Adv Data Anal Classif (2011) 5:57–75
DOI 10.1007/s11634-010-0083-2

REGULAR ARTICLE

Generalized GIPSCAL re-revisited: a fast convergent algorithm with acceleration by the minimal polynomial extrapolation

Sébastien Loisel · Yoshio Takane

Received: 15 December 2009 / Revised: 15 December 2010 / Accepted: 21 December 2010 /
Published online: 9 January 2011

Abstract Generalized GIPSCAL, like DEDICOM, is a model for the analysis of square asymmetric tables. It is a special case of DEDICOM, but unlike DEDICOM, it ensures the nonnegative definiteness (*nnd*) of the model matrix, thereby allowing a spatial representation of the asymmetric relationships among "objects". A fast convergent algorithm was developed for GIPSCAL with acceleration by the minimal polynomial extrapolation. The proposed algorithm was compared with Trendafilov's algorithm in computational speed. The basic algorithm has been adapted to various extensions of GIPSCAL, including off-diagonal DEDICOM/GIPSCAL, and three-way GIPSCAL.

Keywords Asymmetric square tables · DEDICOM · Singular value decomposition (SVD) · Dynamical system algorithm · Diagonal estimation · Three-way data

Mathematics Subject Classification (2000) 62H25 · 65B05

1 Introduction

Asymmetric square tables arise in many scientific disciplines. Social mobility tables (sociology), brand switching data (marketing), stimulus identification data

The work reported here has been supported by a grant 10630 from the Natural Sciences and Engineering Research Council of Canada.

S. Loisel
Department of Mathematics, Heriot-Watt University, Edinburgh EH14 4AS, UK
e-mail: S.Loisel@hw.ac.uk

Y. Takane (✉)
Department of Psychology, McGill University, 1205 Dr. Penfield Avenue,
Montreal, QC H3A 1B1, Canada
e-mail: takane@psych.mcgill.ca

(psychology) are but a few examples. Generalized GIPSCAL (Generalized Inner Product SCALing; Kiers and Takane 1994; hereafter simply referred to as GIPSCAL) is a model for such tables. It is similar to DEDICOM (DEcomposition into DIrectional COMponents; Harshman 1978; Harshman et al. 1981), but unlike DEDICOM, the model matrix in GIPSCAL is constrained to be *nnd* (nonnegative definite). This allows a visualization of the asymmetric relationships among the n objects in a low (p) dimensional space.

More formally, let $\mathbf{A}^*$ denote an n by n model matrix in DEDICOM describing the asymmetric relationships among n objects. (In principle, $\mathbf{A}^*$ could also be symmetric. However, the main objective of DEDICOM is to analyze asymmetric data. We thus assume that $\mathbf{A}^*$ is asymmetric throughout this paper, unless otherwise stated, as is the matrix $\mathbf{A}$ to be introduced in (8).) DEDICOM postulates that this matrix can be expressed as

$$\mathbf{A}^* = \mathbf{YBY}', \tag{1}$$

where $\mathbf{Y}$ is an n by p columnwise nonsingular matrix that relates p latent "objects" to n observed objects, and $\mathbf{B}$ is a square asymmetric matrix of order p describing the asymmetric relationships among the latent objects. There are many decomposition methods that take a similar quadratic/bilinear form. What characterizes DEDICOM is the assumption that $\mathbf{B}$ is square and asymmetric.

Let $\mathbf{B}_s = (\mathbf{B} + \mathbf{B}')/2$, and $\mathbf{B}_{sk} = (\mathbf{B} - \mathbf{B}')/2$ represent the symmetric and skew-symmetric parts of $\mathbf{B}$, respectively. Then the DEDICOM model can be rewritten as

$$\mathbf{A}^* = \mathbf{Y}(\mathbf{B}_s + \mathbf{B}_{sk})\mathbf{Y}'. \tag{2}$$

Kiers and Takane (1994) assumed that $\mathbf{B}_s$ was *pd* (positive definite), and further rewrote (2) as follows. Let

$$\mathbf{B}_s = \mathbf{PD}^2\mathbf{P}' \tag{3}$$

denote the spectral decomposition of $\mathbf{B}_s$ where $\mathbf{D}^2 > \mathbf{0}$, and let

$$\mathbf{D}^{-1}\mathbf{P}'\mathbf{B}_{sk}\mathbf{PD}^{-1} = \mathbf{Q}\Delta\mathbf{R}' \tag{4}$$

be the singular value decomposition (SVD) of $\mathbf{D}^{-1}\mathbf{P}'\mathbf{B}_{sk}\mathbf{PD}^{-1}$. Note that the singular values of a skew symmetric matrix come in pairs except possibly for 0. Hence, Δ consists of diagonal sub-matrices of the form $\delta_j\mathbf{I}_2$ ($1 \leq j \leq (p-1)/2$; if p is odd, an additional 0 is appended to the diagonal elements of Δ). Note also that (4) can be rewritten as (Constantine and Gower 1978),

$$\mathbf{Q}\Delta\mathbf{R}' = \mathbf{Q}\Delta\mathbf{JQ}', \tag{5}$$

where $\mathbf{J}$ is a block diagonal matrix with diagonal blocks of the form $\left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right)$ (again, when p is odd, an additional 0 is appended to the diagonals). Then

$$\mathbf{A}^* = \mathbf{X}^*(\mathbf{I}_p + \Delta\mathbf{J})\mathbf{X}^{*'}, \tag{6}$$

where $\mathbf{X}^* = \mathbf{YPDQ}$. Kiers and Takane (1994) developed an alternating least squares algorithm to fit (6) to an observed square asymmetric table.

More recently, Trendafilov (2002) developed a dynamical system algorithm for GIPSCAL. While doing so, he also changed the model slightly. He required that $\mathbf{B}_s$ be *nnd* rather than *pd*. Let (3) denote the spectral decomposition of $\mathbf{B}_s$, where now $\mathbf{D}^2$ may have zero diagonal elements ($\mathbf{D}^2 \geq \mathbf{0}$). Then,

$$\mathbf{A}^* = \mathbf{X}(\mathbf{D}^2 + \mathbf{K})\mathbf{X}', \tag{7}$$

where $\mathbf{X} = \mathbf{YP}$, and $\mathbf{K} = \mathbf{P}'\mathbf{B}_{sk}\mathbf{P}$ is a skew-symmetric matrix. Trendafilov (2002) showed that his dynamical system algorithm worked better than Kiers and Takane's algorithm in two respects. On average, the computation time is shorter, and the value of the minimization criterion is smaller with his algorithm.

The better performance of Trendafilov's algorithm, however, may be due to the fact that he allowed $\mathbf{B}_s$ to be *nnd*, while Kiers and Takane assumed $\mathbf{B}_s$ to be strictly *pd*. The set of *pd* matrices is an open set within which a least squares (LS) loss function may not have a minimum (but only an infimum), in which case Kiers and Takane's algorithm never converges, while monotonically reducing the value of the loss function. (A minimum is attained on the boundary of the parameter space, but the boundary is not part of the feasible parameter space.) In such situations, Kiers and Takane (1994) algorithm continues to iterate forever (increasing the average convergence time) or it stops prematurely (giving rise to a larger value of fitting criterion). This point has been directly verified by running Kiers and Takane (1994) algorithm on data sets generated from *psd* (positive-semidefinite) D^2 matrices in (7).

In this paper, we develop an algorithm for model (7) that works better than Trendafilov's algorithm, thereby reaffirming the above contention. In the following section (Sect. 2.1), we present our basic algorithm, followed by an exposition of an acceleration technique called the minimal polynomial extrapolation (Sect. 2.2). Then Trendafilov's dynamical system algorithm is briefly discussed (Sect. 2.3). The three algorithms (the basic algorithm, the accelerated algorithm, and Trendafilov's algorithm) are then empirically evaluated (Sect. 3). Some extensions of the proposed algorithm to similar situations are considered in Sect. 4. These extensions include an additive constant incorporated into GIPSCAL, off-diagonal DEDICOM/GIPSCAL and three-way GIPSCAL. Section 5 concludes the paper.

2 Algorithms to be compared

In this section, we describe in some detail three algorithms to be compared in later sections. We start with our basic algorithm, which is then combined with the minimal

polynomial extrapolation (MPE) for acceleration, and then Trendafilov's dynamical system algorithm.

2.1 The basic algorithm

Let $\mathbf{A}$ be a square asymmetric data matrix. The two-way GIPSCAL model postulates

$$\mathbf{A} = \mathbf{A}^* + \mathbf{E}, \tag{8}$$

where $\mathbf{A}^*$ is as given in (7), and $\mathbf{E}$ is a matrix of disturbance terms. We estimate the parameters in the model in such a way that the following least squares (LS) criterion is minimized, namely

$$f(\mathbf{X}, \mathbf{D}^2, \mathbf{K}) = \mathrm{SS}(\mathbf{E}) = \mathrm{SS}(\mathbf{A} - \mathbf{X}(\mathbf{D}^2 + \mathbf{K})\mathbf{X}'), \tag{9}$$

where $\mathrm{SS}(\mathbf{E}) = \mathrm{tr}(\mathbf{E}'\mathbf{E})$. The above criterion is minimized using a conditional minimization strategy. That is, we first minimize $f(\mathbf{X}, \mathbf{D}^2, \mathbf{K})$ with respect to $\mathbf{D}^2$ and $\mathbf{K}$ conditional on $\mathbf{X}$, and then with respect to $\mathbf{X}$. This is written as

$$\min_{X, D^2, K} f(\mathbf{X}, \mathbf{D}^2, \mathbf{K}) = \min_{X} \min_{D^2, K|X} f(\mathbf{X}, \mathbf{D}^2, \mathbf{K}). \tag{10}$$

The conditional minimum of f with respect to $\mathbf{D}^2$ and $\mathbf{K}$ given $\mathbf{X}$ is obtained by

$$\hat{\mathbf{D}}^2 = \max(\mathrm{diag}(\mathbf{X}'\mathbf{A}_s\mathbf{X}), \mathbf{0}), \tag{11}$$

and

$$\hat{\mathbf{K}} = \mathbf{X}'\mathbf{A}_{sk}\mathbf{X}, \tag{12}$$

where $\mathbf{A}_s = (\mathbf{A}+\mathbf{A}')/2$ and $\mathbf{A}_{sk} = (\mathbf{A}-\mathbf{A}')/2$ are the symmetric and skew-symmetric parts of $\mathbf{A}$, respectively. Let

$$g(\mathbf{X}) = f(\mathbf{X}, \hat{\mathbf{D}}^2, \hat{\mathbf{K}}) = \min_{D^2, K|X} f(\mathbf{X}, \mathbf{D}^2, \mathbf{K}). \tag{13}$$

To minimize this function with respect to $\mathbf{X}$ subject to $\mathbf{X}'\mathbf{X} = \mathbf{I}$, we define

$$g^*(\mathbf{X}, \mathbf{S}) = g(\mathbf{X}) + \mathrm{tr}(\mathbf{S}(\mathbf{X}'\mathbf{X} - \mathbf{I})), \tag{14}$$

where $\mathbf{S}$ is a symmetric matrix of Lagrange multipliers. Differentiating (14) with respect to $\mathbf{X}$ and $\mathbf{S}$ and setting the results equal to zero gives

$$-\frac{1}{2}\frac{\partial g^*}{\partial \mathbf{X}} = \mathbf{G} - \mathbf{X}\hat{\mathbf{B}}'\hat{\mathbf{B}} - \mathbf{X}\hat{\mathbf{B}}\hat{\mathbf{B}}' - \mathbf{X}\mathbf{S} = \mathbf{0}, \tag{15}$$

where

$$\mathbf{G} = \mathbf{A}'\mathbf{X}\hat{\mathbf{B}} + \mathbf{A}\mathbf{X}\hat{\mathbf{B}}', \tag{16}$$

with $\hat{\mathbf{B}} = \hat{\mathbf{D}}^2 + \hat{\mathbf{K}}$, and

$$\mathbf{X}'\mathbf{X} - \mathbf{I} = \mathbf{0}. \tag{17}$$

Note that the derivatives of $g(\mathbf{X})$ with respect to $\mathbf{X}$ in (15) can be taken as if $\hat{\mathbf{D}}^2$ and $\hat{\mathbf{K}}$ were constant, whereas they are in fact functions of $\mathbf{X}$. This is justified by the fact that $\hat{\mathbf{D}}^2$ and $\hat{\mathbf{K}}$ are obtained by minimizing f conditional on $\mathbf{X}$. See Takane et al. (2010, Appendix) for full technical details. Premultiplying (15) by $\mathbf{X}'$ and considering (17), we obtain

$$\mathbf{S} = \mathbf{X}'\mathbf{G} - \hat{\mathbf{B}}'\hat{\mathbf{B}} - \hat{\mathbf{B}}\hat{\mathbf{B}}'. \tag{18}$$

Putting this into (15), we obtain

$$\mathbf{G} = \mathbf{X}\mathbf{X}'\mathbf{G}, \tag{19}$$

or

$$(\mathbf{I} - \mathbf{X}\mathbf{X}')\mathbf{G} = \mathbf{0}. \tag{20}$$

This equation is solved by (e.g., Jennrich 2001)

$$\mathbf{X} = \mathbf{U}\mathbf{V}', \tag{21}$$

where $\mathbf{U}$ and $\mathbf{V}$ are such that $\mathbf{G} = \mathbf{U}\mathbf{D}\mathbf{V}'$ is the SVD of $\mathbf{G}$.

Algorithm 1 (*GIPSCAL*) Let $\mathbf{A}$ be a square asymmetric matrix of order n, and let $\mathbf{X}^{(0)}$ be an n by p columnwise orthogonal matrix, where $p \leq n$. For each $j = 0, 1, \ldots$, compute $\mathbf{X}^{(j+1)}$ using the following steps:

(1) Compute $\hat{\mathbf{D}}^2$ and $\hat{\mathbf{K}}$ using (11) and (12), with $\mathbf{X} = \mathbf{X}^{(j)}$.
(2) Compute $\mathbf{X}^{(j+1)} = \mathbf{X}$ using (21).

Remark 1 Algorithm 1 can be rephrased as a fixed-point iteration of the form

$$\mathbf{X}^{(j+1)} = h_{\text{GIPSCAL}}(\mathbf{X}^{(j)}), \tag{22}$$

where h_{GIPSCAL} is the process described by steps (1) and (2) of Algorithm 1.

The above algorithm can easily be combined with acceleration by the minimal polynomial extrapolation (MPE) method to be described in the following subsection. Note that GIPSCAL reduces to the spectral decomposition of $\mathbf{A}$ when $\mathbf{A}$ is symmetric and *nnd*, and consequently $\mathbf{K} = \mathbf{0}$.

2.2 Acceleration of Algorithm 1 by the MPE method

In this subsection, we outline the MPE method of convergence acceleration for vector sequences, and we explain how this method can be used to accelerate the convergence of our basic GIPSCAL algorithm presented in the previous subsection. A good overview of vector acceleration techniques can be found in Smith et al. (1987). See also Loisel and Takane (2009, 2010), and Takane et al. (2010).

We begin by defining the MPE algorithm, which accelerates the convergence of vector sequences.

Algorithm 2 (*Minimal Polynomial Extrapolation*) Let $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(k)}$ be vector iterates. Define

$$\mathbf{u}^{(j)} = \mathbf{x}^{(j+1)} - \mathbf{x}^{(j)}, \tag{23}$$

for $j = 0, \ldots, k$, and let

$$\mathbf{U} = [\mathbf{u}^{(0)}, \ldots, \mathbf{u}^{(k-1)}]. \tag{24}$$

Define $\mathbf{c} = [c_0, c_1, \ldots, c_{k-1}]'$ by

$$\mathbf{c} = -\mathbf{U}^{+}\mathbf{u}^{(k)}. \tag{25}$$

Then, the limit of the vector sequence $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots$ predicted by MPE is given by

$$\mathbf{x}_{\mathrm{MPE}} = \sum_{j=0}^{k} c_j \mathbf{x}^{(j)} \Big/ \sum_{j=0}^{k} c_j, \tag{26}$$

where we have defined $c_k = 1$.

To understand how MPE works, it is best to consider a fixed point iteration whose update function is linear. Let

$$h_{\mathrm{lin}}(\mathbf{x}) = \mathbf{H}\mathbf{x} + \mathbf{b}, \tag{27}$$

and for a given $\mathbf{x}^{(0)}$, consider the vector sequence defined by

$$\mathbf{x}^{(j+1)} = h_{\mathrm{lin}}(\mathbf{x}^{(j)}) = \mathbf{H}\mathbf{x}^{(j)} + \mathbf{b}. \tag{28}$$

If the sequence converges to a point $\mathbf{x}_{conv}$, then we have

$$\mathbf{x}_{conv} = (\mathbf{I} - \mathbf{H})^{-1}\mathbf{b}, \tag{29}$$

where $\mathbf{I} - \mathbf{H}$ is assumed nonsingular. We define the increments $\mathbf{u}^{(j)}$ of the iteration using (23) for $j = 0, 1, \ldots$ The matrix $\mathbf{H}$ and its *minimal polynomial with respect to* $\mathbf{u}^{(0)}$ play an important role in the analysis of the convergence of the MPE acceleration method defined by Algorithm 2.

Definition 1 The minimal polynomial $\mathbf{P}(\mathbf{H})$ of the matrix $\mathbf{H}$ with respect to $\mathbf{u}^{(0)}$ is the unique polynomial in $\mathbf{H}$ whose leading coefficient is 1, and whose degree k is the smallest possible, such that

$$\mathbf{P}(\mathbf{H})\mathbf{u}^{(0)} = \mathbf{0}. \tag{30}$$

The theory of minimal polynomials is a standard component of linear algebra. The existence of a minimal polynomial follows immediately from the fact that the vector space is finite dimensional. Indeed, in a d-dimensional vector space, the vectors $\{\mathbf{u}^{(0)}, \mathbf{H}\mathbf{u}^{(0)}, \ldots, \mathbf{H}^d\mathbf{u}^{(0)}\}$ must be linearly dependent, since there are $d+1$ of them. Hence, for some $k \leq d$, there is a set of coefficients $c_0, \ldots, c_k$ such that

$$\sum_{j=0}^{k} c_j \mathbf{H}^j \mathbf{u}^{(0)} = \mathbf{0}. \tag{31}$$

By taking k as small as possible, we may further assume that $c_k = 1$, and this gives (30).

Lemma 1 *Let* $\mathbf{x}^{(0)}$ *be given, and* $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$ *be defined by* (28)*, and let* $\mathbf{u}^{(0)}$ *be defined by* (23)*. Let k be the degree of the minimal polynomial* $\mathbf{P}(\mathbf{H})$ *of* $\mathbf{H}$ *with respect to* $\mathbf{u}^{(0)}$*. Then, the limit* $\mathbf{x}_{conv}$ *of* (28) *is* $\mathbf{x}_{\mathrm{MPE}}$*, as defined by Algorithm 2.*

In other words, the MPE algorithm computes the true limit $\mathbf{x}_{conv}$.

Proof of Lemma 1 Let

$$\mathbf{P}(\mathbf{H}) = c_0\mathbf{I} + c_1\mathbf{H} + \cdots + \mathbf{H}^k \tag{32}$$

be the minimal polynomial $\mathbf{P}(\mathbf{H})$ of $\mathbf{H}$ with respect to $\mathbf{u}^{(0)}$. We now show how to recover the coefficients $c_0, c_1, \ldots, c_{k-1}$ from the increments $\mathbf{u}^{(0)}, \ldots, \mathbf{u}^{(k-1)}$. Observe that $\mathbf{u}^{(j)} = \mathbf{H}^j\mathbf{u}^{(0)}$ for $j = 0, 1, \ldots, k$. Hence, from (30) and (32), we have that

$$\sum_{j=0}^{k} c_j \mathbf{u}^{(j)} = \mathbf{0}, \tag{33}$$

where $c_k = 1$. We may rewrite this as the linear system

$$\mathbf{U}\begin{bmatrix} c_0 \\ \vdots \\ c_{k-1} \end{bmatrix} = -\mathbf{u}^{(k)}, \tag{34}$$

where we have moved the data $c_k = 1$ from the left-hand side over to the right-hand side. Although $\mathbf{U}$ may be rectangular, since the system (34) is obtained from the minimal polynomial equation (30), we know that (34) has a unique solution. Any method can be used to obtain this solution, but certainly one may use Eq. (25). We have thus now shown that the vector $\mathbf{c}$ is indeed the vector of the coefficients of the minimal

polynomial of $\mathbf{H}$ with respect to $\mathbf{u}^{(0)}$, provided that the number k is the degree of that minimal polynomial.

We now turn our attention to the next task, which is to recover $\mathbf{x}_{conv}$ from the iterates $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(k+1)}$. Observe that

$$\mathbf{u}^{(0)} = (\mathbf{I} - \mathbf{H})(\mathbf{x}_{conv} - \mathbf{x}^{(0)}). \tag{35}$$

(The above identity is readily verified.) We substitute this into Eq. (30) (and take into account that $\mathbf{I} - \mathbf{H}$ commutes with $\mathbf{P}(\mathbf{H})$, and it is invertible) to obtain the relation

$$\begin{aligned} \mathbf{0} &= \mathbf{P}(\mathbf{H})(\mathbf{x}_{conv} - \mathbf{x}^{(0)}) \\ &= \sum_{j=0}^{k} c_j \mathbf{H}^j (\mathbf{x}_{conv} - \mathbf{x}^{(0)}) \\ &= \sum_{j=0}^{k} c_j (\mathbf{x}_{conv} - \mathbf{x}^{(j)}); \end{aligned} \tag{36}$$

where we have used the following relation:

$$\begin{aligned} \mathbf{x}_{conv} - \mathbf{x}^{(j)} &= (\mathbf{H}\mathbf{x}_{conv} + \mathbf{b}) - (\mathbf{H}\mathbf{x}^{(j-1)} + \mathbf{b}) = \mathbf{H}(\mathbf{x}_{conv} - \mathbf{x}^{(j-1)}) \\ &= \cdots = \mathbf{H}^j (\mathbf{x}_{conv} - \mathbf{x}^{(0)}). \end{aligned}$$

By solving Eq. (36) for the unknown $\mathbf{x}_{conv}$, we obtain that $\mathbf{x}_{conv} = \mathbf{x}_{\mathrm{MPE}}$, where $\mathbf{x}_{\mathrm{MPE}}$ is defined by (26). □

The fixed point iteration defined by Algorithm 1 is nonlinear; i.e., the function h_{GIPSCAL}, defined in Remark 1, is nonlinear. Therefore, MPE as defined in Algorithm 2 will not generally produce the limit point of the iteration for any finite value of k. Nevertheless, we can define an MPE accelerated version of Algorithm 1, as follows.

Algorithm 3 (*GIPSCAL-MPE*) Let $\mathbf{A}$ be an n by n asymmetric matrix, and let $\mathbf{X}^{(0)}$ be an n by p columnwise orthogonal matrix, with $p \leq n$. Let $k \geq 1$ be an integer.

(1) Compute $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(k)}$ using Algorithm 1.
(2) Convert the n by p matrices $\mathbf{X}^{(0)}, \ldots, \mathbf{X}^{(k)}$ into np-dimensional column vectors $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(k)}$. Compute $\mathbf{x}_{\mathrm{MPE}}$ using Algorithm 2. Convert $\mathbf{x}_{\mathrm{MPE}}$ into a matrix, re-orthonormalize it by SVD (see (21)), and store it into $\mathbf{X}^{(0)}$.
(3) Iterate steps (1) and (2) until convergence.

The above algorithm assumes that the value of k is known. However, as far as the MPE method has to be applied repeatedly as described in Algorithm 3 because of the nonlinearity of the updating equation h_{GIPSCAL}, a precise value of k is generally not required. It suffices to have a value of k for which (33) holds approximately. Typically, there is a wide range of values of k for which the MPE algorithm works well. In the following numerical experiments, we vary the value of k (= 5, 10, 15, and 20) systematically, and choose the best value.

2.3 Trendafilov's algorithm

Trendafilov (2002) proposed an algorithm for GIPSCAL which also minimizes the same criterion (9). His method reformulates the problem as an ordinary differential equation, whose asymptotic solution as $t \to \infty$ is a solution of the generalized GIPSCAL problem. Indeed, we can regard (9) as an energy which is to be minimized. One method for minimizing an energy is to simulate a physical system in which "particles" are following the steepest descent direction.

Assume that we are given an *energy functional* $E(\mathbf{Y})$, which is a non-negative function of the vector or matrix $\mathbf{Y}$. Given initial estimates $\mathbf{Y}(0)$, we can define the function $\mathbf{Y}(t)$ of the time parameter t to be the unique solution to the differential equation

$$\frac{d}{dt}\mathbf{Y}(t) = -(\nabla E)(\mathbf{Y}(t)), \tag{37}$$

where ∇E denotes the gradient of E with respect to $\mathbf{Y}$. This defines a "gradient dynamical system", which can be interpreted physically as particles following the gradient of the energy functional. Under some conditions, one may show that $\mathbf{Y}(t)$ converges to some limit as $t \to \infty$. Moreover, we have that

$$(\nabla E)(\mathbf{Y}(\infty)) = 0. \tag{38}$$

In other words, $\mathbf{Y}(\infty)$ is a critical point of $E(\mathbf{Y})$.

We now relate this gradient dynamical system to our optimization problem (9). We consolidate the variables $\mathbf{X}$, $\mathbf{D}$ and $\mathbf{K}$ into a single object $\mathbf{Y} = (\mathbf{X}, \mathbf{D}, \mathbf{K})$. (In Trendafilov's algorithm, $\mathbf{D}$ rather than $\mathbf{D}^2$ is estimated directly. This ensures the *nnd*-ness of $\mathbf{D}^2$.) We can then define $E(\mathbf{Y}) = f(\mathbf{X}, \mathbf{D}, \mathbf{K})$ (cf. (9)). However, there is a significant pitfall. The gradient (∇E) appearing on the right-hand side of (37) must be understood in terms of the tangent space of the manifold in which $\mathbf{Y}$ resides. This requires some further technical reasoning, which we now outline [and we refer to Trendafilov (2002) for details].

Recall that the variable $\mathbf{X}$ is an $n \times p$ columnwise orthogonal matrix, which we write as $\mathbf{X} \in O(n, p)$. Likewise, we write $\mathbf{D} \in D(p)$ and $\mathbf{K} \in Sk(p)$ to denote that $\mathbf{D}$ and $\mathbf{K}$ are diagonal and skew-symmetric, respectively. In this notation, we therefore have that $\mathbf{Y} \in O(n, p) \times D(p) \times Sk(p) =: \mathcal{Y}$. This set is a smooth manifold. Hence, for any $\mathbf{Y} \in \mathcal{Y}$, there is a corresponding tangent space, denoted $\mathcal{T}_{\mathbf{Y}}$. Because $\mathcal{Y}$ is a product, the tangent spaces are also given by the following product:

$$\mathcal{T}_{\mathbf{Y}} = \mathcal{T}_{(\mathbf{X},\mathbf{D},\mathbf{K})} = \mathcal{T}_{\mathbf{X}} O(n, p) \times D(p) \times Sk(p), \tag{39}$$

where

$$\mathcal{T}_{\mathbf{X}} O(n, p) = \{\mathbf{H} \in \mathbb{R}^{n \times p} | \mathbf{X}'\mathbf{H} \text{ is skew-symmetric}\}. \tag{40}$$

The gradient $\nabla_{\mathbf{X}} f$ should then be understood in terms of a tangential derivative in the manifold $O(n, p)$. In other words, for any fixed $\mathbf{X}$, the gradient $(\nabla_{\mathbf{X}} f)(\mathbf{X})$ is a linear

function defined for all tangent directions in $\mathcal{T}_{\mathbf{X}}O(n,\,p)$. This derivative can be made explicit in terms of the entrywise derivatives $\frac{\partial f}{\partial x_{ij}}$, where $\mathbf{X} = (x_{ij})$.

We can assemble these entrywise derivatives into an $n \times p$ matrix

$$f_{\mathbf{X}} = \begin{pmatrix} \frac{\partial f}{\partial x_{11}} & \cdots & \frac{\partial f}{\partial x_{1p}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_{n1}} & \cdots & \frac{\partial f}{\partial x_{np}} \end{pmatrix} \in \mathbb{R}^{n\times p}. \tag{41}$$

The difficulty is that this entrywise derivative is not likely to be in the tangent space $\mathcal{T}_{\mathbf{X}}O(n,\,p)$. The derivative "in the tangent space" is instead defined to be a projection of this entrywise derivative to the tangent space. This projection is defined by

$$\pi_{\mathcal{T}_{\mathbf{X}}}(\mathbf{Z}) = \mathbf{X}\frac{\mathbf{X}'\mathbf{Z} - \mathbf{Z}\mathbf{X}}{2} + (\mathbf{I} - \mathbf{X}\mathbf{X}')\mathbf{Z}. \tag{42}$$

Then, the derivative $\nabla_{\mathbf{X}} f$ "in the tangent space" is defined by

$$\nabla_{\mathbf{X}} f = \pi_{\mathcal{T}_{\mathbf{X}}}(f_{\mathbf{X}}). \tag{43}$$

The variables $\mathbf{D}$ and $\mathbf{K}$ reside in vector spaces instead of curved manifolds, and hence no such subtlety arises for these variables. After further simplifications, the resulting system of ordinary differential equations is given by (Trendafilov 2002)

$$\frac{d\mathbf{X}}{dt} = \mathbf{X}([\mathbf{K}, \mathbf{X}'\mathbf{A}_{sk}\mathbf{X}] - [\mathbf{D}^2, \mathbf{X}'\mathbf{A}_s\mathbf{X}]) + 2(\mathbf{I} - \mathbf{X}\mathbf{X}')(\mathbf{A}_s\mathbf{X}\mathbf{D}^2 - \mathbf{A}_{sk}\mathbf{X}\mathbf{K}), \tag{44}$$

$$\frac{d\mathbf{D}}{dt} = 2(\mathbf{X}'\mathbf{A}_s\mathbf{X} - \mathbf{D}^2) \odot \mathbf{D}, \tag{45}$$

$$\frac{d\mathbf{K}}{dt} = \mathbf{X}'\mathbf{A}_{sk}\mathbf{X} - \mathbf{K}, \tag{46}$$

where we have used the notation $\odot$ to denote the entrywise product, and $[\mathbf{A}, \mathbf{B}] = \mathbf{A}\mathbf{B} - \mathbf{B}\mathbf{A}$ indicates the Lie bracket.

Trendafilov's algorithm then consists of solving the system (44), (45), and (46), given a starting point $\mathbf{Y}(0)$, using the MATLAB ODE solver `ode15s` from the initial time $t = 0$ until the time $t = 100$ (with possible early termination if E decreases very slowly). The value $\mathbf{Y}(100)$ is then returned as a local minimizer of (9).

3 Numerical experiments

In this section, we report the results of a numerical experiment (Experiment 1) on the algorithms described in the previous section. In our first numerical experiment, we compare the mean CPU time and the average fit reached by Algorithm 1, Algorithm 3, and Trendafilov's. Two hundred fifty data sets each were generated by varying the number of objects at three levels ($n = 10$, 20, and 30) with each entry of the data tables

following the uniform distribution between −.5 and .5 (Trendafilov 2002). Randomly generated data sets presumably impose the toughest condition for algorithms because they must look for structures that "do not exist" in the data, and if they work well under these conditions, they are bound to work well in more natural settings, where some GIPSCAL structures exist. Jennrich (2001) and Takane and Zhang (2009) tested their algorithms under similar conditions.

The data were analyzed by Algorithm 1, Algorithm 3 with $k = 5$, 10, 15, and 20, and Trendafilov's algorithm. Only 3-component solutions were obtained for $n = 10$, while both 3- and 5-component solutions were obtained for $n = 20$ and $n = 30$. For Algorithms 1 and 3, the normalized Frobenius norm of the projected gradients was defined as the square root of the sum of squares of the left-hand side of (20) divided by the sum of squares of data elements, and this quantity being less than 10^{-7} was used as the convergence criterion. Trendafilov's algorithm, on the other hand, used the criterion that the improvement in the loss function between two consecutive output points was less than 10^{-6}. This criterion turned out to be much more lenient than that used for Algorithms 1 and 3. [This was directly verified by evaluating the normalized Frobenius norm of the projected gradients (defined above) at the convergence points of Trendafilov's algorithm. This quantity was almost always larger than 10^{-7}.] However, no further effort was made to equate the two convergence criteria because Algorithm 3 was found much faster than Trendafilov's algorithm despite the fact that it used a more stringent convergence criterion.

For Algorithms 1 and 3, initial estimates of $\mathbf{X}$ were first generated by uniform random numbers between −.5 and .5 followed by the orthonormalization step by SVD (see (21)). In Trendafilov's algorithm, initial estimates were calculated by a matrix of eigenvectors of the symmetric part of the data matrix corresponding to the p largest eigenvalues. Initial estimates of $\mathbf{D}$ and $\mathbf{K}$ were then calculated by $\hat{\mathbf{D}} = \{\max(\mathrm{diag}(\mathbf{X}'\mathbf{A}_s\mathbf{X}), \mathbf{0})\}^{1/2}$, and $\hat{\mathbf{K}} = \mathbf{X}'\mathbf{A}_{sk}\mathbf{X}$.

The main results of the simulation study are summarized in Table 1. In the table, two numbers are given in each cell, one without and the other with parentheses. The

Table 1 The comparison of the mean cpu time and fit among Algorithm 1, Algorithm 3 ($k = 5$, 10, 15, and 20), and Trendafilov's

n	p		Algorithm 1	Algorithm 3				Trendafilov
				$k = 5$	$k = 10$	$k = 15$	$k = 20$	
10	3	cpu	0.1633	0.0183	0.0114	0.0149	0.0217	0.1394
		fit	(.6482)	(.6001)	(.6003)	(.6011)	(.6031)	(.6010)
20	3	cpu	0.2454	0.0316	0.0179	0.0204	0.0209	0.2401
		fit	(.7904)	(.7643)	(.7635)	(.7634)	(.7635)	(.7651)
20	5	cpu	0.5061	0.0966	0.0512	0.0440	0.0465	0.5417
		fit	(.6702)	(.6377)	(.6377)	(.6380)	(.6379)	(.6376)
30	3	cpu	0.3540	0.0602	0.0365	0.0276	0.0271	0.3544
		fit	(.8537)	(.8321)	(.8322)	(.8323)	(.8322)	(.8338)
30	5	cpu	0.7085	0.1551	0.0894	0.0655	0.0629	1.0029
		fit	(.7627)	(.7372)	(.7376)	(.7375)	(.7376)	(.7372)

former indicates the mean cpu time, and the latter the mean fit value at the convergence point. It is clear that the MPE method speeds up the convergence substantially. An optimal value of k ranges between 10 and 15, although for larger problems, a larger value of k may be desired. It is important to observe that within this range, Algorithm 3 works very well, and that it is nearly 10 times as fast as both Algorithm 1 and Trendafilov's algorithm. Trendafilov's algorithm has a slight edge over Algorithm 1 for smaller problems, although its advantage disappears for larger problems.

The computation times given in Table 1 were measured between the start of the algorithms and whatever stationary points the algorithms first reached. The stationary points reached, however, may not be a global minimum of the loss function. To compare the quality of solutions obtained by the various algorithms, Table 1 also reports the average fit value at the convergence points. The average fits obtained by the three algorithms are very comparable (Algorithm 1 is somewhat worse than the other two). This means that there is not much difference in the quality of solutions obtained by Algorithm 3 and Trendafilov's.

But just how serious is the problem of suboptimal solutions under the conditions examined above? To investigate this problem, Algorithm 3 with $k = 10$ was run 50 times for each data set starting from 50 random initials. The best solution among the 50 solutions was considered as the globally optimal solution. (This is justified by the following reasoning. Even if the chance of convergence to a suboptimal solution in a single run is as high as .80, the probability of hitting the globally optimal solution at least once in 50 runs is quite high; this is calculated by $1 - .80^{50} \approx .999986$.) The globally optimal solution was then compared with a solution in a single run to see if the latter is a suboptimal solution or not. The incidence of suboptimal solutions is then averaged over 250 data sets in each condition. The probabilities of suboptimal solutions thus obtained were .232, .260, .404, .236, and .428 for the five conditions (the five combinations of n and p) in Table 1. It seems that they are more heavily affected by the dimensionality (p) of solutions than the number (n) of objects. When $p = 3$, the probability of suboptimal solutions is around .25; this jumps up to above .40 for $p = 5$.

4 Some extensions

Model (8) is the very basic model for GIPSCAL, and various extensions of the basic model are possible. In this section, we consider three such extensions along with the corresponding extensions of Algorithms 1 and 3.

4.1 Incorporating an additive constant into GIPSCAL

In many areas of social sciences, data are often measured on an interval scale with no intrinsic zero point. In order to account for the effect of an arbitrary zero point, Chino (1990) considered incorporating an additive constant to his original GIPSCAL model. This model, with his original GIPSCAL model replaced by the generalized GIPSCAL model (7), can be written as

$$\mathbf{A} = \mathbf{X}(\mathbf{D}^2 + \mathbf{K})\mathbf{X}' + c\mathbf{1}\mathbf{1}' + \mathbf{E}, \tag{47}$$

where c is the additive constant. We can easily modify Algorithm 1 to estimate the additional parameter c.

Algorithm 4 (*GIPSCAL-c*) Let $\mathbf{A}$ be a square asymmetric matrix of order n, and let $\mathbf{X}^{(0)}$ be a given n by p columnwise orthogonal matrix. Let $\hat{\mathbf{D}}^2$ be a non-negative diagonal matrix of order p (an initial estimate of $\mathbf{D}^2$). For $j = 0, 1, \ldots$, compute $\mathbf{X}^{(j+1)}$ as follows:

(1) Compute $\hat{c}$ by taking the average of $\mathbf{A} - \mathbf{X}^{(j)}\hat{\mathbf{D}}^2(\mathbf{X}^{(j)})'$. (That is, $\hat{c} = \mathbf{1}'(\mathbf{A} - \mathbf{X}^{(j)}\hat{\mathbf{D}}^2(\mathbf{X}^{(j)})')\mathbf{1}/n^2$, where $\mathbf{1}$ is the n-component vector of ones.)
(2) Compute updated $\hat{\mathbf{D}}^2$ and $\hat{\mathbf{K}}$ values using

$$\hat{\mathbf{D}}^2 = \max(\mathrm{diag}((\mathbf{X}^{(j)})'(\mathbf{A}_s - \hat{c}\mathbf{1}\mathbf{1}')\mathbf{X}^{(j)}), \mathbf{0}), \tag{48}$$

and

$$\hat{\mathbf{K}} = (\mathbf{X}^{(j)})'\mathbf{A}_{sk}\mathbf{X}^{(j)}, \tag{49}$$

where, as before, $\mathbf{A}_s = (\mathbf{A} + \mathbf{A}')/2$ is the symmetric part of $\mathbf{A}$, and $\mathbf{A}_{sk} = (\mathbf{A} - \mathbf{A}')/2$ is the skew-symmetric part of $\mathbf{A}$.
(3) Set $\mathbf{G} = (\mathbf{A} - \hat{c}\mathbf{1}\mathbf{1}')'\mathbf{X}^{(j)}\hat{\mathbf{B}} + (\mathbf{A} - \hat{c}\mathbf{1}\mathbf{1}')\mathbf{X}^{(j)}\hat{\mathbf{B}}'$, where $\hat{\mathbf{B}} = \hat{\mathbf{D}}^2 + \hat{\mathbf{K}}$. Then, set

$$\mathbf{X}^{(j+1)} = \mathbf{U}\mathbf{V}', \tag{50}$$

where $\mathbf{U}$ and $\mathbf{V}$ are such that $\mathbf{G} = \mathbf{U}\mathbf{D}\mathbf{V}'$ is the SVD of $\mathbf{G}$.

The above algorithm can be easily combined with the MPE acceleration method in the same manner as Algorithm 1 was combined with Algorithm 2 to produce Algorithm 3. We call this algorithm **Algorithm 4′**.

Remark 2 Algorithm 4 can be written as a fixed-point iteration of the form

$$\begin{bmatrix} \mathbf{X}^{(j+1)} \\ (\mathbf{D}^2)^{(j+1)} \end{bmatrix} = f_{\mathrm{GIPSCAL-c}}\left(\begin{bmatrix} \mathbf{X}^{(j)} \\ (\mathbf{D}^2)^{(j)} \end{bmatrix}\right). \tag{51}$$

This suggests that we would need to pass the matrix iterates $(\mathbf{D}^2)^{(j)}$ as well as $\mathbf{X}^{(j)}$, to Algorithm 2. However, we found that passing the matrices $\mathbf{X}^{(j)}$ was sufficient to accelerate the convergence of the overall iteration given by (51).

Remark 3 In Algorithm 4, no initial value for $\mathbf{K}$ is needed. Indeed, we can estimate c by averaging $\mathbf{A} - \mathbf{X}^{(j)}(\hat{\mathbf{D}}^2 + \hat{\mathbf{K}})(\mathbf{X}^{(j)})'$. The matrix $\hat{\mathbf{K}}$ is skew-symmetric, and for such a matrix, we have that $\mathbf{u}'\mathbf{K}\mathbf{u} = 0$, for any vector $\mathbf{u}$. Hence, $\mathbf{1}'\mathbf{X}\mathbf{K}\mathbf{X}'\mathbf{1} = 0$, and the term in $\hat{\mathbf{K}}$ contributes nothing to the average.

4.2 Off-diagonal DEDICOM/GIPSCAL

Diagonal entries of a square asymmetric table may have different meanings from its off-diagonal entries. For example, in the case of trade between nations, diagonal elements represent the amount of domestic trade, and off-diagonal elements the amount of international trade. The two parts of the table may be governed by different principles.

To account for the difference between the two, Takane (1985) considered incorporating an additional diagonal matrix into GIPSCAL in a manner similar to uniqueness in common factor analysis. This model may be written as

$$\mathbf{A} = \mathbf{X}\mathbf{B}\mathbf{X}' + \mathbf{C} + \mathbf{E}, \tag{52}$$

where $\mathbf{C}$ is the additional diagonal matrix to be estimated. This model is often called off-diagonal DEDICOM (ten Berge and Kiers 1989). We may further require $\mathbf{C}$ to be *nnd*. Matrix $\mathbf{B}$ in DEDICOM is analogous to $\mathbf{D}^2 + \mathbf{K}$ in GIPSCAL. In DEDICOM, however, we do not separate $\mathbf{D}^2$ and $\mathbf{K}$ because no *nnd* restriction is imposed on $\mathbf{B}$. For the purpose of comparing our algorithm with ten Berge and Kiers (1989) algorithm for off-diagonal DEDICOM, we first present an algorithm for this model, and then extend it to off-diagonal GIPSCAL.

Algorithm 5 (*Off-diagonal DEDICOM*) Let $\mathbf{A}$ be a square asymmetric matrix of order n. Let $X^{(0)}$ be a given n by p columnwise orthogonal matrix, and let $\hat{\mathbf{B}}$ be a given nonsingular square matrix of order p (an initial estimate of $\mathbf{B}$). For $j = 0, 1, \ldots$, compute $\mathbf{X}^{(j+1)}$ as follows:

(1) Compute $\hat{\mathbf{C}}$ using

$$\hat{\mathbf{C}} = \text{diag}(\mathbf{A} - \mathbf{X}^{(j)}\hat{\mathbf{B}}(\mathbf{X}^{(j)})'). \tag{53}$$

(2) Update $\hat{\mathbf{B}}$ by computing

$$\hat{\mathbf{B}} = (\mathbf{X}^{(j)})'(\mathbf{A} - \hat{\mathbf{C}})\mathbf{X}^{(j)}. \tag{54}$$

(3) Compute $\mathbf{X}^{(j+1)}$ using

$$\mathbf{X}^{(j+1)} = \mathbf{U}, \tag{55}$$

where $\mathbf{U}$ is such that $\mathbf{G} = \mathbf{U}\Delta\mathbf{V}'$ is the SVD of $\mathbf{G} = (\mathbf{A} - \hat{\mathbf{C}})'\mathbf{X}^{(j)}\hat{\mathbf{B}} + (\mathbf{A} - \hat{\mathbf{C}})\mathbf{X}^{(j)}\hat{\mathbf{B}}'$.

Again, the above algorithm can be easily combined with the MPE algorithm, which we call **Algorithm 5′**. When $\mathbf{C} = 0$ is assumed, (58) reduces to the basic DEDICOM model, and Algorithm 5 without Step 1 reduces to Takane and Zhang (2009) algorithm for DEDICOM.

Remark 4 The matrix $\mathbf{U}$ appearing in Step (3) of Algorithm 5, is unique up to reflections and permutations of its column vectors. This is not a big problem when acceleration by the MPE method is not incorporated. However, the MPE method is sensitive

to the directions and permutations of singular vectors. This means that columns of successive $\mathbf{U}$'s should be matched in order and sign.

Remark 5 The diagonal matrix $\mathbf{C}$ may further be constrained to be nonnegative definite. This can be done by replacing Step 1 of the above algorithm by $\hat{\mathbf{C}} = \max(\mathrm{diag}(\mathbf{A} - \mathbf{X}^{(j)}\hat{\mathbf{B}}(\mathbf{X}^{(j)})'), \mathbf{0})$.

Remark 6 Similarly to Remark 2, we found that applying MPE to the iterates $\mathbf{X}^{(k)}$ was sufficient, and that it was not necessary to also include the $\mathbf{B}$ iterates in the call to the MPE acceleration routine.

Remark 7 The above algorithm can be easily modified to fit the off-diagonal GIPSCAL model $\mathbf{A} = \mathbf{X}(\mathbf{D}^2 + \mathbf{K})\mathbf{X}' + \mathbf{C} + \mathbf{E}$ by replacing Step (2) of Algorithm 5 by $\hat{\mathbf{D}}^2 = \max(\mathrm{diag}((\mathbf{X}^{(j)})'(\mathbf{A}_s - \mathbf{C})\mathbf{X}^{(j)}), \mathbf{0})$, $\hat{\mathbf{K}} = (\mathbf{X}^{(j)})'\mathbf{A}_{sk}\mathbf{X}^{(j)}$, and $\hat{\mathbf{B}} = \hat{\mathbf{D}}^2 + \hat{\mathbf{K}}$, and Step (3) of Algorithm 5 by $\mathbf{X}^{(j+1)} = \mathbf{U}\mathbf{V}'$, where $\mathbf{U}$ and $\mathbf{V}$ are such that $\mathbf{G} = \mathbf{U}\Delta\mathbf{V}'$ is the SVD of $\mathbf{G}$.

Remark 8 Incorporation of an additive constant described in Sect. 4.1, and off-diagonal DEDICOM/GIPSCAL described in Sect. 4.2 may be combined. It should not make very much difference whether we update $\hat{c}$ or $\hat{\mathbf{C}}$ first.

Algorithm 5′ was compared to ten Berge and Kiers (1989) algorithm for off-diagonal DEDICOM. The data were generated exactly as in Experiment 1 (three levels of n (10, 20, 30), and each entry of the data tables generated by the uniform random number between $-.5$ and .5), but all analyses were conducted with $p = 3$. Algorithm 5′ used the same initialization procedure and convergence criterion as in Experiment 1. Ten Berge and Kier's algorithm used the same initialization procedure as Algorithm 5′, but used a convergence criterion similar to the one used in Trendafilov's algorithm. That is, an improvement in fit between two successive iterations is less than 10^{-7}.

The results are reported in Table 2. The basic construction of the table remains the same as in Table 1. The MPE method seems to have a considerable advantage over ten Berge and Kiers (1989) algorithm for off-diagonal DEDICOM. The former is about 3 times faster than the latter. The average quality of the solutions is comparable between the two algorithms. Note that, as the authors themselves note, it is difficult to impose the *nnd* restriction on $\mathbf{C}$ in ten Berge and Kiers algorithm, whereas it is straightforward to do so in Algorithm 5′. Also, ten Berge and Kiers' algorithm is not easily extensible to GIPSCAL, whereas Algorithm 5′ is, as has been demonstrated above.

Table 2 Off-diagonal DEDICOM: the comparison of the mean cpu time between Algorithm 5′ and ten Berge and Kiers (1989) minres algorithm

n		Algorithm 5′ ($k = 10$)	minres
10	cpu	0.0141	0.0500
	fit	(.4338)	(.4398)
20	cpu	0.0198	0.0598
	fit	(.6904)	(.6913)
30	cpu	0.0435	0.1221
	fit	(.7851)	(.7854)

4.3 Three-way GIPSCAL

So far, it is assumed that there is a single square asymmetric table to be analyzed by GIPSCAL. In some cases, however, there may be more than one such table available, possibly obtained under different conditions. Trendafilov (2002) proposed a model for such data, called three-way GIPSCAL. Suppose there are N square asymmetric data matrices of order n. The three-way GIPSCAL model is written as

$$\mathbf{A}_i = \mathbf{X}(\mathbf{D}_i^2 + \mathbf{K}_i)\mathbf{X}' + \mathbf{E}_i \tag{56}$$

for $i = 1, \ldots, N$. This model postulates an $\mathbf{X}$ (the matrix that relates latent "objects" to observed objects) common to all N data matrices, while accounting for their differences by allowing a distinct $\mathbf{D}_i^2$ and $\mathbf{K}_i$ for each data matrix. This model is an extension of the INDSCAL model for N symmetric tables to N square asymmetric tables.

In the three-way GIPSCAL model, we minimize

$$f(\mathbf{X}, \mathbf{D}_1^2, \ldots, \mathbf{D}_N^2, \mathbf{K}_1, \ldots, \mathbf{K}_N) = \sum_{i=1}^{N} \mathrm{SS}(\mathbf{E}_i) = \sum_{i=1}^{N} \mathrm{SS}(\mathbf{A}_i - \mathbf{X}(\mathbf{D}_i^2 + \mathbf{K}_i)\mathbf{X}'), \tag{57}$$

where $\mathrm{SS}(\mathbf{E}_i) = \mathrm{tr}(\mathbf{E}_i'\mathbf{E}_i)$. Following a similar line of reasoning to Sect. 2.1, we arrive at the following algorithm.

Algorithm 6 (*Three-way GIPSCAL)* Let $\mathbf{A}_1, \ldots, \mathbf{A}_N$ be n by n asymmetric matrices, and let $\mathbf{X}^{(0)}$ be an n by p columnwise orthogonal matrix. For $j = 0, 1, \ldots$, compute $\mathbf{X}^{(j+1)}$ using the following steps:

(1) Compute $\hat{\mathbf{D}}_1^2, \ldots, \hat{\mathbf{D}}_N^2$ using

$$\hat{\mathbf{D}}_i^2 = \max(\mathrm{diag}((\mathbf{X}^{(j)})'\mathbf{A}_{i,s}\mathbf{X}^{(j)}), \mathbf{0}), \tag{58}$$

and $\hat{\mathbf{K}}_1, \ldots, \hat{\mathbf{K}}_N$ using

$$\hat{\mathbf{K}}_i = (\mathbf{X}^{(j)})'\mathbf{A}_{i,sk}\mathbf{X}^{(j)}, \tag{59}$$

where $\mathbf{A}_{i,s} = (\mathbf{A}_i + \mathbf{A}_i')/2$ and $\mathbf{A}_{i,sk} = (\mathbf{A}_i - \mathbf{A}_i)/2$.

(2) Let

$$\mathbf{G} = \sum_{i=1}^{N} (\mathbf{A}_i'\mathbf{X}^{(j)}\hat{\mathbf{B}}_i + \mathbf{A}_i\mathbf{X}^{(j)}\hat{\mathbf{B}}_i'), \tag{60}$$

where $\hat{\mathbf{B}}_i = \hat{\mathbf{D}}_i^2 + \hat{\mathbf{K}}_i$ for $i = 1, \ldots, N$. Then, compute

$$\mathbf{X}^{(j+1)} = \mathbf{U}\mathbf{V}', \tag{61}$$

where $\mathbf{U}$ and $\mathbf{V}$ are such that $\mathbf{G} = \mathbf{U}\mathbf{D}\mathbf{V}$ is the SVD of $\mathbf{G}$.

It is straightforward to accelerate Algorithm 6 with the MPE method. We call this accelerated algorithm **Algorithm 6′**.

Remark 9 The above algorithm for three-way GIPSCAL can easily be modified to accommodate an additive constant and the diagonal modification discussed in the previous two subsections.

We compare Algorithm 6′ with Trendafilov (2002) algorithm for three-way GIPSCAL. His algorithm for three-way GIPSCAL works similarly to our description of his algorithm for two-way GIPSCAL in Sect. 2.3. We interpret the LS loss function (57) as an energy functional $E(\mathbf{Y}) = f(\mathbf{X}, \mathbf{D}_1^2, \ldots, \mathbf{D}_N^2, \mathbf{K}_1, \ldots, \mathbf{K}_N)$, where $\mathbf{Y} \in \mathcal{Y} := O(n, p) \times D(p) \times \cdots \times D(p) \times Sk(p) \times \cdots \times Sk(p)$. A gradient dynamic system like (37) is formed, and again the gradient is understood in the sense of continuous manifolds. The resulting set of differential equations are also similar to (44)–(46). (Matrices $\mathbf{A}_s$, $\mathbf{A}_{sk}$, $\mathbf{D}^2$, and $\mathbf{K}$ should have an additional subscript i, and in (44) the gradient has to be summed over i.)

In our final numerical experiment, we compared Algorithm 6, Algorithm 6′ ($k = 5$, 10, 15, and 20), and Trendafilov's. As in Experiment 1, the data were generated by uniform random numbers between −.5 and .5 for a fixed N ($N = 10$), but varying n at three levels (5, 10, and 20). These data were analyzed with dimensionalities $p = 2$ for $n = 5$, $p = 3$ for $n = 10$, and $p = 4$ for $n = 20$. For Algorithms 6 and 6′, initial estimates for $\mathbf{X}$ remained the same as in Experiment 1. The convergence criterion was also similar to the one used in Experiment 1, although it was normalized by the sum of squared norms of the N data matrices. Trendafilov's algorithm used the same convergence criterion as in Experiment 1. Initial estimates were also obtained similarly to Experiment 1, except that for $\mathbf{X}$, matrix $\mathbf{A}_s$ is replaced by the average of $\mathbf{A}_{i,s}$ over i, and for $\mathbf{D}_i^2$ and $\mathbf{K}_i$ ($i = 1, \ldots, \mathbf{N}$), $\mathbf{A}_s$ and $\mathbf{A}_{sk}$ are replaced by $\mathbf{A}_{i,s}$ and $\mathbf{A}_{i,sk}$, respectively.

The results are given in Table 3. Again the basic construction of the table remains the same as in Table 1. Algorithm 6′ with $k = 10$ is roughly 10 times faster than Trendafilov's algorithm across all conditions. This is despite the fact that the latter used a more lenient convergence criterion, and a semi-rational starting point. The

Table 3 The mean cpu time by Algorithms 6 and 6′ and Trendafilov (2002) algorithm for three-way GIPSCAL

n	p		Algorithm 6	Algorithm 6′				Trendafilov
				$k = 5$	$k = 10$	$k = 15$	$k = 20$	
5	2	cpu	0.0652	0.0435	0.0358	0.0430	0.0506	0.3250
		fit	(.8203)	(.8210)	(.8208)	(.8210)	(.8203)	(.8244)
10	3	cpu	0.1407	0.1106	0.0876	0.0712	0.0897	0.8849
		fit	(.8683)	(.8686)	(.8687)	(.8684)	(.8682)	(.8701)
20	4	cpu	0.4567	0.2318	0.2251	0.2015	0.2746	3.1930
		fit	(.9198)	(.9199)	(.9199)	(.9199)	(.9200)	(.9204)

quality of solutions in terms of average fit is slightly better for Algorithms 6 and 6′ than Trendafilov's, although the difference is relatively minor in all cases.

As in Experiment 1, we also investigated the seriousness of suboptimal solutions for three-way GIPSCAL. Probabilities of suboptimal solutions were obtained in a manner similar to those in two-way GIPSCAL. (Algorithm 6′ with $k = 10$ was run 50 times for each data set with 50 random initials to identify a globally optimal solution for each data set. The optimal solution was then compared to a solution obtained in a single run, and the probabilities of suboptimal solutions were calculated over the 250 data sets in each condition.) These probabilities are .268, .632, and .812 for the three conditions in Table 3. The probability of suboptimal solutions goes up very quickly, as more parameters are estimated. (In this case, we are not sure which is more influential, n or p.) That the probability of suboptimal solutions is over .80 is rather daunting. However, this is for completely random data. It should be much smaller for data with some three-way GIPSCAL structures.

5 Concluding remarks

In this paper, we have discussed several related algorithms for analysis of square asymmetric tables. We studied GIPSCAL, GIPSCAL-c, off-diagonal DEDICOM/GIPSCAL, and three-way GIPSCAL. For each model, we have given an iteration which converges to a stationary point, and an MPE acceleration of the same algorithm. We have also compared our algorithms with respective algorithms by Trendafilov (2002), and in the case of off-diagonal DEDICOM, with ten Berge and Kiers (1989) minres-like algorithm. Our numerical experiments show that our new MPE algorithms are very efficient and compare favorably with existing algorithms.

It should be noted that monotonic convergence is not assured of the proposed MPE algorithms, and consequently no theoretical proof of convergence is given. This is obviously a weakness of the algorithms. However, nonmonotonic convergence is rare in practice, and nonconvergence is even rarer. We therefore argue that the benefit of the proposed algorithms far exceeds the weakness, as amply demonstrated in the empirical studies reported.

In addition to the speed advantage, the MPE method is easily adaptable to other similar situations. There are many models in science other than GIPSCAL that may potentially benefit from the MPE method. The MPE method has been successfully incorporated into iterative algorithms for two-way single-domain DEDICOM (Takane and Zhang 2009), and orthogonal INDSCAL (Takane et al. 2010). We can readily give a couple more examples from quantitative psychology (in which the second author of this paper is a specialist): Communality estimation in common factor analysis, and orthogonal and oblique factor rotation problems. It will be of great interest to compare the MPE algorithms with non-accelerated algorithms for these problems.

Acknowledgments We would like to thank Nicolay Trendafilov at Open University who kindly provided MATLAB codes for his algorithm, and Henk Kiers at the University of Groningen for providing Kiers and Takane's algorithm for generalized GIPSCAL.

References

Chino N (1990) A generailzed inner product model for the analysis of asymmetry. Behaviormetrika 27:25–46

Constantine GA, Gower JC (1978) A graphical representations of asymmetric matrices. Appl Stat 27:297–304

Harshman RA (1978) Models for analysis of asymmetrical relationships among *N* objects or stimuli. In: Paper presented at the first joint meeting of the Psychometric Society and the Society of Mathematical Psychology, Hamilton, Ontario

Harshman RA, Green PE, Wind Y, Lundy ME (1981) A model for the analysis of asymmetric data in marketing research. Market Sci 1:205–242

Jennrich RI (2001) A simple general procedure for orthogonal rotation. Psychometrika 66:289–306

Kiers HAL, Takane Y (1994) A generalization of GIPSCAL for the analysis of nonsymmetric data. J Classif 11:79–99

Loisel S, Takane M (2009) Fast indirect robust generalized method of moments. Comput Stat Data Anal 53:3571–3579

Loisel S, Takane Y (2010) Minimal polynomial extrapolation in MATLAB and in R. (submitted)

Smith DA, Ford WF, Sidi A (1987) Extrapolation methods for vector sequences. SIAM Rev 29:199–233

Takane Y (1985) Diagonal estimation in DEDICOM. In: Proceedings of the 13th annual meeting of the Behaviormetric Society. Behaviormetric Society of Japan, Tokyo, pp 100–101

Takane Y, Zhang Z (2009) Algorithms for DEDICOM: acceleration, deceleration, or neither? J Chemometr 23:364–370

Takane Y, Jung K, Hwang H (2010) An acceleration method for ten Berge et al.'s algorithm for orthogonal INDSCAL. Contr Stat 25:409–428

ten Berge JMF, Kiers HAL (1989) Fitting the off-diagonal DEDICOM model in the least squares sense by a generalization of the Harman and Jones Minres procedure to factor analysis. Psychometrika 54:333–337

Trendafilov N (2002) GIPSCAL revisited: a projected gradient approach. Stat Comput 12:135–145