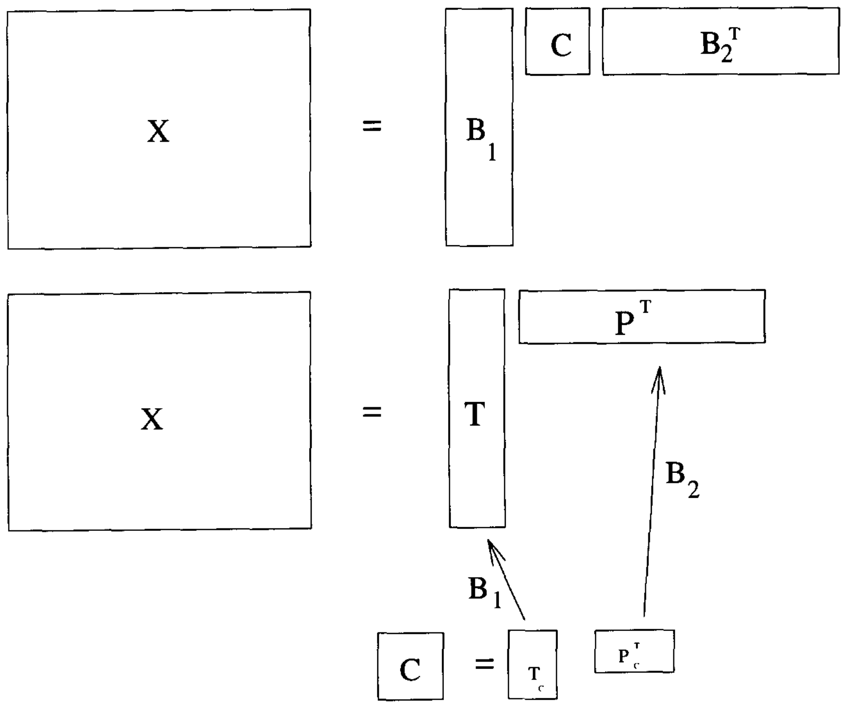

Fig. 1. The various matrices used in the compression and PCA analysis. By utilizing the fact that **X** can be represented by linear combinations of compression bases, **C** can be used instead. It is a problem, however, that the scores and loadings vectors from analysis of **C**, $\mathbf{T}_c$ and $\mathbf{P}_c^T$ cannot be the true estimates of **T** and $\mathbf{P}^T$ of **X**. This is the reason for using PBM-PCA which compensates for this and yet reduces the number of FLOPS.

identity matrix. This is almost never the case. It must be emphasized that orthogonalization/orthonormalization of $\mathbf{T}_b$ and $\mathbf{P}_b^T$ *do not* produce the correct results. Visual comparison of both $\mathbf{T}_c$ with $\mathbf{T}$ and $\mathbf{T}_b$ with $\mathbf{T}$ often reveals similar trends albeit somewhat distorted. The same observation applies for the loading matrices. For many problems this is satisfactory but it is possible to imagine situations where it is not, i.e., where the quantitative information is more important than the qualitative information. The aim of this article is to present a method which can be applied to rewrite algorithms to compensate for the distortion effects observed when using a coefficient matrix $\mathbf{C}$ instead of the original matrix $\mathbf{X}$. In addition, we wish to minimize the number of floating point operations (FLOPS) needed for the analysis. It should be kept in mind that the method of postponed basis matrix multiplication (PBM) cannot be applied to *any* multivariate algorithm. The basic idea of PBM is to separate the basis matrix multiplication parts from expressions included in time consuming iterations. The following criteria must be met for the different expressions in an iteration for the PBM method to work:

- An expression must be expandable in terms of the compression model.
- Each term in a sum must be pre- or/and postmultiplied by the same basis matrices.
- Equations that cannot be written in terms of the compression model can be included in the iteration unless they depend on the *whole* uncompressed input matrix.
- There must be no non-linear operations on the uncompressed input matrix $\mathbf{X}$ or the basis matrices.

2. The method of postponed basis matrix multiplication

The steps of the PBM method are:

1. The data matrix $\mathbf{X}$ at hand is modeled by one or two basis matrices: $\{\mathbf{B}_1, \mathbf{B}_2\}$ and a coefficient matrix $\mathbf{C}$. This means that one of the three possible compression models is possible:

$$\tilde{\mathbf{X}} = \mathbf{B}_1\mathbf{C} \tag{9}$$

$$\tilde{\mathbf{X}} = \mathbf{C}\mathbf{B}_2^T \tag{10}$$

$$\tilde{\mathbf{X}} = \mathbf{B}_1\mathbf{C}\mathbf{B}_2^T \tag{11}$$

   The choice of compression model depends on whether it is possible or necessary to compress along a mode.
   For N-mode arrays the number of basis matrices may be equal to N and there are in general $2^N - 1$ different compression models.
2. Results (e.g., scores and loading vectors from PCA) in the algorithm are assumed to be linear combinations of one or both of the basis matrices (here $\mathbf{B}_1$ or $\mathbf{B}_2$).
3. When corresponding basis matrices can be found at both sides of the equation sign as pre- and/or postmultiplication this multiplication is *postponed* until the end of an iteration. This is the case for PCA where an iteration must converge for each factor.
4. The new algorithm produces vectors and matrices which are comparable to the $\mathbf{C}$ matrix in size. The basis matrices are pre-or post-multiplied with the result vectors/matrices to obtain the correct results.

2.1. The PBM method applied to the NIPALS algorithm

The main idea of this paper is to utilize the compressed description of the data matrix $\mathbf{X}$ in the nonlinear iterative partial least squares (NIPALS) algorithm such that most of the time consuming steps in the algorithm are performed effectively on the *coefficients* and at the same time avoiding the undesirable transformations of the scores and loadings.

The standard NIPALS algorithm is here included because it will clarify how each step is transformed by the PBM method. The following steps are repeated until convergence for each factor:

$$\boldsymbol{p}_0^T = \boldsymbol{t}_0^T\mathbf{X} \tag{12}$$

$$\boldsymbol{p}_1^T = \frac{\boldsymbol{p}_0^T}{(\boldsymbol{p}_0^T\boldsymbol{p}_0)^{1/2}} \tag{13}$$

$$\boldsymbol{t}_1 = \mathbf{X}\boldsymbol{p}_1 \tag{14}$$

Here subscripts 0 and 1 are used to distinguish between current (1) and previous (0) iteration steps of the two estimates of the scores and loading vectors.

The norm $\| \boldsymbol{t}_0 - \boldsymbol{t}_1 \|$ is used to decide on the termination of the iteration for a given factor. Subsequently the obtained score and loading vectors are subtracted from the **X** matrix:

$$\mathbf{X} = \mathbf{X} - \boldsymbol{t}_1 \boldsymbol{p}_1^{\mathrm{T}} \tag{15}$$

It is assumed that

$$\tilde{\mathbf{X}} = \mathbf{B}_1 \mathbf{C} \mathbf{B}_2^{\mathrm{T}} \tag{16}$$

$$\boldsymbol{p}^{\mathrm{T}} = \boldsymbol{v}^{\mathrm{T}} \mathbf{B}_2^{\mathrm{T}} \tag{17}$$

$$\boldsymbol{t} = \mathbf{B}_1 \boldsymbol{u} \tag{18}$$

where $\boldsymbol{t}$ is the score vector and $\boldsymbol{p}^{\mathrm{T}}$ the loading vector of a factor. Hereafter it will be assumed that $\tilde{\mathbf{X}} = \mathbf{X}$. The vectors $\boldsymbol{u}$ and $\boldsymbol{v}^{\mathrm{T}}$ are the corresponding compressed representations. It is important to emphasize here that $\boldsymbol{u}$ and $\boldsymbol{v}^{\mathrm{T}}$ are *not* in general equal to $\boldsymbol{t}_{\mathrm{c}}$ and $\boldsymbol{p}_{\mathrm{c}}^{\mathrm{T}}$ described in the introduction. That is the reason for not adopting those names for the vectors.

Note that score and loading vectors without a numerical subscript are assumed to have converged for the current factor.

The first step to be investigated is Eq. (12). The first estimate of the vector $\boldsymbol{u}$ can, e.g., be a column in **C**. In the following we rewrite step by step each expression in the traditional NIPALS algorithm presented in Eqs. (12)–(14) in terms of the compression model matrices and vectors ($\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{C}$, $\boldsymbol{u}$, $\boldsymbol{v}$). The equations are formulated such that the $\mathbf{B}_1$ matrix is always premultiplied and the $\mathbf{B}_2$ matrix is always postmultiplied in an expression. This is to follow the compression model for the **X** matrix in Eq. (16).

The first estimate of $\boldsymbol{p}$ for factor a (see Eq. (12)) is:

$$\boldsymbol{p}_0^{\mathrm{T}} = \boldsymbol{u}_0^{\mathrm{T}} (\mathbf{B}_1^{\mathrm{T}} \mathbf{B}_1) \mathbf{C} \mathbf{B}_2^{\mathrm{T}} \tag{19}$$

where $\boldsymbol{t}_0^{\mathrm{T}} = \boldsymbol{u}_0^{\mathrm{T}} \mathbf{B}_1^{\mathrm{T}}$ and $\mathbf{X} = \mathbf{B}_1 \mathbf{C} \mathbf{B}_2^{\mathrm{T}}$ have been inserted into the formula $\boldsymbol{p}_0^{\mathrm{T}} = \boldsymbol{t}_0^{\mathrm{T}} \mathbf{X}$. It is nothing more than a reformulation of the existing equation. Eq. (19) is equal to

$$\boldsymbol{p}_0^{\mathrm{T}} = \boldsymbol{v}_0^{\mathrm{T}} \mathbf{B}_2^{\mathrm{T}} \tag{20}$$

where

$$\boldsymbol{v}_0^{\mathrm{T}} = \boldsymbol{u}_0^{\mathrm{T}} (\mathbf{B}_1^{\mathrm{T}} \mathbf{B}_1) \mathbf{C} \tag{21}$$

The scaling of the loading function is (reformulation of Eq. (13)):

$$\boldsymbol{p}_1^{\mathrm{T}} = \frac{\boldsymbol{v}_0^{\mathrm{T}}}{\left[\boldsymbol{v}_0^{\mathrm{T}} (\mathbf{B}_2^{\mathrm{T}} \mathbf{B}_2) \boldsymbol{v}_0\right]^{1/2}} \mathbf{B}_2^{\mathrm{T}} \tag{22}$$

This is equal to

$$\boldsymbol{p}_1^{\mathrm{T}} = \boldsymbol{v}_1^{\mathrm{T}} \mathbf{B}_2^{\mathrm{T}} \tag{23}$$

where

$$\boldsymbol{v}_1^{\mathrm{T}} = \frac{\boldsymbol{v}_0^{\mathrm{T}}}{\left[\boldsymbol{v}_0^{\mathrm{T}} (\mathbf{B}_2^{\mathrm{T}} \mathbf{B}_2) \boldsymbol{v}_0\right]^{1/2}} \tag{24}$$

The new estimate of the score vector can be written as (reformulation of Eq. (14)):

$$\boldsymbol{t}_1 = \mathbf{B}_1 \mathbf{C} (\mathbf{B}_2^{\mathrm{T}} \mathbf{B}_2) \boldsymbol{v}_1 \tag{25}$$

This is equal to

$$\boldsymbol{t}_1 = \mathbf{B}_1 \boldsymbol{u}_1 \tag{26}$$

where $\boldsymbol{u}_1$ is:

$$\boldsymbol{u}_1 = \mathbf{C} (\mathbf{B}_2^{\mathrm{T}} \mathbf{B}_2) \boldsymbol{v}_1 \tag{27}$$

By observing closely the steps in the NIPALS algorithm it is observed that the large matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ do not participate in the key steps of the equations. All the equations have one of the following forms:

$$\mathbf{B}_1 \mathrm{array}_1 = \mathbf{B}_1 \mathrm{array}_2 \tag{28}$$

$$\mathrm{array}_1 \mathbf{B}_2^{\mathrm{T}} = \mathrm{array}_2 \mathbf{B}_2^{\mathrm{T}} \tag{29}$$

$$\mathbf{B}_1 \mathrm{array}_1 \mathbf{B}_2^{\mathrm{T}} = \mathbf{B}_1 \mathrm{array}_2 \mathbf{B}_2^{\mathrm{T}} \tag{30}$$

where ‘array’ means either a vector or a matrix. When such equations are involved in an iteration the pre- and/or postmultiplications of the basis matrix are redundant and thus the total consumption of FLOPS is reduced by postponing the multiplication to after the iteration has terminated. Figuratively speaking they have a property similar to enzymes: necessary for the reaction but not taking part in the reaction itself. The steps containing the $\mathbf{B}_1^{\mathrm{T}} \mathbf{B}_1$ and $\mathbf{B}_2^{\mathrm{T}} \mathbf{B}_2$ matrices will of course require more FLOPS than using the coefficients alone. This is the price which must be

Table 1
Central steps in the PBM method applied to the NIPALS algorithm. Here $\mathbf{\Gamma}_1 = \mathbf{B}_1^T\mathbf{B}_1$ and $\mathbf{\Gamma}_2 = \mathbf{B}_2^T\mathbf{B}_2$

Traditional NIPALS	NIPALS with compression model	Necessary kernel (PBM)
$\boldsymbol{p}_0^T = \boldsymbol{t}_0^T \boldsymbol{X}$	$\boldsymbol{v}_0^T \mathbf{B}_2^T = \boldsymbol{u}_0^T \mathbf{\Gamma}_1 \mathbf{C} \mathbf{B}_2^T$	$\boldsymbol{v}_0^T = \boldsymbol{u}_0^T \mathbf{\Gamma}_1 \mathbf{C}$
$\boldsymbol{p}_1^T = \frac{\boldsymbol{p}_0^T}{\lVert \boldsymbol{p}_0^T \rVert}$	$\boldsymbol{v}_1^T \mathbf{B}_2^T = \frac{\boldsymbol{v}_0^T}{(\boldsymbol{v}_0^T \mathbf{\Gamma}_2 \boldsymbol{v}_0)^{1/2}} \mathbf{B}_2^T$	$\boldsymbol{v}_1^T = \frac{\boldsymbol{v}_0^T}{(\boldsymbol{v}_0^T \mathbf{\Gamma}_2 \boldsymbol{v}_0)^{1/2}}$
$\boldsymbol{t}_1 = \mathbf{X} \boldsymbol{p}_1^T$	$\boldsymbol{B}_1 \boldsymbol{u}_1 = \mathbf{B}_1 \mathbf{C} \mathbf{\Gamma}_2 \boldsymbol{v}_1$	$\boldsymbol{u}_1 = \mathbf{C} \mathbf{\Gamma}_2 \boldsymbol{v}_1$
$\boldsymbol{t}_0 = \boldsymbol{t}_1$	$\mathbf{B}_1 \boldsymbol{u}_0 = \mathbf{B}_1 \boldsymbol{u}_1$	$\boldsymbol{u}_0 = \boldsymbol{u}_1$
$\mathbf{X} = \mathbf{X} - \boldsymbol{t}_1 \boldsymbol{p}_1^T$	$\mathbf{B}_1 \mathbf{C} \mathbf{B}_2^T = \mathbf{B}_1 \mathbf{C} \mathbf{B}_2^T - \mathbf{B}_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T \mathbf{B}_2^T$	$\mathbf{C} = \mathbf{C} - \boldsymbol{u}_1 \boldsymbol{v}_1^T$

paid in order to get the correct estimates of the reconstructed models.

Table 1 presents the effects of applying the PBM method to the NIPALS algorithm. In the table the traditional NIPALS algorithm is located at the left side, the algorithm rewritten in terms of its basis matrices and coefficient matrix in the middle and to the right side the necessary kernel when the multiplication of the basis matrices is postponed to after all the factor iterations have finished. In the Appendix 6, a MATLAB program is presented which shows an implementation of the PBM-PCA algorithm.

The output from the PBM algorithm is two matrices **U** (scores like matrix) and **V** (loadings like matrix) which do *not* share the orthogonality properties of the traditional NIPALS algorithm. Thus $\mathbf{U}^T\mathbf{U} \neq \mathbf{D}$ where **D** is a diagonal matrix (eigenvalues along the diagonal) and $\mathbf{V}^T\mathbf{V} \neq \mathbf{I}$. On the other hand, however, we have that $\mathbf{T}_h = \mathbf{B}_1\mathbf{U}$ and $\mathbf{P}_h^T = \mathbf{V}^T\mathbf{B}_2^T$ *do* have these properties: $\mathbf{T}_h^T\mathbf{T}_h = \mathbf{U}^T\mathbf{\Gamma}_1\mathbf{U} = \mathbf{D}$ and $\mathbf{P}_h^T\mathbf{P}_h = \mathbf{V}^T\mathbf{\Gamma}_2\mathbf{V} = \mathbf{I}$, where $\mathbf{\Gamma}_i = \mathbf{B}_i^T\mathbf{B}_i$ $i \in \{1, 2\}$. The last projections are much less time consuming than using a lot of computer resources to find the eigenvectors of very large covariance matrices.

2.2. FLOPS estimations

The equations giving the estimate of required FLOPS have been developed to be in concordance with the results obtained by using the *flops* command in MATLAB. Table 2 describes the FLOPS equations for some simple linear algebra operations.

By analyzing the PCA algorithm it was found that the approximate number of FLOPS consumed can be expressed by the following equation:

$$\hat{F}_1 = A[q_a(4NM + 5M) + 3NM] \tag{31}$$

where $\mathrm{Dim}(\mathbf{X}) = [N \times M]$, A is the total number of factors extracted and q_a is the number of iterations per factor.

Table 2
Number of FLOPS required for some example matrix operations

Matrix operation	FLOPS required	Dimensions of matrices
$\mathbf{XY}$	$2nmk$	$\mathrm{Dim}(\mathbf{X}) = [n \times m]$, $\mathrm{Dim}(\mathbf{Y}) = [m \times k]$
$\mathbf{X}\boldsymbol{y}$	$2nm$	$\mathrm{Dim}(\mathbf{X}) = [n \times m]$, $\mathrm{Dim}(\boldsymbol{y}) = [m \times 1]$
$\boldsymbol{y}^T\boldsymbol{y}$	$2m$	$\mathrm{Dim}(\boldsymbol{y}) = [m \times 1]$
$\mathbf{X} + \mathbf{X}$	nm	$\mathrm{Dim}(\mathbf{X}) = [n \times m]$
$\mathbf{X} = \mathbf{X} - \boldsymbol{t}\boldsymbol{p}^T$	$3nm$	$\mathrm{Dim}(\mathbf{X}) = [n \times m]$, $\mathrm{Dim}(\boldsymbol{t}) = [n \times 1]$ $\mathrm{Dim}(\boldsymbol{p}) = [m \times 1]$,
$(\mathbf{XY})\mathbf{Z}$	$2nk(m + r)$	$\mathrm{Dim}(\mathbf{X}) = [n \times m]$, $\mathrm{Dim}(\mathbf{Y}) = [m \times k]$ $\mathrm{Dim}(\mathbf{Z}) = [k \times r]$
$\mathbf{X}(\mathbf{YZ})$	$2mr(k + n)$	

The approximate FLOPS consumption for the PBM-PCA algorithm is expressed as

$$\hat{F}_2 = A\left[q_a(4nm + 2m^2 + 3m + 2) + 3nm + 2n^2m + 2nm^2\right] + 2n^2N + 2m^2M + 2n^2m + 2nm^2 \qquad (32)$$

where Dim(C) = $[n \times m]$. As can be seen the dimensions of the coefficient matrix must be much smaller than the original matrix dimensions in order to obtain significant FLOPS ratios (the number of FLOPS required for standard NIPALS algorithm divided by the number of FLOPS required for the PBM algorithm). Based on these equations it is possible to get an approximate FLOPS ratio given the n, m, N, M, A, q_a variables. If the following simplifications are made that $n = m$, $N = M$, $A = 5$ and $q_a = 10$ it is possible to investigate the properties of the PBM-PCA algorithm by selecting ranges for N and n. Here the additional simplification has been made that $n = N/r$ where r signifies the compression and is assumed to be the same for both modes. The following ranges were selected: $N \in [200, 1000]$, $r \in [2, 30]$. The results are presented in Fig. 2.

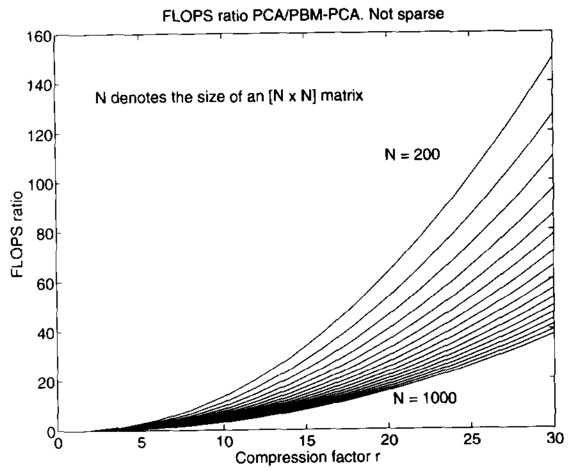

Fig. 2. FLOPS ratio between PCA and PBM-PCA algorithms when using various values for N and r. N denotes the size of a matrix **X** of dimensions $[N \times N]$, r is the compression along each mode. The compressed **C** has dimensions $[N/r \times N/r]$. No sparse representation is assumed here and thus the FLOPS ratios are smaller.

2.3. *Sparse representations*

Significant reductions in the FLOPS requirements for PBM algorithms can be accomplished by using *sparse* technology [8]. This is a standard way of speeding up algorithms which operate on matrices containing a large number of zero elements. The main idea behind sparse technology is to efficiently perform matrix operations on the nonzero elements only. The storage of the matrices are not in arrays but in lists of nonzero elements with information about their values and positions in the array. B-spline compression basis matrices are to some extent sparse in their structure and this can be utilized to make the PBM algorithms run faster. The multiplication of the $\mathbf{\Gamma}_i$ matrices in the PBM methods will be the largest contributor to the increase in FLOPS compared to just using the coefficient matrix on standard methods.

The density d_i of an array can be defined as the number of nonzero elements divided by the total number of elements in the array. It will always be such that $d_i \in [0,1]$ where $d_i = 1$ represents a *full* matrix with no zeros and the sparse representation will not reduce the required number of FLOPS for such cases. In the equations for the PBM algorithm, densities of two different matrix types will be considered: (i) d_i which is the density of the compression matrix $\mathbf{B}_i$; (ii) g_i which is the density of the Grammian matrix $\mathbf{B}_i^T\mathbf{B}_i$.

It was found that the FLOPS requirements for different matrix operations of sparse matrices were dependent on the density of the matrices involved. If we take the first example in Table 2 it would look like

$$2nmkd_xd_y \qquad (33)$$

where d_x is the density of **X** and d_y is the density of **Y**.

2.4. *FLOPS estimations of PBM with sparse representation*

For B-spline bases their Grammian matrices are diagonally dominant which results in several matrix elements of zero value. An example illustrates the saving in FLOPS using sparse representation. A B-spline basis set with dimensions Dim(**B**) = [991 × 36] was constructed from a homogeneous knot vector with polynomial degree 3. The number of FLOPS consumed for the $\mathbf{\Gamma} = \mathbf{B}^T\mathbf{B}$ operation without sparse technique was 2568672. The number of FLOPS with sparse representation was 31204, i.e., the sparse operations required 82.3 times less FLOPS! In the PBM method the $\mathbf{\Gamma}$ matrix is a part of the projection of vectors. A vector projection $v^T\mathbf{\Gamma}$ required 2592 FLOPS for non-sparse and 480 for the sparse representation, which is 5.4 times faster.

The FLOPS Eq. (32) including sparsity is

$$\mathscr{F}_2 = A\big[q_a(4nm + 2m^2g_2 + 3m + 2) + 3nm + 2n^2mg_1 + 2nm^2g_2\big] + 2n^2Nd_1^2 + 2m^2Md_2^2 + 2n^2mg_1 + 2nm^2g_2 \quad (34)$$

Where d_1 is the density of basis matrix $\mathbf{B}_1$, d_2 is the density of basis matrix $\mathbf{B}_2$, g_1 the density of matrix $\mathbf{B}_1^T\mathbf{B}_1$ and g_2 the density of matrix $\mathbf{B}_2^T\mathbf{B}_2$. It was found that $g_i \approx 2d_i$. It is now possible to repeat the simulation above with selected values for $\{d_i, g_i\}$. Of course, small enough densities will give rise to enormous FLOPS ratios but it is more interesting to investigate the case when the density is not too small, e.g., $d_1 = d_2 = 0.1$. Fig. 3 is the same simulation as in Fig. 2 with sparse representation. The results are approximately ten times better, i.e., PBM–PCA for this particular choice of densities of the basis matrices will run ten times faster than PBM–PCA without sparse matrix representation.

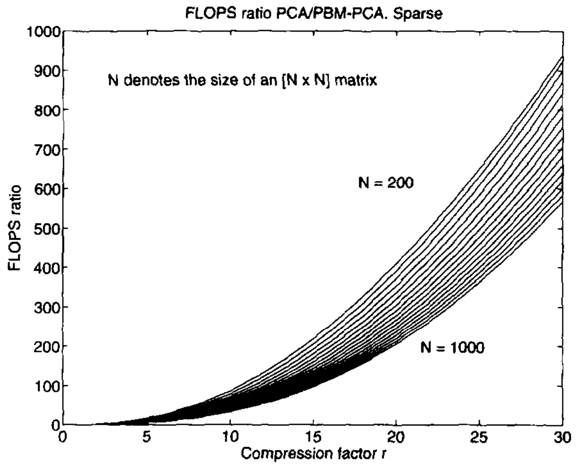

Fig. 3. FLOPS ratio between PCA and PBM–PCA algorithms when using various values for N and r. Here sparse representation is assumed. N denotes the size of a matrix **X** of dimensions $[N \times N]$, r is the compression along each mode. The compressed **C** has dimensions $[N/r \times N/r]$. The following densities $d_1 = d_2 = 0.1$ and $g_1 = g_2 = 0.2$ are assumed.

Table 3
Parameter settings for the two basis matrices used in data set 1. The parameters a, b, c are included in the formula for Gaussian curves

a	b (range)	c	No. of curves
0.5	[−0.8, 1.0]	0.2	7
0.2	[−0.5, 0.6]	0.1	12

3. Results

3.1. Data set 1

This is a data set where the basis set perfectly describes the data, i.e., $\tilde{\mathbf{X}} = \mathbf{X}$. Two basis matrices were constructed using Gaussian curves. A Gaussian curve can be described by the formula $f(x) = ae^{-(x-b)^2/c}$. The different parameter settings for the two Gaussian basis sets are presented in Table 3. Both basis sets are constructed

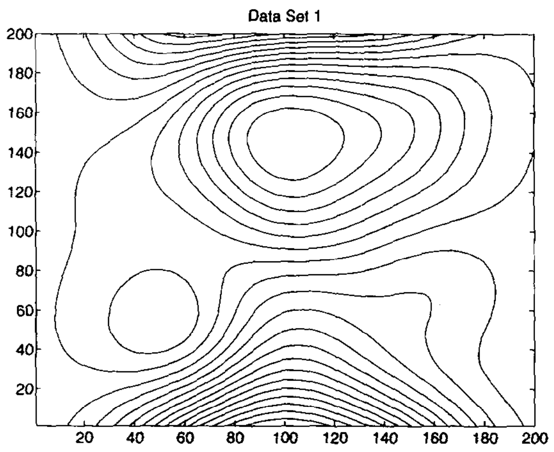

Fig. 4. Data set 1, Dim(**X**) = [200 × 200].

by shifting a single Gaussian curve along the x axis and this explains the range of b values in the table.

Dim(**X**) = [200 × 200] and the dimensions of the coefficient matrix is Dim(**C**) = [7 × 12]. Here the basis set has such a structure that sparse representation will not give any reduction in FLOPS. The original data are depicted in Fig. 4. The number of MFLOPS used for the PCA on the original data set was ca. 5. The corresponding number of MFLOPS consumed using the PBM-PCA was ca. 0.11. The same number of iterations in both algorithms was used to get comparable results. The PBM algorithm is approximately 45 times faster than the original algorithm on this data set. Three sets of matrices were computed. The first set is the scores and loading matrices based on Eqs. 5 and 6, $\mathbf{T}_b$ and $\mathbf{P}_b^T$. The second set of matrices are the estimated scores and loading matrices using the PBM method, $\mathbf{T}_h = \mathbf{B}_1\mathbf{U}$ and $\mathbf{P}_h^T = \mathbf{V}^T\mathbf{B}_2^T$. The third set of matrices are the scores and loading matrices from the PCA of the uncompressed matrix, **T** and $\mathbf{P}^T$. The first and the

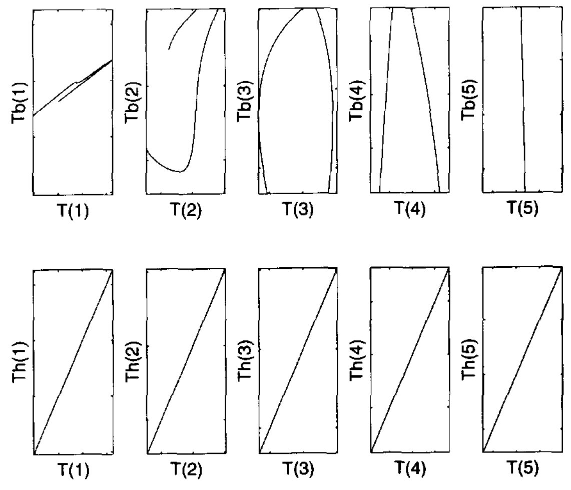

Fig. 5. Results from analysis of data set 1. Upper row shows the comparison between true score vectors (**T** is printed as T(i) in the figure where i is the ith factor) for the first five components versus the estimated score vectors based on standard PCA on the coefficient matrix alone and multiplied by the respective basis matrix ($\mathbf{T}_b$ is printed as Tb(i) in the figure). The lower part shows the comparison between true score vectors versus PBM-estimated score vectors ($\mathbf{T}_h$ is printed as Th(i) in the figure).

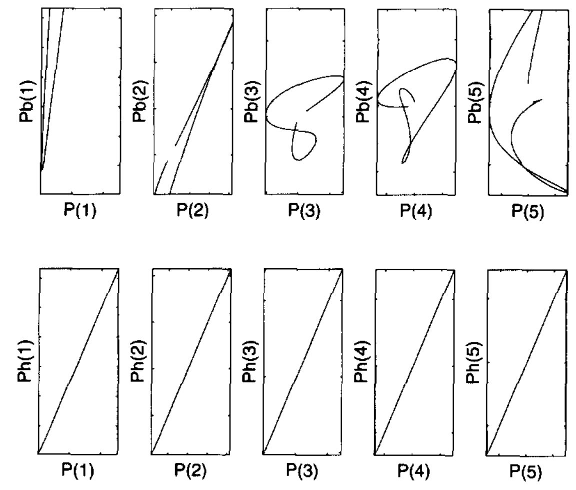

Fig. 6. Results from analysis of data set 1. Upper row shows the comparison between true loading vectors ($\mathbf{P}^T$ is printed as P(i) in the figure where i is the ith factor) for the first five components versus the estimated loading vectors based on standard PCA on the coefficient matrix alone and multiplied by the respective basis matrix ($\mathbf{P}_b^T$ is printed as Pb(i) in the figure). The lower part shows the comparison between true loading vectors versus PBM-estimated loading vectors ($\mathbf{P}_h^T$ is printed as Ph(i) in the figure).

second set of matrices were each compared with the third set. This is illustrated in Figs. 5 and 6. The upper part of Fig. 5 shows **T** versus $\mathbf{T}_b$ for each of the five factors. The lower part of Fig. 5 shows **T** versus $\mathbf{T}_h$ for each of the five factors. As

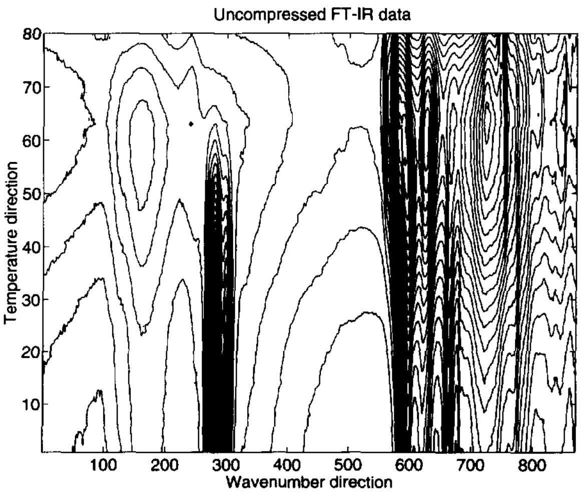

Fig. 7. The temperature-IR data set. Dim(**X**) = [80 × 869].

expected $\mathbf{T}_h$ is equal to $\mathbf{T}$. The upper part of Fig. 6 shows $\mathbf{P}^T$ versus $\mathbf{P}_b^T$ for each of the five factors. The lower part of Fig. 6 shows $\mathbf{P}^T$ versus $\mathbf{P}_h^T$ for each of the five factors. It is observed that the result $\mathbf{P}_h^T$ from the PBM method is equal to $\mathbf{P}^T$.

3.2. Data set 2

The details of the preparation of this data set have been described elsewhere [9]. This is a data matrix (see Fig. 7) obtained from a two-dimensional infrared experiment (temperature versus IR spectrum) of a pyrolysis. The process was started at 200°C and increased to 397.7°C with a step of 2.5°C. The spectral range was 4000–650 cm^{-1}. The dimensions of the data set were [80 × 869].

Before PCA the original $\mathbf{X}$ matrix was prepared by removing the mean. $\mathbf{X}$ was analyzed with traditional PCA using NIPALS. $\mathbf{C}$, $\mathbf{B}_1$ and $\mathbf{B}_2$ were the input arguments to the PBM–PCA algorithm. PBM–PCA has two output arguments: $\mathbf{U}$ and $\mathbf{V}^T$. It is important to remember that none of these two matrices are orthogonal as explained previously in this article. If $\mathbf{U}$ and $\mathbf{V}^T$ are multiplied by the two basis matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ the resulting scores and loading matrices are orthogonal. In Ref. [9] only two principal components were found adequate to explain the $\mathbf{X}$. Even though the importance of later factors (3 and 4) for data set 2 are low they were still extracted to investigate when the PBM approximation was deviating from the true eigenvectors. Fig. 8 shows the scores matrix $\mathbf{T}$ compared to $\mathbf{T}_h$ for each factor. Each score vector in $\mathbf{T}$ is illustrated as a dash-dotted line. For the fourth factor some deviation from the true spectrum is observed. Note that the deviations are due to the fact that the compressed representation is not perfect, i.e., it has nothing to do with the PBM–PCA algorithm itself. One explanation for deviation may be that the noise level is increased for later factors and thus the deviations from the compression model become more pronounced. Fig. 9 shows the loading matrix $\mathbf{P}^T$ compared to $\mathbf{P}_h^T$ for each factor. Each loading vector in $\mathbf{P}^T$ is illustrated as a dash-dotted line. Again some deviations from the true line are observed for the fourth factor. The ordinary PCA used approximately 20 times more FLOPS than PBM–PCA. The FLOPS $F_1 =$

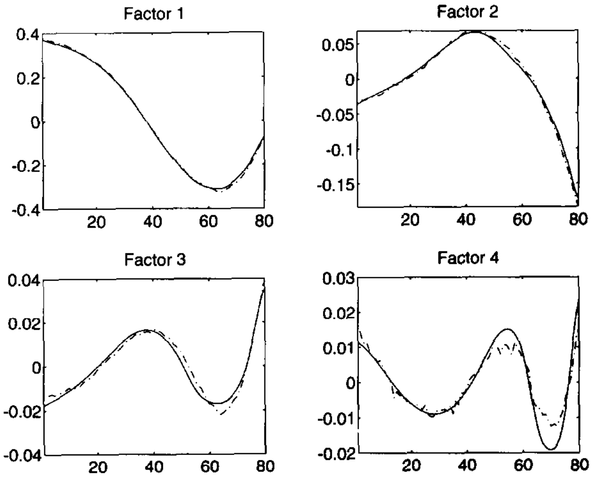

Fig. 8. Comparison of the scores vectors for PCA on uncompressed matrix (dash-dotted lines) and PBM–PCA using compression model (solid lines). Results from analysis of data set 2.

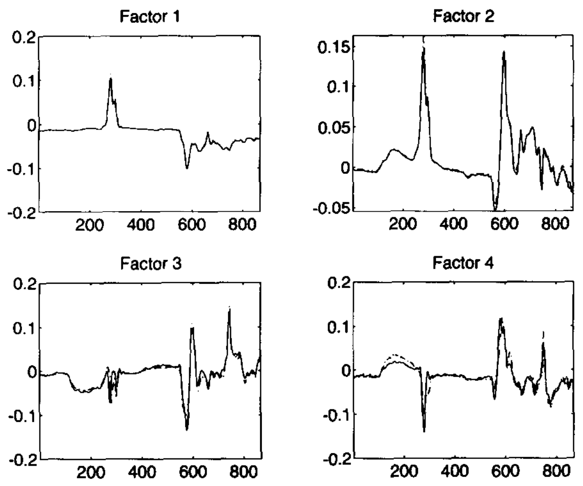

Fig. 9. Comparison of the loadings vectors for PCA on uncompressed matrix (dash-dotted lines) and PBM–PCA using compression model (solid lines). Results from analysis of data set 2.

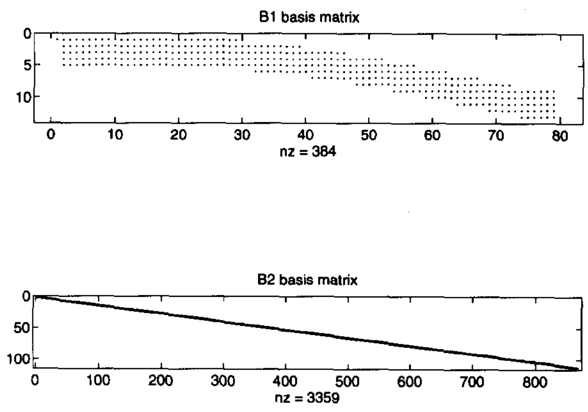

Fig. 10. Sparsity of the two basis matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ used on data set 2. nz stands for the number of nonzero elements in the matrix.

12 131 240 and $\mathscr{F}_2 = 579\,054$ are measured under the same conditions, i.e., the number of iterations per factor was set to 10. This was done in order to get comparable results. The estimated number of FLOPS based on the formula including sparse technology was $\hat{\mathscr{F}}_2 = 463\,750$ which gives a FLOPS ratio of ca. 26. The estimation formula overestimates the gain in FLOPS by using sparse representation. The densities of the basis matrices and their Grammians are: $d_1 = 0.0336$, $d_2 = 0.3692$, $g_1 = 0.0600$ and $g_2 = 0.5740$. Fig. 10 illustrates the sparsity of the two basis matrices $\mathbf{B}_1$ and $\mathbf{B}_2$; only nonzero elements are indicated.

The effect of using sparse matrix representation is significant. Without sparse representation of the basis matrices in data set 2 the PBM algorithm would have used *twice* as many FLOPS as the standard PCA algorithm on the uncompressed **X**! Using **C** instead of **X** in ordinary PCA was approximately 40 times faster [9].

The calculations were performed on a HP 9000/730 with 64 Mbyte RAM and 1.3 Gbyte hard disk. All programs were written in MATLAB using version 4.0.

4. Discussion

The PBM method can be applied to methods where some kernel of compression coefficients can be used without invoking the full basis set directly into the computation. It is assumed that several analyses of the same data set **X** will be done both by PCA or some other similar method (e.g., PLS) which means the compression initially will become more useful for each new computation on the compressed representation. In the previous FLOPS calculations the computations of the $\mathbf{B}_1^T\mathbf{B}_1$ and $\mathbf{B}_2^T\mathbf{B}_2$ matrices were included in the FLOPS count, but these matrices can be obtained from the compression step. The PBM method is of special interest for algorithms of N-mode data arrays. In such algorithms the computational problem is acute. To handle this problem compression may be successful. Part II of this article [10] shows that three-mode PCA can be rewritten using the PBM method. Several of the key steps in the algorithm contains unnecessary manipulation of basis set matrices. The gain in reduced FLOPS is proportional to the compression ratio.

It is also possible to reformulate PLS using the PBM approach. This will probably be most useful for the iterative PLS2 method. It must be stressed for PLS and other regression methods that compression along the object mode must be used with caution. The reason for this is that if a compression along the object mode is selected, a compression of the Y vector/matrix should also be applied. The next problem is whether to choose the same basis set for both the X space and Y space. Selecting different basis matrices for X and Y may cause the whole reduction of the workload to vanish. On the other hand it may be unrealistic to assume that both the X and Y space can be well modeled by the same basis set.

5. Conclusion

The PBM method is capable of solving the problem with distorted models from analysis of a compressed representation. The cost is using an increased number of FLOPS. This cost can be significantly minimized by using sparse representations of basis matrices if they have a large number of zero elements. B-spline bases have a diagonally dominant structure which often gives rise to a large number of zero elements.

In addition it is necessary to rewrite the actual algorithm to make use of the PBM method.

Acknowledgement

BKA wishes to thank the Royal Norwegian Council for Scientific and Industrial Research (NTNF) for financial support.

Appendix. PBM–PCA algorithm in MATLAB 4.0 code

```
function [uu,vv]=
pbmpca(C,A,B,comp)
% [uu,vv]=pbmpca(C,A,B,comp);
% A=is the basis set for the rows
% B=is the basis set for the
% columns
% It is assumed that
% X=A*C*B'
% If X=[N×M], C=[n×m] then
% A=[N×n], B=[M×m]
% comp is the number of principal
% components

A=sparse(A);
B=sparse(B);
% The sparse command converts A and
% B into sparse representations

G1=A'*A;
Ge=B'*B;
% These multiplications can be car-
% ried out using approximately half
% the no. of FLOPS by utilizing the
% fact that G1 and G2 are symmet-
% ric. This is not shown here (but
% taken into consideration for the
% FLOPS formulas presented in the
% text) since it will complicate
% the code. In addition, even if
% the no. of FLOPS consumed can be
% made less, the code presented
% here is still faster in execution
% time because MATLAB so ineffi-
% ciently handles FOR loops
Q1=G1*C;
Q2=C*G2;
[a,b]=size(C);
stop=3;
for i=1:comp
 av=std(C);
 [av2,indx]=sort(av);
 u0=C(:,indx(b));
  while stop>1,
   v0=u0'*Q1;
   v1=v0*(v0*G2*v0')^(-1/2);
   u1=Q2*v1';
   if norm(u1-u0)<0.00005,
    uu(:,i)=u1;
    vv(i,:)=v1;
    stop=0;
   end;
   u0=u1;
  end;
 stop=3;
 C=C-u1*v1;
 Q1=G1*C;
 Q2=C*G2;
end;
```

References

[1] B.K. Alsberg, Representation of spectra by continuous functions, *Journal of Chemometrics*, 7 (1993) 177–193.

[2] B.K. Alsberg and O.M. Kvalheim, Compression of *n*th-order data arrays by B-splines. Part 1. Theory, *Journal of Chemometrics*, 7 (1993) 61–73.

[3] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design, a Practical Guide*, Academic Press, Boston, MA, 2nd edn., 1990.

[4] W. Cheney and D. Kincaid, *Numerical Mathematics and Computing*, Brooks/Cole, Monterey, CA, 1980.

[5] H. Martens and T. Næs, *Multivariate Calibration*, Wiley, New York, 1989.

[6] I. Cowe and J.W. McNicol, The use of principal component in the analysis of near-infrared spectra, *Applied Spectroscopy*, 39 (1985) 257–266.

[7] S. Wold, K. Esbensen and P. Geladi, Principal component analysis, *Chemometrics and Intelligent Laboratory Systems*, 2 (1987) 37–52.

[7] S. Wold, K. Esbensen and P. Geladi, Principal component analysis, *Chemometrics and Intelligent Laboratory Systems*, 2 (1987) 37–52.

[8] S. Pissanetsky, *Sparse Matrix Technology*, Academic Press, London, 1984.

[9] B.K. Alsberg, E. Nodland and O.M. Kvalheim, Compression of *n*th-order data arrays by B-splines. Part 2. Application to second-order FT-IR spectra, *Journal of Chemometrics*, 8 (1994) 127–145.

[10] B.K. Alsberg and O.M. Kvalheim, Speed improvement of multivariate algorithms by the method of postponed basis matrix multiplication. Part II. Three-mode principal component analysis, *Chemometrics and Intelligent Laboratory Systems*, 24 (1994) 43–54.