Development of methods for improved storage and faster computation of large two- and three-mode chemical data sets

Bjørn K. Alsberg

Department of Chemistry
University of Bergen

Pieter Kroonenberg
Leiden, 30-11-94

Development of methods for improved storage and faster computation of large two- and three-mode chemical data sets

Bjørn K. Alsberg

Department of Chemistry
University of Bergen

UNIVERSITAS BERGENSIS

Contents

Acknowledgement

The author wishes to thank advisor Professor Olav M. Kvalheim at University of Bergen (UiB) for stimulating support and guidiance in the research. The Royal Norwegian Council for Scientific and Industrial Research (NTNF) is thanked for the financial support. Dr. Knut J. Børve and cand.scient. Egil Nodland at UiB are thanked for several of the data sets used. Professor John H. Husøy at Rogaland University Center is thanked for helpful discussion about compression techniques in general. Professor Tormod Næs at University of Oslo is thanked for valuable comments to the programming of the traditional fuzzy c-means clustering algorithm. Professor Rolf Manne is acknowledged for interesting and helpful discussions.

Preface

This thesis is submitted for the degree of dr.scient. at the University of Bergen and consists of seven papers.

In 1991 the author was granted a scholarship at the University of Bergen through the the NTNF Strategic Technology Program (STP) given to Professor Olav M. Kvalheim. The work presented in this thesis was done in the period 1992-1994.

List of papers

Paper I	Alsberg, B. and Kvalheim, O.M. "Compression of nth-order data arrays by B-splines. Part 1: Theory." *Journal of Chemometrics*,7, 61-73 (1993)
Paper II	Alsberg, B., Nodland, E. and Kvalheim, O.M. "Compression of nth-order data arrays by B-splines. Part 2: Application to second-order FT-IR spectra". *Journal of Chemometrics*, 8, 127-145 (1994).
Paper III	Alsberg, B. and Kvalheim, O.M. "Compression of three-mode data arrays by B-splines prior to three-mode principal component analysis (PCA)", *Chemometrics and Intelligent Laboratory Systems*, 23 (1994), 29-38
Paper IV	Alsberg B. and Kvalheim O.M. "Speed improvement of multivariate algorithms by the postponing of basis matrix multiplication method. Part I. Principal component analysis", *Chemometrics and Intelligent Laboratory Systems*, 24 (1994) 31-42.
Paper V	Alsberg B. and Kvalheim O.M., "Speed improvement of multivariate algorithms by the postponing of basis matrix multiplication method. Part II. Three-mode principal component analysis", *Chemometrics and Intelligent Laboratory Systems*, 24 (1994) 43-54.
Paper VI	Alsberg, B. "A diagram notation for N-mode array equations". Submitted to *Psychometrika*.
Paper VII	Alsberg, B. " Fuzzy c-means clustering of data sets with large number of variables". Accepted for publication in *Journal of Computational Chemistry*.

Chapter 1

Introduction

Large amounts of data is emerging as an important problem in chemistry. The cost for extracting a large number of variables measured for each sample is becoming lower. Instruments producing N-mode arrays are already beginning to appear in chemistry laboratories. In e.g. NMR spectroscopy the introduction of multipulse techniques such as n-dimensional versions COSY and NOESY, has made it possible to perform proton and ^{13}C assignments more efficiently than provided by one-dimensional techniques [1]. Hyphenated chromatography, e.g. with the acquisition of full UV and IR spectra at different time intervals or temperatures is today an important technique for the analytical chemist [2,3]. In addition to analytical instruments, different theoretical methods can also give rise to a large amount of data. One particular example is the many conformations arising from molecular dynamics simulations of proteins and polypeptides.

In order to extract information from data, techniques found in *chemometrics* can provide the investigator with tools for understanding the underlying structure of the data set. By using multivariate methods such as principal components analysis (PCA) [4–6], principal component regression (PCR) [4] and partial least squares regression (PLS) [7,8] it is possible to model the data structure using latent variables. Even though the algorithms may be efficient for interpretational purposes they can be unnecessarily computer demanding. One particular class of computer intensive methods are N-mode techniques (e.g. N-mode principal component analysis).

Usually the data analytical process is interactive where the investigator can very rapidly see the results of the treatment of the data set. When the data set is very large this process will slow down and sometimes the time spent in waiting for the results is prohibitively long. A simple but expensive approach to the problem is to buy a larger computer or more memory. For several cases this will be impractical and cannot be done each time a too large data set shows up. In this thesis the compression approach will be used for most of the problems. The basic idea is very simple: A cost efficient pretreatment compresses the original data array into a smaller array which is subsequently inserted into the chosen data analytical algorithm. The approach is not without problems and ways of solving them are discussed.

Handling of large data sets can be divided into two major subproblems: Storage and computation. There are several strategies for compression of data, some of them are discussed in the next section. But there is a problem that the compressed represention in general must be *uncompressed* in order to be subjected to numerical analysis. Usually the compressed representation itself cannot be used directly without transferring it back into the original representation. When the large data array is uncompressed into the original large representation again there is still a computational problem. This thesis attempts to achieve the best of both worlds: An efficient storage of the data and computation on the compressed representation only, without the need to uncompress to the original array dimensions. It is assumed that the compression is initially performed to store large amount of data and that the compressed representation will be analyzed later several times by different techniques. It is therefore not imposed strict requirements for the computational efforts involved in the compression stage itself.

The computational problem can also be partly solved by writing more efficient algorithms. In spite of the fact that this will not be the major focus of the thesis, the last chapter deals with an example where rewriting of the fuzzy c-means clustering algorithm produced large savings in floating point operations (FLOPS) when the number of variables is much larger than the number of objects in the data matrix studied.

1.1 Examples of compression techniques

Several schemes have been constructed for compression of text, images and sound-signals. A few common compression methods found in electrical engineering and signal processing are described below. Not all of the compression methods presented are suitable for chemical data.

1.1.1 Run length coding

This is a lossless[1] coding which is based on a very simple principle. It has been successfully applied to compression of binary (black and white) images.

A naïve implementation of this algorithm is best explained by an example. Assume that the following binary sequence should be stored:

0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

This sequence has 34 bits, but can be more efficiently stored by saying 9 zeros, 11 ones and 14 zeros. Using b bits it is possible to address 2^b numbers. Assuming a maximum of 256 repetitions of 1 and 0 we would need 18 bits instead of 34. Of course, if the data set does not have this kind of structure it will be *less* efficient than using the original uncompressed representation.

1.1.2 Huffman coding

This method represents each symbol by a binary code of length inversely proportional to its probability. Assume a file of numbers where each number

[1]This means that the uncompressed representation is identical to the original large representation. In lossy compression the uncompressed representation will be similar but in most cases not equal to the original data.

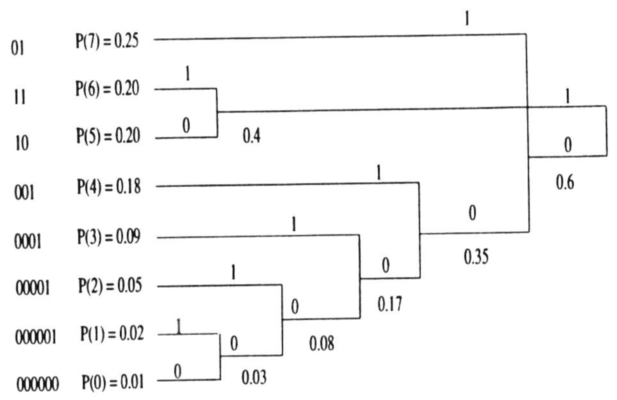


Figure 1.1: Demonstration of the how the Huffman algorithm gives the largest bit patterns to symbols with low probabilities

initially is represented by e.g. 16 bits. It is not optimal to let symbols (numbers) occurring with high frequency be represented by the same number of bits as the symbols with low frequency. An analogous problem occurs in everyday life. We don't use the full name of people we meet every day or are familiar with. Unique identification is often obtained by a small number of letters and we have a tendency to shorten names of friends. For example, "Robert" is called "Bob" and William is called "Bill".

Huffman coding perform this systematically by first determining the occurrences (probabilities) of the different symbols in the sequence. A tree is constructed where each node branches to a 0 or a 1. Each leaf represents a symbol (a number or a character). Each symbol is sorted with respect to their individual probabilities . Those with the highest probabilities are at the top and those with the lowest are at the bottom. Those with higher probabilities will have a shorter path from leaf to top node. An example for a set of eight source symbols is presented in Figure 1.1.

1.1.3 Suband coding

A set of filters splitting the input signal into subbands in the frequency domain is constructed. Each subband is downsampled by a factor equal to the number of subbands. An encoder is made that represents the subbands signals in a bit efficient manner. Generally, low frequency subbands require less bits than high frequency subbands. The highest frequency bands are discarded with the assumption that they contain mostly noise. This method is much used in video and image compression [9].

1.1.4 Quadtrees

The term *quadtree* [10] is a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space. One example of a quadtree approach is to divide subimages into four rectangles recursively. A rectangle is decomposed further if a measure of a rectangle is above a certain limit. Such a measure can be e.g. the standard deviation of all pixels in that subrectangle. The principle that smooth areas need less subdivision than more "rugged" ones is applied. The resulting representation of the image is a tree which can be further subjected to lossless compression schemes as Huffman or Lempel-Ziv. Quadtree segmentation techniques are also used to find contours and edges of objects in images.

1.1.5 Vector quantization (VQ)

A common way to compress images is to divide an image into segments, often of size $[8 \times 8]$ pixel blocks. Such a subimage or segment requires $64b$ bits to store (b bits per pixel). The total number of possible subimages are 2^{64b}. The main idea of VQ is to find a much smaller set of basis segments, called *code vectors*, such that every segment in the original image is replaced by one of the code vectors that *closest resembles the original segment*. The actual storage of the image is now a set of index pointers to the code vectors. If 32 different code vectors is selected, a 5 bits pointer for each segment is needed, instead of the $64b$ bits in the original representation.

The problem in VQ is to choose the best code vectors for a given image or set of images. This problem is related to clustering and segmentation of

a_{i1}	a_{i2}	a_{i3}	a_{i4}	a_{i5}	a_{i6}	p_i
0.50	0.00	0.00	0.50	1.00	1.00	0.33
0.50	0.00	0.00	0.50	1.00	50.00	0.33
0.50	0.00	0.00	0.50	50.00	50.00	0.34

Table 1.1: Table showing the IFS parameters for the Sierpinsky triangle .

the variable space.

1.1.6 Iterative maps/fractal coding

The main idea of using iterated maps (or fractal coding) in compression is to let the parameters for an affine mapping[2] be the code for an image. These parameters are called *iterated function system* (IFS) codes. Given the IFS codes the mapping will iterate to the desired image after some time. If the image is two dimensional, the image generation is performed by letting a single 2D point trace out the image iteratively. The procedure can be very slow for large images. In general we have the following type of mappings for 2D images:

$$w_i(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} a_5 \\ a_6 \end{bmatrix} = \mathbf{Ax} + \mathbf{a}. \tag{1.1}$$

The affine map w_i takes a point $[x_1, x_2]$ and generates another point $[x_1', x_2']$. This new point is inserted back into the w_i and so on. An image is thus described by

[2]An affine transformation consists of a linear transformation followed by a translation.

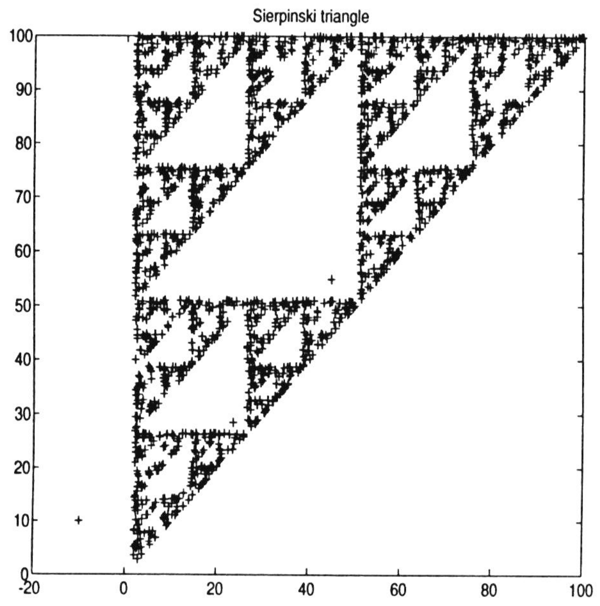


Figure 1.2: The Sierpinsky triangle.

- A set of affine mappings $w_i(\mathbf{x})$, $i \in [i, 2, \cdots, m]$.
- Parameters a_j for each $w_i(\mathbf{x})$.
- A set of probabilities p_i associated with each $w_i(\mathbf{x})$.

When more than one $w_i(\mathbf{x})$ is used for each iteration step it is decided which of the mappings is to be used on the current coordinate. Those $w_i(\mathbf{x})$ with the larger probabilities are selected more frequently in the process of image generation. In figure 1.2 the Sierpinsky triangle is generated by selecting the values of a_j for the three different mappings w_1, w_2, w_3 as shown in Table 1.1.

Chapter 2

The compression approach

The compression approach chosen for this thesis is based on projecting the different spectra [1] onto a selected basis set. PCA is such a method where the basis set is generated from vectors corresponding to directions along the maximum variance in the data structure. Fourier transform is another example of a projection technique. In this case the basis set is sine and cosine functions which are invariant with respect to the selected data set. Fourier transform can be used to compress data when coefficients corresponding to high frequencies are discarded. It is often assumed that the noise contain mostly high frequency components. The same principle is used in PCA as a compression technique where the eigenvectors corresponding to small eigenvalues are not used in the reconstruction. A reconstructed spectrum will in general be smoother and have a smaller information content. One particular type of spectra which have been studied in this thesis is *infrared spectra* which to some degree are smooth.

The type of basis set selected for solving the compression problem is the *B-spline basis* [11].

[1] A spectrum can be itself be an N-mode array of data.

2.1 Requirements and definitions

The following criteria has been used as *guidelines* in the present work:

1. For each spectrum the variables in the compressed representation *must* be comparable.

   This means that if a set of spectra is to be compressed, the same compression strategy must be applied to every spectrum. The compressed variable $c_{ij_1j_2\cdots j_N}$ for spectrum i, must contain the similar/comparable information to the compressed variable $c_{kj_1j_2\cdots j_N}$ for spectrum k. In some cases this is a suboptimal strategy from a storage viewpoint.

2. The compressed representations must retain all important features of the spectrum.

   In this thesis a lossy compression approach is used. This means that the reconstructed spectra from the compressed representation are not identical to the original data. For most practical purposes this is an acceptable strategy. Of course, to decide upon what is an acceptable reconstruction must be the responsibility of the investigator.

3. The compression must be reversible in the sense that it is possible to obtain an approximation of the original spectrum.

   There are some compression strategies that are in theory and/or practice irreversible.

4. The results from the analysis of the compressed data should be possible to interpret in the same manner as the original spectrum.

   It is possible to obtain compressed representations using nonlinear mappings from original to compressed space. Such a data set will often be qualitatively different from the original representation and thus the investigator has less chance of using external knowledge in interpreting the data.

5. It should be be possible to obtain "back-estimates" of the data analytical results of the *original data set* based on the results from the compressed representation only.

   One example used in this thesis is how to obtain the scores and loadings from PCA of the original data set using the scores and loadings of the compressed representation and the compression bases only.

6. It should not be necessary to rewrite the standard data analytical algorithms to make use of the compressed representation. This is not an absolute requirement. It is very handy when the same type of methods and programs can be used without needing to buy a new program.

Unfortunately, it is not possible to satisfy all these criteria simultaneously. The author found especially item 5 difficult and this has been partly solved by a method where it was necessary to rewrite the algorithms. This is of course in conflict with requirement 6.

2.2 The chosen compression method

The method of fitting B-splines to spectra has been chosen for the following reasons:

- B-spline fitting is a relatively simple method to understand.
- n'th degree continuity across knot points [2] is ensured.
- There is a *linear* relation between coefficients and the original data. This fact makes the theoretical treatment of the methods described in the thesis possible.
- A recursive formula generates the B-spline basis set from the vector of knot points (a knot vector).

[2] A curve is divided into segments defined by the knot points and a polynomial is defined within each segment.

- The basis matrices contain several zero elements which is useful for *sparse* representations [15].

A non-linear compression method could have been chosen which in general obtains much better compression rates than linear ones. When non-linear methods work, they are often much better than linear ones in capturing the essential features of the data. One such method could have been e.g. fitting gaussian/lorenzian peaks to the observed spectra. This is illustrated in Figure 2.1 where the one dimensional scores space of an **X** matrix of gaussian non-linear coefficients is compared to the corresponding 2D scores space of the actual gaussian curves. The one dimensional non-linear gaussian coefficient representation explains 100% of the variance, but the model on the gaussian curves themselves only explains 57.6% of the variance by two factors. The construction of this data set was to let one gaussian peak shift from left to right and increase the peak height.

A non-linear approach faces the following problems:

- The fitting procedure itself is slow (computer demanding).
- The iteration may not converge to an acceptable solution.
- The fitting procedure is sensitive to starting conditions.

2.2.1 Some details of B-splines

B-splines [11–13] or *basis*-splines represent curves as linear combinations of basis functions [14](pp.163-176):

$$f(x) = \sum_{j=1}^{n} c_j B_{j,k}(x) \ . \tag{2.1}$$

The basis functions $B_{j,k}(x)$ are generated algorithmically from a much smaller vector of *knots*. The knot vector is a nondecreasing sequence of

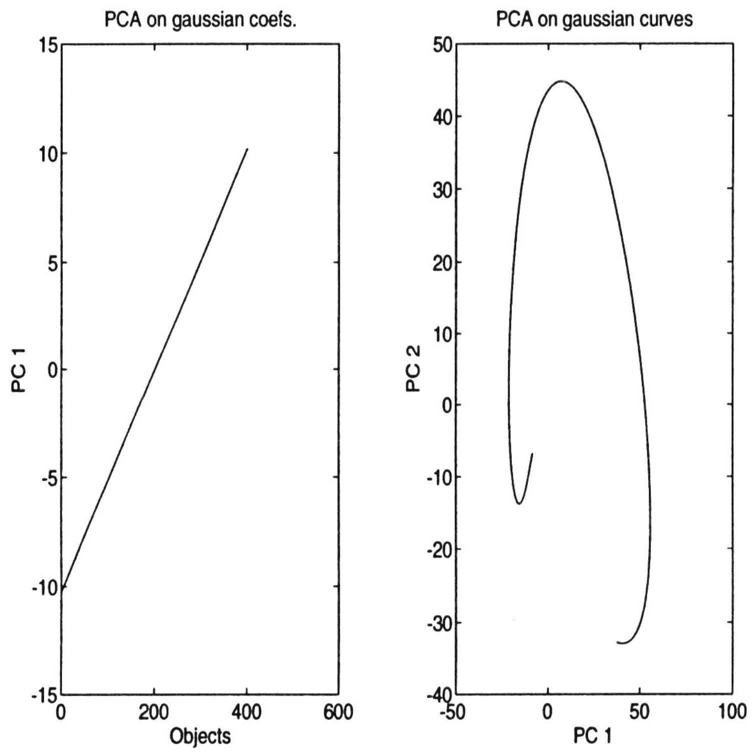


Figure 2.1: The left plot is 2D score plot from PCA on the gaussian coefficients alone. The right plot is the corresponding score plot from analysis of the curves generated from the gaussian coefficients.

numbers [3]:

$$h_1 \leq h_2 \leq \cdots \leq h_{n+k+1}$$

where k is the maximum degree of any polynomial.

Thus the function f can be estimated from the coefficient and knot vector *alone*. The actual set of basis functions is not stored, only the knot vector.

An example of a basis set based on a non-uniform knot distribution is shown in Figure 2.2.

[3] t_i is usually used in spline theory to indicate a knot point. This letter is not chosen since in chemometrics it is usually associated with score values .

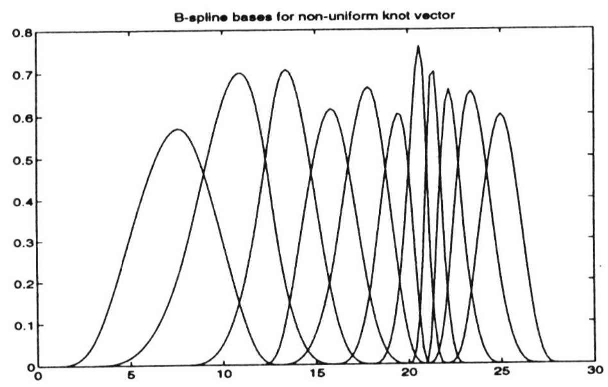


Figure 2.2: The B-splines bases for a non-uniform knot distribution. The knot sequence is: $[1, 3, 8, 12, 13, 16, 18, 20, 21, 21, 22, 23, 25, 27, 28]$. The interval is from 1 to 30.

The coefficient c_j associated with each basis function j may be interpreted as the height of the basis function. The functions $B_{j,k}(x)$ are constructed by a recursive algorithm [14]. The zero'th degree B-spline is defined as :

$$B_{j,0} = \begin{cases} 1 & \text{if } h_j \leq x < h_{j+1} \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

B-splines belong to a broad class of functions called *wavelets*. Some of the motivation for constructing wavelets may best be understood by comparing with Fourier transform. The bases used in Fourier transform are sine and cosine functions. These are localized in the frequency domain but not in the time (or space) domain. Wavelet bases are chosen such that they are localized in *both* the frequency and the time domain. Wavelets are constructed by translation and dilation of a "mother-wavelet" function $\phi(x)$. The basic dilation equation is :

$$\phi(x)_j = \sum_{k=1}^{K} c_k \phi_{j-1}(2x - k) \tag{2.3}$$

where one example of $\phi(x)_0$ is equal to $B_{j,0}$ (the "box function"), see equation 2.2 above. Different types of basis sets can be constructed by

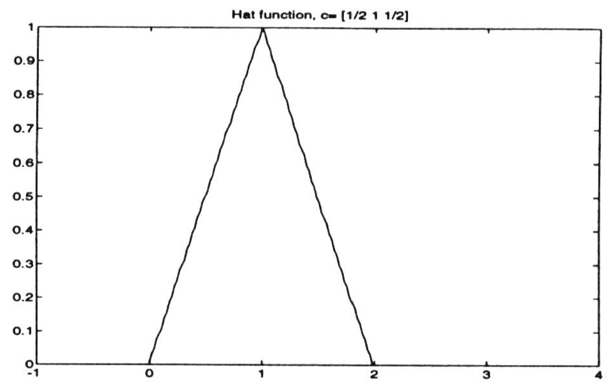


Figure 2.3: The hat function generated by recursive dilation and translation. $\mathbf{c} = \frac{1}{2}[1 \ \ 2 \ \ 1]$.

selecting the appropriate values for c_k in equation 2.3. For $\mathbf{c} = [\frac{1}{2} \ \ 1 \ \ \frac{1}{2}]$ the hat function appears as $j \rightarrow \infty$. The approximative hat function for $j = 4$ is shown in Figure 2.3.

The B-spline basis functions on a homogeneous knot distribution is generated by selecting $\mathbf{c} = \frac{1}{8}[1 \ \ 4 \ \ 6 \ \ 4 \ \ 1]$. This is illustrated in Figure 2.4.

The B-spline basis is diagonally dominant, i.e., the row/column sum of any row/column of the off-diagonal elements is smaller than, or equal to, the absolute value of the corresponding diagonal element [14]:

$$| \mathbf{B}_k^T \mathbf{B}_k | \geq \sum_{j \neq k} | \mathbf{B}_k^T \mathbf{B}_j | \tag{2.4}$$

This structure is especially convenient for *sparse* representation [15] of the basis matrices in computations. Sparseness means that the arrays contain a large number of zero elements. There are now available packages which takes this kind of structure into consideration when performing standard matrix and vector operations. Multiplication of elements which are known in beforehand to produce a zero element is a waste of computer resources and therefore avoided. We have found that sparse representations speed up several algorithms presented in this thesis considerably.

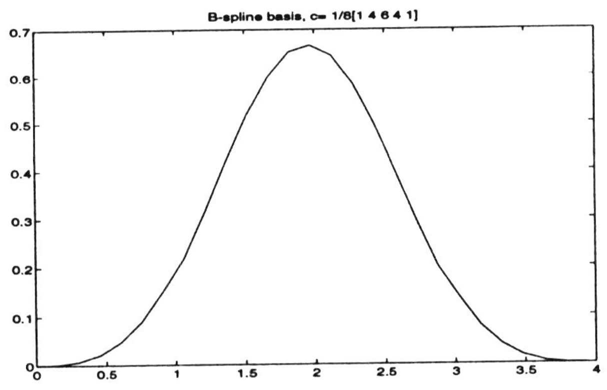


Figure 2.4: The B-spline function generated by recursive dilation and $\mathbf{c} = \frac{1}{8}[1 \quad 4 \quad 6 \quad 4 \quad 1]$.

Chapter 3

Two-mode arrays

3.1 Compression of 2D IR spectra

In paper II the application of B-splines in compressing a 2D IR spectrum is presented. The steps in compressing a two-mode data set using B-splines is as follows:

- Compute the mean or standard deviation vectors along the columns and/or rows for the data matrix. Such vectors will hereafter be referred to as *representative vectors* (RV) since they act as representatives for all the other vectors along a particular mode. Of course, other types of RV's different from the mean and standard deviation are possible.

- Each RV is used to construct a knot vector. For this we have chosen to use the maximum entropy method [16] (MEM). It was shown in paper I that MEM is similar to a zero'th degree B-spline technique.

- One B-spline basis matrix $\mathbf{B}_j$ for mode j is constructed from a single knot vector $\mathbf{h}_j$.

- The compressed representation $\mathbf{C}$ is constructed as:

$$\mathbf{C} = ((\mathbf{B}_1^T\mathbf{B}_1)^{-1})^T\mathbf{B}_1^T\mathbf{X}^T\mathbf{B}_2(\mathbf{B}_2^T\mathbf{B}_2)^{-1} \tag{3.1}$$

In order to measure the real compression it was necessary to perform *scalar quantization*. Intuitively one is lead to believe that the dimensions of the **C** matrix should be enough for measuring the compression. This is, however, not a safe method since the number of bits used in the representation of real values is important. MATLAB, which was used in all calculations, uses 64 bits in representing a real number. The input data set from the Nicolet 800 spectrometer uses 16 bit to represent each number. The multiplication of the original data set with the B-spline bases in MATLAB produced a **C** matrix using all the 64 bits in representing the coefficients. In this case a data "concentration" instead of compression could be in effect. A simple example illustrate the concept of data concentration. Assume a matrix **X** with dimensions $Dim(\mathbf{X}) = [2^8 \times 2^8]$ with 2^4 bits per data element. If the data matrix is concentrated to a matrix **C** with dimensions $Dim(\mathbf{C}) = [2^7 \times 2^7]$ with 2^6 bits per data element, it is realized that the total number of bits for **X** and **C** is equal ($2^8 2^8 2^4 = 2^7 2^7 2^6$). From a storage point of view this is not an acceptable situation, but can influence the floating point operation (FLOPS) consumption in algorithms. If the processor can manipulate on 64 word length vectors, it will not process any faster for a word length of 16. The number of multiplications will depend on the dimensions of the matrix which directly influences the number of FLOPS consumed by an algorithm.

In order to measure the compression rate, restrictions on the number of bits per symbol is required. This enables us to determine the amount of bits needed to obtain a satisfactory reconstruction.

A two-dimensional temperature-infrared spectrum was recorded of pyrolysis of kerogen in KBr. The data set is shown in Figure 3.1.

The standard deviation vectors along the two modes were used as RVs. Two B-spline basis matrices were constructed, and the coefficient matrix **C** was computed. A frequency distribution of all the values in **C** was performed. This distribution was used to partition the real line into intervals where more intervals were added to regions with high frequency. This analysis resulted in a set of J *source symbols*. Each value in **C** was then replaced with one of the

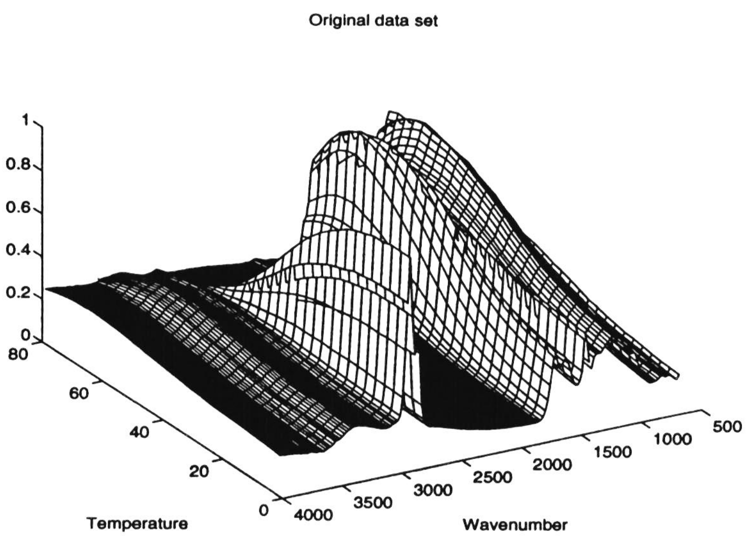


Figure 3.1: Original data set used in compression with quantization experiment.

closest lying J source symbols, producing a new matrix $\mathbf{C}_s$. This constituted the scalar quantization step. The new coefficient matrix $\mathbf{C}_s$ was used in the reconstruction (and storage) of the original array $\mathbf{X}$. The $[80 \times 869]$ data matrix $\mathbf{X}$ was compressed to a $[14 \times 115]$ data matrix $\mathbf{C}_s$, where the number of bits per data element (with respect to $\mathbf{X}$) was reduced from 16 to 0.16. Very little qualitative information was lost in this compression (see paper II for details). The scores and loadings vectors of the compressed representation $\mathbf{C}_s$ were qualitatively similar to the the scores and loadings vectors of the original data array $\mathbf{X}$.

This experiment demonstrated that B-splines captures the main features of smooth curves (here infrared spectra) and that the coefficients can be stored in a bit-efficient manner. The scalar quantization approach can also be used for data arrays having any number of modes.

3.2 A solution to "back-estimation" in PCA

Let the compressed matrix of $\mathbf{X}$ be $\mathbf{C}$. A PCA analysis of $\mathbf{X}$ produces a score and a loading matrix $\mathbf{T}$ and $\mathbf{P}^T$. A PCA analysis of $\mathbf{C}$ produces a score and a loading matrix $\mathbf{T}_c$ and $\mathbf{P}_c^T$. Since a compression has been employed, we wish to avoid direct computation of $\mathbf{T}$ and $\mathbf{P}^T$. Given $\mathbf{T}_c$, $\mathbf{P}_c^T$ and the basis matrices $\mathbf{B}_j, i \in \{1,2\}$, is it possible to generate perfect estimates of $\mathbf{T}$ and $\mathbf{P}^T$? It is here assumed that the compression is perfect and thus $\mathbf{C}$ reproduces $\mathbf{X}$ perfectly. Of course, this is not the case in most practical situations, but simplifies the theoretical discussion. A suggested method for calculating the back-estimates is:

$$\begin{aligned} \mathbf{P}_b^T &= \mathbf{P}_c^T \mathbf{B}_2^T \\ \mathbf{T}_b &= \mathbf{B}_1 \mathbf{T}_c \end{aligned} \tag{3.2}$$

where $\mathbf{T}_c$ and $\mathbf{P}_c^T$ are the scores and loadings matrices of the compressed representation $\mathbf{C}$. The $\mathbf{T}_b$ and $\mathbf{P}_b^T$ are the estimated scores and loadings matrices of the original matrix $\mathbf{X}$. Unfortunately the equations in 3.2 *do not* in general produce the true scores and loadings of $\mathbf{X}$. Neither $\mathbf{T}_b$ nor $\mathbf{P}_b^T$ are orthogonal as required for the true scores and loadings.

Even if the equations in 3.2 do not produce the perfect results, they will often give qualitatively the same results for certain compression matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ which satisfy the diagonally dominant criterion. For other compression matrices the $\mathbf{T}_b$ and $\mathbf{P}_b^T$ may not even be qualitatively similar to $\mathbf{T}$ and $\mathbf{P}^T$. The method constructed to achieve perfect back-estimates is called *postponed basis matrix multiplication* (PBM). This is a method for rewriting algorithms. The basic idea is to postpone the large multiplications between compression and kernel (compressed) matrices in the algorithms. The method will not work for all kinds of algorithms. 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- and/or 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.

When applying the PBM method to the NIPALS algorithm, the first step is to assume analysis on the whole data matrix $\mathbf{X}$ and substituting later for the compression model $\mathbf{X} = \mathbf{B}_1\mathbf{C}\mathbf{B}_2^T$ whenever $\mathbf{X}$ occurs in the equations. In addition, it is assumed that all the loadings and scores vectors of $\mathbf{X}$ can be written as linear combinations of the basis matrices in the compression. Thus it is assumed that

$$\mathbf{p}^T = \mathbf{v}^T\mathbf{B}_2^T \tag{3.3}$$

$$\mathbf{t} = \mathbf{B}_1\mathbf{u}\ . \tag{3.4}$$

By substituting the new expressions for the scores and loadings in the NIPALS algorithm, it is observed that the basis matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ are redundant in several of the steps in the iteration for each factor. Since this can be computationally demanding multiplications, they are postponed until the iteration has finished for all the factors. The new algorithm together with the standard NIPALS is shown in paper IV.

Unfortunately, the algorithm is more FLOPS demanding than just applying NIPALS directly on the compressed representation $\mathbf{C}$. This is due to the fact that we have extra kernel matrices in the iteration originating from the compression model: $\mathbf{G}_1 = \mathbf{B}_1^T\mathbf{B}_1$ and $\mathbf{G}_2 = \mathbf{B}_2^T\mathbf{B}_2$. It is dependent on the data structure and compression model whether the PBM-PCA algorithm will be any faster than direct analysis of the large $\mathbf{X}$. If B-spline basis is used, it is often observed that the $\mathbf{G}_1, \mathbf{G}_2$ matrices are sparse which increase the speed of the PBM algorithms considerably.

The output from the PBM algorithm is two matrices $\mathbf{U}$ (scores-like matrix) and $\mathbf{V}$ (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 $\mathbf{D}$ 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: $\mathbf{T}_h^T\mathbf{T}_h = \mathbf{U}^T\mathbf{G}_1\mathbf{U} = \mathbf{D}$ and $\mathbf{P}_h^T\mathbf{P}_h = \mathbf{V}^T\mathbf{G}_2\mathbf{V} = \mathbf{I}$. If the compression is perfect we have that $\mathbf{T} = \mathbf{T}_h$ and $\mathbf{P} = \mathbf{P}_h$.

3.2.1 Deducting the PBM-PCA equations from generalized eigenvalue decomposition [1]

In the following subsection it is shown that the particular PBM-PCA algorithm described in paper IV can be deducted from another theoretical viewpoint. Instead of presenting PCA as an algorithm (NIPALS), it is possible to obtain the scores and loadings of the matrix $\mathbf{X}$ by solving the following eigenvalue equations:

$$\begin{aligned} \mathbf{X}\mathbf{X}^T\mathbf{U} &= \mathbf{U}\Gamma && (3.5) \\ \mathbf{X}^T\mathbf{X}\mathbf{V} &= \mathbf{V}\Gamma. && (3.6) \end{aligned}$$

As will be demonstrated, the same approach is also possible for the PBM-PCA algorithm.

Hereafter, the equation for the scores will be treated in detail. The same will apply for the loadings equation. Inserting the compression model :

[1]The material in the present subsection has not been published or submitted to an international scientific journal. A short communication is under preparation.

$$\mathbf{X} = \mathbf{B}_1\mathbf{C}\mathbf{B}_2^T \tag{3.7}$$

into equation 3.5, we get

$$\mathbf{B}_1\mathbf{C}\mathbf{B}_2^T\mathbf{B}_2\mathbf{C}^T\mathbf{B}_1^T\mathbf{U} = \mathbf{U}\Gamma\ . \tag{3.8}$$

In PBM a matrix $\mathbf{U}_0$ is found such that $\mathbf{U} = \mathbf{B}_1\mathbf{U}_0$. By inserting this into formula 3.8 and setting $\mathbf{G}_i = \mathbf{B}_i^T\mathbf{B}_i,\ i \in \{1,2\}$ we get:

$$\mathbf{B}_1\mathbf{C}\mathbf{G}_2\mathbf{C}^T\mathbf{G}_1\mathbf{U}_0 = \mathbf{B}_1\mathbf{U}_0\Gamma. \tag{3.9}$$

Equation 3.9 is now premultiplied by $\mathbf{B}_1^T$:

$$\mathbf{G}_1\mathbf{C}\mathbf{G}_2\mathbf{C}^T\mathbf{G}_1\mathbf{U}_0 = \mathbf{G}_1\mathbf{U}_0\Gamma. \tag{3.10}$$

Note that the matrix dimensions in the equation have been reduced. Solving this equation gives the same $\mathbf{U}_0$ as found by the PBM method applied to the NIPALS algorithm. This is a generalized eigenvalue equation. It should be noted that $\mathbf{U}_0$ in general is not columnwise orthonormal. This equation must be used if $\mathbf{G}_1$ is not invertible. If it is invertible, pre-multiplication by $\mathbf{G}_1^{-1}$ is possible:

$$\mathbf{C}\mathbf{G}_2\mathbf{C}^T\mathbf{G}_1\mathbf{U}_0 = \mathbf{U}_0\Gamma. \tag{3.11}$$

This is the asymmetric eigenvalue equation ($\mathbf{C}\mathbf{G}_2\mathbf{C}^T\mathbf{G}_1$ is a quadratic, but not symmetric matrix). Again, the $\mathbf{U}_0$ in general is not columnwise orthonormal. Due to normalization differences in the generalized and normal eigenvalue algorithms, $\mathbf{U}_0$ must be multiplied by a diagonal matrix $\mathbf{D}$ such that

$$\mathbf{U} = \mathbf{B}_1\mathbf{U}_0\mathbf{D}\ . \tag{3.12}$$

The following matrices are defined:

$$\mathbf{U} = \mathbf{B}_1\mathbf{U}_0^{(1)} \tag{3.13}$$

and

$$\mathbf{U}_0^{(1)} = \mathbf{U}_0^{(2)}\mathbf{D} \tag{3.14}$$

where $\mathbf{U}_0^{(2)}$ is the eigenvectors from solving equation 3.11.

$$\mathbf{U}^T\mathbf{U} = \mathbf{I} = \mathbf{U}_0^{(1)T}\mathbf{G}_1\mathbf{U}_0^{(1)} = \mathbf{D}\mathbf{U}_0^{(2)T}\mathbf{G}_1\mathbf{U}_0^{(2)}\mathbf{D} \tag{3.15}$$

$$\mathbf{D} = (\mathbf{U}_0^{(2)T}\mathbf{G}_1\mathbf{U}_0^{(2)})^{-1/2} . \tag{3.16}$$

Dropping the superscript, the total backtransformation for the orthonormal score matrix $\mathbf{U}$ is:

$$\mathbf{U} = \mathbf{B}_1\mathbf{U}_0(\mathbf{U}_0^T\mathbf{G}_1\mathbf{U}_0)^{-1/2} . \tag{3.17}$$

The corresponding eigen equation for the loading matrix is:

$$\mathbf{G}_2\mathbf{C}^T\mathbf{G}_1\mathbf{C}\mathbf{G}_2\mathbf{V}_0 = \mathbf{G}_2\mathbf{V}_0\Gamma \tag{3.18}$$

or if the $\mathbf{G}_2$ matrix is invertible:

$$\mathbf{C}^T\mathbf{G}_1\mathbf{C}\mathbf{G}_2\mathbf{V}_0 = \mathbf{V}_0\Gamma . \tag{3.19}$$

Taking into consideration the normalization differences the total backtransformation equation for the loading matrix becomes:

$$\mathbf{P} = \mathbf{B}_2\mathbf{V}_0(\mathbf{V}_0^T\mathbf{G}_2\mathbf{V}_0)^{-1/2} . \tag{3.20}$$

Chapter 4

N-mode arrays

4.1 The need for a new notation

The notation traditionally used for scalars, vectors and matrices is very powerful. The typographic formulation has no problem with capturing the different ways to manipulate the data objects. Figuratively speaking, the typographical nature of scalars is zero dimensional, one dimensional for vectors and two dimensional for matrices. We are suddenly faced with a typographical problem if the same logic is to be followed for N-mode arrays. Three mode arrays can always be described on paper by 2-D drawings of boxes. It is a cumbersome notation, but it has the advantage of rapidly conveying the basic idea of the data structure. For four- mode or even higher mode arrays the "drawing" approach quickly collapses. Useful notations therefore do not employ such approaches. The following criteria should apply to any notation:

- Unambiguous.
- Intuitive
- The notation should represent the underlying data structure in a logical way

- Unnecessary features introduced by notation should be kept at a minimum.
- Symmetries and patterns in the equations should be possible to discover.
- Easy to use even for larger problems.

The explicit summation notation is the only notation in addition to the diagram notation which will be mentioned here. The diagram notation is really a visualization of the explicit summation notation. The following equation is an example of the explicit summation notation:

$$\sum_i \sum_j x_{ik} y_{ij}. \tag{4.1}$$

The operations between array elements are indicated. The array elements are represented by an array name symbol where the indices for the different modes are written as subscripts. Summation signs are used to signify which indices are to be summed over. For large N-mode problems this notation gets complicated and it is not easy to discover symmetries and patterns in the index topology.

In order to make the explicit summation notation more convenient to use for N-mode array equations, a diagram notation has been constructed. Details of this notation are discussed below.

4.2 Properties and rules of the diagram notation

The need for an intuitive and powerful notation for N-mode array equations has also been present in physics (quantum mechanics) and group theory. The famous physicist Richard P. Feynman constructed a diagram notation (so-called Feynman diagrams) for manipulation of complex equations involved in

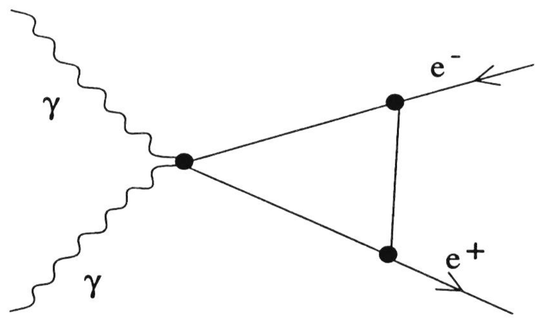


Figure 4.1: Example of a Feynman diagram. This demonstrates the scattering of two photons into an electron and a positron.

N-body interactions. The Feynman diagrams became successful and is today a powerful tool in high energy physics and group theory, see Figure 4.1 for an example of a Feynman diagram. Here the scattering of two photons into an electron and a positron is shown.

Since chemometrics and psychometrics have similar notational problems, it is to be expected that the solution strategy employed by the high energy physicists may also be of advantage for chemometricians and psychometricians.

In the past few years, N-mode array problems in chemistry have been more common and the need for understanding/manipulation of different N-mode algorithms/theories has increased. The manipulation of diagrams in general is not that strange to the chemist. This is the usual way of displaying molecular structures. It is therefore the author's personal opinion that chemists and chemometricians prefer a graphical and intuitive notation to e.g. the Kronecker product notation (see paper VI).

The diagram notation is a graphical visualization of the index topology in the explicit summation notation. The diagrams have the appearances of *graphs*, and use of graph terminology is therefore appropriate [17]. A diagram contains *nodes* and *edges*. Nodes signify array elements (e.g. x_{ijk}), and edges signify indices. An edge is either *connected* or *unconnected*. An unconnected edge is attached to one node only whereas a connected edge is attached to

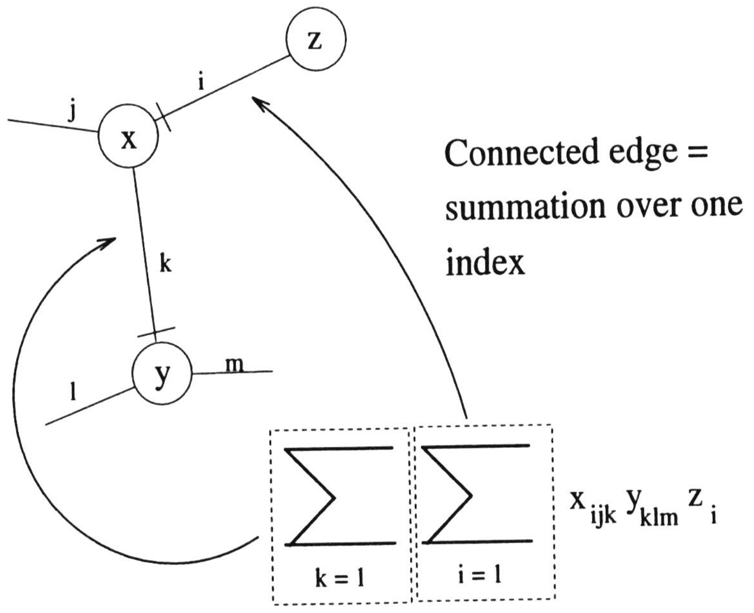


Figure 4.2: The essential idea of the diagram notation is illustrated. Lines signify summation signs. The small lines perpendicular to the edges are first index pointers (fips).

two nodes. A connected edge signifies *summation* over *one* index. When two arrays share a common index which is to be summed over, an edge is drawn between them. This is illustrated in Figure 4.2.

The total number of unconnected edges of an expression is the number of modes (or the *mode number*) of the result. If two 3-arrays combine by summing one common index, the mode number of the result is $3 + 3 - 2 = 4$.

In the center of the node the name of the array is placed. Lower case characters are used since a node represents a single array *element* (scalar). In the vicinity of the the edges the correct index names can be written in order to clarify. The index names are written *anti-clockwise* from the first index (this follows the convention in Feynman diagrams). Sometimes it is necessary to indicate which edge signifies the *first* index. For this, a small bar perpendicular to the edge is used; this mark is called the *first index pointer*

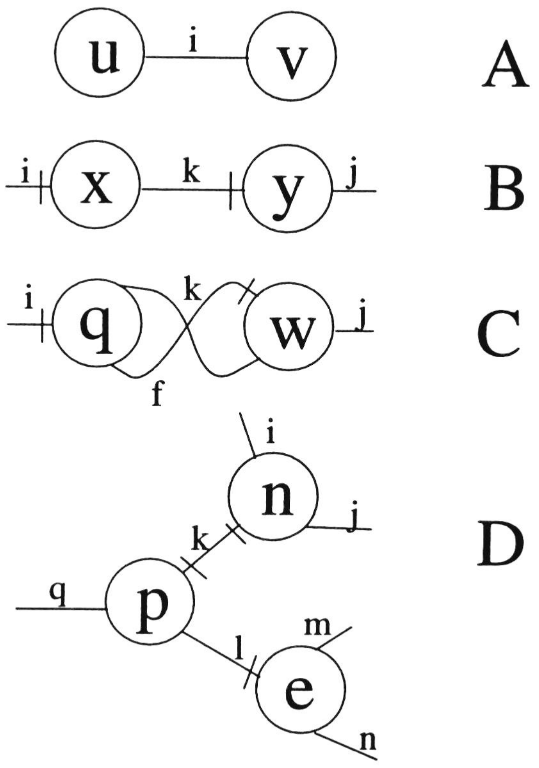


Figure 4.3: Example use of diagram notation on arrays. See text for explanation.

(the *fip*).

Figure 4.3 shows a few examples of array diagram equations, and the corresponding summation formulas for the diagram equations presented are:

(A)

$$\sum_i u_i v_i$$

This is the standard inner product.

(B)

$$\sum_k x_{ik} y_{kj}$$

This is a standard matrix product.

(C)

$$\sum_k \sum_f q_{ifk} w_{fkj}$$

An array product between two three-mode arrays. The result is a matrix because the number of free indices or unconnected edges is two.

(D)

$$\sum_k \sum_l p_{kql} n_{kji} e_{lnm}$$

Here the result is a fifth mode array since five free edges are seen.

Due to the imposed topological constraints by the notation, it is easy to see that summing over an index shared by a number of nodes different from two will cause problems. This is solved by introducing arrays containing just zeros and ones. By connecting to such arrays new summations are introduced such that summation over an index shared by more than two arrays can be represented within the framework of the topological constraints. These special arrays are N-mode Kronecker deltas $\delta_{i_1 i_2 \cdots i_N}$. Two types are used in the diagram notation:

- Type I:

  This Kronecker delta is used in defining trace of an N-array and summing of an index involving $N \neq 2$ nodes; here symbolized as the array $\underline{\Delta}_N^{(I)}$ and is defined as

$$\delta^{(I)}_{i_1 i_2 \cdots i_N} = \begin{cases} 1 & \text{if } i_1 = i_2 = \cdots = i_N \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

  This Kronecker delta is referred to as the *sum nexus symbol* and is displayed in the diagram notation as a filled rectangle, see Figure 4.4.

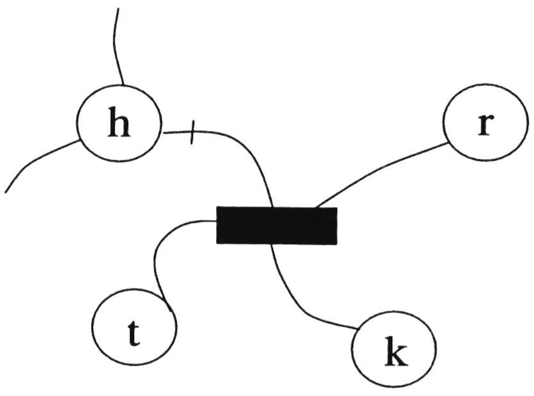


Figure 4.4: The filled rectangle is the *sum nexus* symbol. This is used when more than two nodes share an index which is to be summed over. The corresponding explicit summation notation equation is $\sum_{i=1}^{N} h_{ijk}t_i k_i r_i$.

- Type II:

  This Kronecker delta is used in N-mode array differentiation; here symbolized as the array $\underline{\Delta}_N^{(II)}$. In the diagram notation, $\underline{\Delta}_N^{(II)}$ is depicted as a filled circle of size smaller than the other array nodes. The general definition of this Type II arrays is:

$$\delta^{(II)}_{i_1 i_2 \cdots i_n o_1 o_2 \cdots o_n} = \begin{cases} 1 & \text{if } i_j = o_j \ \forall j \in \{1, 2, \cdots, n\} \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

  The properties of these Kronecker deltas are discussed in more detail in paper VI.

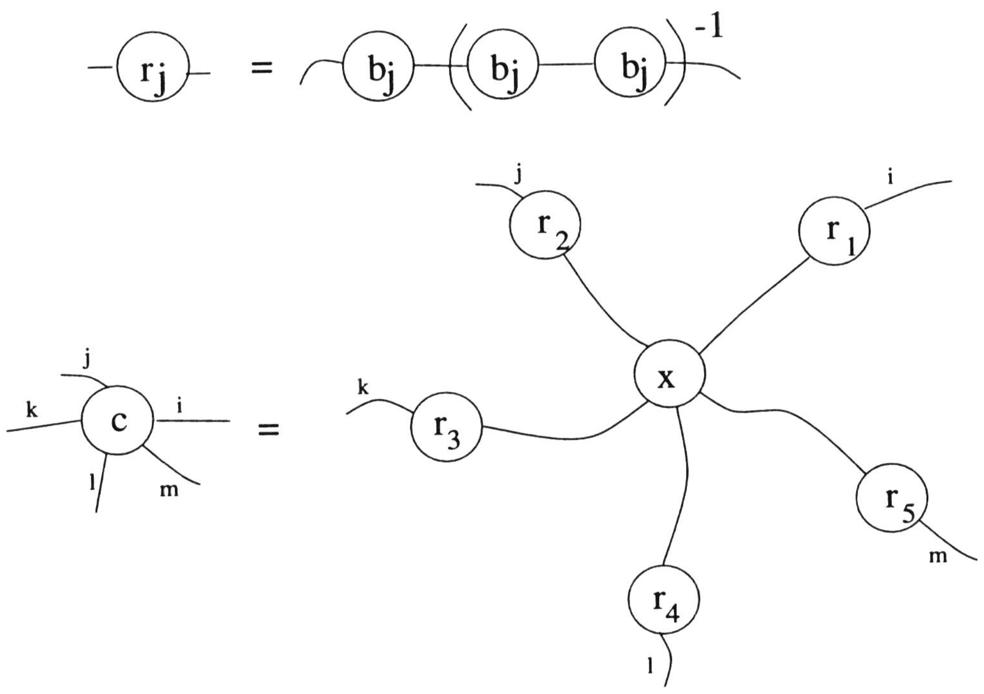


Figure 4.5: A diagram of a B-spline compression of a five mode array. Here it is assumed that each mode is compressible by the B-spline basis. This may not always be so. See text for explanation of the $\mathbf{R}_j$ matrix.

4.3 Compression of N-mode arrays

The idea of N-mode compression by B-splines is very simple. For each mode an associated B-spline basis matrix $\mathbf{B}_j$ is used to compress mode j; in practice it is the matrix $\mathbf{R}_j = \mathbf{B}_j(\mathbf{B}_j^T\mathbf{B}_j)^{-1}$ which is multiplied with each mode. Some modes may not be compressible by the B-spline basis and another basis may be efficient. Alternatively, some modes can be left uncompressed. In Figure 4.5 the diagram notation is used to illustrate the compression of a five mode data array. In this example all modes are assumed to be compressible.

The result, a smaller array $\underline{\mathbf{C}}$, is used instead of the original and much larger array $\underline{\mathbf{X}}$. When a B-spline basis is used, the coefficients and the results from the analysis are similar to the results obtained from analysis of the original data array (see paper V). One reason for this is again the diagonally dominant structure of the basis matrices in B-splines. If back estimation becomes a problem, the PBM approach described below can be used to solve the problem.

4.4 Three-mode principal component analysis

Three mode PCA is an extension of standard PCA or singular value decomposition (SVD). The SVD model is stated as

$$x_{ij} = \sum_k \sum_p u_{ik} s_{kp}^{1/2} v_{pj} \tag{4.4}$$

where u_{ik} and v_{pj} are obtained by solving eigenvalue equations involving the following grammian matrices **C1** and **C2**:

$$\begin{aligned} c1_{ii'} &= \sum_j x_{ij} x_{i'j} && (4.5) \\ c2_{jj'} &= \sum_i x_{ij} x_{ij''} \;. && (4.6) \end{aligned}$$

The three-mode Tucker 3 model is stated as follows:

$$x_{ijk} = \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{ip} e_{jq} h_{kr} c_{pqr} \tag{4.7}$$

where c_{pqr} is a *core array*. The Tucker 3 model formulated in diagram notation is shown in Figure 4.6.

It is possible to construct corresponding grammian-like matrices:

$$c1_{ii'} = \sum_{j=1}^{J} \sum_{k=1}^{K} x_{ijk} x_{i'jk} \tag{4.8}$$

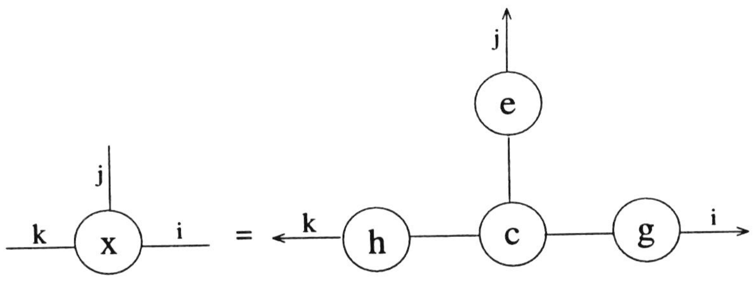


Figure 4.6: The Tucker 3 model. Note the arrows which signify that the different loadings matrices $\mathbf{G}, \mathbf{E}, \mathbf{H}$ are columnwise orthonormal.

$$c2_{jj'} = \sum_{i=1}^{I} \sum_{k=1}^{K} x_{ijk} x_{ij'k} \tag{4.9}$$

$$c3_{kk'} = \sum_{i=1}^{I} \sum_{j=1}^{J} x_{ijk} x_{ijk'}. \tag{4.10}$$

One should now expect that taking the eigenvalue decomposition of the $\mathbf{C1}, \mathbf{C2}, \mathbf{C2}$ matrices would produce the correct loading matrices for the three-mode array $\underline{\mathbf{X}}$. If we extract *all* the eigenvectors of $\mathbf{C1}, \mathbf{C2}, \mathbf{C2}$, this would be true, but if a smaller number of factors is extracted, it can be shown that the estimators of the core array are not longer least squares ones. This rather unsuspected result emphasizes that going from two modes to higher modes does not necessary involve a trivial extension of algorithms and theory. In order to solve the problem, the Alternating Least Squares (ALS) algorithm is employed. This algorithm iteratively estimates the different loading matrices using estimates from previous iteration steps. A central part of the ALS algorithm is shown in Figure 4.7 using the diagram notation.

4.5 A PBM solution to the three-mode back-estimate problem

When the PBM method was found to work with standard PCA it was believed that it should work for the three-mode PCA also. The diagram nota-

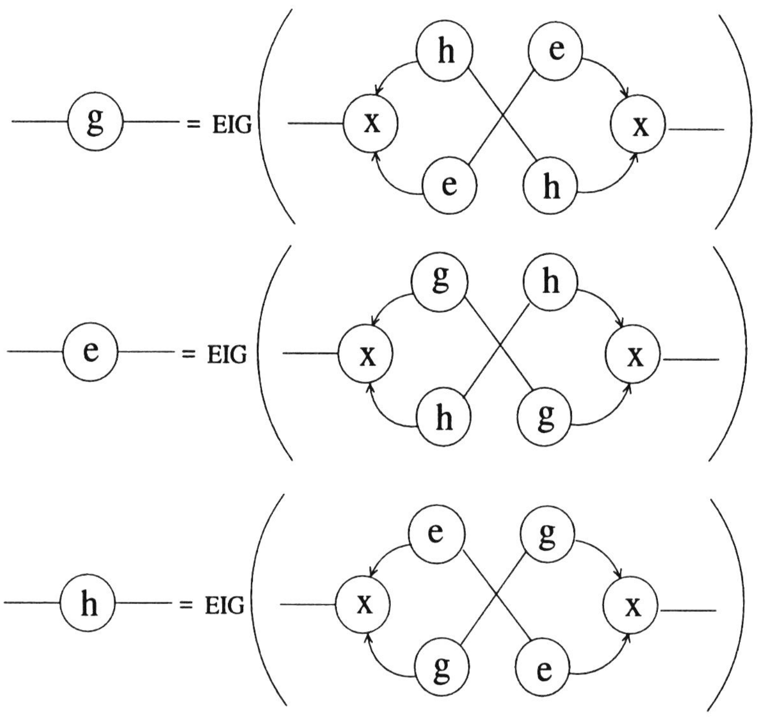


Figure 4.7: These steps in the ALS algorithm are iterated several times.

tion was used to solve the problem theoretically. After some manipulations of the diagram equations, the resulting equations for the PBM applied to three-mode PCA were deducted. Later implementations in MATLAB confirmed the theoretical results. The diagrams also provided important guidelines in the actual programming of the algorithm.

By rewriting the diagram equations in Figure 4.7 in terms of the three-mode B-splines model of the original array $\underline{\mathbf{X}}$, it was easy to see that the different basis matrices were redundant in parts of the computations. Since the estimate of each loading matrix involves an eigenvalue decomposition, it was necessary to rewrite the eigenvalue decomposition method as a PBM method also. The power method was used as the eigenvalue decomposition algorithm.

Figure 4.8 and 4.9 demonstrate how the diagram notation was used to

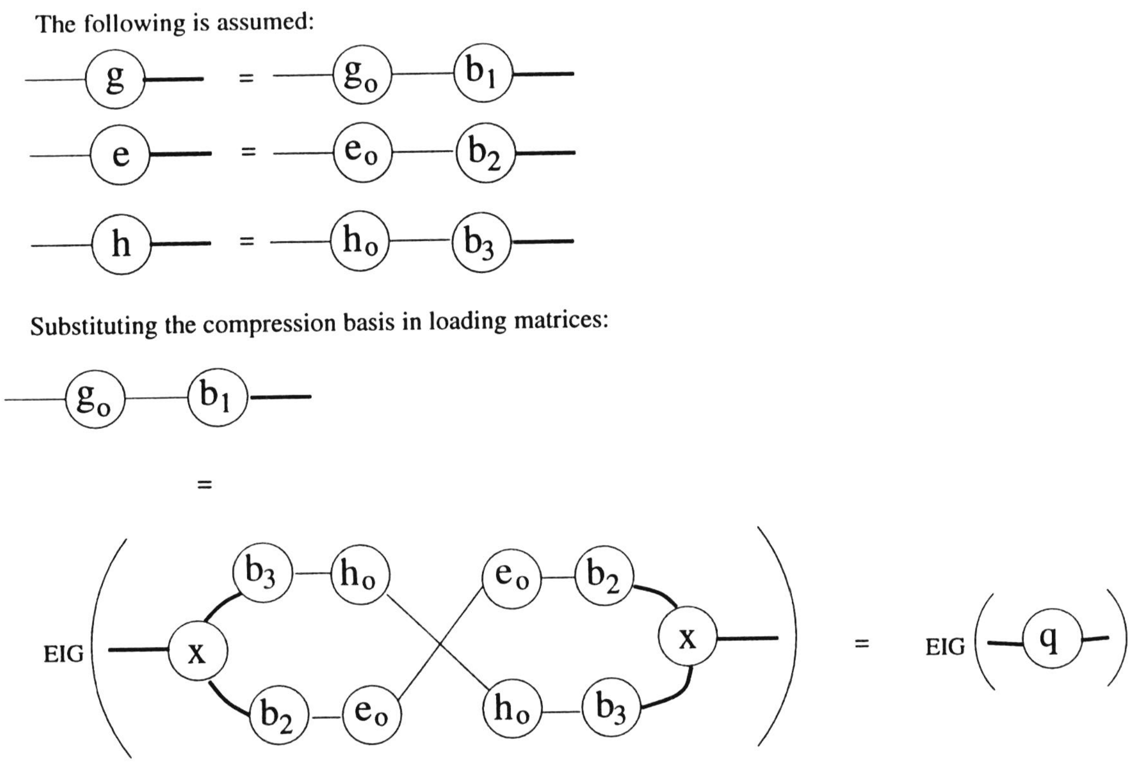

The following is assumed:
g = g_o b_1
e = e_o b_2
h = h_o b_3
Substituting the compression basis in loading matrices:
g_o b_1
=
EIG
x
b_3 h_o
b_2 e_o
e_o b_2
h_o b_3
x
=
EIG
q

Figure 4.8: This figure demonstrate how the diagram notation was used to deduct the PBM equations for the three-mode PCA. Thick lines identify large modes.

deduce the correct equations (see paper V and VI for details). Note in Figure 4.8 that it is assumed that every result in the algorithm can be written in terms of the original compression model. The deduction is really a trivial substitution of the compression model in each of the steps involving loading matrices and the original array $\underline{\mathbf{X}}$. $\underline{\mathbf{X}}$ is not yet replaced by the compression model in Figure 4.8. Only one of the expressions for the loading matrices are shown ($\mathbf{G}$). The equations are analogous for the other loading matrices.

In Figure 4.9 the substitution of the $\underline{\mathbf{X}}$ with the compression model has been performed. Figure 4.9 shows the pre- and postmultiplication of the basis matrix to the expression enclosed by dotted lines ("matrix used

Now we substitute X with the compression model:

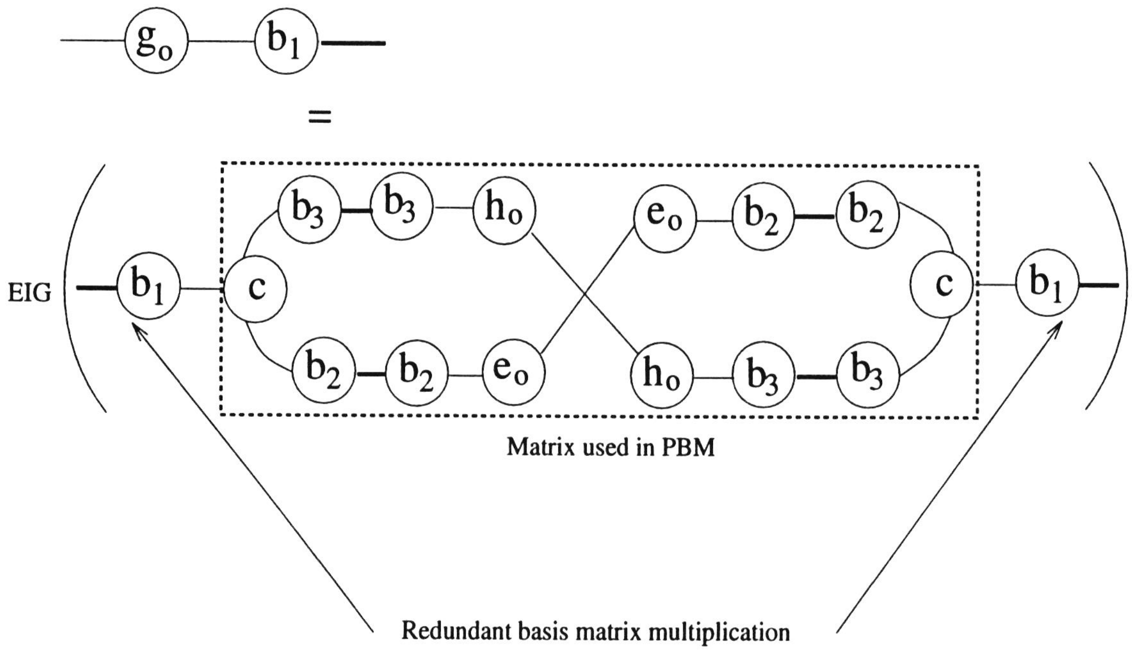


Figure 4.9: This figure clearly shows why the basis matrices are redundant in the ALS iteration steps. Here the compression model for $\underline{\mathbf{X}}$ has been written out explicitly.

in PBM"). It is evident from this figure that these matrix multiplications are redundant and can be postponed to after the ALS iteration steps have finished.

Note in the figure the grammians introduced by the basis matrices. These extra matrices makes the PBM methods slower than just analyzing the compressed representation $\mathbf{C}$ directly. In order to minimize the effects from these grammians, the use of *sparse matrix technology* has been found efficient [15]. When B-spline bases are used, the grammians $\mathbf{B}_i^T\mathbf{B}_i$ of these bases have a structure suitable for sparse representations, and savings in FLOPS consumption is achieved.

Chapter 5

Fast fuzzy c-means clustering

It is well known that PCA can be performed by extracting the largest eigenvectors of the kernel matrices $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$. For those problems when the data matrix $\mathbf{X}$ is rectangular, i.e. more objects than variables, or more variables than objects, the eigenvectors can easily be obtained by performing the analysis on the kernel matrix with the smallest dimensions. A similar approach is also possible for the PLS method. This has been demonstrated by Lindgren *et.al.* [18] and Manne [19]. Lindgren *et.al.* rewrote the NIPALS algorithm, but Manne used the more efficient Lanzcos formulation of PLS1.

The success of using kernel matrices for speeding up algorithms inspired the present author to test if this approach could be of use for the fuzzy c-means (FCM) algorithm also. The algorithm usually employed is unnecessary slow for data set with many variables compared to the number of objects. In chemistry it is often the case that the number of variables are much larger than the number of feature vectors and such data sets slow down the computation in FCM considerably.

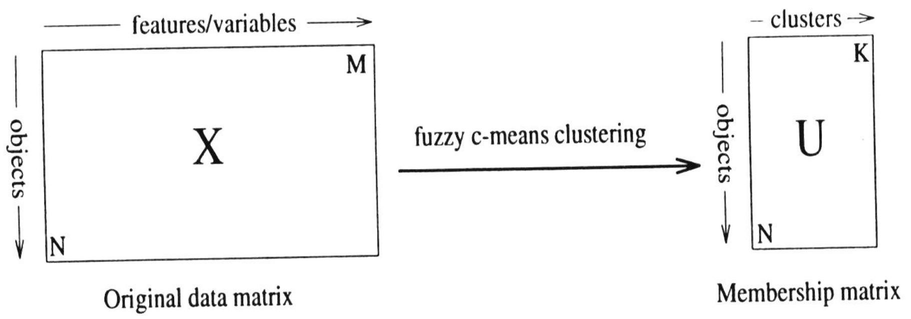


Figure 5.1: An explanation of the word "object vector"and some of the data matrices involved in FCM clustering.

5.1 General definitions

Let $X = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ be a finite set of *feature vectors* [1] $\mathbf{x}_i$ where $x_{ij} \in \mathcal{R}$ is the j'th feature of each $\mathbf{x}_i$. The aim of any clustering method is to decide if X can be divided naturally into a number of partitions. A *hard* K-partition of X is a K-tuple $P = \{S_1, \cdots, S_K\}$ of non-empty subsets of X such that

$$X = \bigcup_{k=1}^{K} S_k \tag{5.1}$$

and

$$S_k \bigcap S_l = \emptyset, \qquad \forall \ k \neq l \ . \tag{5.2}$$

K (the number of clusters) is an integer satisfying $2 \leq K < N$. Let $\mathbf{W}$ be the *membership matrix* for the objects. In a hard K-partition we have that:

$$w_{ik} = \begin{cases} 1 & \text{if } \mathbf{x}_i \in S_k \\ 0 & \text{otherwise.} \end{cases} \tag{5.3}$$

The sum of the membership values for one object must be unity:

[1] Also referred to as *objects*, see Figure 5.1 for an explanation.

$$\sum_{k=1}^{K} w_{ik} = 1 \ , \quad \forall\, i. \tag{5.4}$$

In addition we have that the sum over all objects for one cluster k must be smaller than the number of objects in the data set:

$$0 < \sum_{i=1}^{N} w_{ik} < N \ , \quad \forall\, k. \tag{5.5}$$

Each entry w_{ik} of a hard clustering membership matrix has thus the value 0 or 1, i.e. $w_{ik} \in \{0, 1\}$.

A much used objective function for defining clusters in X is the within sum of squares [20] or minimum variance payoff:

$$J(\mathbf{V}) = \sum_{i=1}^{N}\sum_{k=1}^{K} w_{ik}\|\mathbf{x}_i - \mathbf{v}_k\|^2 \tag{5.6}$$

where $\mathbf{v}_k$ are the cluster centers of S_k and $\mathbf{V}$ is the matrix containing all the cluster center vectors. A clustering program will try to minimize $J(\mathbf{V})$.

For a *fuzzy* K-partition of X we have a membership matrix $\mathbf{U}$ which satisfy the criteria described in equations 5.4 and 5.5, but the individual elements u_{ik} can be any number between (and including) zero and unity, i.e. $u_{ik} \in [0, 1]$.

5.2 The standard FCM algorithm

The FCM algorithm is started by obtaining an initial estimate of the matrix $\mathbf{V}$ containing the cluster centers as row vectors. The dimensions of the $\mathbf{V}$ matrix are $[K \times M]$ where K is the number of clusters and M is the number of features. The initial estimate of $\mathbf{V}$ can be produced by e.g. random perturbations of the mean vector of all the feature vectors. The feature vectors are stored as rows in the matrix $\mathbf{X}$. The dimensions of $\mathbf{X}$ are $[N \times M]$, where N is the number of objects. The following steps are iterated until convergence:

$$u_{ik} = \left(\sum_{j=1}^{K} \left(\frac{d_{ik}}{d_{ij}} \right)^{2/(m-1)} \right)^{-1} \tag{5.7}$$

$$\mathbf{v}_k = \sum_{i=1}^{N} (u_{ik})^m \mathbf{x}_i \left(\sum_{i=1}^{N} (u_{ik})^m \right)^{-1} \tag{5.8}$$

where d_{ik} and d_{ij} are the A-distances

$$d_{ik} = (\|\mathbf{x}_i - \mathbf{v}_k\|_A)^{1/2} = ((\mathbf{x}_i - \mathbf{v}_k)^T \mathbf{A} (\mathbf{x}_i - \mathbf{v}_k))^{1/2}. \tag{5.9}$$

$\mathbf{v}_k$ is the k'th cluster center and $\mathbf{x}_i$ is the i'th feature vector. $\mathbf{A}$ is a positive definite matrix chosen to control the shape of the optimal clusters. Here it is assumed that $\mathbf{A}$ is the identity matrix, i.e. for hyperspherical clusters. The iteration is stopped when the change in

$$J(\mathbf{V}) = \sum_{i=1}^{n} \sum_{k=1}^{c} u_{ik}^m \|\mathbf{x}_i - \mathbf{v}_k\|^2 \tag{5.10}$$

is below a certain value ϵ. Typical values for m are in the range $1 < m < 3$

.

5.3 Matrix representation of distance calculation

The main idea of the new algorithm is based on a different way of formulating distance matrices. The Euclidean distance can be computed using matrix algebra as:

$$\mathbf{D}^2 = \mathbf{M} + \mathbf{M}^T - 2\mathbf{R} \tag{5.11}$$

where $\mathbf{R} = \mathbf{X}\mathbf{X}^T$ and $\mathbf{M} = \mathbf{1}_{[N \times 1]} \text{diag}(\mathbf{R})^T$. The *diag* operator generates a column vector of the diagonal of a square matrix. The $\mathbf{M}$ matrix is not symmetric.

In the FCM algorithm the distances between objects and cluster centers are calculated, not between the objects themselves. This gives rise to an *asymmetric* distance matrix:

$$\mathbf{Q}^2 = \mathbf{A} + \mathbf{B}^T - 2\mathbf{L} \tag{5.12}$$

where $\mathbf{A} = \mathbf{1}_{[N\times 1]}\text{diag}(\mathbf{F})^T$,$\mathbf{F} = \mathbf{V}\mathbf{V}^T$, $\mathbf{B} = \mathbf{1}_{[K\times 1]}\text{diag}(\mathbf{R})^T$, and $\mathbf{L} = \mathbf{X}\mathbf{V}^T$. $\mathbf{R}, \mathbf{L}$ and $\mathbf{F}$ are referred to as "kernel matrices".

By updating the kernel matrices, the repeated distance calculations for every iteration is avoided. Based on this, the new FCM-M algorithm is as follows for every iteration step:

$$\begin{aligned}
\mathbf{Q} &= \left(\mathbf{A} + \mathbf{B}^T - 2\mathbf{L}\right)^{1/2} && (5.13)\\
\mathbf{U} &= \varphi(\mathbf{Q})^T && (5.14)\\
\mathbf{P} &= \mathbf{U}^m && (5.15)\\
\mathbf{H} &= \mathbf{P}^T\text{diag}(\mathbf{1}./(\mathbf{1}^T\mathbf{P}^T)) && (5.16)\\
\mathbf{L} &= \mathbf{R}\mathbf{H} && (5.17)\\
\mathbf{F} &= \mathbf{H}^T\mathbf{L} && (5.18)
\end{aligned}$$

./ is elementwise division where the dot follows the notation as used in MATLAB. The equations $\mathbf{L} = \mathbf{R}\mathbf{H}$ and $\mathbf{F} = \mathbf{H}^T\mathbf{L}$ constitute the updating steps of the $\mathbf{L}$ and $\mathbf{F}$ kernel matrices. The grammian matrix $\mathbf{R}$ is computed *once* and not updated. The function φ is formula 5.7 applied to the asymmetric distance matrix $\mathbf{Q}$; i.e. the matrix is transformed such that the sum of the membership values for one object satisfy:

$$\sum_{k=1}^{K} u_{ik} = 1 \ , \quad \forall\, i. \tag{5.19}$$

The most computer demanding part of the algorithm is calculation of the different kernel matrices. In most analyses using FCM on a data set where the optimal number of clusters is unknown, repeated clusterings are necessary. It is also common to delete different objects and perform subclustering. In such a scheme it is easy to see that the **R** matrix can be computed once and used repeatedly in new analyses of the same data set. The other kernel matrices, however, need to be calculated again for new values of K. In paper VII, two methods are suggested, called FCM-M2 and FCM-M3 which utilize this fact when analyzing for several different K values. The FCM-M2 method is a trivial expansion of FCM-M since it just stores the **R** matrix computed the first time and uses it for the next K values. FCM-M3 is based on FCM-M2, but requires that the operator starts the clustering with a K which is the maximum number of clusters that is being tested for. The reason for this is that the **F** and **V** matrices for all analyses $K < K_{\text{max}}$ can be generated from $\mathbf{F}_{\text{max}}$ and $\mathbf{V}_{\text{max}}$ by just deleting the appropriate number of rows and/or columns. It may not be feasible to use the FCM-M3 method for all kinds of problems, but when it is possible, this is definitely the most FLOPS-efficient method. Subclustering is achieved by deleting rows and columns in **R**, **F** and rows in **L**.

Chapter 6

Conclusion

The following three separate conclusions of the thesis is presented:

- For relatively smooth spectra the B-splines compression approach is efficient for both storage and computation. The methods developed enables the investigator to work on the compressed representation instead of the original representation.

- An intuitive and powerful new notation for manipulating N-mode array equations has been constructed.

- A fast algorithm for fuzzy c-means clustering of data set with large number of variables has been constructed.

6.1 Possible future prospects

- There are several possible extensions and new developments which can be performed in connection with the compression approach. The algorithm for selecting the positions of knot points can be made more efficient. Other wavelet bases should be investigated for their efficiency in compression.

- Efforts should be made to construct a program that enables the investigator to program with the diagram notation. Such a program would be an N-mode analogue to MATLAB.
- The way the fuzzy c-means clustering algorithm was reprogrammed to be more efficient, should be applicable to other fuzzy clustering algorithms as well. Those algorithms which have particular interest are those where the clusters can be lines or planes (hyperplanes).

Bibliography

[1] I. Pelczer and S. Szalma. Multidimensional NMR and data processing. *Chem. Rev.*, 91:1507–1524, 1991.

[2] T. Hirschfeld. The hy-phen-ated methods. *Analytical Chemistry*, 52:297A–310A, 1980.

[3] T. Hirschfeld. Instrumentation in the next decade. *Science*, 230:286–291, 1985.

[4] H. Martens and T. Naes. *Multivariate Calibration.* John Wiley & Sons, New York, 1989.

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

[6] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemom. Intell. Lab. Syst.*, 2:37–52, 1987.

[7] S. Wold, A. Ruhe, H. Wold, and W.J. Dunn III. The collinearity problem in linear regression. the partial least squares (PLS) approach to generalized inverses. *SIAM Journal of Scientific and Statistical Computations*, 5:735–743, 1984.

[8] A. Lorber, L. Wangen, and B.R. Kowalski. A theoretical foundation for PLS. *J. Chemom.*, 1:19–31, 1987.

[9] J.H. Husø y and T.A. Ramstad. *Application of an efficient parallel IIR filter bank to image subband coding.* Elsevier, Amsterdam, 1990.

[10] H. Samet. The quadtree and related hierarchical data structures. *Computing Surveys*, 6:187–260, 1984.

[11] C. de Boor. *A practical guide to splines.* Applied Mathematics Sciences. Springer-Verlag, New York, 1978.

[12] G. Farin. *Curves and surfaces for computer aided geometric design, a practical guide.* Academic Press, Boston, 2nd. ed. edition, 1990.

[13] *MATLAB Reference Guide.* Natick, Mass. The MathWorks Inc., 1992.

[14] W. Cheney and D. Kincaid. *Numerical mathematics and computing.* Brooks/Cole Publishing Company, 1980.

[15] S. Pissanetsky. *Sparse matrix technology.* Academic Press, London, 1984.

[16] T.V Karstang and R.J. Eastgate. Multivariate calibration of an X-ray diffractometer by partial least squares regression. *Chemom. Intell. Lab. Syst.*, 2:209–219, 1987.

[17] J.R. Wilson. *Introduction to graph theory.* Longman Scientific & Technical, UK, Essex, 3rd edition, 1985.

[18] F. Lindgren, P. Geladi, and S. Wold. The kernel algorithm for PLS. *J. Chemom.*, 7:45–59, 1992.

[19] R. Manne. Personal communication.

[20] J.C. Bezdek. Numerical taxonomy with fuzzy sets. *Journal of Mathematical Biology*, 1:57–71, 1974.

Paper I

JOURNAL OF CHEMOMETRICS, VOL. 7, 61–73 (1993)

COMPRESSION OF *n*th-ORDER DATA ARRAYS BY B-SPLINES. PART 1: THEORY

BJØRN K. ALSBERG* AND OLAV M. KVALHEIM

Department of Chemistry, University of Bergen, Allegt. 41, N-5007 Bergen, Norway

SUMMARY

For efficient handling of very large data arrays, pretreatment by compression is mandatory. In the present paper B-spline methods are described as good candidates for such data array compression. The mathematical relation between the maximum entropy method for compression of data tables and the B-spline of zeroth degree is described together with the generalization of B-spline compression to *n*th-order data array tables in matrix and tensor algebra.

KEY WORDS Compression Multivariate analysis B-splines Maximum entropy

INTRODUCTION

Instruments producing higher-order arrays are becoming common in chemistry. In NMR spectroscopy, for example, the introduction of multipulse techniques such as COSY and NOESY has made it possible to identify molecular structure in more detail than provided by one-dimensional techniques.[1] Hyphenated chromatography with the acquisition of full UV and IR spectra is today an important technique for the analytical chemist.[2] By using multivariate methods such as principal component analysis (PCA),[3–5] principal component regression (PCR)[3] and partial least squares regression (PLS),[6,7] it is possible to compare and relate the spectra to one or several external variables (e.g. concentration). Usually, spectroscopic instruments produce large amounts of data which need to be processed. Since there is data redundancy, compression techniques can be applied with success. Although data arrays from one-dimensional spectroscopy (frequency versus transmittance) often need some kind of compression, the situation is much more dramatic for higher-order data arrays. It is possible to imagine objects of order $\omega \in [1, 2, 3, \ldots, O]$ which are to be compared. Data objects of order $\omega > 3$ or 4 are presently uncommon, but it is easy to imagine a relevant application, e.g. a comparison of several 3D NMR spectra which produces a 4D data array.

For a 2D IR[8] or a GC–IR spectrum recorded at 3000 wavelengths and 500 retention times the number of variables is the product of the number of elements along each order, i.e. 1 500 000. If data arrays of this size are to be handled, compression is necessary. For the compression to be of use in chemistry it must satisfy some important criteria: (i) for each spectrum the variables in the compressed representation *must* be comparable; (ii) the compressed representations must retain all important features of the spectrum; (iii) the compression must be reversible in the sense that it is possible to obtain an approximation of the original spectrum.

* Author to whom correspondence should be addressed.

Received 21 May 1992
Revised 1 October 1992

Item (ii) is especially important for all studies of relations between IR spectra and molecular structure/properties.

Even if the computer resources available today are powerful enough to analyse very large data sets, efficient compression is still important. Chemometric data exploration often results in several analyses of the same matrix or matrices. For very large data sets this is inefficient and will be slow even for today's moderate-size Unix workstations. Experience has shown that a PCA algorithm (written in C) on large third-order data sets of dimensions $[100 \times 256 \times 256]$ requires *several hours* of CPU time on a Solbourne S600 computer (Unix operating system) with 32 MB RAM, 2 GB hard disk, 30 MIPS and 3–4 Mflops. In addition to the *computational* advantage of using reduced-size data arrays, the reduction in file size may also be significant.

BASIC IDEAS BEHIND COMPRESSION IN CHEMOMETRICS

The conditions for the sampling method

PLS, PCR and PCA are methods often used for analysis of spectra where each sample spectrum is sampled at discrete wavelengths and represented as a vector. These vectors are used to build up the **X**-matrix which contains the various spectra as row vectors and each variable is designated a wavelength. This representation is very intuitive and is much used. The fact that the *ordering* of the variables has significance may seem trivial but has consequences for our thinking about compression. This ordering or index information contains information about e.g. the smoothness of the curves. If e.g. a parabola is to be analysed, it is represented as a vector of *sampled* points along the *x*-axis. This is why this representation of curves (or surfaces) has been referred to as the *sampling representation method.*[9] A random permutation of the vector containing the sampling points will disrupt the parabola form. The ordering of variables in PLS, PCA or PCR, however, is of no concern to the algorithm or the mathematics. A symmetry is involved where row *and* column permutation of the **X**-matrix is *invariant* with respect to the eigenvector directions or their corresponding eigenvalues. If $\mathbf{G}_{r,c}$ is a permutation operator either row-wise or column-wise, **T** is the score matrix and $\mathbf{P}^{\mathrm{T}}$ is the transposed loading matrix, this can be written as[9]

$$\mathbf{X} = \mathbf{TP}^{\mathrm{T}} \tag{1}$$

$$\mathbf{G}_r\mathbf{XG}_c = \mathbf{G}_r\mathbf{TP}^{\mathrm{T}}\mathbf{G}_c \tag{2}$$

Compression utilizes the fact that the vectors are indeed curves (or surfaces) with certain properties (e.g. smooth and continuous). If random shuffling of variables was performed, these properties would be lost and most compression methods which rely on interpolating curves would fail to obtain good compression rates. On the other hand, more optimal permutations of the data table columns may exist which enhance the compression rates.

Comparable objects

Compression of data tables cannot be done for each object separately. The reason for this is that the comparability of the object vectors (it is always possible to speak of vectors in this matter, because any data array can be unfolded to a vector) must be preserved. Variable v in object i must have the same meaning and describe the same phenomena as variable v in object j. Since the spectral structure is determined by the ordering of indices in the data table, the ordering for the whole data array *must* be the same. Any variable manipulation must be the

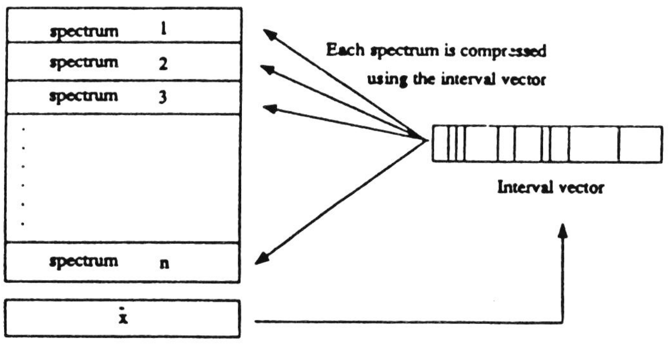


Figure 1. Illustration of the principle behind maximum entropy (and B-splines in general) for compression of a matrix of spectra. The same compression must be applied to every spectrum and the basis for selection of the interval vector is the mean spectrum. This may not be the optimal curve for interval vector selection with respect to compression

same for every object. This is of course not optimal from a compression rate viewpoint, since better compression rates may be attained if the spectra are compressed individually. This may be a logical solution if the only purpose of the compression is for storing of the data array. If the compression aims at reduction of data array elements for use in numerical calculations, the compression must retain the comparability for the data set as a whole. For this reason the mean spectrum is used as a guide to the total compression. Other types of spectra can be used, e.g. the variance spectrum or selected object vectors of importance, but this is a problem-relevant choice.

The importance of comparability can be further emphasized by an example. If **s** is a vector containing the uncompressed spectrum and F is some function which gives a molecular-property-related value p (here assumed to be a scalar), a desirable situation is obtained when

$$F: \mathcal{R}^n \mapsto \mathcal{R} \tag{3}$$

$$F_c: \mathcal{R}^m \mapsto \mathcal{R} \tag{4}$$

$$m \ll n \tag{5}$$

$$p = F(\mathbf{s}) = F_c(\mathbf{s}_c) \tag{6}$$

where $\mathbf{s}_c$ is the compressed representation of **s** and F_c is the analogous function to F which maps a vector into a scalar. In other words, the compressed representation should give the same p-value as the uncompressed representation.

The total compression *description* for data objects is contained in one or several interval vectors $\mathbf{h}^T$ and is applied to each object vector. An illustration of this is given in Figure 1. The number of such interval vectors is related to the order of the data array. Interval vectors are synonymous with *knot* vectors in spline theory, which is described later in this paper.

THE MAXIMUM ENTROPY METHOD

The term *entropy* as used here is not exactly the same as entropy in classical thermodynamics. When compressing curves, the *information* content in the curves is of central interest.

Therefore the entropy definition introduced by Shannon[10] is used:

$$E = -\sum_{i=1}^{Q} p_i \ln(p_i) \tag{7}$$

where the classical information theoretical interpretation of Q is the number of intervals subdividing a curve and p_i is the probability of an event occurring in the interval i. Traditionally p_i is calculated by dividing the frequency within that interval by the total frequency count. The important task is to define the interval such that any single event has equal probability of occurring in any interval. The word 'event' is replaced by the word 'variables' and the 'probability' is the sum of selected variables divided by the number of variables for an interval. A new number p_i (average or 'probability') is assigned to every interval; p_i is just given a different interpretation. By putting the different p_i (i.e. the compressed variables which now are used instead of several of the old variables) into formula (7), the intervals are divided such that equation (7) is *maximized.* The effect of this is that the overall curve profile has the maximum probability of being retained. The best number α of compressed variables p_i is chosen by selecting the right interval subdivision. The number α must be provided by the investigator and determines how accurate the compressed representation will be.

The *maximum entropy method* described below is constructed to produce intervals which maximize equation (7). The maximum entropy method as used in chemometrics was first presented as an algorithm:[11]

1. Calculate the mean spectrum $\bar{\mathbf{x}}^{\mathrm{T}}$ from the sample spectrum matrix $\mathbf{X}$.
2. If $\min(\bar{\mathbf{x}}^{\mathrm{T}}) < 0$, the negative values are eliminated by adding a constant to the spectrum (e.g. $|(\min(\bar{\mathbf{x}}^{\mathrm{T}})|$). The reason for this is the step in the algorithm which adds variable values together. The presence of both negative and positive values will create errors.
3. The $\bar{\mathbf{x}}^{\mathrm{T}}$-vector is used to generate an interval vector $\mathbf{h}^{\mathrm{T}}$. The interval vector $\mathbf{h}^{\mathrm{T}}$ defines the intervals of variables which are to be added together. if $\mathbf{h}^{\mathrm{T}} = [1\ 5\ 8\ 12\ 25]$, it is to be interpreted as definition of four intervals: $[1\ 5\rangle$, $[5\ 8\rangle$, $[8\ 12\rangle$ and $[12\ 25]$. If an interval is $[a\ b\rangle$, all the variables from *a up to b* are included but the *b*th variable itself is excluded. The steps for making the $\mathbf{h}^{\mathrm{T}}$-vector are as follows.
   (a) $I = (1/\alpha)\ \Sigma_{i=1}^{n}\ \bar{\mathbf{x}}_i^{\mathrm{T}}$, where α is the maximum number of *allowed* variables in the compressed representation.
   (b) Starting at the first variable, add consecutive intensities in the average spectrum until the partial sum is equal to or exceeds I. The position of the *next* variable to the last variable added in the partial sum is included in the vector $\mathbf{h}^{\mathrm{T}}$. If the last variable added was 3, the number 4 is stored, i.e. the next variable number.
4. For each row vector (spectral profile) in $\mathbf{X}$ do: sum the variables defined in the $\mathbf{h}^{\mathrm{T}}$-vector, including the first but not the last variable, i.e. if $\mathbf{h}^{\mathrm{T}}(3) = 5$ and $\mathbf{h}^{\mathrm{T}}(4) = 10$, the variables 5, 6, 7, 8 and 9 are summed but *not* 10. This variable will be summed in the next segment. This is illustrated in Figure 2.
5. The new variable (sum over interval) is divided by the number of variables in the sum to obtain an average value.

When the $\mathbf{h}^{\mathrm{T}}$-vector is present, the algorithm can be formulated in matrix algebra. Let $\bar{\mathbf{x}}^{\mathrm{T}} = [1{\cdot}2, 2{\cdot}3, 2{\cdot}1, 3{\cdot}4, 4{\cdot}4, 1{\cdot}1, 1{\cdot}0, 0{\cdot}4]$ be the average vector of some data table $\mathbf{X}$. An $\mathbf{h}^{\mathrm{T}}$ is constructed for $\alpha = 3$, i.e. the maximum allowed number of compressed variables must not be larger than three. The threshold intensity is $I = 15{\cdot}9/3 = 5{\cdot}3$. By definition $\mathbf{h}^{\mathrm{T}}(1) = 1$ and

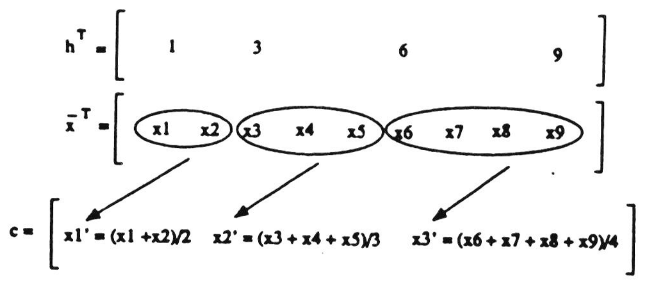


Figure 2. Detailed illustration of how the interval vector influences the summation of variables, i.e. the generation of compressed variables

$\mathbf{h}^T(\text{last}) = n$, where n is the number of elements in $\bar{\mathbf{x}}^T$ (in this example $n = 8$). Writing the detailed calculation of the partial summing process will illustrate the process further:

$$1\cdot 2 + 2\cdot 3 + 2\cdot 1 = 5\cdot 6 > 5\cdot 3 \Rightarrow \mathbf{h}^T(2) = 3 + 1 = 4 \tag{8}$$

$$3\cdot 4 + 4\cdot 4 = 7\cdot 8 > 5\cdot 3 \Rightarrow \mathbf{h}^T(3) = 5 + 1 = 6 \tag{9}$$

The sum of $1\cdot 1 + 1\cdot 0 + 0\cdot 4$ is not included above, since here is the endpoint situation where $\mathbf{h}^T(4) = 8$. In the interval $[1, 4\rangle$ there are three variables, the interval $[4, 6\rangle$ contains two variables and in the interval $[6, 8]$ three variables are located. The total interval vector $\mathbf{h}^T$ looks like

$$\mathbf{h}^T = [1, 4, 6, 8] \tag{10}$$

The calculation of the three compressed variables will now be

$$(1\cdot 2 + 2\cdot 3 + 2\cdot 1)/3 = 5\cdot 6/3 = 1\cdot 87 \tag{11}$$

$$(3\cdot 4 + 4\cdot 4)/2 = 7\cdot 8/2 = 3\cdot 90 \tag{12}$$

$$(1\cdot 1 + 1\cdot 0 + 0\cdot 4)/3 = 2\cdot 5/3 = 0\cdot 83 \tag{13}$$

This process can be formulated with matrix multiplication. The addition of the right variables given $\mathbf{h}^T$ can be done by multiplying $\bar{\mathbf{x}}^T$ by a special matrix

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

to give

$$\bar{\mathbf{x}}^T\mathbf{B} = [5\cdot 6, 7\cdot 8, 2\cdot 5] \tag{14}$$

Dividing by the number of elements can be done by using the **B**-matrix:

$$(\mathbf{B}^{\mathrm{T}}\mathbf{B})^{-1} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{3} \end{bmatrix} \tag{15}$$

By combining equation (15) with equation (16), the following is obtained:

$$\mathbf{c}^{\mathrm{T}} = [1{\cdot}87, 3{\cdot}9, 0{\cdot}83] = \bar{\mathbf{x}}^{\mathrm{T}}\mathbf{B}(\mathbf{B}^{\mathrm{T}}\mathbf{B})^{-1} \tag{16}$$

The compressed variables $\mathbf{c}^{\mathrm{T}}$ will also be referred to as *coefficients* or the *coefficient vector.* For a whole **X**-matrix this can be written as

$$\mathbf{C} = \mathbf{X}\mathbf{B}(\mathbf{B}^{\mathrm{T}}\mathbf{B})^{-1} \tag{17}$$

where **C** is the matrix containing the compressed variables. Given **C**, it is possible to calculate back into the **X**-domain:

$$\hat{\mathbf{X}} = \mathbf{C}\mathbf{B}^{\mathrm{T}} \tag{18}$$

SPLINE THEORY

Piecewise polynomial splines

In order to interpolate curves or surfaces, simple least squares optimization of the coefficients in a polynomial of the form

$$P(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_n x^n = \sum_{i=0}^{n} \alpha_i x^i$$

often generates oscillations and other undesired effects in real-world situations. The use of *splines* solves this problem by dividing a curve into a set of segments or intervals. A polynomial of some degree k is defined within the limits of the intervals. The intervals are defined by a set of points called *knots*. Within each interval a polynomial is defined. The polynomials are connected between neighbouring intervals such that continuity and differentiability are preserved. This type of spline is the *piecewise polynomial* spline. Given intervals s_j, there exists a set of polynomial coefficients c_{jk} associated with s_j. All splines must ensure that the function is continuous between knots. For splines of degree k the function has $k - 1$ continuous derivatives on an interval I. Each interval s_j is defined by two knots h_1 and h_2 (beginning and end of interval).

B-splines

B-splines[12–14] or *basis*-splines represent curves as linear combinations of basis functions. Each basis function is described by its knot sequence and often has a similar form to a Gaussian curve; see Figure 3, where the knot distribution is uniform. When the knot distribution is not uniform, the basis functions will differ in size and shape. The coefficient associated with each basis function may be interpreted as the height of the basis function. The basis function form and shape are defined by a recursive algorithm described below which uses the knot distribution.

It is possible to construct a function f as a linear combination of basis functions (in the 1D

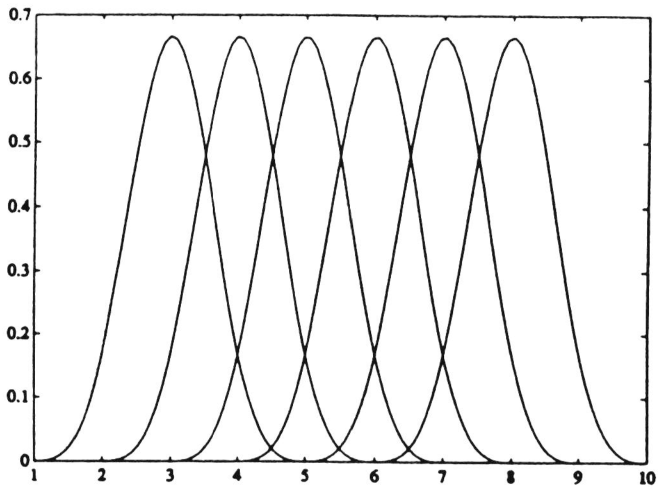


Figure 3. Six B-spline basis functions depicted separately for $k = 3$ (cubic interpolation). It is important to note that the shape of basis functions is determined by the positions of the knots. The coefficients related to very basis function determine the ***height*** of the basis function in the function to be interpolated

case) (see Reference 15, pp. 163–176):

$$f(x) = \sum_{i=1}^{n} c_i B_{i,k}(x) \tag{19}$$

where $B_{i,k}(x)$ is the basis function. The function f is therefore described by its knot/interval vector $\mathbf{h}^T$ and coefficient vector $\mathbf{c}^T$. The knot vector is a non-decreasing sequence of numbers*

$$h_0 \leqslant h_1 \leqslant \cdots \leqslant h_{n+k}$$

where k is the maximum degree of any polynomial.

$B_{i,k}$ is the basis and is defined by a recursive formula (see Reference 15, p. 164):

$$B_{j,k}(x) = \frac{x - h_j}{h_{j+k} - h_j} b_{j,k-1}(x) + \frac{h_{j+k+1} - x}{h_{j+k+1} - h_{j+1}} B_{j+1,k-1}(x) \quad k \geqslant 1 \tag{20}$$

$$j = 0, \pm 1, \pm 2, \ldots, \qquad k = 1, 2, 3, \ldots$$

For $B_{i,0}(x)$ we have

$$B_{i,0} = \begin{cases} 1 & \text{if } h_i \leqslant x \leqslant h_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

This function is depicted in Figure 4.

There can be several knots located at the same point along the x-axis. This is called *multiplicity* and influences the smoothness.

The B-matrix

If $\mathbf{h}^T$ is a knot sequence, $B_{i,k}(x)$ can be represented as a matrix $\mathbf{B}$ and computed for different values of k.

*The notation t_i is usually used in spline theory to designate the knot sequence. This letter is not chosen here since it is usually associated with score values.

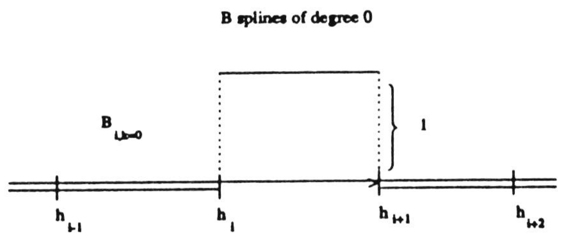


Figure 4. Basis function for B-splines of degree zero

The same interval vector $\mathbf{h}^T$ as in equation (10) is used to represent knots in B-spline bases and equation (20) is used to generate the **B**-matrices for $k = 0, 1, 2$:

$$\mathbf{B}_0 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{B}_1 = \begin{bmatrix} 0 & 0 \\ 0\cdot3333 & 0 \\ 0\cdot6667 & 0 \\ 1\cdot0000 & 0 \\ 0\cdot5000 & 0\cdot5000 \\ 0 & 1\cdot0000 \\ 0 & 0\cdot5000 \\ 0 & 0 \end{bmatrix} \tag{22}$$

$$\mathbf{B}_2 = \begin{bmatrix} 0 \\ 0\cdot0667 \\ 0\cdot2667 \\ 0\cdot6000 \\ 0\cdot7750 \\ 0\cdot5000 \\ 0\cdot1250 \\ 0 \end{bmatrix} \tag{23}$$

The effect of enlarging the k-value is that the curve can be represented with fewer parameters. This is only true if the fitted data have a smoothness corresponding to k. i.e. containing $k - 1$ derivatives (assuming no multiplicity).

A matrix formulation of equation (19) can be made. Let $\mathbf{x}^T$ be the curve vector, $\mathbf{c}^T$ the coefficient vector and **B** the function $B_{i,k}(x)$ represented as a matrix. Then

$$\mathbf{x}^T = \mathbf{c}^T\mathbf{B}^T \tag{24}$$

which written in terms of $\mathbf{c}^T$ is

$$\mathbf{c}^T = \mathbf{x}^T\mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1} \tag{25}$$

For several $\mathbf{x}^T$s we have

$$\mathbf{C} = \mathbf{X}\mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1} \tag{26}$$

which is exactly the same equation as (17). This leads to the following conclusion: *The B-spline of zero degree is the same method as what has been called the maximum entropy method in the chemometrics literature.*[16] This method is hereafter referred to as B0 to indicate it is a B-spline of zero degree. B-spline compression of degree *n* will be referred to as B*n* methods.

Generally, when presented with a matrix **C** the estimated $\hat{\mathbf{X}}$ can be written as

$$\hat{\mathbf{X}} = \mathbf{C}\mathbf{B}^T \tag{27}$$

2D method

When presented to higher-order data tables, it has been shown that unfolding[9,17] is an efficient method for representing higher-order data tables as matrices. Some would perhaps suggest the 1D compression should be applied to the unfolded vectors of the higher-order objects. This is of course possible, but certainly a suboptimal strategy for compression, since the unfolding process (a) introduces artefacts from the unfolding which require additional information for representation and (b) destroys the spatial correlation between neighbouring points in the higher-order data table. Compression for 2D, 3D and *n*D data tables is described below without the use of unfolding.

Multivariate splines have been described[18] and it is therefore natural to propose the use of multivariate B-splines for compression of higher-order data arrays. It is now natural to apply the multivariate extension to the B0 method and B*n* methods in general. For 2D methods two knot/interval vectors $\mathbf{h}_1$ and $\mathbf{h}_2$ are needed (see Figure 5). For *n*D methods *n* such knot/interval vectors must be used.

The methods presented are all formulated in matrix and tensor algebra. First the matrix equation for B*n* methods in the 2D case is formulated where the result in equation (26) is used (the subscripts on the $\mathbf{B}_i$-matrices indicate different types of compression matrices for the

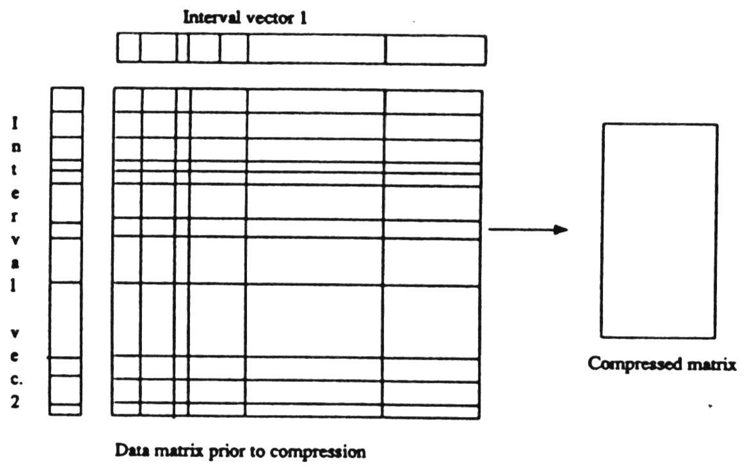


Figure 5. For 2D compression two (not necessary identical) interval vectors compress along the two different orders of the data matrix. The rectangles in the data matrix defined by the Cartesian product of the two interval vectors are represented by a single coefficient

different orders; the same also applies for the $\mathbf{C}_i$-matrices):

$$\mathbf{C}_a = \mathbf{X}\mathbf{B}_1(\mathbf{B}_1^{\mathrm{T}}\mathbf{B}_1)^{-1} \tag{28}$$

$$\mathbf{C}_b = \mathbf{C}_a^{\mathrm{T}}\mathbf{B}_2(\mathbf{B}_2^{\mathrm{T}}\mathbf{B}_2)^{-1} \tag{29}$$

$$\mathbf{C}_b = [\mathbf{X}\mathbf{B}_1(\mathbf{B}_1^{\mathrm{T}}\mathbf{B}_1)^{-1}]^{\mathrm{T}}\mathbf{B}_2(\mathbf{B}_2^{\mathrm{T}}\mathbf{B}_2)^{-1} \tag{30}$$

$$\mathbf{C}_b = [(\mathbf{B}_1^{\mathrm{T}}\mathbf{B}_1)^{-1}]^{\mathrm{T}}\mathbf{B}_1^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{B}_2(\mathbf{B}_2^{\mathrm{T}}\mathbf{B}_2)^{-1} \tag{31}$$

$$\mathbf{R}_i = \mathbf{B}_i(\mathbf{B}_i^{\mathrm{T}}\mathbf{B}_i)^{-1} \tag{32}$$

$$\mathbf{C}_b = \mathbf{R}_1^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{R}_2 \tag{33}$$

The subscripts a and b are used to discriminate between a matrix compressed along one order only (a) and along two orders (b). It is easy to see that the $\mathbf{C}_b$-matrix is the same whether the compression is started along the first or second order. The compression matrices $\mathbf{R}_1$ and $\mathbf{R}_2$ are associated with $\mathbf{X}$ and $\mathbf{X}^{\mathrm{T}}$ respectively. If $\mathbf{X}$ has dimensions $[i \times j]$, the dimensions of $\mathbf{R}_1$ are $[j \times c_1]$ and $\mathbf{R}_2$ has dimensions $[i \times c_2]$. It is easy to see that $\mathbf{C}_a$ has dimensions $[i \times c_1]$ if $\mathbf{R}_1$ has been used first and $\mathbf{C}_b$ has dimensions $[c_1 \times c_2]$. This can be further illustrated by considering an example $\mathbf{X}$-matrix with e.g. dimensions $[6 \times 8]$ and where the knot vectors are $\mathbf{h}_1^{\mathrm{T}} = [h_0, h_1, h_2, h_3]$ and $\mathbf{h}_2^{\mathrm{T}} = [h_0, h_1, h_2]$. The dimensions of $\mathbf{R}_1 = \mathbf{B}_1(\mathbf{B}_1^{\mathrm{T}}\mathbf{B}_1)^{-1}$ are $[8 \times 3]([3 \times 8][8 \times 3]) = [8 \times 3]$. For $\mathbf{R}_2 = \mathbf{B}_2(\mathbf{B}_2^{\mathrm{T}}\mathbf{B}_2)^{-1}$ the dimensions are $[6 \times 2]([2 \times 6][6 \times 2]) = [6 \times 2]$. The dimensions of $\mathbf{C}_a = \mathbf{X}\mathbf{R}_1$ are $[6 \times 8][8 \times 3] = [6 \times 3]$. The dimensions of $\mathbf{C}_b = \mathbf{C}_a^{\mathrm{T}}\mathbf{R}_2$ are $[3 \times 6][6 \times 2] = [3 \times 2]$.

Starting with $\mathbf{R}_2$ instead of $\mathbf{R}_1$ gives (subscript 1 or 2 is added to a and b to indicate where the calculation is started from)

$$\mathbf{C}_{a2} = \mathbf{X}^{\mathrm{T}}\mathbf{R}_2 \tag{34}$$

$$\mathbf{C}_{b2} = (\mathbf{C}_{a2})^{\mathrm{T}}\mathbf{R}_1 \tag{35}$$

$$\mathbf{C}_{b2} = \mathbf{R}_2^{\mathrm{T}}\mathbf{X}\mathbf{R}_1 \tag{36}$$

$$\mathbf{C}_{b1} = \mathbf{R}_1^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{R}_2 = (\mathbf{R}_2^{\mathrm{T}}\mathbf{X}\mathbf{R}_1)^{\mathrm{T}} = \mathbf{C}_{b2}^{\mathrm{T}} \tag{37}$$

which proves that the same matrix is obtained whether $\mathbf{R}_1$ or $\mathbf{R}_2$ is used first. The conclusion is that the different orders are independent of each other with respect to the order of multiplication of the $\mathbf{R}_i$-matrices.

The subscripts of $\mathbf{R}$ indicate that the compression may be different (it usually is) along the different orders. The process of compressing along two orders is depicted in Figure 5. Generalizing this for a stack of 2D samples requires the introduction of tensors.[19] If $\mathbf{X}^3$ is a third-order tensor (three-way data array) with order $(i \times j \times f)$, it is possible to write

$$\mathbf{C}^{(3)} = \mathbf{R}_1^{\mathrm{T}}\mathbf{X}^{(3)}\mathbf{R}_2 \tag{38}$$

where $\mathbf{C}^{(3)}$ has dimensions $(i \times c_1 \times c_2)$, $\mathbf{R}_1$ has dimensions $(j \times c_1)$ and $\mathbf{R}_2$ has dimensions $(f \times c_2)$.

3D method

The 3D compression method can be formulated as an extension of equation (38). Here the notation convention of Reference 19 is used (see Figure 6):

$$\mathbf{C}^{(3)} = \mathbf{R}_1^{\mathrm{T}} \overset{\mathbf{R}_3}{\underset{\mathbf{X}^{(3)}}{}} \mathbf{R}_2 \tag{39}$$

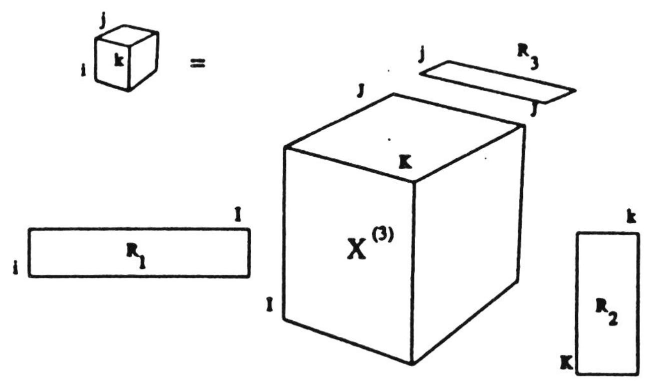


Figure 6. This figure is similar to Figure 7 but the details of the $\mathbf{R}_i$-matrices used in the compression are shown. Each compression matrix $\mathbf{R}_i$ (i.e. for order i) can be applied independently of the other compression matrices. The placement of the compression matrices is an attempt to illustrate that each matrix is associated with the respective order of the original tensor

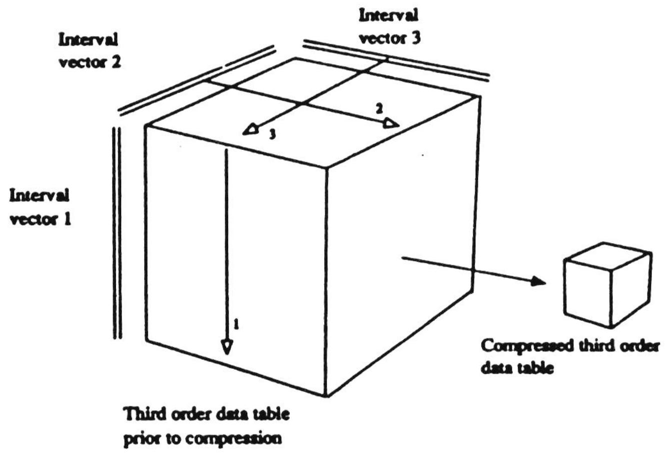

Interval vector 2
Interval vector 3
Interval vector 1
2
3
1
Compressed third order data table
Third order data table prior to compression

Figure 7. Illustration of 3D compression which is analogous to Figure 5. Here open-headed arrows emphasize along which data array order direction each interval vector performs

where $\mathbf{R}_3$ has dimensions $(i \times C_3)$. Equation (39) and its corresponding interval vectors are illustrated in Figure 7.

Generalization to *n*th-order

A formula for *n*th-order compression cannot use the notation described in equation (39) for typographical reasons. In general a compression matrix $\mathbf{R}_i$ is associated with each order of the tensor. If the subscript i of the $\mathbf{R}_i$-matrix is associated with the order of the data array (tensor) $\mathbf{X}^{(n)}$, it is possible to suggest a general formula for compression of *n*th-order data arrays:

$$\mathbf{C}^{(n)} = (\mathbf{R}^n \ldots ((\mathbf{R}_2(\mathbf{R}_1\mathbf{X}^{(n)})) \ldots) \tag{40}$$

CONCLUSIONS

It has been demonstrated that the maximum entropy method is a zeroth-degree B-spline where the construction of the knot sequence follows the maximum entropy criterion. This fact suggests the use of higher-degree B-splines when presented with curves which are smooth, which is determined by continuity of the kth derivative. Since the compression methods are presented in matrix algebra, the generalization to nth-order data arrays (tensors) is logical.

ACKNOWLEDGEMENT

The authors wish to thank Tom Kavli at SI and a referee for valuable comments on the paper.

APPENDIX: NOTATION

All scalars are displayed in italic lowercase letters. All vectors are displayed in boldface lowercase letters.

All matrices are displayed in boldface uppercase letters. When dealing with data arrays of any order, it is necessary to distinguish these from data arrays of order two (ordinary matrices). There are different uses of the term 'tensor'. Here tensor is used as an extension of the matrix in several orders. In physics, however, the tensor definition contains constraints about invariance to rotations and translations.

$\mathbf{X}^{(3)}$ is a third-order data array (tensor of order three). If several data arrays are presented which it is necessary to identify, this can be written as $\mathbf{X}_i^{(n)}$ or $\mathbf{X}_i$.

Row vectors are denoted as transposed column vectors, e.g. $\mathbf{v}^{\mathrm{T}}$. However, the matrix $\mathbf{V}$ composed of the row vectors $\mathbf{v}_i^{\mathrm{T}}$ does *not* contain the transpose sign.

REFERENCES

1. I. Pelczer and S. Szalma, *Chem. Rev.* **91**, 1507–1524 (1991).
2. R. A. Nyquist, M. A. Leugers, M. L. McKelvy, R. R. Papenfuss, C. L. Putzig and L. Yurga, *Anal. Chem.* **62**, 223R–255R (1990).
3. H. Martens and T. Naes, *Multivariate Calibration*, Wiley, Chichester (1989).
4. I. Cowe and J. W. McNicol, *Appl. Spectrosc.* **39**, 257–266 (1985).
5. S. Wold, K. Esbensen and P. Geladi, *Chemometrics Intell. Lab. Syst.* **2**, 37–52 (1987).
6. S. Wold, A. Ruhe, H. Wold and W. J. Dunn III, *SIAM J. Sci. Stat. Comput.* **5**, 735–743 (1984).
7. A. Lorber, L. Wangen and B. R. Kowalski, *J. Chemometrics*, **1**, 19–31 (1987).
8. I. Noda, *Appl. Spectrosc.* **44**, 550–561 (1990).
9. B. K. Alsberg and B. G. Remseth, *J. Chemometrics*, **6**, 135–150 (1992).
10. C. E. Shannon, *Bell Syst. Tech. J.* **379**, 623 (1948).
11. T. V. Karstang and R. J. Eastgate, *Chemometrics Intell. Lab. Syst.* **2**, 209–219 (1987).
12. C. de Boor, *A Practical Guide to Splines*, Springer, New York/Heidelberg/Berlin (1978).
13. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design, a Pratical Guide*, 2nd edn, Academic, New York (1990).
14. The MATLAB spline toolbox manual by Carl de Boor.
15. W. Cheney and D. Kincaid, *Numerical Mathematics and Computing*, Brooks/Cole, Monterey, California (1980).
16. A. A. Christy, R. A. Velapoldi, T. V. Karstang, O. M. Kvalheim, E. Sletten and N. Telnaes, *Chemometrics Intell. Lab. Syst.* **2**, 199–207, (1987).

17. K. H. Esbensen and S. Wold, in *Proc. Nordic Symp. on Applied Statistics*, ed. by Christie, pp. 11–36, Stokkland (1983).
18. C. K. Chui, *Multivariate Splines*, CBMS–NSF Regional Conference Series in Applied Mathematics, Philadelphia, PA (1988).
19. P. Geladi, *Chemometrics Intell. Lab. Syst.* 7, 11–30, (1989).

Paper II

JOURNAL OF CHEMOMETRICS, VOL. 8, 127–145 (1994)

COMPRESSION OF *n*th-ORDER DATA ARRAYS BY B-SPLINES. PART 2: APPLICATION TO SECOND-ORDER FT-IR SPECTRA

BJØRN K. ALSBERG,* EGIL NODLAND AND OLAV M. KVALHEIM
Department of Chemistry, University of Bergen, Allegt. 41, N-5007 Bergen, Norway

SUMMARY

In order to improve the storage and CPU time in the numerical analysis of large two-dimensional (hyphenated, second-order) infrared spectra, a data-preprocessing technique (compression) is presented which is based on *B-splines*. B-splines have been chosen as the compression method since they are well-suited to model smooth curves. There are two primary goals of compression: a reduction of file size and a reduction of computation when analyzing the compressed representation. The compressed representation of the spectra is used as a substitute for the original representation. For the particular example used here, approximately 0·16 bit per data element was required for the compressed representation in contrast with 16 bits per data element in the uncompressed representation. The compressed representation was further analysed using principal component analysis and compared with a similar analysis on the original data set. The results shows that the principal compotent model of the compressed representation is directly comparable with the principal component model of the original data.

KEY WORDS Compression Multivariate analysis B-splines FT-IR spectra Second-order Two-dimensional Hyphenated methods

1. INTRODUCTION

New instruments produce huge amounts of data which put new demands on the storage and processing time used in the analysis of spectra. Higher-order spectra often comprise large data arrays and have become common in both NMR and IR spectroscopy.[1,2] Such spectra provide the basis for a better understanding of e.g. spin systems (COSY, 3D-COSY, etc.) or the number of components in a solution[3,4] than is possible using traditional one-dimensional methods. There are at least two ways to solve the problem with large data sets:

(i) increase the computer resources (larger hard discs, more RAM, faster CPU, etc.)
(ii) use of compression.

In this paper compression is described as a practical solution to the large data array problem. For compression to be of practical use in chemistry, the following requirements must be fulfilled.

* Author to whom correspondence should be addressed.

Received 22 April 1993
Accepted 9 September 1993

1. It must be possible to compress the spectra using the chosen method. Here the B-spline[5-7] method is chosen, which assume some degree of local smoothness in each spectrum. Otherwise the number of bytes required to represent the spectra will be too large. It is of course possible to compress other types of signals efficiently, but this is not discussed here.
2. The compression method must not be too computer-demanding.
3. The discrepancy between the reconstructed (from the compressed representation) and original data must not be too severe.
4. The compressed representation itself should be *operational*, i.e. common numerical methods can be applied to the compressed representation *in the same manner* as for the original uncompressed representation.

Item 1 is related to item 3. Algorithms for *perfect* reconstruction of the original data already exist (e.g. Huffman, Ziv-Lempel[8]), but a higher degree of compression can be accomplished if some discrepancy is allowed. Often spectra contain errors and there are no reasons for using bits to store noise. Of course, this puts more demand on the investigator, who is responsible for what kind of compression is admissible. Item 2 depends on whether the compressed representation (**C**) is to be used one or several times. In this paper it is assumed that **C** will be used *several* times and therefore more computing time in the compression step can be allowed. Often the investigator wants to use several different techniques when analyzing **C**. If the compression takes an excessive amount of time, it cannot be considered effective for practical problems. The exact meaning of the term 'excessive amount of time' depends on the local user's computer resources and time available. Item 4 is important. In the experiments described in this paper the original data (**X**) and **C** are *comparable*. This means that the numerical algorithms used for analysis of the data can be used without modifications on both the original and compressed representations. An example of a matrix representation which requires special algorithms is that of *sparse matrices.*[9] Today this technology is so common that several available packages can take care of this. It is possible to compress matrices using e.g. run-length coding,[8] which produces non-comparable representations which in principle can be used in numerical algorithms *if they have been modified to handle the new data structure*. Thus the aim of this paper is to present a method which generates a compressed representation **C** from an original large representation **X** and which satisfies the four requirements presented above.

2. METHODS

2.1. General description of method

The basic idea behind the compression used here is as follows: find a set of relatively smooth basis functions and represent the original matrix as linear combinations of these basis functions. Each spectrum is thus regarded as a *function* rather than a vector.[10] For second-order (two-dimensional) data two basis sets are needed which will be referred to as $\mathbf{B}_1$ and $\mathbf{B}_2$. The generation and use of these two basis set matrices are described in detail in Section 2.2. The coefficients found in the linear combinations are used instead of the original representation (see Figure 1). Given a matrix **X** with dimensions $[n, m]$, this is replaced in future computations (and storage) by a smaller matrix **C** with dimensions $[c_1, c_2]$, where $c_1 < n$ and $c_2 < m$. We have chosen to use B-splines[11] for construction of the smooth basis functions, but other basis functions could have been used as well. The discrete Fourier transform (DFT) is

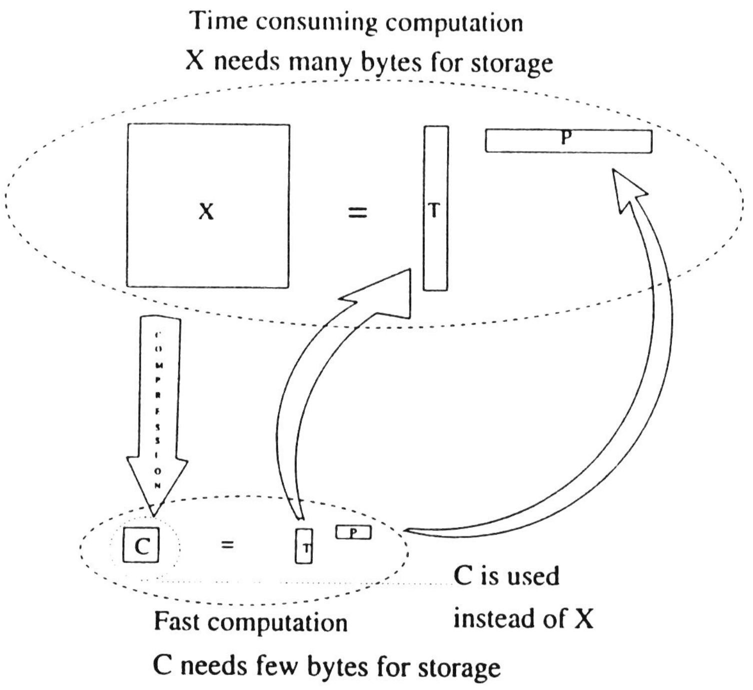

Time consuming computation
X needs many bytes for storage
X
=
T
P
COMPRESSION
C
=
T
P
C is used instead of X
Fast computation
C needs few bytes for storage

Figure 1. Illustration of the basic idea behind the compression

such an example and the discrete cosine transform (DCT) another; both methods form the backbone of several compression strategies in e.g. image and video compression.[12-14] Another family of increasingly popular basis functions for compression is *wavelets.*[15]

In this paper the numerical method used for analysing **X** and **C** will be principal component analysis (PCA)[16-18] and thus hereafter PCA will be discussed in particular instead of just any numerical method. In PCA the matrix under investigation is decomposed into the product of two orthogonal matrices. For **X** we have

$$\mathbf{X} = \mathbf{T}\mathbf{P}^{\mathrm{T}} + \mathbf{E} \tag{1}$$

where **T** is the score matrix, $\mathbf{P}^{\mathrm{T}}$ is the loading matrix and **E** is the error matrix. **T** corresponds to the eigenvectors of the covariance matrix $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ and $\mathbf{P}^{\mathrm{T}}$ corresponds to the eigenvectors of the covariance matrix $\mathbf{X}^{\mathrm{T}}\mathbf{X}$. A popular method for estimating the score and loading vectors is the NIPALS (non-linear iterative partial least squares) algorithm,[19] which calculates the eigenvectors one at a time starting with those having the largest eigenvalues. Plotting such eigenvectors (either scores or loadings) is a powerful tool for understanding underlying tendencies in the data set.

PCA of the compressed matrix **C** gives correspondingly

$$\mathbf{C} = \mathbf{T}_{\mathrm{c}}\mathbf{P}_{\mathrm{c}}^{\mathrm{T}} + \mathbf{E}_{\mathrm{c}} \tag{2}$$

One way of transforming $\mathbf{T}_{\mathrm{c}} \rightarrow \hat{\mathbf{T}}$ and $\mathbf{P}_{\mathrm{c}}^{\mathrm{T}} \rightarrow \hat{\mathbf{P}}^{\mathrm{T}}$ is

$$\hat{\mathbf{T}} = \mathbf{B}_2\mathbf{T}_{\mathrm{c}} \tag{3}$$

$$\hat{\mathbf{P}}^{\mathrm{T}} = \mathbf{P}_{\mathrm{c}}^{\mathrm{T}}\mathbf{B}_1^{\mathrm{T}} \tag{4}$$

Unfortunately, for non-orthonormal $\mathbf{B}_1$ and $\mathbf{B}_2$ (which is the most common situation) the estimated score and loading matrices are not orthogonal as they should be. Simple orthonormalization of $\hat{\mathbf{T}}$ and $\hat{\mathbf{P}}^T$ will not give the correct result. Since B-splines are used, this effect is minimized by the fact that the B-spline basis matrix is *diagonally dominant*. For the experiment in this paper it is shown that the major features of the original model of $\mathbf{X}$ are present as more crude versions in the model of $\mathbf{C}$.

The discrepancy between the results from analysis of $\mathbf{C}$ and $\mathbf{X}$ must always be kept in mind when interpreting the data. For many situations the model from $\mathbf{C}$ will give the investigator an adequate idea of the latent variable structure in the original data set.

2.2. B-splines

B-splines[5-7] represent curves as linear combinations of basis functions. Each basis function is fully described by a vector of real numbers which determines its size and shape. This vector ($\mathbf{h}^T$) is referred to as a *knot vector*. The knots are values corresponding to positions along the independent variable. The B-spline basis set ($B_j(x)$, $j \in [1, \ldots, q-k-1]$, where q is the number of elements in the knot vector and k is the degree of the spline) is completely described by the knot vector

$$\mathbf{h}^T \rightarrow \{B_1(x), B_2(x), \ldots, B_{q-k-1}(x)\} \tag{5}$$

The formula for generating the B-spline basis from the knot vector is given in Appendix I.

A function f can thus be formulated as a linear combination of $B_j(x)$:

$$f(x) = \sum_{j=1}^{n} c_j B_j(x) \tag{6}$$

Therefore f is described by its knot vector $\mathbf{h}^T$ and coefficient vector $\mathbf{c}^T$ only.

The extension to two dimensions is straightforward. Two knot vectors $\mathbf{h}_1$ and $\mathbf{h}_2$ are needed, one for each order/dimension of the data set (see Figure 2). In Part 1 of this series[11] the following equation was used to obtain the coefficient matrix $\mathbf{C}$:

$$\mathbf{C} = [(\mathbf{B}_2^T\mathbf{B}_2)^{-1}]\mathbf{B}_2^T\mathbf{X}\mathbf{B}_1(\mathbf{B}_1^T\mathbf{B}_1)^{-1} \tag{7}$$

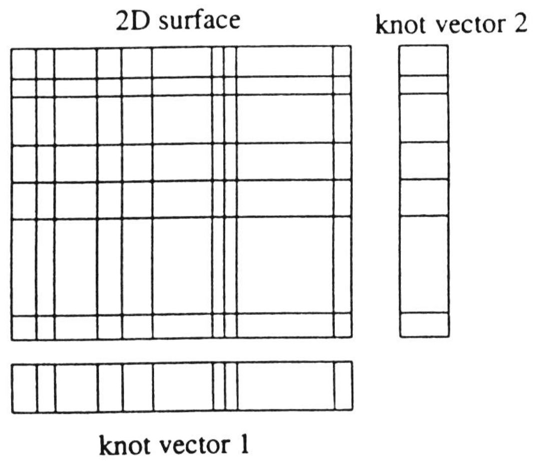


Figure 2. For two-dimensional (or second-order) problems in B-splines two knot vectors $\mathbf{h}_1$ and $\mathbf{h}_2$ are needed

By introducing

$$\mathbf{R}_i = \mathbf{B}_i(\mathbf{B}_i^{\mathrm{T}}\mathbf{B}_i)^{-1}, \quad i \in [1,2] \tag{8}$$

C can be written more compactly as

$$\mathbf{C} = \mathbf{R}_2^{\mathrm{T}}\mathbf{X}\mathbf{R}_1 \tag{9}$$

The subscripts on **R** indicate that the compression may be different (it usually is) along the two orders. The reconstruction can be written as

$$\hat{\mathbf{X}} = \mathbf{B}_2\mathbf{C}\mathbf{B}_1^{\mathrm{T}} \tag{10}$$

3. MEASURING AND EVALUATING COMPRESSION

3.1. Scalar quantization

In order to evaluate the number of bits required to represent the coefficients, *quantization*[20] is necessary. MATLAB (a program from MathWorks Inc.) uses 64 bit representation of its numbers, which is much more than the original 16 bit representation of the data from the Nicolet 800 instrument.

The basic idea behind scalar quantization (SQ) is to represent the data in terms of a finite number of basis symbols ($\mathcal{K}$). This set of symbols is referred to as the *source alphabet*. Even if the input is continuous, only a few output values are allowed. A simple example will illustrate the principle. Assume a measured signal with values between zero and unity. A simple source alphabet $\mathcal{K}$ may look like

$$\mathcal{K} = \{0, 0{\cdot}2, 0{\cdot}4, 0{\cdot}6, 1{\cdot}0\} \tag{11}$$

An incoming sequence of continuous signals is e.g.

$$\mathbf{v} = \{0{\cdot}2190, 0{\cdot}0470, 0{\cdot}6789, 0{\cdot}6793, 0{\cdot}9347, 0{\cdot}3835, 0{\cdot}5194\} \tag{12}$$

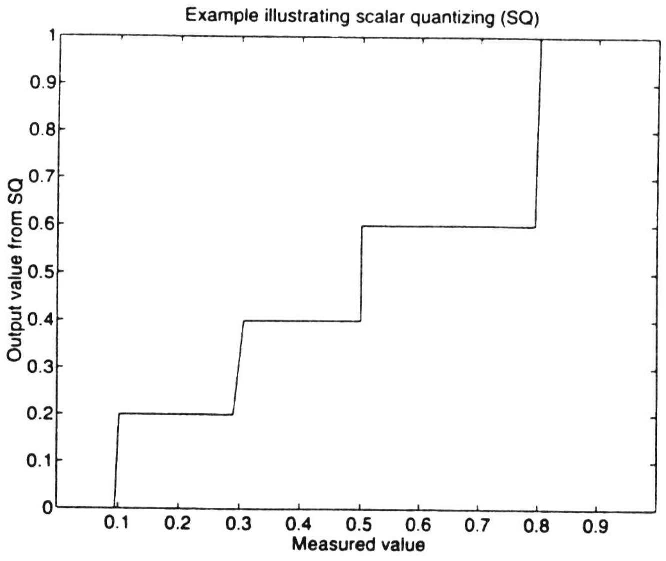


Figure 3. Result from an SQ of a set of 300 random numbers in the range zero to unity using the source symbols described in the text

For each element in $\mathbf{v}$ the value is checked against the values stored in $\mathcal{X}$ and the current element is assigned the value in $\mathcal{X}$ to which it is closest to. The scalar-quantized vector $\mathbf{v}_s$ obtained from $\mathbf{v}$ using the source alphabet $\mathcal{X}$ is

$$\mathbf{v}_s = \{0{\cdot}2000, 0, 0{\cdot}6000, 0{\cdot}6000, 1{\cdot}0000, 0{\cdot}4000, 0{\cdot}6000\} \tag{13}$$

In Figure 3 a sequence of 300 random symbols was converted using $\mathcal{X}$ to an output signal $\mathbf{v}_s$ and the characteristic stair-like structure is clearly visible. Appendix II treats quantization in more detail.

3.2. Measuring information

Given a vector of size N, each element can be addressed using $b = \log_2(N)$ bits. For the small example ($\mathbf{v}_s$) above $N = 7$ and thus $b = \log_2(7) = 2{\cdot}8074$, but on average the number of bits per symbol required will in general be smaller. b bits are needed when the probability of each symbol in the sequence is equal. This can be shown using the entropy[21] formula

$$H = -\sum_{i=1}^{N} p_i \log_2(p_i) \text{ bits/symbol} \tag{14}$$

where p_i is the probability of a symbol occurring in a sequence. If $p_i = p_j$, $i \neq j$, and $p_i = 1/N$ is assumed and used in equation (14), the maximum entropy is obtained as

$$H_{\max} = -Np_i \log_2(p_i) = -\log_2(p_i) = -\log_2(1/N) = \log_2(N) \tag{15}$$

The *redundancy* in information per symbol can now be defined as

$$r = \log_2(N) - H \tag{16}$$

For the small example above the number of bits needed per data element in $\mathbf{v}_s$ becomes $H = 2{\cdot}1281$. Here it is the symbol 0·6 which makes the number of bits per data element smaller since it occurs three times (all the others occur just once). For the example $r = 2{\cdot}8074 - 2{\cdot}1281 = 0{\cdot}6793$.

There are several strategies for selecting the SQ source alphabet. Most strategies base the selection on some assumption or knowledge of the *probability density function* (PDF), which corresponds to the histogram distribution of discrete signals. For SQ of an unknown signal evolving in time the source alphabet should be constructed on the basis of the most probable PDF. In the experiment described here it was possible to select the source alphabet *after* measuring the data distribution. The intuitive way of selecting source symbols is to have more symbols where the frequency of values is high. The actual method used here is described in detail in Section 4.

3.3. Compression evaluation

Since compression as used here cannot reproduce the original signal profiles perfectly, deviations from the original representation are inevitable. One possible way of quantifying the error would be to compute an error measure, e.g.

$$\varepsilon = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}(|\mathbf{X}_{ij} - \hat{\mathbf{X}}_{ij}|)\mathbf{W}_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{m}|\mathbf{X}_{ij} - \bar{\mathbf{X}}_{ij}|} \tag{17}$$

Here **W** is a weighting matrix which weights down certain areas which are not regarded as important for the evaluation of the compression. The reason for this is explained below.

Of course, if ε is close to zero, the discrepancy between the original and reconstructed representations is small and this will probably signify an excellent reconstruction. As the error measure increases, the problem is to tell whether one compression is better than another. The same problem occurs in image compression, where images with a very good error measure can sometimes have a very low perceptual quality. The reason for this might be that critical but small regions of the image can be far from the true value but large uninteresting areas are well explained by the compression method. One can imagine a picture of a house in a desert: if the house is poorly represented but the sand is well represented, this is not acceptable. In chemistry and spectroscopy the same problem occurs. Some areas of the spectrum are much more important than others. In such instances one may solve the problem by introducing the matrix **W** which weights down unimportant areas. In addition, the overall quality of the spectrum must be retained. It is also possible to imagine a spectrum with an acceptable error measure which contains unnecessary local oscillations around the true spectrum profile. Another requirement which was emphasized in Reference 11 is that physical quantities computed from the spectrum profile should be as close as possible to the original value. Unfortunately, such quantities cannot be used directly as 'objective' measurements of error, only as a guidance in the evaluation of the compression performance. Owing to these problems, we have chosen not to use exclusively ε as the final and only evaluation of our experiments. An alternative to using ε is to use the *distribution* vector of error. Consider the two matrices **X** and $\hat{\mathbf{X}}$ defining an error matrix

$$\mathbf{E} = |\mathbf{X} - \hat{\mathbf{X}}| = |\mathbf{X} - \mathbf{B}_2\mathbf{C}\mathbf{B}_1^T| \tag{18}$$

and let **e** be the vector containing the frequency distribution of values of **E**. Plotting this vector gives the investigator a much better idea about the fit than the scalar ε.

The reconstructed and original representations have also been compared visually, which we believe is satisfactory for all practical purposes in this situation. Since it is the chemist who will evaluate the results anyway, the use of error measures has no practical meaning if the chemist decides that the reconstruction is not satisfactory.

3.4. CPU performance evaluation

What computational benefits are achieved by performing e.g. PCA on the compressed instead of the original representation? One way of measuring this is to count the number of floating point operations (FLOPS) needed for the analysis of **X** and **C**. Here the FLOPS counting encountered in MATLAB was used as the basis for our results. Since the method of data analysis used is PCA, a formula was found which accounts for the number of FLOPS required using the NIPALS algorithm (not including the FLOPS required for calculating and subtracting the mean vector):

$$F_{\mathrm{NIPALS}} = A\,[d_a(4nm + 2n + 6m) + 3nm] \tag{19}$$

where n and m are the numbers of rows and columns respectively and d_a is the number of iterations required for each component a. d_a is not a predefined number and the iteration stops when there is little difference between the present estimate of the latent variables and the previous one. A is the number of components. From the above formula it is easy to see that the number of FLOPS for the power method is proportional to nmd_{m}, where d_{m} is the mean number of iterations required for each component. By introducing the variables $\alpha = n/c_1$ and

$\beta = m/c_2$ and assuming d_a is the same number of iterations for both representations, the ratio

$$Q = [d_a(4nm + 2n + 6m) + 3nm]\left[d_a\left(\frac{4nm}{\alpha\beta} + \frac{2n}{\alpha} + \frac{6m}{\beta}\right) + \frac{3nm}{\alpha\beta}\right]^{-1} \tag{20}$$

can be formulated which gives the number of times more FLOPS required for PCA of **X** compared with PCA of **C**.

What about the FLOPS required for the compression itself? This part will become important if e.g. PCA is to be used *once* on the compressed data set and no further analyses are to be performed. In such situations there will be values of α and β which are too small to be of any computational benefit. However, the situation is often that the experimenter wants to perform *several* analyses on the same data set. Under such circumstances the compression very quickly pays off in more rapid analyses. Our experience with the B-spline algorithm is that the use of *sparse matrix*[9] technology significantly increases the computational efficiency.

4. EXPERIMENTS

4.1. Details of the SQ

All experiments were performed using MATLAB 4.0a on an HP 9000/730 Unix machine. The SQ used Huffman coding[8] to obtain the measured number of bits per coefficient. The maximum number of source symbols possible for the Huffman routine was $2^8 = 256$. Experiments showed that this was not a hindrance for reliable reproduction of the spectra. In fact, it was found acceptable to use just 64 source symbols, i.e. $\log_2(64) = 6$ bits per symbol.

The construction of the source symbols is based on the method previously referred to as the *maximum entropy method.*[11,22] This method uses a vector $\mathbf{g}^T$ to describe the histogram distribution of the data set which contains $J = 256$ cells, i.e. a vector of 256 elements. The J-number is arbitrarily chosen. Another number could have been used. The steps for determining the source symbols are as follows.

1. $I = (1/N)\ \Sigma_{i=1}^{J}\ \mathbf{g}_i^T$, where N is the maximum number of *allowed* source symbols.
2. Starting at the first cell, consecutive intensities are added in $\mathbf{g}^T$ until the partial sum is equal to or exceeds I. The position of the *next* cell after the last cell added in the partial sum is included in a vector which defines the new cells. If the last cell added is 3, the number 4 is stored, i.e. the next cell member.

4.2. Data set

A Nicolet 800 spectrometer with a 680 DSP workstation and a diffuse reflectance unit with a high-temperature–high-pressure chamber from Spectra Tech were used for the spectral measurements. The sample was a 5 w/w% kerogen in KBr. The starting temperature was 200 °C and the system was then kept isothermally for 1 min to ensure that the pyrolysis started at a known temperature. During the heating, adsorbed water was also removed from the powdered sample. The pyrolysis was carried out using a heating rate of 5 °C min^{-1} between 200 and 400 °C. A nitrogen purge gas rate of 50 ml min^{-1} was employed to remove the gaseous products and to provide an inert atmosphere in the sample chamber. A full spectrum comprising 16 scans was collected, Fourier transformed and stored every 30 s. A medium-band MCT detector was used. After the termination of the pyrolysis the 80 sample spectra were first background corrected with spectra of pure KBr taken under the same conditions and then

Kubelka–Munk transformed. The spectral range was 4000–650 cm^{-1} and the optical resolution was set to 8 cm^{-1}. The data point resolution was 3·85 cm^{-1}. On the basis of these settings, a data set of size [80 × 869] was obtained.

5. RESULTS

The parameters for the compression experiment were set to $c_1 = 115$ (number of coefficients for the wave number direction), $c_2 = 14$ (number of coefficients for the temperature direction), $k_1 = 3$ (local degree of spline for the wave number direction) and $k_1 = 4$ (local degree of spline for the temperature direction). The consequence of selecting a high local degree and few knot points for the temperature direction is that an increased smoothing is achieved. Figure 4 illustrates the effect of smoothing. The small-scale variations in the temperature variables observed in the original representation are due to noise. The two-dimensional IR experiment was set to a continuous and smooth change in temperature. The estimated B-spline coefficient matrix **C** from equation (7) was inserted into the SQ routine and the returned $\mathbf{C}_s$-matrix was the result from the quantization. Figures 5 and 6 show two selected rows in the coefficient matrix. Figures 7 and 8 show the two same rows for the variables [70, 71, ..., 90].* The SQ approximation is satisfactory. From the $\mathbf{C}_s$-matrix the matrix $\hat{\mathbf{X}}$ was constructed according to equation (10). Figure 9 is a graphical representation of the distribution of the error vector $|\mathbf{X} - \hat{\mathbf{X}}|$. The error value ε derived from equation (17) is 0·0459 ($\mathbf{W} = \mathbf{I}$, i.e. no weighting). Figures 10 and 11 shows two representative IR profiles for the original and reconstructed representations. There are no major visual discrepancies between these two representations,

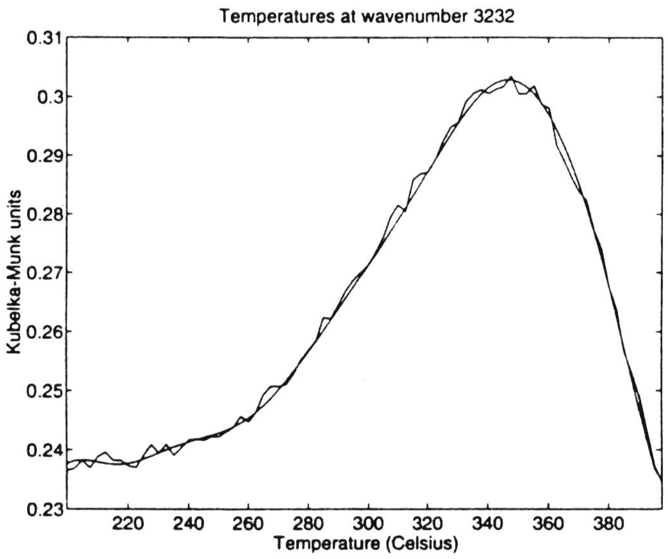


Figure 4. This shows the smoothing effect of using few knots and high local degree of B-spline for the temperature direction. Here the wave number 3232 in the original and reconstructed (smooth line) matrices is compared

* It is not correct to use the actual wave numbers for the coefficient matrix, so the word 'variables' corresponds to the wave number direction on the original representation.

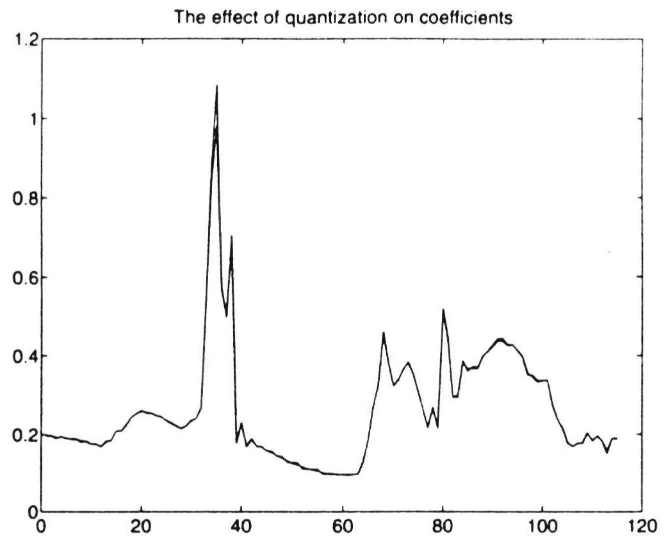


Figure 5. The effect of quantization on the coefficient matrix is shown by plotting a row (no. 2) before and after SQ with 64 source symbols in the source alphabet

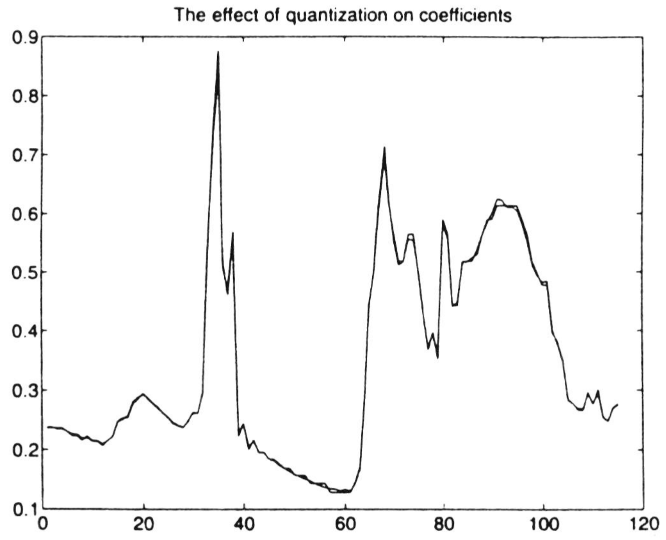


Figure 6. The effect of quantization on the coefficient matrix is shown by plotting a row (no. 6) before and after SQ with 64 source symbols in the source alphabet

which is very encouraging. Figure 12 shows some additional selected IR profiles plotted versus each other in the original and reconstructed representations.

For this experiment the total number of bits needed to represent the coefficient matrix $\mathbf{C}_s$ using Huffman coding was 9600. Since the number of data elements in the original data matrix was $80 \times 869 = 69\,520$, the number of bits required per data element was $9600/69\,520 = 0{\cdot}1381 \approx 0{\cdot}14$. However, this does not include the number of bits required to

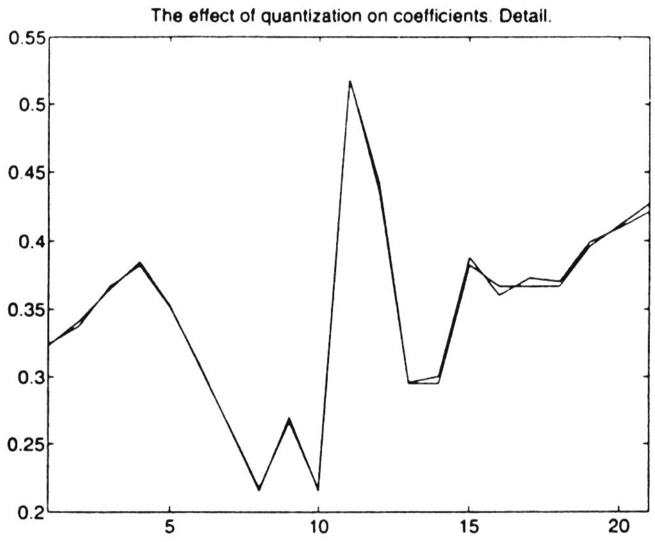


Figure 7. The effect of quantization on the coefficient matrix is shown by plotting a row (no. 2) before and after SQ with 64 symbols in the source alphabet (region [70, 71, ..., 90] is expanded)

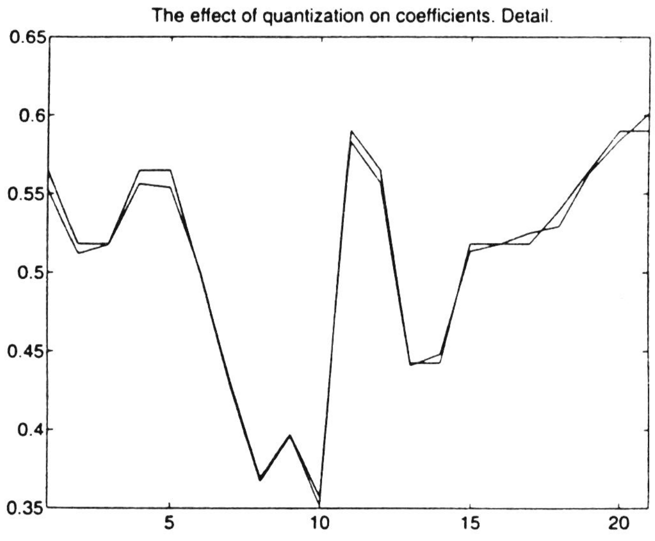


Figure 8. The effect of quantization on the coefficient matrix is shown by plotting a row (no. 6) before and after SQ with 64 source symbols in the source alphabet (region [70, 71, ..., 90] is expanded)

store the knot vectors defining the two basis set matrices $\mathbf{B}_1$ and $\mathbf{B}_2$. Here a high bit representation can be afforded. The original representation contains 869 variables, so giving each knot a 10 bit representation ($2^{10} = 1024$ possible symbols) should be enough. For $115 + k_1 + 1 = 119$ knots we need $10 \times 119 = 1190$ bits. For the temperature region there are 80 variables originally and so 7 bits per knot ($2^7 = 128$) should be enough for representation of the knots. The number of bits required to store the knot vector (containing $14 + k_2 + 1 = 19$

knot elements) along the temperature is thus $19 \times 7 = 133$ bits. A total of 1323 bits to store the knot vectors is needed. If this figure is included, we get approximately $(9600 + 1323)/69\,520 = 0{\cdot}1571 \approx 0{\cdot}16$ bits per data element. Analogously to equation (16), the redundancy for this set of parameters with respect to the original bit representation from the instrument can be computed as $r = 16 - 0{\cdot}16 = 15{\cdot}84$ bits per data element. For comparison, the number of bits per data element when using lossless compression on the original data set

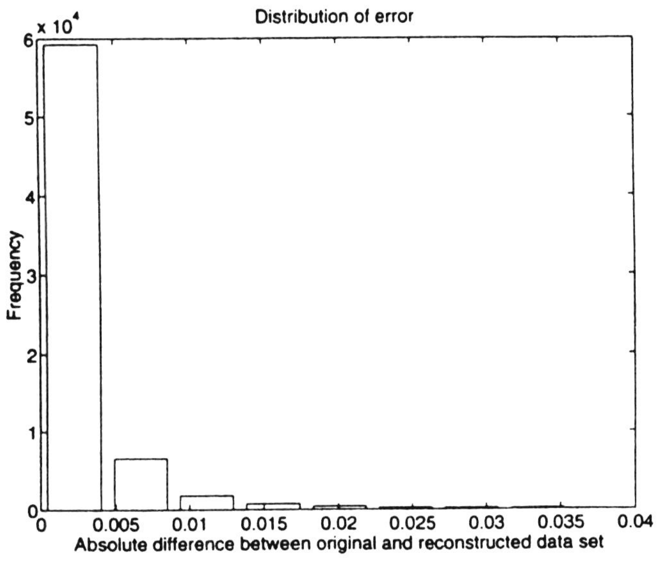


Figure 9. Bar plot of the distribution of error between reconstructed and original representations

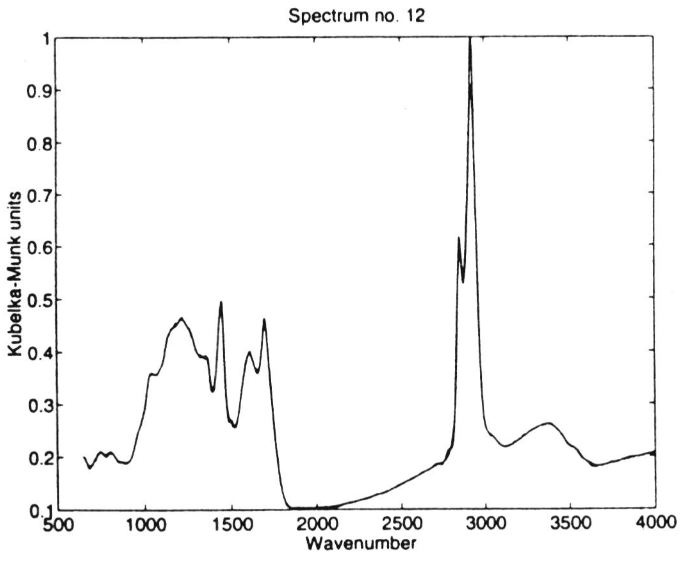


Figure 10. Comparison between temperature variable no. 12 for the original and reconstructed representations ($m = 115$, $n = 14$, $k_1 = 3$ and $k_2 = 4$)

is 15·4, which is approximately 96 times larger than for the B-spline method with SQ. The *calculated* entropy of the original data sequence was about 12·2 bits per data element.

The next step was to compare the PCA of the original uncompressed matrix with the PCA of the compressed representation. The PCA of $\mathbf{X}$ and $\mathbf{C}_s$ resulted in two significant components (see Figure 13). The PCA of the original representation required approximately 40 times more FLOPS than the PCA of the compressed representation. This was also

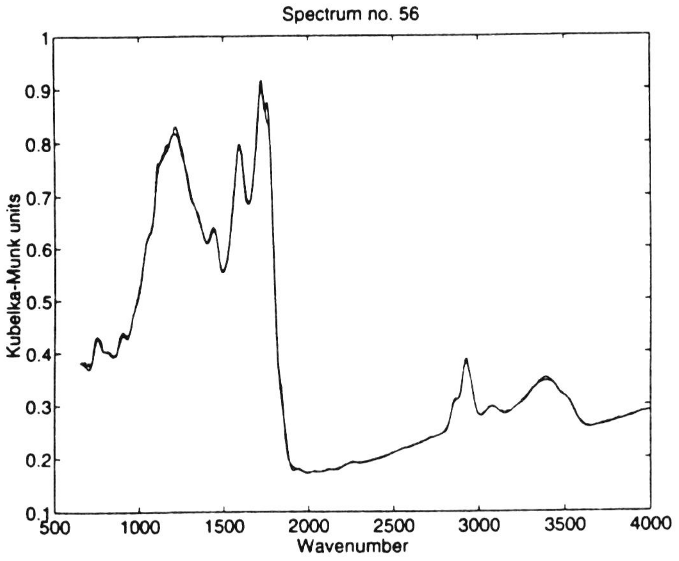


Figure 11. Comparison between temperature variable no. 56 for the original and reconstructed representations (m = 115, n = 14, k_1 = 3 and k_2 = 4)

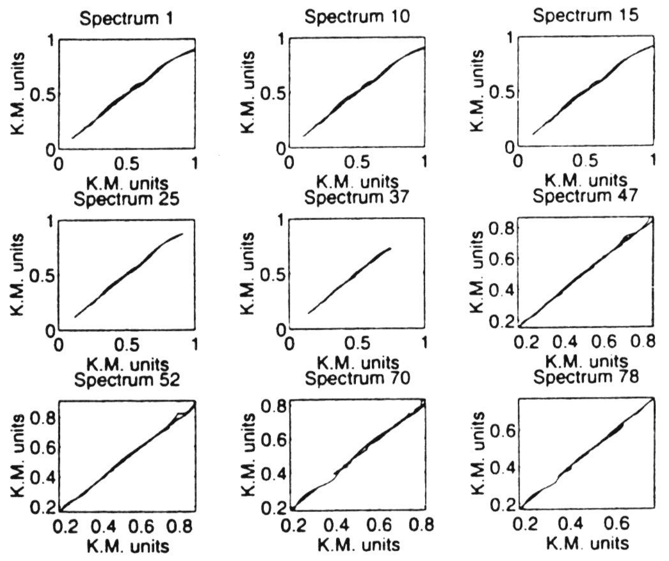


Figure 12. Nine different spectral profiles were selected. Each original spectrum was compared with the corresponding reconstructed spectrum by plotting the two vectors versus each other. The parameters for the compression are m = 115, n = 14, k_1 = 3 and k_2 = 4. k_2 = 4. K.M. stands for Kubelka–Munck

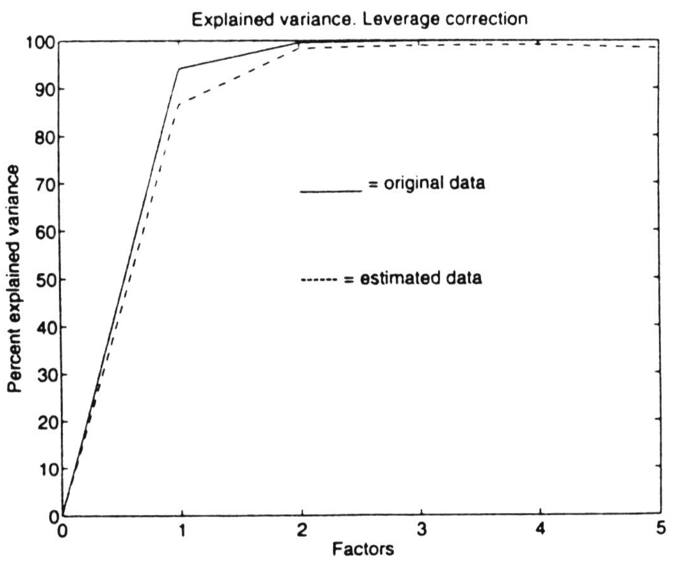


Figure 13. Ten components were tested for and the results for the compressed and original representations (the first five factors only are shown in the figure)

confirmed using equation (20). In the PCA model for the original representation the first two components accounted for 99·5% of the variance. The corresponding value for the compressed representation was 98·4%. Thus only the first two components for the different representations were analysed. In Figure 14 the scores of the original (left) and compressed (right) representations are shown. The model of $\mathbf{C}_s$ contains fewer points but has a similar shape to the left part of the figure. Corresponding similarities can also be observed for the

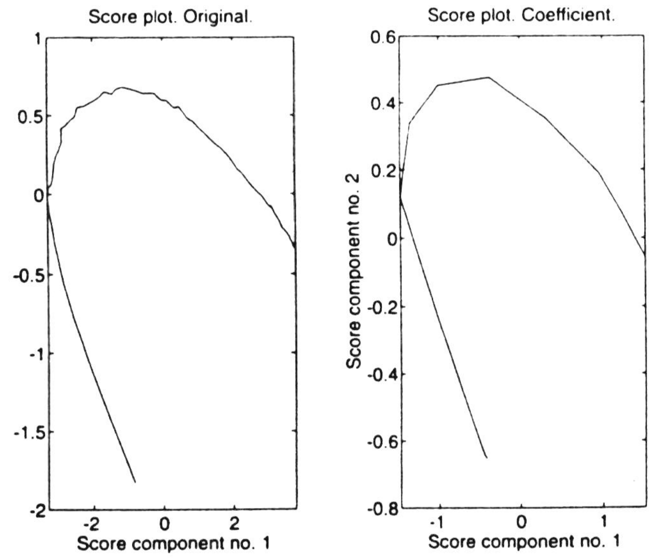


Figure 14. Score 1 versus 2 for the original and compressed representations. The compressed representation has similar main features to the original score plot

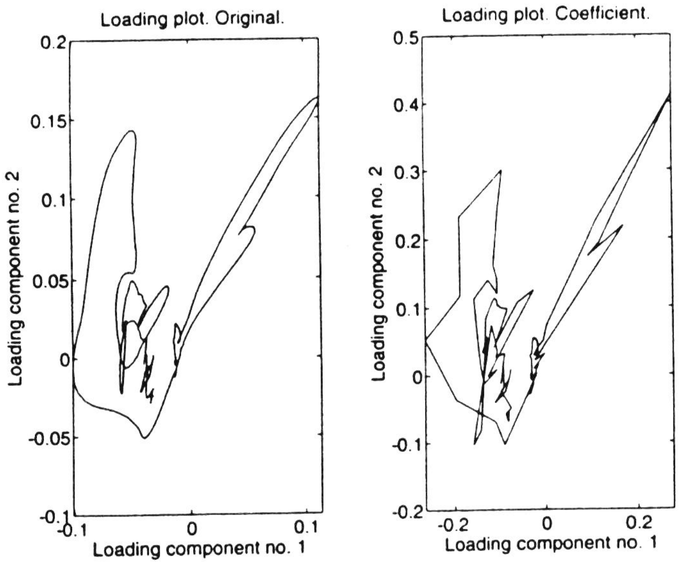


Figure 15. Loading 1 versus 2 for the original and compressed representations. The compressed representation has similar main features to the original loading plot

loading vectors of the two-component model (see Figure 15). The same tendencies are thus observed in the compressed PCA model as in the original PCA model. This further emphasizes that our compressed representations are *comparable*. The next step was to use the knowledge of the basis functions to compute estimates of the original $\mathbf{T}$ and $\mathbf{P}^{\mathrm{T}}$. The corresponding model matrices for the compressed representation were $\mathbf{T}_{\mathrm{c}}$ and $\mathbf{P}_{\mathrm{c}}^{\mathrm{T}}$. $\hat{\mathbf{T}}$ and $\hat{\mathbf{P}}^{\mathrm{T}}$ were computed according to equations (3) and (4) respectively. Figure 16 shows the first two score vectors for the $\mathbf{T}$- and $\hat{\mathbf{T}}$-matrices. Figure 17 shows the first two loading vectors for the $\mathbf{P}^{\mathrm{T}}$- and $\hat{\mathbf{P}}^{\mathrm{T}}$-

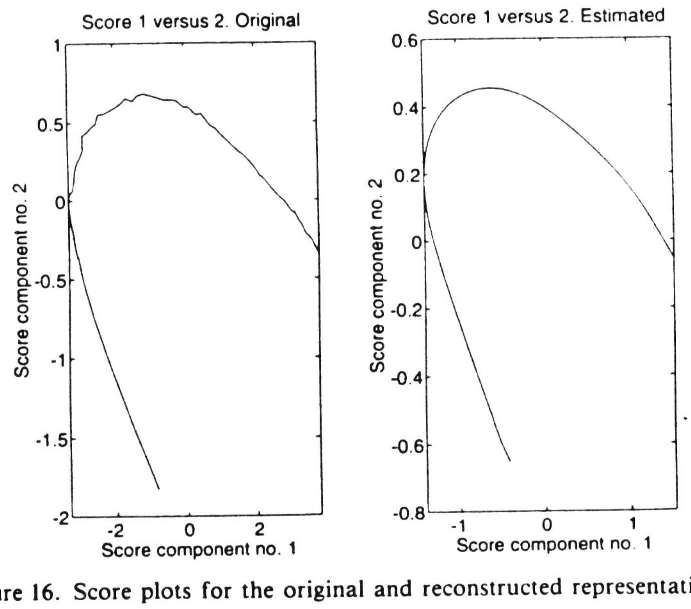


Figure 16. Score plots for the original and reconstructed representations

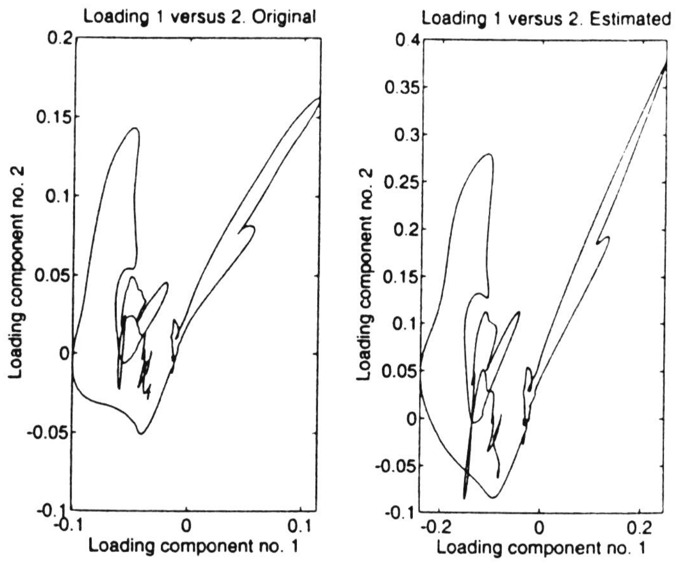


Figure 17. Loading plots for the original and reconstructed representations

matrices. Equations (3) and (4) are not correct for non-orthonormal basis matrices but can be used as good approximations, since the B-spline basis set is diagonally dominant, i.e. the row/column sum of any row/column of the off-diagonal element is smaller than or equal to the absolute value of the corresponding diagonal element.[10] Again it must be stressed that a simple orthonormalization of $\hat{\mathbf{P}}^T$ and $\hat{\mathbf{T}}$ will *not* give the correct results. The fact that $\hat{\mathbf{P}}^T\hat{\mathbf{P}} \neq \mathbf{I}$ and $\hat{\mathbf{T}}^T\hat{\mathbf{T}}$ is not diagonal will for many applications not be a serious problem. In some instances it will, however, and we have therefore rewritten the PCA algorithm to compensate for this effect. These results will be published in a future paper.

6. DISCUSSION

Large databases containing IR spectra with high resolution exist and a reduction of the size of such databases is needed. The type of compression discussed here allows comparison between compressed representations of different spectra. This would not have been easily accomplished if e.g. a separate run-length coding or Huffman coding was applied to each spectrum. The approach used here assumes that the compression codes *must* be comparable. In large databases the problem of finding the matching spectrum for an unknown has a higher cost for uncompressed data. Since the compression codes for each spectrum are comparable, classification based on the compressed variables alone is theoretically possible. The comparability of the compression codes has a large benefit also from a computational point of view. The argument that computers are getting more powerful is in practice not a realistic excuse for not doing compression. Even for medium sized workstations such as Sun, HP and DEC the workload can be of such size that the computations may take hours instead of minutes and the files require hundreds of megabytes instead of tens of megabytes.

For higher-order numerical methods such as third-order principal component analysis the data size problem will easily get into the supercomputer realm if some type of compression is not used.

ACKNOWLEDGEMENTS

The authors wish to thank Professor John Håkon Husøy at Rogaland University Center for valuable discussions and help. B. A. and E. N. thank The Royal Norwegian Council for Scientific and Industrial Research (NTNF) for financial support.

APPENDIX I: THE B-SPLINE BASIS

The coefficients may be interpreted as the importance of a particular basis function. The basis functions are defined by the recursive algorithm[23]

$$B_{j,k}(x)=\frac{x-h_j}{h_{j+k}-h_j}B_{j,k-1}(x)+\frac{h_{j+k+1}-x}{h_{j+k+1}-h_{j+1}}B_{j+1,k-1}(x),\quad k\geqslant 1 \tag{21}$$

$j=0, \pm 1, \pm 2, \ldots,\ k=1,2,3,\ldots,$ where $\mathbf{h}$ is the knot vector, which is a non-decreasing sequence of numbers*

$$h_1\leqslant h_2\leqslant\cdots\leqslant h_q \tag{22}$$

and k governs the smoothness of the basis functions. If the data at hand are very smooth, fewer coefficients are needed to represent them.

APPENDIX II: QUANTIZATION

An N-point one-dimensional quantizer Q (a scalar quantizer (SQ)) can be formulated as the mapping

$$Q\colon \mathscr{R}\mapsto\mathscr{K} \tag{23}$$

where $\mathscr{R}$ is the real line and $\mathscr{K}$ is the source alphabet

$$\mathscr{K}\equiv\{y_1,y_2,\ldots,y_N\} \tag{24}$$

It is assumed that the output levels y_i are real numbers such that

$$y_1<y_2<\cdots<y_N \tag{25}$$

Associated with every element in the source alphabet is a partition of the real line $\mathscr{R}$ into N cells. The ith cell is defined as

$$R_i=\{x\in\mathscr{R}\colon Q(x)=y_i\} \tag{26}$$

Often the cells are defined such that

$$\cup R_i=\mathscr{R} \tag{27}$$

but this is not used in this paper. The beginning and end cells are defined from the minimum and maximum values observed in the data matrix. In addition, the cells are divided such that there is no overlap:

$$R_i\cap R_j=\emptyset,\quad i\neq j \tag{28}$$

Every quantizer can be regarded as a combination of an *encoding* ($\mathscr{E}$) and *decoding* ($\mathscr{D}$)

*The notation t_i is normally used in spline theory to designate the knot sequence. We have chosen not to use this letter since it is usually associated with score values.

operation/mapping. The encoder is the mapping

$$\mathscr{E}: \mathscr{R} \mapsto \mathscr{I} \tag{29}$$

where $\mathscr{I} = \{1, 2, 3, \ldots, N\}$ and the decoder is the mapping

$$\mathscr{D}: \mathscr{I} \mapsto \mathscr{K} \tag{30}$$

If $Q(x) = y_i$, then $\mathscr{E}(x) = i$ and $\mathscr{D}(i) = y_i$, which can also be written as

$$Q(x) = \mathscr{D}(\mathscr{E}(x)) \tag{31}$$

The encoder can be described in terms of a *selector function* $S_i(x)$ which is associated with every cell in Q. Given a number x, the selector function for the partition cell i determines whether x belongs to that cell or not:

$$S_i(x) = \begin{cases} 1 & \text{if } x \in R_i \\ 0 & \text{otherwise} \end{cases} \tag{32}$$

Using this, the quantizer can be written as

$$Q(x) = \sum_{i=1}^{N} y_i S_i(x) \tag{33}$$

where

$$\sum_{i=1}^{N} S_i(x) = 1 \tag{34}$$

To separate explicitly the encoder and decoder operations of a quantizer, an *address generator* must be specified as

$$A: \mathscr{F} \mapsto \mathscr{I} \tag{35}$$

and the inverse as

$$A^{-1}: \mathscr{I} \mapsto \mathscr{F} \tag{36}$$

where $\mathscr{F}$ denotes the set of N binary vectors associated with every source symbol in $\mathscr{K}$ for a specific input x. Each vector $\mathbf{f} \in \mathscr{F}$ consists of just zeros and one element of unity, i.e. $f_i \in \{0, 1\}$. Each element of the vector originates from the selector function:

$$\mathbf{f} = \{S_1(x), S_2(x), \ldots, S_N(x)\} \tag{37}$$

Thus a vector equation can be written from equation (33) as

$$Q(x) = \mathbf{f}^{\mathrm{T}}\mathbf{y} \tag{38}$$

Since $A(\mathbf{f}) = i$, this can be used to formulate the encoder operation as

$$\mathscr{E}(x) = A(\mathbf{f}) = A(\{S_1(x), S_2(x), \ldots, S_N(x)\}) \tag{39}$$

and the decoder is written

$$\mathscr{D}(i) = \sum_{k=1}^{N} y_k A^{-1}(i)_k$$

An SQ will be used on the resulting B-spline coefficients. There are several ways to construct the source alphabet and how the encoding/decoding should operate. The selector function used here is based on finding the smallest element in the vector $\mathbf{v}_i = |x - y_i|$. The position j which satisfies min(**v**) is the return value from the encoder function $\mathscr{E}(x)$. The next problem is to select the best source alphabet. In this paper it is based on the probability density function over the measured range of the B-spline coefficients. How this was done in the subsequent experiments is described in more detail in Section 4. The simplest selection of partition cells is the *uniform* quantizer where each cell has equal size. Uniform quantizers have *not* been adopted in this paper.

REFERENCES

1. I. Pelczer and S. Szalma, *Chem. Rev.* **91**, 1507–1524 (1991).
2. R. A. Nyquist, M. A. Leugers, M. L. McKelvy, R. R. Papenfuss, C. L. Putzig and L. Yurga, *Anal. Chem.* **62**, 223R–255R (1990).
3. O. M. Kvalheim and Y.-Z. Liang, *Anal. Chem.* **64**, 937–946 (1992).
4. Y.-Z. Liang, O. M. Kvalheim, H. R. Keller, D. L. Massart, P. Kiechle, and F. Erni, *Anal. Chem.* **64**, 947–953 (1992).
5. C. de Boor, *A Practical Guide to Splines*, Springer, New York (1978).
6. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design, a Practical Guide*, 2nd edn, Academic, New York, (1990).
7. C. de Boor, *Spline Toolbox for Use with MATLAB. User's Gude*, MathWorks Inc., South Natick, MA (1990).
8. J. A. Storer, *Data Compression. Methods and Theory*, Computer Science Press, Rockville, MD (1988).
9. S. Pissanetsky, *Sparse Matrix Technology*, Academic, London (1984).
10. B. Alsberg, *J. Chemometrics*, **7**, 177–193 (1993).
11. B. Alsberg and O. M. Kvalheim, *J. Chemometrics*, **7**, 61–73 (1993).
12. R. C. Gonzales and P. Wintz, *Digital Image Processing*, 2nd edn, Addison–Wesley, Reading, MA (1987).
13. A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, Englewood Cliffs, NJ (1989).
14. W.-H. Chen and W. Pratt, *IEEE Trans. Communications*, **COM-32**, 225–232 (1984).
15. Y. Meyer, *Wavelets. Algorithms and Applications*, SIAM, Philadelphia, PA (1993).
16. H. Martens and T. Naes, *Multivariate Calibration*, Wiley, New York (1989).
17. I. Cowe and J. W. McNicol, *Appl. Spectrosc.* **39**, 257–266 (1985).
18. S. Wold, K. Esbensen and P. Geladi, *Chemometrics Intell. Lab. Syst.* **2**, 37–52 (1987).
19. H. Wold, in *Research Papers in Statistics*, ed. by F. David, p. 411, Wiley, New York (1966).
20. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer, Norwell, MA (1992).
21. C. E. Shannon, *Bell Syst. Tech. J.* **379**, 623 (1948).
22. T. V. Karstang and R. J. Eastgate, *Chemometics Intell. Lab. Syst.* **2**, 209–219 (1987).
23. W. Cheney and D. Kincaid, *Numerical Mathematics and Computing*, Brooks/Cole, Monterey, California (1980).

Paper III

ELSEVIER

Chemometrics and Intelligent Laboratory Systems 23 (1994) 29–38

Chemometrics and intelligent laboratory systems

Compression of three-mode data arrays by B-splines prior to three-mode principal component analysis

Bjørn K. Alsberg *, Olav M. Kvalheim

Department of Chemistry, University of Bergen, Allegt. 41, N-5007 Bergen, Norway

(Received 6 July 1993; accepted 14 January 1994)

Abstract

Three-mode PCA is very computer demanding. It requires a large amount of storage space and many floating point operations (FLOPS). By using three-mode B-spline compression of three-mode data arrays, the original data array can be replaced by a smaller coefficient array. Three-mode principal component analysis (PCA) is then performed on the much smaller coefficient array instead of on the original array. For the compression approach to be efficient the three-mode data array is assumed to be well approximated by smooth functions. The smoothness affects the dimensions of the coefficient array. It is always possible to approximate the data to any precision but the reward in reduced computation time and storage is lost when the dimensions of the coefficient array approach the dimensions of the original array.

1. Introduction

N-mode arrays represent extensions of two-mode arrays, i.e., matrices. There are several other names for these objects: tensors, *N*-way arrays, *N*-arrays, and multilinear forms. We will here refer to such data objects as *N-mode arrays* or *N-arrays*.

Several analytical instruments produce data sets of the *N*-array type. Such arrays can be very useful for obtaining, e.g., information about constituents in solutions [1,2] or atom assignment of crosspeaks in multidimensional NMR [3]. Unfortunately, the *N*-arrays present serious storage and computational problems. The number of data elements increases rapidly and the need for some reduction and improvement in the data handling procedure is necessary. In previous papers [4–6] we have suggested that compression by B-splines or by means of another suitable basis may be an efficient way of partially solving the increased data size problem.

There are two main types of compressions: lossless and lossy [7]. Lossless compression restores the data perfectly but does not attain large compression ratios as in lossy compression. The B-spline method used here is a lossy compres-

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

sion. What is an acceptable error in the reconstruction compared to the original array must be decided by the investigator and must be considered to be problem dependent.

In this article we focus on three-mode principal component analysis (3MPCA) which is computer demanding [8] for arrays of size larger than $[50 \times 50 \times 50]$. In such cases powerful workstations are necessary. If data from an instrument should be analyzed directly, arrays of, e.g., dimensions $[1000 \times 500 \times 500]$ are realistic. Examples of instruments giving rise to such data sets are, e.g., three-dimensional NMR spectra or two-dimensional IR versus time/temperature. Inserting such an array without any compression or pretreatment into a standard 3MPCA program would require a very powerful computer. Fortunately, the spectra from analytical instruments often contain some smoothness which enables the use of compression methodology. The compression part of course, must not be too computer demanding in itself.

Before reading the next section the reader is encouraged to read the appendix which explains the different notations used in this article.

2. Three-mode PCA

In singular value decomposition (SVD) for 2-arrays an $\mathbf{X}$ matrix can be decomposed as:

$$\mathbf{X} = \mathbf{U}\mathbf{S}^{1/2}\mathbf{V}^{\mathrm{T}} \tag{1}$$

where $\mathbf{U}$ and $\mathbf{V}$ are column-wise orthonormal matrices and $\mathbf{S}^{1/2}$ is a diagonal matrix containing the square root of eigenvalues of the two covariance matrices $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ and $\mathbf{X}^{\mathrm{T}}\mathbf{X}$. The two associated eigen-equations are:

$$\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{U} = \mathbf{U}\mathbf{S} \tag{2}$$

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{V} = \mathbf{V}\mathbf{S} \tag{3}$$

In SVD for 3-arrays an $\underline{\mathbf{X}}$ array can be decomposed as:

$$\mathbf{X} = \mathbf{G}\mathbf{D}(\mathbf{H}^{\mathrm{T}} \otimes \mathbf{E}^{\mathrm{T}}) \tag{4}$$

where $\mathbf{X}$ and $\mathbf{D}$ are unfolded representations of the 3-array (see Appendix) and $\mathbf{G}$, $\mathbf{H}$ and $\mathbf{E}$ are column-wise orthonormal loading matrices for each mode. Eq. 4 is the Tucker3 model [8,9]. It should be stressed that the 3-arrays (represented as matrices) in Eq. 4 must be unfolded in the same way. $\mathbf{D}$ is the core matrix (or core array). Eq. 4 written in explicit summation is:

$$x_{ijk} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{ip} h_{jq} e_{kr} d_{pqr} \tag{5}$$

There are different methods for solving the Tucker3 model. A method similar to the approach used for SVD of 2-arrays (see Eqs. 2 and 3) is the Tucker Method I [8,9] which obtains estimates for the loading matrices $\mathbf{E}$, $\mathbf{H}$ and $\mathbf{G}$ by extracting all eigenvectors corresponding to nonzero roots of the three covariance matrices with elements:

$$l_{ii'} = \sum_{j=1}^{J} \sum_{k=1}^{K} x_{ijk} x_{i'jk} \tag{6}$$

$$m_{jj'} = \sum_{i=1}^{I} \sum_{k=1}^{K} x_{ijk} x_{ij'k} \tag{7}$$

$$n_{kk'} = \sum_{i=1}^{I} \sum_{j=1}^{J} x_{ijk} x_{ijk'} \tag{8}$$

Using the estimated three loading matrices, Eq. 5 shows the core array $\underline{\mathbf{D}}$ can be expressed as:

$$d_{pqr} = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} g_{ip} h_{jq} e_{kr} x_{ijk} \tag{9}$$

This equation is also shown in Fig. 1 using the diagram notation.

In practice an investigator is often interested in only the first largest eigenvectors of $\mathbf{E}$, $\mathbf{H}$ and $\mathbf{G}$. Using the Tucker Method I, however, the estimators for d_{pqr} will no longer be least-squares ones. In order to achieve least squares estimates

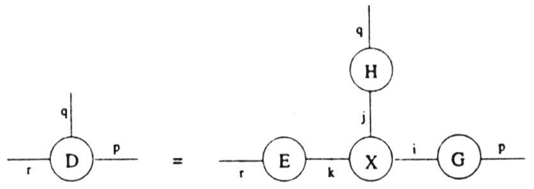


Fig. 1. A diagram equation which shows how to obtain the core array given the loading matrices G, H, E and the original data array X.

an alternating least squares (ALS) algorithm [8] can be used. The ALS approach is a common method for solving the Tucker3 model. The algorithm is initiated by first computing estimates of the loading matrices $\mathbf{G}_0$, $\mathbf{E}_0$ and $\mathbf{H}_0$ by the Tucker Method I. $\mathbf{G}_0$, $\mathbf{E}_0$ and $\mathbf{H}_0$ are subsequently inserted into the following iteration steps:

```
For $i = 1$ to iterations
  begin
    $\mathbf{A} = \mathbf{X}_1(\mathbf{H}_{i-1} \otimes \mathbf{E}_{i-1})$      (10)
    $\mathbf{G}_i = eig(\mathbf{AA}^T)$      (11)
    $\mathbf{A} = \mathbf{X}_2(\mathbf{E}_{i-1} \otimes \mathbf{G}_i)$      (12)
    $\mathbf{H}_i = eig(\mathbf{AA}^T)$      (13)
    $\mathbf{A} = \mathbf{X}_3(\mathbf{G}_i \otimes \mathbf{H}_i)$      (14)
    $\mathbf{E}_i = eig(\mathbf{AA}^T)$      (15)
  end
```

Matrix **A** temporarily stores the results and is overwritten when new values are assigned to its matrix elements. The function *eig* returns the first largest eigenvectors of $\mathbf{AA}^T$ (this is a reduced decomposition). This algorithm is also formulated in the diagram notation in Fig. 2. $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$ are unfolded representations of the original 3-array with dimensions $Dim(\mathbf{X}_1) = [N \times (MK)]$, $Dim(\mathbf{X}_2) = [M \times (NK)]$, $Dim(\mathbf{X}_3) = [K \times (NM)]$. The number of iterations can be determined by minimizing the error of fit of the model to the observed data. This approach, however, is not used in the present paper. The reason for this is to ensure total control with respect to the number of iterations when investigating the FLOPS usage of the ALS algorithm applied to different, but comparable, data representations.

3. FLOPS estimations

The 3MPCA program is written in MATLAB [10] which is very powerful for matrix computations. In order to obtain a measure of the efficiency of the ALS algorithm on uncompressed and compressed representations the *flops* command in MATLAB was employed. The FLOPS equation and observed performance of the ALS algorithm presented in this paper are based on the results from this command. In Table 1 a few examples are given where the FLOPS equations for simple matrix expressions are presented. For each matrix expression in an algorithm the FLOPS formula is found and the total FLOPS consumption is obtained by summing over the different contributions. Only the most important parts of an algorithm are investigated. All WHILE loops

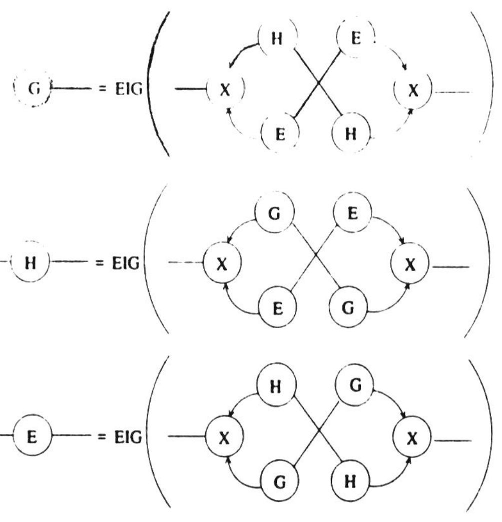


Fig. 2. Illustration of the ALS algorithm using diagram formulation.

Table 1
This table shows the number of FLOPS required for some example matrix operations

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

are changed to FOR loops to determine the FLOPS consumption for such steps.

The detailed description of how a FLOPS formula is constructed will not be presented.

Using the same approach to the ALS algorithm the following equation is obtained:

$$F_{\text{tot}} = I\left[2A(3NMK + NMA + NKA + MKA + N^2A + M^2A + K^2A) + N^3 + M^3 + K^3\right] \qquad (16)$$

where the last entries $N^3 + M^3 + K^3$ stem from the eigenvalue decomposition which is approximately a third degree increase in FLOPS. Here it is assumed that the core array is symmetric, i.e., its dimensions are $[A \times A \times A]$. This will also be the case in the examples presented below. I is the number of iterations. Note that this formula slightly underestimates the number of FLOPS required but is close to the true FLOPS count.

4. B-splines

B-splines [11,12] can be used to fit almost any type of function. Significant compression, however, is obtained if the data at hand have some smoothness. A one-dimensional B-spline is a linear combination of basis functions b_j:

$$f(x) = \sum_{j=1}^{u-\alpha-1} c_j b_j(x) \qquad (17)$$

where $f(x)$ is the function to be approximated, c_j the B-spline coefficients, u is the number of knot points and α the local polynomial degree. The basis functions $b_j(x)$ (which for discrete representations are located in a matrix called **B**) are defined by the knot vector $\boldsymbol{h}$. Knots are points located along the independent axis which define the shape and location of the B-spline basis functions. The basis can be constructed by a recursive formula [12,13] using the information in the vector $\boldsymbol{h}$. The knot vector $\boldsymbol{h}$ is therefore stored instead of the much larger B-spline basis matrix **B**.

For compression of N-mode arrays N B-spline basis matrices (or N $\boldsymbol{h}$ vectors) are needed.

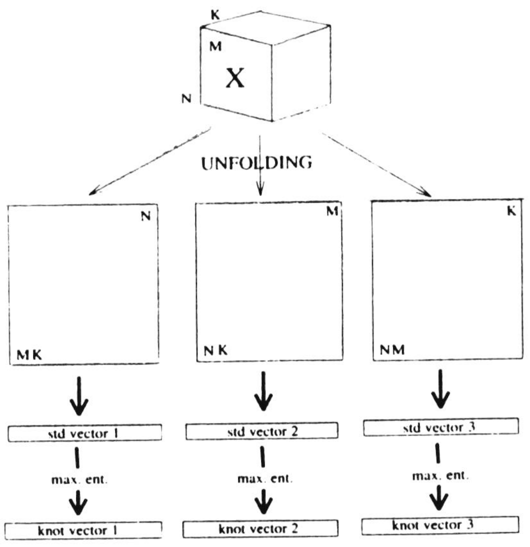


Fig. 3. Illustration of the steps for finding the knot vectors for the three different modes. For each unfolding we find a representative vector (RV) which is said to represent the current mode. In this case the RV is the standard deviation vector (indicated as 'std. vector' in the figure). The maximum entropy method as described in ref. [4] is applied to the standard deviation vector and a knot vector is obtained.

The steps for obtaining the knot vectors for each mode are:

(i) The original data array is unfolded to a matrix. One mode of this matrix corresponds to the current mode of the original data array for which the knot vector is to be found.

(ii) A representative vector (RV) is obtained for the current mode. An RV can be, e.g., the mean or the standard deviation vector of the unfolded matrix along the current mode. In this paper the standard deviation vector is used as the RV.

(iii) The so-called maximum entropy method [4,14] is applied to the RV and the interval vector from this procedure is used as the knot vector for the current mode.

See Fig. 3 for a graphical illustration of the method.

Given three knot vectors named $\boldsymbol{h}_n$, $\boldsymbol{h}_m$, and $\boldsymbol{h}_k$ three corresponding basis matrices $\mathbf{B}_n$, $\mathbf{B}_m$ and $\mathbf{B}_k$ can be generated (by using a recursive formula [12,13]). The three different modes were

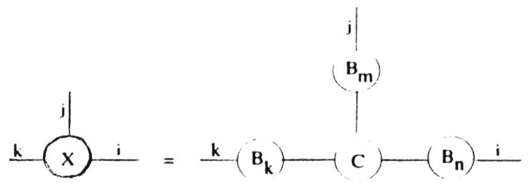


Fig. 4. Diagram of the three mode B-spline model. Illustration of how the three knot vectors for three-mode B-spline compression divide the data array into segments.

thus labeled n, m and k. If the three-mode data matrix $\underline{\mathbf{X}}$ has dimensions $[N \times M \times K]$ then the dimensions of the basis matrices are $[N \times n_c]$ for $\mathbf{B}_n$, $[M \times m_c]$ for $\mathbf{B}_m$ and $[K \times k_c]$ for $\mathbf{B}_k$. Let $\{\alpha_n,\alpha_m,\alpha_k\}$ be the local degree of polynomial for the spline and $Dim(\boldsymbol{h}_i)$ be the number of elements in a knot vector, then we have that $n_c = Dim(\boldsymbol{h}_n) - \alpha_n - 1$, $m_c = Dim(\boldsymbol{h}_m) - \alpha_m - 1$ and $k_c = Dim(\boldsymbol{h}_k) - \alpha_k - 1$. It is desirable to have small values of n_c, m_c and k_c without too much error. The coefficient array $\underline{\mathbf{C}}$ has thus dimensions $[n_c \times m_c \times k_c]$. The model assumption for the three-mode B-spline model is shown in Fig. 4 which has the same index topology as the Tucker3 model. The equation for the generation of the core array $\underline{\mathbf{C}}$ is shown in Fig. 5.

4.1. Compression of a subset of modes

As mentioned earlier an efficient compression ratio is achieved if the data array can be well represented by smooth basis functions. It is a problem when some of the modes are not smooth. B-spline compression may still be of benefit if applied to the smooth modes only. In spectral problems it is realistic to encounter, e.g., three-mode arrays where only two of the modes are smooth. The third mode may also be much smaller than the smooth modes.

In refs. [15,16] Kiers et al. presented two efficient algorithms for the PARAFAC and TUCKALS3 algorithms for cases when the size of one of the modes is much larger than the other two. These algorithms do not cover the case discussed in this article when all modes are large, but it is possible to imagine that two large modes are compressed and the third is handled by the algorithms developed by Kiers et al.

5. Experiments

5.1. Data set

The data used for this example are the three-dimensional electron density distribution of the inhibitory neurotransmitter γ-aminobutyric acid (GABA). This distribution was calculated for the molecule using the AM1 quantum mechanical model [17]. The electron density surface was computed on a grid of size [71 × 38 × 44].

5.2. Results

The $\boldsymbol{h}_n$ knot vector was generated by using the maximum-entropy method [4,14] on the standard deviation vector obtained from the unfolded matrix of size $[(38 \cdot 44) \times 71]$. Correspondingly the $\boldsymbol{h}_m$ vector was generated from the unfolded matrix of size $[(71 \cdot 44) \times 38]$ and $\boldsymbol{h}_k$ from the unfolded matrix of size $[(71 \cdot 38) \times 44]$. The MATLAB B-spline toolbox [18] was used to produce the basis matrices. The sizes of the knot vectors after assuming third degree polynomial B-splines for the n and k mode and second degree polyno-

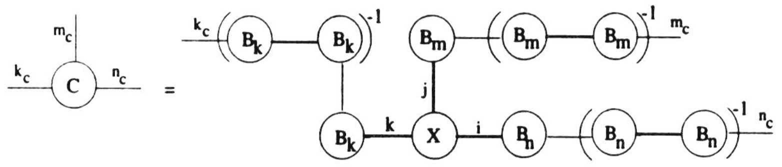


Fig. 5. Diagram of how to find the coefficient core matrix in three-mode B-spline compression. This diagram is obtained by solving for $\underline{\mathbf{C}}$ in Fig. 4.

Table 2
Dimensions of matrices involved in the experiment

Matrix	Dimensions
G	[71 × 3]
H	[38 × 3]
E	[44 × 3]
$\mathbf{G}_c$	[30 × 3]
$\mathbf{H}_c$	[13 × 3]
$\mathbf{E}_c$	[17 × 3]

mial for the m mode were [1 × 34] for $\boldsymbol{h}_n$, [1 × 16] for $\boldsymbol{h}_m$ and [1 × 21] for $\boldsymbol{h}_k$. The basis matrix $\mathbf{B}_n$ had size [71 × 30], $\mathbf{B}_m$ had size [38 × 13] and $\mathbf{B}_k$ had size [44 × 17]. The resulting $\underline{\mathbf{C}}$ 3-array from the B-spline compression had consequently dimensions [30 × 13 × 17] which is a compression ratio of ca.

$$\frac{71 \cdot 38 \cdot 44}{30 \cdot 13 \cdot 17} \approx 17.9$$

The 3MPCA program was used on both the original representation $\underline{\mathbf{X}}$ and the compressed representation $\underline{\mathbf{C}}$.

Two sets of loading matrices were produced: {**G**, **H**, **E**} for the original uncompressed representation and {$\mathbf{G}_c$, $\mathbf{H}_c$, $\mathbf{E}_c$} for the compressed representation. Three factors were extracted for both representations. The sizes of the different loading matrices are shown in Table 2.

The total number of FLOPS consumed for the original data array was $F = 75677508$. The corresponding number of FLOPS using the compressed representation was $F_c = 4799174$. The ra-

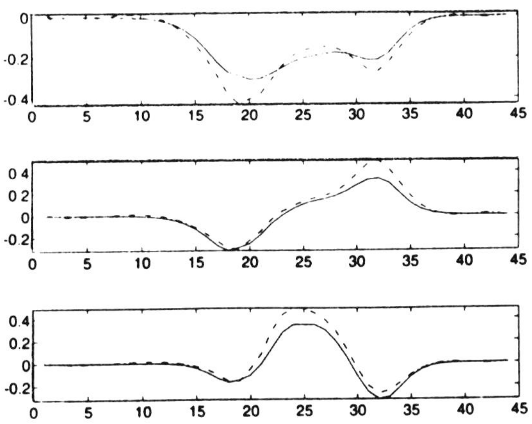


Fig. 7. Comparison between estimated and original representation of the **E** matrix. The dotted line indicates the estimated **E** matrix. The upper figure is the first factor, the middle figure is the second factor and the bottom figure is the third factor.

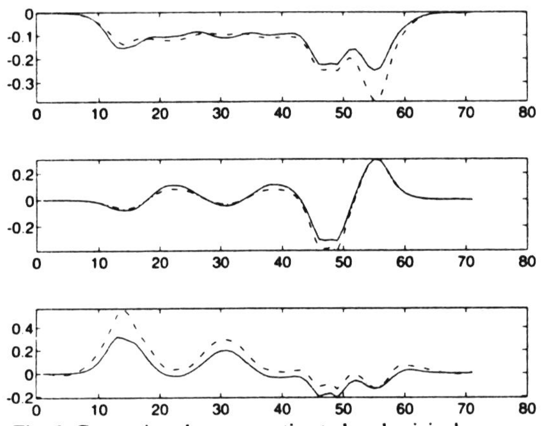


Fig. 6. Comparison between estimated and original representation of the **G** matrix. The dotted line indicates the estimated **G** matrix. The upper figure is the first factor, the middle figure is the second factor and the bottom figure is the third factor.

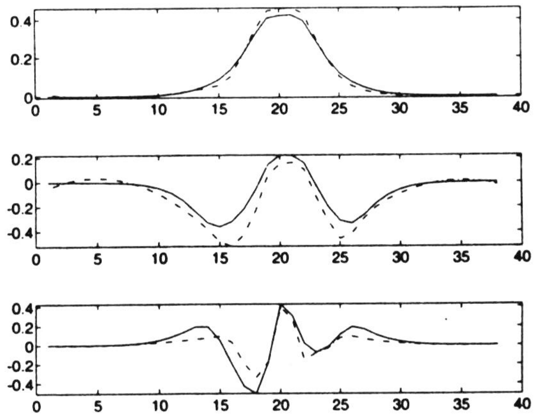


Fig. 8. Comparison between estimated and original representation of the **H** matrix. The dotted line indicates the estimated **H** matrix. The upper figure is the first factor, the middle figure is the second factor and the bottom figure is the third factor.

tio $F/F_c \approx 15.8$ is in the same range as the compression ratio. Using the derived formula 16 for the number of FLOPS consumed by the ALS algorithm the estimated ratio was approximately 14.7 which is in close agreement with the observed values. A comparison between the original loading matrices **G**, **H** and **E** and the estimated loading matrices $\hat{\mathbf{G}}$, $\hat{\mathbf{H}}$ and $\hat{\mathbf{E}}$ from the compressed representation was made. The following equations were used:

$$\hat{\mathbf{G}} = \mathbf{B}_n \mathbf{G}_c \tag{18}$$

$$\hat{\mathbf{H}} = \mathbf{B}_m \mathbf{H}_c \tag{19}$$

$$\hat{\mathbf{E}} = \mathbf{B}_k \mathbf{E}_c \tag{20}$$

It must be stressed that these estimates are not strictly correct, i.e., they do not provide true estimates which should be column-wise orthonormal. Orthogonalization of the loading matrix does not produce the correct result. At the present this is the only way of obtaining such estimates without rewriting 3MPCA to compensate for distortion effects in backestimates. It is possible to construct a 3MPCA program which compensates for these effects. A detailed presentation of a solution to the problem is presented in two forthcoming papers [19,20].

The comparison between $\hat{\mathbf{G}}$ and **G** can be seen in Fig. 6, between $\hat{\mathbf{E}}$ and **E** in Fig. 7 and between $\hat{\mathbf{H}}$ and **H** in Fig. 8. The results for this data set must be characterized as satisfactory.

6. Discussion

B-splines are not necessarily the optimal choice of basis functions for compression. B-splines can be viewed as a subset of the much broader class of wavelets [21] which has been used in several problems in physics, chemistry and image compression. The problem with compression as used in 3MPCA is that the estimates of later factors increasingly deviate from the true factors of the uncompressed array. The compression stage *does* remove information so there is a trade-off between faster computations and accuracy. In addition it has been stressed that the data must be smooth which is not always the case. For the case of non-smooth *N*-arrays when all the modes are large, no solution to the increased computational problem has yet been found. The use of coefficients instead of the original data has similarities to the sparse matrix technology [22]. Both methods utilize the special structure in data arrays to achieve compression and faster computation. Sparse algorithms can be constructed based on the fact that the matrix contains a large amount of zeros. These algorithms give the exact answer and therefore no errors are introduced because of the compressed representation. B-splines could have been fitted to the data such that it was perfect, but for real world data no compression or improvement in speed or storage would have been obtained. Still the development of more efficient higher mode algorithms is very important and should be used in combination with new ways of compressing/representing the array structure.

Acknowledgements

The authors wish to thank Dr. René Henrion at Humboldt University in Berlin for discussing the ALS algorithm and compression. Dr. Knut Børve at the University of Bergen, is thanked for providing the electron density distribution data of GABA. BKA is indebted to The Royal Norwegian Council for Scientific and Industrial Research (NTNF) for financial support.

Appendix A. Definitions and terminology

A.1. Names of data objects

The data objects discussed in the article are referred to by many different names. Some of the most common ones are tensor, *N*-order array, higher order array/matrix, *N*-way array, *N*-array, *N*-mode array or *N*-dimensional array. Thus the names mode, order, way and dimension are used to designate the different indices in the array. In this article we have chosen to use the word 'mode' when discussing the different indices in the arrays. A matrix has two indices and thus two modes. A 3-array has three indices and thus three modes and so on.

A.2. Typographical notes

All scalars are written as lowercase letters, e.g., k. All vectors are written as bold italic lowercase letters, e.g., $\boldsymbol{u}$. All matrices are written as bold uppercase letters, e.g. **P**. We use underlined, boldface uppercase letters to indicate an array with three modes, e.g. $\underline{\mathbf{X}}$. Transpose is indicated by a superscript T.

A.3. Definition of unfolding

The word 'unfolding' is defined to be the reorganization of an N-mode array into a vector or a matrix. A more detailed description of unfolding can be found in refs. [23,24].

Appendix B. Notations for N-mode equations

B.1. The Kronecker notation

The $\otimes$ symbol signifies the Kronecker product [23]. The right-oriented Kronecker product of two matrices **A** and **B** of dimensions $[n \times m]$ and $[k \times j]$ is

$$\mathbf{M} = \mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \dots & a_{1m}\mathbf{B} \\ \vdots & & \vdots \\ a_{n1}\mathbf{B} & \dots & a_{nm}\mathbf{B} \end{bmatrix} \tag{21}$$

where **M** has dimensions $[nk \times mj]$.

If a 3-array $\underline{\mathbf{X}}$ has dimensions $[I \times J \times K]$ it can be rearranged into three essential matrices: $[K \times IJ]$, $[I \times KJ]$ and $[J \times KI]$ (or $[K \times JI]$, $[I \times JK]$ and $[J \times IK]$). There are in general $N!$ possible unfoldings or permutation of indices. The Kronecker notation thus represents all N-arrays as ordinary matrices by unfolding.

B.2. The diagram notation

The diagram notation is fully described in ref. [25] and only a short summary will be given here. The diagram notation is a graphical visualization of the index topology in explicit summation notation. The diagrams have the appearances of graphs and use of graph terminology is therefore appropriate [26]. A diagram contains nodes and edges. Nodes signify array elements (e.g., x_{ijk}) and one edge can signify either summation over one index or an index which is not involved in summation. An edge can be attached to one or several nodes. An edge connected to one node only is called *unconnected*. An edge connected to more than one node is called *connected*. It is possible to enable connection between more than two nodes by introducing a special *sum nexus* symbol, but this is not presented in this article. For more details see ref. [25]. A node with N unconnected edges will be the diagram representation of a single array element not connected to any other. A connected edge signifies summation of one index. When two arrays share a common index a connected edge is drawn between them. The total number of unconnected edges of the expression is the number of modes (or the mode number) of the result. If, e.g., two 3-arrays combine by summing one common index the mode number of the result will be $3 + 3 - 2 = 4$. In the center of the node the name of the array is placed. In the vicinity of the edges the correct index names can be written in order to clarify. The index names are written anti-clockwise from the first index. Sometimes it is necessary to indicate which edge signifies the first index. For this a small bar perpendicular to the edge is used; this mark is called the *first index pointer* or just the *fip*.

Fig. 9 shows a few examples of array diagram equations. The corresponding summation formulas for the diagram equations presented are:

$$\sum_i u_i v_i \tag{A}$$

This is the standard inner product.

$$\sum_k x_{ik} y_{kj} \tag{B}$$

This is a standard matrix product.

$$\sum_k \sum_f q_{ifk} w_{kfj} \tag{C}$$

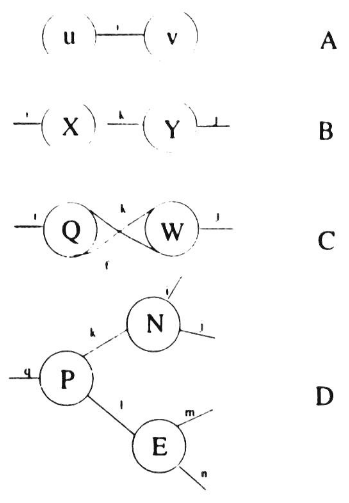


Fig. 9. Examples of diagrams. (A) is an inner product of two vectors. (B) is a matrix product. (C) and (D) are examples of products between three-mode arrays.

An array product between two three-mode arrays. The result is a matrix because the number of free or unconnected indices is two.

$$\sum_k \sum_l p_{qkl} n_{kij} e_{lnm} \qquad \text{(D)}$$

Here the result is a five mode array since five free indices are seen.

Instead of using index names and fips for indicating the different modes of the arrays a shorthand notation has been constructed. There are especially two cases where a shorthand notation has been found useful: (i) matrices with orthonormal column vectors; (ii) modes of large size.

If **W** is a matrix with orthonormal column vectors we have that $\mathbf{W}^T\mathbf{W} = \mathbf{I}$ where **I** is the identity matrix. If **W** is not square we have that $\mathbf{W}\mathbf{W}^T \neq \mathbf{I}$. Thus we need to discriminate between the two different modes of the matrix. Arrows have been chosen to distinguish between the modes. When two arrows meet head to head we have the case $\mathbf{W}^T\mathbf{W} = \mathbf{I}$. Fig. 10 illustrates the idea. The identity matrix is for convenience drawn as a connected or as an unconnected edge with no matrix element attached.

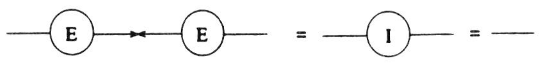


Fig. 10. A diagram with arrows is used to mark the two different indices on orthogonal matrices. Here we have chosen the convention to let two arrows heading versus each other signify an identity.

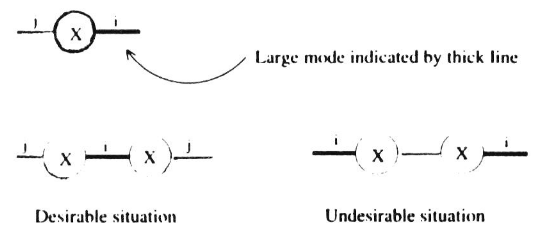


Fig. 11. For a SVD of a matrix of size [10×10000] it is desirable to avoid the large mode. The large mode is indicated by a thick line.

In some problems it is necessary to avoid that large modes become unconnected edges. A simple example from SVD illustrates the main idea. If the dimension of **X** is [10 × 10000] and the object is to find the eigenvalues the fastest method is to calculate the eigenvalues of the covariance matrix $\mathbf{XX}^T$ which has a dimension of [10 × 10]. The rank of **X** cannot be larger than 10 which means that eigenvalue decomposition of $\mathbf{X}^T\mathbf{X}$ (dimension [10000 × 10000]) would be a waste of resources; see Fig. 11 for illustration.

References

[1] O.M. Kvalheim and Y. Liang, Heuristic evolving latent projections: resolving two-way multicomponent data. 1. Selectivity, latent-projective graph, datascope, local rank, and unique resolution, *Analytical Chemistry*, 64 (1992) 937–946.

[2] Y. Liang, O.M. Kvalheim, H.R. Keller, D.L. Massart, P. Kiechle and F. Erni, Heuristic evolving latent projections: resolving two-way multicomponent data. 2. Detection and resolution of minor constituents, *Analytical Chemistry*, 64 (1992) 947–953.

[3] I. Pelczer and S. Szalma, Multidimensional NMR and data processing, *Chemical Review*, 91 (1991) 1507–1524.

[4] 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.

[5] 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 FTIR spectra, *Journal of Chemometrics*, 8 (1994) 127–146.

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

[7] J.A. Storer, *Data Compression. Methods and Theory*, Computer Science Press, Rockville, MD, 1988.

[8] P.M. Kroonenberg, *Three Mode Principal Component Analysis*, DSWO Press, Leiden, 1983.

[9] L. Tucker, Some mathematical notes on three-mode factor analysis, *Psychometrika*, 31 (1966) 279-311.

[10] *MATLAB Reference Guide*, The MathWorks Inc., Natick, MA, 1992.

[11] C. de Boor, *A Practical Guide to Splines. Applied Mathematics Sciences*, Springer, New York, 1978.

[12] G. Farin, *Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide*, Academic Press, San Diego, CA, 2nd edn., 1990.

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

[14] T.V. Karstang and R.J. Eastgate, Multivariate calibration of an X-ray diffractometer by partial least squares regression, *Chemometrics and Intelligent Laboratory Systems*, 2 (1987) 209-219.

[15] H.A.L. Kiers and W.P. Krijnen, An efficient algorithm for PARAFAC of three-way data with large number of observation units, *Psychometrika*, 56 (1992) 147-152.

[16] H.A.L. Kiers, P.M. Kroonenberg and J.M. ten Berge, An efficient algorithm for Tuckals3 on data with large number of observation units, *Psychometrika*, 57 (1992) 415-422.

[17] M.J.S. Dewar, E.G. Zoebisch, F.H. Eamonn and J.P. Stewart, AM1: A new general purpose quantum mechanical molecular model, *Journal of the American Chemical Society*, 107 (1985) 3902-3909.

[18] C. de Boor, *Spline Toolbox for Use with MATLAB. User's Guide*, The MathWorks Inc., South Natick, MA, 1990.

[19] B.K. Alsberg and O.M. Kvalheim, Speed improvement of multivariate algorithms by the method of postponed basis matrix multiplication. Part I. Principal component analysis, *Chemometrics and Intelligent Laboratory Systems*, (1994) in press.

[20] 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*, (1994) in press.

[21] Y. Meyer, *Wavelets. Algorithms and Applications*, SIAM Book, 1993.

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

[23] J.R. Magnus and H. Neudecker, *Matrix Differential Calculus with Applications in Statistics and Econometrics*, Wiley, Essex, 1988.

[24] R. Henrion, N-way principal component analysis — theory, algorithms and applications, *Chemometrics and Intelligent Laboratory Systems*, (1993) submitted.

[25] B.K. Alsberg, A diagram notation for *N*-mode array equations, *Psychometrika*, (1993) in press.

[26] J.R. Wilson, *Introduction to Graph Theory*, Longman Scientific and Technical, Essex, 3rd edn., 1985.

Paper IV

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 $\mathbf{B}_2$:

$$\tilde{\mathbf{X}} = \mathbf{B}_1\mathbf{C}\mathbf{B}_2^T \tag{1}$$

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.

SSDI 0169-7439(94)00013-9

structure in $\mathbf{X}$. The estimation of $\mathbf{C}$ 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 $\mathbf{C}$ with PCA the following is obtained (the subscript c is used to designate matrices and vectors associated with the compressed representation $\mathbf{C}$):

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

which is comparable to:

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

The dimensions of the matrices are: Dim($\mathbf{X}$) = $[N \times M]$, Dim($\mathbf{C}$) = $[n \times m]$, Dim($\mathbf{T}$) = $[N \times A]$, Dim($\mathbf{P}$) = $[M \times A]$, 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

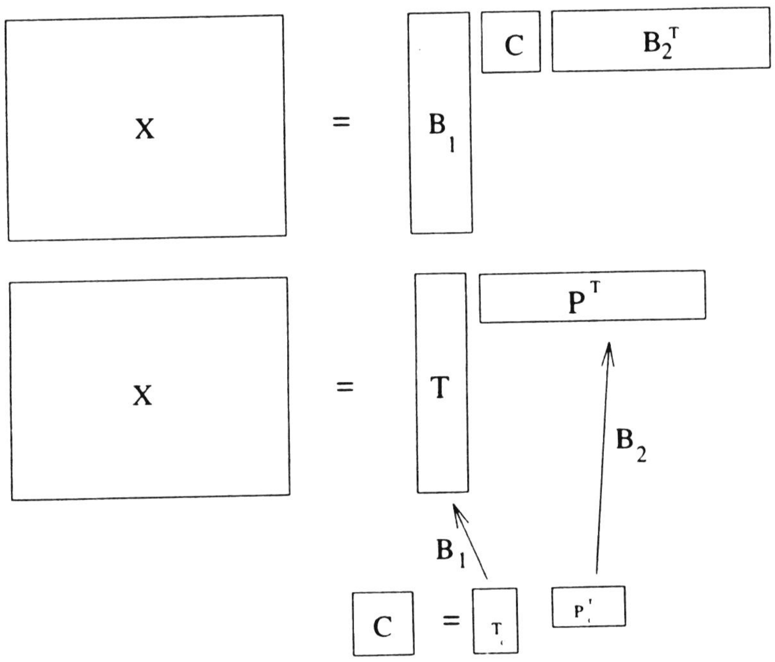


Fig. 1. The various matrices used in the compression and PCA analysis. By utilizing the fact that $\mathbf{X}$ can be represented by linear combinations of compression bases, $\mathbf{C}$ can be used instead. It is a problem, however, that the scores and loadings vectors from analysis of $\mathbf{C}$, $\mathbf{T}_c$ and $\mathbf{P}_c^T$ cannot be the true estimates of $\mathbf{T}$ and $\mathbf{P}^T$ of $\mathbf{X}$. 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 $\| t_0 - t_1 \|$ 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 $\boldsymbol{t}_c$ and $\boldsymbol{p}_c^{\mathrm{T}}$ 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 ($\mathbf{B}_1$, $\mathbf{B}_2$, **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)
$\boldsymbol{p}_0^T = \boldsymbol{t}_0^T \boldsymbol{X}$	$\boldsymbol{v}_0^T\mathbf{B}_2^T = \boldsymbol{u}_0^T\mathbf{\Gamma}_1\mathbf{C}\mathbf{B}_2^T$	$\boldsymbol{v}_0^T = \boldsymbol{u}_0^T\mathbf{\Gamma}_1\mathbf{C}$
$\boldsymbol{p}_1^T = \dfrac{\boldsymbol{p}_0^T}{\Vert \boldsymbol{p}_0^T \Vert}$	$\boldsymbol{v}_1^T\mathbf{B}_2^T = \dfrac{\boldsymbol{v}_0^T}{(\boldsymbol{v}_0^T\mathbf{\Gamma}_2\boldsymbol{v}_0)^{1/2}}\mathbf{B}_2^T$	$\boldsymbol{v}_1^T = \dfrac{\boldsymbol{v}_0^T}{(\boldsymbol{v}_0^T\mathbf{\Gamma}_2\boldsymbol{v}_0)^{1/2}}$
$\boldsymbol{t}_1 = \mathbf{X}\boldsymbol{p}_1^T$	$\boldsymbol{B}_1\boldsymbol{u}_1 = \mathbf{B}_1\mathbf{C}\mathbf{\Gamma}_2\boldsymbol{v}_1$	$\boldsymbol{u}_1 = \mathbf{C}\mathbf{\Gamma}_2\boldsymbol{v}_1$
$\boldsymbol{t}_0 = \boldsymbol{t}_1$	$\mathbf{B}_1\boldsymbol{u}_0 = \mathbf{B}_1\boldsymbol{u}_1$	$\boldsymbol{u}_0 = \boldsymbol{u}_1$
$\mathbf{X} = \mathbf{X} - \boldsymbol{t}_1\boldsymbol{p}_1^T$	$\mathbf{B}_1\mathbf{C}\mathbf{B}_2^T = \mathbf{B}_1\mathbf{C}\mathbf{B}_2^T - \mathbf{B}_1\boldsymbol{u}_1\boldsymbol{v}_1^T\mathbf{B}_2^T$	$\mathbf{C} = \mathbf{C} - \boldsymbol{u}_1\boldsymbol{v}_1^T$

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 $\text{Dim}(\mathbf{X}) = [N \times M]$, 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
$\mathbf{XY}$	$2nmk$	$\text{Dim}(\mathbf{X}) = [n \times m]$, $\text{Dim}(\mathbf{Y}) = [m \times k]$
$\mathbf{X}\boldsymbol{y}$	$2nm$	$\text{Dim}(\mathbf{X}) = [n \times m]$, $\text{Dim}(\boldsymbol{y}) = [m \times 1]$
$\boldsymbol{y}^T\boldsymbol{y}$	$2m$	$\text{Dim}(\boldsymbol{y}) = [m \times 1]$
$\mathbf{X} + \mathbf{X}$	nm	$\text{Dim}(\mathbf{X}) = [n \times m]$
$\mathbf{X} = \mathbf{X} - \boldsymbol{t}\boldsymbol{p}^T$	$3nm$	$\text{Dim}(\mathbf{X}) = [n \times m]$, $\text{Dim}(\boldsymbol{t}) = [n \times 1]$ $\text{Dim}(\boldsymbol{p}) = [m \times 1]$,
$(\mathbf{XY})\mathbf{Z}$	$2nk(m + r)$	$\text{Dim}(\mathbf{X}) = [n \times m]$, $\text{Dim}(\mathbf{Y}) = [m \times k]$ $\text{Dim}(\mathbf{Z}) = [k \times r]$
$\mathbf{X}(\mathbf{YZ})$	$2mr(k + n)$	

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.

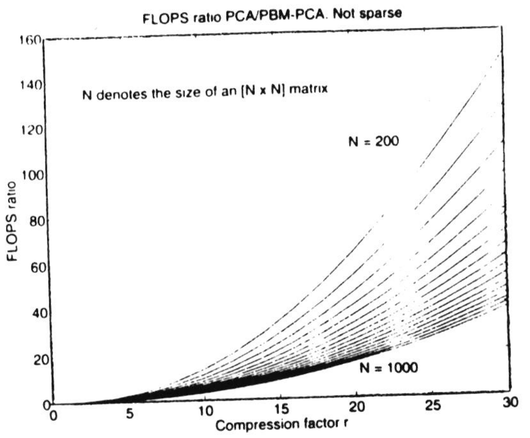


Fig. 2. FLOPS ratio between PCA and PBM–PCA algorithms when using various values for N and r. N denotes the size of a matrix **X** of dimensions $[N \times N]$, r is the compression along each mode. The compressed **C** has dimensions $[N/r \times N/r]$. No sparse representation is assumed here and thus 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 $\mathbf{B}_i^{\mathrm{T}}\mathbf{B}_i$.

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 **Γ** 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

$$\mathcal{F}_2 = A\left[q_a(4nm + 2m^2g_2 + 3m + 2) + 3nm + 2n^2mg_1 + 2nm^2g_2\right] + 2n^2Nd_1^2 + 2m^2Md_2^2 + 2n^2mg_1 + 2nm^2g_2 \tag{34}$$

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.

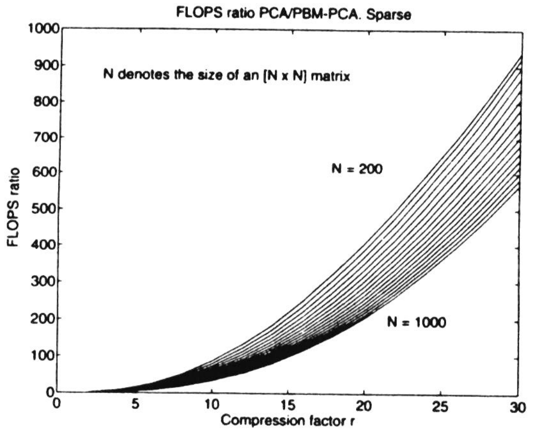


Fig. 3. FLOPS ratio between PCA and PBM–PCA algorithms when using various values for N and r. Here sparse representation is assumed. N denotes the size of a matrix **X** of dimensions $[N \times N]$, r is the compression along each mode. The compressed **C** has dimensions $[N/r \times N/r]$. The following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

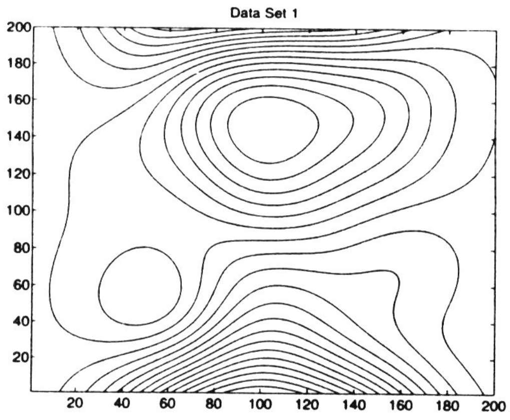


Fig. 4. Data set 1, Dim(**X**) = [200 × 200].

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 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

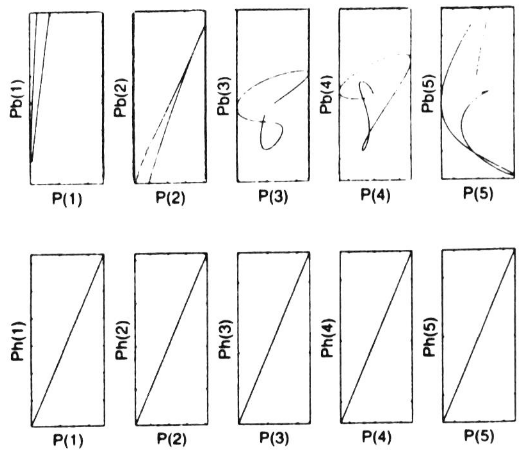


Fig. 6. Results from analysis of data set 1. Upper row shows the comparison between true loading vectors ($\mathbf{P}^T$ is printed as P(i) in the figure where i is the ith factor) for the first five components versus the estimated loading vectors based on standard PCA on the coefficient matrix alone 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).

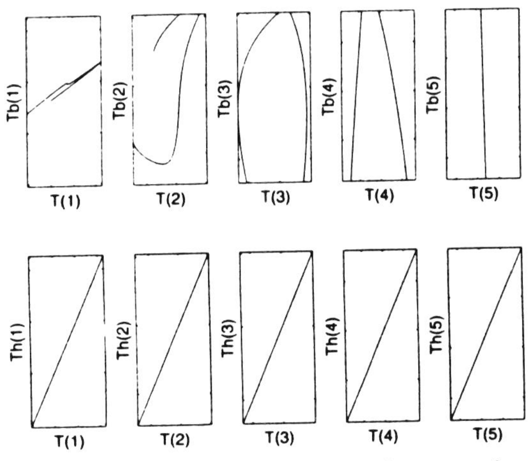


Fig. 5. Results from analysis of data set 1. Upper row shows the comparison between true score vectors (**T** is printed as T(i) in the figure where i is the ith factor) for the first five components versus the estimated score vectors based on standard PCA on the coefficient matrix alone and multiplied 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).

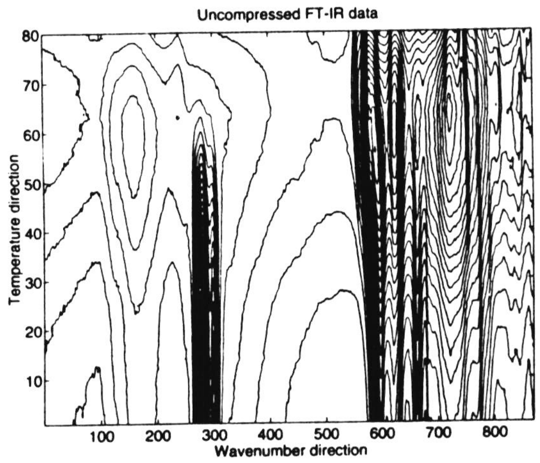


Fig. 7. The temperature–IR data set. Dim(**X**) = [80 × 869].

expected $\mathbf{T}_h$ is equal to $\mathbf{T}$. The upper part of Fig. 6 shows $\mathbf{P}^T$ versus $\mathbf{P}_h^T$ 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 multi-

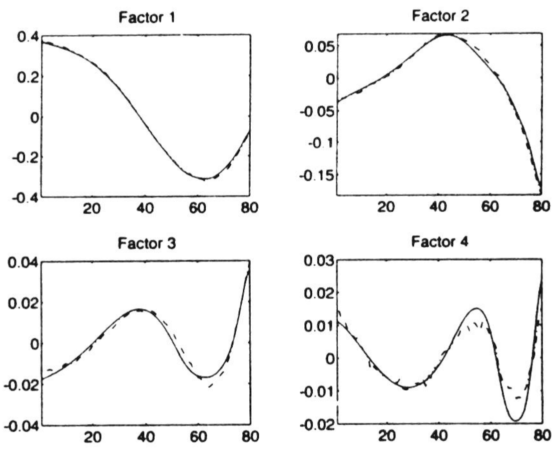


Fig. 8. Comparison of the scores vectors for PCA on uncompressed matrix (dash-dotted lines) and PBM–PCA using compression model (solid lines). Results from analysis of data set 2.

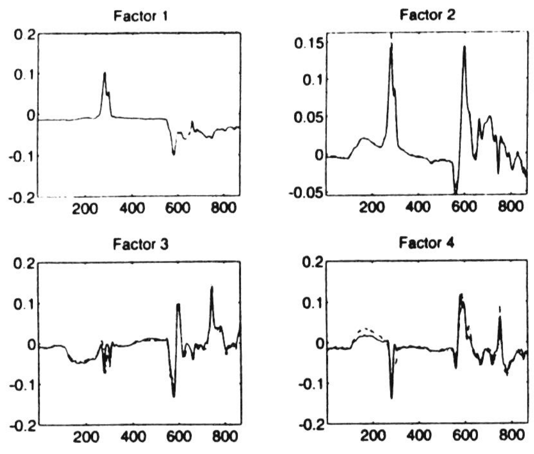


Fig. 9. Comparison of the loadings vectors for PCA on uncompressed matrix (dash-dotted lines) and PBM–PCA using compression model (solid lines). Results from analysis of data set 2.

plied 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 $F_1 =$

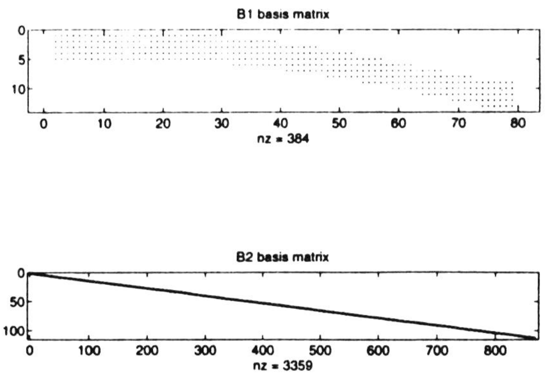


Fig. 10. Sparsity of the two basis matrices $\mathbf{B}_1$ and $\mathbf{B}_2$ used on data set 2. nz stands for the number of nonzero elements in the matrix.

12 131 240 and $\mathcal{F}_2 = 579054$ 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 $\hat{\mathcal{F}}_2 = 463750$ 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.

Paper V

ELSEVIER

Chemometrics and Intelligent Laboratory Systems 24 (1994) 43–54

Chemometrics and intelligent laboratory systems

Speed improvement of multivariate algorithms by the method of postponed basis matrix multiplication Part II. Three-mode 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 arrays faster. Compression is here defined as the case when a large array $\underline{\mathbf{X}}$ is replaced by a smaller *coefficient* array $\underline{\mathbf{C}}$. The coefficients are obtained by least squares fitting to some compression basis. This paper will deal with three-mode arrays. When performing, e.g., three-mode principal component analysis of $\underline{\mathbf{C}}$ the results are comparable but not equal to the results from analyzing $\underline{\mathbf{X}}$. In this paper we suggest a solution to this problem by rewriting the three-mode PCA algorithm which utilizes $\underline{\mathbf{C}}$ 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 three-mode PCA algorithm.

1. Introduction

Compression of raw data from analytical instruments is useful for storage and faster computation of large arrays [1,2]. The approach advocated by us is to fit the original data to a suitable chosen basis set and use the *coefficients* obtained from the fitting instead of the original data. When using such coefficients in, e.g., a principal component analysis (PCA) scores and loadings are not directly comparable to the corresponding scores and loadings of the uncompressed array. One way to circumvent this problem is to premultiply the scores and loading matrices with the compression matrices used [3]. This multiplication, however, does not produce orthogonal scores and loading vectors. A simple orthogonalization/orthonormalization of the transformed matrices does not provide the correct result. In Part I of this article [4], a method was developed for PCA of two-mode arrays which solved the problem. This method is referred to as the method of postponed basis matrix multiplication (PBM) where the central idea is to postpone the multiplication of large basis matrices until after the converged PCA solution is obtained for all factors. The aim of this work is to show the PBM method can be applied to PCA of three-mode arrays. The method can be extended to *N*-mode PCA also.

A three-array $\underline{\mathbf{X}}$ is assumed to be modeled by three compression matrices $\{\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3\}$. After

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

0169-7439/94/$07.00 © 1994 Elsevier Science B.V. All rights reserved
SSDI 0169-7439(94)00014-A

compression the original three-array can be represented by a much smaller three-array of coefficients $\underline{\mathbf{C}}$. $\tilde{\underline{\mathbf{X}}}$ is the *reconstructed* array which is constructed exclusively from $\underline{\mathbf{C}}$ and the basis matrices ($\{\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3\}$). Of course it is not necessary to compress along all the modes. This is the case if some modes are much smaller than the other modes or uncompressible by the chosen basis. If the error $\underline{\mathbf{E}} = \underline{\mathbf{X}} - \tilde{\underline{\mathbf{X}}}$ is large and contains important components the compression will not be satisfactory and the coefficient representation does not reflect the essential structures in $\underline{\mathbf{X}}$. The scores and loadings from the analysis of $\underline{\mathbf{C}}$ have smaller dimensions since the coefficient array is much smaller than $\tilde{\underline{\mathbf{X}}}$ and $\underline{\mathbf{X}}$. As was mentioned above a premultiplication of the basis matrix associated with each mode can be used to increase the dimensions of the scores and loadings vectors obtained from analysis from $\underline{\mathbf{C}}$ to the dimensions of the scores and loadings vectors of $\tilde{\underline{\mathbf{X}}}$ or $\underline{\mathbf{X}}$. When these reconstructed scores and loadings vectors of $\underline{\mathbf{C}}$ are compared to the corresponding vectors of $\tilde{\underline{\mathbf{X}}}$, it is found that a distortion has been introduced. Sometimes this distortion is so small that it has no practical consequence. For other problems the distortions can be quite large. Simple orthogonalizations/orthonormalizations or other transformations have been unsuccessful in trying to nullify the distortions. The PBM method, however, solves the problem by rewriting the algorithm in question such that when the basis matrices are premultiplied with the PBM scores and loadings they are identical to the scores and loadings obtained from direct analysis of $\tilde{\underline{\mathbf{X}}}$.

A diagram notation [5] is used to represent the three-mode array equations and the reader is encouraged to study the Appendix for an explanation of the notation used.

2. Three-mode PCA

It is instructive to start an explanation of three-mode PCA by first looking at how singular value decomposition (SVD) is performed. In SVD for matrices (two-arrays) an $\mathbf{X}$ matrix can be decomposed as

$$\mathbf{X} = \mathbf{U}\mathbf{S}^{1/2}\mathbf{V}^{\mathrm{T}} \tag{1}$$

where $\mathbf{U}$ and $\mathbf{V}$ are columnwise orthonormal matrices and $\mathbf{S}^{1/2}$ is a diagonal matrix containing the square root of eigenvalues of the two covariance matrices $\mathbf{X}\mathbf{X}^{\mathrm{T}}$ and $\mathbf{X}^{\mathrm{T}}\mathbf{X}$. The two associated eigen equations are

$$\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{U} = \mathbf{U}\mathbf{S} \tag{2}$$

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{V} = \mathbf{V}\mathbf{S} \tag{3}$$

In SVD for 3-arrays an $\underline{\mathbf{X}}$ array can be decomposed as shown in Fig. 1. This equation is the Tucker3 model [6,7]. The diagram equation in Fig. 1 can also be expressed in explicit summation notation:

$$x_{ijk} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} h_{ip} g_{jq} e_{kr} d_{pqr} \tag{4}$$

There are different methods for solving the Tucker3 model. A method similar to the approach used for SVD of two-arrays (see Eqs. (2) and (3)) is the Tucker Method I [6,7] which obtains estimates for the loading matrices $\mathbf{H}$, $\mathbf{G}$, $\mathbf{E}$ by extracting *all* eigenvectors corresponding to non-zero roots of the three covariance matrices with elements, see Fig. 2:

$$l_{ii'} = \sum_{j=1}^{J} \sum_{k=1}^{K} x_{ijk} x_{i'jk} \tag{5}$$

$$m_{jj'} = \sum_{i=1}^{I} \sum_{k=1}^{K} x_{ijk} x_{ij'k} \tag{6}$$

$$n_{kk'} = \sum_{i=1}^{I} \sum_{j=1}^{J} x_{ijk} x_{ijk'} \tag{7}$$

Using the estimated three loading matrices, Eq. (4) shows the core array $\underline{\mathbf{D}}$ can be expressed as

$$d_{pqr} = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} h_{ip} g_{jq} e_{kr} x_{ijk} \tag{8}$$

This equation is also shown in Fig. 3 using the diagram notation.

In practice an investigator is often interested in only the first largest eigenvectors of $\mathbf{H}$, $\mathbf{G}$, $\mathbf{E}$. Using the Tucker Method I, however, the estimators for d_{pqr} will no longer be least-squares ones.

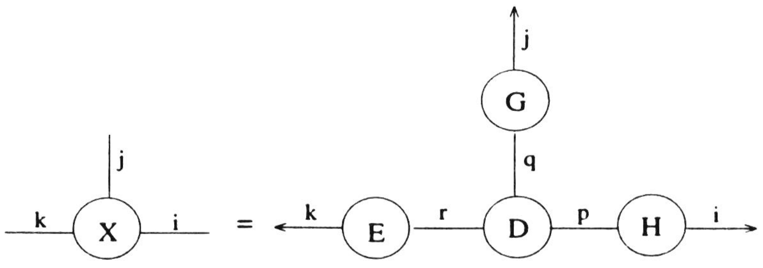


Fig. 1. Illustration of the Tucker3 model.

In order to achieve least squares estimates an alternating least squares (ALS) algorithm [6] can be used. The diagram formulation of this algorithm is presented in Fig. 4. The ALS algorithm is very computer demanding and the idea of using the compressed representation $\underline{\mathbf{C}}$ instead of $\underline{\mathbf{X}}$ to

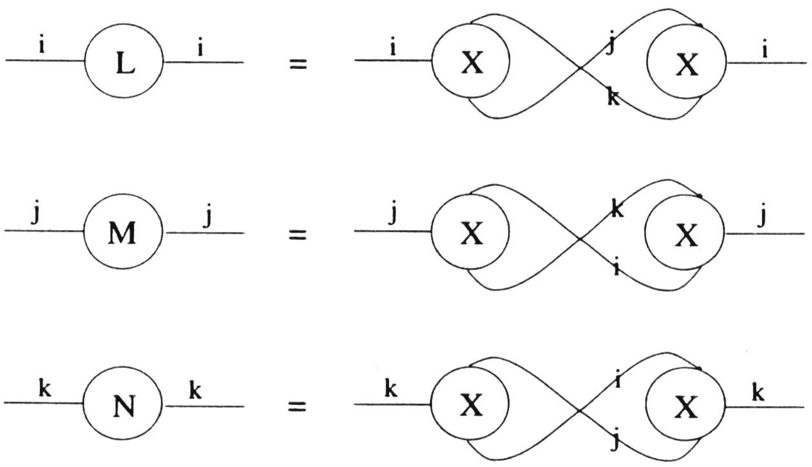


Fig. 2. Construction of the covariance like matrices L, M, and N used in the Tucker 1 algorithm. In the Tucker3 model the eigenvectors of these matrices are used as initial estimates before entering the alternating least squares iterations.

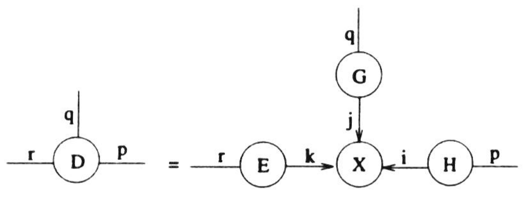


Fig. 3. How to find the core array using a Tucker Method I (here $p = i$, $q = j$, and $r = k$). For cases when $p < i$, $q < j$, and $r < k$, the solution is more complicated.

enable faster computations on three-mode arrays is promising [2,8]. When using $\underset{\sim}{\mathbf{C}}$ the ALS algorithm returns the much smaller loading matrices $\mathbf{H}_c$, $\mathbf{G}_c$ and $\mathbf{E}_c$. The incorrect transformation back into the original domain by premultiplication of the corresponding compression matrices as mentioned in Part I of this article is formulated as

$$\mathbf{H}_b = \mathbf{B}_1\mathbf{H}_c \tag{9}$$

$$\mathbf{G}_b = \mathbf{B}_2\mathbf{G}_c \tag{10}$$

$$\mathbf{E}_b = \mathbf{B}_3\mathbf{E}_c \tag{11}$$

The PBM method will provide different $\mathbf{H}_c$, $\mathbf{G}_c$, and $\mathbf{E}_c$ which subjected to proper premultiplication of basis matrices are identical to the loading matrices obtained from direct analysis of $\underset{\sim}{\tilde{\mathbf{X}}}$.

3. The PBM method on three-mode PCA

For an introduction to the PBM method please consult Part I of this article.

There are two different places in the ALS algorithm where postponing of basis matrix multiplication can be made:

1. The computation of matrices which are to be diagonalized. These steps are the array equations as described in detail for one of the three loadings matrices in Fig. 5. These steps are referred to as the 'ALS iteration steps'.

2. The eigenvalue decomposition algorithm which is used to obtain estimates of the loading matrices $\mathbf{H}_c$, $\mathbf{G}_c$, $\mathbf{E}_c$.

The following two subsections will discuss in detail how each of the two parts of the ALS

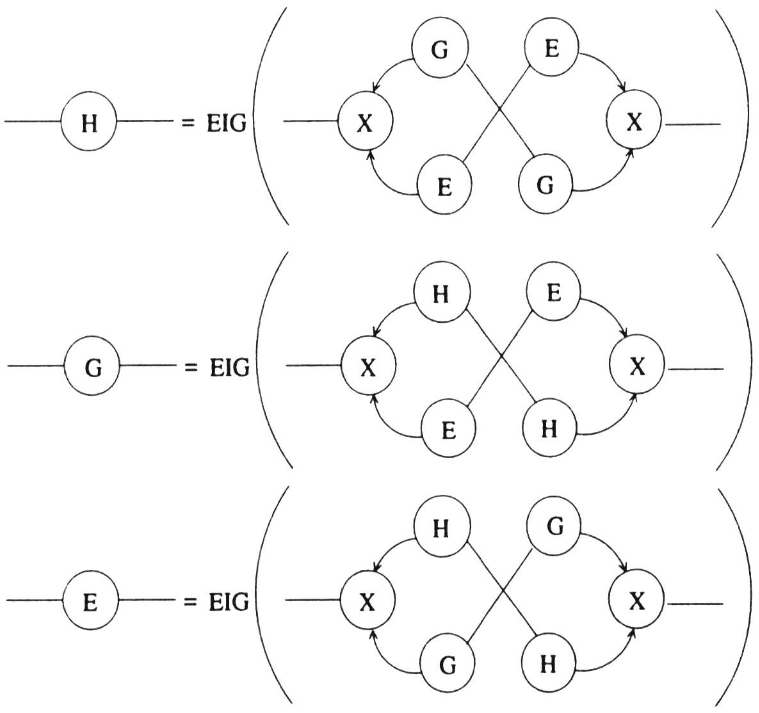


Fig. 4. Central part of the ALS algorithm. Here estimates of the loadings matrices for each mode are generated. The figure shows one iteration.

algorithm is changed according to the PBM method.

3.1. PBM eigenvalue decomposition

In Fig. 4 the word 'EIG' is used to indicate that the resulting matrix of the diagram expression inside the parentheses is diagonalized (EIG returns the A largest eigenvectors). Using the PBM method it is easy to see that the basis matrix associated with the mode is redundant and unnecessary for the computation of eigenvalues and eigenvectors.

Let $\mathbf{K}$ be a symmetric matrix which is to be subjected to an eigenvalue decomposition and assumed to be generated from

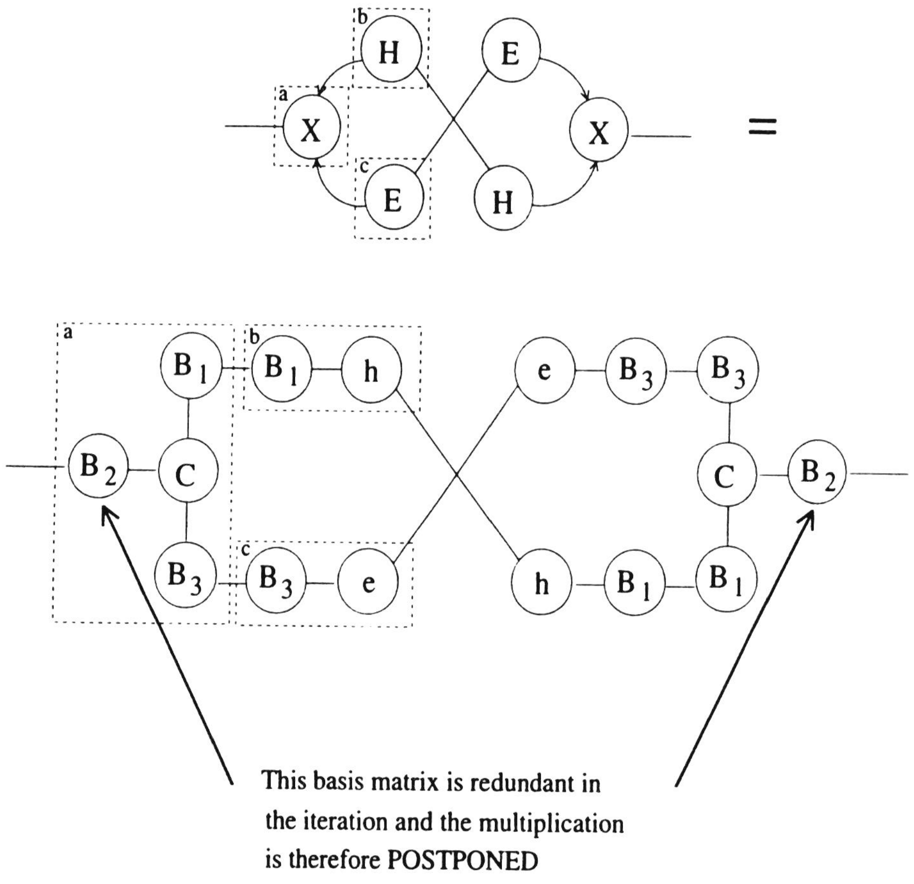
$$\mathbf{K} = \mathbf{R}^T\mathbf{R} \tag{12}$$

where

$$\mathbf{R} = \mathbf{W}\mathbf{B}^T \tag{13}$$

i.e., a result from a compression model ($\mathbf{B}^T$ is the compression basis matrix). An alternative formulation of $\mathbf{K}$ is

$$\mathbf{K} = \mathbf{B}\mathbf{W}^T\mathbf{W}\mathbf{B}^T \tag{14}$$



a
b
c
H
E
X
X
E
H
=
B1
B1
h
e
B3
B3
B2
C
C
B2
B3
B3
e
h
B1
B1
This basis matrix is redundant in the iteration and the multiplication is therefore POSTPONED

Fig. 5. The central part of the PBM method is illustrated for one part of the ALS iteration steps using diagram notation. When the model equations are written out it is possible to discover that redundant multiplication of large basis matrices is present. The multiplication of these large basis matrices is postponed to after the iteration has converged.

The algorithm chosen for the eigenvalue decomposition of real symmetric matrices is a power method approach [9]. For each component the following is iterated:

$$\boldsymbol{v}_0 = \mathbf{K}\boldsymbol{u}_0 \tag{15}$$

$$\boldsymbol{u}_1 = \frac{\boldsymbol{v}_0}{\left(\boldsymbol{v}_0^{\mathrm{T}}\boldsymbol{v}_0\right)^{1/2}} \tag{16}$$

$$\boldsymbol{u}_0 = \boldsymbol{u}_1 \tag{17}$$

After each such step the difference between earlier iteration steps is measured and the iteration will halt when this difference is small enough. The following updating of the symmetric **K** matrix is performed after termination of the iteration:

$$\lambda = \boldsymbol{u}_0^{\mathrm{T}}\mathbf{K}\boldsymbol{u}_0 \tag{18}$$

$$\mathbf{K} = \mathbf{K} - \lambda\boldsymbol{u}_0\boldsymbol{u}_0^{\mathrm{T}} \tag{19}$$

where λ is the eigenvalue.

Two new vectors $\boldsymbol{a}$ and $\boldsymbol{f}$ are introduced which have the same dimensions as a row or a column of **W** and the following relations are assumed to be true:

$$\boldsymbol{u} = \mathbf{B}\boldsymbol{a} \tag{20}$$

$$\boldsymbol{v} = \mathbf{B}\boldsymbol{f} \tag{21}$$

Using this information it is possible to reformulate the eigenvalue decomposition iteration:

$$\mathbf{B}\boldsymbol{f}_0 = \mathbf{B}\mathbf{W}^{\mathrm{T}}\mathbf{W}\mathbf{B}^{\mathrm{T}}\mathbf{B}\boldsymbol{a}_0 \tag{22}$$

$$\mathbf{B}\boldsymbol{a}_1 = \frac{\mathbf{B}\boldsymbol{f}_0}{\left(\boldsymbol{f}_0^{\mathrm{T}}\mathbf{B}^{\mathrm{T}}\mathbf{B}\boldsymbol{f}_0\right)^{1/2}} \tag{23}$$

$$\mathbf{B}\boldsymbol{a}_0 = \mathbf{B}\boldsymbol{a}_1 \tag{24}$$

Setting $\boldsymbol{\Gamma} = \mathbf{B}^{\mathrm{T}}\mathbf{B}$ and observing that the premultiplication by **B** is redundant in the iteration, the PBM iteration will look like:

$$\boldsymbol{f}_0 = \mathbf{W}^{\mathrm{T}}\mathbf{W}\boldsymbol{\Gamma}\boldsymbol{a}_0 \tag{25}$$

$$\boldsymbol{a}_1 = \frac{\boldsymbol{f}_0}{\left(\boldsymbol{f}_0^{\mathrm{T}}\boldsymbol{\Gamma}\boldsymbol{f}_0\right)^{1/2}} \tag{26}$$

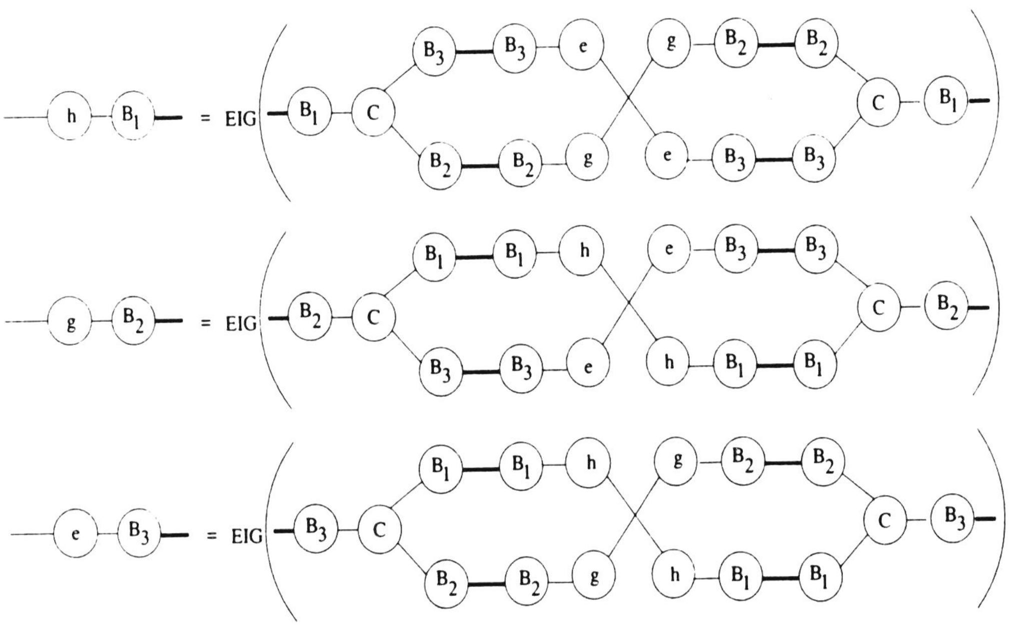


Fig. 6. The iteration steps in the ALS algorithm written in terms of the compression model.

$$\boldsymbol{a}_0 = \boldsymbol{a}_1 \tag{27}$$

The updating steps now become:

$$\lambda = \boldsymbol{a}_0^{\mathrm{T}} \boldsymbol{\Gamma} \mathbf{W}^{\mathrm{T}} \mathbf{W} \boldsymbol{\Gamma} \boldsymbol{a}_0 \tag{28}$$

$$\mathbf{W}^{\mathrm{T}}\mathbf{W} = \mathbf{W}^{\mathrm{T}}\mathbf{W} - \lambda \boldsymbol{a}_0 \boldsymbol{a}_0^{\mathrm{T}} \tag{29}$$

If the compression is perfect, the eigenvalue λ from the PBM method is identical to the original eigenvalue of the uncompressed representation.

3.2. PBM applied to the ALS iteration steps

In Fig. 5 the matrix to be diagonalized (i.e., the matrix inserted into the 'EIG' routine) in order to obtain a current estimate of the **G** loading matrix is shown. The computations for the other loadings are analogous. The upper part of Fig. 5 can be said to be a three-mode analogue to a covariance matrix (similar to the **L**, **M**, **N** matrices mentioned above). One way to understand the PBM method is to write out the compression model for each of the arrays in the equation. It is assumed that a loading matrix for a particular mode can be written as linear combinations of the basis matrix for that mode, i.e.,

$$\mathbf{H}_{\mathrm{h}} = \mathbf{B}_1 \mathbf{h} \tag{30}$$

$$\mathbf{G}_{\mathrm{h}} = \mathbf{B}_2 \mathbf{g} \tag{31}$$

$$\mathbf{E}_{\mathrm{h}} = \mathbf{B}_3 \mathbf{e} \tag{32}$$

where {**h**, **g**, **e**} are matrices *not* vectors. The lower part of Fig. 5 shows the upper part of the figure written out in terms of the compression models. Rectangles with dotted lines are drawn around important parts together with a small label symbol in the upper left of the rectangles to indicate which rectangles are comparable. The rectangle labeled 'a' shown in the upper part of the figure encircles the uncompressed $\underline{\mathbf{X}}$ array while in the lower part this array is written in terms of its compression model. The rectangles 'b' and 'c' contain the expressions for the loading matrices **H** and **E**, respectively. Since the basis matrix does not change from one iteration to another it is redundant in the expressions of the ALS algorithm. The redundant basis matrix is indicated with arrows in Fig. 5. Fig. 6 shows the rewriting of the equations for all the loading matrices. Thick lines are used to designate large modes (see Appendix for a short introduction to the diagram notation). By removing the redundant basis matrices on each side of the equation sign and using the PBM version of the eigenvalue decomposition algorithm described above (designated 'PBM-EIG' in the figure) the new algorithm depicted in Fig. 7 is obtained. No thick lines are 'exposed' or 'free' and thus there are no large modes in the calculation. Of course it must again be stressed that the Grammian matrices $\boldsymbol{\Gamma}_i = \mathbf{B}_i^{\mathrm{T}}\mathbf{B}_i$, $i \in [1,2,3]$ should be much smaller than the dual Grammian matrices $\mathbf{B}_i\mathbf{B}_i^{\mathrm{T}}$, $i \in [1,2,3]$. The results from this algorithm (the PBM loading matrices **h**, **g**, **e**) are similar to the PBM scores and loadings in standard (two-mode) principal component analysis as described in Part I of this article. They are not orthogonal, i.e., $\mathbf{h}^{\mathrm{T}}\mathbf{h} \neq \mathbf{I}$, $\mathbf{g}^{\mathrm{T}}\mathbf{g} \neq \mathbf{I}$ and $\mathbf{e}^{\mathrm{T}}\mathbf{e} \neq \mathbf{I}$. If premultiplication of the corresponding compression basis matrices is performed we get that $\mathbf{H}_{\mathrm{h}}^{\mathrm{T}}\mathbf{H}_{\mathrm{h}} = \mathbf{h}^{\mathrm{T}}\boldsymbol{\Gamma}_1\mathbf{h} = \mathbf{I}$, $\mathbf{G}_{\mathrm{h}}^{\mathrm{T}}\mathbf{G}_{\mathrm{h}} = \mathbf{g}^{\mathrm{T}}\boldsymbol{\Gamma}_2\mathbf{g} = \mathbf{I}$ and $\mathbf{E}_{\mathrm{h}}^{\mathrm{T}}\mathbf{E}_{\mathrm{h}} = \mathbf{e}^{\mathrm{T}}\boldsymbol{\Gamma}_3\mathbf{e} = \mathbf{I}$.

4. FLOPS estimations

All FLOPS equations were generated on the basis of the *flops* command in MATLAB. Part I of this article discusses in more detail how FLOPS can be calculated for different types of matrix operations. The FLOPS equations below are constructed on the assumption that the basis matrices are *sparse* [10], which significantly reduces the number of FLOPS required. Sparsity means that a matrix contains a large number of zero elements. See Part I of this article for more details.

The expression for the estimate of the approximate number of FLOPS required for the three-mode PCA is

$$\begin{aligned}\hat{F}_1 = I\{&2A(3NMK + NMA + NKA \\ &+ MKA + N^2A + M^2A + K^2A) \\ &+ A\left[q_a(2N^2 + 8N + 1) + 5N^2 + 3N\right] \\ &+ A\left[q_a(2M^2 + 8M + 1) + 5M^2 + 3M\right] \\ &+ A\left[q_a(2K^2 + 8K + 1) + 5K^2 + 3K\right]\}.\end{aligned} \tag{33}$$

where q_a is the number of iterations per eigenvector in the diagonalization routine. For the estimations performed here it is always assumed that the number of iterations is the same for each factor. This is of course not true, but is introduced to simplify the analysis of the algorithmic performance. I is the number of ALS iteration steps, A is the maximum number of factors and N, M, K are the dimensions of the $\underline{\mathbf{X}}$ array. Before defining the FLOPS formula for the PBM algorithm it is convenient to define the formula for estimated number of FLOPS required for the PBM–EIG routine described above:

$$\begin{aligned} J(n, d_1, g_1, A) &= 2n^2Nd_1^2 + 2n^3g_1 \\ &+ A\left[q_a(2n^2 + 2n^2g_1 + 3n + 1) \right. \\ &\left. + 2n^2g_1 + 5n^2 + 3n + 2n^3g_1\right] \end{aligned} \tag{34}$$

The approximate number of FLOPS for the PBM algorithm is

$$\begin{aligned} \hat{\mathcal{F}}_2 &= 2m^2Ag_2 + 2n^2Ag_2 + I(6knmA + 2kA^2n \\ &+ 2k^2A^2 + 2k^2Ag_3 + 2mA^2k + 2m^2A^2 \\ &+ 2m^2Ag_2 + 2nA^2m + 2n^2A^2 + 2n^2Ag_1) \\ &+ I[J(n, d_1, g_1, A) + J(m, d_2, g_2, A) \\ &+ J(k, d_3, g_3, A)] \end{aligned} \tag{35}$$

where n, m, k are the dimensions of the $\underline{\mathbf{C}}$ array; d_1, d_2, d_3 are the *densities* of the basis matrices $\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{B}_3$. The density is the number of non-zero elements in a matrix divided by the total number of elements; g_1, g_2, g_3 are the densities of the Grammian matrices $\mathbf{B}_1^T\mathbf{B}_1$, $\mathbf{B}_2^T\mathbf{B}_2$, $\mathbf{B}_3^T\mathbf{B}_3$.

In order to analyze the two FLOPS formulas some simplifications are introduced: $N = K = M$ and $n = k = m$ and $n = N/r$; r is a parameter for

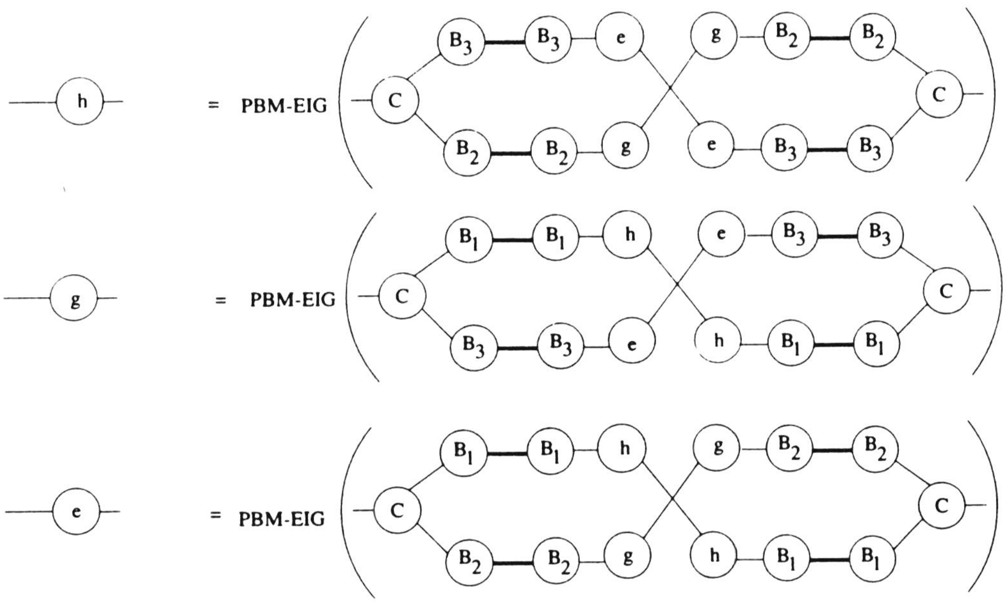


Fig. 7. Same as Fig. 6 except that the redundant basis matrix multiplications have been postponed to after the iteration has converged. Note that now we must use the PBM–EIG routine and not the usual eigenvalue algorithm for decomposition of each of the covariance like matrices.

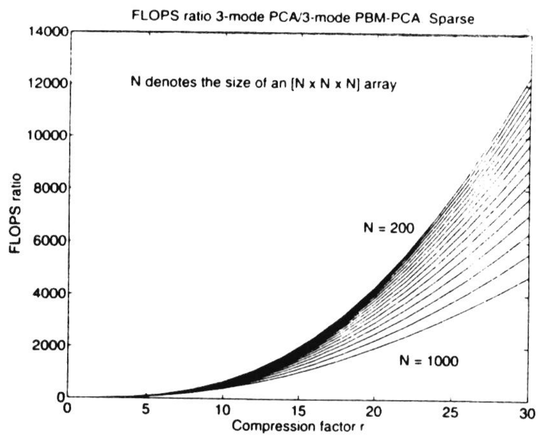


Fig. 8. This figure uses the FLOPS formulas which are discussed in the text and adequately estimates the FLOPS usage of the three-mode PBM–PCA algorithm. Here it is assumed that sparse technology can be used on the basis matrices. A density of 0.1 for every basis matrix is assumed and 0.2 for the corresponding Grammian matrices.

the compression. In a general case this is a vector where each element is a parameter for the different modes. For this particular case the same compression parameter has been selected for each mode. It is desirable that the observed FLOPS ratio

$$f = \frac{F_1}{\mathscr{F}_2} > 1 \tag{36}$$

should be as large as possible. Fig. 8 shows the result of a calculation based on the formulas for $\hat{F}_1$ and $\hat{\mathscr{F}}_2$ assuming the simplifications mentioned above. The following ranges were selected: $N \in [200, 1000]$, $r \in [2, 30]$, five factors are assumed to be extracted and $I = 5$, $q_a = 15$. The densities of the basis matrices were chosen to be 0.1 and the densities of their Grammians to be 0.2. This example corresponds to the one presented in Part I of this article. The highest values of f is obtained when r is large and N is small. If, e.g., $r = 20$, $N = 600$, i.e., from a $[600 \times 600 \times 600]$ array to a $[30 \times 30 \times 30]$ array the PBM algorithm will run approximately 3579 times faster.

5. Results

5.1. Data set 1

An artificial three-mode data set was used for these experiments. A large ratio between the compressed and original representation was constructed. Each basis matrix was given the same dimensions: $\mathrm{Dim}(\mathbf{B}_i) = [84 \times 7]$, $i \in \{1, 2, 3\}$, i.e., each mode in the original representation was more than ten times the size of the coefficient array $\underline{\mathbf{C}}$ with dimensions $\mathrm{Dim}(\underline{\mathbf{C}}) = [7 \times 7 \times 7]$. The number of factors extracted was three and number of ALS iterations was set to five. The number of iterations per factor was set to 15. The number of FLOPS consumed for the uncompressed representation was $F_1 = 118\,803\,325$, and $\mathscr{F}_2 = 486\,025$ FLOPS for the PBM algorithm. The FLOPS ratio is ca. 244. The theoretical estimates of the FLOPS consumption are $\hat{F}_1 = 118\,787\,625$ and $\hat{\mathscr{F}}_2 = 591\,050$ which gives a FLOPS ratio of ca. 200. In this case the formulas underestimate the effect of the PBM approach.

A much larger example could have been chosen but the standard three-mode PCA program could not handle arrays of size larger than $[90 \times 90 \times 90]$.

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 (version 4.0).

5.2. Data set 2

The data matrix was a three-dimensional electron density distribution of a molecule [8] with dimensions $\mathrm{Dim}(\underline{\mathbf{X}}) = [71 \times 38 \times 44]$. This data set was compressed using B-splines to a three mode array with dimensions $\mathrm{Dim}(\underline{\mathbf{C}}) = [30 \times 13 \times 17]$. The compression parameters for the different modes are approximately $\mathbf{r} = [2.4, 2.9, 2.6]$. In this data set it was unfortunately not possible to compress the original $\underline{\mathbf{X}}$ further without losing significant information. The number of FLOPS consumed for the original data set was $F_1 = 16\,829\,025$ and for the PBM algorithm $\mathscr{F}_2 = 2\,146\,461$ which corresponds to a FLOPS ratio of ca. 8. In order to compare the different results the number of iterations per factor was set to 15. The estimated FLOPS were $\hat{F}_1 = 16820415$ and $\hat{\mathcal{F}}_2 = 2074699$ which gives a FLOPS ratio of ca. 8 also. The results from the PBM algorithm were converted to the original domain and compared with the results of the three-mode PCA of the original data set, see Fig. 9. The results for this data set are very satisfactory. In addition to a relatively modest compression the three basis matrices were not sparse enough. The densities of the three basis matrices were approximately 0.21, 0.20, 0.12 and this is illustrated in Fig. 10.

6. Conclusion

The PBM method applied to three-mode PCA is efficient if the compression ratio is high and/or the basis matrices are sparse. The PBM should also be effective for N-mode PCA in general.

Appendix. Notation

A.1. Name of data objects

Data objects which are extensions of matrices have different names. Some names used are: tensors; multilinear forms; multidimensional arrays; N-arrays; N-mode arrays; N-order arrays; N-way arrays.

In this article we refer to such objects as arrays, N-arrays or N-mode arrays.

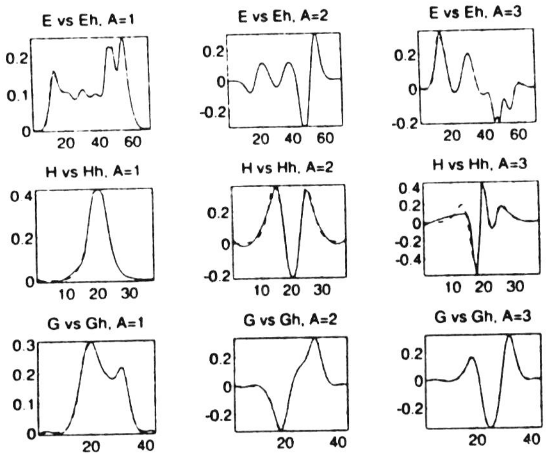


Fig. 9. Reconstruction of loadings for three factors of data set 2. It is in excellent agreement with the loadings from analysis of the uncompressed array. Dotted lines are the loadings from analysis of the uncompressed array.

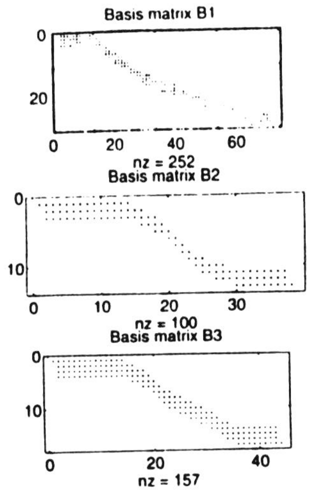


Fig. 10. The densities of the basis matrices used in data set 2.

A.2. The diagram notation

The diagram notation for use in chemometrics is fully described in Ref. [5] and only a short summary is given here. The diagram notation is a graphical visualization of the index topology in explicit summation notation. The diagrams have the appearances of *graphs* and use of graph terminology is therefore appropriate [11]. A diagram contains *nodes* and *edges*. An edge can be attached to one or several nodes. An edge connected to one node only is called *unconnected*. An edge connected to more than one node is called *connected*. It is possible to enable connection between more than two nodes by introducing a special *sum nexus* symbol, but this will not be presented in this article. For more details see Ref. [5]. A node with N unconnected edges is the diagram representation of a single N-mode array not connected to any other. A connected edge signifies summation of *one* index. For the continuous case a connected edge signifies an *integra-*

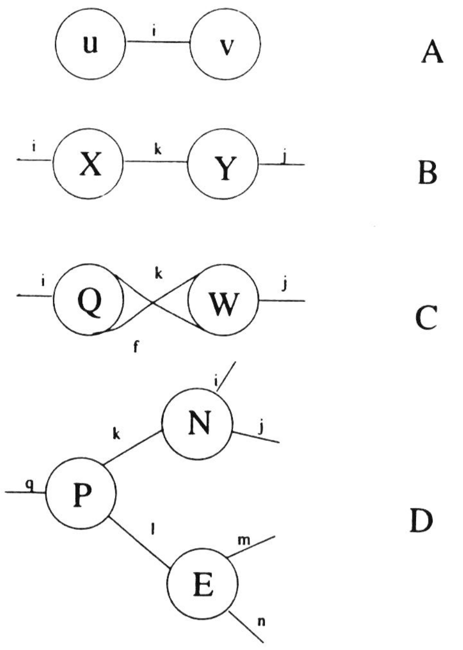


Fig. 11. Presentation of examples from matrix algebra in diagram notation. (A) Scalar product; (B) product of two matrices; (C) and (D) equations involving products of three-mode arrays.

tion. The total number of unconnected edges of an expression is the number of modes (or the *mode number*) of the result. If, e.g., two 3-arrays combine by summing one common index the mode number of the result is $3 + 3 - 2 = 4$. In the center of the node the name of the array is placed. In the vicinity of the edges the correct index names are sometimes written in order to clarify. The index names are written anti-clockwise from the first index. Sometimes it is necessary to indicate which edge signifies the *first* index. For this we use a small bar perpendicular to the edge and this mark is called the *first index pointer* or for short the *fip*.

Mirror reflections of the diagrams are not allowed since this will change the unique placement of the different indices. If reflections were allowed it would not be possible to discriminate between, e.g., the four-mode array x_{ijkl} and the four-mode array x_{ijkl} (assuming a fip has been used to indicate that i is the first index). The consequence of this is that, e.g., when two identical nodes which are connected through more than one edge, a crossing of edges will occur (as in Fig. 2). The crossings themselves have no

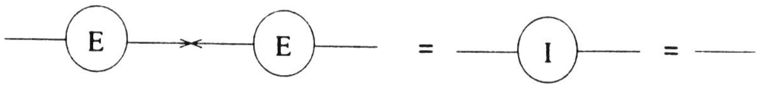


Fig. 12. For orthonormal matrices a shorthand notation is introduced which enables a faster manipulation of diagrams. Two arrows which meet head to head will result in an identity matrix. An identity matrix is drawn as a line.

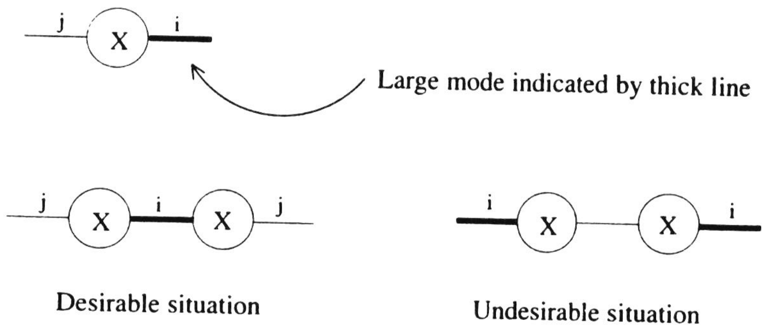


Fig. 13. For a SVD of a matrix of size [10 × 10000] it is desirable to avoid the large mode. The large mode has been signified by a thick line.

meaning and are just a result of the topological constraints imposed on the notation.

Fig. 11 shows a few examples of array diagram equations. The corresponding summation formulas for the diagram equations presented are:

(A) $\Sigma_i u_i v_i$. This is the standard inner product.

(B) $\Sigma_k x_{ik} y_{kj}$. This is standard matrix product.

(C) $\Sigma_k \Sigma_f q_{ifk} w_{kfj}$. An array product between two three-mode arrays. The result is a matrix because the number of unconnected edges is two.

(D) $\Sigma_k \Sigma_l p_{qlk} n_{ikj} e_{lnm}$. Here the result is a five-mode array since five free indices are seen.

Instead of using index names and fips for indicating the different modes of the arrays a shorthand notation has been constructed. There are especially two cases where a shorthand notation has been found useful: (i) matrices with orthonormal column vectors; (ii) modes of large size.

If $\mathbf{Q}$ is a matrix with orthonormal column vectors we have that $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ where $\mathbf{I}$ is the identity matrix. If $\mathbf{Q}$ is not square we have that $\mathbf{Q}\mathbf{Q}^T \neq \mathbf{I}$. Thus we need to discriminate between the two different modes of the matrix. *Arrows* have been chosen to distinguish between the modes. When two arrows meet head to head we have the case $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$. Fig. 12 illustrates the idea (here we see $\mathbf{E}^T\mathbf{E} = \mathbf{I}$). The identity matrix is for convenience drawn as a connected or as an unconnected edge with no matrix element attached.

In some problems it is necessary to avoid that large modes become unconnected edges. A simple example from SVD illustrates the main idea. If the dimension of $\mathbf{X}$ is $[10 \times 10\,000]$ and the object is to find the eigenvalues the fastest method is to calculate the eigenvalues of the covariance matrix $\mathbf{X}\mathbf{X}^T$ which has a dimension of $[10 \times 10]$. The rank of $\mathbf{X}$ cannot be larger than 10 which means that eigenvalue decomposition of $\mathbf{X}^T\mathbf{X}$ (dimension $[10\,000 \times 10\,000]$) would be a waste of resources; see Fig. 13 for illustration.

References

[1] I. Pelczer and S. Szalma, Multidimensional NMR and data processing, *Chemical Reviews*, 91 (1991) 1507-1524.

[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] 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.

[4] B.K. Alsberg and O.M. Kvalheim. Speed improvement of multivariate algorithms by the method of postponed basis matrix multiplication. Part I. Principal component analysis, *Chemometrics and Intelligent Laboratory Systems*, 24 (1994) 31-42.

[5] B.K. Alsberg, A diagram notation for *N*-mode array equations, *Psychometrika*, (1993) submitted.

[6] P.M. Kroonenberg, *Three Mode Principal Component Analysis*, DSWO Press, Leiden, 1983.

[7] L. Tucker, Some mathematical notes on three-mode factor analysis, *Psychometrika*, 31 (1966) 279-311.

[8] B.K. Alsberg and O.M. Kvalheim, Compression of three-mode data arrays by B-splines prior to three-mode principal component analysis (PCA), *Chemometrics and Intelligent Laboratory Systems*, 23 (1994) 29-38.

[9] G.H. Golub and C.F. van Loan, *Matrix Computations* (John Hopkins Series in the Mathematical Sciences, No. 3), The John Hopkins University Press, Baltimore, MD, 2nd edn., 1989.

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

[11] J.R. Wilson, *Introduction to graph theory*, Longman Scientific & Technical, Essex, 3rd edn., 1985.

Paper VI

A diagram notation for N-mode array equations

Bjørn K. Alsberg

Department of Chemistry
University of Bergen
Allegt. 41
N-5007 Bergen, Norway
e-mail: `alsberg@kj.uib.no`

August 24, 1994

Abstract

A diagram notation similar to that used in group theory is suggested to be a powerful way of representing and manipulating N-mode arrays equations in general. The diagram notation uses a bookkeeping of N-mode array indices such that the resulting equations have the appearance of graphs.

Keywords

N-arrays, N-mode arrays, multi-way arrays, higher order arrays, multilinear forms, notation,graphs

1 Introduction

Standard vector and matrix notation is very powerful. The typographic formulation captures without difficulty the different ways to manipulate the data objects. Typographically one may regard scalars as zero dimensional, vectors as one dimensional and matrices as two dimensional objects. A similar extension to N-mode arrays will not be successful. Three mode arrays can always be described on paper as drawings of cubes. It is a cumbersome notation and cannot be used efficiently in algebraic manipulations. It has, however, the advantage of rapidly conveying the basic idea of the three-mode data structure. For four- mode or higher mode arrays, the "drawing" notation quickly collapses. Useful notations therefore do not employ such approaches. The following criteria should in principle apply to any N-mode array notation:

- The notation must be unambiguous.
- The manipulation of the notation symbols and rules should be intuitive and simple.
- It must be useful for any number of modes.
- The notation must represent the underlying data structure in a logical way.
- Unnecessary features introduced by the notation should be kept at a minimum.
- Symmetries and patterns in the equations should be easy to discover.
- The notation must be scalable to larger problems.

Most notations available do not satisfy all the criteria above simultaneously. The following is a list of some notations used in connection with N-mode array equations:

- **Explicit summation notation**

  If three-mode singular value decomposition (SVD) is used as an example the Tucker 3 [Tucker (1964), Tucker (1966), Kroonenberg (1983)] model is written as follows in the explicit summation notation:

$$x_{ijk} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{ip} e_{jq} h_{kr} c_{pqr}. \tag{1}$$

  The type of formula presented at the right hand side of the equation is referred to as an array expression. One or several array elements are part of an expression.

  The explicit summation notation is consistent and much used. For large equations with many modes, however, it is difficult to see patterns and symmetries. Index assignment requires special attention.

- **Einstein index notation**

  This notation is much used in tensor analysis in physics[Arfken (1985)]. The idea is to suppress summation signs in the explicit summation notation. The following example shows the use of the Einstein index notation (at the right side of the arrow):

$$\sum_{\alpha} \sum_{\beta} p^{\alpha} q_{\alpha\beta} p^{\beta} \rightarrow p^{\alpha} q_{\alpha\beta} p^{\beta}. \tag{2}$$

  Greek letters are used to denote the indices to be summed over. Covariance and contravariance is indicated by putting the indices as super-

or subscripts. If an index appears on one side of the equation, once as a superscript and once as a subscript sum over that index is implied [Blinn (1992)a, Arfken (1985)]. Even with this improvement the equations rapidly get complicated when N-mode equations are considered. Index assignment and handling is still a problem.

- **Kronecker notation**

  This notation represents N-mode arrays by rearranging N-arrays into matrices. If a 3-array $\underline{\mathbf{X}}$ [1] has dimensions $[I \times J \times K]$ it can be rearranged into a $[IJ \times K]([JI \times K])$ or $[I \times KJ]([I \times JK])$ or $[KI \times J]([IK \times J])$ matrix. Such a rearrangement is referred to as an unfolding. The following equation has the same meaning as equation 1:

  $$\mathbf{X} = \mathbf{G}\ \mathbf{C}(\mathbf{E}^T \bigotimes \mathbf{H}^T)$$

  where $\mathbf{X}$ and $\mathbf{C}$ are unfolded representations of the 3-array. The $\otimes$ symbol signifies the Kronecker product[Magnus and Neudecker (1988)]. The right-oriented Kronecker product between two matrices $\mathbf{A}$ and $\mathbf{B}$ of dimensions $[n \times m]$ and $[k \times j]$ is

  $$\mathbf{M} = \mathbf{A} \bigotimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1m}\mathbf{B} \\ \cdot & & \cdot \\ \cdot & & \cdot \\ \cdot & & \cdot \\ a_{n1}\mathbf{B} & \cdots & a_{nm}\mathbf{B} \end{bmatrix} \tag{3}$$

  where $\mathbf{M}$ has dimensions $[nk \times mj]$. It is an advantage that mathematicians use this notation and that the algebraic properties of the notation

[1]Underlined upper-case boldface characters denote N-mode arrays, $N > 2$. Upper-case boldface characters denote matrices. Boldface lower-case characters denote vectors. Upper- and lower-case characters denote scalars.

have been well studied. Yet the unfolding operations implied in the matrix representations of N-arrays, give rise to unnecessary complexity which further reduce the visualization and intuitive way of thinking about such arrays. For three-mode problems the idea of unfolding is manageable but for fourth or higher modes this is difficult.

- **"Stacking" notation**

  This notation has been applied to three-mode data arrays by indicating a third mode using unconventional placement of matrices[Geladi (1989)]. The following equation has the same meaning as equation 1:

$$\underline{\mathbf{X}} = \mathbf{G}^T \overset{\mathbf{E}}{\underline{\mathbf{C}}} \mathbf{H} \tag{4}$$

  where $\underline{\mathbf{X}}$ and $\underline{\mathbf{C}}$ are 3-arrays. The rest of the symbols represents ordinary matrices. This notation has intuitive aspects but cannot be used for arrays with more than three modes and is therefore not interesting as a general tool.

Another way of representing N-mode array equations is the use of diagrams [Blinn (1992)b], which is the main concern of the present paper. In this article a diagram technique is presented which is inspired by (but not equal to) Feynman diagrams. Feynman diagrams have been used in many-body problems and include several additional assumptions related to quantum mechanics [Mattuck (1976)] which are not necessary for solving problems in psychometrics and chemometrics. An example of a Feynman diagram is shown in Figure 1 which shows the photon (γ) scattering into to an electron (e^-) and a positron (e^+) pair.

The type of diagrams which is of interest to higher-mode problems have beed adapted from the field of group theory [Stedman (1990)]. The diagrams are used to visualize *and* manipulate the index structure and topology of higher-mode array equations. The details of the suggested diagram notation are discussed below.

2 The diagram notation

2.1 Basic rules and properties

The diagram notation is based on a mapping from the explicit summation notation into structures which have appearances of graphs [Wilson (1985)]. Some graph terminology is therefore appropriate to use here.

The following rules and properties are associated with the diagram equation notation:

1. A diagram contains nodes and edges. An edge is drawn as a line attached to either one or two nodes. An edge attached to two nodes is said to be connected and an edge attached to one node only is said to be unconnected.

2. A node corresponds to an N-mode array element. For continuous cases it can signify a function element. The edges signifies the indices of the array element.

   A scalar has no unconnected edges, see Figure 2, mode zero. A vector element is expressed in the diagram notation as in Figure 2, mode 1. A matrix element has two indices associated with it (Figure 2, mode 2) and the extension to 3- and 4-arrays is logical (Figure 2, mode 3 and 4).

3. A node is drawn as a circle with the name of the array inside. In order to clarify, index names may be introduced in the vicinity of the edges. Often it is necessary to indicate the first index or mode so that it is possible to distinguish between e.g. x_{ij} and x_{ji}. In such cases the first index is marked with a small bar perpendicular to the edge. This small bar is called the <u>first index pointer</u> or just the <u>fip</u>. The remaining modes and their indices are labeled <u>anti-clockwise</u> with respect to the first mode. The fip together with the anti-clockwise orientation uniquely identifies the indices associated with a node.

4. A connected edge is defined to be a <u>summation</u> over <u>one</u> index. For continuous cases an edge is defined to be an <u>integration</u>. Figure 3 gives an overview of the main idea.

   A few examples from vector and matrix algebra illustrate the use of connected edges. The scalar product $x = \sum_{i=1}^{n} v_i w_i$ is written in diagram notation as in Figure 4A. The length of vector **v** written as a diagram is shown in Figure 4B. Figure 4C shows the matrix product:

$$d_{ji} = \sum_{k=1}^{n} a_{jk} b_{ki} \qquad (5)$$

   in the new notation.

   The SVD equation for 2-mode data arrays

$$x_{ji} = \sum_{k} \sum_{l} u_{jk} s_{kl} p_{li} \qquad (6)$$

   is shown in Figure 4D.

5. Only the topology of a diagram is important. Rotations, translations and scaling of the nodes and edges are allowed, but mirror reflections are not, see Figure 5. A consequence of not allowing mirror reflections of diagrams is that connecting more than two indices between arrays will result in crossing of edges. The crossings themselves have no meaning.

   Prohibition of mirror reflections for N-mode array equations ($N \geq 1$) originates from the requirement of unique identification of indices (edges).

6. Two unconnected edges can combine to form a connected edge if the size of that mode is the same for both edges. For matrix $\mathbf{X}$ with dimensions $[n \times m]$ and matrix $\mathbf{Y}$ of dimensions $[k \times r]$ it is possible to calculate $\mathbf{XY}$ if $m = k$. The same also applies for arrays of higher modes. Edges satisfying this criterion are said to be *comparable*.

7. Each part of a summation of diagram expressions must contain the same number of comparable unconnected edges. Figure 6 illustrate this point by an example where four-mode arrays are added.

8. Mathematical operations not directly associated with the index topology may also be present in the diagram notation. Here are some examples:

   - The inverse is drawn as $(< \textit{diagram} >)^{-1}$.
   - *eig* denotes the *a* largest eigenvectors of a matrix.
   - $*$ is needed to signify complex conjugation.
   - Operations performed on each array element is denoted by indicating the operation on the array name placed inside the node.

Figure 7 illustrate the concept for cosine , logarithm and exponential of array elements.

9. The number of unconnected edges of a node or an expression, corresponds to the number of modes of the N-array. This number is hereafter referred to as the mode number. This number is the same as the degree of a node in graph theory. For an N-mode array the mode number is N. An operator named $\mathcal{M}$ applied to an array $\underline{\mathbf{X}}$ returns the mode number of $\underline{\mathbf{X}}$. A single 3-array will therefore have three unconnected edges. An array expression consisting of two 3-arrays connected with one edge has a mode number of four, i.e. $3 + 3 - 2 = 4$.

   The following property holds for all diagrams:

   Property 1 *Let a diagram expression of array $\underline{\mathbf{Z}}$ consist of P nodes (arrays elements) N_i, $i \in [1, 2, \cdots P]$. The the total number of unconnected edges or the mode number of $\underline{\mathbf{Z}}$ given the number of connected edges k, is*

$$\mathcal{M}(\underline{\mathbf{Z}}) = \sum_{i}^{P} \mathcal{M}(N_i) - 2k. \tag{7}$$

2.2 Necessary extensions of the notation

2.2.1 The sum nexus

The diagram notation rules stated so far cannot handle cases when the number of array elements involved in a summation of an index is different from two. This is due to the fact that an edge or a line can only connect to two nodes. The problem is solved by using more edges which are connected to a new type of node. This new array contains zeros and ones only. The effect is that new summations are added to the equation. The array used is one version of the N-mode Kronecker delta ($\delta_{ijk\cdots}$). In diagram notation it is depicted as a filled rectangle. This filled rectangle is also referred to as the sum nexus symbol, see Figure 8A and B. The corresponding explicit summation formulas for this figure are:

$$g_j = \sum_{i=1}^{N} a_i b_i c_i x_{ij} \qquad (A) \tag{8}$$

$$p = \sum_{i} h_{iiii} \qquad (B) \tag{9}$$

When a sum nexus is seen, this means that one and only one index is summed over for all the array elements connected to the sum nexus.

Equation 8 written out in vector and matrix algebra is:

$$\mathbf{g} = (\mathbf{a} \odot \mathbf{b} \odot \mathbf{c})^T \mathbf{X} \tag{10}$$

where $\odot$ is the Hadamard product, i.e. elementwise multiplication. Writing this equation without the Hadamard operator is somewhat more difficult even in standard notation:

$$\mathbf{g} = (\mathrm{diag}(\mathrm{diag}(\mathbf{a}\mathbf{b}^T)\mathbf{c}^T))^T \mathbf{X} \tag{11}$$

where diag returns the diagonal of a square matrix as a column vector. This example demonstrates that additional features of the standard matrix and vector notation is needed also. The Cayley vector/matrix multiplication is not sufficient for all types of operations.

In the following example two sum nexus symbols are necessary since there are two different indices which are summed over:

$$z_{i_3 i_4} = \sum_{i_1} \sum_{i_2} v_{i_1 i_2} w_{i_1 i_2 i_4} x_{i_1 i_2} y_{i_1 i_2 i_3}, \tag{12}$$

see Figure 9 for a diagram illustration.

2.2.2 N-mode Kronecker delta definitions

The two-mode Kronecker delta is defined as:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

The δ_{ij} is just the identity matrix. It is here defined a Kronecker delta for one-mode arrays also: $\delta_i = 1, \quad \forall \; i$.

In this paper only two types of N-mode Kronecker deltas are used:

- Type I:

  This Kronecker delta is used for summing of an index involving $N \neq 2$ nodes and displaying the trace of an N-array; here symbolized as the array $\underline{\Delta}_N^{(I)}$.

- Type II:

  This Kronecker delta is used in array differentiation; here symbolized as the array $\underline{\Delta}_N^{(II)}$. In the diagram notation $\underline{\Delta}_N^{(II)}$ is depicted as a filled circle of size smaller than the other array nodes. The diagram symbol is also referred to as the <u>differentiation symbol</u>.

The N-mode Type I Kronecker delta ($\underline{\Delta}_N^{(I)}$) is defined as

$$\delta^{(I)}_{i_1 i_2 \cdots i_N} = \begin{cases} 1 & \text{if } i_1 = i_2 = \cdots = i_N \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

The $\underline{\Delta}_N^{(II)}$ array used in differentiation consists of two sets of indices, $S_1 = \{i_1, i_2, \cdots, i_n\}$ and $S_2 = \{o_1, o_2, \cdots, o_n\}$. S_1 is referred to as the set of *input* indices and S_2 is referred to as the set of *output* indices.

The N-mode Kronecker delta Type II ($\underline{\Delta}_N^{(II)}$) is defined as:

$$\delta^{(II)}_{i_1 i_2 \cdots i_n o_1 o_2 \cdots o_n} = \begin{cases} 1 & \text{if } i_j = o_j \ \forall j \in \{1, 2, \cdots, n\} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

Type I arrays have much stronger requirements for when an element becomes non-zero than Type II. In an N-mode $[q \times q \times \cdots \times q]$ Type I array there are q non-zero elements. In an M-mode ($M \in \{2, 4, 6, \cdots\}$) $[q \times q \times \cdots \times q]$ Type II array there are on the other hand $q^{M/2}$ non-zero elements.

Note that $\underline{\Delta}_2^{(I)} = \underline{\Delta}_2^{(II)}$ which means that both are equal to the identity matrix and it is arbitrary which Kronecker delta is used. The identity array is just written as $\underline{\Delta}_2$.

Property 2 *All δ arrays with an odd number of edges belong to the Type I Kronecker delta. Type I can contain an even number of edges also. Type II Kronecker deltas contain an even number of edges only.*

2.2.3 Chaining

Sometimes array elements are included in expressions that contain indices not summed over or shared by other array elements. Such nodes cannot be connected to other nodes through edges or sum nexus symbols. Since no nodes in a diagram expression are allowed to be isolated, some other strategy must be employed. Chaining is here defined to be nodes that touches other nodes but do not connect to them through summation. The perimeters of the nodes coincide in at least one point. Such nodes can in principle be chained to any node in a diagram expression. No configuration is preferred to another. In spite of this it may be visually convenient to collect all such nodes at one place in a linear string of nodes. Chaining of nodes is used to express outer products and treated in more detail below.

2.3 Some properties of Kronecker deltas

The Type I Kronceker delta can also be written as:

$$\delta^{(I)}_{i_1 i_2 \cdots i_n} = \sum_p \delta_{i_1 p} \delta_{i_2 p} \cdots \delta_{i_n p}. \tag{16}$$

"Linking" the two-mode Kronecker products by summing over the variable p, the different $\delta_{i_j p}$ matrices become dependent and produces a non-zero element only when $i_1 = i_2 = \cdots i_n = p$.

Type II Kronecker deltas can be written as a product (a chain) of two-mode Kronecker deltas:

$$\delta^{(II)}_{i_1 i_2 \cdots i_n o_1 o_2 \cdots o_n} = \prod_{j=1}^{n} \delta_{i_j o_j}. \tag{17}$$

What is produced when several Type I Kronecker deltas are multiplied together? A multiplication by two Type I Kronecker deltas can be written as:

$$\sum_{i_1 i_2 \cdots i_q} \delta^{(I)}_{i_1 i_2 \cdots i_q i_{q+1} i_{q+2} \cdots i_N} \delta^{(I)}_{i_1 i_2 \cdots i_q \cdots j_1 j_2 \cdots j_M} = \delta^{(I)}_{i_{q+1} i_{q+2} \cdots i_N j_1 j_2 \cdots j_M} \tag{18}$$

Here it is assumed that the first q indices are common in the two arrays which are being summed over. There is only one term in the sum which gets a non-zero element and that is when $i_1 = i_2 = \cdots = i_q = i_{q+1} = i_{q+2} = \cdots i_N = j_1 = j_2 = \cdots = j_M$. This is the same as the definition of a Type I array, thus the result of the multiplication of two Type I array is another

Type I array. This fact can be applied pairwise to any multiplication of Type I arrays.

When Type I and Type II Kronecker deltas are multiplied together ($N > 2$), the result is a Type I Kronecker delta if there are no free corresponding input and output indices in the Type II array. This will be illustrated with an example. Here is an example where the result is not a Type I array:

$$\sum_{i_2}\sum_{i_3} \delta^{(I)}_{j_1 j_2 i_2 i_3} \delta^{(II)}_{i_1 i_2 i_3 o_1 o_2 o_3} = \sum_{i_2}\sum_{i_3} \delta^{(I)}_{j_1 j_2 i_2 i_3} \delta_{i_2 o_2} \delta_{i_3 o_3} \delta_{i_1 o_1} \tag{19}$$

The output from this is an array with the indices $i_1 j_1 j_2 o_1 o_2 o_3$. If this is to be a Type I array then it is required that the only non-zero values are located in elements satisfying $i_1 = j_1 = j_2 = o_1 = o_2 = o_3$. It is easy to see that there will also be non-zero values in elements when $i_1 = o_1 = p$ and $j_1 = j_2 = o_2 = o_3 = q$ and $p \neq q$. If, however, equation 19 sums over i_1 and/or o_1 in addition to i_2 and i_3, the result becomes a Type I array.

2.4 Marking of modes in diagram notation

The placement of indices in the vicinity of the edges together with the fip makes it possible to identify the correct modes. However, in complex expressions it may be awkward to write the individual index names for all edges so shorthand notations may be of help. Marking of modes will also be of use to avoid ambiguity in multiplication of array expressions.

There are especially three cases where marking of modes have been found useful:

- Matrices with orthonormal column vectors.

  If $\mathbf{E}$ is suborthonormal, i.e. contains orthonormal column vectors, we have that $\mathbf{E}^T\mathbf{E} = \mathbf{I}$ where $\mathbf{I}$ is the identity matrix. If $\mathbf{E}$ is not square we have that $\mathbf{E}\mathbf{E}^T \neq \mathbf{I}$. Thus we need to discriminate between the two different modes of the matrix. Arrows have been chosen to distinguish between the modes. When two arrows meet head to head we have the case $\mathbf{E}^T\mathbf{E} = \mathbf{I}$. Figure 10 illustrate the idea. The identity matrix is for convenience drawn as a connected or as an unconnected edge with no matrix element attached.

- Modes of large size.

  In some problems it is necessary to avoid that large modes become unconnected edges. A simple example illustrates the main idea. If the dimensions of $\mathbf{X}$ are $[10 \times 10000]$ and the objective is to find the eigenvalues, the fastest method is to calculate the eigenvalues of the grammian matrix $\mathbf{X}\mathbf{X}^T$ which has dimensions of $[10 \times 10]$. The rank of $\mathbf{X}$ cannot be larger than 10 which means that eigenvalue decomposition of $\mathbf{X}^T\mathbf{X}$ (dimensions $[10000 \times 10000]$) would be a waste of resources; see Figure 11 for illustration.

- Multiplication of expressions

  When an array expression is to be multiplied with another we need to specify which index is to be summed over, e.g.

$$\sum_i \sum_g a_{efg} p_{ghi} (c_{ijk} + \sum_m d_{im} q_{mjk}) =$$
$$\sum_g \sum_i a_{efg} p_{ghi} c_{ijk} + \sum_g \sum_i \sum_m a_{efg} p_{ghi} d_{im} q_{mjk} \qquad (20)$$

where it is necessary to specify to sum over index i. Equation 20 is shown in Figure 12 where the i index is specified by drawing an additional edge parallel to the existing unconnected edge. It is of course not necessary to do this if only one unconnected edge exists for all the terms. If more than one index needs to be summed over, additional edges are added to all the necessary unconnected edges. This marking is just temporary, i.e. it is not necessary to carry it along the diagram expressions during all the steps in a deduction. It is only of interest prior to a multiplication as shown in equation 20 in order to avoid ambiguities in which indices are to be summed.

It should be noted that the use of arrows in Feynman diagrams signify covariant and contravariant directions depending whether the arrow point toward or away from the center of the node. For most purposes this is not necessary for problems in psychometrics and chemometrics but if such is the case, other notational solutions should be used to signify directions of suborthonormal matrices.

2.5 Outer products

2.5.1 Vector outer products

An N-mode outer product of N vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}, \cdots, \mathbf{q}$ produces an N-mode array $\underline{\mathbf{X}}$. It is here defined as:

$$x_{i_1 i_2 i_3 \cdots i_n} = a_{i_1} b_{i_2} c_{i_3} \cdots q_{i_n} \tag{21}$$

It is common in the psychometric and chemometric literature to write this as:

$$\underline{\mathbf{X}} = \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{c} \otimes \cdots \otimes \mathbf{q}. \tag{22}$$

The outer product is expressed through chaining of nodes. Figure 13 shows the principle of node-chaining for the outer product between two vectors, e.g. $x_{ij} = v_i w_j$. The outer product $x_{ijk} = a_i b_j c_k$ is shown in Figure 14.

It may be an advantage to decide on an ordering of the nodes in a chain. In the example in Figure 14 the **a** vector contains the fip and **b** contains index j which is the next index after i (the fip). Thus the array element b_j must be placed <u>after</u> the vector element a_i. c_k must be placed after b_j.

The following is an example where outer products and summing over indices takes place in the same expression:

$$\sum_p \sum_q a_i b_j c_k x_{prt} y_{qph} u_p. \tag{23}$$

The corresponding diagram is shown in Figure 15 where other possible placements of the chaining nodes are indicated. For simplicity the outer product nodes are kept together . They could in principle be chained to any other node in the expression.

The use of outer products is of importance for e.g. the PARAFAC model where the core array is restricted to be an identity array (a $\underline{\Delta}_N^{(I)}$ array). The PARAFAC model for the three-mode case is shown using diagrams in Figure 16.

2.5.2 Array outer products

An array product between different arrays $\underline{\mathbf{A}}, \underline{\mathbf{B}}, \cdots, \underline{\mathbf{Q}}$ which may or may not have the same mode number is defined as:

$$x_{a_1 a_2 \cdots a_M b_1 b_2 \cdots b_N \cdots q_1 q_2 \cdots q_P} = a_{a_1 a_2 \cdots a_M} b_{b_1 b_2 \cdots b_N} \cdots q_{q_1 q_2 \cdots q_P} \tag{24}$$

It is rather straightforward to extend the node-chaining to encompass N-mode array outer products. Each node which is chained to another now contain more than one edge. Figure 17 illustrate the following array outer product:

$$x_{klimjphtg} = b_{kli}a_{mj}r_p g_{htg}. \tag{25}$$

2.6 Translation between explicit summation and diagram notations

It is possible to set up rules for translating from explicit summation notation to diagrams and vice versa. Such a translation is <u>not</u> necessary for doing any manipulations with diagrams, but is included here for cases when an investigator for whatever reasons need to work with both notations. The steps for translating from explicit summation notation to diagram notation are:

1. A node is drawn for each array element. Unique lower-case letters are inserted as array element identification into the center of the nodes.
2. Edges corresponding to each index is added to every node. Whether the edge is connected or not is determined until step no. 3. Proper names of indices and fips are added if necessary. The index names are placed anti-clockwise with respect to the fip.
3. For each index which is to be summed (a sum-index) do the following:
    - Find all array elements containing the current sum-index.

- If one node only contains the sum-index, draw a sum nexus (a 1-mode Kronecker delta δ_i) at the end of the edge corresponding to the correct index, see Figure 18 ($\sum_i x_{ij\cdot}$).
- If two nodes contain the sum-index, draw a connected edge between the two nodes.
- If three or more nodes contain the sum-index, draw connected edges from all the nodes to a sum nexus symbol.

4. Array elements containing <u>no</u> indices summed are collected in a chain of nodes and attached to one of the other nodes (which contain indices being summed over).

The steps for translating from diagram notation to explicit summation notation are:

1. Identify all nodes and their names.
2. Count the number of edges on each node and identify each index. This will give each array element the right number of subscripts (or superscripts if contravariant tensors are considered) and subscript/superscript names.
3. All elements in a single diagram expression are multiplied (of course this is not performed for elements in different diagram expressions that are summed).
4. Place the number of summation signs in front of the array element names which is the same as connected edges in the diagram. The placement of an edge will uniquely define which of indices are being summed over.

5. Identify which array elements are connected to a particular sum nexus. These arrays will share one common index that is summed over.

6. Nodes in a chain are inserted into the explicit summation expression with indices which are not part of any sum.

3 Manipulation of diagrams

3.1 Solving equations

Diagram equations can be solved and manipulated very much the same way as for matrix equations. Those manipulations that are allowed for the explicit summation notation are also valid for the diagram notation. Figure 19 is an example where a three-mode array is equal to the sum of two other three-mode arrays. The **P** matrix indicated and multiplied with the three-mode array $\underline{\mathbf{R}}$ is assumed to be invertible. The first step is to subtract the 3-array $\underline{\mathbf{G}}$ from both sides of the equation. Secondly, the equation is multiplied by the inverse matrix $\mathbf{P}^{-1}$ on both sides. At the bottom of Figure 19 the $\mathbf{P}^{-1}$ has not been multiplied with any of the expressions on the left side within the parentheses. Note the use of double edges which emphasizes which index is to be summed over. In Figure 20 $\mathbf{P}^{-1}$ has been multiplied with both expressions at the left side in the equation.

3.2 Differentiation

3.2.1 Differentiation with respect to vectors

Let F be a linear M-mode array function which maps a vector $\mathbf{x}$ into an M-mode array:

$$F: \mathcal{R}^q \rightarrow \mathcal{R}^{p_1 \times p_2 \times \cdots \times p_M}. \tag{26}$$

Let $\mathbf{x}$ be a vector of q independent variables. Assuming that vector $\mathbf{x}$ is multiplied along r of the N modes of the N-mode array $a_{i_1 i_2 \cdots i_N}$ of coefficients, a general expression for the differentiation can be stated:

$$\begin{aligned}
F(x_q; a_{i_1 i_2 \cdots i_N}) &= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_r} x_{i_1} x_{i_2} \cdots x_{i_r} a_{i_1 i_2 \cdots i_N} \\
\frac{\partial F(x_q; a_{i_1 i_2 \cdots i_N})}{\partial x_k} &= \\
& \sum_{i_1}\sum_{i_2}\cdots\sum_{i_r} \left(\frac{\partial x_{i_1}}{\partial x_k}\right) x_{i_2} \cdots x_{i_r} a_{i_1 i_2 \cdots i_N} \\
&+ \sum_{i_1}\sum_{i_2}\cdots\sum_{i_r} x_{i_1} \left(\frac{\partial x_{i_2}}{\partial x_k}\right) \cdots x_{i_r} a_{i_1 i_2 \cdots i_N} \\
&+ \cdots \sum_{i_1}\sum_{i_2}\cdots\sum_{i_r} x_{i_1} x_{i_2} \cdots x_{i_{r-1}} \left(\frac{\partial x_{i_r}}{\partial x_k}\right) a_{i_1 i_2 \cdots i_N}.
\end{aligned} \tag{27}$$

The following relation is used:

$$\left(\frac{\partial x_j}{\partial x_k}\right) = \delta_{jk} \tag{28}$$

which simplifies equation 27 :

$$\begin{aligned}
\frac{\partial F(x_q; a_{i_1 i_2 \cdots i_N})}{\partial x_k} &= \\
& \sum_{i_2}\sum_{i_3}\cdots\sum_{i_r} x_{i_2} x_{i_3} \cdots x_{i_r} a_{i_1 i_2 \cdots i_N} \\
&+ \sum_{i_1}\sum_{i_3}\cdots\sum_{i_r} x_{i_1} x_{i_3} \cdots x_{i_r} a_{i_1 i_2 \cdots i_N} \\
&+ \cdots \sum_{i_1}\sum_{i_2}\cdots\sum_{i_{r-1}} x_{i_1} x_{i_2} \cdots x_{i_{r-1}} a_{i_1 i_2 \cdots i_N}.
\end{aligned} \tag{29}$$

When translating this into diagram notation it may be instructive to substitute $\left(\frac{\partial x_j}{\partial x_k}\right)$ with δ_{jk} (the identity matrix). δ_{jk} is treated as another matrix element in the expressions. In Figure 21 the principle is demonstrated for a four mode array multiplied by three copies of the vector $\mathbf{x}$ to give a one mode array (vector) as the result. The corresponding summation expressions for this figure are:

$$\begin{aligned} F(x_q; a_{lijk}) &= \sum_i \sum_j \sum_k x_i x_j x_k a_{lijk} \\ \frac{\partial F(x_q; a_{lijk})}{\partial x_p} &= \sum_i \sum_j \sum_k (\delta_{ip})\, x_j x_k a_{lijk} + \sum_i \sum_j \sum_k x_i\, (\delta_{jp})\, x_k a_{lijk} \\ &+ \sum_i \sum_j \sum_k x_i x_j\, (\delta_{kp})\, a_{lijk} \\ &= \sum_j \sum_k x_j x_k a_{ijkl} + \sum_i \sum_k x_i x_k a_{ijkl} \\ &+ \sum_i \sum_j x_i x_j a_{lijk} \end{aligned} \tag{30}$$

In the *Examples* section, the deduction of the least squares solution is demonstrated using diagram differentiation of a scalar error term with respect to a vector.

3.2.2 Differentiation with respect to N-arrays

Let G be a linear M-mode array function which maps an N-mode array $\underline{\mathbf{X}}$ into an M-mode array:

$$G \colon \mathcal{R}^{q_1 \times q_2 \times \cdots \times q_N} \longrightarrow \mathcal{R}^{p_1 \times p_2 \times \cdots \times p_M}. \tag{31}$$

Let $\underline{\mathbf{X}}$ be an N-mode array containing $[q_1 \times q_2 \times \cdots \times q_N]$ independent variables. The following property holds:

Property 3 *If $\underline{\mathbf{D}}$ is the result of differentiating $G(\underline{\mathbf{X}})$ with respect to $\underline{\mathbf{X}}$, then mode number of $\underline{\mathbf{D}}$ obeys: $\mathcal{M}(\underline{\mathbf{D}}) = \mathcal{M}(G(\underline{\mathbf{X}})) + \mathcal{M}(\underline{\mathbf{X}})$.*

Since each element in $\underline{\mathbf{D}}$ is addressed by both the indices from $G(\underline{\mathbf{X}})$ and $\underline{\mathbf{X}}$, the number of indices in $\underline{\mathbf{D}}$ must be the sum of indices in $G(\underline{\mathbf{X}})$ and $\underline{\mathbf{X}}$.

The Type II Kronecker delta array is involved in all differentiation where $\mathcal{M}(\underline{\mathbf{X}}) \geq 1$. When $\mathcal{M}(\underline{\mathbf{X}}) = 1$ both the Type I and Type II Kronecker delta can be used. We have that:

$$\frac{\partial x_{i_1 i_2 \cdots i_M}}{\partial x_{j_1 j_2 \cdots j_M}} = \delta^{(II)}_{i_1 i_2 \cdots i_M j_1 j_2 \cdots j_M}, \quad M \geq 1. \tag{32}$$

The general steps for performing array differentiation are:

- Substitute $\underline{\mathbf{X}}$ with a $2\mathcal{M}(\underline{\mathbf{X}})$ mode $\underline{\Delta}^{(II)}$ array. $\mathcal{M}(\underline{\mathbf{X}})$ of the indices in the differentiation symbol must be free (unconnected). More than $\mathcal{M}(\underline{\mathbf{X}})$ indices can be free in the result if $\underline{\mathbf{X}}$ before the differentiation contained free indices also.

  The connected edges of the differentiation symbol are by definition the input indices of the $\underline{\Delta}^{(II)}_N$ array. If the fip of $\underline{\mathbf{X}}$ is to be "inherited" by the differentiation symbol, it must be such that going anti-clockwise from the fip, the N first edges are the input indices.

- If $\underline{\mathbf{X}}$ occurs P times in an expression, the result of the differentiation results in P expressions where each has <u>one</u> $\underline{\Delta}^{(II)}$ array substituted for $\underline{\mathbf{X}}$. This is a result of the chain-rule in differentiation.

In the following example differentiation with respect to the matrix $\mathbf{X}$ is performed:

$$\begin{aligned} G(x_{ij}) &= \sum_i \sum_k x_{ip} x_{kq} a_{ijk} \\ \frac{\partial G(x_{ij})}{\partial x_{st}} &= \sum_i \sum_k \left(\delta^{(II)}_{ipst}\right) x_{kq} a_{ijk} + \sum_i \sum_k x_{ip} \left(\delta^{(II)}_{kqst}\right) a_{ijk} \end{aligned} \tag{33}$$

This equation is illustrated in diagram notation in Figure 22.

As shown in the previous section discussing the properties of the Kronecker deltas, the Type II Kronecker delta can be written as an array outer product of identity matrices. It is therefore possible to substitute the differentiation symbol by a chain of identity matrices. This is illustrated in Figure 23 for the six-mode differentiation symbol. This relation is useful when determining the result of connecting the Kronecker Type II array to other arrays. The same relation is used to demonstrate that differentiation of an expression with respect to the matrix **X** which is connected to two arrays, results in a chaining of the two arrays connected to **X**, see Figure 24. The same phenomenon is involved in the "ring-opening" of differentiation of trace expressions (see below).

4 Examples

4.1 Trace

In many optimization problems the trace of a matrix or an N-mode array is used as the objective criterion. The diagram notation offers for matrices a simple way of recognizing different trace rules. For a given N-mode array the trace is defined as:

$$tr(\underline{\mathbf{Z}}) = \sum_i Z_{iii\cdots i} \tag{34}$$

which is a scalar. For simplicity it is assumed that $\underline{\mathbf{Z}}$ has dimensions $[N \times N \times \cdots \times N]$. The N-mode trace uses a single sum nexus symbol. A three-mode trace is shown in Figure 25.

The diagram notation may be better than the usual matrix notation to explain some of the properties observed for trace of matrices. Constructing the trace of an N-mode array means that every unconnected edge must be connected.

The diagram rule that no mirror reflections are allowed is no longer valid since a trace is a scalar value. The upper part of Figure 26 shows the effect of mirroring the trace diagram. Below the diagram is also the matrix algebra results which shows that $tr(\mathbf{ABC}) = tr(\mathbf{CAB}) = tr(\mathbf{BCA}) = tr(\mathbf{C}^T\mathbf{B}^T\mathbf{A}^T) = tr(\mathbf{A}^T\mathbf{C}^T\mathbf{B}^T) = tr(\mathbf{B}^T\mathbf{A}^T\mathbf{C}^T)$. The ring structure of the trace in diagrams illustrates clearly that there is no particular startpoint that is preferable.

The lower part of Figure 26 shows that the trace of an expression does not change when pre- and postmultiplied by a suborthonormal matrix. In this case we have that $\mathbf{T}^T\mathbf{T} = \mathbf{I}$. The figure displays that

$$tr(\mathbf{T}^T\mathbf{X}\mathbf{T}) = tr(\mathbf{X}). \tag{35}$$

This is made clear by noting the arrow shorthand used to distinguish the different indices of the suborthonormal matrix **T**. The arrows meet head to head and as expected they "cancel" and give rise to the identity matrix which is here written as a connected edge. We now expect that the following should not in general be true:

$$tr(\mathbf{T}\mathbf{Q}\mathbf{T}^T) = tr(\mathbf{Q}). \tag{36}$$

It will be true if **T** is full orthonormal. In the diagram notation this is easily seen because the arrows of the suborthonormal matrices do not point head to head (this is illustrated in Figure 27). When **T** is fully orthonormal it will be signified in the diagram notation by having arrows on both edges.

4.1.1 "Ring opening"

In the next subsection, differentiation of matrix trace equations with respect to an N-mode array $\underline{\mathbf{X}}$ is discussed. Performing such a differentiation involves substitution of $\underline{\mathbf{X}}$ with a $\underline{\Delta}_{2N}^{(II)}$ symbol. After such a substitution the expression is no longer a trace since N edges from the $\underline{\Delta}_{2N}^{(II)}$ symbol is free. Here the trace diagram expression is referred to as a "ring" and that the differentiation symbol "opens" this ring when it substitutes $\underline{\mathbf{X}}$. "Ring opening" is demonstrated by the following example:

$$\begin{aligned} G(x_{i_1 i_2 \cdots i_N}) &= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N} y_{i_1 i_2 \cdots i_N} x_{i_1 i_2 \cdots i_N} \\ \frac{\partial G(x_{i_1 i_2 \cdots i_N})}{\partial x_{o_1 o_2 \cdots o_N}} &= \sum_{i_1}\sum_{i_2}\cdots\sum_{i_N} y_{i_1 i_2 \cdots i_N} \delta^{(II)}_{i_1 i_2 \cdots i_N o_1 o_2 \cdots o_N}. \end{aligned} \tag{37}$$

By rewriting the $\underline{\Delta}_{2N}^{(II)}$ array in terms of two-mode arrays, it is easy to see that this corresponds to the multiplication (chaining) of an identity array to each of the modes of $\underline{\mathbf{Y}}$:

$$\sum_{i_1}\sum_{i_2}\cdots\sum_{i_N} y_{i_1 i_2\cdots i_N}\delta_{i_1 o_1}\delta_{i_2 o_2}\cdots\delta_{i_N o_N} = y_{o_1 o_2\cdots o_N}. \tag{38}$$

The $\underline{\mathbf{Y}}$ array now contains unconnected edges only.

4.1.2 Differentiation of matrix traces

Using the rules for differentiating arrays it is possible to investigate differentiation of traces with respect to matrices. $\underline{\Delta}_4^{(II)}$ opens the trace-ring to produce matrices. In this subsection the fip indicating the first index plays an important role.

The following examples are presented in diagram form in Figure 28 and every scalar is here differentiated with respect to a matrix $\mathbf{X}$:

$$\begin{aligned}
\frac{\partial tr(\mathbf{X})}{\partial \mathbf{X}} &= \mathbf{I} \\
\frac{\partial tr(\mathbf{X}^T\mathbf{X})}{\partial \mathbf{X}} &= 2\mathbf{X} \\
\frac{\partial tr(\mathbf{X}^T\mathbf{A}\mathbf{X})}{\partial \mathbf{X}} &= \mathbf{A}\mathbf{X} + \mathbf{A}^T\mathbf{X} \\
\frac{\partial tr(\mathbf{X}\mathbf{A}\mathbf{X}^T\mathbf{B})}{\partial \mathbf{X}} &= \mathbf{B}^T\mathbf{X}\mathbf{A}^T + \mathbf{B}\mathbf{X}\mathbf{A}
\end{aligned} \tag{39}$$

In these examples an important rule not mentioned earlier must be stated: The Type II Kronecker delta replacing the $\mathbf{X}$ matrix gets (or "inherits") a fip in a position comparable to $\mathbf{X}$. When the $\underline{\Delta}_4^{(II)}$ opens up the trace-ring, the position of the matrices are read in the direction <u>from</u> the filled circle of the $\underline{\Delta}_{2N}^{(II)}$ array to the fip along the edge to the next matrix. The same is followed for all the other ring openings. Arrows are included in Figure 28 to emphasize the directions directed by the fips. Figure 29 and 30 are diagrams of the last two examples in equations 39.

Similarily, differentiating with respect to a three-mode array, produces a six-mode Kronecker Type II array which splits open the N-mode trace ring. The results for higher mode differentiation is very similar to what is obtained for differentiation with respect to matrices. Differentiating the trace of a three-mode array with respect to itself results in a three mode Type I Kronecker delta, see Figure 31. In Figures 32 and 33 three-mode differentiation with respect to the $\underline{\mathbf{C}}$ array is applied to the error term of the Tucker3 model. The result is the same as to perform multiplication on both sides by the transposed loading matrices, **E**, **G** and **H**. This estimate of the core arrays is true only when **E**, **G** and **H** are full orthonormal (see below).

4.2 Least-squares solution in the diagram notation

The least squares approach obtains a solution of

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \mathbf{e} \tag{40}$$

by minimizing the squared error $\mathbf{e}^T\mathbf{e}$. Thus we wish to find a $\mathbf{b}$ vector such that

$$\frac{\partial \mathbf{e}^T\mathbf{e}}{\partial \mathbf{b}} = \frac{\partial q}{\partial \mathbf{b}} = 0. \tag{41}$$

Writing this out we get:

$$\frac{\partial q}{\partial \mathbf{b}} = \frac{\partial}{\partial \mathbf{b}}((\mathbf{y}^T - \mathbf{b}^T\mathbf{X}^T)(\mathbf{y} - \mathbf{X}\mathbf{b}))$$

$$\begin{aligned} &= \frac{\partial}{\partial \mathbf{b}}(\mathbf{y}^T\mathbf{y} - 2\mathbf{b}^T\mathbf{X}^T\mathbf{y} + \mathbf{b}^T\mathbf{X}^T\mathbf{X}\mathbf{b}) \\ &= -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{b} = 0. \end{aligned} \tag{42}$$

Solving the last equation for **b** produces:

$$\begin{aligned} \mathbf{X}^T\mathbf{X}\mathbf{b} &= \mathbf{X}^T\mathbf{y} \\ (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\mathbf{b} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \\ \mathbf{b} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \end{aligned} \tag{43}$$

These steps are presented in diagram notation in Figure 34. Note that in equation $< 5 >$ in the figure the two-mode Type II Kronecker deltas have been written out explicitly in order to clarify the differentiation step. In step $< 7 >$ the equation is premultiplied by the inverse of the grammian matrix, $(\mathbf{X}^T\mathbf{X})^{-1}$.

4.3 N-mode SVD/PCA

For 2-mode SVD it is possible to formulate the **U** and **V** matrices in the following equation:

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{44}$$

as eigenvectors of grammian matrices $\mathbf{X}\mathbf{X}^T$ and $\mathbf{X}^T\mathbf{X}$. **S** is a diagonal matrix containing the square root of eigenvalues. **U** is the first largest eigenvectors of $\mathbf{X}\mathbf{X}^T$ and **V** is the first largest eigenvectors of $\mathbf{X}^T\mathbf{X}$. The three-mode SVD decomposition of an $\underline{\mathbf{X}}$ array is formulated as:

$$x_{ijk} = \sum_{p=1}^{P}\sum_{q=1}^{Q}\sum_{r=1}^{R} g_{ip}e_{jq}h_{kr}c_{pqr}. \tag{45}$$

This is the Tucker3 model which is illustrated in the diagram notation in Figure 35. There are several ways of solving this equation but the case is not a simple extension of two-mode SVD. For N-mode data arrays it is possible to formulate grammian-like matrices for each mode. The number of grammian matrices is the same as the mode number. For the three-mode case it is possible to create three grammian-like matrices and find <u>all</u> the eigenvectors for each grammian matrix. This is referred to as the Tucker Method I [Kroonenberg (1983)]. The grammian matrices are produced as follows:

$$c1_{ii'} = \sum_{j=1}^{J} \sum_{k=1}^{K} x_{ijk} x_{i'jk} \tag{46}$$

$$c2_{jj'} = \sum_{i=1}^{I} \sum_{k=1}^{K} x_{ijk} x_{ij'k} \tag{47}$$

$$c3_{kk'} = \sum_{i=1}^{I} \sum_{j=1}^{J} x_{ijk} x_{ijk'}. \tag{48}$$

The eigenvectors associated with $\mathbf{C1}, \mathbf{C2}, \mathbf{C3}$ are labeled $\mathbf{G}, \mathbf{E}$ and $\mathbf{H}$. Using these matrices it is possible to get an estimate of the core array $\underline{\mathbf{C}}$:

$$c_{pqr} = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{k=1}^{K} g_{ip} e_{jq} h_{kr} x_{ijk}. \tag{49}$$

The diagram of this solution is shown in Figure 33 where differentiation with respect to the core array $\underline{\mathbf{C}}$ gave the same result.

Since all factors have been extracted the loading matrices for the Tucker Method I are full orthonormal.

The Tucker Method I decomposition is not optimal when the a factors with largest eigenvalues only are extracted. The estimators of the core array are not longer least squares ones as the deletion of later eigenvectors in the principal component matrices affect the estimators in a complicated way. A least squares fit can be accomplished using the Alternating Least Squares (ALS) algorithm [Kroonenberg (1983)]. This algorithm will first be shown using the Kronecker notation and eventually in diagram notation. Let $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3$ be the three matrices which are the unfolding along the three modes of the 3-array $\underline{\mathbf{X}}$. The algorithm is initiated by first computing estimates of the loading matrices $\mathbf{G}_{\text{start}}$, $\mathbf{E}_{\text{start}}$ and $\mathbf{H}_{\text{start}}$ by the Tucker Method I. $\mathbf{G}_{\text{start}}$, $\mathbf{E}_{\text{start}}$ and $\mathbf{H}_{\text{start}}$ are subsequently inserted into the following ALS iteration steps:

```
for i = 1 to iterations
 begin
        $\mathbf{A} = \mathbf{X}_1(\mathbf{H}_{i-1} \otimes \mathbf{E}_{i-1})$
        $\mathbf{G}_i = eig(\mathbf{A}\mathbf{A}^T)$                (50)
        $\mathbf{A} = \mathbf{X}_2(\mathbf{E}_{i-1} \otimes \mathbf{G}_i)$
        $\mathbf{H}_i = eig(\mathbf{A}\mathbf{A}^T)$                (51)
        $\mathbf{A} = \mathbf{X}_3(\mathbf{G}_i \otimes \mathbf{H}_i)$
        $\mathbf{E}_i = eig(\mathbf{A}\mathbf{A}^T)$                (52)
 end
```

Matrix $\mathbf{A}$ temporarily stores the results and is overwritten when new values are assigned to its matrix elements. The function eig returns the first largest eigenvectors of $\mathbf{A}\mathbf{A}^T$ (this is a reduced decomposition).

Equations 50,51,52 have been written in diagram notation in Figure 36.

4.4 B-spline compression of N-mode arrays

N-mode data arrays constitute a serious storage and computational problem when the size of the modes are large. Instruments in e.g. chemistry now produce N-mode data arrays with large modes. In order to partially solve the increasing data size problem, it has been previously suggested that compression by B-splines may be efficient [Alsberg, Kvalheim (1993)]. If $\underline{\mathbf{X}}$ is the original N-array the idea is to construct a smaller N-array $\underline{\mathbf{C}}$ which is used instead of $\underline{\mathbf{X}}$. In B-splines [de Boor (1978), Farin (1990), de Boor (1990)] it is assumed that a function can be written as a linear combination of B-spline bases. The B-spline basis set $\mathbf{B}_i$ is constructed from a <u>knot vector</u> $\mathbf{h}_i$. One such basis set is constructed for each mode of the data array. The B-spline basis matrix $\mathbf{B}_i$ can be computed systematically by a recursive formula [Cheney, Kincaid (1980)].

For 2-arrays it is possible to formulate the B-spline compression as:

$$\mathbf{C} = ((\mathbf{B}_1^T\mathbf{B}_1)^{-1})^T\mathbf{B}_1^T\mathbf{X}^T\mathbf{B}_2(\mathbf{B}_2^T\mathbf{B}_2)^{-1} \tag{53}$$

where $\mathbf{C}$ is the compressed matrix. Analogously it is possible to find the corresponding $\underline{\mathbf{C}}$ for N-mode arrays. This is demonstrated for the three-mode data array in a series of figures showing in detail the deduction of the formula for the core array which is similar to equation 53. In Figure 37 the three-mode B-spline model is stated (note here the use of thick lines for large modes). In Figure 38 the equation is multiplied by the basis matrices $\mathbf{B}_i$, $i \in \{1,2,3\}$ on both sides. In order to transform away the $\mathbf{B}_i^T\mathbf{B}_i$ $i \in \{1,2,3\}$ grammian matrices, each mode is multiplied by the inverse $(\mathbf{B}_i^T\mathbf{B}_i)^{-1}$ $i \in \{1,2,3\}$, this is shown in Figure 39. The final formula is shown in Figure 40.

4.5 The method of postponed basis matrix multiplication (PBM) applied to three-mode PCA

This is an example where the diagram notation from the start was used to solve a problem [Alsberg, Kvalheim (1994)b]. Three-mode PCA algorithms such as the ALS, is very computer demanding for large arrays. In this case a "large array" means that every mode is large. In addition, it was assumed that the array was "smooth", i.e. it could be modeled by smooth basis functions for the different modes. The three-mode B-spline routine as described above was applied to a large data array $\underline{\mathbf{X}}$ and compressed to a smaller 3-array named $\underline{\mathbf{C}}$. The idea is to use $\underline{\mathbf{C}}$ instead of $\underline{\mathbf{X}}$ in the three-mode PCA routine. Since it is smaller the computation requires less floating point operations (FLOPS). The results from the analysis of this smaller array are comparable but not equal to the results obtained from a direct analysis of the larger array. A naïve approach is to pre-multiply the loading matrices from the ALS algorithm of $\underline{\mathbf{C}}$ by the corresponding basis matrices used in the compression:

$$\mathbf{G}_b = \mathbf{B}_1\mathbf{G}_c \tag{54}$$

$$\mathbf{E}_b = \mathbf{B}_2\mathbf{E}_c \tag{55}$$

$$\mathbf{H}_b = \mathbf{B}_3\mathbf{H}_c \tag{56}$$

where $\mathbf{G}_c, \mathbf{E}_c, \mathbf{H}_c$ are the loading matrices from the analysis of $\underline{\mathbf{C}}$. Unfortunately, $\mathbf{G}_b, \mathbf{E}_b, \mathbf{H}_b$ will not in general be equal to the loading matrices $(\mathbf{G}, \mathbf{E}, \mathbf{H})$ from analysis of the original large array. For some B-spline bases the back-estimates will to some extent have similarities to the correct loadings of the larger array $\underline{\mathbf{X}}$. For compression matrices in general, however, the results are often very different from the loadings of the original array. The estimates are not suborthonormal as they should be. Simple orthonormalization is not sufficient to compensate for the distortion effects.

The PBM method was first developed for standard PCA [Alsberg, Kvalheim (1994)a] and is based on the following simple idea: When an $\mathbf{X}$ matrix is written in terms of a compression model, there are parts of the PCA algorithm where the multiplication of basis matrices are redundant, i.e. those multiplications can be *postponed* to later when the algorithm has converged for all the factors. By rewriting the two-mode PCA NIPALS algorithm in terms of the compression model it is easy to observe that every part of the algorithm can be written as:

$$\begin{aligned}
\mathbf{B}_1 \text{array}_1 &= \mathbf{B}_1 \text{array}_2 && (57) \\
\text{array}_1 \mathbf{B}_2^T &= \text{array}_2 \mathbf{B}_2^T && (58) \\
\mathbf{B}_1 \text{array}_1 \mathbf{B}_2^T &= \mathbf{B}_1 \text{array}_2 \mathbf{B}_2^T && (59)
\end{aligned}$$

where "array" means either a vector or a matrix. When such equations are involved in an iteration, the pre- and/or post multiplications 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. When this was found to be efficient for the PCA algorithm, it was natural to test whether the same could be done for the ALS algorithm used to solve the Tucker3 model. Using the diagram notation the author found it very simple to see how this could be accomplished. To apply the PBM method it is assumed that the arrays in the different iteration steps in the analysis of $\underline{\mathbf{X}}$ can be written in terms of the compression bases:

$$\mathbf{G} = \mathbf{B}_1\mathbf{G}_0 \tag{60}$$
$$\mathbf{E} = \mathbf{B}_2\mathbf{E}_0 \tag{61}$$
$$\mathbf{H} = \mathbf{B}_3\mathbf{H}_0 \tag{62}$$

and that $\underline{\mathbf{X}}$ can be modeled as shown in Figure 35.

The matrices $\mathbf{G}_0, \mathbf{E}_0, \mathbf{H}_0$ are in general not suborthonormal.

In order to deduct the equations for the PBM version of the ALS algoritm, different array elements are substituted with the corresponding compression model. In Figure 41 the step for computing the $\mathbf{G}$ matrix in the ALS algorithm is shown. Each of the loading matrices in the diagram expression are rewritten in terms of their corresponding compression basis matrices. Note the placement of the thick lines that signify the large modes. In order to compute an estimate of $\mathbf{G}$ in the ALS iteration step, an eigenvalue decomposition of the grammian like matrix $\mathbf{Q}$ (see figure) is necessary. In Figure 42, $\underline{\mathbf{X}}$ is substituted by its compression model. It is now easy to see that the basis matrix $\mathbf{B}_1$ for $\mathbf{G}$ is multiplied on both sides of the equation and thus is redundant. It is now possible to rewrite $\mathbf{Q}$ in terms of the compression model:

$$\mathbf{Q} = \mathbf{B}_1\mathbf{Q}_0\mathbf{B}_1^T. \tag{63}$$

The power method approach for computing eigenvalues is assumed. Every current estimate of the eigenvectors can be described in terms of the compression model. The iteration steps in the power method have the following structure when written in terms of the compression model:

$$\mathbf{B}_1\mathbf{f}_0 = \mathbf{B}_1\mathbf{Q}_0\mathbf{B}_1^T\mathbf{B}_1\mathbf{a}_0 \tag{64}$$
$$\mathbf{B}_1\mathbf{a}_1 = \frac{\mathbf{B}_1\mathbf{f}_0}{(\mathbf{f}_0^T\mathbf{B}_1^T\mathbf{B}_1\mathbf{f}_0)^{1/2}} \tag{65}$$
$$\mathbf{B}_1\mathbf{a}_0 = \mathbf{B}_1\mathbf{a}_1. \tag{66}$$

Setting $\Gamma = \mathbf{B}_1^T\mathbf{B}_1$ and observing that the pre multiplication by $\mathbf{B}_1$ is redundant in the iteration, the PBM iteration will look like:

$$\mathbf{f}_0 = \mathbf{Q}_0\Gamma\mathbf{a}_0 \tag{67}$$
$$\mathbf{a}_1 = \frac{\mathbf{f}_0}{(\mathbf{f}_0^T\Gamma\mathbf{f}_0)^{1/2}} \tag{68}$$
$$\mathbf{a}_0 = \mathbf{a}_1. \tag{69}$$

The updating steps are:

$$\lambda = \mathbf{a}_0^T\Gamma\mathbf{Q}_0\Gamma\mathbf{a}_0 \tag{70}$$
$$\mathbf{Q}_0 = \mathbf{Q}_0 - \lambda\mathbf{a}_0\mathbf{a}_0^T. \tag{71}$$

If the compression is perfect, the eigenvalue λ from the PBM method is identical to the original eigenvalue of the uncompressed representation. The same PBM procedure is repeated for the other loading matrices **E** and **H** also.

The PBM version of the ALS algorithm achieves the following:

- Computation on the $\underline{\mathbf{C}}$ and the grammians $\mathbf{B}_i^T\mathbf{B}_i$ only .
- Reduced number of floating point operations needed.

- Perfect back-estimates of the original loading matrices of the uncompressed array $\underline{\mathbf{X}}$ when premultiplying the loading-like matrices with the corresponding compression matrices.

The PBM approach is extendable to N-mode PCA.

5 Discussion

Some drawbacks associated with the suggested diagram method are:

- It requires a graphical display which means that regular word processors cannot be used. Making diagrams for publishing involves more work than e.g. writing ordinary formulas in a word processor.
- No commercially programs are available to use these diagrams.
- Not every mathematical manipulation is easy to display in the diagram notation. The reason for this is that the index topology is given special attention which imply graphically difficulties in some mathematical operations.

The major advantage of the diagram notation is that mathematical manipulation analogous to what is done on matrix symbols is possible for array equations in general. Diagrams are <u>not</u> just nice presentations of algebraic structures, but symbols that can be manipulated. The diagram notation presents graphically the index topology of complex array equations which should make it easier for the investigator to detect e.g. symmetries and interesting patterns. The use of diagrams for expressing new ideas and teaching students should also be easier.

The diagram notation opens for the possibility of interactive computer tools operating on N-mode array equations. To the present author's knowledge there are today no high level programming tools (languages) available for the investigator who wishes to work efficiently with N-mode data array equations and algorithms. For matrix manipulation there are powerful programs such as MATLAB and Maple which lets the investigator formulate his/hers matrix expressions in the computer almost as they appear on paper. Construction of an analogous tool for N-mode problems based on diagram notation is possible. Such a program for manipulation of traditional Feynman diagrams has already been constructed [Kübleck, Böhm, and Denner (1990)] but is of little use to scientist in the field of psychometrics and chemometrics.

The field of array (and tensor) algebra is so complex that it may not be realistic to expect that one type of notation will cover all the needs for manipulation and understanding of such equations. The diagram notation should therefore be viewed as supplementary to the already existing notations, and not as an attempt to completely replace the other notations. The diagram notation may be an excellent tool for e.g. introducing new scientist to N-mode array (tensor) algebra and problems related to N-mode index topology. The demand for a more user-friendly notation will increase in proportion to the spread and use of programs and theory based on N-mode arrays. The diagram notation may be the answer to such a demand.

Author note

The author is indebted to The Royal Norwegian Council for Scientific and Industrial Research (NTNF) for financial support. The author wishes to thank the professors Rolf Manne and Gerhard Berge for valuable comments on this paper.

References

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

[Alsberg, Kvalheim (1994)a] Alsberg, B.K. and Kvalheim, O.M., (1994), Speed improvement of multivariate algorithms by the method of postponed basis matrix multiplication. Part I. Principal component analysis, Chemometrics and Intelligent Laboratory Systems,in press

[Alsberg, Kvalheim (1994)b] Alsberg, B.K. and Kvalheim, O.M., (1994), 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,in press

[Arfken (1985)] Arfken, G. (1985) Mathematical methods for physicists, 3rd ed. Academic Press, San Diego

[Blinn (1992)a] Blinn, J.F. (1992) Uppers and downers. IEEE Computer Graphics and Applications, March, 85-91

[Blinn (1992)b] Blinn, J.F. (1992) Uppers and downers: Part 2. IEEE Computer Graphics and Applications, May, 80-85

[Cheney, Kincaid (1980)] Cheney, W. and Kincaid, D. (1980), Numerical mathematics and computing, Brooks/Cole Publishing Company [de Boor (1978)] de Boor, C. (1978) A practical guide to splines Applied Mathematics Sciences, Springer-Verlag, New York

[de Boor (1990)] de Boor, C.(1990) Spline toolbox for use with MATLAB. User's Guide.,South Natick, MathWorks Inc.

[Farin (1990)] Farin, G. (1990) Curves and surfaces for computer aided geometric design, a practical guide 2nd. ed. Academic Press

[Geladi (1989)] Geladi, P. (1989) Analysis of multi-way (multi-mode) data. Chemometrics and intelligent laboratory systems, 7 11-30

[Kroonenberg (1983)] Kroonenberg, P.M. (1983) Three mode principal component analysis. DSWO Press, Leiden

[Kruskal (1977)] Kruskal, J.B. (1977) Three-way arrays: Rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. Linear algebra and its applications, 18, 95-138

[Carrol, Pruzansky, Kruskal (1980)] Carrol, J.D., Pruzansky, S. and Kruskal, J.B., (1980) CANDELINC: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters. Psychometrika, 45 (1), 3-24

[Kübleck, Böhm, and Denner (1990)] Kübleck, J., Böhm, and Denner, A. (1990) Feyn Arts - computer-algebraic generation of Feynman graphs and amplitudes. Computer Physics Communications, 60, 165-180

[Magnus and Neudecker (1988)] Magnus, J.R. and Neudecker, H. (1988) Matrix differential calculus with applications in statistics and econometrics John Wiley & Sons Ltd. ,Great Britain, Essex

[Mattuck (1976)] Mattuck, R.D.(1976) A guide to Feynman diagrams in the many-body problem, McGraw-Hill Inc., USA, New York.

[Smilde (1992)] Smilde, A. K. (1992) Three-way analyses. Problems and prospects. Chemometrics and intelligent laboratory systems, 15, 143-157

[Stedman (1990)] Stedman, G.E.(1990)Diagram techniques in group theory Cambridge University Press, Cambridge

[Tucker (1966)] Tucker, L. (1966) Some mathematical notes on three-mode factor analysis. Psychometrika, 31, 279-311

[Tucker (1964)] Tucker, L. (1964) The extension of factor analysis to three-dimensional matrices. in Frederikson, N. and Gulliksen, H. (Editors) Contributions to mathematical psychology, Holt, Rinehart and Winston, New York, 110-182

[van der Kloot (1985)] van der Kloot, W.A. (1985) External analysis with three mode-mode principal component models. Psychometrika, vol.50, no.4 479-494,

[Wilson (1985)] Wilson, J.R. (1985) Introduction to graph theory, 3rd ed. Longman Scientific & Technical, UK, Essex

A Figure captions

- Figure 1: Illustration of a Feynman diagram. Here the scattering of two photons into an electron and a positron pair is shown.
- Figure 2: A table showing the different types of notations and their representations of N-mode arrays.
- Figure 3: This figure illustrates the main ideas behind the diagram notation.
- Figure 4: Some examples of standard vector and matrix algebra using the diagram notation. See text for details about these examples.
- Figure 5: Rotations and translations of the diagram are allowed but mirror reflections are not. It is only the topology which is important.
- Figure 6: When summing different diagrams the number of unconnected edges must be the same. Here it is four unconnected edges.
- Figure 7: Illustration of how different mathematical operations on each array element is depicted in the diagram notation.
- Figure 8: When more than two arrays are involved in summing of an index it is necessary to introduce a new symbol which signify that all the edges belong to the same summation. The sum nexus symbol (a filled rectangle) has therefore been introduced. The sum nexus is a Type I Kronecker delta array.
- Figure 9: Example showing summing over two indices shared by more than two array elements. Two sum nexus symbols means two summations.

- Figure 10: Arrows are used to indicate that the matrix at hand is sub- or full orthonormal. By convention it is decided that two arrows pointing head to head result in the identity matrix. Two-mode Kronecker delta Type I and II are both equal to the identity matrix; both the filled rectangle and the filled circle give the same results.
- Figure 11: The use of thick lines to indicate large modes is of interest in problems where it is important to keep such edges from becoming unconnected.
- Figure 12: This example illustrates how an array expression is multiplied to another consisting of a sum two other array expressions.
- Figure 13: Chaining of nodes is used for expressing outer products in the diagram notation. Here the outer product of two vectors producing a matrix is shown.
- Figure 14: A three-mode vector outer product is shown using chaining of nodes.
- Figure 15: A more complex example of chaining where the expression also contains nodes connected via a sum nexus symbol is shown. Other chaining patterns are possible, but three configurations are shown where all the three nodes not involved in any summation $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ are collected in one chain.
- Figure 16: This figure shows an example of the the trilinear PARAFAC decomposition.
- Figure 17: Example of array outer product using chaining.

- Figure 18: The one-mode Type I Kronecker delta δ_i (the sum nexus) is used when only one array contain the sum-index. This example illustrate $\sum_i x_{ij} = \sum_i x_{ij}\delta_i$.

- Figure 19: This and the next figure shows how it is possible to solve an array equation. The aim in this example is to write the array $\underline{\mathbf{R}}$ in terms of the other arrays. Note that it is here assumed that the $\mathbf{P}$ has an inverse.

- Figure 20: This figure shows the last part of the array equation solving from the previous figure.

- Figure 21: This example shows the differentiation of an expression involving a three mode array with respect to the vector $\mathbf{x}$. Three terms are obtained if no symmetry in $\underline{\mathbf{A}}$ is assumed.

- Figure 22: In this example a three-mode array is differentiated with respect to a matrix. In general, when differentiating with respect to an N-mode array we replace that array in the expression with a $2N$-mode Type II Kronecker delta. Symmetry in the $\underline{\mathbf{A}}$ array is not assumed.

- Figure 23: The differentiation symbol can be depicted as a chain of identity matrices. Here a six-mode differentiation symbol is shown.

- Figure 24: Converting the differentiation symbol to a chain of identity matrices is here demonstrated to be useful for simplifying equations. The result is a chaining of the arrays $\underline{\mathbf{Y}}$ and $\underline{\mathbf{Z}}$. There are no summing or any shared indices in the result.

- Figure 25: Illustration of the trace for a three mode array. Unfortunately, the simple rules for trace of pre- and post multiplication of suborthonormal matrices found in two-mode arrays, do not hold in general for arrays of order $N > 2$.

- Figure 26: The upper part of the figure shows that the prohibition of mirror reflection of diagrams does not hold for *scalar* values. The lower part of the figure shows that the trace of a matrix $\mathbf{X}$ pre- and postmultiplied by orthonormal matrices is the same as the trace of $\mathbf{X}$ itself. Since the arrows of the edges point head to head, we know this is equal to an identity matrix.

- Figure 27: This is an example predicted by the diagram notation to be true, i.e. the trace for pre- and post multiplication of a suborthonormal matrix transposed the "wrong" way will <u>not</u> in general be the same as the trace of the matrix $\mathbf{Q}$.

- Figure 28: The upper part of the figure shows differentiation of a trace of a matrix with respect to itself. This is equal to the identity matrix. Note that a $\underline{\Delta}_4^{(II)}$ array is substituted for every occurrence of $\mathbf{X}$ in an expression. In the lower part is the differentiation of $tr(\mathbf{X}\mathbf{X}^T)$ with respect to $\mathbf{X}$. Note the dashed arrows which indicate along which way it is necessary to read in order to obtain the correct transposition of the matrices. The arrows direction are determined by the direction <u>from</u> $\underline{\Delta}_4^{(II)}$ to its nearest fip.

- Figure 29: Diagram deduction of $\frac{\partial \mathbf{X}^T\mathbf{A}\mathbf{X}}{\partial \mathbf{X}}$.

- Figure 30: Diagram deduction of $\frac{\partial \mathbf{X}^T\mathbf{A}\mathbf{X}\mathbf{X}}{\partial \mathbf{X}}$.

- Figure 31: The upper illustration shows differentiation of the trace of a three-mode array which is equal to the three-mode identity array

(Type I). Below differentiation of the trace of an expression involving two three-mode $\underline{\mathbf{X}}$ arrays is demonstrated to be the same as $2\underline{\mathbf{X}}$.

- Figure 32: The Tucker3 model with the error array $\underline{\mathbf{R}}$ is expressed in terms of $\underline{\mathbf{X}}$, $\underline{\mathbf{C}}$, and the loading matrices. In the lower part of the figure the error scalar value is written in terms of the other array elements.

- Figure 33: Here the error scalar value in the previous figure is differentiated with respect to the $\underline{\mathbf{C}}$ array. In each occurrence of the $\underline{\mathbf{C}}$ array, it is substituted with the six-mode Kronecker Type II array. At the bottom part of the figure, the ring-opening by the Type II Kronecker delta has been performed and one expression has been moved to the other side of the equation sign. This result is the same as if $\underline{\mathbf{X}}$ in Figure 32 had been multiplied by the transposed versions of the loading matrices.

- Figure 34: This figure shows how to find the least squares solution by differentiating the the error scalar $\mathbf{e}^T\mathbf{e}$ with respect to the $\mathbf{b}$ vector using the diagram notation.

- Figure 35: The Tucker3 model expressed in diagram notation.

- Figure 36: A central part of the ALS algorithm is shown in diagram notation. Note that "EIG" returns the a largest eigenvectors of a matrix.

- Figure 37: The three-mode B-spline model is shown in diagram notation. Note the thick lines indicating large modes. The next figures will show in how it is possible to extract the $\underline{\mathbf{C}}$ array.

- Figure 38: The basis matrices $\mathbf{B}_i$, $i \in \{1,2,3\}$ are multiplied on both sides of the equation sign.
- Figure 39: The inverse matrices $(\mathbf{B}_i^T\mathbf{B}_i)^{-1}, i \in \{1,2,3\}$ are multiplied on both sides of the equation sign. The idea is to remove the $\mathbf{B}_i^T\mathbf{B}_i$ parts at the right-hand side of the equation.
- Figure 40: This the final diagram expression for generating the compressed core array $\underline{\mathbf{C}}$.
- Figure 41: The upper part emphasizes that the loading matrices can be written in terms of the compression model. Note that the matrices $\mathbf{G}_0, \mathbf{E}_0, \mathbf{H}_0$ in general are <u>not</u> suborthonormal as normal loading matrices are supposed to be. The lower part of the figure shows the ALS iteration part for the estimation of the $\mathbf{G}$ matrix only. It is analogous for the other two loading matrices. Compare this figure with figure 36. Note that the $\mathbf{E}$ and $\mathbf{H}$ matrices in matrix $\mathbf{Q}$ are here substituted with their corresponding compression models.
- Figure 42: This is the same as previous figure but here the $\underline{\mathbf{X}}$ array is substituted with the compression model. It is easy to see that the pre- and post multiplication of the basis matrix $\mathbf{B}_2$ is redundant. It can be postponed to after the iteration has converged for all the factors.

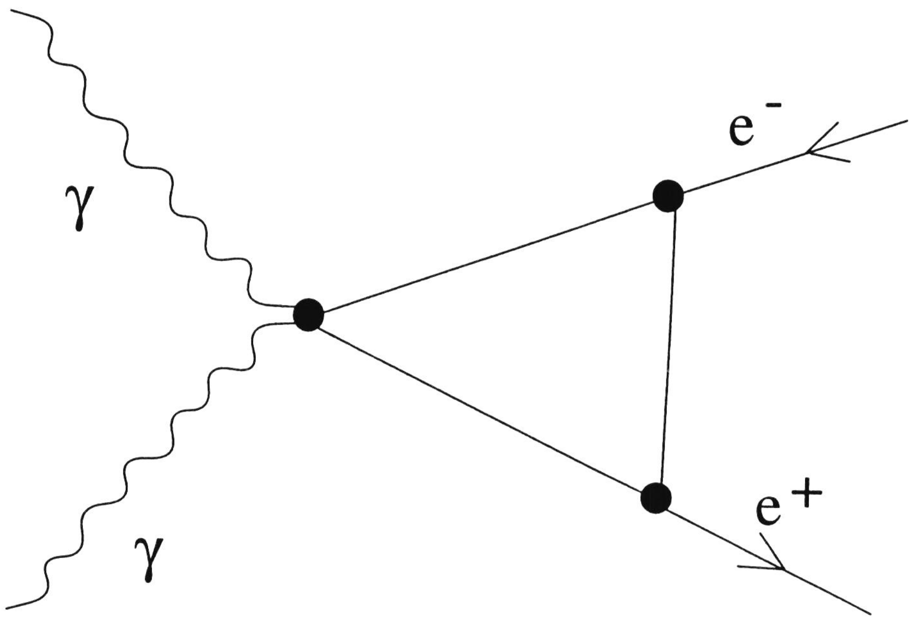


Figure 1:

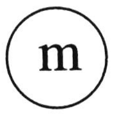
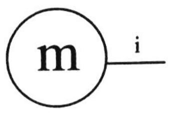
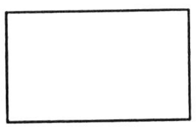
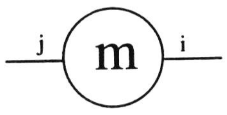
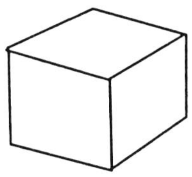
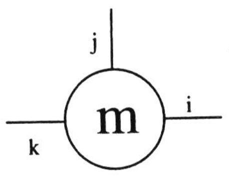
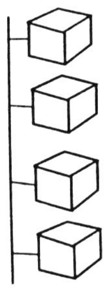
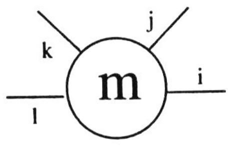
Mode	Symbol	Figure	Diagram
0	m	•	
1	m_i		
2	m_{ij}		
3	m_{ijk}		
4	m_{ijkl}		

Figure 2:

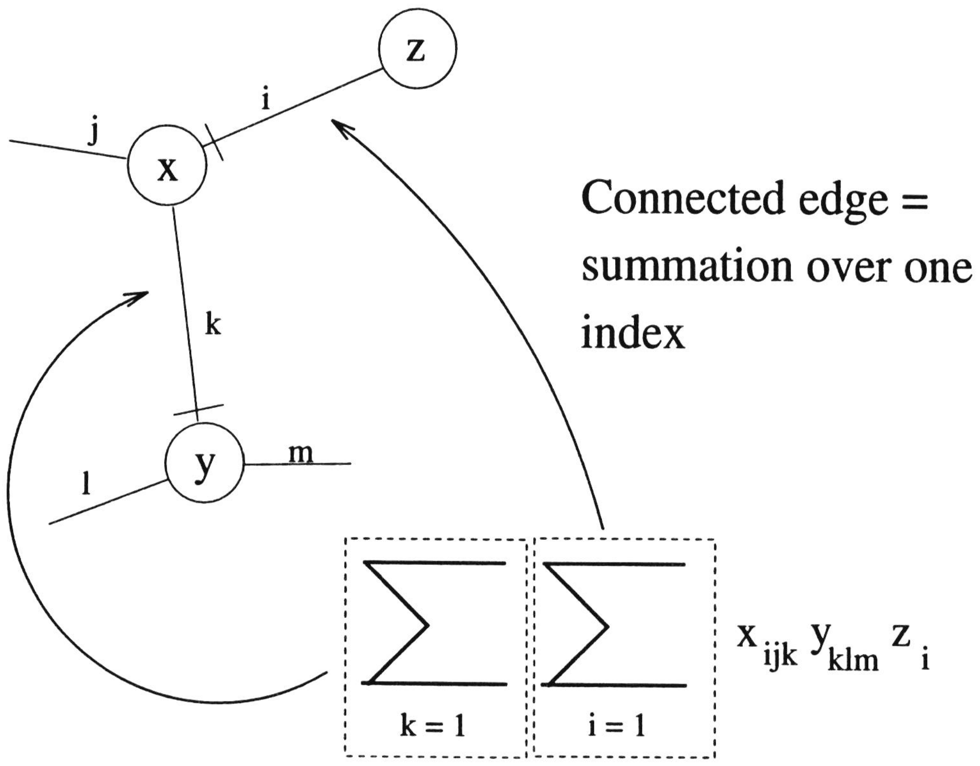

z
i
j
x
k
y
l
m
Connected edge = summation over one index
$\sum_{k=1} \sum_{i=1} x_{ijk} y_{klm} z_i$

Figure 3:

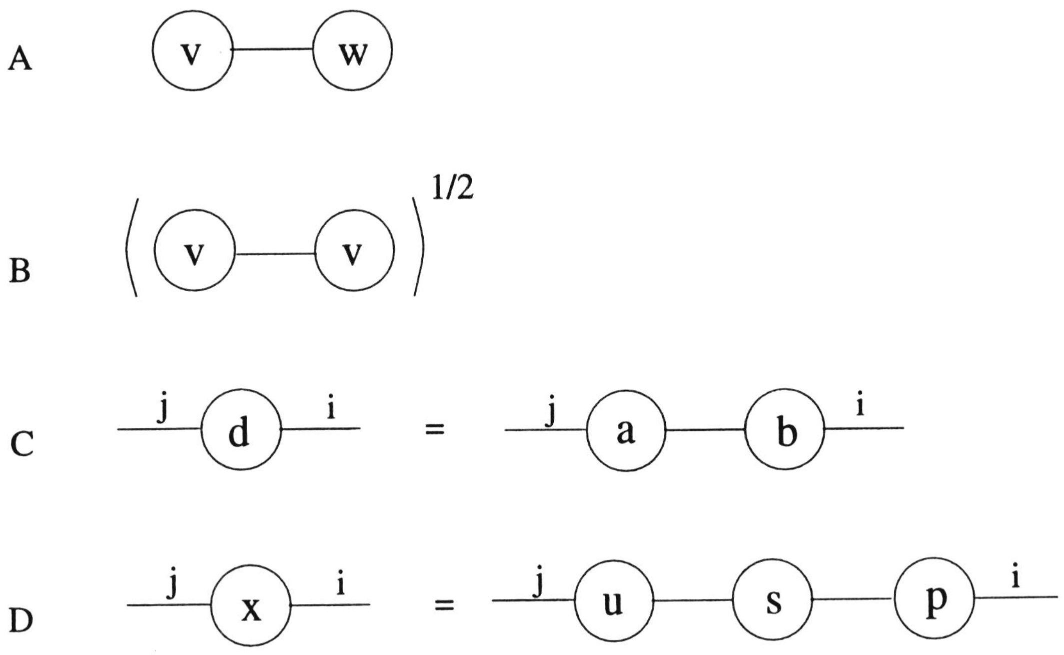


Figure 4:

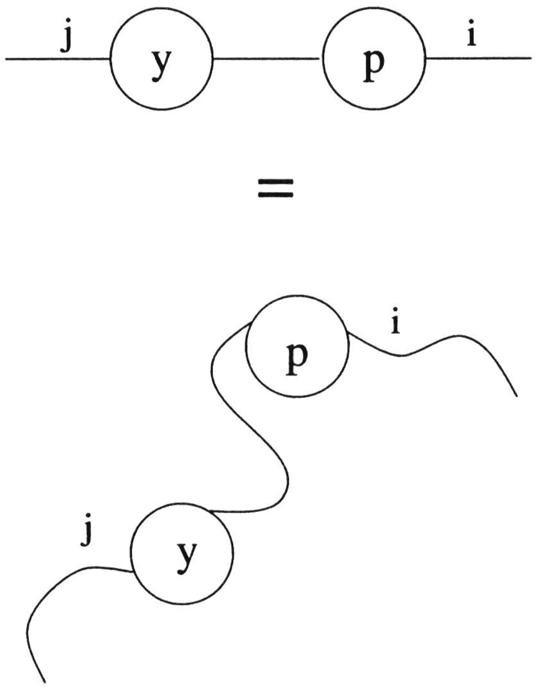


Figure 5:

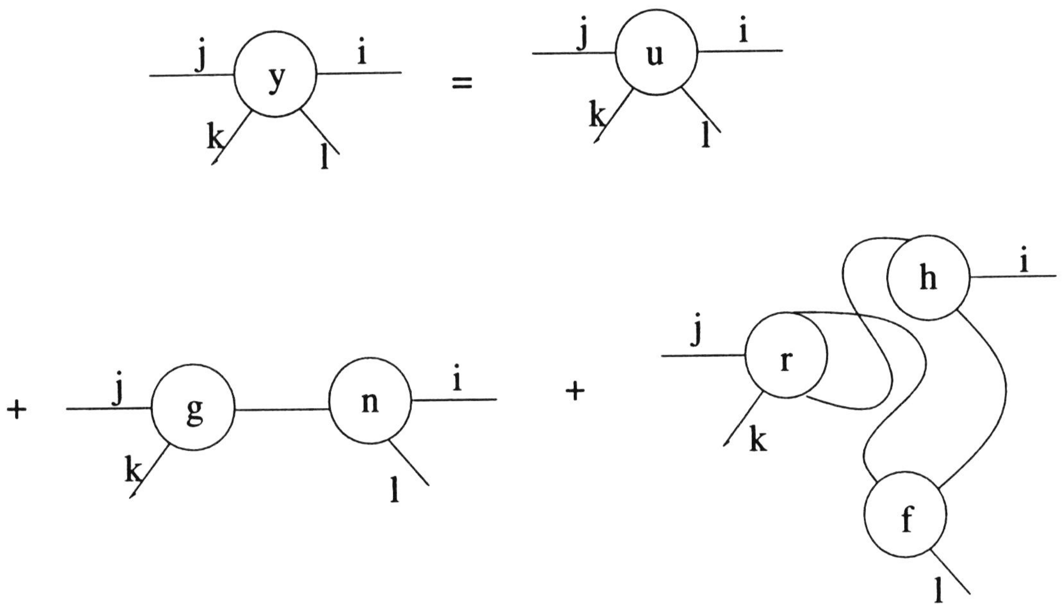
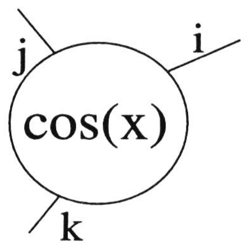
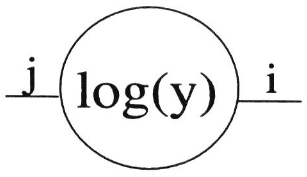


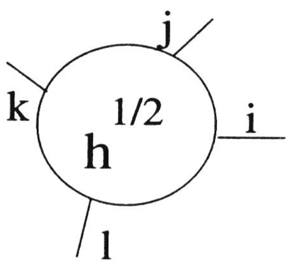
Figure 6:



$$= \cos(x_{ijk})$$

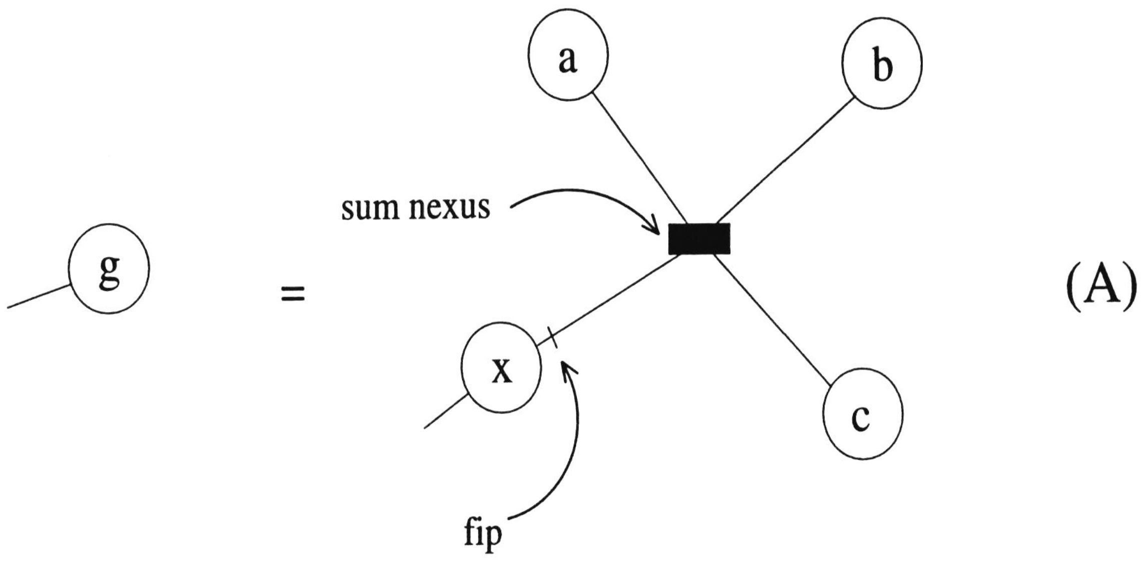


$$= \log(y_{ij})$$



$$= (h_{ijkl})^{1/2}$$

Figure 7:



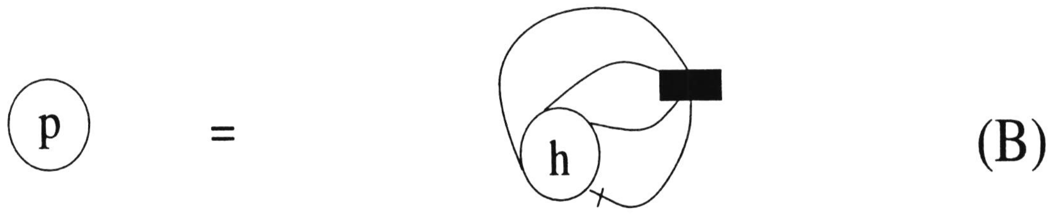


Figure 8:

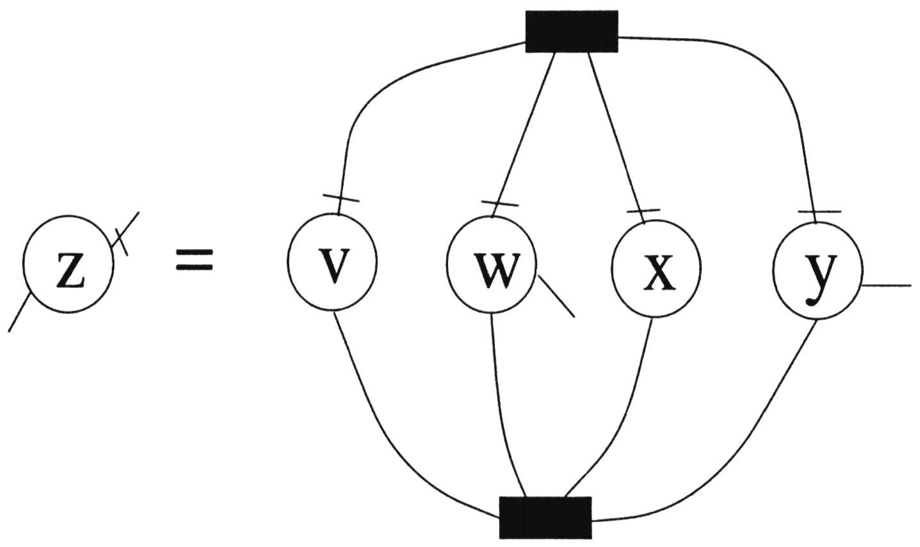


Figure 9:

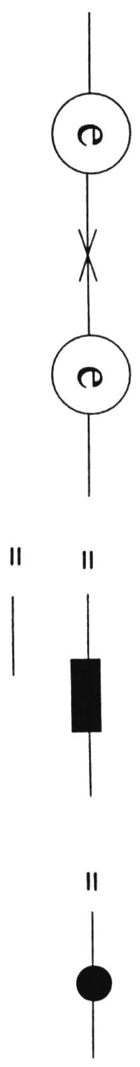


Figure 10:

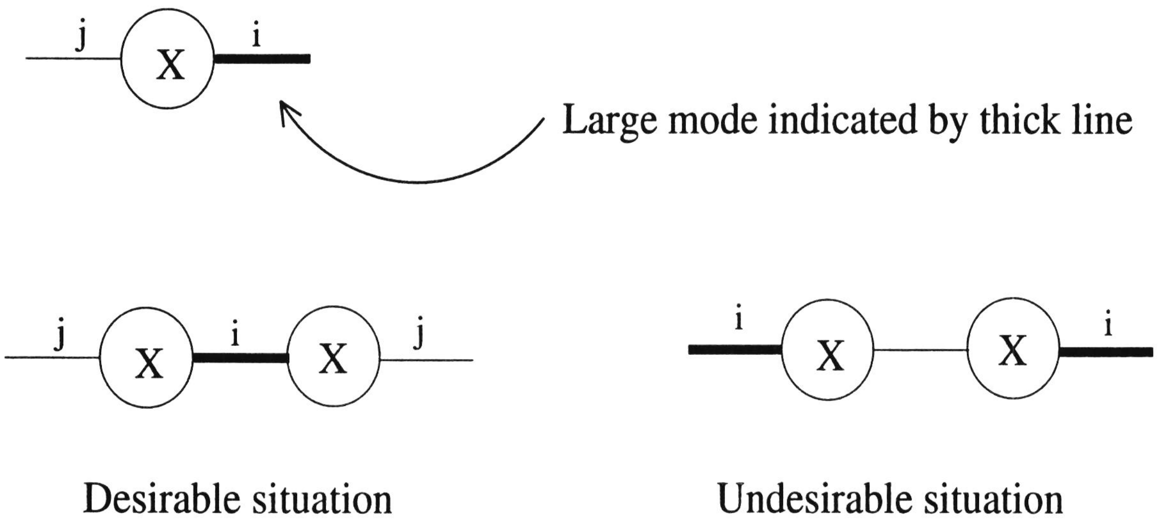

j
X
i
Large mode indicated by thick line
j
X
i
X
j
Desirable situation
i
X
X
i
Undesirable situation

Figure 11:

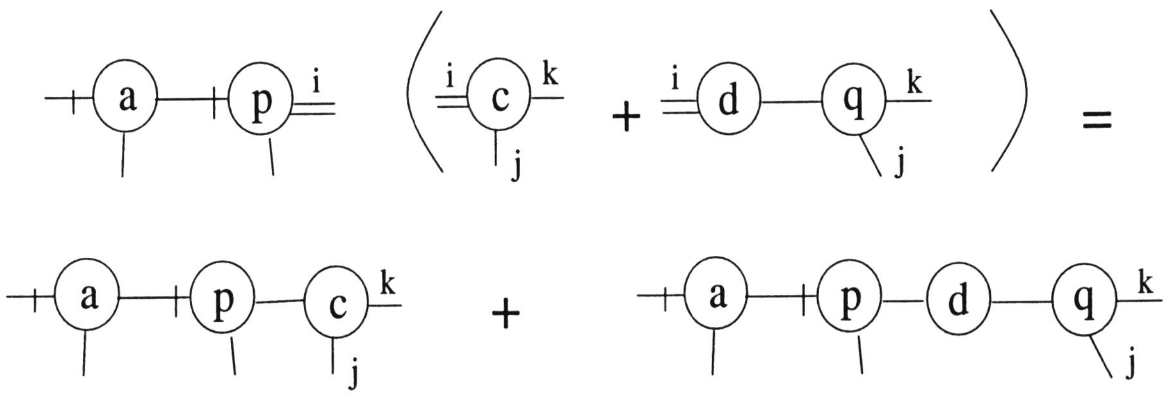


Figure 12:

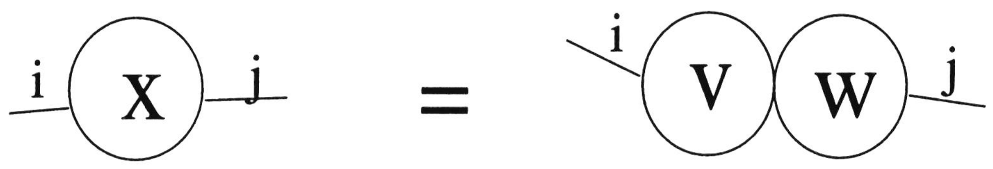


Figure 13:

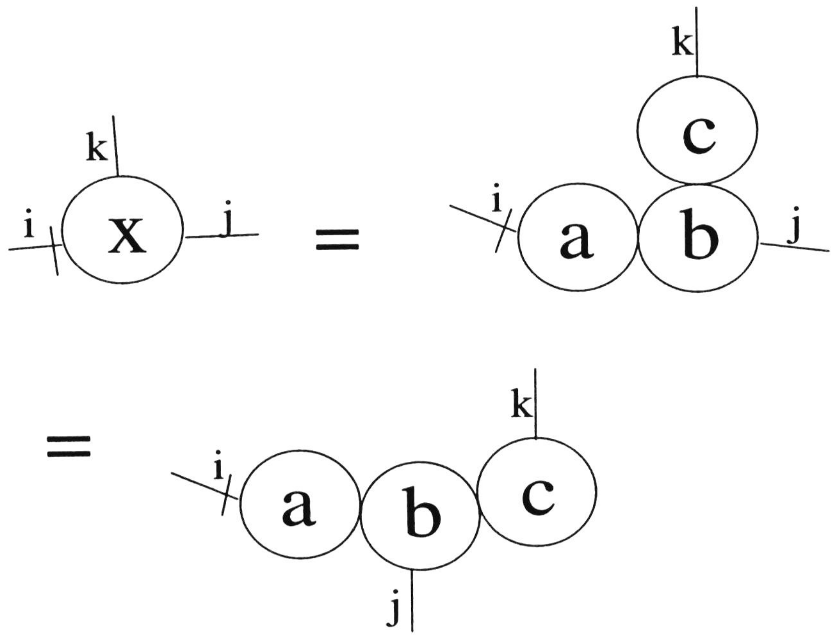


Figure 14:

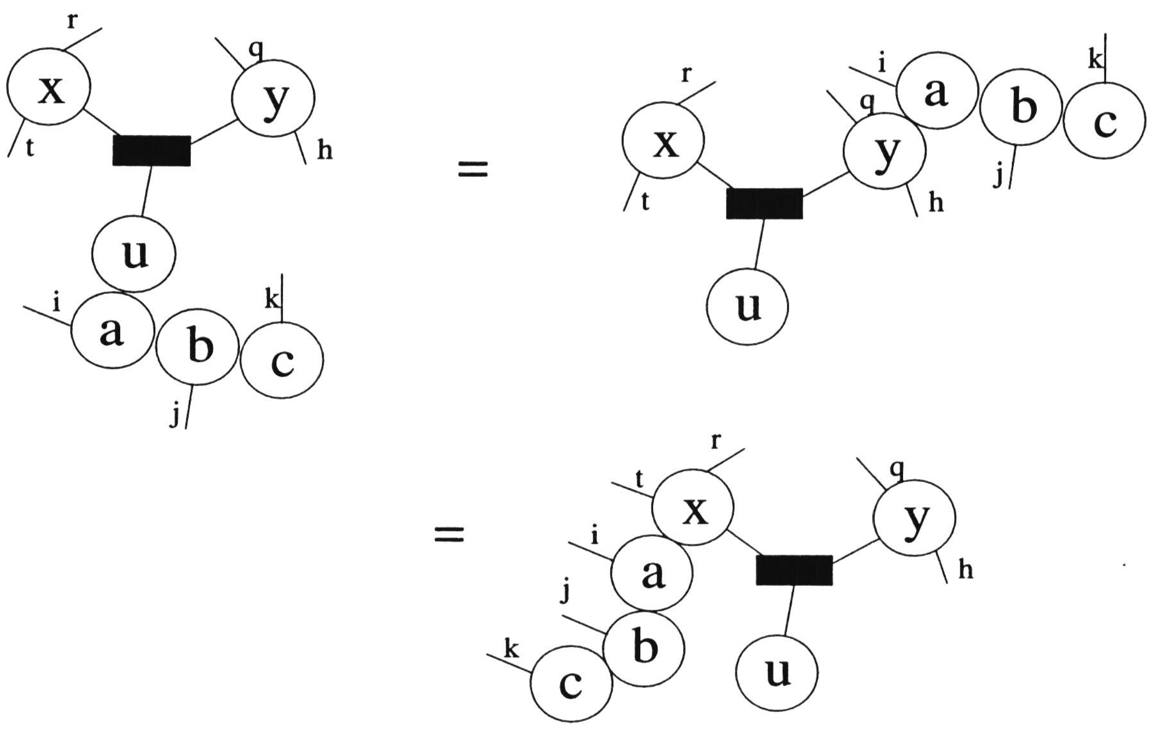


Figure 15:

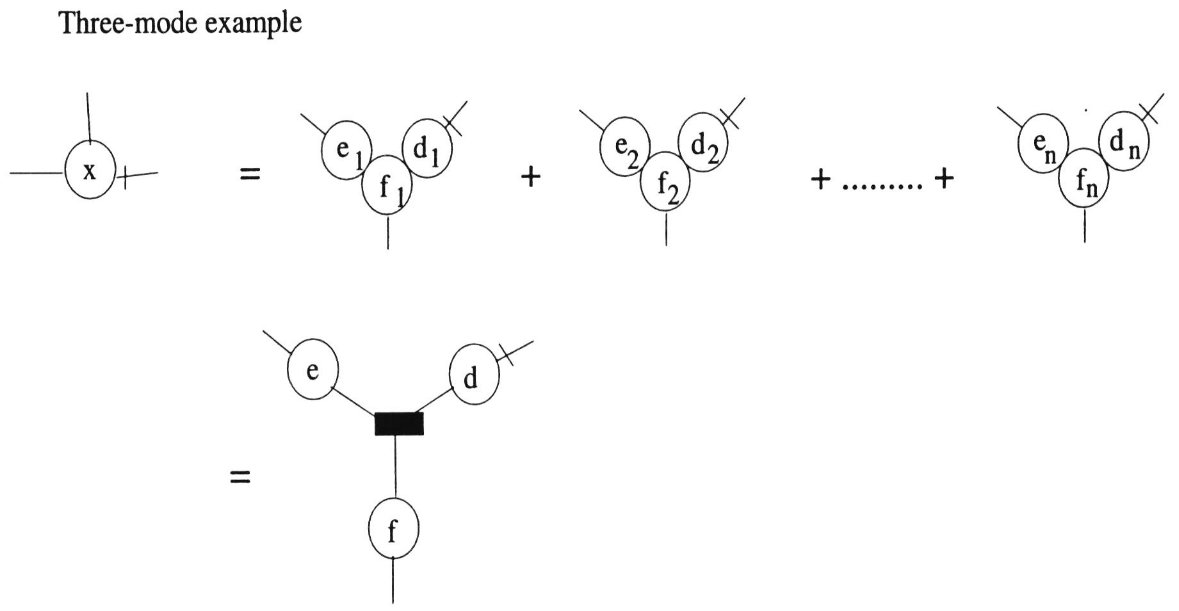


Figure 16:

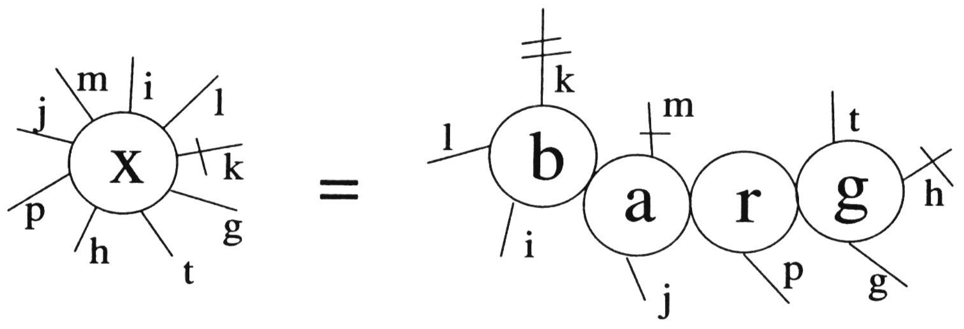


Figure 17:

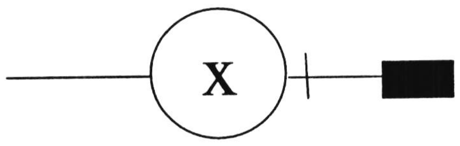


Figure 18:

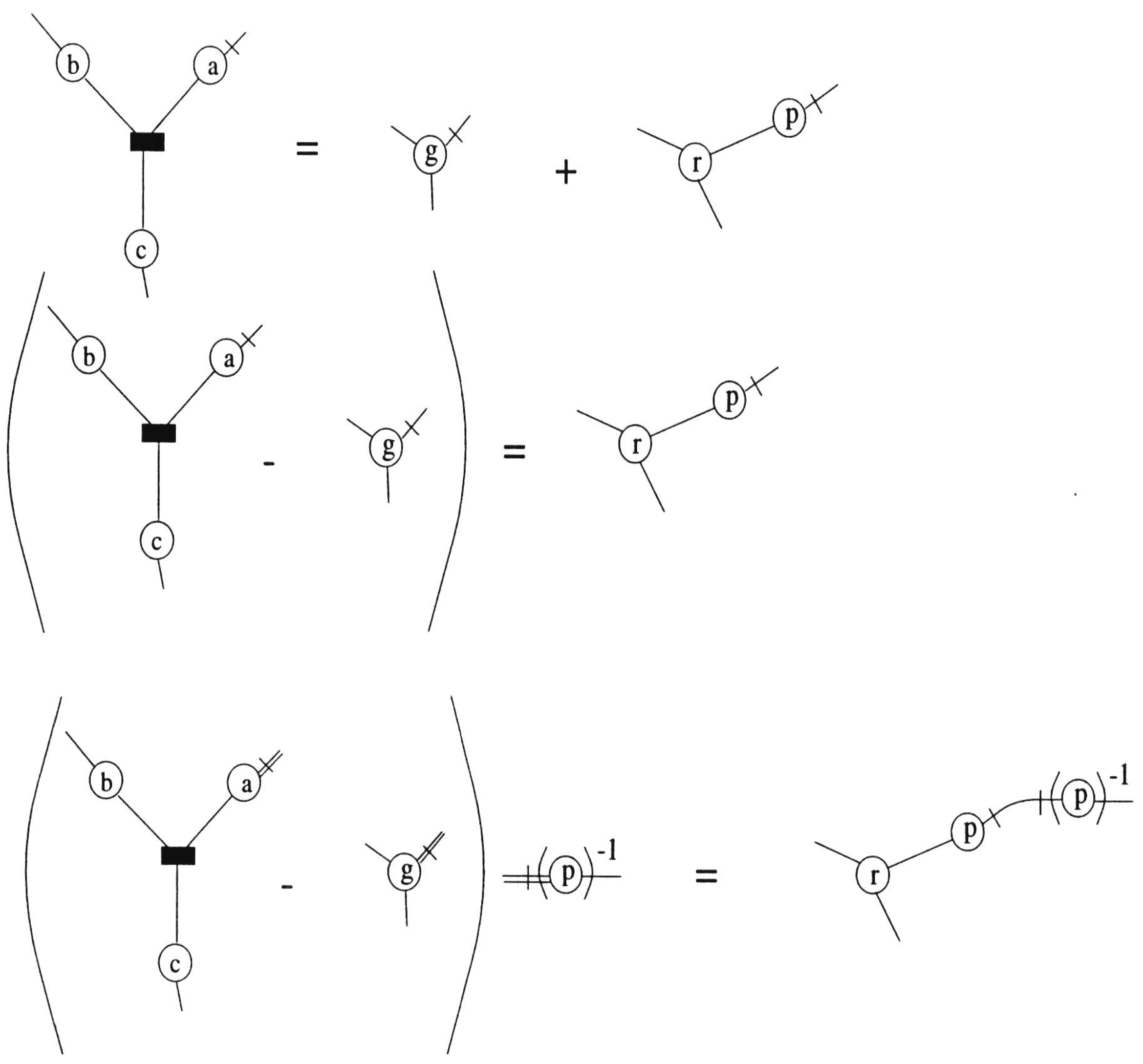


Figure 19:

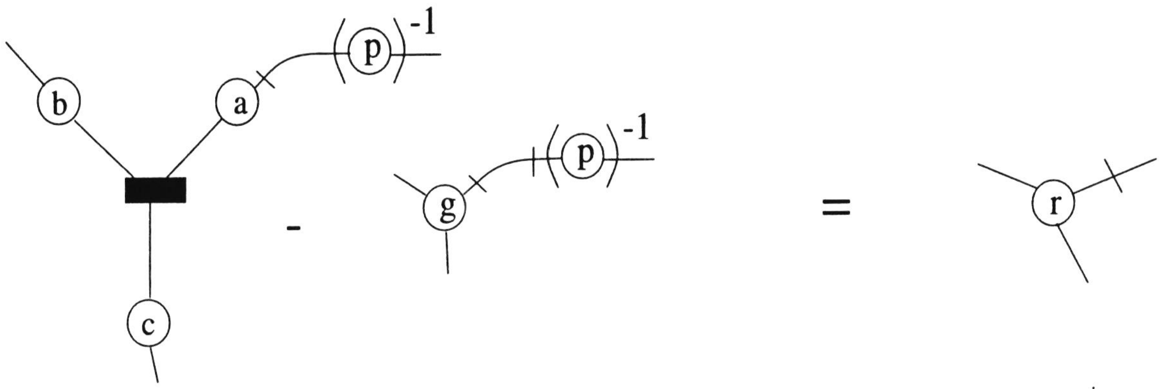


Figure 20:

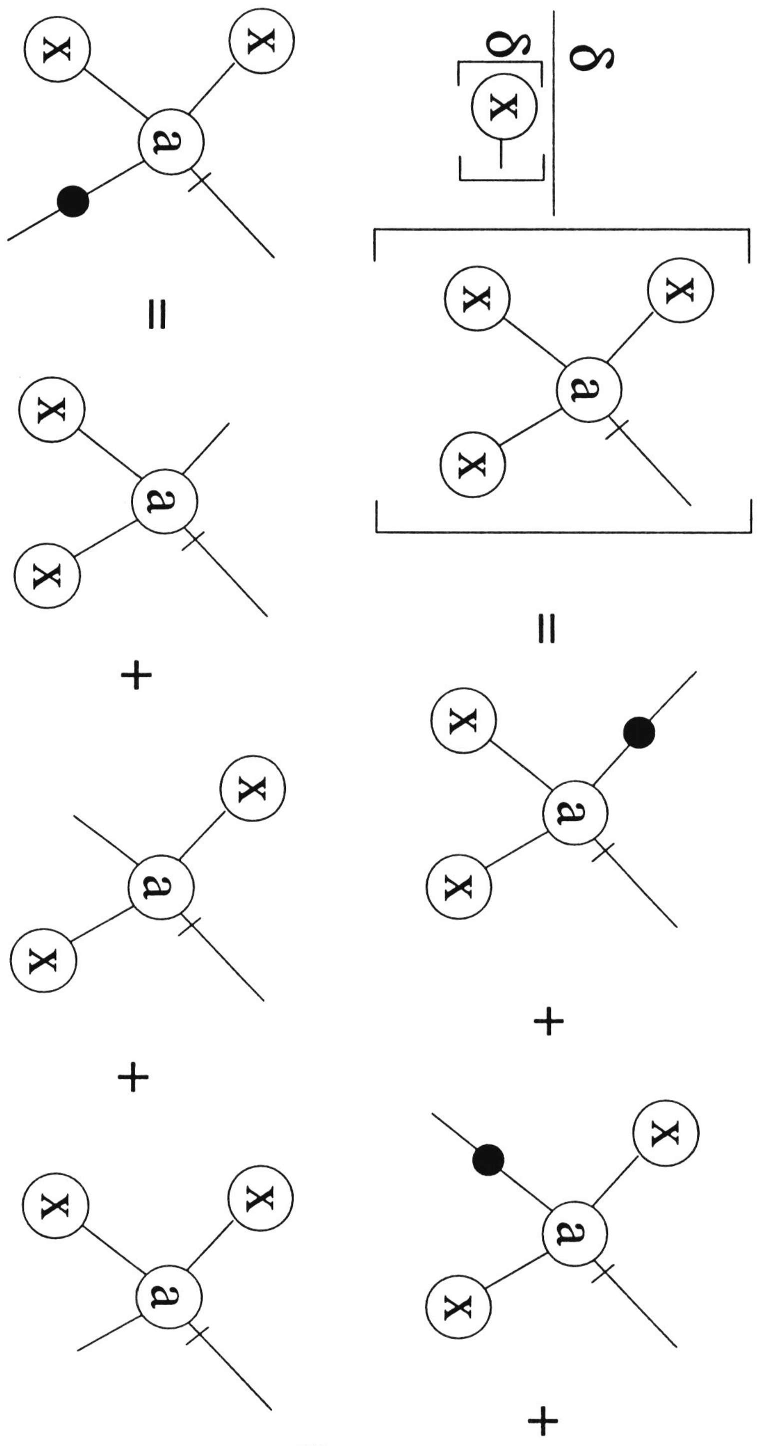


Figure 21:

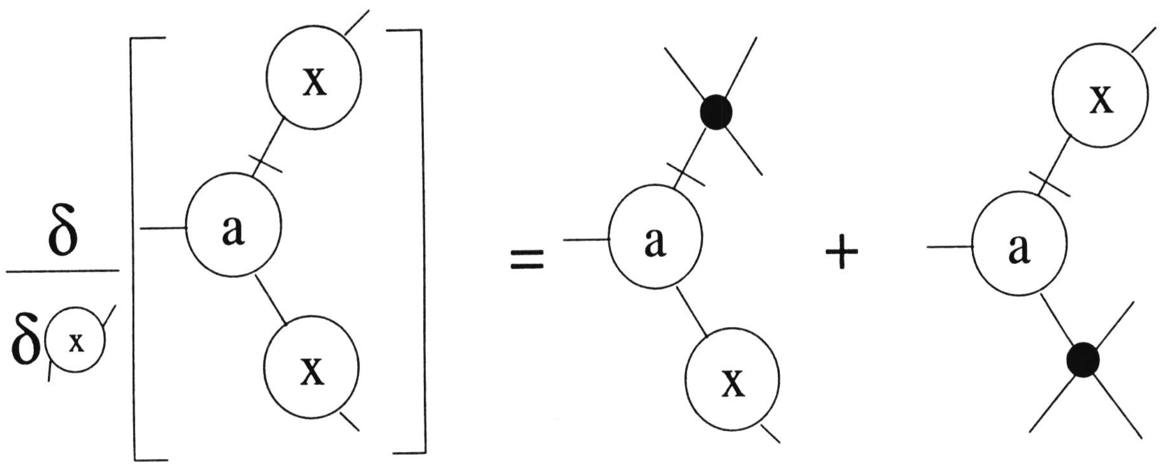


Figure 22:

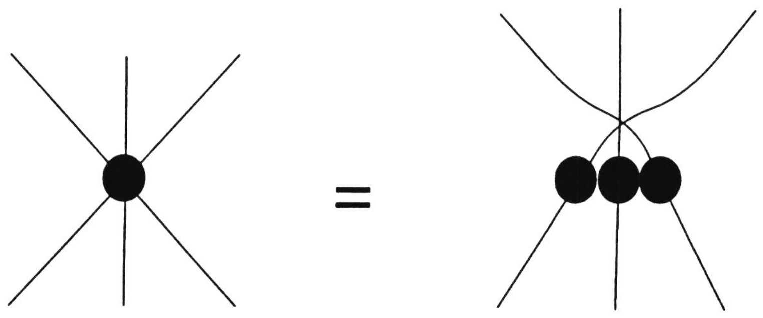


Figure 23:

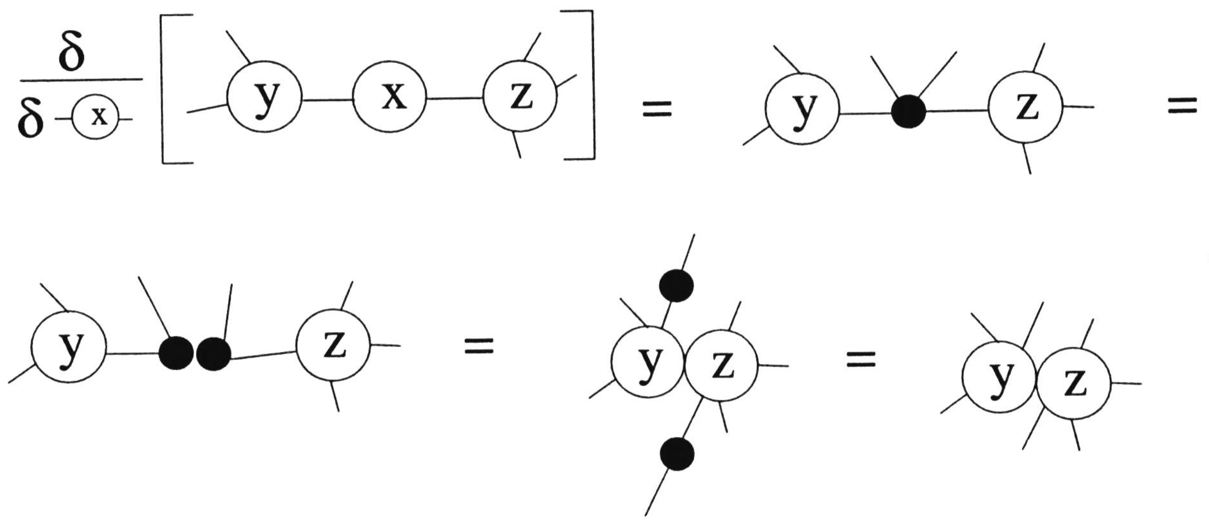


Figure 24:

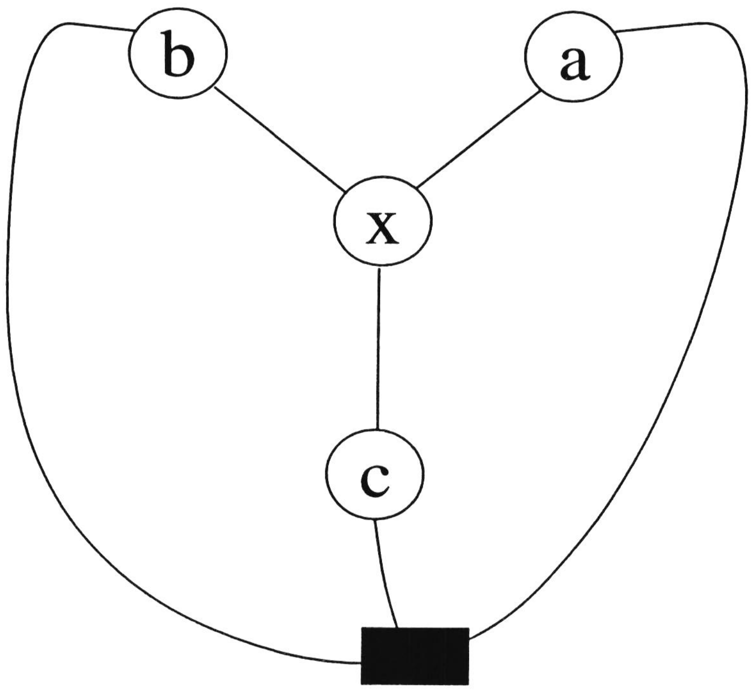


Figure 25:

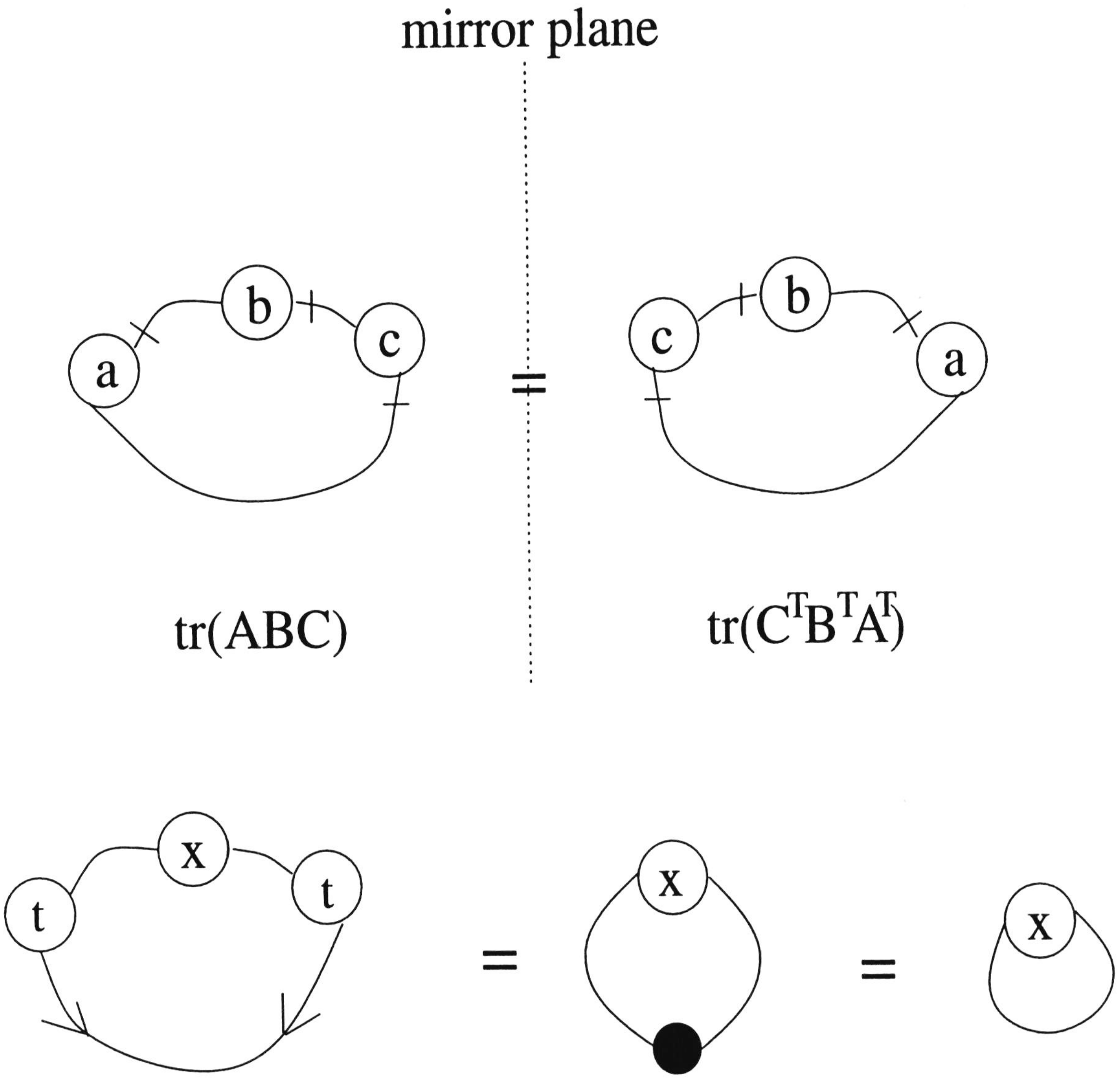


Figure 26:

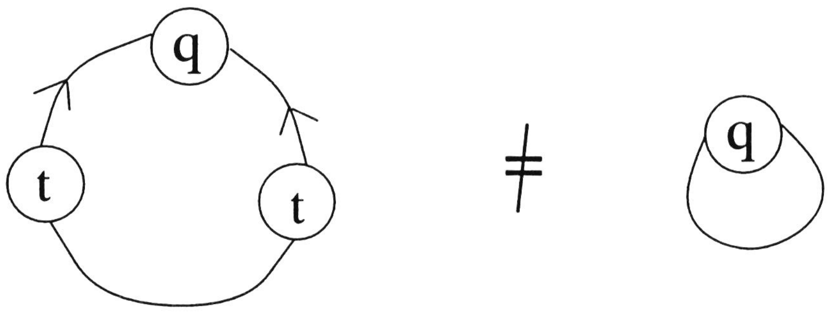


Figure 27:

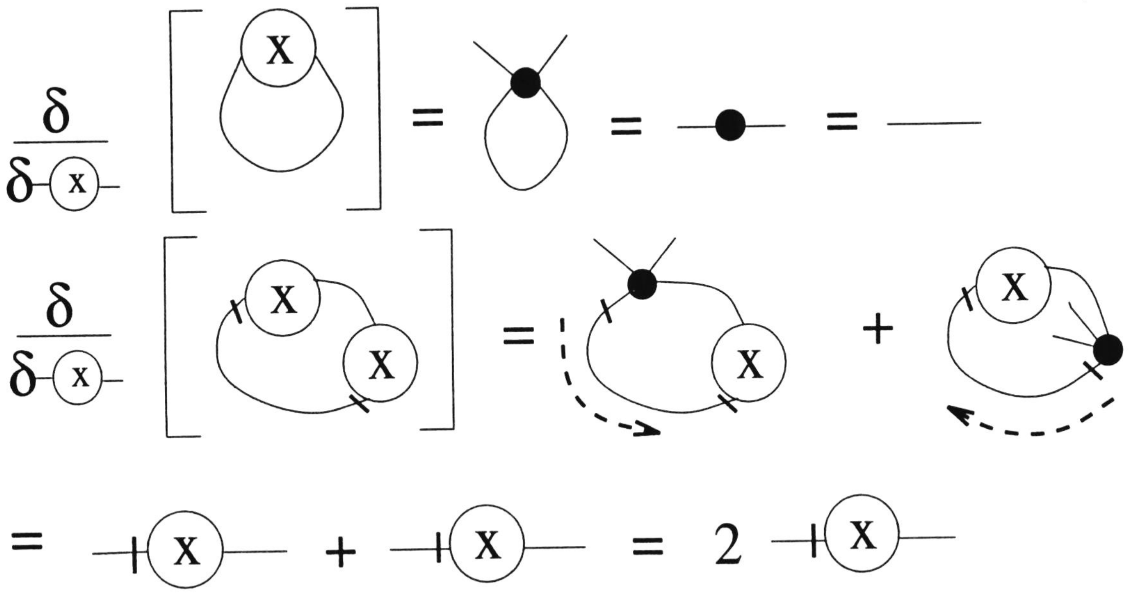


Figure 28:

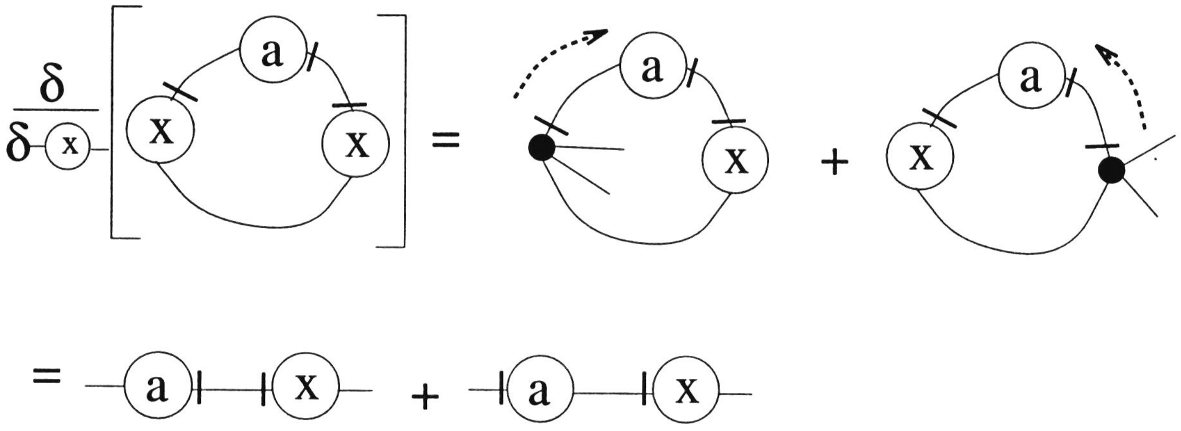


Figure 29:

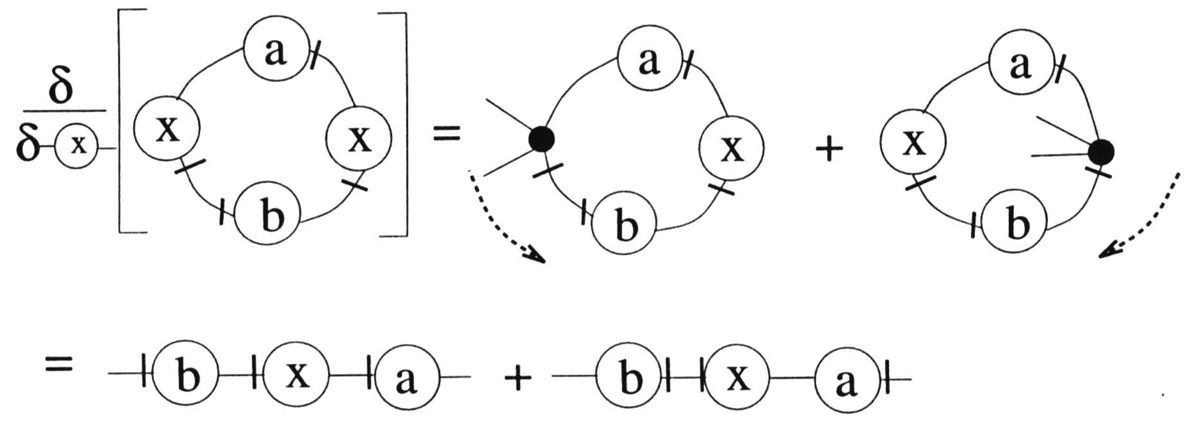


Figure 30:

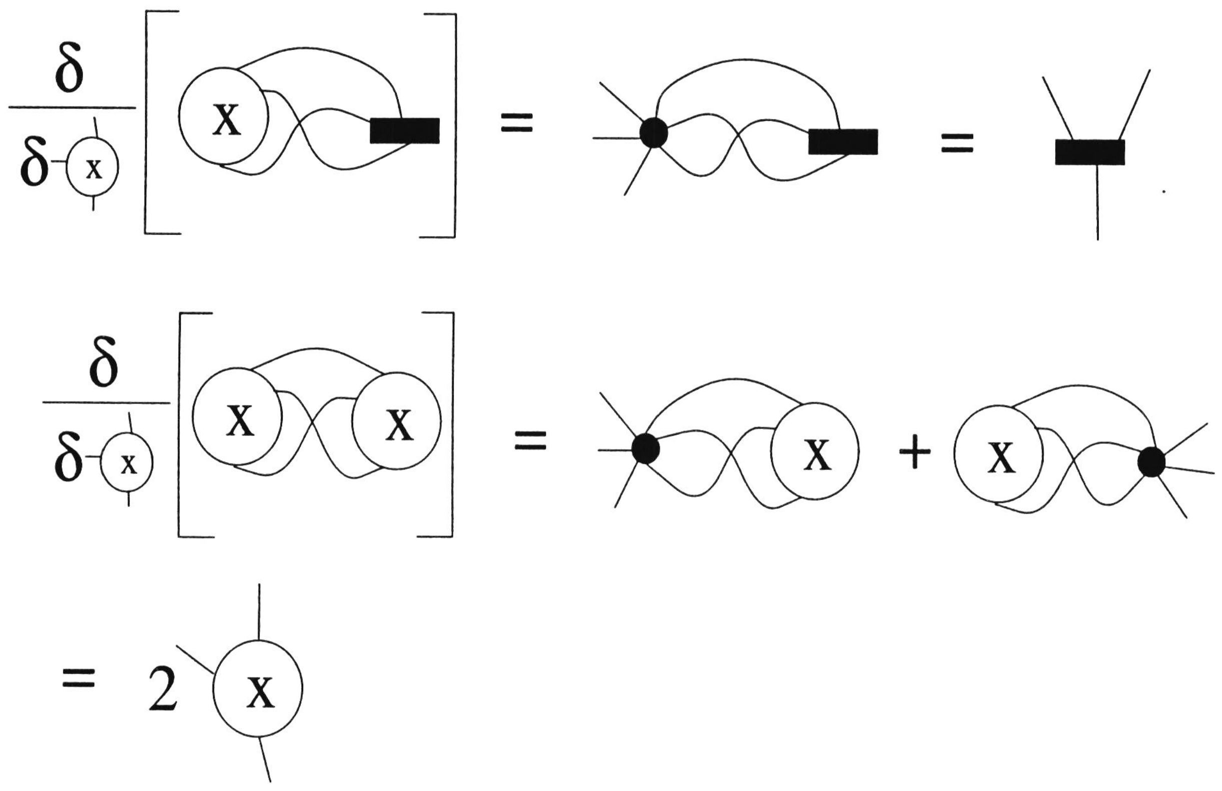


Figure 31:

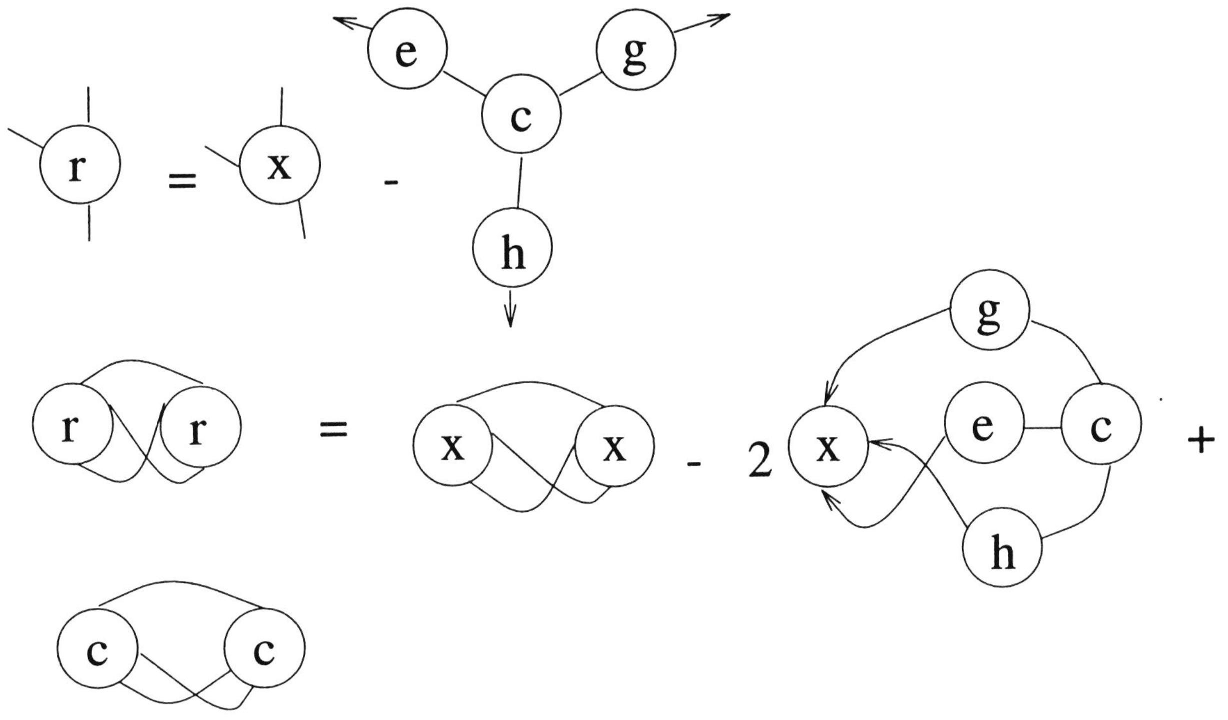


Figure 32:

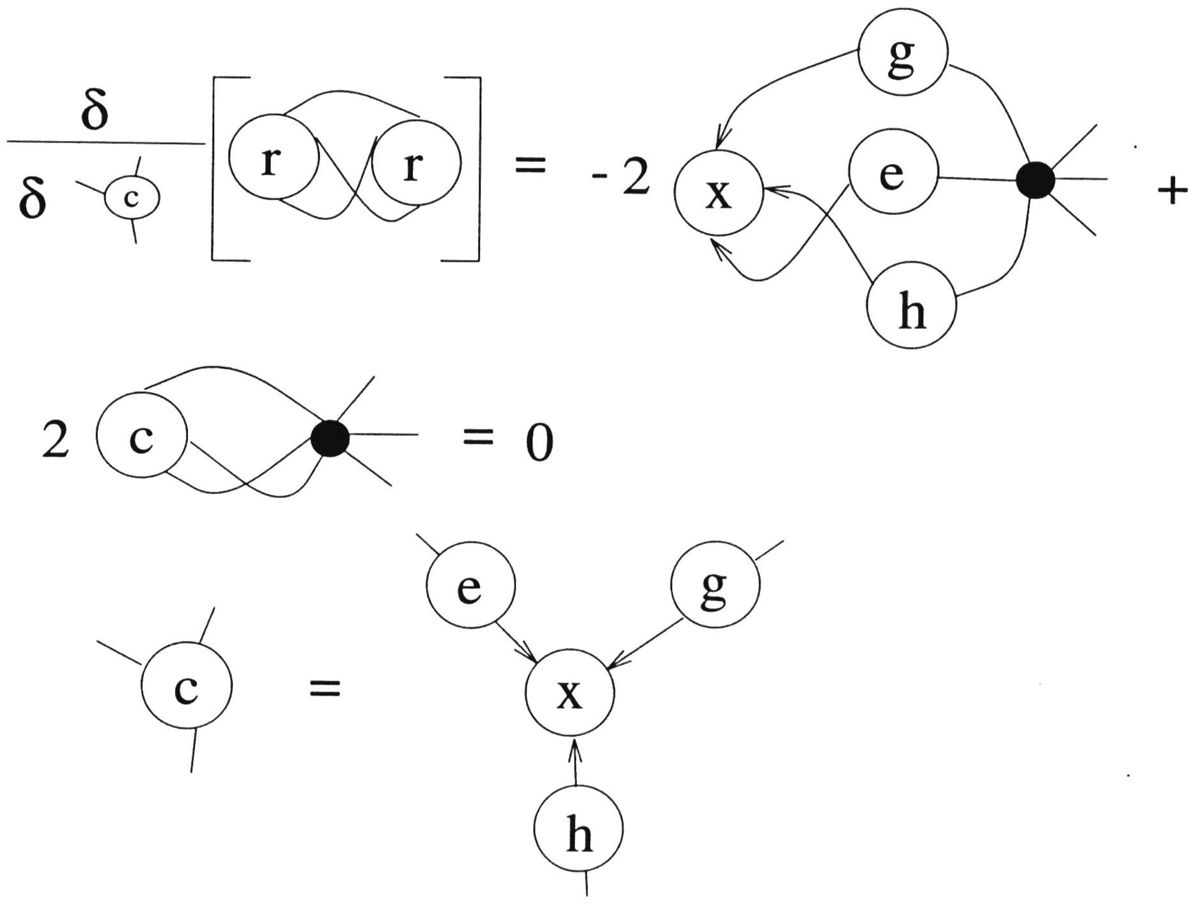


Figure 33:

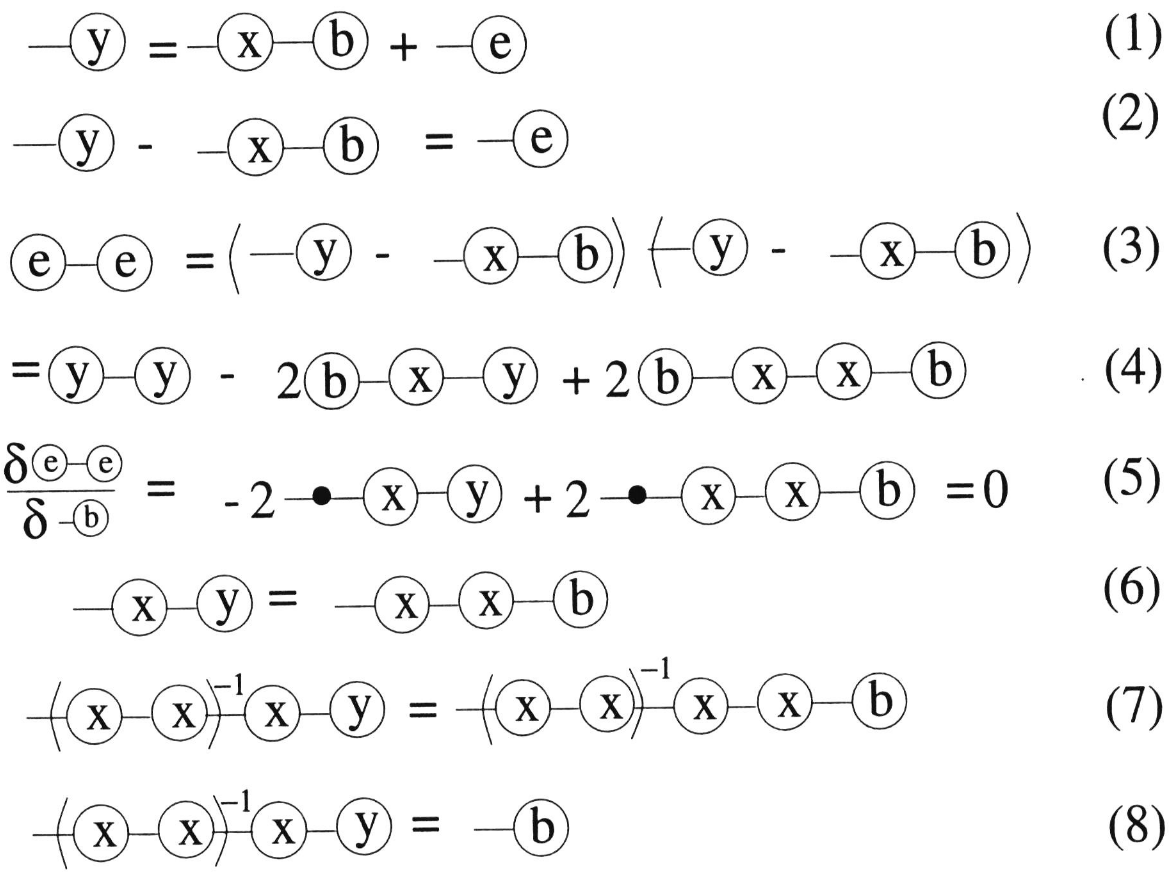


Figure 34:

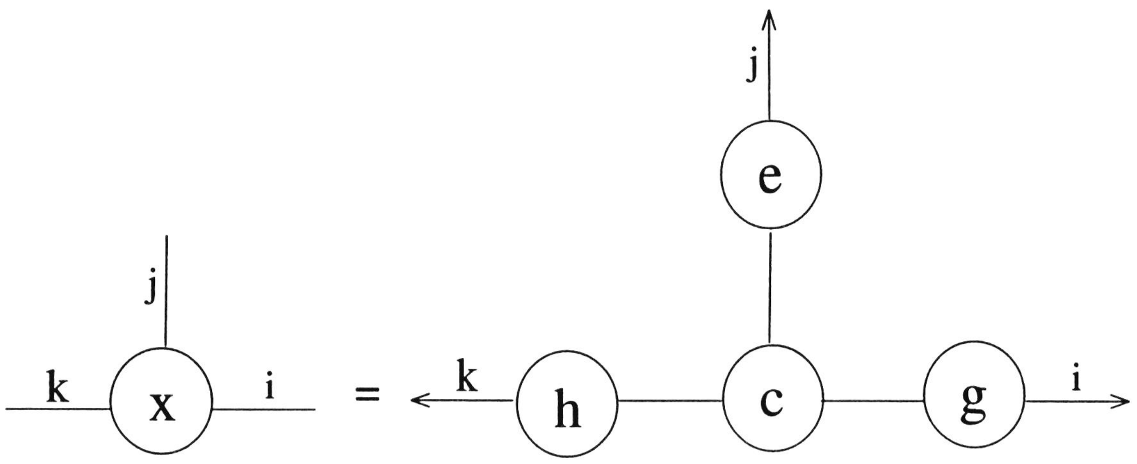


Figure 35:

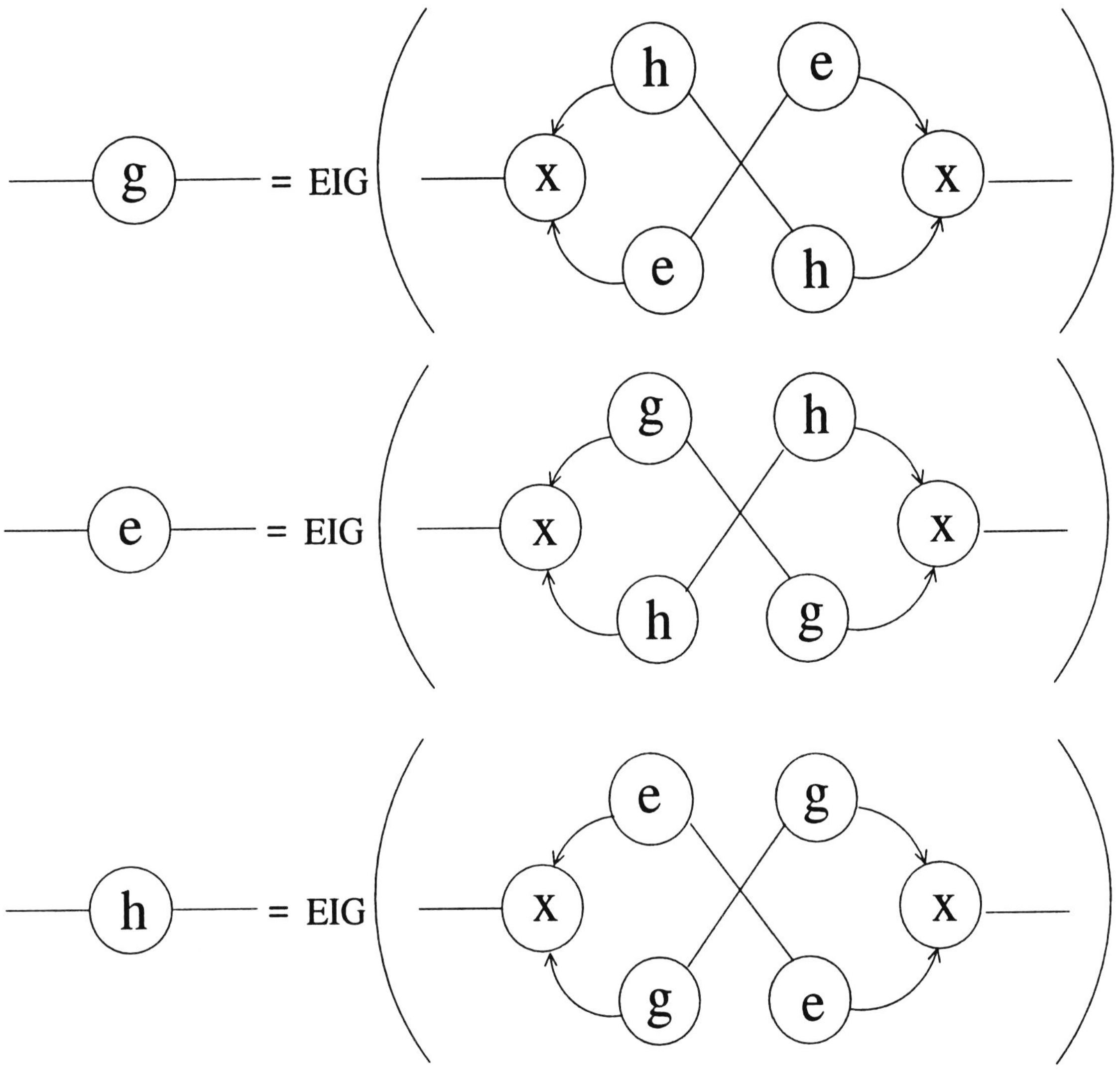


Figure 36:

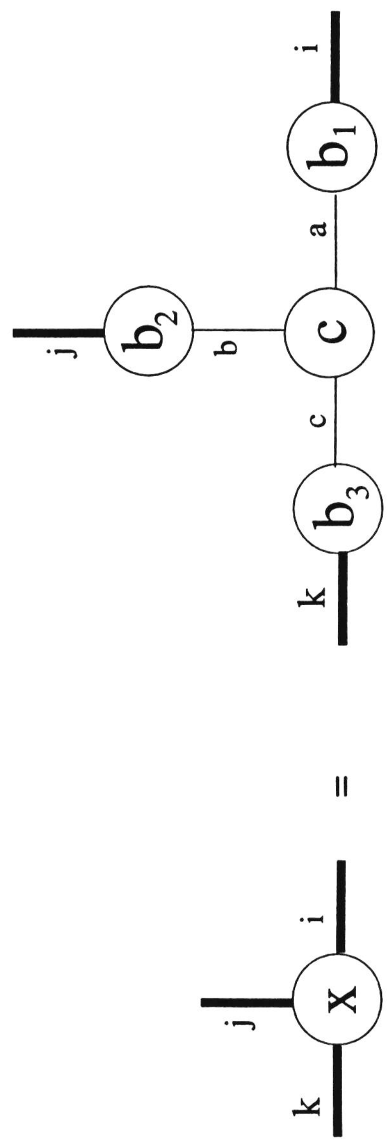


Figure 37:

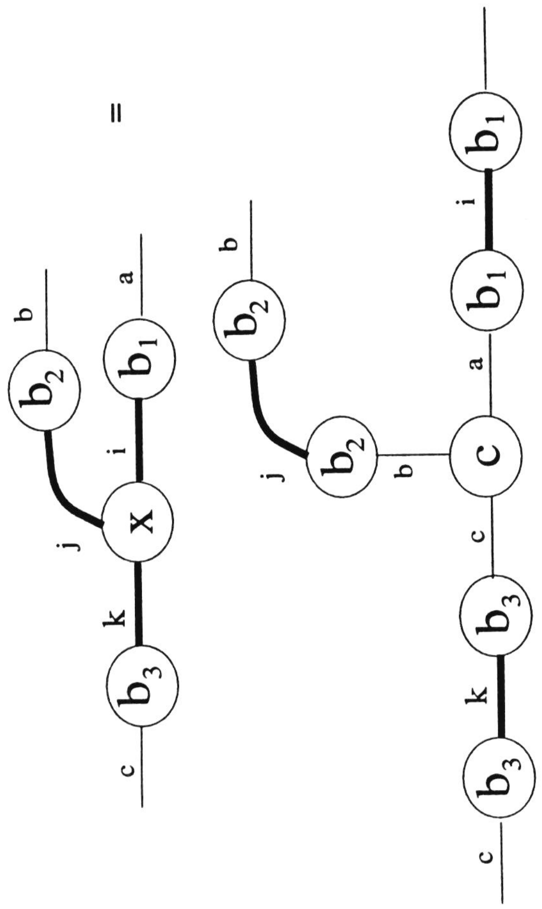


Figure 38:

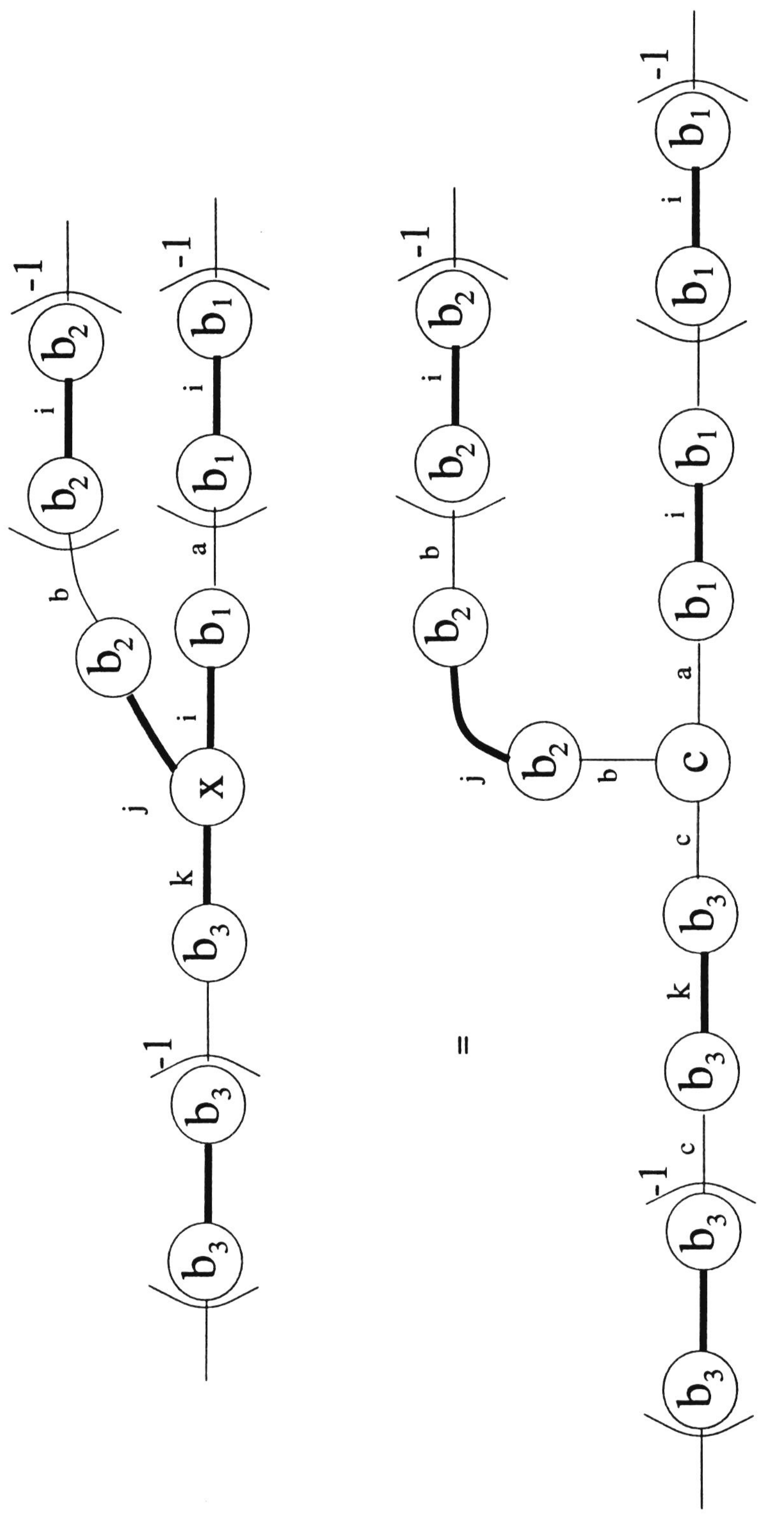


Figure 39:

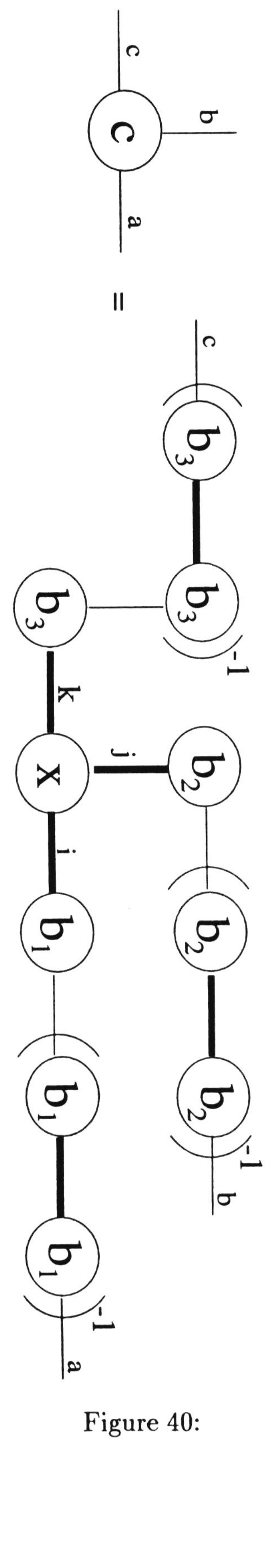
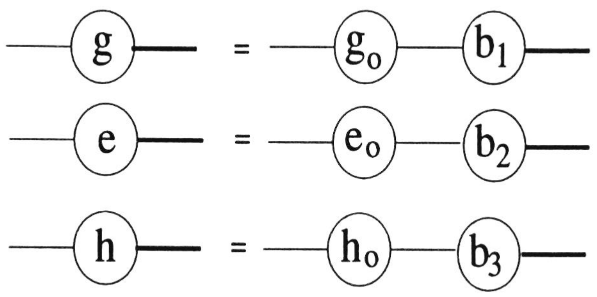


Figure 40:

The following is assumed:



Substituting the compression basis in loading matrices:

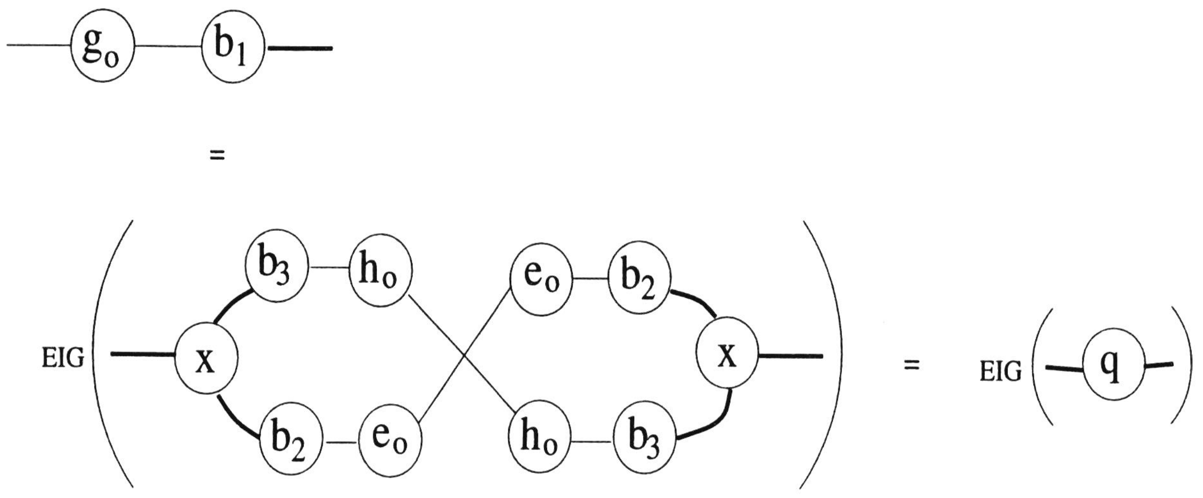


Figure 41:

Now we substitute X with the compression model:

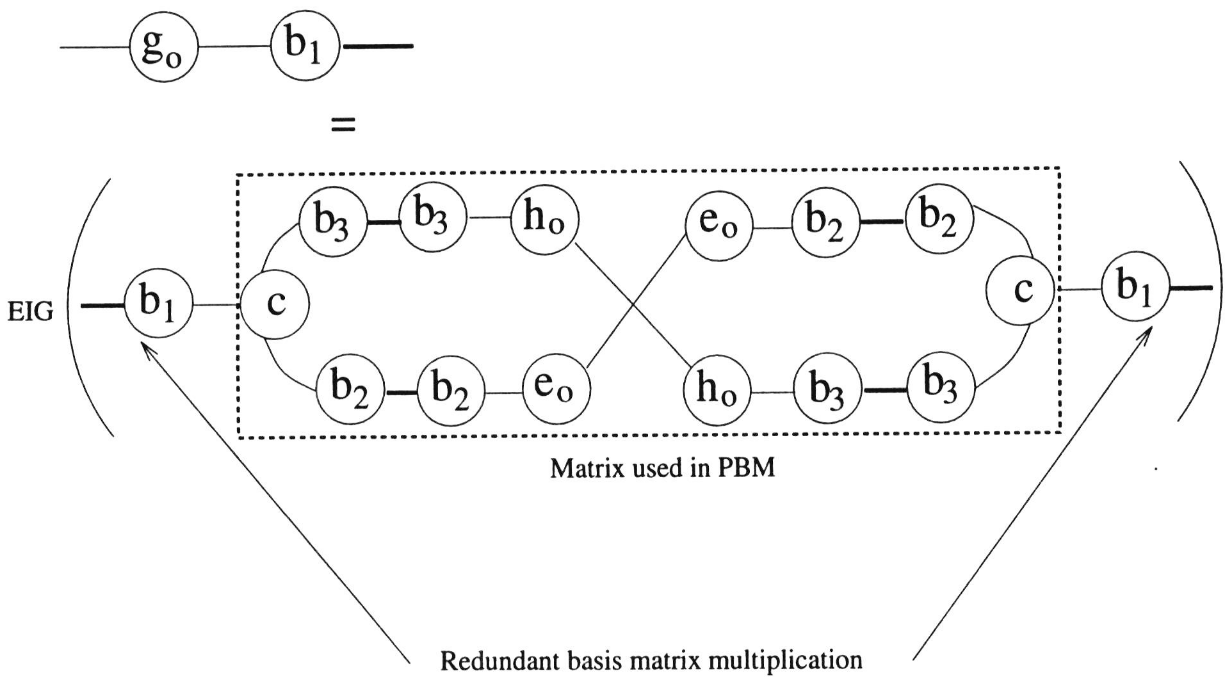


Figure 42:

Paper VII

Fast fuzzy c-means clustering of data sets with many features

Bjørn K. Alsberg

Department of Chemistry
University of Bergen
Allegt. 41
5007 Bergen
Norway
e-mail: `bjoern.alsberg@kj.uib.no`

Abstract

A fuzzy c-means clustering algorithm is presented which is much faster than the traditional algorithm for data sets where the number of features is significantly larger than the number of feature vectors. The algorithm is constructed by utilizing the covariance structure of feature vectors and cluster centers. By using results from a previous clustering, modified versions of the new algorithm achieve additional reductions in floating point operations.

Keywords

fuzzy c-means clustering, fast computation, grammian matrices, kernel matrices

Introduction

Classification using different types of clustering techniques are efficient tools for the chemist [1]. Hard clustering techniques such as MASLOC[2], minimal spanning tree [3] and Wards method [4] locate a feature vector (also referred to as an "object", see Figure 1 for explanation) into one class only. Fuzzy clustering techniques on the other hand, allow an object to have membership to more than one cluster. The need for fuzzy clustering occurs when objects tend to occupy positions in the feature space between more clearly defined clusters.

In spectroscopy analyses of data matrices containing whole or large parts of spectra are common. Such data matrices often contain large numbers of features or intensities compared to the number of objects. The standard fuzzy c-means (FCM) algorithm is unnecessarily computer demanding for such cases. In this article a reformulation of the FCM algorithm in terms of matrix algebra is presented which makes it very efficient for such data sets. This new algorithm is referred to as FCM-M. There is an algorithm called Approximate FCM (AFCM) [5] which is constructed to solve the problem when the number of objects is large compared to the number of features. But as the name of the algorithm indicates, it is approximate and does not give the identical results as if FCM was used on the data set. The FCM-M on the other hand, is *not* an approximation since it produces the same results as FCM.

The new algorithm makes it more practical to analyze objects described by e.g. full infrared or nuclear magnetic resonance (NMR) spectra. Objects described by N-dimensional (also referred to as N-*mode*) spectra contain even larger numbers of features. Several two- and three dimensional techniques are becoming common (e.g. hyphenated methods like GC-IR, MS-IR[6, 7] and 3- and 4-D NMR [8]) and it may be desirable to compare whole data arrays by fuzzy clustering. Such a comparison is possible by *unfolding* the different N-mode data arrays into feature vectors. Unfolding is here meant to be a reorganization of N-mode array elements into a single vector.

The use of large feature vectors is also of relevance to quantitative structure activity relationships (QSAR) studies. Molecular descriptors often originate from spectra and/or theoretical computations. In [9] the CoMFA method is described where the molecular structure is represented as a vector containing energies of steric and electrostatic interactions between the compound of interest and a "probe atom" placed at intersections of a regular three-dimensional grid. Depending on the size of molecules to be compared and the density of the grid, the number of features used to describe a molecular structure can be very large (e.g. hundreds of thousands of features). The number of molecules, however, is much smaller, e.g. between 10 and 100. The computer workload on such matrices using the fuzzy c-means clustering algorithm as presented in [10] is significant. If the fuzzy clustering is performed *once* on such data sets it may not differ whether the investigator waits five minutes or an hour for the result. Fuzzy clustering and other data analytical methods, however, are often used several times on the same data matrix with different parameter settings in order to detect the underlying structures of the data set. It is common in cluster analysis to search for the optimal number of clusters and the existence of subclusters. If the standard FCM algorithm is used on data matrices described above, the whole analysis becomes unnecessarily slow and tedious. Faster algorithms such as FCM-M also enables the investigator to quickly test different feature vector representations which otherwise would have been impractical.

Theory

The standard fuzzy c-means (FCM) algorithm

The FCM algorithm is started by obtaining an initial estimate of the matrix **V** containing the cluster centers as row vectors. The dimensions of the **V** matrix are $[K \times M]$ where K is the number of clusters and M is the number of features. The initial estimate of **V** can be produced by e.g. random perturbations of the mean vector of all the feature vectors. The feature vectors are stored as rows in the matrix **X**. The dimensions of **X** are $[N \times M]$ where N is the number of objects. The following steps are iterated until convergence:

$$u_{ik} = \left(\sum_{j=1}^{K} \left(\frac{d_{ik}}{d_{ij}} \right)^{2/(m-1)} \right)^{-1} \tag{1}$$

$$\mathbf{v}_k = \sum_{i=1}^{N} (u_{ik})^m \mathbf{x}_i \left(\sum_{i=1}^{N} (u_{ik})^m \right)^{-1} \tag{2}$$

where d_{ik} and d_{ij} are the A-distances

$$d_{ik} = (\|\mathbf{x}_i - \mathbf{v}_k\|_A)^{1/2} = ((\mathbf{x}_i - \mathbf{v}_k)^T \mathbf{A} (\mathbf{x}_i - \mathbf{v}_k))^{1/2}. \tag{3}$$

$\mathbf{v}_k$ is the k'th cluster center and $\mathbf{x}_i$ is the i'th feature vector. u_{ik} is the membership value of object i to cluster k. **A** is a positive definite matrix chosen to control the shape of the optimal clusters. In this article **A** is the identity matrix, i.e. for hyperspherical clusters. The iteration is stopped when the change in

$$J(\mathbf{V}) = \sum_{i=1}^{N} \sum_{k=1}^{K} u_{ik}^m \|\mathbf{x}_i - \mathbf{v}_k\|^2 \tag{4}$$

is below a certain value ϵ. The weighting constant m has the range $1 \leq m < \infty$. Typical values for m are in the range $1 < m < 3$.

The matrix formulation of the fuzzy c-means algorithm (FCM-M)

Distance equations expressed in matrix algebra

Different types of distance norms are used in cluster techniques. A much used distance measure is the Minkowski distance between object i and j:

$$d_{ij} = \left(\sum_{k=1}^{M} |x_{ik} - x_{jk}|^{w} \right)^{1/w} \quad , \quad i,j \in [1, \cdots, N] \tag{5}$$

where $w \in \{1, 2, 3, \cdots\}$ and x_{ik}, x_{jk} are elements in the matrix $\mathbf{X}$. In this article the Euclidean norm ($w = 2$) is always assumed.

The approach presented here is an application and extension of the formula for the Euclidean distance matrix presented in [11]. Let $\mathbf{D}$ be the matrix containing the distances between the row vectors in $\mathbf{X}$. The squared distances can be written as:

$$d_{ij}^2 = \sum_{k=1}^{M} (x_{ik} - x_{jk})^2 = \sum_{k=1}^{M} (x_{ik}^2 1_{jk} + 1_{ik} x_{jk}^2 - 2 x_{ik} x_{jk}) \tag{6}$$

where 1_{jk} and 1_{ik} denote matrix elements of the same matrix containing ones. This matrix is also written as $\mathbf{J}$ and has the same dimensions as $\mathbf{X}$. A vector containing ones only is written $\mathbf{1}_{[N \times 1]}$ where the subscript is used to denote the dimensions of the vector.

The matrix expression for equation 6 is :

$$\mathbf{D}^2 = \mathbf{X}^2 \mathbf{J}^T - 2\mathbf{X}\mathbf{X}^T + \mathbf{J} \left[\mathbf{X}^2 \right]^T \tag{7}$$

where the superscript 2 signify that each element in the matrix is raised to the power of 2. Equation 7 is more efficiently formulated as :

$$\mathbf{D}^2 = \mathbf{M} + \mathbf{M}^T - 2\mathbf{R} \tag{8}$$

where $\mathbf{R} = \mathbf{X}\mathbf{X}^T$ and $\mathbf{M} = \mathbf{1}_{[N \times 1]}\text{diag}(\mathbf{R})^T$. The *diag* operator generates a column vector of the diagonal of a square matrix. The $\mathbf{M}$ matrix is not symmetric.

In general we can write the following distance equation based on the Minkowski equation 5 using matrix algebra:

$$\mathbf{D}^w = \mathbf{M}_w + \mathbf{M}_w^T + \sum_{i=1}^{w-1}(-1)^i \begin{pmatrix} w \\ i \end{pmatrix} \mathbf{X}^{w-i}(\mathbf{X}^T)^i \tag{9}$$

where $\mathbf{M}_w = \mathbf{1}_{[N \times 1]}\text{diag}(\mathbf{X}^{w-1}\mathbf{X}^T)^T$ and $\begin{pmatrix} w \\ i \end{pmatrix}$ is the binomial function. Since the absolute value $|x_{ik} - x_{jk}|$ in equation 5 is raised to the power of w, equation 9 is true for *even* w only.

In the FCM algorithm the distances between objects and cluster centers are calculated, not between the objects themselves. This gives rise to an *asymmetric* distance matrix:

$$q_{ic}^2 = \sum_{k=1}^{M}(x_{ik} - v_{ck})^2 = \sum_{k=1}^{M}(x_{ik}^2 1_{ck} + 1_{ik}v_{ck}^2 - 2x_{ik}v_{ck}) \tag{10}$$

where q_{ic} is the distance between object i and cluster c and v_{ck} is an element in the matrix $\mathbf{V}$. By using the matrix result in equation 8, equation 10 can be written as:

$$\mathbf{Q}^2 = \mathbf{A} + \mathbf{B}^T - 2\mathbf{L} \tag{11}$$

where $\mathbf{A} = \mathbf{1}_{[N\times 1]}\text{diag}(\mathbf{F})^T$, $\mathbf{F} = \mathbf{V}\mathbf{V}^T$, $\mathbf{B} = \mathbf{1}_{[K\times 1]}\text{diag}(\mathbf{R})^T$, and $\mathbf{L} = \mathbf{X}\mathbf{V}^T$. $\mathbf{R}, \mathbf{L}$ and $\mathbf{F}$ are referred to as "kernel matrices" and have the following dimensions:

$$Dim(\mathbf{R}) = [N \times N] \tag{12}$$
$$Dim(\mathbf{L}) = [N \times K] \tag{13}$$
$$Dim(\mathbf{F}) = [K \times K] \tag{14}$$

The dimensions of $\mathbf{A}$ and $\mathbf{B}^T$ are both $[N \times K]$.

The main idea behind the new algorithm is to avoid the numerous distance computations (i.e. when M is large) for each iteration step and instead update just the smaller kernel matrices in each iteration.

The FCM-M algorithm

In the new FCM-M algorithm an estimate of the $\mathbf{V}$ cluster centers is initially produced.

The following steps are iterated:

$$\mathbf{Q} = \left(\mathbf{A} + \mathbf{B}^T - 2\mathbf{L}\right)^{1/2} \tag{15}$$
$$\mathbf{U} = \varphi(\mathbf{Q})^T \tag{16}$$
$$\mathbf{P} = \mathbf{U}^m \tag{17}$$
$$\mathbf{H} = \mathbf{P}^T\text{diag}(\mathbf{1}./(\mathbf{1}^T\mathbf{P}^T)) \tag{18}$$
$$\mathbf{L} = \mathbf{R}\mathbf{H} \tag{19}$$
$$\mathbf{F} = \mathbf{H}^T\mathbf{L} \tag{20}$$

./ is elementwise division where the dot follows the notation as used in MATLAB. $\mathbf{U}$ is the fuzzy membership matrix. The equations $\mathbf{L} = \mathbf{R}\mathbf{H}$ and $\mathbf{F} = \mathbf{H}^T\mathbf{L}$ constitute the updating steps of the $\mathbf{L}$ and $\mathbf{F}$ kernel matrices. The grammian matrix $\mathbf{R}$ is computed *once* and not updated. The function φ is formula 1 applied to the asymmetric distance matrix $\mathbf{Q}$; i.e. the matrix is transformed such that the sum of the membership values for one object satisfy:

$$\sum_{k=1}^{K} u_{ik} = 1 \, , \quad \forall \, i. \tag{21}$$

Note that in equation 1 the term $\|\mathbf{x}_i - \mathbf{v}_k\|^m$ corresponds to d_{ik}^m, so only entries from the calculated asymmetric distance matrix is needed. The iteration is stopped when the change in J (see equation 4) is below a certain value ϵ.

Efficiency of the FCM and FCM-M algorithms

The most common way of measuring the efficiency of a numerical algorithm is to obtain the number of *floating point operations* (FLOPS) used by the algorithm. Knowing the algorithm it is also possible to construct a *FLOPS formula* which estimates the FLOPS consumption for the algorithm given the dimensions of the different input matrices and the number of iterations. For the FCM and FCM-M algorithms the FLOPS formulas are expressed in terms of the dimensions of the input matrix $\mathbf{X}$ ($Dim(\mathbf{X}) = [N \times M]$), the number of clusters to be analyzed (K) and the number of iterations I. The number I varies according to the structure of data set $\mathbf{X}$ and it is not possible to obtain a perfect estimate in beforehand for a given stopping criterion ϵ.

The standard FCM algorithm has the following FLOPS formula (see appendix for a discussion of how to obtain FLOPS formulas):

$$F_1 = IK\,(7\,NM + 5\,KNM + 3\,KN + 4\,N + M) \tag{22}$$

The FCM-M algorithm has the following FLOPS formula:

$$\begin{aligned} F_2 &= NM(N+1) + 2\,KNM + KM(K+1) \\ &+ IK\left(14\,N + 11\,KN + 2\,N^2 + 1\right) \end{aligned} \tag{23}$$

The computation of **R** requires $NM(N+1)$ FLOPS, the computation of **L** requires $2\,KNM$ FLOPS and the computation of **F** requires $KM(K+1)$ FLOPS. These are the inital calculations of the kernel matrices. The saving in FLOPS using FCM-M (equation 23) is obtained from the term describing the FLOPS usage in the iteration part of the algorithm ($IK\ (14\,N + 11\,KN + 2\,N^2 + 1)$). This term does not include M which is assumed to be large. In the FLOPS formula for the standard FCM algorithm (equation 22) we see that M is involved in each iteration I.

In order to compare the two different methods a small example using the two equations 22 and 23 is presented. It is assumed that the number of objects is constant, $N = 20$. In addition, the number of iterations ($I = 15$) and the number of clusters ($K = 4$) are constants also . The number of features varies from 500 to 9500.

The FLOPS consumptions for the two algorithms in this particular example are presented in Table 1 and in Figure 2. This table clearly shows that FCM requires significantly more MFLOPS (MegaFLOPS) than FCM-M. It is also interesting to observe the FLOPS ratio $r = \frac{F_1}{F_2}$ between the two algorithms, i.e. the number of FLOPS of FCM divided by the number of FLOPS consumed by the FCM-M. FCM-M is of course useful only when $r > 1$. For each M the FLOPS ratio is calculated and located in the fourth column of Table 1. It is evident from the table that the FLOPS ratio converges to an upper limit. This upper limit is referred to as r_{max}. Using the two equations for the FLOPS consumptions of FCM (equation 22) and FCM-M (equation 23) algorithms it is possible to express r_{max} as a function of N, K and I. By ignoring parts of the equations not containing M (assuming they are much smaller than M) the following equation gives an approximate estimate of r_{max} :

$$r_{max} = \frac{F_1}{F_2} \approx \frac{IK(7N + 5KN + 1)}{(K + N + 1)(K + N)} \tag{24}$$

For the example shown in Table 1 the estimated r_{max} is 53. This implies that the FCM-M for this setting of K, N and I can never be more than 53 times faster than FCM. For small values of I and K and very large value of N there is little difference between the two algorithms and the use of FCM-M will not be efficient. In a real analysis, however, the r has either a larger r_{max} than indicated in equation 24 or is proportional to M since several clusters must be tested for. This is discussed in the next section. The FCM-M algorithm can work with the kernel matrices from a previous clustering as a start and therefore avoid unnecessary computations.

Improved speed in multiple cluster analyses and sub-cluster determination

In a typical fuzzy clustering analysis, the investigator tries to find the optimal number of clusters. This implies that the fuzzy clustering algorithm is used for different values of K. Starting a clustering using the FCM-M algorithm without taking into consideration earlier analyses is inefficient since it is possible to utilize start kernel matrices from a previous clustering. The initial estimate of the set of kernel matrices is in fact the largest contributor to the FLOPS usage in the FCM-M algorithm. The following two schemes are presented which use kernel matrices of a previous clustering:

- The **R** matrix from an earlier computation is used in all the other cluster analyses. It is a waste of computer resources to calculate **R** more than once for the same data set. The earlier clustering can be for any $K \geq 2$. Both the initial estimates of the kernel matrices **L** and **F**, however, are computed. This algorithm is referred to as FCM-M2. It should be noted that **R** may be obtained from other algorithms which have used the full grammian matrix of the data set. If a principal component analysis has been performed earlier on the data set, the grammian matrix from this analysis can be used again in the fuzzy clustering. The FLOPS equation for this algorithm is:

$$F_{2M2} = \begin{cases} F_2 & K = k \\ (F_2 - NM(N+1)) & K \neq k \end{cases} \tag{25}$$

where $K = k$ is the *first* clustering. F_2 is the FLOPS equation 23 for FCM-M.

- It is sometimes possible for an investigator to set an *upper limit* on the number of possible clusters in the data set. Let this upper limit be κ. If the *first* analysis is for κ clusters, the initial kernel matrices $\mathbf{L}_\kappa$ and $\mathbf{F}_\kappa$ can be used for all computations of clusters $K < \kappa$. The $\mathbf{L}_K$ and $\mathbf{F}_K$ matrices are generated by deleting the appropriate number of columns/rows in $\mathbf{L}_\kappa$ and $\mathbf{F}_\kappa$. Thus no FLOPS are needed for the estimation of the initial kernel matrices in this algorithm. This algorithm is referred to as FCM-M3. The FLOPS equation for this algorithm is

$$F_{2M3} = \begin{cases} F_2 & K = \kappa \\ (IK(14N + 11KN + 2N^2 + 1) & K < \kappa \, . \end{cases} \tag{26}$$

The FCM-M3 achieves the greatest reductions in FLOPS but has the disadvantage that some information about the upper limit of interesting clusters of a data set must be known.

The FCM-M2 algorithm does not achieve as much FLOPS reductions as FCM-M3, but does not require that the investigator starts with a specific $K = \kappa$. Often it is efficient to start with $K = 2$ and subsequently increase the number of clusters to be tested for. The r_{max} formula for FCM-M2 is approximately:

$$r_{max} \approx \frac{I(7N + 5KN + 1)}{2N + K + 1} \,. \tag{27}$$

This equation will give rise to higher values of r_{max} than equation 24.

It is customary to analyze certain subsets of objects to discover subclusters. Subclustering is straightforward with the new FCM-M algorithm. The rows and columns corresponding to objects not in the subcluster are removed from matrix $\mathbf{R}$. In $\mathbf{L}_K$ the corresponding rows are removed. It is not necessary to change $\mathbf{F}_K$.

Experimental

The following two-dimensional infrared data set has many variables compared to the number of objects and is here used as an example to illustrate the efficiency of the new FCM algorithms. The detailed cluster structure of this data set is not of importance for the current discussion and is therefore not presented.

MATLAB [12] was used in all the experiments.

Data set

The data set in the analysis is a pyrolysis carried out using a heating rate of 5^oC/min between 200 and 400^oC. After the termination of the pyrolysis the 80 sample spectra were background corrected with spectra of pure KBr taken at the same conditions, and then Kubelka-Munk transformed. The spectral range was $4000 - 650$ cm^{-1} and the optical resolution was set to 8 cm^{-1}. The data point resolution was 3.85 cm^{-1}. Based on these settings a data set of size $[80 \times 869]$ was obtained.

More details about this data set is found in [13].

Results

Experiment 1

The data set was first analyzed for $K = \{2,3,4,5\}$ using FCM. For all the experiments, m as described in the theory section was set to two. All algorithms used the same initial $\mathbf{V}$ matrix. This provided a setting where the different algorithms could be compared. A number of iteration steps I was required for each K before convergence ($\epsilon = 0.0001$). The values of the different I's were $\{14(K = 2), 24(K = 3), 44(K = 4), 58(K = 5)\}$. Each iteration number I needed for the estimates of the FLOPS usages was inserted into the formulas 22 and 23. The *measured* MFLOPS consumptions for FCM are presented in the first row in Table 2. The estimated number of MFLOPS are indicated in parentheses. From the partition coefficients (see last row in Table 2) it is suggested that $K = 2$ is the optimum number of clusters for this particular example. The partition coefficient is defined as $P(\mathbf{U}) = \frac{1}{n}tr(\mathbf{U}\mathbf{U}^T)$ where $\mathbf{U}$ is the fuzzy membership matrix.

The FLOPS ratio r (FCM/FCM-M) is increasing for increasing number of clusters K as is expected. For $K = 2$ the FCM-M algorithm is approximately five times faster but over fifty times faster for $K = 5$. The estimated r_{max} for this data set (assuming M increases to infinity) is 5.6 for $K = 2$, 18.2 for $K = 3$, 53.3 for $K = 4$ and 101.6 for $K = 5$.

Experiment 2

In this experiment the effect of not computing the kernel matrices was investigated by using the FCM-M2 and FCM-M3 algorithms. For the FCM-M2 algorithm it was assumed that $K = 2$ was the first clustering. An additional significant reduction in MFLOPS compared to the FCM-M algorithm was observed. The FLOPS ratio for $K = 3$ is much larger (FCM-M2 is 68.9 times faster than FCM compared to 15.3 times faster for FCM-M). The estimated r_{max} of FCM-M2 for this data set (assuming M increases to infinity) is 257.7 for $K = 3$, 576.3 for $K = 4$ and 894.8 for $K = 5$.

When the FCM-M3 algorithm can be used, this is definitely the best method. The results in Table 2 indicate this, but this fact can be better illustrated by looking at the FLOPS ratio when M increases to infinity. The following equation is obtained:

$$r \approx \frac{IKM(7N + 5KN + 1)}{IK(14N + 11KN + 2N^2 + 1)} \approx M. \tag{28}$$

This equation shows that the FLOPS ratio r is proportional to the number of features M, i.e. there is no upper limit r_{max} for this algorithm.

Discussion

The efficient analysis of kernel matrices instead of the original large data array as used here in the FCM-M algorithm has been applied with success to other algorithms as well. Principal component analysis (PCA) [14] can be formulated as an eigenvector decomposition of the matrices $\mathbf{X}^T\mathbf{X}$ or $\mathbf{X}\mathbf{X}^T$. Partial least squares regression (PLS) [15] is another algorithm which can also be formulated in terms of kernel matrices [16]. For both PCA and PLS it is straightforward to construct algorithms for data matrices containing many features and few objects or many objects and few features. The situation is not that symmetric for fuzzy c-means clustering. It is very difficult to find a correspondingly efficient FCM algorithm for the case when the number of objects is much larger than the number of features. For this situation there are algorithms which finds the *approximate* solution [5]. The most difficult case is when both the numbers of objects *and* features are large. One possible approach which *may* be efficient in some instances is the use of *compression* along the object mode/direction. This is also an approximate solution to the problem. Compression is suitable for cases when there is a functional relation between the feature vectors. Examples of such data sets are spectra evolving in time [17] or with respect to temperature (as the data set used in the present article). Such data sets can be compressed using e.g. spline techniques [18] and the coefficient matrix which has a smaller number of objects can be analyzed efficiently with the FCM-M algorithms.

Acknowledgements

The author wishes to thank Tormod Næs at University in Oslo for helpful discussions on the FCM algorithm , Jostein Toft and Olav M. Kvalheim at University of Bergen for providing valuable comments on the manuscript. The Royal Norwegian Council for Scientific and Industrial Research (NTNF) is thanked for financial support.

FLOPS estimations

In order to obtain a measure of the efficiency of algorithms the `flops` command in MATLAB is employed. All the FLOPS equations presented in this paper are based on the results from this command. In Table 3 a few examples are given where the FLOPS equations for simple matrix expressions are presented. For each matrix expression in an algorithm the corresponding FLOPS formula is found. The total FLOPS consumption is obtained by summing over the different contributions. Only the most important parts of an algorithm are investigated. All **WHILE** loops are changed to **FOR** loops to determine the FLOPS consumption for such steps.

As an example a small part of a MATLAB program is presented. This is a **FOR** loop which iterates I steps. At the right margin for each MATLAB statement the FLOPS usage for each step is provided. "%" defines the beginning of a comment in the MATLAB language.

```
for i = 1 : I
H = P'*D;  %f1 = 2*N^2*K
F = H'*L;  %f2 = 2*K^2*N
end
```

The total number of FLOPS for this simple algorithm is $I(f1 + f2) = 2INK(N+K)$. The same procedure has been applied to the FCM, FCM-M, FCM-M2 and the FCM-M3 algorithms presented in this paper.

Figure captions

Figure 1: This figure explains "objects" and "features" together with the dimensions of the original data matrix $\mathbf{X}$ and the fuzzy membership matrix $\mathbf{U}$.

Figure 2: This figure depicts table 1 in a $\log(F_1 - F_2)$ plot. The figure also shows the results for $K = 2, 3, 5$ which are not included in the table.

Table captions

Table 1: Table showing the MFLOPS usage of FCM and FCM-M based on the FLOPS equations for the two algorithms. In this example $I = 15$, $K = 4$ and $N = 20$. r is the FLOPS ratio which cannot be larger than 53 for this particular example.

Table 2: Table showing the MFLOPS usage for the FCM and FCM-M algorithms using the IR data set. The numbers in parentheses indicate theoretical estimations of the FLOPS usage. r is used to indicate the FLOPS ratios. In the estimations and observations of the MFLOPS the computations of the $J(\mathbf{V})$ are not included.

Table 3: Table showing the number of FLOPS required for some example matrix operations.

References

[1] Massart, D.; Kaufman, L. *The interpretation of analytical chemical data by the use of cluster analysis.* Chemical Analysis, A series of monographs on analytical chemistry and its applications, vol. 65. John Wiley & Sons, New York, 1983.

[2] Massart, D.; Lauwereys, M.; Lenders, P. *J. Chromatogr. Sci.*, **1974**, *12*, 617.

[3] Kruskal, J. *Proc. Am. Math. Soc.*, **1956**, *7*, 48–50.

[4] Ward, J. *J. Am. Stat. Ass.*, **1963**, *58*, 236–244.

[5] Cannon, R.; Jitendra, V.; Bezdek, J. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **1986**, *PAMI-8*(2), 249–255.

[6] Hirschfeld, T. *Analytical Chemistry*, **1980**, *52*, 297A–310A.

[7] Hirschfeld, T. *Science*, **1985**, *230*, 286–291.

[8] Pelczer, I.; Szalma, S. *Chem. Rev.*, **1991**, *91*, 1507–1524.

[9] Cramer, R.; Patterson, D.; Bunce, J. *Journal of American Chemical Society*, **1988**, *110*, 5959–5967.

[10] Bezdek, J.; Coray, C.; Gunderson, R.; Watson, J. *SIAM J. Appl. Math.*, **1981**, *40*, 339–357.

[11] Alsberg, B.; Esbensen, K. *Chemom. Intell. Lab. Syst.*, **1992**, *16*, 127–137.

[12] *MATLAB Reference Guide.* Natick, Mass. The MathWorks Inc., 1992.

[13] Alsberg, B.; Nodland, E.; Kvalheim, O. *J. Chemom.*, **1994**, *8*(2), 127–146.

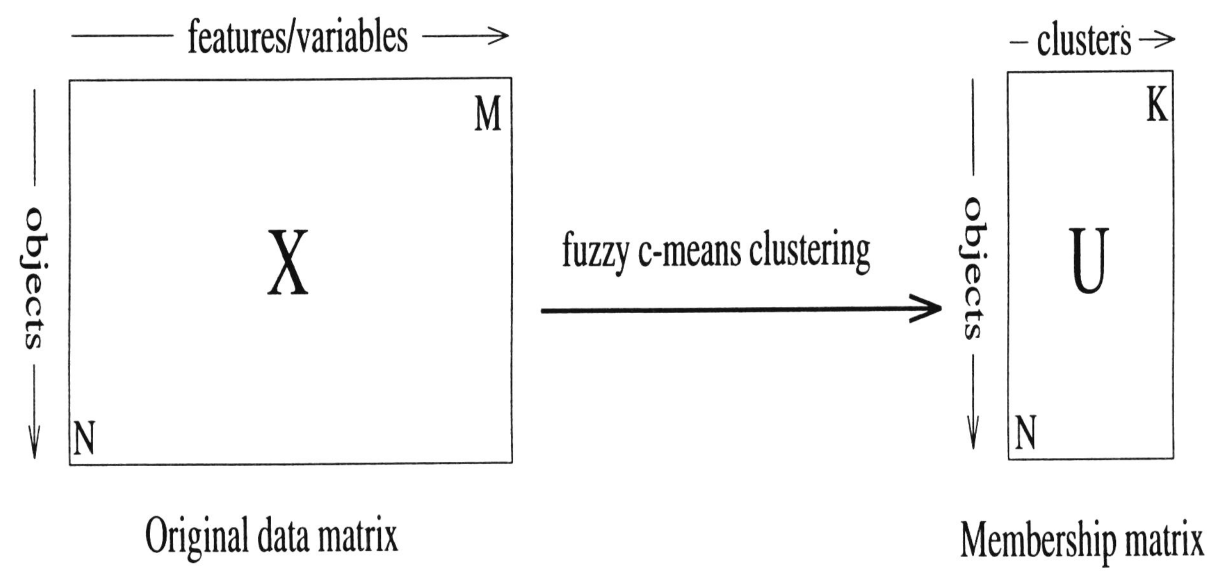
[14] Wold, S.; Esbensen, K.; Geladi, P. *Chemom. Intell. Lab. Syst.*, **1987**, *2*, 37–52.

[15] Martens, H.; Naes, T. *Multivariate Calibration.* John Wiley & Sons, New York, 1989.

[16] Lindgren, F.; Geladi, P.; Wold, S. *J. Chemom.*, **1992**, *7*, 45–59.

[17] Gordon, H. L.; Somorjai, R. L. *Proteins: Struct., Funct., Genet.*, **1992**, *14*(2), 249–64.

[18] Alsberg, B.; Kvalheim, O. *J. Chemom.*, **1993**, *7*, 61–73.



features/variables
objects
M
X
N
Original data matrix
fuzzy c-means clustering
clusters
objects
K
U
N
Membership matrix

Figure 1:

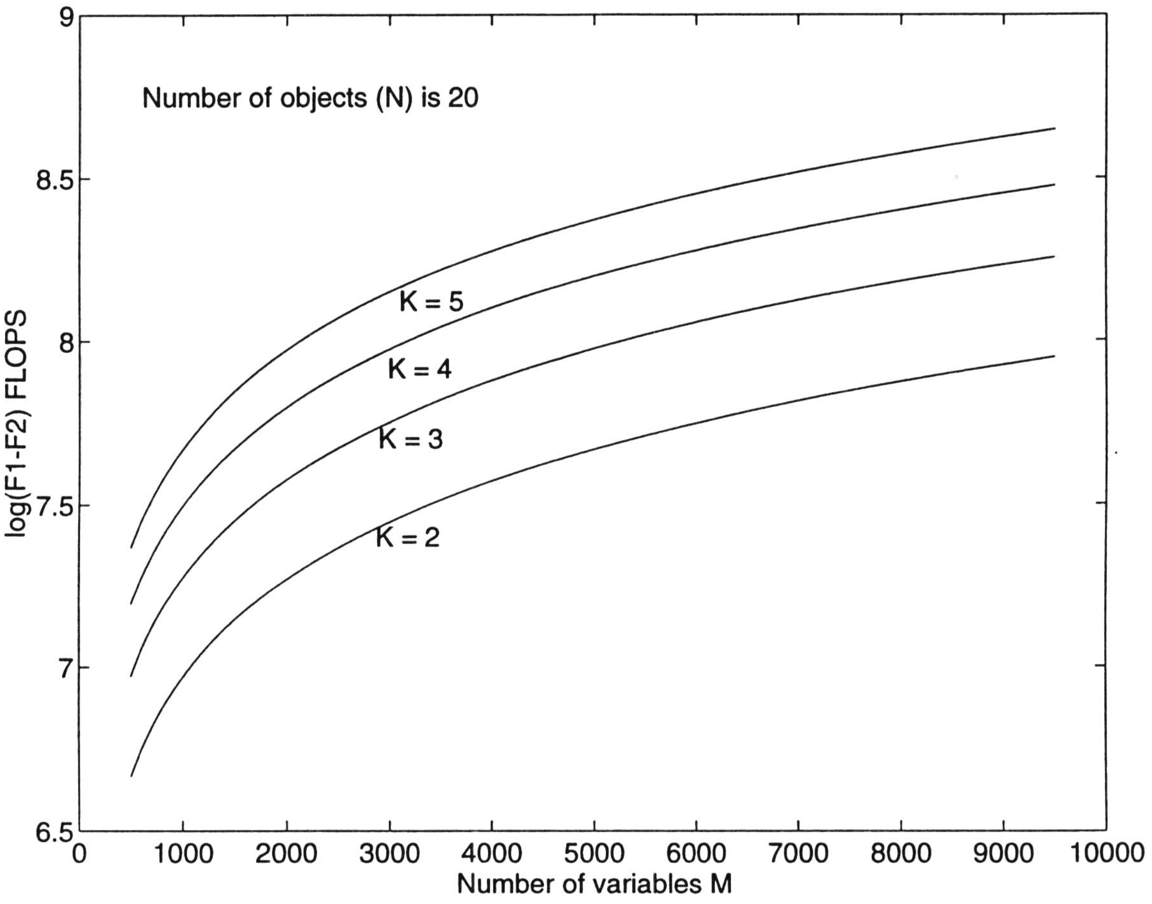

Number of objects (N) is 20
K = 5
K = 4
K = 3
K = 2
log(F1-F2) FLOPS
Number of variables M
9
8.5
8
7.5
7
6.5
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

Figure 2:

M	FCM	$FCM\text{-}M$	r
500	16.25	0.61	26.48
1100	35.73	1.21	29.55
1700	55.20	1.80	30.60
2300	74.68	2.40	31.13
2900	94.15	2.99	31.44
3500	113.63	3.59	31.65
4100	133.11	4.18	31.81
4700	152.58	4.78	31.92
5300	172.06	5.38	32.01
5900	191.53	5.97	32.08
6500	211.01	6.57	32.14
7100	230.49	7.16	32.19
7700	249.96	7.76	32.23
8300	269.44	8.35	32.26
8900	288.91	8.95	32.29
9500	308.39	9.54	32.32

Table 1:

	$K = 2$	$K = 3$	$K = 4$	$K = 5$
FCM	33.1 (33.1)	110.3 (110.3)	330.7 (330.7)	645.8 (645.8)
FCM-M	6.3 (6.4)	7.2 (7.3)	9.3 (9.3)	11.7 (11.7)
FCM-M2	6.3 (6.4)	1.6 (1.6)	3.6 (3.6)	6.0 (6.0)
FCM-M3	0.4 (0.4)	1.2 (1.2)	3.0 (3.1)	11.7 (11.7)
r (FCM/FCM-M)	5.3	15.3	35.6	55.2
r (FCM/FCM-M2)	5.3	68.9	91.9	107.6
r (FCM/FCM-M3)	82.8	91.9	110.2	55.2
Partition coef.	0.88	0.80	0.75	0.71

Table 2:

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

Table 3: