Least Squares Euclidean Multidimensional Scaling

Jan de Leeuw

Started October 02, 2020. Last update December 02, 2021

Contents

Note

This book will be expanded/updated frequently. The directory github.com/deleeuw/stress has a pdf version, a html version, the bib file, the complete Rmd file with the codechunks, and the R and C source code. All suggestions for improvement of text or code are welcome, and some would be really beneficial. For example, I only use base R graphics, nothing more fancy, because base graphics is all I know.

All text and code are in the public domain and can be copied, modified, and used by anybody in any way they see fit. Attribution will be appreciated, but is not required. For completeness we include a slighty modified version of the Unlicense as appendix C.

I number and label *all* displayed equations. Equations are displayed, instead of inlined, if and only if one of the following is true.

- They are important.
- They are referred to elsewhere in the text.
- Not displaying them messes up the line spacing.

All code chunks in the text are named. Theorems, lemmas, chapters, sections, subsections and so on are also named and numbered, using bookdown/Rmarkdown.

I have been somewhat hesitant to use lemmas, theorems, and corollaries in this book. But ultimately they enforce precision and provide an excellent organizational tool. If there is a proof of a lemma, theorem, or corollary, it ends with a $\square$.

Another idiosyncracy: if a line in multiline displayed equation ends with "=", then the next line begins with "=". If it ends with "+", then the next line

begin with "+", and if it ends with "-" the next line begins with "+" as well. I'll try to avoid ending a line with "+" or "-", especially with "-", but if it happens you are warned. A silly example is

$$
\begin{aligned}
&(x+y)^2- && (1)\\
&+4x= && (2)\\
&=x^2+y^2-2x= && (3)\\
&=(x-y)^2\geq && (4)\\
&\geq 0. && (5)
\end{aligned}
$$

Just as an aside: if I refer to something that has been mentioned "above" I mean something that comes earlier in the book and "below" refers to anything that comes later. This always confuses me, so I had to write it down.

The dilemma of whether to use "we" or "I" throughout the book is solved in the usual way. If I feel that a result is the work of a group (me, my coworkers, and the giants on whose shoulders we stand) then I use "we". If it's an individual decision, or something personal, then I use "I". The default is "we", as it always should be in scientific writing.

Most of the individual chapters also have some of the necessary mathematical background material, both notation and results, sometimes with specific eleborations that seem useful for the book. Sometimes this background material is quite extensive. Examples are splines, majorization, unweighting, monotone regression, and the basic Zangwill and Ostrowski fixed point theorems we need for convergence analysis of our algorithms.

There is an appendix A with code, and an appendix B with data sets. These contain brief descriptions and links to the supplementary materials directories, which contain the actual code and data.

Something about code and R/C

I will use this note to thank Rstudio, in particular J.J. Allaire and Yihui Xi, for their contributions to the R universe, and for their promotion of open source software and open access publications. Not too long ago I was an ardent LaTeX user, firmly convinced I would never use anything else again in my lifetime. In the same way thatI was convinced before I would never use anything besides, in that order, FORTRAN, PL/I, APL, and (X)Lisp.

And PHP/Apache/MySQL. But I lived too long. And then, in my dotage, lo and behold, R, Rstudio, (R)Markdown, bookdown, blogdown, Git, Github, Netlify came along.

Figure 1: Forrest Young, Bepi Pinner, Jean-Marie Bouroche, Yoshio Takane, Jan de Leeuw at La Jolla, August 1975

Preface

This book is definitely *not* an impartial and balanced review of all of multidimensional scaling (MDS) theory and history. It emphasizes computation, and the mathematics needed for computation. In addition, it is a summary of over 50 years of MDS work by me, either solo or together with my many excellent current or former co-workers and co-authors. It is heavily biased in favor of the smacof formulation of MDS (De Leeuw (1977a), De Leeuw and Heiser (1977), De Leeuw and Mair (2009)), and the corresponding majorization (or MM) algorithms. And, moreover, I am shamelessly squeezing in as many references to my published and unpublished work as possible, with links to the corresponding pdf's if they are available. Thus this book is also a jumpstation into my bibliography.

I have not organized the book along historical lines because most of the early techniques and results have been either drastically improved or completely abandoned. Nevertheless, some personal historical perspective may be useful. I will put most of it in this preface, so uninterested readers can easily skip it.

I got involved in MDS in 1968 when John van de Geer returned from a visit to Clyde Coombs in Michigan and started the Department of Data Theory in the Division of Social Sciences at Leiden University. I was John's first hire, although I was still a graduate student at the time.

Remember that Clyde Coombs was running the Michigan Mathematical Psychology Program, and he had just published his remarkable book "A Theory of Data" (Coombs (1964)). The name of the new department in Leiden was taken from the title of that book, and Coombs was one of the first visitors to give a guest lecture there.

This is maybe the place to clear up some possible misunderstandings about the name "Data Theory". Coombs was mainly interested in a taxonomy of

data types, and in pointing out that "data" were not limited to a table or data-frame of objects by variables. In addition, there were also similarity ratings, paired comparisons, and unfolding data. Coombs also emphasized that data were often non-metric, i.e. ordinal or categorical, and that it was possible to analyze these ordinal or categorical relationships directly, without first constructing numerical scales to which classical techniques could be applied. One of the new techniques discussed in Coombs (1964) was a ordinal form of MDS, in which not only the data but also the representation of the data in Euclidean space were non-metric.

John van de Geer had just published Van de Geer (1967). In that book, and in the subsequent book Van de Geer (1971), he developed his unique geometric approach to multivariate analysis. Relationship between variables, and between variables and individuals, were not just discussed using matrix algebra, but were also visualized in diagrams. This was related to the geometric representations in Coombs' Theory of Data, but it concentrated on numerical data in the form of rectangular matrices of objects by variables.

Looking back it is easy to see that both Van de Geer and Coombs influenced my approach to data analysis. I inherited the emphasis on non-metric data and on visualization. But, from the beginning, I interpreted "Data Theory" as "Data Analysis", with my emphasis shifting to techniques, loss functions, implementations, algorithms, optimization, computing, and programming. This is of interest because in 2020 my former Department of Statistics at UCLA, together with the Department of Mathematics, started a bachelor's program in Data Theory, in which "Emphasis is placed on the development and theoretical support of a statistical model or algorithmic approach. Alternatively, students may undertake research on the foundations of data science, studying advanced topics and writing a senior thesis." This sounds like a nice hybrid of Data Theory and Data Analysis, with a dash of computer science mixed in.

Computing and optimization were in the air in 1968, not so much because of Coombs, but mainly because of Roger Shepard, Joe Kruskal, and Doug Carroll at Bell Labs in Murray Hill. John's other student Eddie Roskam and I were fascinated by getting numerical representations from ordinal data by minimizing explicit least squares loss functions. Eddie wrote his dissertation in 1968 (Roskam (1968)). In 1973 I went to Bell Labs for a year, and Eddie went to Michigan around the same time to work with Jim Lingoes, resulting

in Lingoes and Roskam (1973).

My first semi-publication was De Leeuw (1968c), quickly followed by a long sequence of other, admittedly rambling, internal reports. Despite this very informal form of publication the sheer volume of them got the attention of Joe Kruskal and Doug Carroll, and I was invited to spend the academic year 1973-1974 at Bell Laboratories. That visit somewhat modified my cavalier approach to publication, but I did not become half-serious in that respect until meeting with Forrest Young and Yoshio Takane at the August 1975 US-Japan seminar on MDS in La Jolla. Together we used the alternating least squares approach to algorithm construction that I had developed since 1968 into a quite formidable five-year publication machine, with at its zenith Takane, Young, and De Leeuw (1977).

In La Jolla I gave the first presentation of the majorization method for MDS, later known as smacof, with the first formal convergence proof. The canonical account of smacof was published in a conference paper (De Leeuw (1977a)). Again I did not bother to get the results into a journal or into some other more effective form of publication. The basic theory for what became known as smacof was also presented around the same time in another book chapter De Leeuw and Heiser (1977).

In 1978 I was invited to the Fifth International Symposium on Multivariate Analysis in Pittsburgh to present what became De Leeuw and Heiser (1980). There I met Nan Laird, one of the authors of the basic paper on the EM algorithm (Dempster, Laird, and Rubin (1977)). I remember enthusiastically telling her on the conference bus that EM and smacof were both special case of the general majorization approach to algorithm construction, which was consequently born around the same time. But that is a story for a companion volume, which currently only exists in a very preliminary stage (https://github.com/deleeuw/bras).

My 1973 PhD thesis (De Leeuw (1973a), reprinted as De Leeuw (1984a)) was actually my second attempt at a dissertation. I had to get a PhD, any PhD, before going to Bell Labs, because of the difference between the Dutch and American academic title and reward systems. I started writing a dissertation on MDS, in the spirit of what later became De Leeuw and Heiser (1982). But halfway through I lost interest and got impatient, and I decided to switch to nonlinear multivariate analysis. This second attempt did produced a finished dissertation (De Leeuw (1973a)), which grew over

time, with the help of multitudes, into Gifi (1990). But that again is a different history, which I will tell some other time in yet another companion volume (https://github.com/deleeuw/gifi). For a long time I did not do much work on MDS, until the arrival of Patrick Mair and the R language led to a resurgence of my interest, and ultimately to De Leeuw and Mair (2009) and Mair, Groenen, and De Leeuw (2019).

I consider this MDS book to be a summary and extension of the basic papers De Leeuw (1977a), De Leeuw and Heiser (1977), De Leeuw and Heiser (1980), De Leeuw and Heiser (1982), and De Leeuw (1988) (published version of De Leeuw (1984b)), all written 30-40 years ago. Footprints in the sands of time. It can also be seen as an elaboration of the more mathematical and computational sections of the excellent and comprehensive textbook of Borg and Groenen (2005). That book has much more information about the origins, the data, and the applications of MDS, as well as on the interpretation of MDS solutions. In this book I concentrate almost exclusively on the mathematical, computational, and programming aspects of MDS.

For those who cannot get enough of me, there is a data base of my published and unpublished reports and papers since 1965, with links to pdf's, at https://jansweb.netlify.app/publication/.

There are many, many people I have to thank for my scientific education. Sixty years is a long time, and consequently many excellent teachers and researchers have crossed my path. I will gratefully mention the academics who had a major influence on my work and who are not with us any more, since I will join them in the not too distant future: Louis Guttman (died 1987), Clyde Coombs (died 1988), Warren Torgerson (died 1999), Forrest Young (died 2006), John van de Geer (died 2008), Joe Kruskal (died 2010), Doug Carroll (died 2011), and Rod McDonald (died 2012).

Notation and Reserved Symbols

intro

0.1 Spaces

- $\mathbb{R}^n$ is the space of all real vectors, i.e. all n-element tuples of real numbers. Typical elements of $\mathbb{R}^n$ are x, y, z. The element of x in position i is x_i. Defining a vector by its elements is done with $x = \{x_i\}$.

- $\mathbb{R}^n$ is equipped with the inner product $\langle x, y\rangle = x'y = \sum_{i=1}^n x_i y_i$ and the norm $\|x\| = \sqrt{x'x}$.

- The canonical basis for $\mathbb{R}^n$ is the $n-$tuple $(e_1, cdots, e_n)$, where e_i has element i equal to $+1$ and all other elements equal to zero. Thus $\|e_i\| = 1$ and $\langle e_i, e_j\rangle = \delta^{ij}$, with δ^{ij} the Kronecker delta (equal to one if $i = j$ and zero otherwise). Note that $x_i = \langle e_i, x\rangle$.

- $\mathbb{R}$ is the real line and $\mathbb{R}_+$ is the half line of non-negative numbers.

- $\mathbb{R}^{n\times m}$ is the space of all $n \times m$ real matrices. Typical elements of $\mathbb{R}^{n\times m}$ are A, B, C. The element of A in row i and column j is a_{ij}. Defining a matrix by its elements is done with $A = \{a_{ij}\}$.

- $\mathbb{R}^{n\times m}$ is equipped with the inner product $\langle A, B\rangle = \text{tr}A'B = \sum_{i=1}^n \sum_{j=1}^m a_{ij} b_{ij}$ and the norm $\|A\| = \sqrt{\text{tr } A'A}$.

- The canonical basis for $\mathbb{R}^{n\times m}$ is the $nm-$tuple $(E_{11}, cdots, E_{nm})$, where E_{ij} has element (i, j) equal to $+1$ and all other elements equal to zero. Thus $\|E_{ij}\| = 1$ and $\langle E_{ij}, E_{kl}\rangle = \delta^{ik}\delta^{jl}$.

vec and vec^{-1}

0.2 Matrices

- $a_{i\bullet}$ is row i of matrix A, $a_{\bullet j}$ is column j.
- $a_{i\star}$ is the sum of row i of matrix A, $a_{\star j}$ is the sum of column j.
- A' is the transpose of A, and $\text{diag}(A)$ is the diagonal matrix with the diagonal elements of A. The inverse of a square matrix A is A^{-1}, the Moore-Penrose generalized inverse of any matrix A is A^{+}.
- If A and B are two $n \times m$ matrices then their Hadamard (or elementwise) product $C = A \times B$ has elements $c_{ij} = a_{ij}b_{ij}$. The Hadamard quotient is $C = A/B$, with elements $c_{ij} = a_{ij}/b_{ij}$. The Hadamard power is $A^{(k)} = A^{(p-1)} \times A$.
- DC matrices. Centering matrix. $J_n = I_n - n^{-1}E_n$. We do not use gthe subscripts if the order is obvious from the context.

0.3 Functions

- $f, g, h, \cdots$ are used for functions or mappings. $f : X \to Y$ says that f maps X into Y.
- σ is used for all real-valued least squares loss functions.

0.4 MDS

- $\Delta = \{\delta_{ij\cdots}\}$ is a matrix or array of dissimilarities.

- $\langle \mathbb{X}, d \rangle$ is a metric space, with $d : \mathcal{X} \otimes \mathcal{X} \to \mathbb{R}_+$ the distance function. If X is is an ordered n-tuple $(x_1, \cdots, x_n)$ of elements of $\mathcal{X}$ then $D(X)$ is $\{d(x_i, x_j)\}$, the elements of which we also write as $d_{ij}(X)$.
- Summation over the elements of vector $x \in \mathbb{R}^n$ is $\sum_{i=1}^n x_i$. Summation over the elements of matrix $A \in \mathbb{R}^{n \times m}$ is $\sum_{i=1}^n \sum_{j=1}^m a_{ij}$. Summation over the elements above the diagonal of A is $\sum \sum_{1 \leq i < j \leq n} a_{ij}$.
- Conditional summation is, for example, $\sum_{i=1}^n \{x_i \mid x_i > 0\}$.

Iteration

Chapter 1

Introduction

In this book we study the smacof family of *Multidimensional Scaling (MDS)* techniques. In MDS the data consist of some type of information about the *dissimilarities* between a pairs of *objects*. These objects can be anything: individuals, variables, colors, locations, chemicals, molecules, works of Plato, political parties, Morse code signals, and so on. The dissimilarities can be approximate or imprecise distances, dissimilarity judgments, or sociometric choices. They generally are *distance-like*, but we do not expect them to satisfy the the triangle inequality, and in general not even non-negativity and symmetry. *Similarities*, such as confusion probabilities, correlations, or preferences, are always converted in some way or another to dissimilarities before they can serve as data for MDS.

The information we have about these dissimilarities can be numerical, ordinal, or categorical. Thus we may have the actual values of some or all of the dissimilarities, we may know their rank order, or we may have a classification of them into a small number of qualitative bins.

Let's formalize this, and introduce some notation at the same time. The set of ojects is $\mathfrak{O}$. For example, it can be the set of all cities with more than 10,000 inhabitants. In our MDS analysis we only use $O := (o_1, \cdots, o_n)$, an n-tuple (i.e. a finite sequence) of n *different* elements of $\mathfrak{O}$, for example n capital cities selected from $\mathfrak{O}$. If you want to, you can call O a *sample* from $\mathfrak{O}$. It is entirely possible, however, that $\mathfrak{O}$ has only n elements, in which case O is just an permutation of the elements of $\mathfrak{O}$.

A dissimilarity is a function δ on all pairs of objects, with values in a set

$\mathfrak{D}$. It can be, for example, the time in seconds for an airline flight from city one to city two. Thus $\delta : \mathfrak{O} \otimes \mathfrak{O} \Rightarrow \mathfrak{D}$. A dissimilaritry is *numerical* if $\mathfrak{D}$ is subset of real line, it is *ordinal* if $\mathfrak{D}$ is a partially ordered set, and it is *nominal* if $\mathfrak{D}$ is neither. Or a dissimilarty is nominal if $\mathfrak{D}$ is any set, and we choose to ignore the ordinal and numerical information if it is there. No matter what $\mathfrak{D}$ is, we suppose it always has the element NA to indicate missing dissimilarities. Cities may not have airports, for example, or we just don't have the information about the airline distances. Define $\delta_{ij} := \delta(o_i, o_j)$ and $\Delta := \delta(O \times O)$. We can think of Δ and an $n \times n$ matrix with elements in $\mathfrak{D}$.

MDS techniques map the objects o_i into *points* x_i in some metric space $\langle \mathfrak{X}, d \rangle$ in such a way that the distances between pairs of points approximate the dissimilarities of the corresponding pairs of objects. Thus we want to find a map $x : \mathfrak{O} \to \mathfrak{X}$ that produces an n-tuple $X = (x_1, \cdots, x_n)$ of elements of $\mathfrak{X}$, where $x_i := x(o_i)$. Also define $d_{ij} := d(x_i, x_j)$ and $D(X) := d(X \times X$. Unlike the dissimilarities the d_{ij} are always numerical, because distances are. So MDS finds X such that $D(X) \approx \Delta$.

For numerical dissimilarities it is clear what "approximation" means, we simply want the distances and the corresponding dissimilarities to be numerically close. Because there are generally many dissimilarities and distances a combined measure of closeness can still be defined in many different ways. For ordinal and nominal dissimilarities the notion of approximation is less clear, and we have to develop more specialized techniques to measure how well the distances fit the dissimilarities.

1.1 Brief History

De Leeuw and Heiser (1980)

This section has a different emphasis. We limit ourselves to developments in Euclidean MDS, and to contributions with direct computational consequences that have a direct or indirect link to psychometrics, and to work before 1960. This is reviewed ably in the presidential address of W. S. Torgerson (1965).

Our history review takes the form of brief summaries of what we consider to be milestone papers or books.

1.1.1 Milestones

W. S. Torgerson (1952) W. S. Torgerson (1965)

Shepard (1962a) Shepard (1962b)

Kruskal (1964a) Kruskal (1964b)

Guttman (1968)

De Leeuw (1977a) De Leeuw and Heiser (1977)

There was some early work by Richardson, Messick, Abelson and Torgerson who combined Thurstonian scaling of similarities with the mathematical results of Schoenberg (1935) and G. Young and Householder (1938).

Despite these early contributions it makes sense, certainly from the point of view of my personal history, but probably more generally, to think of MDS as starting as a widely discussed, used, and accepted technique since the book by W. S. Torgerson (1958). This was despite the fact that in the fifties and sixties computing eigenvalues and eigenvectors of a matrix of size 20 or 30 was still a considerable challenge.

A few years later the popularity of MDS got a large boost by developments centered at Bell Telephone Laboratories in Murray Hill, New Jersey, the magnificent precursor of Silicon Valley. First there was nonmetric MDS by Shepard (1962a), Shepard (1962b) and Kruskal (1964a), Kruskal (1964b), And later another major development was the introduction of individual difference scaling by Carroll and Chang (1970) and Harshman (1970). Perhaps even more important was the development of computer implementations of these new techniques. Some of the early history of nonmetric MDS is in De Leeuw (2017e).

Around the same time there were interesting theoretical contributions in Coombs (1964), which however did not much influence the practice of MDS. ..... And several relatively minor variations of the Bell Laboratories approach were proposed by Guttman (1968), but Guttman's influence on further MDS implementations turned out to be fairly localized and limited.

The main development in comptational MDS after the Bell Laboratories surge was probably smacof. Initially, in De Leeuw (1977a), this stood for *Scaling by Maximizing a Convex Function*. Later it was also used to mean

Scaling by Majorizing a Complicated Function. Whatever. In this book smacof just stands for smacof. No italics, no boldface, no capitals.

The first smacof programs were written in 1977 in FORTRAN at the Department of Data Theory in Leiden (Heiser and De Leeuw (1977)). Eventually they migrated to SPSS (for example, Meulman and Heiser (2012)) and to R (De Leeuw and Mair (2009)). The SPSS branch and the R branch have diverged somewhat, and they continue to be developed independently.

Parallel to this book there is an attempt to rewrite the various smacof programs in C, with the necessary wrappers to call them from R (De Leeuw (2017f)). The C code, with makefiles and test routines, is at github.com/deleeuw/smacof

1.2 Basic MDS

Following Kruskal, and to a lesser extent Shepard, we measure the fit of distances to dissimilarities using an explicit real-valued *loss function* (or *badness-of-fit measure*), which is minimized over the possible maps of the objects into the metric space. This is a very general definition of MDS, covering all kinds of variations of the target metric space and of the way fit is measured. Obviously we will not discuss *all* these possible forms of MDS, which also includes various techniques more properly discussed as cluster analysis, classification, or discrimination.

To fix our scope we first define *basic MDS*, which is short for *Least Squares Euclidean Metric MDS*. It is defined as MDS with the following characteristics.

1. The metric space is a Euclidean space.
2. The dissimilarities are numerical, symmetric, and non-negative.
3. The loss function is a weighted sum of squares of the *residuals*, which are the differences between dissimilarities and Euclidean distances.
4. Weights are numerical, symmetric, and non-negative.
5. Self-dissimilarities are zero and the corresponding terms in the loss function also have weight zero.

By a *Euclidean space* we mean a finite dimensional vector space, with addition and scalar multiplication, and with an inner product that defines the

distances. For the *inner product* of vectors x and y we write $\langle x, y\rangle$. The *norm* of x is defined as $\|x\| := \sqrt{\langle x, x\rangle}$, and the *distance* between x and y is $d(x, y) := \|x - y\|$.

The *loss function* we use is called *stress*. It was first explicitly introduced in MDS as *raw stress* by Kruskal (1964a) and Kruskal (1964b). We define stress in a slightly different way, because we want to be consistent over the whole range of the smacof versions and implementations. In smacof stress is the real-valued function σ, defined on the space $\mathbb{R}^{n\times p}$ of configurations, as

$$\sigma(X) := \frac{1}{4}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(\delta_{ij} - d_{ij}(X))^2. \tag{1.1}$$

Note that we use := for definitions, i.e. for concepts and symbols that are not standard mathematical usage, when they occur for the first time in this book. Through the course of the book it will probably become clear why the mysterious factor $\frac{1}{4}$ is there. Clearly it has no influence on the actual minimization of the loss function.

In definition (1.1) we use the following objects and symbols.

1. $W = \{w_{ij}\}$ is a symmetric, non-negative, and hollow matrix of *weights*, where *hollow* means zero diagonal.
2. $\Delta = \{\delta_{ij}\}$ is a symmetric, non-negative, and hollow matrix of *dissimilarities*.
3. X is an $n \times p$ *configuration*, containing coordinates of n *points* in p dimensions.
4. $D(X) = \{d_{ij}(X)\}$ is a symmetric, non-negative, and hollow matrix of *Euclidean distances* between the n points in X. Thus $d_{ij}(X) := \sqrt{\sum_{s=1}^{p}(x_{is} - x_{js})^2}$.

Note that symmetry and hollowness of the basic objects W, Δ, and D allows us carry out the summation of the weighted squared residuals in formula (1.1) over the upper (or lower) diagonal elements only. Thus we can also write

$$\sigma(X) := \frac{1}{2}\sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij} - d_{ij}(X))^2. \tag{1.2}$$

We use the notation $\sum\sum_{1\leq i<j\leq n}$ for summation over the lower-diagonal elements of a matrix.

The function D, which computes the distance matrix $D(X)$ from a configuration X, is matrix-valued. It maps the $n \times p$-dimensional *configuration space* $\mathbb{R}^{n\times p}$ into the set $D(\mathbb{R}^{n\times p})$ of Euclidean distance matrices between n points in $\mathbb{R}^p$, which is a subset of the convex cone of hollow, symmetric, non-negative matrices in the linear space $\mathbb{R}^{n\times n}$ (Datorro (2015)).

In basic MDS the weights and dissimilarities are given numbers, and we minimize stress over all $n \times p$ configurations X. Note that the *dimensionality* p is also supposed to be known beforehand, and that MDS in p dimensions is different from MDS in $q \neq p$ dimensions. We sometimes emphasize this by writing $pMDS$, which indicates that we will map the points into p-dimensional space.

Two boundary cases that will interest us are *Unidimensional Scaling* or *UDS*, where $p = 1$, and *Full-dimensional Scaling* or *FDS*, where $p = n$. Thus UDS is 1MDS and FDS is nMDS. Most actual MDS applications in the sciences use 1MDS, 2MDS or 3MDS, because configurations in one, two, or three dimensions can easily be plotted with standard graphics tools. Note that MDS is not primarily a tool to tests hypotheses about dimensionality and to find meaningful dimensions. It is a mostly a mapping tool for data reduction, to graphically find interesting aspects of dissimilarity matrices.

The projections on the dimensions are usually ignored, it is the configuration of points that is the interesting outcome. This distinguishes MDS from, for example, factor analysis. There is no Varimax, Oblimax, Quartimax, and so on. Exceptions are confirmatory applications of MDS in genetic mapping along the chromosome, in archeological seriation, in testing psychological theories of cognition and representation, in the conformation of molecules, and in geographic and geological applications. In these areas the dimensionality and general structure of the configuration are given by prior knowledge, we just do not know the precise location and distances of the points. For more discussion of the different uses of MDS we refer to De Leeuw and Heiser (1982).

1.2.1 Kruskal's stress

Definition (1.1) differs from Kruskal's original stress in at least three ways: in Kruskal's use of the square root, in our use of weights, and in our different approach to normalization.

We have paid so much attention to Kruskal's original definition, because the choices made there will play a role in the normalization discussion in the ordinal scaling chapter (section 7.4.1), in the comparison of Kruskal's and Guttman's approach to ordinal MDS (sections 10.3 and 10.4), and in our discussions about the differences between Kruskal's stress (10.15) and smacof's stress (1.1) in the next three sections of this chapter.

1.2.1.1 Square root

Let's discuss the square root first. Using it or not using it does not make a difference for the minimization problem. Using the square root, however, does give a more sensible root-mean-square scale, in which stress is homogeneous of degree one, instead of degree two. But I do not want to compute all those unnecessary square roots in my algorithms, and I do not want to drag them along through my derivations. Moreover the square root potentially causes problems with differentiability at those X where $\sigma(X)$ is zero. Thus, througout the book, we do not use the square root in our formulas and derivations. In fact, we do not even use it in our computer programs, except at the very last moment when we return the final stress after the algorithm has completed.

1.2.1.2 Weights

There were no weights $W = \{w_{ij}\}$ in the original definition of stress by Kruskal (1964a), and neither are they there in most of the basic later contributions to MDS by Guttman, Lingoes, Roskam, Ramsay, or Young. We will use weights throughout the book, because they have various interesting applications within basic MDS, without unduly complicating the derivations and computations. In Groenen and Van de Velden (2016), section 6, the various uses of weights in the stress loss function are enumerated. They generously, but correctly, attribute the consistent use of weights in MDS to me. I quote from their paper:

1. Handling missing data is done by specifying $w_{ij} = 0$ for missings and 1 otherwise thereby ignoring the error corresponding to the missing dissimilarities.
2. Correcting for nonuniform distributions of the dissimilarities to avoid dominance of the most frequently occurring dissimilarities.
3. Mimicking alternative fit functions for MDS by minimizing Stress with w_{ij} being a function of the dissimilarities.
4. Using a power of the dissimilarities to emphasize the fitting of either large or small dissimilarities.
5. Special patterns of weights for specific models.
6. Using a specific choice of weights to avoid nonuniqueness.

In some situations, for example for huge data sets, it is computationally convenient, or even necessary, to minimize the influence of the weights on the computations. We can use *majorization* to turn the problem from a weighted least squares problem to an iterative unweighted least squares problem. The technique, which we call *unweighting*, is discussed in detail in section 5.4.10.

1.2.1.3 Normalization

This section deals with a rather trivial problem, which has however caused problems in various stages of smacof's 45-year development history. Because the problem is trivial, and the choices that must be made are to a large extent arbitrary, it has been overlooked and somewhat neglected.

In basic MDS we scale the weights and dissimilarities. It is clear that if we multiply the weights or dissimilarities by a constant, then the optimal approximating distances $D(X)$ and the optimal configuration X will be multiplied by the same constant. That is exactly why Kruskal's raw stress had to be normalized. Consequently we in basic MDS we always scale weights and dissimilarities by

$$\sum_{1\leq i<j\leq n}\sum w_{ij} = 1, \tag{1.3}$$

$$\sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}^2 = 1. \tag{1.4}$$

This simplifies our formulas and makes them look better (see, for example, section 2.1.1 and section 2.3.0.2). It presupposes, of course, that $w_{ij}\delta_{ij} \neq 0$ for at least one $i \neq j$, which we will happily assume in the sequel, because otherwise the MDS problem is trivial. Note that if all weights are equal (which we call the *unweighted case*) then they are equal to $1/\binom{n}{2}$ and thus we require $\sum\sum_{1\leq i<j\leq n}\delta_{ij}^2 = \frac{1}{2}n(n-1)$.

Using normalized dissimilarities amounts to the same defining stress as

$$\sigma(X) = \frac{1}{2}\frac{\sum\sum_{1\leq i<j\leq n} w_{ij}(\delta_{ij}^2 - d_{ij}(X))^2}{\sum\sum_{1\leq i<j\leq n} w_{ij}\delta_{ij}^2}. \tag{1.5}$$

This is useful to remember when we discuss the various normalizations for non-metric MDS in section 7.4.1.

1.3 Local and Global

In basic MDS our goal is to compute both $\min_X \sigma(X)$ and $\operatorname{Argmin}_X \sigma(X)$, where $\sigma(X)$ is defined as (1.1), and where we minimize over all configurations in $\mathbb{R}^{n\times p}$.

In this book we study both the properties of the stress loss function and a some of its generalizations, and the various ways to minimize these loss functions over configurations (and sometimes over transformations of the dissimilarities as well).

Emphasis local minima

Compute stationary points

Note we use the notation $\operatorname{Argmin}_{x\in X} f(x)$ for the set of minimizers of f over X. Thus $z \in \operatorname{Argmin}_{x\in X} f(x)$ means z minimizes f over X, i.e. $f(z) = \min_{x\in X} f(x)$. If it is clear from theory that the minimum is necessarily unique, we use argmin instead of Argmin.

1.4 Generalizations

The most important generalizations of basic MDS we will study in later chapters of this book are discussed briefly in the following sections.

1.4.1 Non-metric MDS

Basic MDS is a form of *Metric Multidimensional Scaling* or *MMDS*, in which dissimilarities are either known or missing. In chapter 7 we relax this assumption. Dissimilarities may be partly known, for example we may know they are in some interval, we may only know their order, or we may know them up to some smooth transformation. MDS with partly known dissimilarities is *Non-metric Multidimensional Scaling* or *NMDS*. Completely unknown (missing) dissimilarities are an exception, because we can just handle this in basic MDS by setting the corresponding weights equal to zero.

In NMDS we minimize stress over all configurations, but also over the unknown dissimilarities. What we know about them (the interval they are in, the transformations that are allowed, the order they are in) defines a subset of the space of non-negative, hollow, and symmetric matrices. Any matrix in that subset is a matrix of what Takane, Young, and De Leeuw (1977) call *disparities*, i.e. imputed dissimilarities. The imputation provides the missing information and transforms the non-numerical information we have about the dissimilarities into a numerical matrix of disparities. Clearly this is an *optimistic imputation*, in the sense that it chooses from the set of admissible disparities to minimize stress (for a given configuration).

One more terminological point. Often *non-metric* is reserved for ordinal MDS, in which we only know a (partial or complete) order of the dissimilarities. Allowing linear or polynomial transformations of the dissimilarities, or estimating an additive constant, is then supposed to be a form of metric MDS. There is something to be said for that. Maybe it makes sense to distinguish non-metric *in the wide sense* (in which stress must be minimized over both X and Δ) and *non-metric in the narrow sense* in which the set of admissible disparities is defined by linear inequalities. Nonmetric in the narrow sense will also be called *ordinal MDS* or *OMDS*.

It is perhaps useful to remember that Kruskal (1964a) introduced explicit loss functions in MDS to put the somewhat heuristic NMDS techniques of

Shepard (1962a) onto a firm mathematical and computational foundation. Thus, more or less from the beginning of iterative least squares MDS, there was a focus on non-metric rather than metric MDS, and this actually contributed a great deal to the magic and success of the technique. In this book most of the results are derived for basic MDS, which is metric MDS, with non-metric MDS as a relatively straightforward extension not discussed until chapter 7. So, at least initially, we take the numerical values of the dissimilarities seriously, as do W. S. Torgerson (1958) and Shepard (1962a), Shepard (1962b).

It may be the case that in the social and behavioural sciences only the ordinal information in the dissimilarities is reliable and useful. But, since 1964, MDS has also been applied in molecular conformation, chemometrics, genetic sequencing, archelogical seriation, and in network design and location analysis. In these areas the numerical information in the dissimilarities is usually meaningful and should not be thrown out right away. Also, the use of the Shepard plot, with dissimilarities on the horizontal axis and fitted distances on the vertical axis, suggests there is more to dissimilarities than just their rank order.

1.4.2 fstress

Instead of defining the residuals in the least squares loss function as $\delta_{ij} - d_{ij}(X)$ chapter 22 discusses the more general cases where the residuals are $f(\delta_{ij}) - f(d_{ij}(X))$ for some known non-negative increasing function f. This defines the *fstress* loss function.

If $f(x) = x^r$ with $r > 0$ then fstress is called *rstress*. Thus stress is rstress with $r = 1$, also written as *1stress* or σ_1. In more detail we will also look at $r = 2$, which is called *sstress* by Takane, Young, and De Leeuw (1977). In chapter 21 we look at the problem of minimizing sstress and weighted version *strain*. The case of rstress with $r \to 0$ is also of interest, because it leads to the loss function in Ramsay (1977).

1.4.3 Constraints

Instead of minimizing stress over all X in $\mathbb{R}^{n\times p}$ we will look in chapter 15 at various generalizations where minimization is over a subset $\otimes$ of $\mathbb{R}^{n\times p}$. This

is often called *Constrained Multidimensional Scaling* or *CMDS*.

The distinction may be familiar from factor analysis, where we distinguish between exploratory and confirmatory factor analysis. If we have prior information about the parameters then incorporating that prior information in the analysis will generally lead to more precise and more interpretable estimates. The risk is, of course that if our prior information is wrong, if it is just prejudice, then we will have a solution which is precise but incorrect. We have the famous trade-off between bias and variance. In MDS this trade-off does not seem to apply directly, because the necessary replication frameworks are missing.

and we do not attach much value to locating the true configuration.

$$\min_{X\in\Omega}\sigma(X)$$

$$\min_{X}\sigma(X)+\lambda\kappa(X,\Omega)$$

where $\kappa(X,\Omega)\geq 0$ and $\kappa(X,\Omega)=0$ if and only if $X\in\Omega$.

1.4.4 Individual Differences

Now consider the situation in which we have m different dissimilarity matrices Δ_k and m different weight matrices W_k. We generalize basic MDS by defining

$$\sigma(X_1,\cdots,X_m):=\frac{1}{2}\sum_{k=1}^m\sum_{1\leq i<j\leq n}w_{ijk}(\delta_{ijk}-d_{ij}(X_k))^2, \tag{1.6}$$

and minimize this over the X_k.

There are two simple ways to deal with this generalization. The first is to put no further constraints on the X_k. This means solving m separate basic MDS problems, one for each k. The second way is to require that all X_k are equal. As shown in more detail in section 16.2.2 this reduced to a single basic MDS problem with dissimilarities that are a weighted sum of the Δ_k. So both these approaches do not really bring anything new.

Minimizing (1.6) becomes more interesting if we constrain the X_k in various ways. Usually this is done by making sure they have a component that

is common to all k and a component that is specific or unique to each k. This approach, which generalizes constrained MDS, is discussed in detail in chapter 16.

1.4.5 Distance Asymmetry

We have seen in section 5.4.6 of this chapter that in basic MDS the assumption that W and Δ are symmetric and hollow can be made without loss of generality. The simple partitioning which proved this was based on the fact that $D(X)$ is always symmetric and hollow. By the way, the assumption that W and D are non-negative cannot be made without loss of generality, as we will see below.

In chapter 17 we relax the assumption that $D(X)$ is symmetric (still requiring it to be non-negative and hollow). This could be called *Asymmetric MDS*, or *AMDS*. I was reluctant at first to include this chapter, because asymmetric distances do not exist. And certainly are not Euclidean distances, so they are not covered by the title of this book. But as long as we stay close to Euclidean distances, least squares, and the smacof approach, I now feel reasonably confident the chapter is not too much of a foreign body.

When Kruskal introduced gradient-based methods to minimize stress he also discussed the possibility to use Minkovski metrics other than the Euclidean metric. This certainly was part of the appeal of the new methods, in fact it seemed as if the gradient methods made it possible to use any distance function whatsoever. This initial feeling of empowerment was somewhat naive, because it ignored the seriousness of the local minimum problem, the combinatorial nature of one-dimensional and city block scaling, the problems with nonmetric unfolding, and the problematic nature of gradient methods if the distances are not everywhere differentiable. All these complications will be discussed in this book. But it made me decide to ignore Minkovski distances (and hyperbolic and elliptic non-Euclidean distances), because life with stress is complicated and challenging enough as it is.

1.5 Models and Techniques

Truth, error

Chapter 2

Properties of Stress

2.1 Notation

The notation used in the smacof approach to MDS first appeared in De Leeuw (1977a), and was subsequently used in several of the later key smacof references, such as De Leeuw and Heiser (1982), De Leeuw (1988), chapter 8 of Borg and Groenen (2005), and De Leeuw and Mair (2009). We follow it in this book.

2.1.1 Expanding

We expand stress by writing out the squares of the residuals and then summing. Define

$$\eta_\delta^2 := \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}^2, \tag{2.1}$$

$$\rho(X) := \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}d_{ij}(X), \tag{2.2}$$

$$\eta^2(X) := \sum_{1\leq i<j\leq n}\sum w_{ij}d_{ij}^2(X). \tag{2.3}$$

More precisely, using conditional summation,

$$\rho(X) := \sum_{1\le i<j\le n}\sum \{w_{ij}\delta_{ij}d_{ij}(X) \mid w_{ij}\delta_{ij} > 0\}, \tag{2.4}$$

$$\eta^2(X) := \sum_{1\le i<j\le n}\sum \left\{w_{ij}d_{ij}^2(X) \mid w_{ij} > 0\right\}. \tag{2.5}$$

Remember that we have normalized by $\eta_\delta^2 = 1$. With our newly defined functions ρ and η^2 we can write stress as

$$\sigma(X) = \frac{1}{2}(1 + \eta^2(X)) - \rho(X). \tag{2.6}$$

The CS inequality implies that for all X

$$\rho(X) = \sum_{1\le i<j\le n}\sum w_{ij}\delta_{ij}d_{ij}(X) \le \eta_\delta\eta(X) = \eta(X), \tag{2.7}$$

and thus, from (2.6),

$$\frac{1}{2}(1 - \eta(X))^2 \le \sigma(X) \le \frac{1}{2}(1 + \eta^2(X)), \tag{2.8}$$

$$\frac{1}{2}(1 - \rho(X))^2 \le \sigma(X) \le \frac{1}{2}(1 - 2\rho(X)). \tag{2.9}$$

2.1.2 Matrix Expressions

Using matrix notation allows us to arrive at compact expressions, which suggest various mathematical and computational shortcuts. In order to use matrix notation for distances we mainly rely on the difference matrices A_{ij}, which we now define.

- A *unit vector* e_i is a vector with element i equal to $+1$ and all other elements equal to 0. A *unit matrix* E_{ij} is a matrix of the form $e_i e_j'$,
- A *diff matrix* A_{ij} is a matrix of the form $(e_i - e_j)(e_i - e_j)'$.

The element in row i and column j of a matrix X is normally referred to as x_{ij}. But in some cases, to prevent confusion, we use the notation $\{X\}_{ij}$. Thus, for example, $\{e_i\}_j = \delta^{ij}$, where δ^{ij} is *Kronecker's delta* (zero when $i = j$ and one otherwise).

The diff matrices A_{ij} with $i \neq j$ have only four non-zero elements

$$\begin{aligned} \{A_{ij}\}_{ii} &= \{A_{ij}\}_{jj} = +1, \\ \{A_{ij}\}_{ij} &= \{A_{ij}\}_{ji} = -1, \end{aligned} \tag{2.10}$$

and all other elements of A_{ij} are zero. Thus $A_{ij} = A_{ji}$ and $A_{ii} = 0$. Diff matrices are symmetric, and positive semidefinite. They are also *doubly-centered*, which means that their rows and columns add up to zero. If $i \not= j$ they are of rank one and have one eigenvalue equal to two, which means $A_{ij}^s = 2^{s-1}A_{ij}$. Also

$$\sum_{1\leq i<j\leq n}\sum A_{ij} = nI - ee' = nJ, \tag{2.11}$$

with J the centering matrix.

We begin our matrix expressions with $d_{ij}^2(X) = \text{tr } X'A_{ij}X$. Define

$$V := \sum_{1\leq i<j\leq n}\sum w_{ij}A_{ij}, \tag{2.12}$$

so that

$$\eta^2(X) = \text{tr } X'VX. \tag{2.13}$$

The matrix V has off-diagonal elements equal to $-w_{ij}$ and diagonal elements $v_{ii} = \sum_{j\neq i} w_{ij}$ It is symmetric, positive semi-definite, and doubly-centered. Thus it is singular, because $Ve = 0$.

To analyze the singularity of V in more detail we observe that $z'Vz = \sum\sum_{1\leq i<j\leq n} w_{ij}(z_i - z_j)^2$. This is zero if and only if all $w_{ij}(z_i - z_j)^2$ are zero. If we permute the elements of z such that $z_1 \leq \cdots \leq z_n$ then the matrix with elements $(z_i - z_j)^2$ can be partitioned such that the diagonal blocks,

corresponding with tie-blocks in z, are zero and the off-diagonal blocks are strictly positive. Thus $z'Vz = 0$ if and only if the corresponding off-diagonal blocks of W are zero. In other words, we can find a z such that $z'Vz = 0$ if and only if W is the direct sum of a number of smaller matrices. If this is not the case we call W *irreducible*, and $z'Vz > 0$ for all $z \neq e$, so that the rank of V is $n - 1$.

If W is reducible the MDS problem separates into a number of smaller independent MDS problems. We will assume in the sequel, without any real loss of generality, that this does not occur, and that consequently W is irreducible.

Because of the singularity of the matrices involved we sometimes have to work with generalized inverses. We limit ourselves to the Moore-Penrose (MP) inverse, which can be defined in terms of the singular value decomposition. If the singular value decomposition is $X = K\Lambda L'$ with $K'K = L'L = I_r$ and Λ a positive definite diagonal matrix of order $r = \text{rank}(X)$, then the MP inverse of X is $X^+ = L\Lambda^{-1}K'$.

Because of irreducibility the MP inverse of V is

$$V^+ = (V + \frac{ee'}{n})^{-1} - \frac{ee'}{n}. \tag{2.14}$$

If all weights are equal, say to w, then $V = nwJ$ and $V^+ = \frac{1}{nw}J$, with J the centering matrix $I - \frac{1}{n}ee'$.

Finding an expression for $\rho(X)$ from (2.2) in matrix form is a bit more complicated. Define

$$r_{ij}(X) := \begin{cases} 0 & \text{if } d_{ij}(X) = 0, \\ \frac{\delta_{ij}}{d_{ij}(X)} & \text{if } d_{ij}(X) > 0, \end{cases} \tag{2.15}$$

and

$$B(X) := \sum_{1 \leq i < j \leq n} \sum w_{ij} r_{ij}(X) A_{ij}. \tag{2.16}$$

Then we have

$$\rho(X) = \text{tr } X'B(X)X. \tag{2.17}$$

Just like V, the matrix-valued function B is symmetric, positive-semidefinite, and doubly-centered. If all dissimilarities and distances are positive then irreducibility of W implies that the rank of $B(X)$ is equal to $n - 1$. Note

that if $\delta_{ij} = d_{ij}(X) > 0$ for all i, j (perfect fit), then the r_{ij} from (2.15) are all equal to one, and $B(X) = V$.

In (2.15) we have set $r_{ij}(X) = 0$ if $d_{ij}(X) = 0$. This is arbitrary. Since $b_{ij}(X) = r_{ij}(X)$ if $d_{ij}(X) = 0$ we get a different matrix $B(X)$ if we choose to set, say, $r_{ij}(X) = 1$ or $r_{ij}(X) = \delta_{ij}$ whenever $d_{ij}(X) = 0$. But

$$B(X)X = \sum_{1\leq i<j\leq n}\sum w_{ij}r_{ij}(X)(e_i - e_j)(x_i - x_j)' \tag{2.18}$$

remains the same, no matter how we choose $r_{ij}(X)$ for the $i < j$ with $d_{ij}(X) = 0$. And, consequently, $\rho(X) = \text{tr } X'B(X)X$ remains the same as well.

We now see, from equation (2.6), that

$$\sigma(X) = 1 - \text{ tr } X'B(X)X + \frac{1}{2}\text{tr } X'VX. \tag{2.19}$$

2.1.3 Coefficient Space

Observe that we distinguish configuration space, which is the linear space $\mathbb{R}^{n\times p}$ of $n \times p$ matrices, from the linear space $\mathbb{R}^{np}$ of np element vectors. The two spaces are isomorphic, and connected by the *vec operator* and its inverse.

Some quick definitions. If $Y \in \mathbb{R}^{n\times p}$ is a configuration, then vec(Y) is an np-element vector obtained by stacking the columns of Y on top of each other. Thus element (i, s) of Y becomes element $i+(s-1)*n$ of vec(Y). If $Z = X+Y$ in $\mathbb{R}^{n\times p}$ then vec$(Z) =$ vec$(X) +$ vec(Y) in $\mathbb{R}^{np}$, and if $Z = \alpha Y$ for some real number α then also vec$(Z) = \alpha$vec(Y). Thus vec is an isomorphism, and so is its inverse vec $* -1$, which transforms an np-element vector into an $n \times p$ matrix. In R we vec a matrix by the as.vector function, which removes the dim attribute from the matrix, and we vec^{-1} a vector by the matrix function, which adds the dim attribute to the vector.

But that is not all. Euclidean spaces are equipped with an inner product and a corresponding metric. The spaces $\mathbb{R}^{n\times p}$ and $\mathbb{R}^{np}$ are also isometric inner product spaces. If x and y are in $\mathbb{R}^{np}$ then their inner product is

$$\langle x, y\rangle_{np} := x'y = \sum_{k=1}^{np} x_k y_k,$$

If X and Y are in $\mathbb{R}^{n\times p}$ their inner product is

$$\langle X, Y\rangle_{n\times p} := \text{tr } X'Y = \sum_{i=1}^{n}\sum_{s=1}^{p} x_{is}y_{is},$$

their lengths are $\|X\| = \sqrt{\text{tr } X'X}$ and $\|Y\| = \sqrt{\text{tr } Y'Y}$, and their distance is $\|X - Y\|$. Now $\langle x, y\rangle = \langle \text{vec}(X), \text{vec}(Y)\rangle$ and $|x-y|=|vec(X)-vec(Y)|.

Some formulas in MDS are more easily expressed in $\mathbb{R}^{np}$ (see, for example, section 2.3), but most of the time we prefer to work in the more intuitive space $\mathbb{R}^{n\times p}$ of configurations (which is after all where our representations and pictures live).

Suppose $Y_1, \cdots, Y_r$ are r linearly independent matrices in configuration space $\mathbb{R}^{n\times p}$. We write $\mathcal{Y}$ for the r-dimensional subspace spanned by the basis $Y_1, \cdots, Y_r$. Of course if $r = np$ then $\mathcal{Y} = \mathbb{R}^{n\times p}$.

If $X \in \mathcal{Y}$ then there is a θ in *coefficient space* $\mathbb{R}^r$ such that $X = \sum_{s=1}^{r}\theta_s Y_s$. We now parametrize basic MDS using the new variables θ. Define

$$\tilde{d}_{ij}^2(\theta) := \text{tr } X'A_{ij}X = \theta'\tilde{A}_{ij}\theta, \tag{2.20}$$

with

$$\{\tilde{A}_{ij}\}_{st} := \text{tr } Y_s'A_{ij}Y_t. \tag{2.21}$$

Now

$$\tilde{B}(\theta) := \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{\tilde{d}_{ij}(\theta)}\tilde{A}_{ij}, \tag{2.22}$$

and

$$\tilde{V} := \sum_{1\leq i<j\leq n}\sum w_{ij}\tilde{A}_{ij}, \tag{2.23}$$

and

$$\tilde{\sigma}(\theta) := 1 - 2\tilde{\rho}(\theta) + \tilde{\eta}^2(\theta) = 1 - 2\ \theta'\tilde{B}(\theta)\theta + \theta'\tilde{V}\theta. \tag{2.24}$$

For the elements of $\tilde{B}$ and $\tilde{V}$ we see

$$\tilde{b}_{st}(\theta) = \text{tr } Y_s' B(X) Y_t, \tag{2.25}$$
$$\tilde{v}_{st} = \text{tr } Y_s' V Y_t. \tag{2.26}$$

Minimizing σ over $X \in \mathcal{Y}$ is now equivalent to minimizing $\tilde{\sigma}$ over $\theta \in \mathbb{R}^r$.

If $\mathcal{Y} = \mathbb{R}^{n\times p}$ then, in a sense, this is just notational sleight of hand. Consider, for example, using the basis where the Y_s are the np matrices $e_i e_q'$. Then

$$\{\tilde{A}_{ij}\}_{kq,lv} := \delta^{qv}\{A_{ij}\}_{kl} \tag{2.27}$$

Using Kronecker products this can be written as $\tilde{A}_{ij} = I_p \otimes A_{ij}$, the direct sum of p copies of A_{ij}. Obviously if $\theta = \text{vec}(Y)$ then $d_{ij}^2(Y) = \theta'\tilde{A}_{ij}\theta$. Also $\tilde{B}(\theta) = I_p \otimes B(Y)$ and $\tilde{V} = I_p \otimes V$. The only thing that changes by moving from configuration space to coefficient space, using the canonical basis of $\mathbb{R}^{n\times p}$, is that the configuration gets strung out to a vector, and the matrices A_{ij} get blown up to p copies of themselves.

But nevertheless it is clear that coefficient space allows us to use different bases as well, and allows us to use bases for proper subspaces of dimension $r < np$. This can be the $p(n-1)$-dimensional space of centered configurations, or the $np - \frac{1}{2}p(p+1)$- dimensional subspace of lower diagonal centered configurations. These configurations can be used to eliminate translational and rotational indeterminacy from basic MDS.

But the basis can also define a subspace of configurations with, for example, a rectangular lattice pattern, with the edges of the rectangle parallel to the horizontal and vertical axes (Borg and Leutner (1983)) or, for that matter, configurations X constrained to satisfy any number of (consistent) linear equality constraints. If $r < np - \frac{1}{2}p(p+1)$ then these applications are properly discussed as constrained multidimensional scaling or CMDS. A discussion of various forms of CMDS is in chapter 15.

2.1.4 Our Friends CS and AM/GM

Perhaps the most frequently used mathematical results in this book are two elementary inequalities: the Cauchy-Schwartz and the Aritmetic-Geometric

Mean inequalities. They are so important that we give them their own section in the book, their own acronyms CS and AM/GM, and we include their statements and even proofs.

Theorem 2.1. *If x and y are vectors in a Euclidean space X then $\langle x, y\rangle \leq \|x\|\|y\|$, with equality if and only if there is a real α such that $x = \alpha y$.*

Proof. If $x = 0$ and/or $y = 0$ then obviously the result is trivially true. If $x \not\emptyset$ and $y \neq 0$ then consider $f(\alpha) := \|x - \alpha y\|^2 = \|x\|^2 + \alpha^2\|y\|^2 - 2\alpha\langle x, y\rangle$. Now

$$\min_{\alpha} f(\alpha) = \|x\|^2 - \frac{\langle x, y\rangle^2}{\|y\|^2} \geq 0. \tag{2.28}$$

It follows that $\langle x, y\rangle^2 \leq \|x\|^2\|y\|^2$, which shows $-\|x\|\|y\|\langle x, y\rangle \leq \|x\|\|y\|$. We have $\min_\alpha f(\alpha) = 0$ if and only if $x = \alpha y$ for some α. □

Theorem 2.2. *If x and y are two non-negative numbers, then $\sqrt{xy} \leq \frac{1}{2}(x + y)$ with equality if and only if $x = y$.*

Proof. Follows directly from $(\sqrt{(}x) - \sqrt{(}y))^2 = x + y - 2\sqrt{xy} \geq 0$. □

This can also be written as

Corollary 2.1. *If x and y are two non-negative numbers, then $xy \leq \frac{1}{2}(x^2 + y^2)$ with equality if and only if $x = y$.*

Combining CS and AM/GM gives

Corollary 2.2. *If x and y are vectors in a Euclidean space X then $\langle x, y\rangle \leq \frac{1}{2}(\|x\|^2 + \|y\|^2)$.*

2.2 Global Properties

2.2.1 Boundedness

Theorem 2.3. *σ is bounded below by zero and unbounded above.*

Proof. Stress is a sum of squares, and thus it is non-negative, i.e. bounded below by zero. Because $\sigma(\alpha X) = 1 - \alpha\rho(X) + \frac{1}{2}\alpha^2\eta^2(X)$ we see that for each $X \neq 0$ and for each $K < +\infty$ there is an α such that $\sigma(\alpha X) > K$. □

2.2.2 Invariance

Theorem 2.4. *We have the following invariances.*

- *Rotational Invariance:* $\sigma(XK) = \sigma(X)$ *for all* K *with* $K'K = KK' = I$.
- *Translational Invariance:* $\sigma(X + eu') = \sigma(X)$ *for all* $u \in \mathbb{R}^p$.
- *Reflectional Invariance:* $\sigma(XK) = \sigma(X)$ *for all diagonal* K *with* $k_{ss} = \pm 1$.
- *Evenness:* $\sigma(-X) = \sigma(X)$.

Proof. Stress only depends on the distances between the points in the configuration, and thus it is invariant under rigid geometrical transformations (rotations, reflections, and translations). Note that reflectional and evenness are actually special cases of rotational invariance. □

It follows directly that the minimizer of stress, if it exists, cannot possibly be unique. Whatever the value at a minimum, it is shared by all rigid transformations of the configuration.

It also follows from translational invariance that we can minimize stress over the $p(n-1)$ dimensional subspace of $\mathbb{R}^{n\times p}$ of all $n \times p$ matrices which are centered, i.e. have $e'X = 0$. Rotational invariance implies we can also require without loss of generality that X is orthogonal, i.e. that $X'X$ is diagonal. This studied in more detail in section #ref(propconfspace).

2.2.3 Continuity

A real-valued function f on an open subset X of a Euclidean space is *Lipschitz* or *Lipschitz continuous* if there is a $K \geq 0$ such that $|f(x)-f(y)| \leq K\|x-y\|$ for all x and y in X. The smallest K for which this inequality holds is called the *Lipschitz constant* of f. Lipschitz functions are uniformly continuous, and thus continuous. Lipschitz functions are almost everywhere differentiable, and where the derivative exists there is an $L \geq 0$ such that $\|df(x)\| \leq L$. Thus differentiable functions with an unbounded derivative are not Lipschitz.

A function f is *locally Lipschitz* on X if for each $x \in X$ there is a open neighborhood $\mathcal{N}(x)$ such that f is Lipschitz on $\mathcal{N}(x)$. A locally Lipschitz

function is continuous and almost everywhere differentiable. Continuously differentiable functions and convex functions are all locally Lipschitz.

Theorem 2.5. *On* $\mathbb{R}^{n\times p}$

- d_{ij} *is Lipschitz continuous with Lipschitz constant* $\sqrt{2}$.
- d_{ij}^2 *is locally Lipschitz, but not globally Lipschitz.*

Proof. To show that d_{ij} is Lipschitz we use the reverse triangle inequality $|\|x\| - \|y\|| \leq \|x - y\|$. It gives

$$|d_{ij}(X)-d_{ij}(Y)| = |\|X'(e_i-e_j)\|-\|Y'(e_i-e_j)\|| \leq \|(X-Y)'(e_i-e_j)\| \leq \sqrt{2}\,\|X-Y\|. \tag{2.29}$$

To show this Lipschitz bound is sharp use $Y = 0$ and $X = \begin{bmatrix} x \\ -x \end{bmatrix}$ with $\|x\| = 1$. Then $|d(X) - d(Y)| = 2$ and $\|X - Y\| = \sqrt{2}$.

Because d_{ij}^2 is continuously differentiable it is locally Lipschitz, and because its derivative is unbounded it is not globally Lipschitz. □

Corollary 2.3. *On* $\mathbb{R}^{n\times p}$

- ρ *is Lipschitz continuous with Lipschitz constant* $\sqrt{2}\ \sum\sum_{1\leq i<j\leq n} w_{ij}\delta_{ij}$.
- η^2 *and* σ *are both locally Lipschitz, but not globally Lipschitz.*

Proof. This follows directly from theorem 2.5. □

2.2.4 Coercivity

Stress is not a quadratic function, and not even a convex function, of the configuration. But it is like a bowl shaped around the origin, with some bumps and creases, in a way we are going to make more precise. First a definition: A real-valued function f is *coercive* if for every sequence $\{x_k\}$ with $\lim_{k\to\infty} \|x_k\| = \infty$ we also have $\lim_{k\to\infty} f(x_k) = +\infty$.

Theorem 2.6. σ *is coercive.*

Proof. From (2.8) we have $\sigma(X) \geq \frac{1}{2}(1-\eta(X))^2$. Now η is clearly coercive, and thus σ is coercive. □

It follows from coercivity that all level sets of stress $\mathcal{L}_s := \{X \mid \sigma(X) = s\}$ are compact, and that there is at least one configuration for which the global minimum of stress is attained (Ortega and Rheinboldt (1970), section 4.3).

The following theorem provides even more bowl-shapedness.

Theorem 2.7. *If $X \neq 0$ then on the ray $\{Y \mid Y = \alpha X$ with $\alpha \geq 0\}$ stress is an unbounded convex quadratic in α. The minimum of this quadratic is at $\alpha = \rho(X)/\eta^2(X)$ and it is equal to $1 - \frac{1}{2}\rho^2(X)/\eta^2(X)$. There is a local maximum at the boundary $\alpha = 0$, equal to 1.*

Proof. We have $\sigma(\alpha X) = 1 - \alpha\rho(X) + \frac{1}{2}\alpha^2\eta^2(X)$. The statements in the theorem follow easily from this. □

2.3 Differentiability

The fact that d_{ij} can be zero for some configurations creates problems with the differentiability of stress. These problems have been largely ignored in the MDS literature, and there are indeed reasons why they are not of great **practical** importance (see section 2.5.2 of this chapter), at least not in basic MDS. But for reasons of completeness, and for later generalizations of basic MDS, we discuss zero distances and the resulting problems with differentiability in some detail.

Historically the complications caused by $d_{ij}(X) = 0$ were one of the reasons why I switched from differentiability to convexity in De Leeuw (1977a) and from derivatives to directional derivatives in De Leeuw (1984c). It turned out that at least some of the important characteristics of the smacof algorithm, and several important aspects of stress surfaces, were better described by inequalities than by equations.

Directional Derivatives

Because we are dealing with minimization of stress, which is not everywhere differentiable, we use one-sided directional derivatives. Our notation largely follows Delfour (2012).

The first three directional derivatives at X in the direction Y are defined recursively by

$$d_+\sigma(X;Y) := \lim_{\epsilon\downarrow 0} \frac{\sigma(X+\epsilon Y)-\sigma(X)}{\epsilon}, \tag{2.30}$$

$$d_+^{(2)}\sigma(X;Y,Y) := \lim_{\epsilon\downarrow 0} \frac{\sigma(X+\epsilon\ Y)-\sigma(X)-\epsilon\ d\sigma(X)(Y)}{\frac{1}{2}\epsilon^2}, \tag{2.31}$$

$$d_+^{(3)}\sigma(X;Y,Y,Y) := \lim_{\epsilon\downarrow 0} \frac{\sigma(X+\epsilon\ Y)-\sigma(X)-\epsilon\ d_+\sigma(X)(Y)-\frac{1}{2}\epsilon^2\ d_+^{(2)}\sigma(X)(Y,Y)}{\frac{1}{6}\epsilon^3}, \tag{2.32}$$

where $\epsilon \downarrow 0$ is understood as ϵ taking only strictly positive values in computing the limit, and where it is also understood that the one-sided limits exist. The directional derivatives used in optimization theory differ from the usual derivatives of analysis because the limits in functions that define them are over the one-dimensional positive real axis and are one-sided (from the right). You may wonder why we need to go as high as order three, but just you wait.

Note that we write $d_+\sigma(X;Y)$ (with a semi-colon) instead of $d_+\sigma(X,Y)$ (with a comma) to emphasize the different roles of X and Y. Also note that by "directional derivatives" we will always mean "one-sided directional derivatives", because the two-sided ones are of limited usefulness in optimization. This is especially true for the two-sided directional derivative defined by

$$d\sigma(X;Y) := \lim_{\epsilon\to 0} \frac{\sigma(X+\epsilon Y)-\sigma(X)}{\epsilon}. \tag{2.33}$$

The two-sided derivative may not exist, while we can still make useful statements of the minima of non-differentiable functions using the one-sided version. Of course for totally differentiable functions in the classical sense the two directional derivatives are equal.

For the higher directional derivatives we can also use alternative, and slightly more general, definitions that follow directly from the idea that the k^{th} directional derivative is the directional derivative of the $(k-1)^{th}$ one. In each step of the recursion we now use a different direction, instead of using the

fixed direction Y in all steps. Thus

$$d_+^{(2)}\sigma(X;Y,Z) := \lim_{\epsilon\downarrow 0} \frac{d_+\sigma(X+\epsilon Z;Y) - d_+\sigma(X;Y)}{\epsilon}, \tag{2.34}$$

$$d_+^{(3)}\sigma(X;Y,Z,U) := \lim_{\epsilon\downarrow 0} \frac{d_+^2\sigma(X+\epsilon U;Y,Z) - d_+^2\sigma(X;Y,Z)}{\epsilon}. \tag{2.35}$$

Note again that $d_+\sigma(X)$ is a function on $\mathbb{R}^{n\times p}$, $d_+^2\sigma(X)$ a is a function on $\mathbb{R}^{n\times p}\otimes\mathbb{R}^{n\times p}$, and $d_+^3\sigma(X)$ is a function on $\mathbb{R}^{n\times p}\otimes\mathbb{R}^{n\times p}\otimes\mathbb{R}^{n\times p}$.

If σ is differentiable at X then $d_+\sigma(X), d_+^{(2)}\sigma(X)$, and $d_+^{(3)}\sigma(X)$ are the usual first, second, and third derivatives of σ at X. In this differentiable case they are, respectively, a linear function, a symmetric bilinear function, and a super-symmetric trilinear function.

We can use the directional derivatives to expand $\sigma(X+\epsilon Y)$ in powers of ϵ. This gives an expansion of the form

$$\begin{aligned}\sigma(X+\epsilon Y) = \sigma(X) + \epsilon d_+\sigma(X)(Y) + \frac{1}{2}\epsilon^2 d_+^{(2)}\sigma(X)(Y,Y) + \\ + \frac{1}{6}\epsilon^2 d_+(3)\sigma(X)(Y,Y,Y) + \mathrm{o}(\epsilon^3),\end{aligned} \tag{2.36}$$

where $\mathrm{o}(\epsilon^3)$ stand for any function of $\epsilon > 0$ such that

$$\lim_{\substack{\epsilon\downarrow 0\\ \epsilon\neq 0}} \frac{\mathrm{o}(\epsilon^3)}{\epsilon^3} \to 0. \tag{2.37}$$

2.3.0.1 Distances

Let us look at the directional differentiability of the distances d_{ij} themselves first. Since ρ is a straightfoward weighted sum of distances and η^2 is a weighted sum of squared distances, the only directional derivatives we really need are those of the squared distances and the distances.

The problems with differentiability are clearly not caused by the squared distances, which form the η^2 component in equation (2.6). The squared distance $d_{ij}^2(X) = \mathrm{tr}\ X'A_{ij}X$ is a quadratic function, and thus it is everywhere infinitely many times continuously differentiable.

On the other hand, $d_{ij}(X) = \sqrt{\text{tr } X'A_{ij}X}$ is not differentiable at points where $d_{ij}(X) = 0$, i.e. where $x_i = x_j$, because the square root is not differentiable at zero. For MDS this means that if $d_{ij}(X) = 0$ for one or more (i, j) with $w_{ij}\delta_{ij} > 0$ then both ρ and σ are not differentiable at X.

For the avalanche of furmulas that will follow in this section it is convenient to define $c_{ij}(X, Y) := \text{tr } Y'A_{ij}X = (y_i - y_j)'(x_i - x_j)$. Note that $c_{ij}(X, X) = d_{ij}^2(X)$ and $c_{ij}(Y, Y) = d_{ij}^2(Y)$. Now

$$d_+ d_{ij}^2(X; Y) = 2c_{ij}(X, Y), \tag{2.38}$$

$$d_+^2 d_{ij}^2(X; Y, Z) = 2cij(Y, Z), \tag{2.39}$$

$$d_+^3 d_{ij}^2(X; Y, Z, U) = 0. \tag{2.40}$$

More involved calculations are needed for the directional derivatives of d_{ij}. First the "problematic" case $d_{ij}(X) = 0$. We have

$$d_+ d_{ij}(X; Y) = d_{ij}(Y), \tag{2.41}$$

$$d_+^2 d_{ij}(X; Y) = 0, \tag{2.42}$$

$$d_+^3 d_{ij}(X; Y) = 0. \tag{2.43}$$

$$\tag{2.44}$$

Note that $d_+ d_{ij}(X)$ is continuous but not linear in Y, which also implies d_{ij} is not differentiable at X. Also note that if we define the two-sided directional derivative of d_{ij} at X in the direction Y

$$dd_{ij}(X; Y) := \lim_{\epsilon \to 0} \frac{d_{ij}(X + \epsilon Y) - d_{ij}(X)}{\epsilon}, \tag{2.45}$$

then this limit only exists if also $d_{ij}(Y) = 0$. The limit from the right is $d_{ij}(Y)$, but the limit from the left is $-d_{ij}(Y)$. This illustrates that the two-sided directional derivative does not give much useful information on the behavior of the distance at zero.

If $d_{ij}(X) > 0$ we have continuous differentiability of all orders at X. To expand

$$d_{ij}(X + \epsilon Y) = \sqrt{d_{ij}^2(X) + 2\ \epsilon\ c_{ij}(X, Y) + \epsilon^2\ d_{ij}^2(Y)} \tag{2.46}$$

we use the truncated Maclaurin series for the square root

$$\sqrt{1 + x} = 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \frac{1}{16}x^3 + \text{o}(x^3). \tag{2.47}$$

For the distance this gives the series

$$\begin{aligned} d_{ij}(X+\epsilon Y) = d_{ij}(X) &+ \epsilon \frac{1}{d_{ij}(X)} c_{ij}(X,Y) + \\ &+ \frac{1}{2}\epsilon^2 \frac{1}{d_{ij}(X)} \left\{ d_{ij}^2(Y) - \frac{c_{ij}^2(X,Y)}{d_{ij}^2(X)} \right\} + \\ &- \frac{1}{2}\epsilon^3 \frac{c_{ij}(X,Y)}{d_{ij}^3(X)} \left\{ d_{ij}^2(Y) - \frac{c_{ij}^2(X,Y)}{d_{ij}^2(X)} \right\} + \mathrm{o}(\epsilon^3), \end{aligned} \tag{2.48}$$

and consequently

$$d_+ d_{ij}(X;Y) = \frac{1}{d_{ij}(X)} c_{ij}(X,Y), \tag{2.49}$$

$$d_+^{(2)} d_{ij}(X;Y,Y) = \frac{1}{d_{ij}(X)} \left\{ d_{ij}^2(Y) - \frac{c_{ij}^2(X,Y)}{d_{ij}^2(X)} \right\}, \tag{2.50}$$

$$d_+^{(3)} d_{ij}(X;Y,Y,Y) = -3 \frac{c_{ij}(X,Y)}{d_{ij}^3(X)} \left\{ d_{ij}^2(Y) - \frac{c_{ij}^2(X,Y)}{d_{ij}^2(X)} \right\}. \tag{2.51}$$

Formulas for the mixed directional derivatives from equations (2.34) and (2.35) are necessarily more complicated. Again assuming $d_{ij}(X) > 0$ we find

$$d_+^{(2)} d_{ij}(X;Y,Z) = \frac{1}{d_{ij}(X)} \left\{ c_{ij}(Y,Z) - \frac{c_{ij}(X,Y) c_{ij}(X,Z)}{d_{ij}^2(X)} \right\}, \tag{2.52}$$

which reduces to (2.50) if $Y = Z$. And, with 95% certainty,

$$\begin{aligned} d_+^{(3)} d_{ij}(X;Y,Z,U) = 3 &\frac{c_{ij}(X,Y) c_{ij}(X,Z) c_{ij}(X,U)}{d_{ij}^5(X)} + \\ &- \frac{c_{ij}(X,Y) c_{ij}(U,Y) + c_{ij}(X,Z) c_{ij}(U,Z) + c_{ij}(X,U) c_{ij}(Y,Z)}{d_{ij}^3(X)} \end{aligned} \tag{2.53}$$

which is obviously symmetric in Y, Z, and U, and if $Y = Z = U$ it reduces to (2.51).

Again, we have to be careful if $d_{ij}(X) = 0$. In that case we know from equation (2.41) that $d_+ d_{ij}(X;Y) = d_{ij}(Y)$. Also

$$d_+ d_{ij}(X + \epsilon Z; Y) = \begin{cases} d_{ij}(Y) & \text{if } d_{ij}(Z) = 0, \\ \frac{1}{d_{ij}(Z)} c_{ij}(Y,Z) & \text{if } d_{ij}(Z) > 0, \end{cases} \tag{2.54}$$

and thus $d_+^{(2)}d_{ij}(X;Y,Z) = 0$ when $d_{ij}(Z) = 0$, but when $d_{ij}(Z) > 0$ the limit defining $d_+^{(2)}d_{ij}(X;Y,Z)$ does not exist unless $Z = Y$. If $Z = Y$ we have $d_+^{(2)}d_{ij}(X;Y,Y) = 0$, in accordance with (2.42).

2.3.0.2 Rho and Stress

We now use the results from the previous section to compute directional derivatives of ρ and σ. They are general, in the sense that they cover cases in which some $d_{ij}(X) > 0$ and some $d_{ij}(X) = 0$. To handle one or more zero distances we define

$$\xi(X;Y) := \sum_{1\leq i<j\leq n}\sum \{w_{ij}\delta_{ij}d_{ij}(Y) \mid w_{ij}\delta_{ij} > 0 \text{ and } d_{ij}(X) = 0\}. \tag{2.55}$$

The directional derivatives of ρ at X in direction Y are

$$d_+\rho(X;Y) = \text{tr } Y'B(X)X + \xi(X,Y), \tag{2.56}$$

$$d_+^{(2)}\rho(X;Y,Y) = \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}(X)}\left\{d_{ij}^2(Y) - \frac{c_{ij}^2(X,Y)}{d_{ij}^2(X)}\right\}, \tag{2.57}$$

$$d_+^{(3)}\rho(X;Y,Y,Y) = -3\sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}c_{ij}(X,Y)}{d_{ij}^3(X)}\left\{d_{ij}^2(Y) - \frac{c_{ij}^2(X,Y)}{d_{ij}^2(X)}\right\}. \tag{2.58}$$

Note that by the CS inequality

$$d_{ij}^2(Y) - \frac{c_{ij}^2(X,Y)}{d_{ij}^2(X)} \geq 0 \tag{2.59}$$

for all Y, and thus $d_+^{(2)}\rho(X)(Y,Y) \geq 0$.

The directional derivatives for stress at X in direction Y are

$$d_+\sigma(X;Y) = \text{tr } Y'(V - B(X))X - \xi(X,Y), \tag{2.60}$$

$$d_+^{(2)}\sigma(X;Y,Y) = \text{tr } Y'VY - \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}(X)}\left\{d_{ij}^2(Y) - \frac{c_{ij}^2(X,Y)}{d_{ij}^2(X)}\right\}, \tag{2.61}$$

$$d_+^{(3)}\sigma(X;Y,Y,Y) = 3\sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}c_{ij}(X,Y)}{d_{ij}^3(X)}\left\{d_{ij}^2(Y) - \frac{c_{ij}^2(X,Y)}{d_{ij}^2(X)}\right\}. \tag{2.62}$$

Note that we can also write (2.61) as

$$d_+^{(2)}\sigma(X;Y,Y) = \text{tr } Y'(V - B(X))Y + \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}(X)}\frac{c_{ij}^2(X,Y)}{d_{ij}^2(X)}. \quad (2.63)$$

Combining (2.61) and (2.63) shows

$$\text{tr } Y'(V - B(X))Y \lesssim d_+^{(2)}\sigma(X;Y,Y) \lesssim \text{tr } Y'VY. \quad (2.64)$$

A convenient upper bound for $d_+^{(3)}\sigma(X;Y,Y,Y)$ is also useful. From (2.62) and the CS inequality

$$d_+^{(3)}\sigma(X;Y,Y,Y) \leq 3\sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}|c_{ij}(X,Y)|}{d_{ij}^3(X)}d_{ij}^2(Y) \leq 3\sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}d_{ij}^3(Y)}{d_{ij}^2(X)}. \quad (2.65)$$

Once more, in the differentiable case the subscript + in d_+ is not necessarily, but if one or more $d_{ij}(X) = 0$ for which $w_{ij}d_{ij} > 0$ then the subscript is needed.

2.3.0.3 expandStress

The R function {expandStress, with arguments w, delta, x, and y, gives the zeroeth, first, second, or third terms in the expansion (2.36), using the formulas for directional derivatives in this section. We use an example with $n = 4$ and $p = 2$. All weights and dissimilarities are equal. Configuration X are the four corners of a square, perturbation Y is a 4×2 matrix of random standard normals.

The function expandTester takes an interval around zero for ϵ and computes the value of $\sigma(X + \epsilon Y)$ at 1000 points in that interval. It also computes the zero, first, second, or third order Maclaurin approximations to $\sigma(X + \epsilon Y)$. In this first example X is a local minimum and thus $d_+\sigma(X;Y) = 0$. The zero-order and first-order approximation are thus the same.

If the interval for ϵ is $[-1, 1]$ the sum of squares of the differences between $\sigma(X + \epsilon Y)$ and its approximations of different orders are

```
##      [,1]
## [1,] +917.455222294743749
## [2,] +917.455222294744658
## [3,] +468.774392806713365
## [4,] +185.377911517761987
```

If the interval is $[-.1, .1]$ the errors of approximation are

```
##      [,1]
## [1,] +0.133635212939676
## [2,] +0.133635212939676
## [3,] +0.000152399828616
## [4,] +0.000001520085462
```

And if the interval is $[-.01, .01]$ the errors of approximation are

```
##      [,1]
## [1,] +0.000013434031631
## [2,] +0.000013434031631
## [3,] +0.000000000149172
## [4,] +0.000000000000015
```

On the smaller intervals the error of second-order approximation is already very small, which is not surprising because twice-differentiable functions are pretty much convex quadratics close to a local minimum.

2.3.1 Partial Derivatives

In the differentiable case we now introduce *gradients* and *Hessians*. The gradient at X is a vector with first-order partial derivatives at X, the Hessian is a matrix with second-order partial derivatives at X. Partial derivatives are the commonly used tool for actual computation of derivatives.

We can obtain the partial derivatives from the directional derivatives in equations (2.60), (2.61), and (2.62) by using the base vectors $E_{is} = e_i e_s'$ to

expand the perturbations Y, Z, and U. So the gradient of stress at X is

$$\{\nabla\sigma(X)\}_{is} := d_+\sigma(X; E_{is}) = \text{tr } e_i e_s'(V - B(X))X = \{(V - B(X))X\}_{is}. \tag{2.66}$$

The gradient $\nabla\sigma(X)$ is an $n \times p$ matrix, and

$$d_+\sigma(X; Y) = \text{tr } Y'\nabla\sigma(X) = \text{tr } Y'(V - B(X))X. \tag{2.67}$$

For the Hessian we use similar calculations, heavily relying on the Kronecker delta, and on cyclic permutations under the trace sign. First

$$c_{ij}(X, E_{ks}) = \text{tr } X'(e_i - e_j)(e_i - e_j)'e_k s_s' = (x_{is} - x_{js})(\delta^{ik} - \delta^{jk}), \tag{2.68}$$

and

$$c_{ij}(E_{ks}, E_{lt}) = \text{tr } e_t e_l'(e_i - e_j)(e_i - e_j)'e_k e_s' = \delta^{st}(\delta^{ik} - \delta^{jk})(\delta^{il} - \delta^{jl}). \tag{2.69}$$

Thus, from equation (2.39),

$$\{\nabla^{(2)} d_{ij}^2(X)\}_{ks,lt} = d_+^{(2)} d_{ij}^2(X; E_{ks}, E_{lt}) = 2\delta^{st}(\delta^{ik} - \delta^{jk})(\delta^{il} - \delta^{jl}), \tag{2.70}$$

and from equation (2.52)

$$\begin{aligned}\{\nabla^{(2)} d_{ij}(X)\}_{ks,lt} &= d_+^{(2)} d_{ij}(X; E_{ks}, E_{lt}) = \\ &= \frac{(\delta^{il} - \delta^{jl})(\delta^{ik} - \delta^{jk})}{d_{ij}(X)} \left\{\delta^{st} - \frac{(x_{is} - x_{js})(x_{it} - x_{jt})}{d_{ij}^2(X)}\right\}.\end{aligned} \tag{2.71}$$

Now take the usual weighted sums of equations (2.70) and (2.71) to find the Hessians of ρ and σ. To get relatively compact expressions we define the symmetric doubly-centered matrices

$$H_{st}(X) := \sum_{1 \le i < j \le n} \sum w_{ij} \frac{\delta_{ij}}{d_{ij}^3(X)}(x_{is} - x_{js})(x_{it} - x_{jt})A_{ij}. \tag{2.72}$$

Then the Hessian of ρ is

$$\{\nabla^{(2)}\rho(X)\}_{ks,lt} = \{\delta^{st}B(X) - H_{st}(X)\}_{kl}, \tag{2.73}$$

and that of σ is

$$\{\nabla^{(2)}\sigma(X)\}_{ks,lt} = \{\delta^{st}(V - B(X)) + H_{st}(X)\}_{kl}. \tag{2.74}$$

This is all somewhat inconvenient because of the double indexing of rows and columns. $\nabla^{(2)}\sigma(X)$ is an element of $\mathbb{R}^{n\times p} \otimes \mathbb{R}^{n\times p}$ and we can represent it numerically either as a matrix or a four-dimensional array.

To cut the cord we use the vec isomorphism from $\mathbb{R}^{n\times p}$ to $\mathbb{R}^{np}$, and its inverse vec^{-1}.

For computational purposes you can think of $\nabla^2\sigma(X)$ as an $np \times np$ matrix $K(X)$, consisting of blocks of symmetric matrices of order n, indexed by points, with p row-blocks and p column-blocks, indexed by dimensions. For $s \neq t$ block (s,t) is the matrix $H_{st}(X)$, the diagonal blocks for $s = t$ are $(V - B(X)) + H_{ss}(X)$. In the same way we can collect the blocks $H_{st}(X)$ in the $np \times np$ matrix $H(X)$.

$$\{\nabla^{(2)}\sigma(X;Y)\}_{ks} := \sum_{l=1}^{n}\sum_{t=1}^{p}\{\nabla^{(2)}\sigma(X)\}_{ks,lt}y_{lt},, \tag{2.75}$$

and

$$\text{tr } Y'\nabla^2\sigma(X)Z := d_+^2\sigma(X;Y,Z) = \sum_{k=1}^{n}\sum_{s=1}^{p}\sum_{l=1}^{n}\sum_{t=1}^{p}\{\nabla^2\sigma(X)\}_{ks,lt}y_{ks}z_{lt}. \tag{2.76}$$

Thus

$$\underline{\nabla}^{(2)}\rho(X) = I_p \otimes B(X) - \underline{H}(X), \tag{2.77}$$

$$\underline{\nabla}^{(2)}\sigma(X) = I_p \otimes (V - B(X)) + \underline{H}(X). \tag{2.78}$$

I do not like to use vec and friends in formulas and derivations, so I try to avoid them. It is a different matter in computation, because in a computer Y is the same as $\text{vec}(Y)$ anyway.

Because the Hessian is important throughout the book we want to make sure we have the correct formulas and code. One way to check this is to compare

it to the numerical approximation of the Hessian from the package numDeriv (Gilbert and Varadhan (2019)). Normally one checks if the code is correct by comparing it with the mathematics, but here we proceed the other way around.

Again we use the four corners of the square as an example, with weights and dissimilarities all equal. The largest absolute difference between the elements of the numerical and the analytical Hessian for ths example is $8.2399365 \times 10^{-14}$, which means that we basically have double-precision equality between the two. We repeat this for a random X, because at a local minimum the approximation may be more precise. For the random configuration we find a maximum deviation of 0.2485061, a bit bigger, but still small.

Here are some useful properties of the Hessians.

Theorem 2.8. *1.* $0 \lesssim H(X) \lesssim I_p \otimes B(X)$.
2. $I_p \otimes (V - B(X)) \lesssim K(X) \lesssim I_p \otimes V$.

Proof. Let $y = \text{vec}(Y)$. Then

$$y'H(X)y = \sum_{s=1}^p \sum_{t=1}^p y_s' H_{st}(X) y_t = \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}^3(X)}c_{ij}^2(X,Y) \geq 0,$$

and thus $H(X) \gtrsim 0$. From () $\nabla^2\rho(X) \gtrsim 0$, and thus $I_p \otimes B(X) \gtrsim H(X)$. This proves the first part. The second part is immediate from the first part and (). □

Another useful property. Let $y = \text{vec}(Y)$.

Theorem 2.9. *ozo*

Proof.

$$H(X)Y = \sum_{t=1}^p H_{st}(X)y_t = \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}c_{ij}(X,Y)}{d_{ij}^3(X)}(x_{is}-x_{js})(e_i-e_j). \quad (2.79)$$

If $Y = X$ then $H(X)X = B(X)X$ and thus $\nabla^2\rho(X)X = 0$ and $\nabla^2\sigma(X)X = VX$. If $Y = XT$ with T anti-symmetric then $c_{ij}(X,Y) = \text{tr } X'A_{ij}XT = 0$ and thus $H(X)Y = 0$. This implies $\nabla^2\rho(X)Y = B(X)XT$ and $\nabla^2\sigma(X)Y = (V-B(X))XT$. If $B(X)X = VX$ then $\nabla^2\rho(X)X = VX$ and $\nabla^2\sigma(X)X = 0$. If $Y = e\alpha'$ then $\nabla^2\rho(X)Y = \nabla^2\sigma(X)Y = 0$. □

In the example with the square the eigenvalues of $\underline{H}(X)$ are

```
## [1] +0.33333333 +0.19526215 +0.19526215 +0.19526215 +0.13807119 +0.00000000
## [7] +0.00000000 +0.00000000
```

```
## [1] +0.52960443 +0.44261066 +0.27431040 +0.15854524 +0.07386436 +0.00000000
## [7] +0.00000000 -0.00000000
```

while those of $\underline{\nabla}^2\sigma(X)$ are

```
## [1] +0.33333333 +0.19526215 +0.13807119 +0.13807119 +0.13807119 +0.00000000
## [7] +0.00000000 +0.00000000
```

```
## [1] +0.33333333 +0.25946897 +0.17478810 +0.05902293 +0.00000000 +0.00000000
## [7] -0.10927733 -0.19627110
```

We can derive nicer expressions for the higher derivatives in coefficient space (see section 2.1.3). This was done, perhaps for the first time, in De Leeuw (1993), which was actually written around 1985. In Kearsley, Tapia, and Trosset (1995) Newton's method was used to minimize stress, so presumably they implemented some formula for the Hessian. In De Leeuw (1988) the expression involving $H(X)$ from (2.72) was first given in configuration space. We could use similar computations to obtain the third-order partial derivatives, but for now we have no need for them in this book.

2.3.2 Special Expansions

$c_{ij}(X,Y) = 0$ for all $i < j$

2.3.2.1 Infinitesimal Rotations

Suppose $Y = XT$, with T antisymmetric, so that $X + \epsilon Y = X(I + \epsilon T)$. Then $c_{ij}(X,Y) = 0$ for all $i < j$, and thus from equations (2.60), (2.61), and (2.62)

$$d_+\sigma(X;Y) = 0, \tag{2.80}$$

$$d_+^{(2)}\sigma(X;Y,Y) = \text{tr } Y'(V - B(X))Y, \tag{2.81}$$

$$d_+^{(3)}\sigma(X;Y,Y,Y) = 0. \tag{2.82}$$

2.3.2.2 Singularities

Suppose $X = [\underline{X} \mid 0]$ and $Y = [0 \mid \underline{Y}]$ so that $X + \epsilon Y = [\underline{X} \mid \epsilon \underline{Y}]$. Here X and Y are $n \times p$, $underline X$ is $n \times r$, with $r < p$, and $\underline{Y}$ is $n \times (p - r)$. Then again $c_{ij}(X, Y) = 0$ for all $i < j$, and thus ()()() again.

$$\sigma(\underline{X} + \epsilon \underline{Y}) = \sigma(X) - \epsilon \sum_{d_{ij}(X)=0} w_{ij}\delta_{ij}d_{ij}(Y) + \frac{1}{2}\epsilon^2 \text{tr } Y'(V - B(X))Y + o(\epsilon^2) \tag{2.83}$$

2.3.2.3 Singularities

Suppose $\underline{X} = [X \mid 0]$ and $\underline{Y} = [Z \mid Y]$ so that $\underline{X} + \epsilon \underline{Y} = [X + \epsilon Z \mid \epsilon Y]$. Here $\underline{X}$ and $\underline{Y}$ are $n \times p$, X is $n \times r$, with $r < p$, and Y is $n \times (p - r)$. Then

$$\sigma(\underline{X} + \epsilon \underline{Y}) = \sigma(X) - \epsilon \sum_{d_{ij}(X)=0} w_{ij}\delta_{ij}d_{ij}(Y) + \frac{1}{2}\epsilon^2 \text{tr } Y'(V - B(X))Y + o(\epsilon^2) \tag{2.84}$$

2.4 Convexity

Remember that a function f on an open subset X of a Euclidean space is *convex* if for all x and y in X and $0 \leq \alpha \leq 1$ we have $f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha) f(y)$. Thus on the line segment connecting x and y the function f is never above the line segment connecting $f(x)$ and $f(y)$. Convex functions are a.e. differentiable, in fact a.e. twice-differentiable. If the derivative exists at x then for all y we have $f(y) \geq f(x) + df(x)(y - x)$, which says the function majorizes its tangent plane at x. If the second derivative exists at x then $d^2 f(x; y, y) \geq 0$ for all y, which says that the Hessian at x is positive semidefinite.

Stress is definitely not a convex function of the configuration. If it actually was convex, or even convex and differentiable, then this book would be much shorter. Nevertheless convexity still play an important part in our development of MDS, ever since De Leeuw (1977a).

2.4.1 Distances

The convexity in the MDS problem comes from the convexity of the distance and the squared distance. Although these are elementary facts, they are important in our context, so we give a proof.

Theorem 2.10. *On $\mathbb{R}^{n\times p}$ both dij and d_{ij}^2 are convex.*

Proof. First, for $0 \leq \lambda \leq 1$,

$$d_{ij}^2(\lambda X + (1-\lambda)Y) = \lambda^2 d_{ij}^2(X) + (1-\lambda)^2 d_{ij}^2(Y) + 2\lambda(1-\lambda)(x_i - x_j)'(y_i - y_j) \tag{2.85}$$

By corollary 2.2,

$$(x_i - x_j)'(y_i - y_j) \leq \sqrt{d_{ij}^2(X) d_{ij}^2(Y)} \leq \frac{1}{2}(d_{ij}^2(X) + d_{ij}^2(Y)). \tag{2.86}$$

Combining (2.85) and (2.86) proves convexity of the squared distance.

Now use equation (2.85) and the CS inequality in the form $(x_i - x_j)'(y_i - y_j) \leq d_{ij}(X) d_{ij}(Y)$. This gives

$$d_{ij}^2(\lambda X + (1-\lambda)Y) \leq (\lambda d_{ij}(X) + (1-\lambda) d_{ij}(Y))^2. \tag{2.87}$$

Taking square roots on both sides of equation (2.87) proves convexity of the distance. □

Both ρ and η are norms on $\mathbb{R}^{n\times p}$.

Proof. homogeneous convex functions vanishing if and only if $X = 0$, which means they are both norms on $\mathbb{R}^{n\times p}$. □

2.4.2 Subdifferentials

Suppose f is a real-valued finite convex function on the finite-dimensional inner-product space $\mathcal{E}$. A *subgradient* of f at $x \in \mathcal{E}$ is a vector $z \in \mathcal{E}$ such that $f(y) \geq f(x) + \langle z, y - x\rangle$ for all $y \in \mathcal{E}$. The set of all subgradients at x is the *subdifferential* at x, written as $\partial f(x)$. In general, the subdifferential is a non-empty, closed, and convex set (rock). If f is differentiable at x then the subdifferential is the singleton which has the gradient $\nabla f(x)$ as its sole element (rock).

Apply this to d_{ij} on $\mathbb{R}^{n\times p}$.

Theorem 2.11. *The subdifferential of d_{ij} at X is*

- *If $d_{ij}(X) > 0$ then $\partial d_{ij}(X) = \left\{(e_i - e_j)\frac{(x_i - x_j)'}{d_{ij}(X)}\right\}$*
- *If $d_{ij}(X) = 0$ then $\partial d_{ij}(X) = \{Y \mid Y = (e_i - e_j)z' \text{ with } \|z\| \leq 1\}$*

$$\partial d_{ij}(X) = \bigcup_{\|z\| \leq 1} \{(e_i - e_j)z'\}$$

Proof. If $d_{ij}(X) = 0$ we must find the set of all $Z \in \mathbb{R}^{n \times p}$ such that

$$d_{ij}(Y) \geq \text{tr } Z'(Y - X) \tag{2.88}$$

for all $Y \in \mathbb{R}^{n \times p}$.

First of all (2.88) must be true for all $Y = \alpha X$ with $\alpha \geq 0$. Thus $(\alpha - 1)\text{tr } Z'X \leq 0$ for all $\alpha \geq 0$, which implies tr $Z'X = 0$. We can use this to simplify (2.88) to $d_{ij}(Y) \geq \text{tr } Z'Y$ for all Y. Next, it follows that $z_k = 0$ for $k \neq i, j$. If $z_k \neq 0$ choose $Y = e_k z_k'$. Then $d_{ij}(Y) = 0$ and $\text{tr} Z'Y = z_k' z_k > 0$. Now (2.88) simplifies to $d_{ij}(Y) \geq z_i' y_i + z_j' y_j$ If $y_i = z_i$ and $y_j = 0$ then we must have $\|z_i\| \geq \|z_i\|^2$ or $\|z_i\| \leq 1$. In the same way $\|z_j\| \leq 1$. Choose $y_i = y_j = y$ and some $y \neq 0$. Then we must have $(z_i + z_j)'y \leq 0$ for all y and thus $z_i = -z_j$. This proves the second part. □

By the sum rule for convex subdifferentials (rock)

$$\partial\rho(X) = B(X)X + \{Y \mid Y = \sum_{1 \leq i < j \leq n} \sum \{w_{ij}\delta_{ij}(e_i - e_j)z_{ij}' \mid d_{ij}(X) = 0\}\}, \tag{2.89}$$

where the z_{ij} are arbitrary vectors satisfying $\|z_{ij}\| \leq 1\|$.

Of course $\partial d_{ij}^2(X) = \{2A_{ij}X\}$ and thus $\partial\eta^2(X) = \{2VX\}$.

But σ is not convex, and we do not have a definition yet for the subdifferential of non-convex functions. We use the generalization introduced by Clarke. Suppose f is locally Lipschitz, and thus differentiable almost everywhere. Let $x^{(k)}$ be a sequence of points converging to x_∞, with
f differentiable at all $x^{(k)}$.

Then y is in the Clarke subdifferential $\partial_C f(x)$ if and only if $y = \lim_{k \to \infty} \nabla f(x^{(k)})$.

Clarke $\partial_C \sigma(X) = \{VX\} - \partial\rho(X)$

Combining this with #ref(eq:propsubdiffrho) gives

$$\partial\sigma(X) = (V - B(X))X - \{Y \mid Y = \sum_{1\leq i<j\leq n}\sum \{w_{ij}\delta_{ij}(e_i - e_j)z'_{ij} \mid d_{ij}(X) = 0\}\}, \tag{2.90}$$

2.4.3 DC Functions

In basic MDS

1. ρ is a non-negative convex function, homogeneous of degree one.
2. η^2 is a non-negative convex quadratic form, homogeneous of degree two.
3. σ is a non-negative difference of two convex functions.

This follows because η^2 is a weighted sum of squared distances and ρ is a weighted sum of distances, both with non-negative coefficients, and thus they are both convex.

Real-valued functions that are differences of two convex functions are also known as a *DC functions* or *delta-convex functions*. DC functions are important in optimization, especially in non-convex and global optimization. For excellent reviews of the various properties of DC functions, see Hiriart-Urruty (1988) or Bacak and Borwein (2011). Interesting for our purposes is that DC functions are almost everywhere twice differentiable, and that all two times continuously differentiable functions are DC.

It follows from the general properties of convex and DC functions that σ is both uniformly continuous and locally Lipschitz, in fact Lipschitz on each compact subset of $\mathbb{R}^{n\times p}$ (Rockafellar (1970), theorem 10.4). The fact that σ is only locally Lipshitz is due entirely to the quadratic part η^2, because ρ is globally Lipschitz.

2.4.4 Negative Dissimilarities

There are perverse situations in which some weights and/or dissimilarities are negative (Heiser (1991)). Define $w_{ij}^+ := \max(w_{ij}, 0)$ and $w_{ij}^- := -\min(w_{ij}, 0)$.

Thus both w_{ij}^+ and w_{ij}^- are non-negative, and $w_{ij} = w_{ij}^+ - w_{ij}^-$. Make the same decomposition of the δ_{ij}.

Then

$$\rho(X) = \sum(w_{ij}^+\delta_{ij}^+ + w_{ij}^-\delta_{ij}^-)d_{ij}(X) - \sum(w_{ij}^+\delta_{ij}^- + w_{ij}^-\delta_{ij}^+)d_{ij}(X), \tag{2.91}$$

and

$$\eta^2(X) = \sum w_{ij}^+ d_{ij}^2(X) - \sum w_{ij}^- d_{ij}^2(X). \tag{2.92}$$

Note that both ρ and η^2 are no longer convex, but both are DC, and consequently so is σ.

A bit more

2.5 Stationary Points

- A function f has a *global minimum* on X at $\hat{x}$ if $f(\hat{x}) \leq f(x)$ for all $x \in X$.
- A function f has a *local minimum* on X at $\hat{x}$ if there is a neighborhood $\mathcal{N}$ of $\hat{x}$ such that $f(\hat{x}) \leq f(x)$ for all $x \in \mathcal{N} \cap X$.
- A function f has a *singular point* at x if it is not differentiable at x.
- A function f has a *stationary point* at x if it is differentiable at x and $df(x) = 0$.
- A function f has a *saddle point* at a stationary point x if it is neither a local maximum nor a local minimum.
- Global and local maxima of f are global and local minima of $-f$.

Theorem 2.12. *At a stationary point of stress we have $\eta(X) \leq 1$.*

Proof. If X is stationary we have $VX = B(X)X$ and thus $\rho(X) = \eta^2(X)$. Consequently $\sigma(X) = 1 - 2\rho(X) + \eta^2(X) = 1 - \eta^2(X)$ and because $\sigma(X) \geq 0$ we see that X must be in the ellipse $\{Z \in \mathbb{R}^{n\times p} \mid \eta^2(Z) \leq 1\}$. □

Theorem 2.12 is important, because it means that we can require without loss of generality that X is in the ellipsoidal disk $\eta(X) \leq 1$, which is a compact convex set.

2.5.1 Local Maxima

Theorem 2.13. *stress has a single local maximum at $X = 0$ with value 1.*

Proof. At $X = 0$ we have for the one-sided directional derivative

$$\mathbb{D}_+\sigma(0,Y) = \lim_{\alpha\downarrow 0}\frac{\sigma(0+\alpha Y)-\sigma(0)}{\alpha} = -2\rho(Y) \leq 0, \tag{2.93}$$

which implies that stress has a local maximum at zero.

To show that the local maximum is unique suppose that there is a local maximum at $X \neq 0$. Then on the line through zero and X there should be a local maximum at X as well. But

$$\sigma(\alpha X) = 1 - 2\alpha\rho(X) + \alpha^2\eta^2(X), \tag{2.94}$$

is a convex quadratic, which consequently cannot have a local maximum at X. □

2.5.2 Local Minima

The main result on local minima of stress is due to De Leeuw (1984c). We give a slight strengthening of the result, along the lines of De Leeuw (2018a), with a slightly simplified proof. Theorem 2.14 proves that a necessary condition for a local minimum at X is that for i and j with $w_{ij}\delta_{ij} > 0$ we have $d_{ij}(X) > 0$, i.e. objects i and j are mapped to different points.

Theorem 2.14. *If stress has a local minimum at X then * $B(X)X = VX$. * $d_{ij}(X) > 0$ whenever $w_{ij}\delta_{ij} > 0$.*

Proof. If stress has a local minimum at X then $d_+\sigma(X;Y) \geq 0$ for all Y. Equation (2.60) tell us that

$$d_+\sigma(X;Y) = \text{tr}\, Y'(V-B(X))X - \sum_{1\ \leq i<j\leq n}\sum \{w_{ij}\delta_{ij}d_{ij}(Y) \mid d_{ij}(X) = 0 \text{ and } w_{ij}\delta_{ij} > 0\}. \tag{2.95}$$

ain) \end{equation} Consider a direction Y with all $d_{ij}(Y) > 0$ and such that then $d_+\sigma(X;Y) \leq 0$. This is always possible, because if we have and Y with $d_+\sigma(X;Y) > 0$ we simply switch to $-Y$. Now $d_+\sigma(X;Y)$ in (2.95) is the sum

of two terms which are both non-positive, and they satisfy $d_+\sigma(X;Y) \geq 0$ if and only if they are both zero. For the second term this means that at a local minimum the summation is empty and there is no $d_{ij}(X) = 0$ whenever $w_{ij}\delta_{ij} = 0$. For the first term it means that $(V - B(X))X = 0$. □

De Leeuw (1984c) concluded that if $w_{ij}\delta_{ij} > 0$ for all $i < j$ then stress is differentiable at a local minimum. But more is true, because it s not necessary to require $w_{ij}\delta_{ij} > 0$ for this result.

Corollary 2.4. *If stress has a local minimum at X then it is differentiable at X and has $\nabla\sigma(X) = 0$.*

Proof. At a local minimum we can indeed have $d_{ij}(X) = 0$ if $w_{ij}\delta_{ij} = 0$. But if $w_{ij} = 0$ then stress does not depend on $d_{ij}(X)$ at all, so $d_{ij}(X) = 0$ does not influence differentiability. If $w_{ij} > 0$ and $\delta_{ij} = 0$ then stress depends on $d_{ij}(X)$ only through the term $w_{ij}d_{ij}^2(X)$, which is differentiable even if $d_{ij}(X) = 0$. □

Theorem 2.14 and its corollary 2.4 are the main reason why, at least in basic scaling, we can largely ignore the problems with differentiability. These results have been extended to least squares MDS with Minkovski distances by Groenen, Mathar, and Heiser (1995). In a neighborhood of each local minimum the loss function is differentiable, so eventually convergent descent algorithms do not have problems with non-differentiable ridges. This result is of major importance for both practical and theoretical reasons, as emphasized for example by Pliner (1996).

Second order necessary conditions (since differentiable at local minimum)

2.5.3 Saddle Points

At a saddle point X stress is differentiable and $d\sigma(X) = 0$. But there are directions of decrease and increase from X, and thus stress does not have a local minimum there.

If $VX = B(X)X$ and $d_{ij}(X) = 0$ for some $w_{ij}\delta_{ij} > 0$ then there is an Y such that $\mathbb{D}_+\sigma(X, Y) < 0$.

Theo Suppose $VX = B(X)X$ then $V(X \mid 0) = B(X|0)(X|0)$

Corr Suppose $VX = B(X)X$ and X is of rank $r < p$. Then $XL = (Z|0)$ and thus $VZ = B(Z)Z$.

If $VX = B(X)X$ and X is singular then X is a saddle point.

2.5.4 An Example

Let's look at a small example, already analyzed in many different places (e.g. De Leeuw (1988), Trosset and Mathar (1997)). It has four points, all dissimilarities to one and all weights are equal to $\frac{1}{6}$.

2.5.4.1 Regular Tetrahedron

In three dimensions the global minimum is equal to zero, with the points mapped into the vertices of a regular tetrahedron. Points can be assigned to the vertices in $4! = 24$ ways, and each such assignment defines a global minimum. In fact each assigment defines a continuum of global minimizers, because all rotations of any of the regular tetrahedra also give global minimizers. It seems as if there are 24 rotation manifolds with global minimizers. But some of the 24 assigments of points to vertices are equivalent in the sense that they are rotations of each other (which includes reflections). It turns out there are three equivalence classes of eight assignments each. Thus there are three different disjoint rotation manifolds of global minimizers, not 24.

The stress value for any regular tetrahedron is zero, and the eigenvalues of the Hessian are

```
##  [1] +0.333333 +0.166667 +0.166667 +0.166667 +0.083333 +0.083333 +0.000000
##  [8] +0.000000 +0.000000 +0.000000 +0.000000 -0.000000
```

For four points in three dimensions we have $np - \frac{1}{2}p(p+1) = 6$ non-trivial eigenvalues. Since the six largest values are all positive our regular tetrahedra are isolated, in the sense that if we move away from the each of the corresponding rotation manifolds the stress increases.

2.5.4.2 Singularity

The next stationary point is somewhat deviously constructed. We take four points equally spaced on a line (a stationary point in one dimension) and add two zero dimensions. Then we do a random rotation of configuration to somwhat hide its singularity. We already know this configuration defines a saddle point in three-dimensional configuration space. For the resulting configuration the stress is 0.083333, and the eigenvalues of the Hessian are

```
##  [1] +0.333333 +0.333333 +0.333333 +0.000000 -0.000000 -0.000000 -0.000000
##  [8] -0.000000 -0.166667 -0.166667 -0.277778 -0.277778
```

There are four negative eigenvalues, and thus we confirm we have a saddle-point.

2.5.4.3 Equilateral Triangle with Centroid

The next configuration is an interesting one. It is in two-dimensions, with three points in the corners of an equilateral triangle, and a fourth point in the centroid of the first three. The 24 assigments of the four points to the four positions in thus case give four rotational equivalence classes of six assignments each. This particular arrangements has four disjoint rotational manifolds. The stress is 0.033494, and the eigenvalues of the Hessian are

```
## [1] +0.333333 +0.255983 +0.255983 +0.000000 +0.000000 +0.000000 +0.000000
## [8] -0.000000
```

Note that the Hessian is positive semi-definite, with three positive eigenvalues. This made De Leeuw (1988) think that this arrangement of points defined a non-isolated local minimum. Bad mistake. Since $3 < np - \frac{1}{2}p(p+1) = 5$ the singular Hessian does not guarantee that we have a local minimum. Trosset and Mathar (1997) showed, using symbolic computations, that the configuration and its permutations and rotations defines a family of saddle points. Further on in this section we will look in more detail what happens in this case.

2.5.4.4 Square

Four points in the corners of a square give the global minimum in two dimensions (De Leeuw and Stoop (1984)). There are three isolated rotational manifolds for such squares, all with stress 0.014298, and with eigenvalues of the Hessian

```
## [1] +0.333333 +0.195262 +0.138071 +0.138071 +0.138071 +0.000000 +0.000000
## [8] -0.000000
```

In one dimension the global minimum is attained for four points equally spaced on a line. Thus there are $n!$ different global minimizers but by reflection only $\frac{1}{2}(n!)$ give different distances.

2.5.4.5 Non-global Local Minima

It turns out be be difficult in our example to find non-global local minima. In an heroic effort we looked at 100,000 smacof runs with a random start to find other local minima. In 9.9996×10^4 cases smacof converges to the square, in 4 it stops at the equilateral triangle with center, but only because the limit on the number of iterations (1000) is reached. This confirms the computational results reported by De Leeuw (1988) and Trosset and Mathar (1997). It also confirms the theoretical
result that gradient descent algorithms with random starts almost surely avoid saddle points and converge to local minima (Lee et al. (2016)), although avoiding the saddle points may take exponential time (Du et al. (2017)). In any case, it seems safe to conjecture that for our small and maximally symmetric example all local minima are global.

Trosset and Mathar (1997), in their search for non-global local minima, consequently are forced to use another example. They used equal weights, but choose the dissimilarities as the Euclidean distances between the four corners of a square, ordered counterclockwise from the origin. Thus the global minimum of stress is zero. In 1000 smacof runs with random start we find a this zero local minimum 599 times, while we converge 401 times to another stationary point with stress 0.0334936.

The two configurations are plotted in figure 2.1, with the global minimizer in red. The non-global configuration (in blue) is rotated to best least squares fit

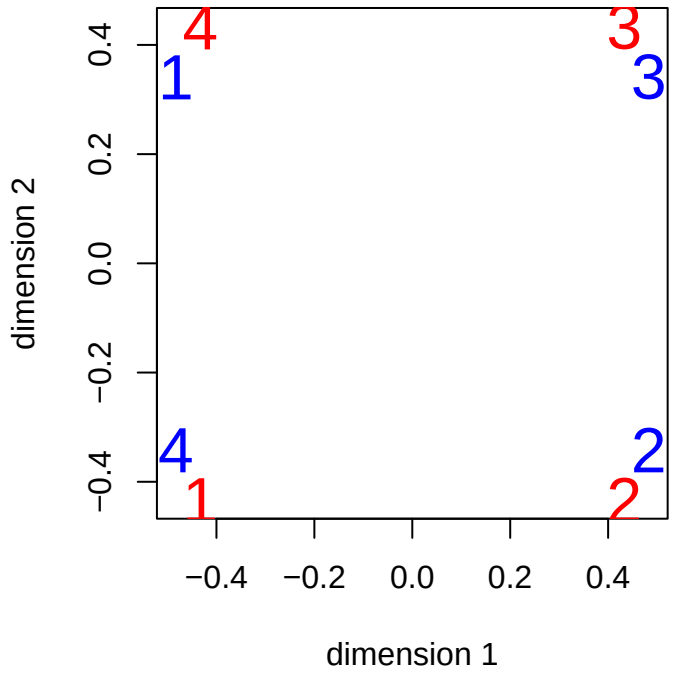


Figure 2.1: Trosset/Mathar Configurations

with the first one, using simple Procrustus (Gower and Dijksterhuis (2004)). Note that it is a rectangle, but not a square. The eigenvalues of the Hessian at the non-global minimum configuration are

```
## [1] +0.333333 +0.211325 +0.122008 +0.122008 +0.122008 +0.000000 +0.000000
## [8] -0.000000
```

verifying that we indeed have an isolated local minimum. Trosset and Mathar (1997) verify this using a mix of symbolic and floating point calculation.

We can generate an additional example using the function equalDelta() in equaldelta.R. Its arguments are n, p, m, where n is the order of the dissimilarity and weight matrices, which have all their non-diagonal elements equal. Argument n and p define the space of configuration matrices, and m is the number of smacof runs with a random start.

```
## [1] 658
```

```
## [1] 342
```

```
## itel      1  eiff    0.0000000000  sold    0.0381249159  snew    0.03812491
```

```
## itel      1  eiff    0.0000000000  sold    0.0357265590  snew    0.03572655
```

```
##  [1] +0.200000 +0.129354 +0.129354 +0.102242 +0.102242 +0.079940 +0.079940
##  [8] +0.005686 +0.005686 +0.000000 -0.000000 -0.000000
```

```
##  [1] +0.200000 +0.114739 +0.114739 +0.107180 +0.073205 +0.073205 +0.051287
##  [8] +0.051287 +0.039230 +0.000000 -0.000000 -0.000000
```

2.5.4.6 Directions of Descent

We now go back to the stationary equilateral triangle with center. We have seen that the gradient at this configuration is zero and the Hessian is positive semi-definite but rank-deficient. A *descent direction* at X is any configuration Y such that $\sigma(X + \epsilon Y) < \sigma(X)$ if ϵ is small enough. In our example, with X

the triangle with center, we must choose Y in the null space of the Hessian, because otherwise Y is a direction of accent. The null space has two trivial dimensions, X and XA with A anti-symmetric. The non-trivial null space has dimension three, and we choose a basis of three orthonormal directions. Then

$$\sigma(X + \epsilon Y) = \sigma(X) + 0 + 0 + \frac{1}{6}\epsilon^3 d^3\sigma(X)(Y, Y, Y) + o(\epsilon^3),$$

and we can find a descent direction if $d^3\sigma(X)(Y, Y, Y) \neq 0$.

```
## [1] +0.066987 -0.000000 +0.000000 +0.015659
```

```
## [1] +0.066987 -0.000000 +0.000000 +0.421558
```

```
## [1] +0.066987 +0.000000 +0.000000 -0.003449
```

Chapter 3

Stress Spaces

Stress is a function of many variables. Consequently there are many equivalent ways to define it by transforming the parameter space. In this chapter we discuss the major parametrizations that have actually been used. Each has its advantages and disadvantages.

3.1 Configuration Space

So far we have defined stress on $\mathbb{R}^{n\times p}$, the space of all matrices with n rows and p columns. We call this *configuration space*. The usual MDS representation of a dissimilarity matrix is a scatterplot of n points in p dimensions. Therefor there is little doubt that configuration space is the most natural MDS space, and most of our theorems and derivations use this space.

Nevertheless, configuration space has some disadvantages. Even for n as small as four and p as small as two the dimension of the space of configurations is eight, and there is no compelling way to visualize stress as a function of eight variables. Secondly, if we work in configuration space we have to keep the indeterminacies of the representation in mind. Because of translational indeterminacy, minimizing stress over $X \in \mathbb{R}^{n\times p}$ will give the same result as minimizing stress over $X \in \mathbb{R}_C^{n\times p}$, the $p(n-1)$-dimensional subspace of all column-centered matrices. Because of translational and rotational indeterminacy it will also give the same result as minimizing stress over $X \in \mathbb{R}_{CT}^{n\times p}$, the $np - \frac{1}{2}p(p+1)$ dimensional subspace of all centered lower triangular

configurat ions (which have $x_{ij} = 0$ for all $j > i$). Or the same result as minimizing stress over the $np - \frac{1}{2}p(p+1)$ dimensional nonlinear manifold $\mathbb{R}_{CO}^{n\times p}$ of all centered orthogonal configurations. Especially rotational indeterminacy complicates our analysis of MDS algorithms that operate on configuration space.

Another disadvantage of configuration space is that it needlessly complicates some of our formulas and derivations. Not all pairs of coordinate in a configuration X have the same status. Some pairs belong to the same row of the matrix, and some pairs to different rows. Some pairs of coordinates are in the same column, and some are in different columns. This complicates formulas which depend on considering pairs of coordinates, such as second derivatives.

One way to make a picture in configuration space is to plot the np individual coordinates as functions of a one-dimensional perturbation. In figure 3.1 we illustrate this with an example. We have four objects, with all weights and dissimilarities equal, and the 4×2 configuration is an equilateral triangle together with its centroid. Everything is suitably scaled, and we perturb each coordinate using 101 values equally spaced between -1 and +1.

```
delta <- wdef(4)
w <- wdef(4)
w <- w / sum(w)
s <- sum (w * delta ^ 2)
delta <- delta / sqrt (s)
x <- matrix(c(0,0,1,0,.5,sqrt(3)/2),3,2,byrow = TRUE)
x <- rbind(x, apply(x, 2, mean))
x <- apply(x, 2, function(x) x - mean(x))
d <- as.matrix(dist(x))
s <- sum (w * delta * d) / sum (w * d * d)
x <- x * s
d <- d * s
h <- smacofR(w,
             delta,
             p=2,
             xold = x,
             eps=1e-15,
             xstop=TRUE)
```

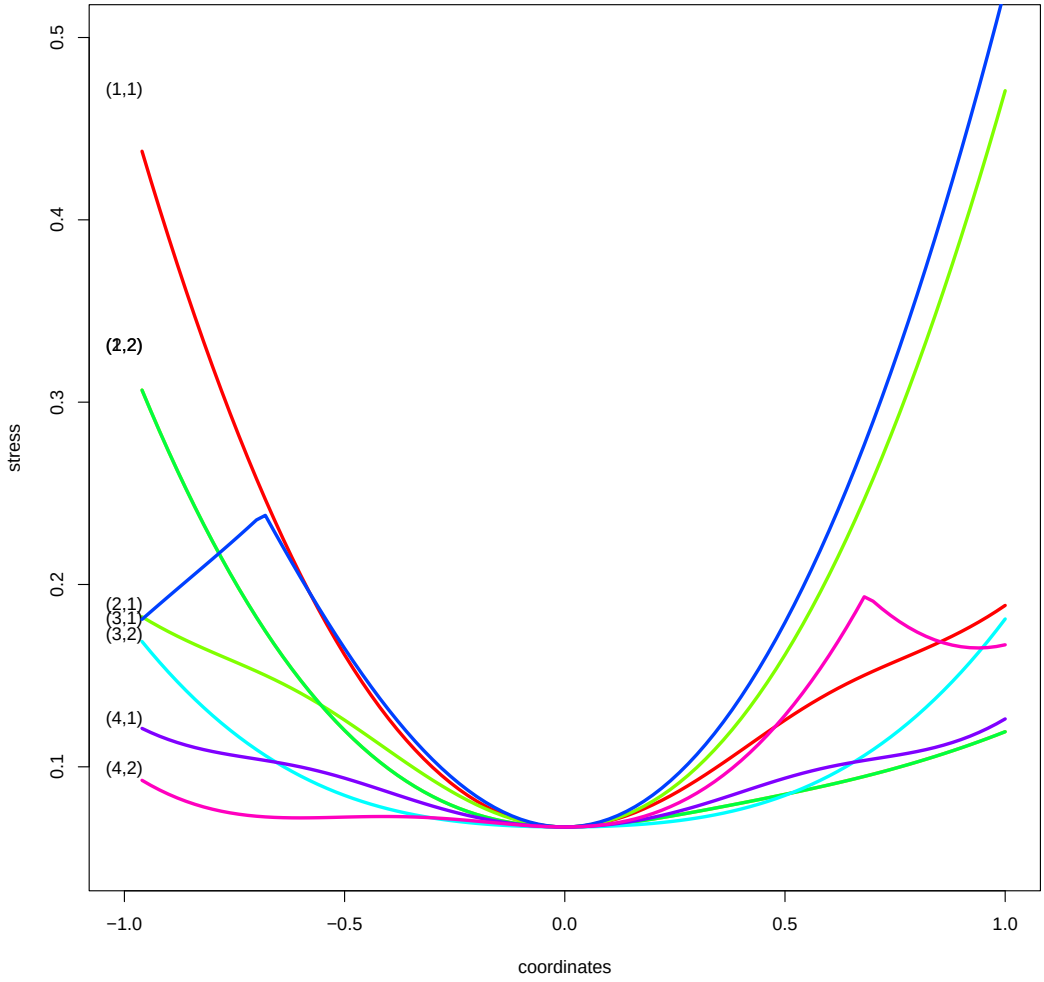


Figure 3.1: One Coordinate at a Time

```
## itel      1  eiff    0.0000000000  sold    0.0334936491  snew    0.03349364
```

```
print(eigen(h$h)$values)
```

```
## [1]  3.333333e-01  2.559831e-01  2.559831e-01  2.580122e-16  2.238367e-16
## [6]  5.765257e-17  3.133650e-17 -1.657110e-16
```

```
x<-matrix(c(0,2,1,1,0,0,sqrt(3),sqrt(3)/3),4,2)
z<-matrix(c(0,-2,sqrt(3),-2-sqrt(3)/3,0,0,sqrt(3),2+sqrt(3)),4,2)
eps <- 0.00001
d <- as.matrix(dist(x + eps * z))
smacofLossR(d, w, delta)
```

```
## [1] 0.2559831
```

3.1.1 Zero Distance Subspaces

A *zero-distance subspace* is a subspace of configuration space in which one or more distances are zero. That a zero distance indeed defines a subspace follows from the fact that the nonlinear equation $d_{ij}(X) = 0$ is equivalent to the homogeneous linear equation $x_i = x_j$.

The number of zero-distance subspaces is the same as the number of set partitions of n objects, which is the Bell number B_n. Bell numbers are defined by the recursion

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k, \tag{3.1}$$

with $B_0 = 1$. The next ten Bell numbers $B_1, \cdots, B_{10}$ are 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 11597. So there are lots of zero-distance subspaces.

The gradient in configuration space for all $X \in \mathbb{R}^{n \times p}$ is

$$\nabla \sigma(X) = (V - B(X))X.$$

$$\nabla \tilde{\sigma}(\theta) = (\tilde{V}\theta - \tilde{B}(\theta)\theta)$$

$$\frac{1}{2}\{\nabla\tilde{\sigma}(\theta)\}_s = \text{tr } Y_s'(VX - B(X)X) = \text{tr } Y_s'\nabla\sigma(X),$$

with $X = \sum_{s=1}^r \theta_s Y_s$.

Thus $\nabla\tilde{\sigma}(\theta) = 0$ if and only if $\nabla\sigma(X) \in \mathcal{Y}_\perp$, the subspace of $\mathbb{R}^{n\times p}$ orthogonal to $\mathcal{Y}$. Specifically $\nabla\sigma(X) = 0$ implies $\nabla\tilde{\sigma}(\theta) = 0$. If the Y_s span all of $\mathbb{R}^{n\times p}$ then $\nabla\tilde{\sigma}(\theta) = 0$ if and only if $\nabla\sigma(X) = 0$.

For the relationship between the minimization problems in coefficient space and configuration space we also study the relationship between the two Hessians.

For all X and Z in configuration space

$$\frac{1}{2}\nabla^2\sigma(X)(Z,Z) = \text{tr } Z'VZ - \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}(X)}\left\{\text{tr } Z'A_{ij}Z - \frac{\{\text{tr } Z'A_{ij}Y\}^2}{d_{ij}^2(X)}\right\}$$

For any θ in coefficient space

$$\frac{1}{2}\nabla^2\tilde{\sigma}(\theta) = \tilde{V} - \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}(\theta)}\left\{\tilde{A}_{ij} - \frac{\tilde{A}_{ij}\theta\theta'\tilde{A}_{ij}}{d_{ij}^2(\theta)}\right\}$$

and thus

$$\xi'\nabla^2\tilde{\sigma}(\theta)\xi = \nabla^2\sigma(X)(Z,Z).$$

where $Z := \sum_{s=1}^r \xi_s Y_s$ and $X := \sum_{s=1}^r \theta_s Y_s$.

Thus if θ is a local minimum in coefficient space then $\nabla^2\sigma(X)$ is positive semi-definite on the subspace $\mathcal{Y}$. If $X \in \mathcal{Y}$ but $Z \notin \mathcal{Y}$ it is perfectly possible that $\nabla^2\sigma(X)(Z,Z) < 0$, and thus X can be a saddle point in configuration space. If $X \in \mathcal{Y}$ is a local minimum in configuration space, then θ is a local minimum in coefficient space. Of course if $\mathcal{Y}$ is the whole space then θ is a local minimum in coefficient space if and only if X is a local minimum in configuration space.

$$\sigma(X(\theta+\epsilon\xi)) = \sigma(X(\theta) + \epsilon\mathcal{D}X(\theta)(\xi) + \frac{1}{2}\epsilon^2\mathcal{D}^2X(\theta)(\xi,\xi)) =$$
$$\sigma(X(\theta))+\epsilon\mathcal{D}\sigma(X(\theta))\mathcal{D}X(\theta)\xi+\frac{1}{2}\epsilon^2\{\mathcal{D}\sigma(X(\theta))\mathcal{D}^2X(\theta)(\xi,\xi)+\mathcal{D}\sigma(X(\theta))\mathcal{D}\sigma(X(\theta))\} \tag{3.2}$$

$$\{\mathcal{D}^2X(\theta)(\xi,\xi)\}_{ip} = \sum\sum \xi_s\xi_t\mathcal{D}_{st}x_{ip}(\theta) = \xi' H_{ip}(\theta)\xi$$
$$\{\mathcal{D}X(\theta)(\xi)\}_{ip} = \sum \xi_s\mathcal{D}_s x_{ip}(\theta) = G_{ip}(\theta)\xi$$

$$\mathcal{D}\sigma(X(\theta)) = \{V - B(X(\theta))\}X(\theta) = F(\theta)$$

$$\mathcal{D}\sigma(\theta) = \mathcal{D}\sigma(X(\theta))\mathcal{D}X(\theta)$$

$\sigma(x_{11}(\theta),\cdots,x_{np}(\theta))$

$$\mathcal{D}_s\sigma(\theta) = \sum_{i=1}^{n}\sum_{s=1}^{p}\mathcal{D}_{ip}\sigma(X(\theta))\mathcal{D}_s x_{ip}(\theta)$$

$$\mathcal{D}_{st}\sigma(\theta) = \sum_{i=1}^{n}\sum_{s=1}^{p}\sum_{j=1}^{n}\sum_{r=1}^{p}\mathcal{D}_{is,jr}\sigma(X(\theta))\mathcal{D}_s x_{is}(\theta)\mathcal{D}_t x_{jr}(\theta) + \sum_{i=1}^{n}\sum_{s=1}^{p}\mathcal{D}_{is}\sigma(X(\theta))\mathcal{D}_{st}x_{is}(\theta)$$

Now let $Y_1, Y_2, \cdots, Y_r$ be linearly independent configurations in $\mathbb{R}^{n\times p}$, and consider minimizing stress over all linear combinations X of the form $X = \sum_{s=1}^{r}\theta_s Y_s$.

Each linear combination can be identified with a unique vector $\theta \in \mathbb{R}^r$, the coefficients of the linear combination. Thus we can also formulate our problem as minimizing stress over *coefficient space*, which is simply $\mathbb{R}^r$. We write $d_{ij}(\theta)$ for $d_{ij}(X)$ and $\sigma(\theta)$ for $\sigma(X)$. Note that $d_{ij}(\theta) = \sqrt{\theta' C_{ij}\theta}$, where C_{ij} has elements $\{C_{ij}\}_{st} := \text{tr } Y_s' A_{ij} Y_t$.

If the Y_t are actually a basis for configuration space (i.e. if $r = np$) then minimizing over configuration space and coordinate space is the same thing. For the Y_t we could choose all rank one matrices, for example, of the form $a_i b_s'$ where the a_i are a basis for $\mathbb{R}^n$ and the b_s are a basis for $\mathbb{R}^p$. And, in particular, the a_i and b_s can be chosen as unit vectors of length n and p, respectively. That case we have $C_{ij} = I_p \otimes A_{ij}$, i.e. the direct sum of p copies of A_{ij}. Also if $\theta = \text{vec}(X)$ then $d_{ij}(X) = \sqrt{\theta'(I_p \otimes A_{ij})\theta}$

If $r < np$ then coefficient space defines a proper subspace of configuration space. If it happens to be the $(n-1)p$ dimensional subspace of all column-centered matrices, then the two approaches still define the same minimization

problem. But in general $r < (n-1)p$ with the Y_s column-centered defines a *constrained MDS problem*, which we analyze in more detail in chapter 15.

Coefficient space is also a convenient place to deal with rotational indeterminacy in basic MDS. It follows from QR decomposition that any configuration matrix can be rotated in such a way that it upper diagonal elements (the x_{ij} with $i < j$) are zero (define X_p to be the first p rows of X, compute $X_p' = QR$ with Q square orthonormal and R upper triangular, thus $X_p = R'Q'$ and $X_pQ = R'$, which is lower triangular). The column-centered upper triangular configurations are a subspace of dimension $p(n-1) - p(p-1)/2$, and we can choose the Y_s as a basis for this subspace. In this way we eliminate rotational indeterminacy in a relatively inexpensive way.

If $X = \sum_{s=1}^r \theta_s Y_s$ then we define the symmetric positive definite matrix $B(\theta)$ of order r with elements

$$b_{st}(\theta) := \text{tr } Y_s' B(X) Y_t, \tag{3.3}$$

where $B(X)$ is the usual B-matrix of order n in configuration space, defined in equation (2.16). Also define V of order r by

$$v_{st} := \text{tr } Y_s' V Y_t, \tag{3.4}$$

where the second V, of order n, is given by equation (2.12). Then

$$\sigma(\theta) = 1 - 2\ \theta' B(\theta)\theta + \theta' V \theta. \tag{3.5}$$

The relationship between the stationary points in configuration space and coefficient space is fairly straightforward.

Theorem 3.1. *Suppose θ is in coefficient space and $X = \sum_{s=1}^r \theta_s Y_s$ is the corresponding point in configuration space.*

1. *If X is a stationary point in configuration space then θ is a stationary point in coefficient space.*
2. *If θ is a stationary point in coefficient space then X is a stationary point in configuration space if and only if rank$(Y_1 \mid \cdots \mid Y_r) \geq n-1$. (THIS IS WRONG)*

Proof. We have $B(X)X = VX$, i.e.

$$\sum_{s=1}^{r} \theta_s B(X) Y_s = \sum_{s=1}^{r} \theta_s V Y_s. \tag{3.6}$$

Premultiplying both sides by Y_t' and taking the trace gives $B(\theta)\theta = V\theta$. This proves the first part.

For the second part, suppose $B(\theta)\theta = V\theta$ and define $X = \sum_{s=1}^{r} \theta_s Y_s$. Then

$$\sum_{t=1}^{r} \text{tr } Y_s'(B(X) - V)X = 0. \tag{3.7}$$

Thus $B(X)X = VX$ if and only if $Y_s'(B(X) - V)X = 0$ for all s, which translates to the rank condition in the theorem (this is WRONG, correct). □

The advantage of working in coefficient space is that formulas tend to become more simple. Functions are defined on $\mathbb{R}^r$, and not a space of matrices, in which some coordinates belong to the same point (row) and others to other points (rows), and some are on the same dimension (column), while others are on different dimensions (columns).

Note that expressions such as (3.6) and (3.6) simplify if the Y_s are V-orthonormal, i.e. if tr $Y_s'VY_t = \delta^{st}$ and thus $V = I$. It is easy to generate such an orthonormal set from the original Y_s by using the Gram-Schmidt process. The R function gramy() in utilities.R does exactly that. Coefficient space, which is the span of the Y_s, is not changed by the orthogonalization process.

For a V-orthonormal set Y we have the stationary equations $B(\theta)\theta = \theta$, which says that θ is an eigenvector of $B(\theta)$ with eigenvalue 1.

The Hessian is

$$\mathcal{D}^2\sigma(\theta) = I - H(\theta), \tag{3.8}$$

with

$$H(\theta) := \mathcal{D}^2\rho(\theta) = \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}(\theta)}\left\{C_{ij} - \frac{C_{ij}\theta\theta' C_{ij}}{\theta' C_{ij}\theta}\right\}. \tag{3.9}$$

We have $0 \lesssim H(\theta) \lesssim B(\theta)$ and thus $I - B(\theta) \lesssim \mathcal{D}^2\sigma(\theta) \lesssim I$.

Hessian in coef and conf space

3.2 Spherical Space

$$\min_X \sigma(X) = \min_{\lambda\geq 0} \min_{\eta^2(X)=1} \sigma(\lambda X) = \min_{\eta^2(X)=1} \min_{\lambda\geq 0} \sigma(\lambda X) = \min_{\eta^2(X)=1} 1 - \rho^2(X). \tag{3.10}$$

We see that basic MDS can also be formulated as maximization of ρ over the ellipsoid $\{X \mid \eta^2(X) = 1\}$ or, equivalently, over the convex ellipsoidal disk $\{X \mid \eta^2(X) \leq 1\}$. A similar formulation is available in coefficient space.

This shows that the MDS problem can be seen as a rather special nonlinear eigenvalue problem. Guttman (1968) also discusses the similarities of MDS and eigenvalue problems, in particular as they relate to the power method. In linear eigenvalue problems we maximize a convex quadratic form, in the MDS problem we maximize the homogeneous convex function ρ, in both cases over an ellipsoidal disk. The sublevel sets of ρ defined as $\mathcal{L}_r := \{X \mid \rho(X) \leq r\}$ are nested convex sets containing the origin. The largest of these sublevel sets that still intersects the ellipsoid $\eta^2(X) = 1$ corresponds to the global minimum of stress.

In two and maybe three dimensions graphical method.

$\alpha X + \beta Y$ in sphere space one-dimensional

3.3 Distance Space

$$\sigma(D) = \min_{D\in\mathbb{D}} \sum_{1\leq i<j\leq n} \sum w_{ij}(\delta_{ij} - d_{ij})^2.$$

$\mathbb{D}$ is the set of p-dimensional Euclidean distance matrices, which is not convex.

$\mathbb{D}$ is the set of Euclidean distance matrices, which is not convex.

If $D_1 \times D_2 = 0$ then they span a convex cone.

$\tau(\alpha D_1 + (1-\alpha)D_2)^{(2)}) = \alpha^2\tau(D_1^2) + (1-\alpha)^2\tau(D_2^2) + 2\alpha(1-\alpha)\tau(D_1 \times D_2)$

So if $\tau(D_1 \times D_2) \gtrsim 0$ convex.

$\mathbb{D}$ is the set of distance matrices, which is convex.

Distance matrices are defined by linear inequalities.

$\mathbb{D}$ is the set of ultrametric matrices, $d_{ij} \leq \max(d_{ik}, d_{jk})$, which is not convex.

If $D \in \mathbb{D}$ then $\sqrt{D} \in \mathbb{D}$

3.4 Squared Distance Space

$$\min_{D \in \mathbb{E}} \sum_{1 \leq i < j \leq n} \sum w_{ij}(\delta_{ij} - d_{ij})^2.$$

$\mathbb{E}$ is the set of squared Euclidean distance matrices, which is convex.

If $E \in \mathbb{E}$ then $\sqrt{E} \in \mathbb{D}$

3.5 Gramian Space

We can write $d_{ij}^2(X) = \text{tr } X'A_{ij}X = \text{tr } A_{ij}C$, with $C = XX'$. This shows minimizing stress can also be formulated as minimizing

$$\sigma(C) = 1 + \text{tr } VC - 2 \sum_{1 \leq i < j \leq n} \sum w_{ij}\delta_{ij}\sqrt{\text{tr } A_{ij}C} \tag{3.11}$$

over all $C \gtrsim 0$ of rank $r \leq p$.

3.6 Pictures of Stress

Even for n as small as four and p as small as two the dimension of the space of centered configurations is six, and there is no natural way to visualize a function of six variables. What we can do is plot stress on two-dimensional subspaces, either as a contour plot or as a perspective plot. Our two-dimensional

subspaces are of the form $\alpha X + \beta Y$, where X and Y are fixed configurations. Much of this chapter is a modified, and in some places expanded, version of De Leeuw (2016c).

Throughout we use a small example of order $n = 4$ which all dissimilarities equal. The same example has been analyzed by De Leeuw (1988), De Leeuw (1993), Trosset and Mathar (1997), and Zilinskas and Poslipskyte (2003). For this example a global minimum in two dimensions has its four points in the corners of a square. That is our X, which has stress 0.0285955. Our Y is another stationary point, which has three points in the corners of an equilateral triangle and the fourth point in the center of the triangle. Its stress is 0.0669873. We column-center the configurations and scale them so that they are actually stationary points, i.e. so that $\eta^2(X) = \rho(X)$ and $\eta^2(Y) = \rho(Y)$. The example is chosen in such a way that there are non-zero α and β such that $d_{12}(\alpha X + \beta Y) = 0$. In fact d_{12} is the only distance that can be made zero by a non-trivial linear combination.

Another way of looking at the two configurations is that X are four points equally spaced on a circle, and Y are three points equally spaced on a circle with the fourth point in the center of the circle. De Leeuw (1988) erroneously claims that Y is a non-isolated local minimum of stress, but Trosset and Mathar (1997) have shown there exists a descent direction at Y, and thus Y is actually a saddle point. Of course the stationary points defined by X and Y are far from unique, because we can permute the four points over the corners of the square and the triangle in many ways.

3.6.1 Coefficient Space

Configurations as a linear combination of a number of given configurations have already been discussed in general in chapter 2, section 3 as the transformation from configuration space to coefficient space. Since we are dealing here with the special case of linear combinations of only two configurations we specialize some of these general results.

We start with $d_{ij}^{(}\theta) = \theta' T_{ij} \theta$, where θ has elements α and β, and where T is the 2×2 matrix with elements

$$t_{ij} := \begin{bmatrix} \text{tr } X' A_{ij} X & \text{tr } X' A_{ij} Y \\ \text{tr } Y' A_{ij} X & \text{tr } Y' A_{ij} Y \end{bmatrix} \tag{3.12}$$

Then

$$\tilde{\sigma}(\theta) := 1 - 2\ \theta' C(\theta)\theta + \theta' U\theta, \tag{3.13}$$

where, using $Z(\theta) = \alpha X + \beta Y$,

$$C(\theta) := \begin{bmatrix} \text{tr}\ X'B(Z(\theta))X & \text{tr}\ X'B(Z(\theta))Y \\ \text{tr}\ Y'B(Z(\theta))X & \text{tr}\ Y'B(Z(\theta))Y \end{bmatrix}, \tag{3.14}$$

and

$$U := \begin{bmatrix} \text{tr}\ X'VX & \text{tr}\ X'VY \\ \text{tr}\ Y'VX & \text{tr}\ Y'VY \end{bmatrix}. \tag{3.15}$$

We have used $\tilde{\sigma}$ in equation (3.13) to distinguish stress on the two-dimensional space of coefficients from stress on the eight-dimensional space of 4×2 configurations. Thus $\tilde{\sigma}(\alpha, \beta) = \sigma(\alpha X + \beta Z)$.

The gradient at θ is

$$\nabla\tilde{\sigma}(\theta) = U\theta - C(\theta)\theta, \tag{3.16}$$

and the Hessian is

$$\nabla^2\tilde{\sigma}(\theta) = U - \sum_{1\le i<j\le n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}(\theta)}\left\{T_{ij} - \frac{T_{ij}\theta\theta' T_{ij}}{\theta' T_{ij}\theta}\right\}. \tag{3.17}$$

Theorem 3.2. *If $B(X)X = VX$ and $\theta = \begin{bmatrix} 1 & 0 \end{bmatrix}$ then $C(\theta)\theta = U\theta$.*

Proof. If $B(X)X = VX$ and $\theta = \begin{bmatrix} 1 & 0 \end{bmatrix}$ then, by equations (3.14) and (3.15),

$$U - C(\theta) = \begin{bmatrix} 0 & 0 \\ 0 & \text{tr}\ Y'(V - B(X))Y \end{bmatrix}. \tag{3.18}$$

Thus $(U - C(\theta))\theta = 0$. □

Thus each stationary point of σ gives a stationary point of $\tilde{\sigma}$. The other way around, however, we are not so lucky.

Theorem 3.3. *If $C(\theta)\theta = U\theta$ and if the $n \times 2p$ matrix $\begin{bmatrix} X & Y \end{bmatrix}$ has rank $n-1$ then $B(Z)Z = VZ$.*

Proof. If $C(\theta)\theta = U\theta$ then both tr $X'(V - B(Z)Z) = 0$ and tr $Y'(V - B(Z))Z = 0$. If the $n \times 2p$ matrix $\begin{bmatrix} X & Y \end{bmatrix}$ has rank $n-1$ then this implies $(V - B(Z))Z = 0$. □

In our example the singular values of $\begin{bmatrix} X & Y \end{bmatrix}$ are 0.4879059, 0.4333287, 0.2242285, $1.4867245 \times 10^{-17}$ and thus there is a one-one correspondence between stationary points of σ and $\tilde{\sigma}$.

Theorem 3.4. *If $B(X)X = VX$ and $\theta = \begin{bmatrix} 1 & 0 \end{bmatrix}$ then*

1. *If tr $Y'(V - B(X))Y > 0$ then σ has a local minimum at theta.*
2. *If σ has a saddle point at θ then tr $Y'(V - B(X))Y < 0$.*

Proof. Suppose $B(X)X = VX$ and $\theta = \begin{bmatrix} 1 & 0 \end{bmatrix}$. Then, from (3.17) and (3.18),

$$\nabla^2\sigma(\theta) = \begin{bmatrix} 0 & 0 \\ 0 & \text{tr } Y'(V - B(X))Y \end{bmatrix} + \sum_{1 \leq i < j \leq n} \sum w_{ij} \frac{\delta_{ij}}{d_{ij}(\theta)} \frac{T_{ij}\theta\theta' T_{ij}}{\theta' T_{ij}\theta}.$$

□

In our example tr $X'(V - B(Y))X$ is -0.1502768 and tr $Y'(V - B(X))Y$ is -0.0533599.

3.6.2 Global Perspective

We first make a global perspective plot, over the range $(-2.5, +2.5)$.

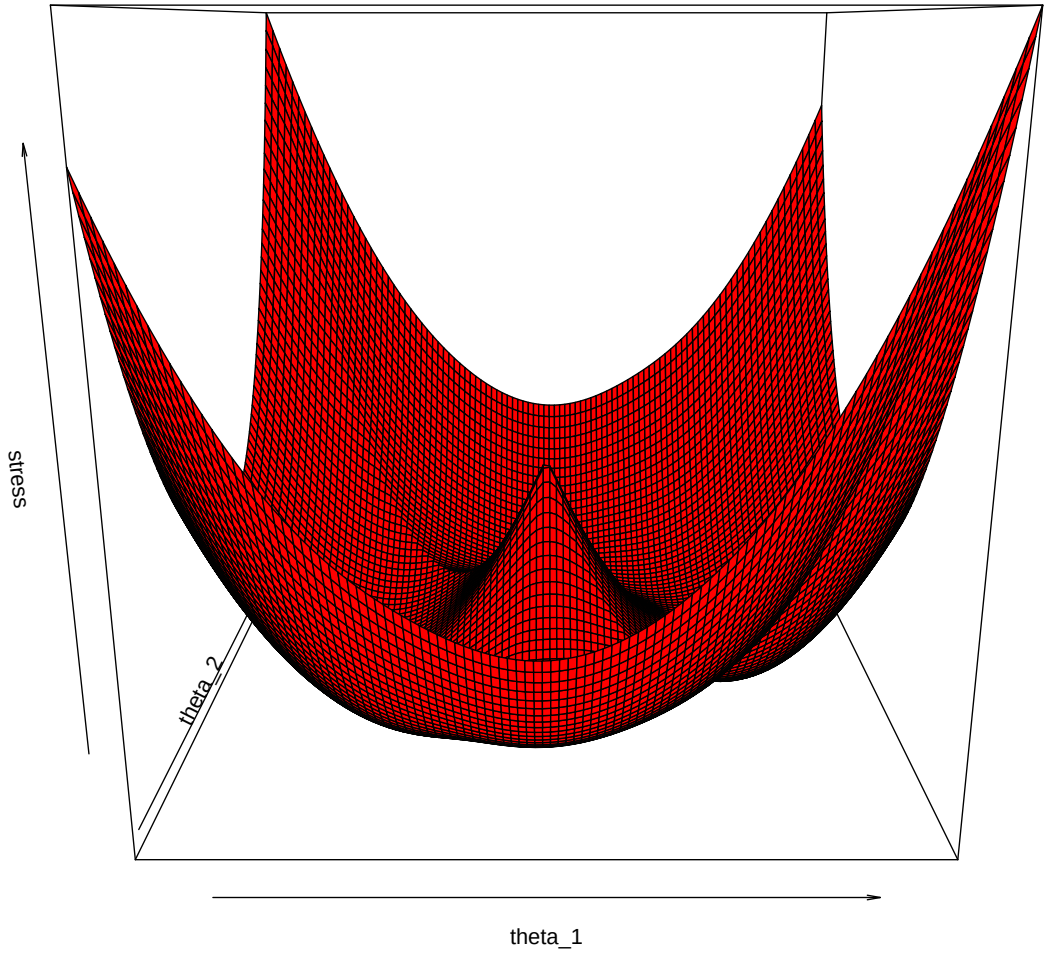


Figure 3.2: Global Perspective

We see the symmetry, following from the fact that stress is even. We also see the local maximum at the origin, where stress is not differentiable. Also note the ridge, where $d_{12}(\theta) = 0$ and where stress is not differentiable either. The ridge shows nicely that on rays emanating from the origin stress is a convex quadratic. Also, far away from the origin, stress globally behaves very much like a convex quadratic (except for the ridge). Clearly local minima must be

found in the valleys surrounding the small mountain at the origin, all within the sphere with radius $\sqrt{2}$.

3.6.3 Global Contour

Figure 3.2 is a contour plot of stress over $(-2, +2) \otimes (-2, +2)$. The red line is $\{\theta \mid d_{12}(\theta) = 0\}$. The blue line has the minimum of the convex quadratic on each of the rays through the origin. Thus all local minima, and in fact all stationary points, are on the blue line. Note that the plot uses θ to define the coordinate axes, not $\gamma = (\alpha, \beta)$. Thus there are no stationary points at $(0, 1)$ and $(1, 0)$, but at the corresponding points (1.3938469, 0) and (1.0406404, 0.8849253) in the θ coordinates (and, of course, at their mirror images).

Besides the single local maximum at the origin, it turns out that in this example there are five pairs of stationary points. Or, more precisely, I have not been able to find more than five. Each stationary point θ has a mirror image $-\theta$. Three of the five are local minima, two are saddle points. Local minima are plotted as blue points, saddle points as red points.

3.6.4 Stationary Points

3.6.4.1 First Minimum

We zoom in on the first local minimum at (1.0406404, 0.8849253). Its stress is 0.0669873, and the corresponding configuration has three points in the corners of an equilateral triangle and the fourth point in its centroid. Note that this local minimum is a saddle point in configuration space $\mathbb{R}^{4\times 2}$ (Trosset and Mathar (1997)). The eigenvalues of $B(\theta)$ are (1.3686346, 1) and those of the Hessian $I - H(\theta)$ are (1, 0.0817218). The area of the contour plot around the stationary value is in figure 3.4.

3.6.5 Second Minimum

The second local minimum (which is the global minimum) at (1.3938469, 0) has stress 0.0285955. The configuration are the four points at the corners of a square. The eigenvalues of $B(\theta)$ are (1.1362799, 1) and those of the

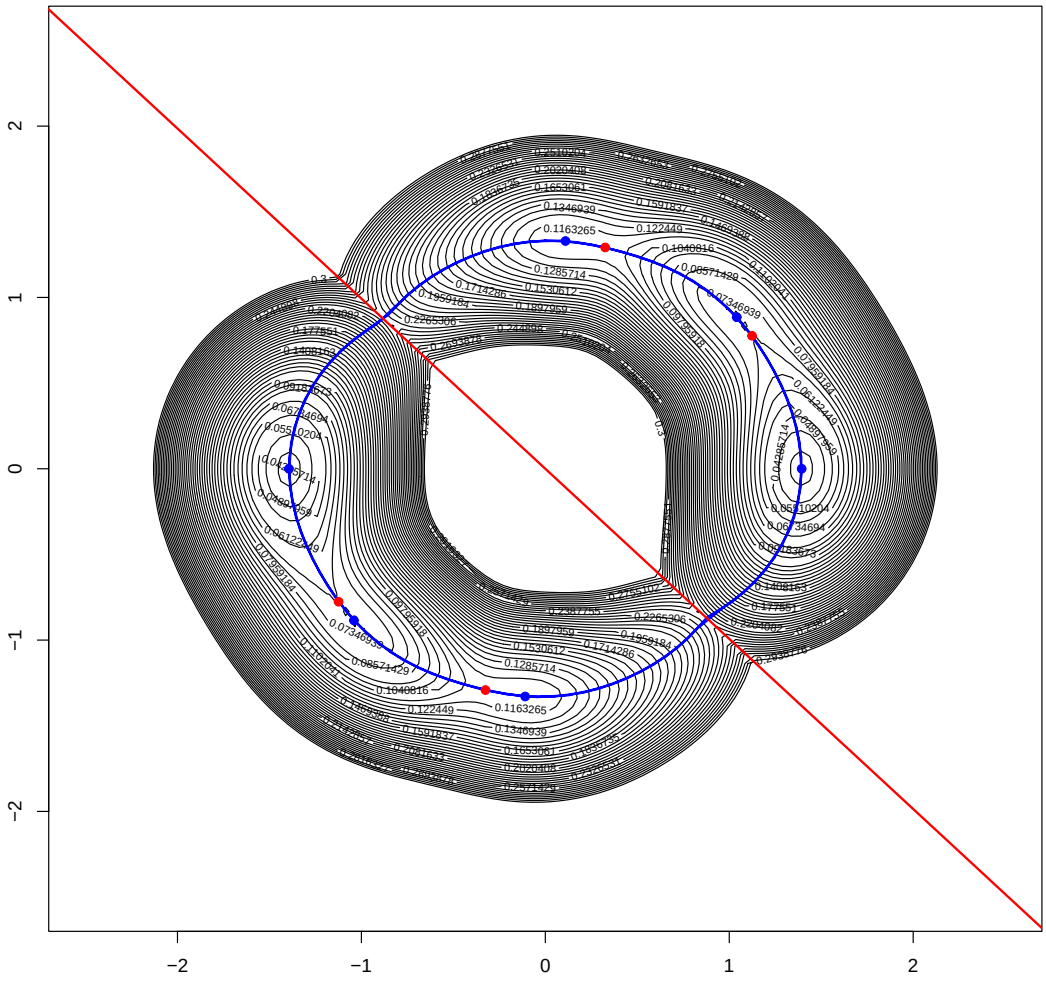


Figure 3.3: Global Contour

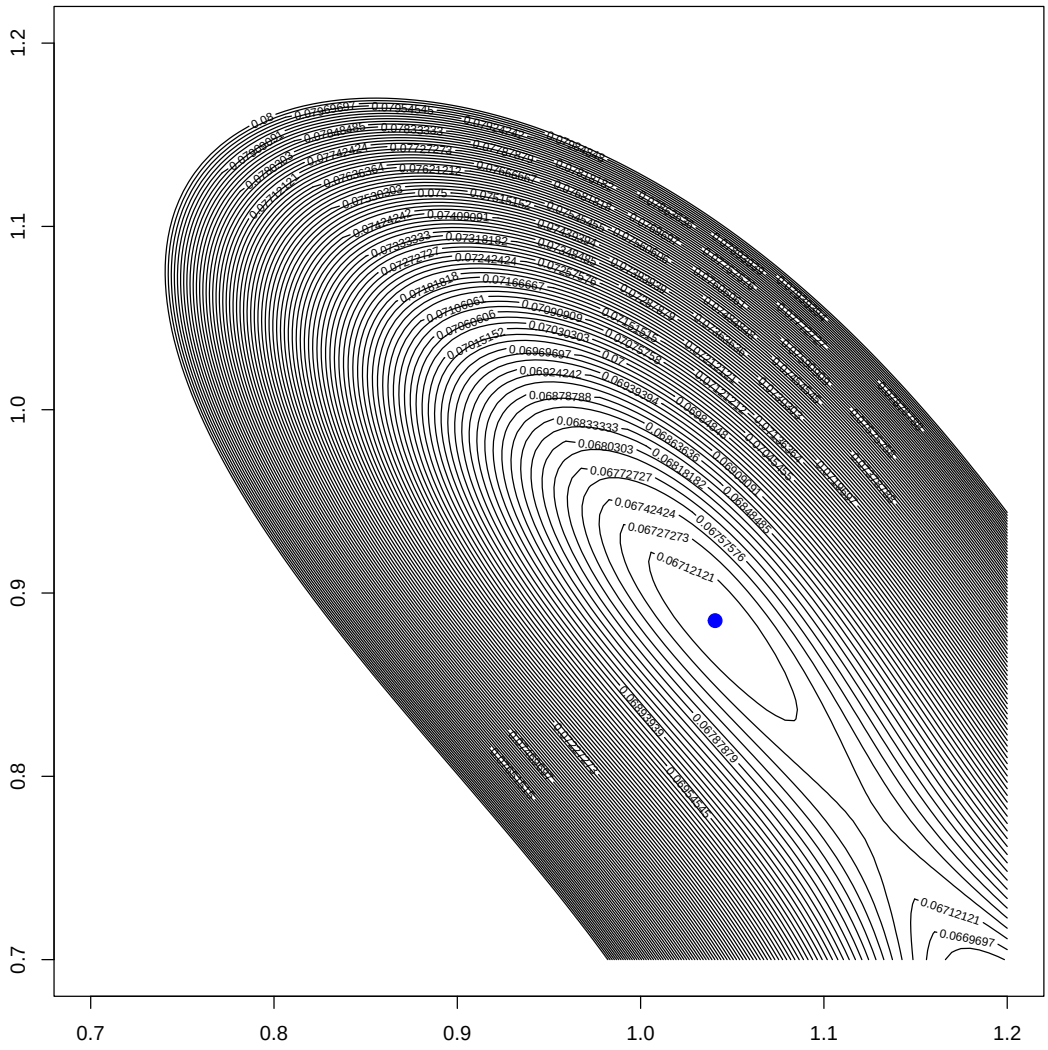


Figure 3.4: Contour Plot First Minimum

Hessian $I - H(\theta)$ are (1, 0.3743105). The area of the contour plot around the stationary value is in figure 3.5.

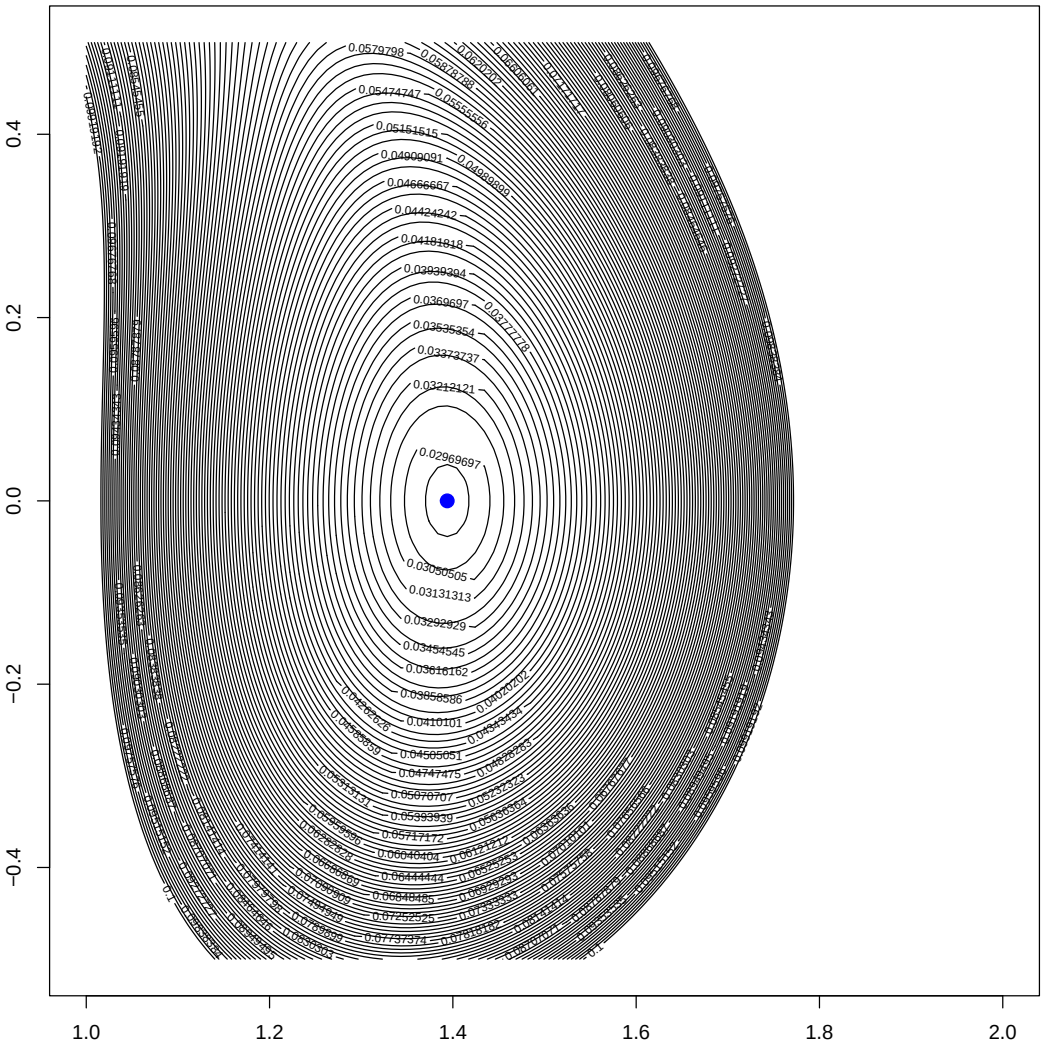


Figure 3.5: Contour Plot Second Minimum

3.6.6 Third Minimum

The third local minimum at (0.1096253, 1.3291942) has stress 0.1106125, and the corresponding configuration is in figure 3.6.

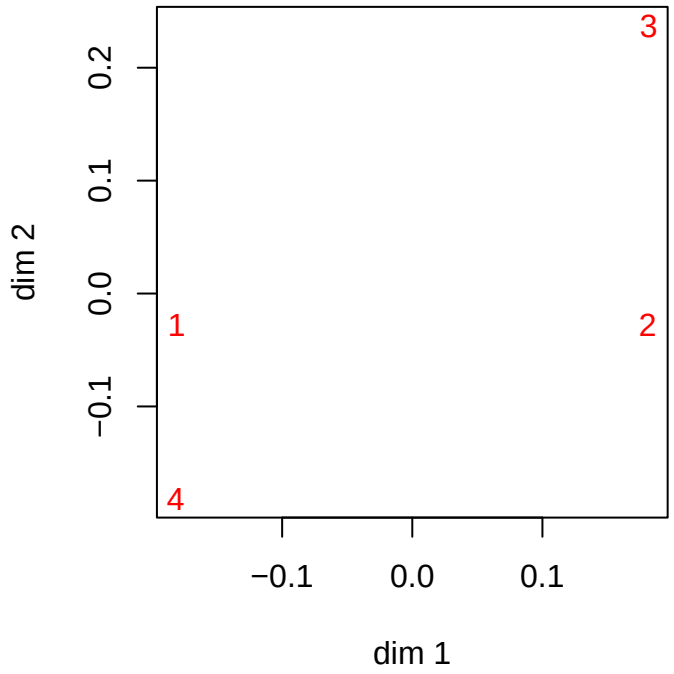


Figure 3.6: Configuration Third Minimum

The eigenvalues of $B(\theta)$ are (1.5279386, 1) and those of the Hessian $I - H(\theta)$ are (1, 0.2362079). The area of the contour plot around the stationary value is in figure 3.7

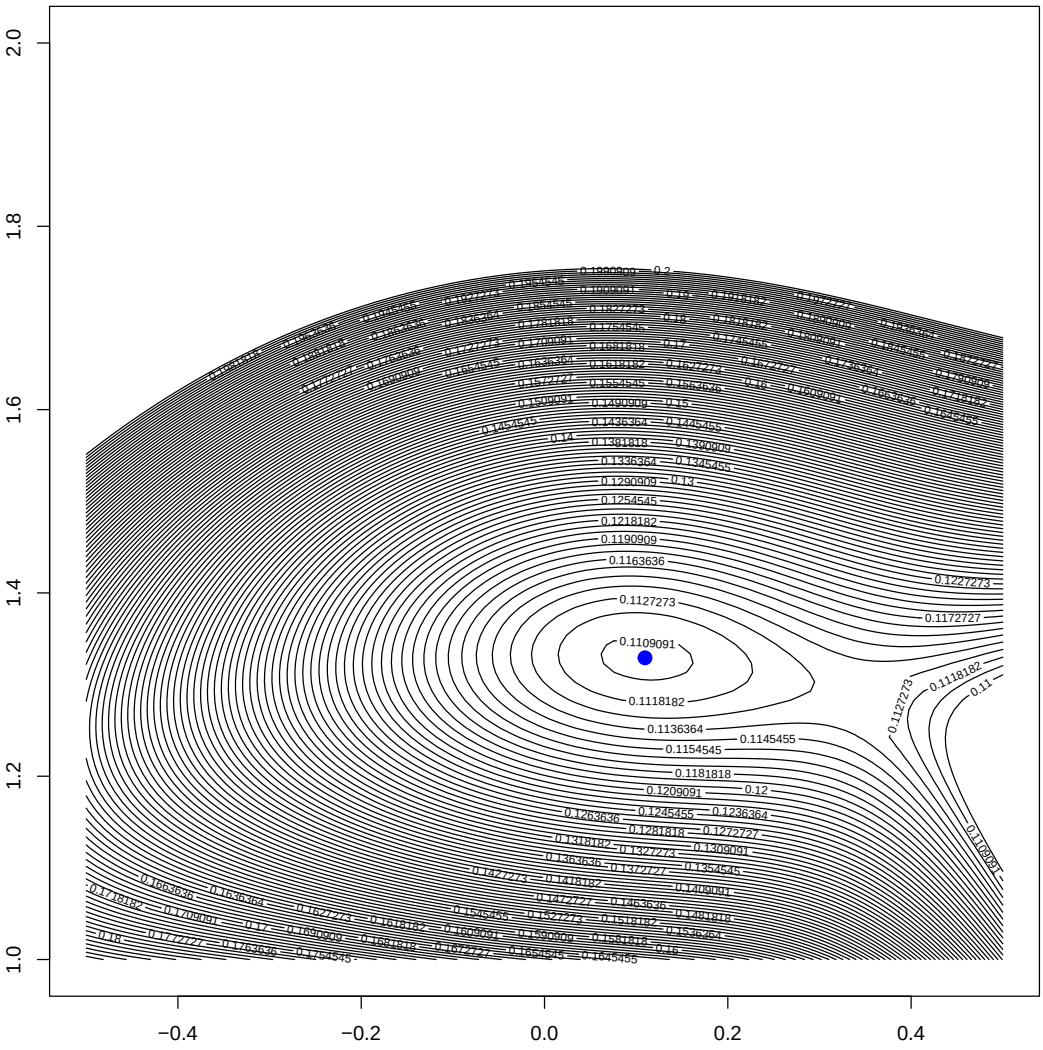


Figure 3.7: Contour Plot Third Minimum

3.6.7 First Saddle Point

The saddle point at (0.3253284, 1.2916758) has stress 0.1128675, and the corresponding configuration is in figure 3.8.

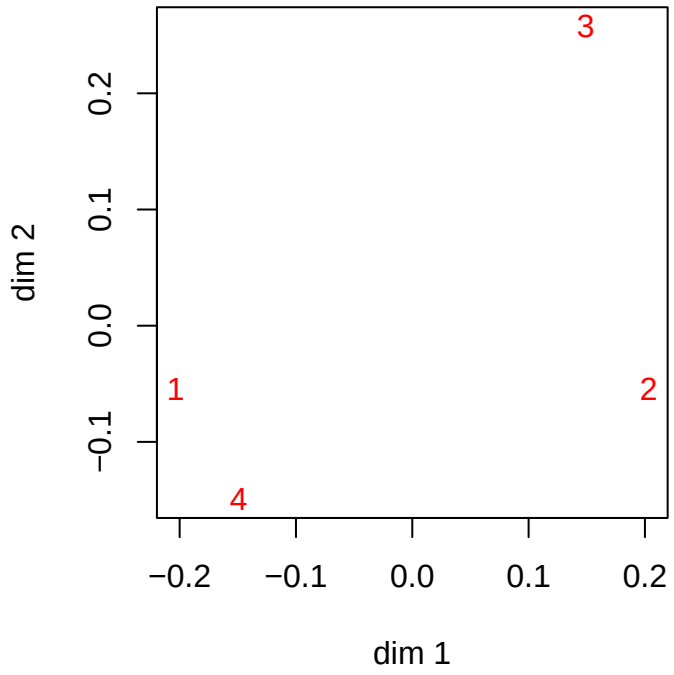


Figure 3.8: Configuration First Saddlepoint

The eigenvalues of $B(\theta)$ are (1.7778549, 1) and those of the Hessian $I - H(\theta)$ are (1, -0.311088). The area of the contour plot around the stationary value is in figure 3.9

3.6.8 Second Saddle Point

The saddle point at (1.1238371, 0.7762046) has stress 0.0672483 and the corresponding configuration is in figure 3.10

The eigenvalues of $B(\theta)$ are (1.4111962, 1) and those of the Hessian $I - H(\theta)$ are (1, -0.0841169). The area of the contour plot around the stationary value is in figure 3.11

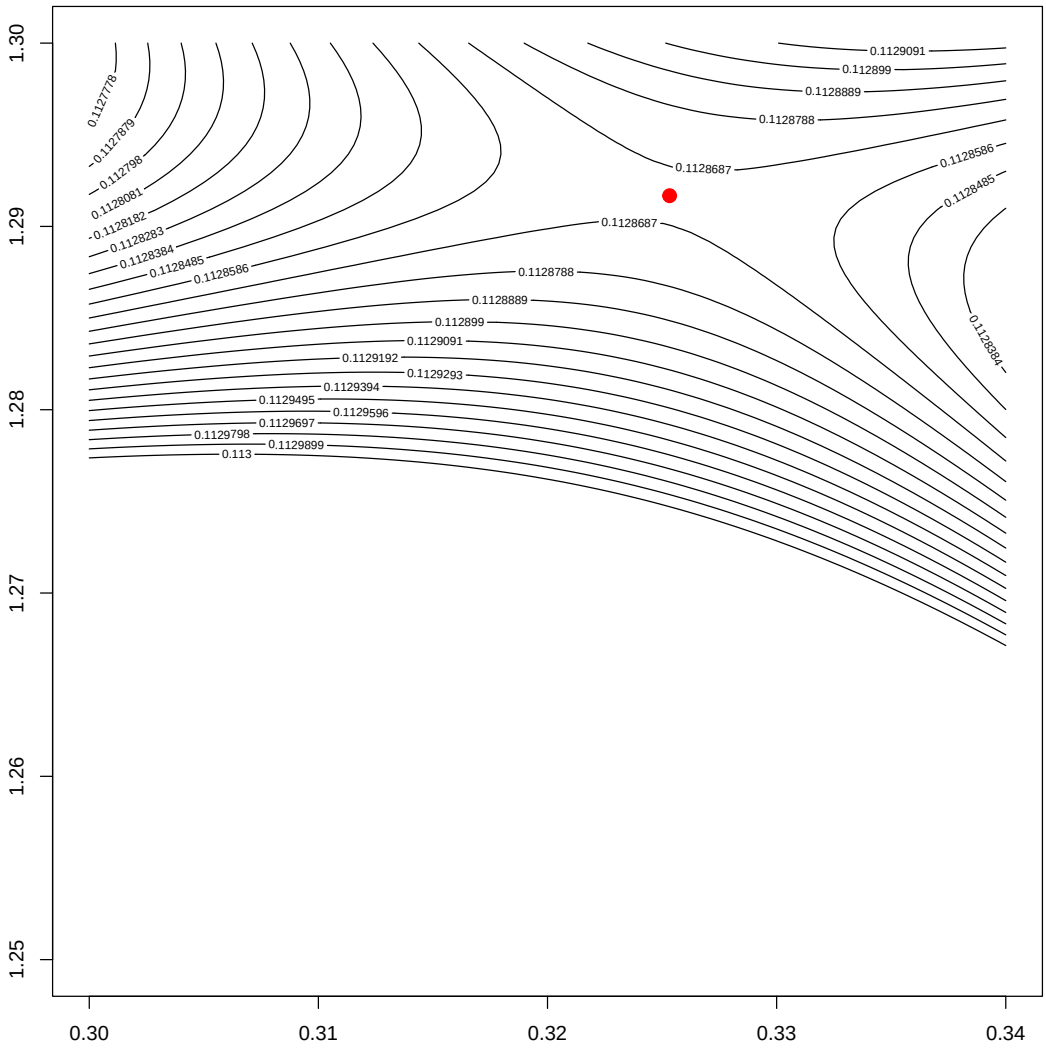


Figure 3.9: Contour First Saddlepoint

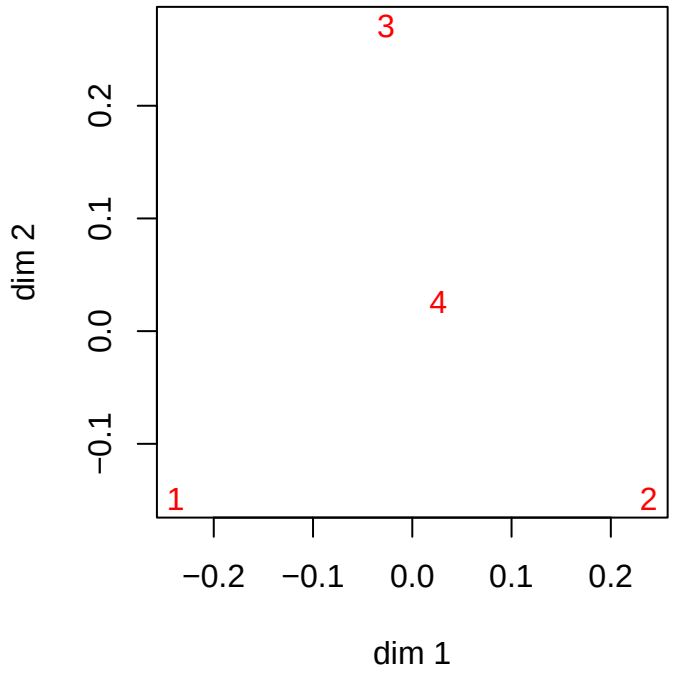


Figure 3.10: Configuration Second Saddlepoint

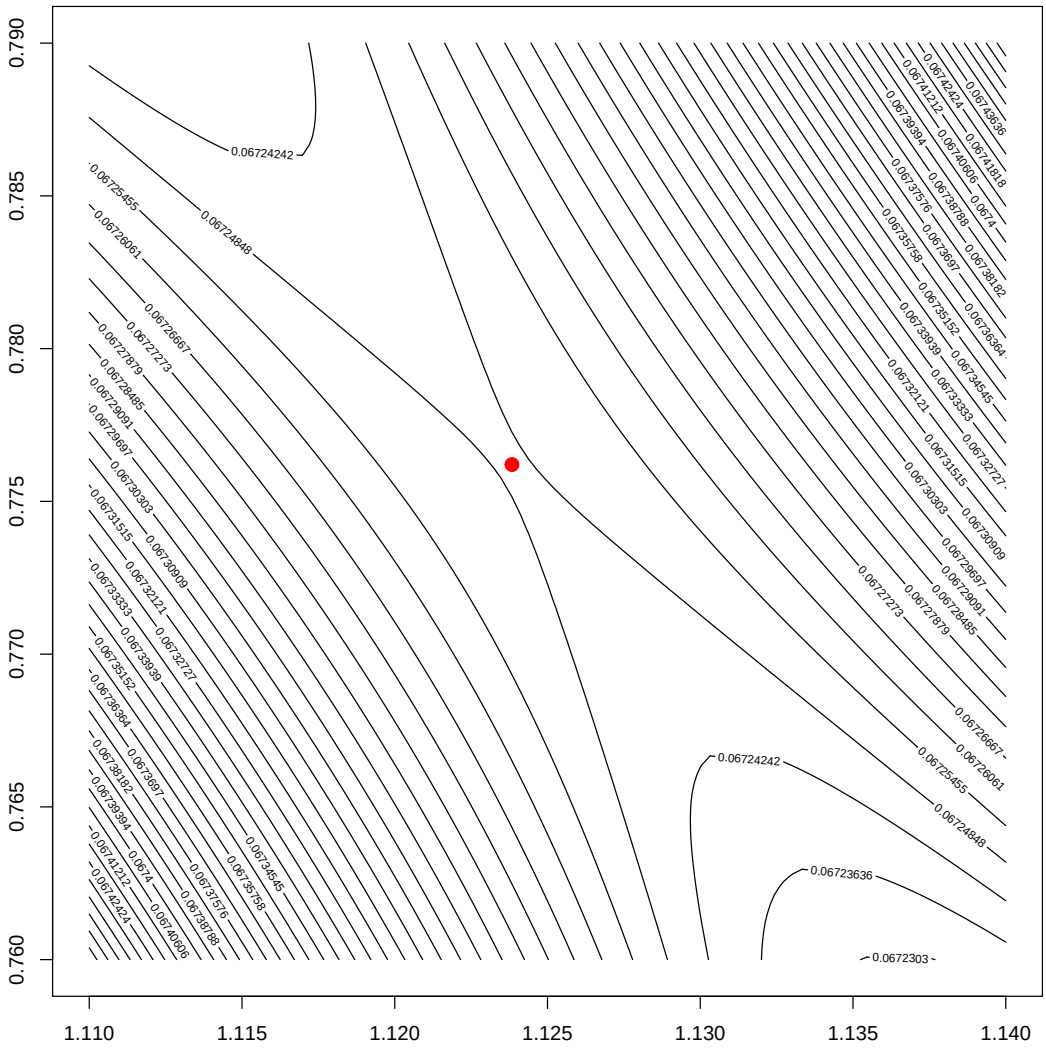


Figure 3.11: Contour Plot Second Saddlepoint

3.7 Another Look

Remember that $\rho(\theta) = \theta' B(\theta)\theta$. Thus $\sigma(\lambda\theta) = 1 - \lambda\rho(\theta) + \frac{1}{2}\lambda^2\theta'\theta$, and

$$\min_\lambda \sigma(\lambda\theta) = 1 - \frac{1}{2}\frac{\rho^2(\theta)}{\theta'\theta}.$$

Thus we can minimize σ over θ by maximizing ρ over the unit circle $\mathcal{S} := \{\theta \mid \theta'\theta = 1\}$. This is a nice formulation, because ρ is norm, i.e. a homogeneous convex function of θ. Consequently we have transformed the problem from unconstrained minimization of the DC function (i.e. difference of convex functions) stress to that of maximization of a ratio of norms. In turn this is equivalent to maximization of the convex function ρ over the unit circle, or, again equivalently, over the unit ball, a compact convex set. This transform was first used in MDS by De Leeuw (1977a), partly because it made the theory developed by Robert (1967) available.

The levels sets $\{\theta \mid \rho(\theta) = \kappa\}$ are the ρ-circles defined by the norm ρ. The corresponding ρ-balls $\{\theta \mid \rho(\theta) \leq \kappa\}$ are closed and nested convex sets containing the origin. Thus we want to find the largest ρ-circle that still intersects $\mathcal{S}$. The similarity with the geometry of eigenvalue problems is obvious.

In our example we know that the global optimum of stress is at (1.3938469, 0), and if we project that point on the circle it becomes (1, 0). The corresponding optimal ρ is 1.3938469. Figure 3.12 gives the contourplot for ρ, with the outer ρ-circle corresponding with the optimal value. The fact that the optimal value contour is disjoint from the interior of $\mathcal{S}$ is necessary and sufficient for global optimality (Dür, Horst, and Locatelli (1998)). Notice the sharp corners in the contour plot, showing the non-diffentiability of ρ at the points where $d_{12}(\theta) = 0$. We could also look for the minimum of ρ on the unit circle, which means finding the largest ρ-circle that touches $\mathcal{S}$ on the inside. Inspecting figure 3.12 shows that this will be a point where ρ is not differentiable, i.e. a point with $d_{12}(\theta) = 0$. This minimum ρ problem does not make much sense in the context of multidimensional scaling, however, and it not related directly to the minimization of stress.

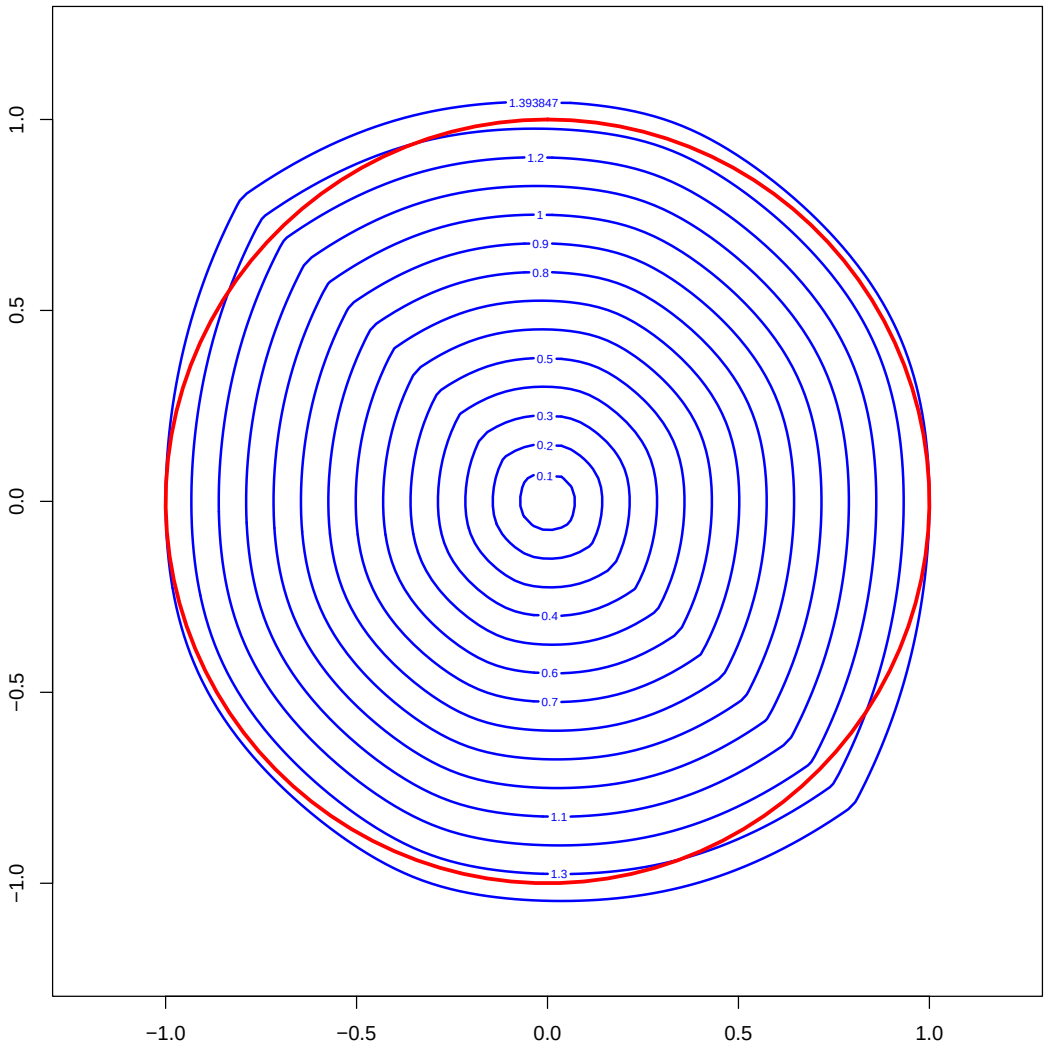


Figure 3.12: Contour Plot for Rho

3.8 A Final Look

Now that we know that the MDS problem is equivalent to maximizing ρ on the unit circle, we can use nonlinear coordinates $(\theta_1, \theta_2) = (\sin\xi, \cos\xi)$ to reduce the problem to a one-dimensional unconstrained one in, say, "circle space''. Thus, with the same abuse of notation as for stress, $\rho(\xi) := \rho(\sin\xi, \cos\xi)$, and we have to maximize ρ over $0 \leq \xi \leq \pi$.

In figure 3.13 we have plotted ρ as a function of η. There are blue vertical lines at the three local minima in coefficient space, red vertical lines at the stationary points, and a green vertical line where $d_{12}(\xi) = 0$. Note that in circle space stress has both multiple local minima and multiple local maxima.

From lemma xxx we see that the second derivative $\mathcal{D}^2\rho(\xi)$ is equal to $\mathbf{tr}\ H(\xi) - \rho(\xi)$. For the three local minima in coordinate space we find second derivatives 0, 0, 0 in circle space, i.e. they are properly converted to local maxima. The two stationary points in coordinate space have second derivatives 0, 0, and are turned into local minima.

For more general cases, with a basis of n configurations, we know from Lyusternik and Schnirelmann (1934) that a continuously differentiable even function on the unit sphere in $\mathbb{R}^n$ has at least n distinct pairs of stationary points.

3.9 Discussion

Note that we have used σ for three different functions. The first one with argument Z is defined on *configuration space*, the second one with argument γ on *coefficient space*, and the third one with argument θ also on *coefficient space*. This is a slight abuse of notation, rather innocuous, but we have to keep it in mind.

From lemma xxx we see that $\mathcal{D}\sigma(X) = \mathcal{D}\sigma(Y) = 0$ then $\mathcal{D}\sigma(1,0) = \mathcal{D}\sigma(0,1) = 0$. Thus stationary points in configuration space are preserved as stationary points in coefficient space, but the reverse implication may not be true. If $\mathcal{D}^2\sigma(X)$ and $\mathcal{D}^2\sigma(Y)$ are positive semi-definite, then so are $\mathcal{D}^2\sigma(1,0)$ and $\mathcal{D}^2\sigma(0,1)$. Thus local minima are preserved. But it is entirely possible that $\mathcal{D}^2\sigma(X)$ and/or $\mathcal{D}^2\sigma(Y)$ are indefinite, and that

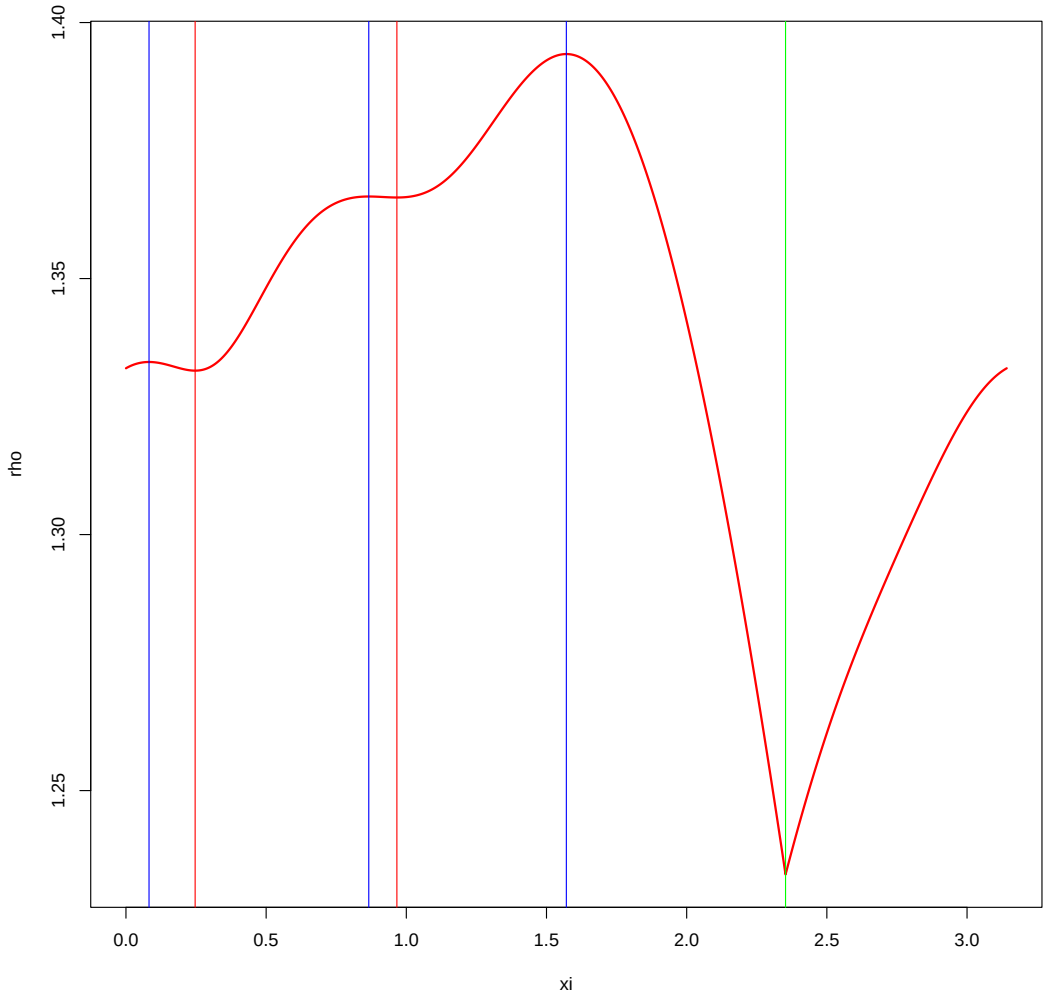
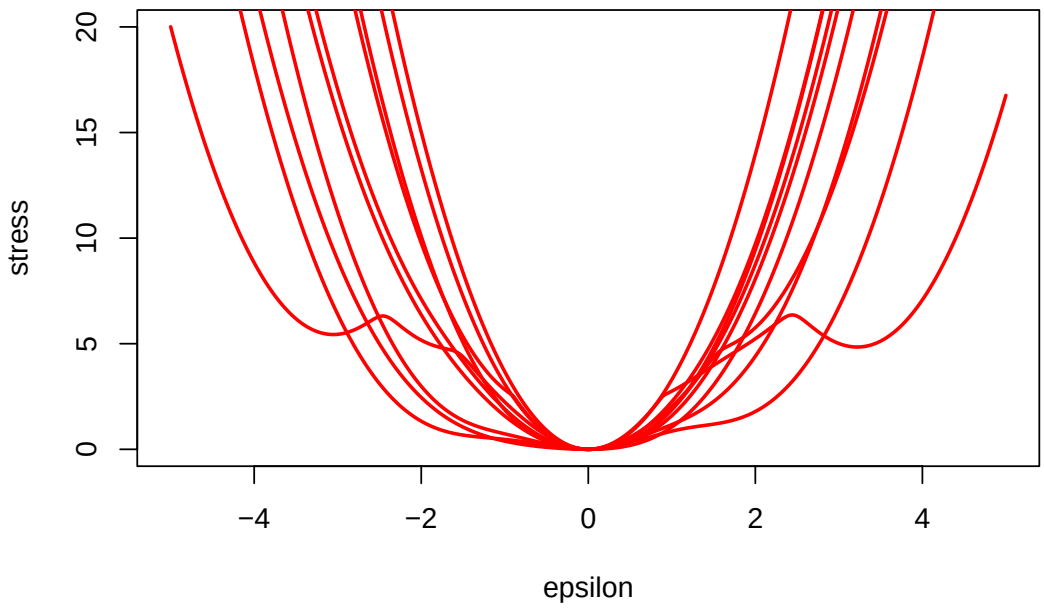
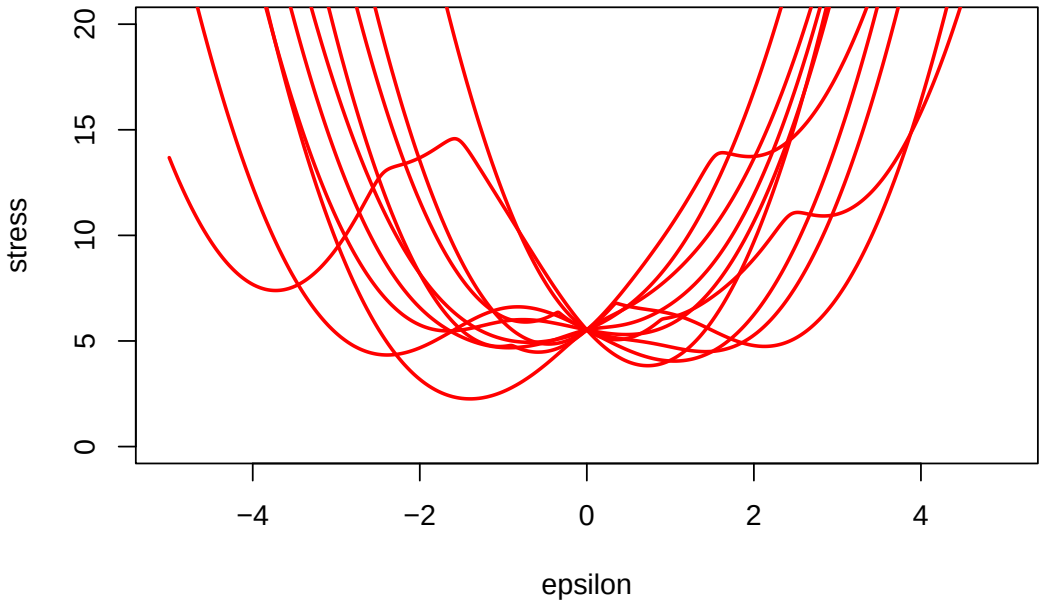


Figure 3.13: One-dimensional Rho

$\mathcal{D}^2\sigma(1,0)$ and/or $\mathcal{D}^2\sigma(0,1)$ are positive semi-definite. Thus saddle points in configuration space can be mapped into local minima in coefficient space. As we will see this actually happens with Y, the equilateral triangle with center, in our example.

3.10 Coordinates





Let's look at a small example with four points, all dissimilarities equal, and all weights equal to one. There is a local minimum with four points in the corners of a square, with stress equal to 0.0285955. And there is another local minimum with three points forming an equilateral triangle, and the fourth point in the center. This has stress 0.0669873. We can compute stress for all points of the form $\alpha X + \beta Y$, where X and Y are the two local minima. Figure 3.14 has a contour plot of $\sigma(\alpha, \beta)$, showing the local minima at $(1, 0)$ and $(0, 1)$, and the local maximum at $(0, 0)$.

Alternatively, we can plot stress on the line connecting X and Y. Note that although stress only has a local maximum at the origin in configuration space, it can very well have local maxima if restricted to lines. In fact on a line connecting two local minima there has to be at least one local maximum.

A fundamental result, which forms the basis of chapter 13 in this book, is that σ is convex on *Gramian Space*, i.e. on the closed convex cone of all $C \gtrsim 0$.

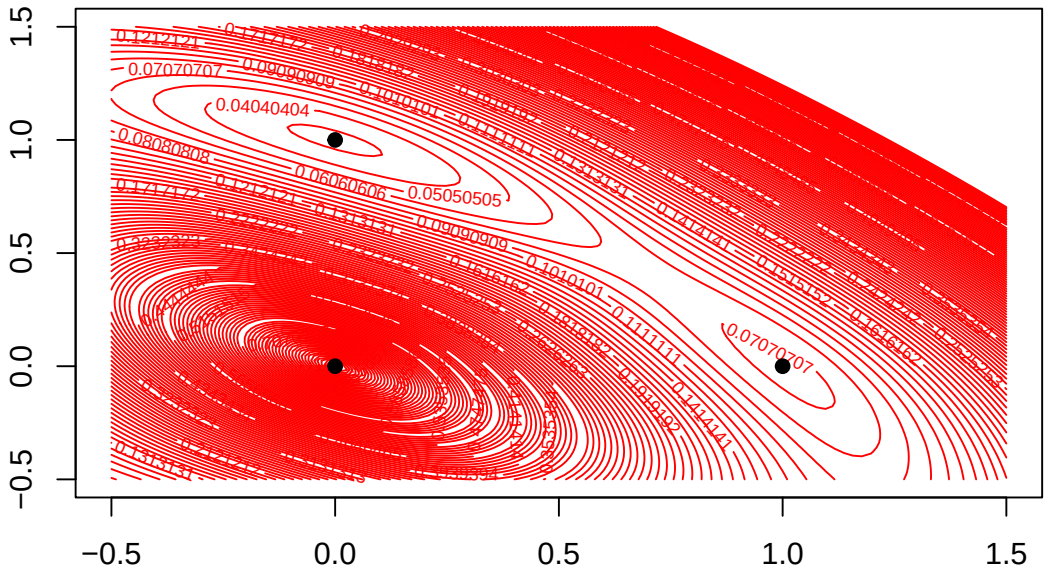


Figure 3.14: Plane spanned by two local minima, equal dissimilarities

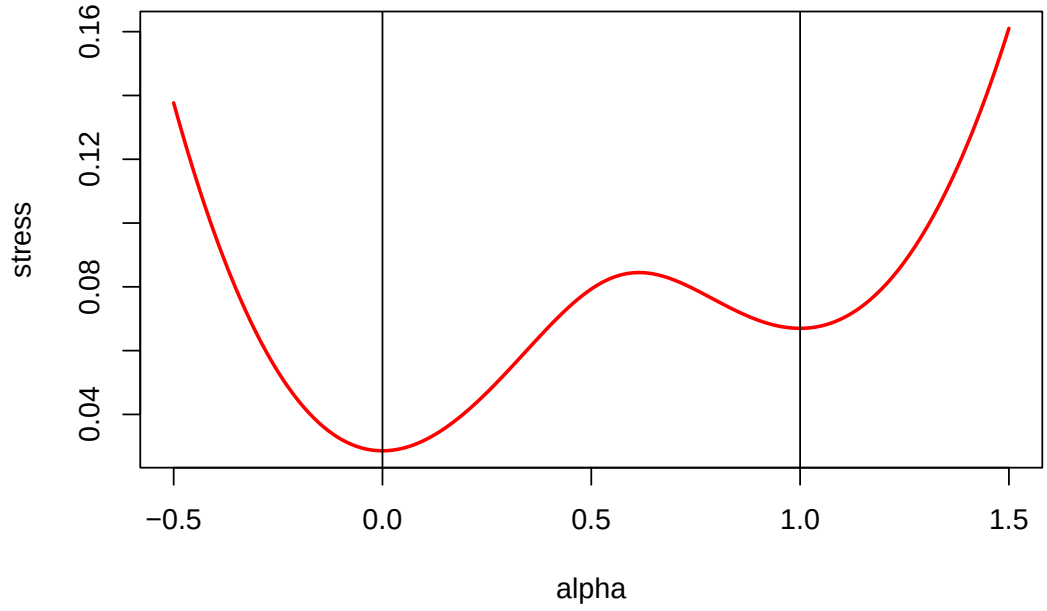


Figure 3.15: Line connecting two local minima, equal dissimilarities

Chapter 4

Classical Multidimensional Scaling

In the early days, exemplified by Messick and Abelson (1956), the key mathematical result used in MDS was Schoenberg's theorem (Schoenberg (1935)), which was made available to psychometricians by G. Young and Householder (1938). Statisticians came to the scene much later, not until Gower (1966).

Schoenberg's theorem gives necessary and sufficient conditions for a symmetric, hollow, and non-negative matrix of dissimilarities to be the distance matrix of n points in $\mathbb{R}^p$. Several variations of this basic theorem are possible (see Blumenthal (1953), chapter 43). We give the result in the form popularized by W. S. Torgerson (1958), with a simplified proof, using notation and terminology due to Critchley (1988).

Classical scaling can be thought of as an independent MDS method, in fact as the only MDS method available until the early 1960's. More commonly these days it is a method to provide a generally excellent starting point for iterative procedures to minimize stress.

4.1 Algebra

4.1.1 Torgerson Transform

The *Torgerson transform* of a matrix is a linear function from the space of real symmetric hollow matrices to the space of doubly-centered real symmetric matrices defined as

$$\tau(S) := -\frac{1}{2}JSJ, \tag{4.1}$$

with J the centering matrix $I - \frac{1}{n}ee'$. For historical reasons @(eq:torgerson) may be more familiar in elementwise notation. Spelled out it is

$$\tau_{ij}(S) = -\frac{1}{2}\{s_{ij} - s_{i\bullet} - s_{\bullet j} + s_{\bullet\bullet}\}, \tag{4.2}$$

where bullets are indices over which we average. The symbol τ was chosen by Critchley (1988) to honor Torgerson.

Some simple calculation with (4.2), using the hollowness of S, gives

$$\tau_{ii}(S) + \tau_{jj}(S) - 2\tau_{ij}(S) = s_{ij}. \tag{4.3}$$

Accordingly, Critchley (1988) defines a linear operator κ on the space of real symmetric matrices by $\kappa(S) := s_{ii} + s_{jj} - 2s_{ij}$. From (4.3) we see that $\kappa(\tau(S)) = S$. Also for all double-centered S we have $\tau(\kappa(S)) = S$. Thus τ and κ are mutually inverse (Critchley (1988), theorem 2.2).

4.1.2 Schoenberg's Theorem

Theorem 4.1. *There is an $X \in \mathbb{R}^{n\times p}$ such that $\Delta = D(X)$ if and only if $\tau(\Delta^{(2)})$ is positive semi-definite of rank $r \leq p$.*

Proof. If Δ are the distances of a column-centered p-dimensional configuration X, then the squared dissimilarities $\Delta^{(2)}$ are of the form $ue' + eu' - 2XX'$, where $u_i = x_i'x_i$ are the squared row lengths. This implies $\tau(\Delta^{(2)}) = XX'$, which is positive semi-definite of rank $r = \text{rank}(X) \leq p$.

Conversely, suppose $\tau(\Delta^{(2)}) = XX'$. Applying κ to both sides and using $\kappa(XX') = D^{(2)}(X)$ gives $\Delta^{(2)} = D^{(2)}(X)$. □

Accordingly, we call a dissimilarity matrix Δ *Euclidean* if it is symmetric, hollow, and non-negative and has $\tau(\Delta^{(2)}) \gtrsim 0$. The *dimension* of a Euclidean dissimilarity matrix is the rank of $\tau(\Delta^{(2)})$, which is always less than or equal to $n - 1$.

4.2 Approximation

4.2.1 Low Rank Approximation of the Torgersen Transform

In classical MDS we minimize

$$\sigma(X) = \text{tr } \left\{W(\tau(\Delta^{(2)}) - XX')W\right\}^2 \tag{4.4}$$

over $X \in \mathbb{R}^{n\times p}$, where W is a square non-singular matrix of weights. The minimizer X for the loss function in (4.4) is given by a slight variation of Eckart and Young (1936), discussed first by Keller (1962). See also De Leeuw (1974) and the generalizations to rectangular and singular weight matrices in De Leeuw (1984d). The solution is $X = W^{-1}K\Lambda$, where K are the normalized eigenvectors corresponding with the p largest eigenvalues of the positive part of $W(\tau(\Delta^{(2)})W$ (i.e. the matrix that results by replacing the negative eigenvalues with zeroes). Thus the rank of the solution X is equal to the minimum of p and the number of positive eigenvalues of $\tau(\Delta^{(2)}$.

In the original versions of classical scaling (W. S. Torgerson (1952), W. S. Torgerson (1958)) there are no weights and the problem that the Torgerson transform may be indefinite is ignored. In fact, going from $\tau(\Delta^{(2)}$ to X is done by "any method of factoring", including the centroid method, so no specific loss function is minimized.

The loss function (4.4) has been called, somewhat jokingly, *strain* by Takane, Young, and De Leeuw (1977), mainly because *stress* and *sstress* had already been taken. Stress is a weighted least squares loss function defined on the distances, sstress on the squared distances, and strain on the inner products.

4.2.2 Minimization of Strain

It may be considered a disadvantage of the classical approach, even if it is described as the minimization of strain, that the MDS equations are $\Delta^{(2)} = D^{(2)(X)}$ while loss is measured on the scalar products and not the distances or the squared distances.

It was first pointed out by De Leeuw and Heiser (1982) that strain can also be written as a loss function defined on the squared distances. In fact strain is equal to

$$\sigma(X) = \frac{1}{4}\text{tr}\ \left\{WJ(\Delta^{(2)} - D^{(2)}(X))JW\right\}^2, \tag{4.5}$$

i.e. to an appropriately weighted version of sstress.

An advantage of the classical scaling approach via minimizing strain is that there is no local minimum problem. Finding the optimal configuration is an eigenvalue problem, which allows us to find the global minimum. This has not been emphasized enough, so I'll emphasize it here once again. If iterative basic MDS algorithms use the classical minimum strain solution as their starting point, then they start at the global minimum of a related loss function. Since they are descent algorithms they will improve their own loss functions, but having such an excellent starting point means they avoid many local minima.

Another advantage of the loss function formulation in @ref{eq:strainer} is that it is immediately obvious how to generalize classical scaling when there are missing data or when there is only rank order information. As with stress and sstress we minimize strain over both the configuration X and the missing information.

4.2.3 Approximation from Below

Suppose the Torgerson transform $\tau(\Delta^{(2)}$ is PSD of rank r, with eigen decomposition $K\Lambda^2K'$, and, using the largest p eigenvalues with corresponding eigenvectors, define $C_p := K_p\Lambda_p^2K_p'$. Then, in the Loewner order,

$$C_1 \lesssim C_2 \lesssim \cdots \lesssim C_r = \tau(\Delta^{(2)}).$$

Define $X_p = K_p\Lambda_p$. Then applying Critchley's inverse Torgerson transform κ it follows from ... that elementwise

$$D^{(2)}(X_1) \leq D^{(2)}(X_2) \leq \cdots \leq D^{(2)}(X_r) = \Delta^{(2)},$$

and thus also

$$D(X_1) \leq D(X_2) \leq \cdots \leq D(X_r) = \Delta.$$

Thus in the PSD case classical scaling we approximate the dissimilarities *from below*. This result is implicit in Gower (1966) and explicit in De Leeuw and Meulman (1986) and Meulman (1986).

If $\tau(\Delta^{(2)}$ is not psd then

$$D(X_1) \leq D(X_2) \leq \cdots \leq D(X_p) = \cdots = D(X_{n-1}) \geq \Delta.$$

Benzecri plot

Chapter 5

Minimization of Basic Stress

5.1 Gradient Methods

The initial algorithms for nonmetric MDS Kruskal (1964b) and Guttman (1968) were gradient methods. Thus the gradient, or vector of partial derivatives, was computed in each iteration step, and a step was taken in the direction of the negative gradient (which is also known as the direction of steepest descent).

Informally, if f is differentiable at x then $f(x+h) \approx f(x) + h'\mathcal{D}f(x)$ and the direction h that minimizes the diferential (the second term) is $-\mathcal{D}f(x)/\|\mathcal{D}f(x)\|$, the normalized negative gradient. Although psychometricians had been in the business of minimizing least squares loss functions in the linear and bilinear case, this result for general nonlinear functions was new to them. And I, and probably many others, hungrily devoured the main optimization reference in Kruskal (1964b), which was the excellent early review by Spang (1962).

Kruskal's paper also presents an elaborate step-size procedure, to determine how far we go in the negative gradient direction. In the long and convoluted paper Guttman (1968) there is an important contribution to gradient methods in basic MDS. Let's ignore the complications arising from zero distances, which is after all what both Kruskal and Guttman do as well, and assume all distances at configuration X are positive. Then stress is differentiable at X, with gradient

$$\nabla\sigma(X) = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(\delta_{ij} - d_{ij}(X))\frac{1}{d_{ij}(X)}(e_i - e_j)(x_i - x_j)'$$

Geometrical interpretation - Gleason, Borg-Groenen p 20

Guttman C-matrix

Ramsay C-matrix

5.1.1 Step Size

5.1.1.1 Kruskal Step Size

elaborate

5.1.1.2 Guttman Step Size

constant

5.1.1.3 Cauchy Step Size

In the classical version of the steepest descent method we choose the step-size α by minimizing $h(\alpha) = f(x + \alpha y)$ over α

$$d_+h(\alpha;\beta) = \lim_{\epsilon\downarrow 0}\frac{f(x + (\alpha + \epsilon\beta)y) - f(x + \alpha y)}{\epsilon} = \beta\ d_+f(x + \alpha y; y)$$

or local minimum closest to zero

Newton version

$$d_+^2h(\alpha;\beta,\gamma) = \beta\gamma d_+^2f(x + \alpha y; y, y)$$

5.1.1.4 Majorization Step Size

Lagrange form of the remainder

$e(\epsilon) = \eta^2(X + \epsilon Y)$ $r(\epsilon) = \rho(X + \epsilon Y)$ $s(\epsilon) = \sigma(X + \epsilon Y) = 1 - r(\epsilon) + \frac{1}{2}e(\epsilon)$

$$r(\epsilon) \geq r(0) + r'(0)\epsilon$$

$$e(\epsilon) = e(0) + e'(0)\epsilon + \frac{1}{2}e''(0)\epsilon^2$$

$$s(\epsilon) \leq s(0) + s'(0)\epsilon + \frac{1}{2}e''(0)\epsilon^2$$

$$\hat{\epsilon} = \frac{s'(0)}{e''(0)}$$

underestimates newton step

$$r(\epsilon) \geq r(0) + \epsilon\, r'(0) + \frac{1}{2}\epsilon^2 \min_{0 \leq \xi \leq \epsilon} r''(\xi)$$

$$\hat{\epsilon} = \frac{s'(0)}{e''(0) - \min_{0 \leq \xi \leq \epsilon} r''(\xi)}$$

$$r''(\xi) = \mathcal{D}^2\sigma(X + \xi Y; Y, Y)$$

5.2 Initial Configurations

Random

L

Torgerson

Guttman

5.3 On MM Algorithms

The term *majorization* is used in mathematics in many different ways. In this book we use it as a general technique for the construction of *stable* iterative minimization algorithms. An iterative minimization algorithm is stable if it decreases the objective function in each iteration.

Ortega, Rheinboldt Weber Bohning, Lindsay Vosz, Eckart

Special cases of majorization had been around earlier, most notably the smacof algorithm for MDS and the EM algorithm for maximum likelihood with missing data, but in full generality majorization was first discussed in De Leeuw (1994) and Heiser (1995).

Majorization can be used to construct minimization methods, while minorization can construct maximization methods. This cleverly suggests to use the acronym MM algorithms for this class of techniques. An excellent comprehensive account of MM algorithms is Lange (2016 (in press)). Another such account is slowly growing in one of the companion volumes in this series of personal research histories (De Leeuw (2021)).

Here we just give a quick introduction to majorization. Suppose f is a real-valued function on $X \subseteq \mathbb{R}^n$. Then a real-valued function g on $X \otimes X$ is said to majorize f on X if

$$g(x,y) \geq f(x) \qquad \forall (x,y) \in X \otimes X, \tag{5.1}$$

and

$$g(y,y) = f(y) \qquad \forall y \in X. \tag{5.2}$$

Thus for each $y \in X$ the function $g(\bullet, y)$ lies above f, and it touches f from above at $x = y$. Majorization is *strict* if $g(x,y) = f(x)$ if and only if $x = y$, i.e. if y is the only point in X where $g(\bullet, y)$ and f touch.

A *majorization algorithm* to minimize f on X starts with an initial estimate, and then updates the estimate in iteration k by

$$x^{(k+1)} \in \operatorname*{argmin}_{x \in X} g(x, x^{(k)}), \tag{5.3}$$

with the understanding that the algorithms stops when $x^{(k)} \in \text{argmin}_{x \in X} g(x, x^{(k)})$.

If we do not stop we have an infinite sequence satisfying the *sandwich inequality*

$$f(x^{(k+1)}) \leq g(x^{(k+1)}, x^{(k)}) \leq g(x^{(k)}, x^{(k)}) = f(x^{(k)}). \tag{5.4}$$

The first inequality in this chain comes from (5.1). It is strict when majorization is strict. The second inequality comes from (5.3), and it is strict if $g(\bullet, y)$ has a unique minimum on X for each y.

Necessary conditions through MM

5.4 Smacof Algorithm

5.4.1 Guttman Transform

The Guttman Transform is named to honor the contribution of Louis Guttman to non-metric MDS (mainly, but by no means exclusively, in Guttman (1968)). Guttman introduced the transform in a slightly different way, and partly for different reasons. In chapter 5 we shall see that the Guttman Transform plays a major role in defining and understanding the smacof algorithm.

The *Guttman Transform* of a configuration X is defined as

$$\Gamma(X) := V^+ B(X) X, \tag{5.5}$$

which is simply equal to $\Gamma(X) = n^{-1} B(X) X$ if all weights are equal. For some purposes it is useful to think of $V^+ B(X) X$ as a function of the weights, the dissimilarities, and the configuration (see, for exam0ple, chapter 24), but we reserve the name Guttman transform for a map from $\mathbb{R}^{n \times p}$ into $\mathbb{R}^{n \times p}$.

What we have called $B(X)$ is what Guttman calls the *correction matrix* or *C-matrix* (see De Leeuw and Heiser (1977) for a comparison).

Completing the square in equation (2.19) gives

$$\sigma(X) = 1 + \eta^2(X - \Gamma(X)) - \eta^2(\Gamma(X)), \tag{5.6}$$

which shows that

$$1 - \eta^2(\Gamma(X)) \leq \sigma(X) \leq 1 + \eta^2(X - \Gamma(X)). \tag{5.7}$$

Note that it follows from (5.7) that $\sigma(X) \geq 1 - \eta^2(\Gamma(X))$, with equality if and only if $X = \Gamma(X)$.

Theorem 5.1. *The Guttman transform is*

- self-scaling *(a.k.a. homogeneous of degree zero) because* $\Gamma(\alpha X) = \Gamma(X)$ *for all* $-\infty < \alpha < +\infty$. *With our definition* (2.16) *of* $B(X)$ *we also have* $\Gamma(0) = 0$.
- self-centering*, because* $\Gamma(X + e\mu') = \Gamma(X)$ *for all* $\mu \in \mathbb{R}^p$.
- *Bounded*
- *Lipschitz*

::: { ,proof} We already know, from the CS inequality, that

$$\rho(X) \leq \eta(X). \tag{5.8}$$

With the Guttman transform in hand we can apply CS once more, and find

$$\rho(X) = \text{tr } X'B(X)X = \text{tr } X'V\Gamma(X) \leq \eta(X)\eta(\Gamma(X)) \tag{5.9}$$

Note that the Guttman transform is bounded. In fact, using the Euclidean norm throughout,

$$\Gamma(X) \leq \|V^+\|\|B(X)X\|$$

Now

$$B(X)X = \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}\frac{x_i - x_j}{d_{ij}(X)}(e_i - e_j),$$

and thus

$$\|B(X)X\| \leq \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}\left\|\frac{x_i - x_j}{d_{ij}(X)}\right\|\|e_i - e_j\| = \sqrt{2}\sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij},$$

and

$$\|\Gamma(X)\| \leq \sqrt{2}\|V^+\| \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}.$$

In fact

$$B(X)X - B(Y)Y = \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}\left\{\frac{x_i - x_j}{d_{ij}(X)} - \frac{y_i - y_j}{d_{ij}(Y)}\right\}(e_i - e_j),$$

and thus

$$\|\Gamma(X) - \Gamma(Y)\| \leq 2\|V^+\| \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij},$$

and thus the Guttman transform is Lipschitz. ::: ;’’’’ytyh ### Subdifferentials

5.4.2 Derivative

The basic smacof algorithm, which is the main building block for most of the MDS techniques discussed in this book, updates the configuration $X^{(k)}$ in iteration k by

$$X^{(k+1)} = \Gamma(X^{(k)}) = V^+B(X^{(k)})X^{(k)}. \tag{5.10}$$

so that first $X^{(1)} = \Gamma(X^{(0)})$, then $X^{(2)} = \Gamma(X^{(1)}) = \Gamma(\Gamma(X^{(0)}))$, and generally $X^{(k)} = \Gamma^k(X^0)$, where Γ^k is the k-times composition (or iteration) of Γ.

We shall show in this chapter that as $k \to \infty$ both $\sigma(X^{(k+1)}) - \sigma(X^{(k)}) \to 0$, and $\eta^2(X^{(k)} - X^{(k+1)}) = (X^{(k+1)} - X^{(k)})'V(X^{(k+1)} - X^{(k)}) \to 0$. The iterations stop either if $\sigma(X^{(k)}) - \sigma(X^{(k+1)}) < \epsilon$ or if $\eta^2(X^{(k)} - X^{(k+1)}) < \epsilon$, where the ϵ are small cut-off numbers chosen by the user, or if we reach a user-defined maximum number of iterations, and we give up on convergence. The user also chooses if the stop criterion is based on function value changes or configuration changes.

Some quick remarks on implementation. We only have to compute V^+ once, but premultiplying by a full symmetric matrix in each iteration does add quite a few multiplications to the algorithm. If all w_{ij} are one, then $V^+ = \frac{1}{n}J$ and thus $\Gamma(X^{(k)}) = \frac{1}{n}B(X^{(k)})X^{(k)}$. In fact we do not even have to carry out this division by n, because the basic algorithm is *self scaling*. which means in this context that $\Gamma(\alpha X) = \Gamma(X)$ for all $\alpha \geq 0$.

5.4.3 Global Convergence

The iterations in (5.10) start at some $X^{(0)}$ and then generate five sequences of non-negative numbers. Define $\lambda(X) := \rho(X)/\eta(X)$ and $\Gamma(X) := \eta^2(X - \Gamma(X))$. The five sequences we will look at are

$$\begin{aligned} \sigma_k &:= \sigma(X^{(k)}), \\ \rho_k &:= \rho(X^{(k)}), \\ \eta_k &:= \eta(X^{(k)}), \\ \lambda_k &:= \lambda(X^{(k)}), \\ \Gamma_k &:= \Gamma(X^{(k)}), \end{aligned} \tag{5.11}$$

Depend on $X^{(0)}$

Zangwill

Argyros

5.4.3.1 From the CS Inequality

Theorem 5.2.

1. *σ_k is a decreasing sequence, bounded below by 0.*
2. *ρ_k, η_k, amd λ_k are increasing sequences, bounded above by 1.*
3. *Γ_k is a null-sequence.*

To prove convergence of these sequences we slightly modify and extend the proofs in De Leeuw (1977a) and De Leeuw and Heiser (1977) (while I say to myself: that's 44 years ago).

Proof. 1. For each $X \in \mathbb{R}^{n\times p}$ we have $\rho(X) \leq \eta(X)$ and thus $\lambda(X) \leq 1$.

2. For each $X, Y \in \mathbb{R}^{n\times p}$ we have $\rho(X) \geq \text{tr } X'B(Y)Y$ and thus $\rho(X) \geq \text{tr } X'V\Gamma(Y)$. Taking $X = \Gamma(Y)$ we see that $\rho(X) \geq \eta^2(\Gamma(Y))$. Now $\sigma(\Gamma(Y)) = 1 - 2\rho(\Gamma(Y)) + \eta^2(\Gamma(Y)) \leq 1 - \eta^2(\Gamma(Y))$ and thus for all X $\eta^2(\Gamma(X)) \leq 1$.

3. For each $X \in \mathbb{R}^{n\times p}$ we have $\rho(X) = \text{tr } X'B(X)X$ and thus $\rho(X) \leq \eta(X)\eta(\Gamma(X))$ and thus $\lambda(X) \leq \eta(\Gamma(X))$

$$\rho(X^{(k)}) = \text{tr } \{X^{(k)}\}'VX^{(k+1)} \leq \eta(X^{(k)})\eta(X^{(k+1)}),$$

$$\rho(X^{(k+1)}) \geq \text{tr } \{X^{(k+1)}\}'B(X^{(k)})X^{(k)} = \eta^2(X^{(k+1)}),$$

$$\eta(X^{(k)}) \leq \lambda(X^{(k)}) \leq \eta(X^{(k+1)})$$

and

$$\rho(X^{(k)}) \leq \frac{\eta(X^{(k)}}{X^{(k+1)}}\rho(X^{(k+1)}) \leq \rho(X^{(k+1)})$$

□

5.4.3.2 From Majorization

Smacof is based on the majorization, valid for all configurations X and Y,

$$\sigma(X) \leq 1 + \eta^2(X - \Gamma(Y)) - \eta^2(\Gamma(Y)), \tag{5.12}$$

with equality if and only if $X \propto Y$. If $Y = \alpha X$ for some α then

$$\sigma(X) = 1 + \eta^2(X - \Gamma(Y)) - \eta^2(\Gamma(Y)), \tag{5.13}$$

and specifically we have (5.13) if $Y = X$.

Now suppose we have an Y with $Y \neq \Gamma(Y)$. If $\eta^2(X-\Gamma(Y)) \leq \eta^2(Y-\Gamma(Y))$ then

$$\sigma(X) \leq 1 + \eta^2(Y - \Gamma(Y)) - \eta^2(\Gamma(Y)) = \sigma(Y) \tag{5.14}$$

The obvious choice for X is $X = \Gamma(Y)$, which makes $\eta^2(X - \Gamma(Y)) = 0$, and thus

$$\sigma(X) \leq 1 - \eta^2(\Gamma(Y)) < 1 + \eta^2(Y - \Gamma(Y)) - \eta^2(\Gamma(Y)) = \sigma(Y) \tag{5.15}$$

5.4.3.3 From Ratio of Norms

De Leeuw (1977a)

DC Algorithm

Robert

5.4.4 Component Rotated Smacof

Consider the modified smacof iterations $\tilde{X}^{(k+1)} = X^{(k+1)}L^{(k+1)}$, where $L^{(k+1)}$ are the normalized eigenvectors of $\{X^{(k+1)}\}^T V X^{(k+1)}$. Then

$$\sigma(\tilde{X}^{(k)}) = \sigma(X^{(k)})$$

Thus the modified update produces the same sequence of stress values as the basic update. Also

$$\Gamma(\tilde{X}^{(k)}) = \Gamma(X^{(k)})L^{(k)}$$

Thus $\tilde{X}^{(k)}$ and $X^{(k)}$ differ by a rotation for each k. It follows that we can actually compute $\tilde{X}^{(k)}$ from the basic sequence $X^{(k)}$ by rotating the $X^{(k)}$ to principal components. Specifically if X_∞ is a subsequential limit of $X^{(k)}$ then the corresponding limit of $\tilde{X}^{(k)}$ is X_∞ rotated to principal components. Modifying the intermediate updates is just a waste of time, we can simply rotate the final smacof solution. And we should, as we explain in the next section, 5.4.5.

5.4.5 Local Convergence

$$\mathcal{D}\Gamma(X)(Y) = V^+\left\{B(X)Y - \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}(X)}\left(\frac{\text{tr } Y'A_{ij}X}{d_{ij}^2(X)}\right)A_{ij}\right\}$$

For any X: one zero eigenvalue $\mathcal{D}\Gamma(X)(X) = 0$

If on $\mathbb{R}^{n\times p}$ then an additional p zero eigenvalues $\mathcal{D}\Gamma(X)(e\mu^T) = 0$

For $X = \Gamma(X)$ and M anti-symmetric: $\frac{1}{2}p(p-1)E$ unit eigenvalues $\mathcal{D}\Gamma(X)(XM) = \Gamma(X)M = XM$

5.4.5.1 Cross Product Iterations

Map of C into C. No rotational indeterminacy. Same stress sequence.

$$C^{(k+1)} = V^+ B(C^{(k)}) C^{(k)} B(C^{(k)}) V^+$$

Map of D into D. Guttman transform as function of distances. Not very nice.

$$D^{(k+1)} := D(X^{(k+1)}) = D(\Gamma(X^{(k)}))$$

5.4.5.2 Rotation to PC

We suppose the configfuration X is $n \times p$, with rank p. If the singular value decomposition is $X = K\Lambda L'$ then the rotation to principle components is $\Gamma(X) := K\Lambda = XL$. Thus $\mathcal{D}\Gamma(X)(Y) = YL + X\mathcal{D}L(X)(Y)$. So we need to compute $\mathcal{D}L(X)$, with L the right singular vectors of X, i.e. the eigenvectors of $X^T X$. We use the methods and results from De Leeuw (2007b), which were applied to similar problems in De Leeuw (2008b), De Leeuw and Sorenson (2012), and De Leeuw (2016a).

Theorem 5.3. *If the $n \times p$ matrix has rank p, singular value decomposition $X = K\Lambda L^T$, with all singular values different, then $\Gamma(X + \Delta) = \Gamma(X) + \Delta L + XLM + o(\|\Delta\|)$, where M is antisymmetric with off-diagonal elements*

$$m_{ij} = \frac{\lambda_i s_{ij} + \lambda_j s_{ji}}{\lambda_i^2 - \lambda_j^2}. \tag{5.16}$$

Proof. The proof involves computing the derivatives of the singular value decomposition of X, which is defined by the equations

$$XL = K\Lambda, \tag{5.17}$$
$$X^T K = L\Lambda, \tag{5.18}$$
$$K^T K = I, \tag{5.19}$$
$$L^T L = LL^T = I. \tag{5.20}$$

We now perturb X to $X + \Delta$, which perturbs L to $L + L_\Delta + o(\|\Delta\|)$, K to $K + K_\Delta + o(\|\Delta\|)$, and Λ to $\Lambda + \Lambda_\Delta + o(\|\Delta\|)$. Substutute this into the four SVD equations for $X + \Delta$ and keep the linear terms.

$$XL_\Delta + \Delta L = K_\Delta \Lambda + K\Lambda_\Delta, \tag{5.21}$$
$$X^T K_\Delta + \Delta^T K = L_\Delta \Lambda + L\Lambda_\Delta, \tag{5.22}$$
$$L_\Delta^T L + L^T L_\Delta = 0, \tag{5.23}$$
$$K_\Delta^T K + K^T K_\Delta = 0. \tag{5.24}$$

Define $M := L^T L_\Delta$ and $N := K^T K_\Delta$. Then equations (5.23) and (5.24) say that M and N are antisymmetric. Also define $S := K^T \Delta L$. Premultiplying equation (5.21) by K^T and (5.22) by L^T gives

$$\Lambda M + S = N\Lambda + \Lambda_\Delta, \tag{5.25}$$
$$\Lambda N + S^T = M\Lambda + \Lambda_\Delta. \tag{5.26}$$

Either of these two equations gives, using the antisymmetry, and thus hollowness, of M and N, that $\Lambda_\Delta = \mathrm{diag}(S)$. Define the hollow matrix $U := S - \mathrm{diag}(S)$. Then

$$\Lambda M - N\Lambda = U, \tag{5.27}$$
$$\Lambda N - M\Lambda = U^T. \tag{5.28}$$

Premultiply (5.27) and postmultiply (5.28) by Λ.

$$\Lambda^2 M - \Lambda N \Lambda = \Lambda U, \tag{5.29}$$
$$\Lambda N \Lambda - M\Lambda^2 = U^T \Lambda. \tag{5.30}$$

If we add these two equations we can solve for the off-diagonal elements of M and find the expression in the theorem. Since $L_\Delta = LM$ this completes the proof. □

5.4.6 Data Asymmetry

The non-basic situation in which there are asymmetric weights and/or asymmetric dissimilarities in basic MDS is analyzed in De Leeuw (1977a), although it is just standard linear least squares projection theory. We give a slightly different partitionng ofd the sum of squares here. Note that it is not even necessary that the weights and dissimilarities are hollow and/or non-negative.

We decompose the weights and dissimilarities additively into a symmetric and anti-symmetric part. Thus $w_{ij} = w_{ij}^S + w_{ij}^A$ and $\delta_{ij} = \delta_{ij}^S + \delta_{ij}^A$. Now in general if A is anti-symmetric and B is symmetric, then tr $AB = 0$. Also their Hadamard (element-wise) product $A * B$ is anti-symmetric, and the Hadamard product of two anti-symmetric matrices is symmetric. Using these rules gives after some calculation

$$\begin{aligned}\sigma(X) &= \frac{1}{4}\sum_{i=1}^n\sum_{j=1}^n w_{ij}(\delta_{ij} - d_{ij}(X))^2 = \\ &= \frac{1}{4}\sum_{i=1}^n w_{ii}\delta_{ii}^2 + \frac{1}{2}\sum_{1\leq i<j\leq n}\sum w_{ij}^S\left\{\underline{\delta}_{ij} - d_{ij}(X)\right\}^2 + \frac{1}{2}\sum_{1\leq i<j\leq n}\sum\frac{w_{ij}w_{ji}}{w_{ij}^S}\{\delta_{ij}^A\}^2,\end{aligned} \tag{5.31}$$

where

$$\underline{\delta}_{ij} := \delta_{ij}^S + \frac{w_{ij}^A\delta_{ij}^A}{w_{ij}^S}. \tag{5.32}$$

Thus minimizing stress in the case of asymmetric weights and dissimilarities, which even can be non-hollow and non-positive, reduces to a symmmetric basic MDS problem for adjusted dissimilarities defined by equation (5.32). If the original weights and dissimilarities are non-negative, then so are the weights w_{ij}^S and the dissimilarities $\underline{\delta}_{ij}$.

5.4.7 Replications

If there are replications in basic MDS we can use a simple partitioning of stress to reduce the problem to standard form. We start with

$$\sigma(X) = \frac{1}{2}\sum_{k=1}^m\sum_{1\leq i<j\leq n}\sum w_{ijk}(\delta_{ijk} - d_{ij}(X))^2.\#(eq : inddiffstress) \tag{5.33}$$

Let

$$w_{ij\bullet} = \sum_{k=1}^{m} w_{ijk} \tag{5.34}$$

$$\delta_{ij\bullet} = \frac{\sum_{k=1}^{m} w_{ijk}\delta_{ijk}}{w_{ij\bullet}}. \tag{5.35}$$

Then

$$\sigma(X) = \frac{1}{2}\sum\sum_{1\leq i<j\leq n} w_{ij\bullet}(\delta_{ij\bullet} - d_{ij}(X))^2 + \frac{1}{2}\sum_{k=1}^{m}\sum\sum_{1\leq i<j\leq n} w_{ijk}(\delta_{ijk} - \delta_{ij\bullet})^2, \tag{5.36}$$

and it suffices to minimize the first term, which is a standard basic MDS problem.

In the nonmetric case, in which in principle each of the Δ_k can be transformed, we must alternate minimization of #ref(eq:inddiffstress) over the Δ_k and minimization of (5.36) over X. In the case in which X_k is different pover replications we use the methods of chapter 16.

5.4.8 Negative Dissimilarities

$$\sigma(X) = 1 - \sum_{k\in\mathcal{K}_{1+}} w_k\delta_k d_k(X) + \sum_{k\in\mathcal{K}_{1-}} w_k|\delta_k| d_k(X) + \frac{1}{2}\sum_{k\in\mathcal{K}} w_k d_k^2(X)). \tag{5.37}$$

Split rho

Heiser (1991)

5.4.9 Normalization

In actual computer output using the scaling in formula (1.3) and (1.3) has some disadvantages. There are, say, M non-zero weights. The summation in #ref(eq:stressall) is really over M terms only. If n is at all large the scaled dissimilarities, and consequently the distances and the configuration, will become very small. Thus, in actual computation, or at least in the computer output, we scale our dissimilarities as $\frac{1}{2}\sum\sum_{1\leq j<i\leq n} w_{ij}\delta_{ij}^2 = M$. So, we scale our dissimilarities to one in formulas and to M in computations. Thus the computed stress will b

In fact, we do not even use it in our computer programs, except at the very last moment when we return the final stress after the algorithm has completed.

5.4.10 Unweighting

Consider the general problem of minimizing a least squares loss function, defined as $f(x) := (x-y)'W(x-y)$ over x in some set X, where W is a symmetric weight matrix. Sometimes W complicates the problem, maybe because it is too big, too full, too singular, or even indefinite. We will use iterative majorization to give W a more subordinate role. See also Kiers (1997) and Groenen, Giaquinto, and Kiers (2003).

Suppose z is another element of X. Think of it as the current best approximation to y that we have, which we want to improve. Then

$$\begin{aligned} f(x) &= (x-y)'W(x-y) \\ &= ((x-z)+(z-y))'W((x-z)+(z-y)) \\ &= f(z)+2(x-z)'W(z-y)+(x-z)'W(x-z) \end{aligned} \tag{5.38}$$

Now choose a non-singular V such that $W \lesssim V$ and define $u := V^{-1}W(z-y)$. Then we have the majorization

$$f(x) \leq f(z)+2(x-z)'W(z-y)+(x-z)'V(x-z) = f(z)+2(x-z)'Vu+(x-z)'V(x-z) = f(z)+(x-(z-u)) \tag{5.39}$$

Here are some ways to choose V. We use $\lambda_{\max}(W)$ and $\lambda_{\min}(W)$ for the largest and smallest eigenvalues of the symmetric matrix W.

For any W we can choose $V = \lambda_{\max}(W)I$. Or, more generally, $V = \lambda_{\max}(D^{-1}W)D$ for any positive definite D. If W is singular we can choose $V = W + \epsilon D$ for any positive definite D. And in the unlikely case that W is indefinite we can choose $V = W + (\epsilon - \lambda_{\min}(W))I$. But if W is indefinite we have more serious problems.

In appendix A.1.19 the R function lsuw(), implements the iterative majorization algorithm minimizing $(x-y)'W(x-y)$ over x in some set X. One of

the parameters of lsuw() is a function proj(), which projects a vector on X in the metric define by V. The projection could be on the positive orthant, on a cone with isotone vectors, on a linear subspace, on a sphere, on a set of low-rank matrices, and so on.

As an example choose W as a banded matrix of order 10 with $w_{ij} = 1$ if $|i - j| \leq 3$ and $i \neq j$, $w_{ij} = i$ if $i = j$, and $w_{ij} = 0$ otherwise. We require all 10 elements of x to be the same, and we use $V = \lambda_{\max}(W)I$ (the default).

The iterations are

```
w<-ifelse(outer(1:10,1:10,function(x,y) abs(x-y) <= 3),1,0)
w <- w + diag(0:9)
h1 <- lsuw(1:10, w, projeq)
```

If we use $\lambda_{\max}(D^{-1}W)D$ with $D = \text{diag}(W)$ for V we see the following majorization iterations.

```
d <- diag(w)
v <- max(eigen((1 / d) * w)$values) * diag(d)
h2 <- lsuw(1:10, w, v = v, projeq)
```

So the second method of choosing V is a tiny bit less efficient in this case, but it really does not make much of a difference. In both cases x is 6.3009558, 6.3009558, 6.3009558, 6.3009558, 6.3009558, 6.3009558, 6.3009558, 6.3009558, 6.3009558, 6.3009558 with function value 595.6699029.

Apply to stress and to

Inner iterations, use one.

$$\sigma_c(X) := \sum_{1\leq i<j\leq n}\sum \sum_{1\leq k<l\leq n}\sum w_{ijkl}(\delta_{ij} - d_{ij}(X))(\delta_{kl} - d_{kl}(X))$$

If $A \leq B$ (elementwise) then $\sum\sum(b_{ij} - a_{ij})(x_i - x_j)^2 \geq 0$ and thus $V(A) \lesssim V(B)$.

5.4.10.1 Symmetric non-negative matrix factorization

$w_{ij} = \sum_{s=1}^r v_{is}^2 v_{js}^2$ for all $i \neq j$. Then

$$\sigma(X) = \sum_{s=1}^p \sum\sum_{1\leq i<j\leq n} (\delta_{ijs} - d_{ijs}(X))^2$$

with $\delta_{ijs} := v_{is}v_{js}\delta_{ij}$ and $d_{ijs}(X) := v_{is}v_{js}d_{ij}(X)$.

5.5 Stress Envelopes

intro

5.5.1 CS Majorization

Theorem 5.4. *σ is the lower envelop of an infinite number of convex quadratics.*

Proof. By the CS inequality

$$d_{ij}(X) = \max_Y \frac{\text{tr } X^T A_{ij} Y}{d_{ij}(Y)}, \tag{5.40}$$

which implies

$$\sigma(X) = \min_Y \left(1 - \text{tr } X^T B(Y) Y + \frac{1}{2}\text{tr } X^T V X\right), \tag{5.41}$$

which is what we set out to prove. □

We can use the lower envelop of a finite number of the quadratics from theorem 5.4 to approximate stress. This is illustrated graphically, using a small example in which the configuration is a convex combination of two fixed configurations. Thus in the example stress is a function of the single parameter $0 \leq \lambda \leq 1$ defining the convex combination. In figure 5.1 stress is in red, and we have used the three quadratics corresponding with λ equal

to 0.25, 0.5, 0.75. The maximum of the three quadratics is in blue, and the approximation is really good, in fact almost perfect in the areas where the blue is not even visible. As an aside, we also see three points in the figure where stress is not differentiable. The minimum of the three quadratics is also not differentiable at a point, but that point is different from the points where stress is non-smooth.

Note that by definition stress and the lower envelop of the quadratics are equal at the three points where λ is 0.25, 0.5, 0.75, i.e at the three vertical lines in the plot.

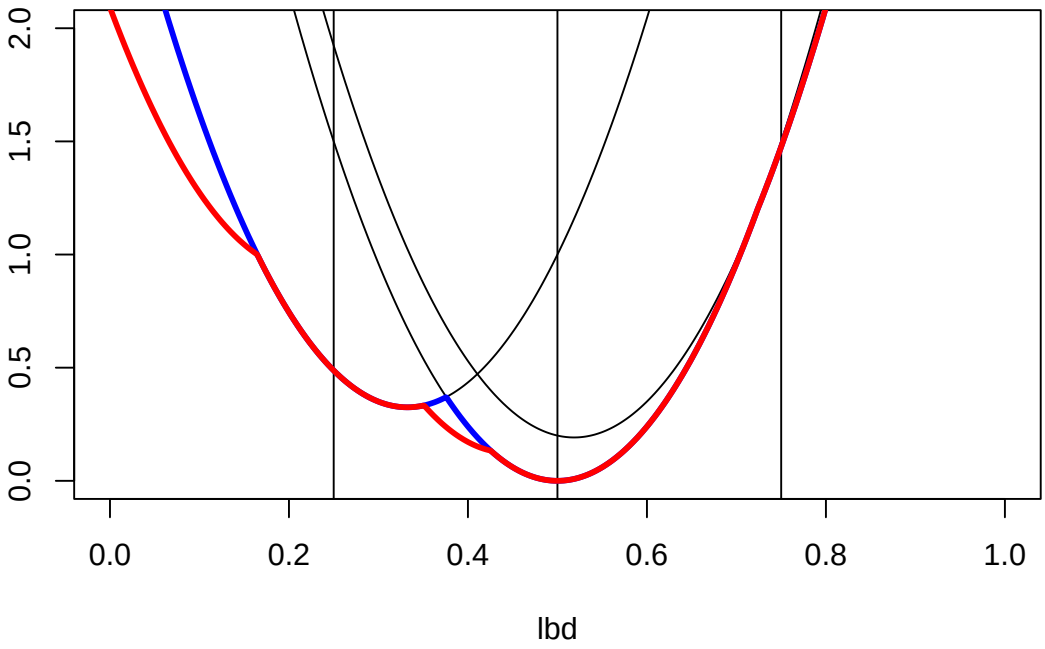


Figure 5.1: Piecewise Quadratic Upper Approximation

5.5.2 AM/GM Minorization

Instead of approximating stress from above, we can also approximate it from below.

Theorem 5.5. *σ is the upper envelop of an infinite number of quadratics.*

Proof. By AM/GM

$$d_{ij}(X) \leq \min \frac{1}{2}\frac{1}{d_{ij}(Y)}\{d_{ij}^2(X) + d_{ij}^2(Y)\} \tag{5.42}$$

Thus

$$\sigma(X) = \max_Y \left(1 - \frac{1}{2}\rho(Y) + \frac{1}{2}\text{tr } X'(V - B(Y))X\right) \tag{5.43}$$

□

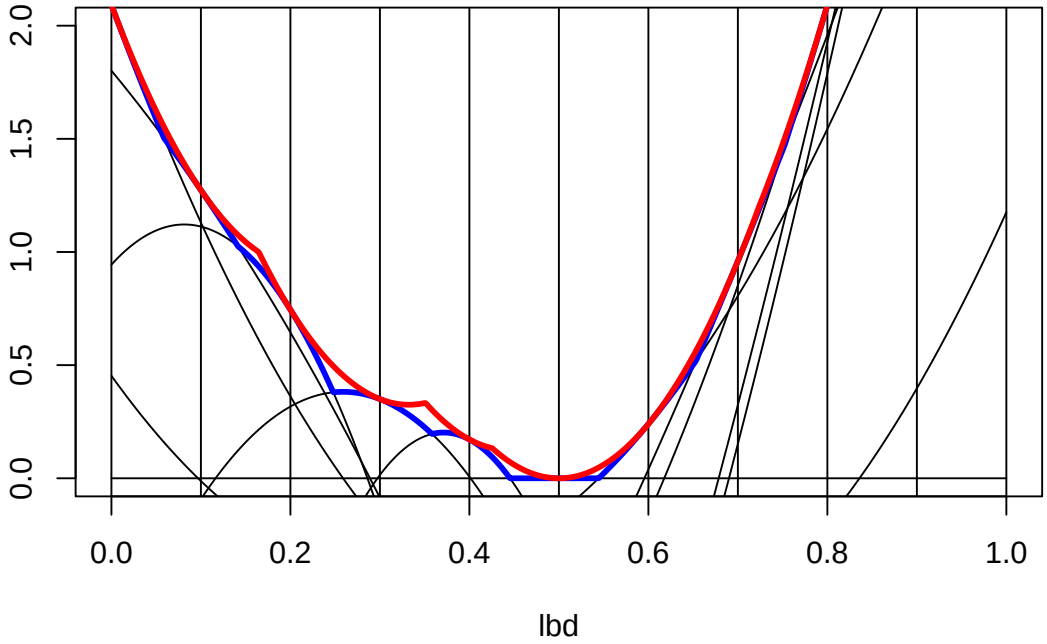


Figure 5.2: Piecewise Quadratic Lower Approximation

Again we illustrate this result using a finite number of quadratics. In figure 5.2 we choose λ equal to 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. Although we now use 11 quadratics, and thus force the envelop to be equal to the function at the 11 points on the vertical lines in the plot, the approximation is poor. This seems to be mainly because the convex-like function stress must be approximated from below by quadratics which are often concave.

5.5.3 Dualities

$$\min_X \sigma(X) = \min_Y \left(1 - \frac{1}{2}\text{tr } Y'B(Y)V^+B(Y)Y\right) = 1 - \frac{1}{2}\max_Y \text{tr } Y'B(Y)V^+B(Y)Y. \quad (5.44)$$

Thus minimizing stress is equivalent to maximizing $\eta^2(V^+B(X)X)$.

$$\min_X \sigma(X) \geq \max_{B(Y)\lesssim V}(1 - \rho(Y))$$

By the minimax inequality $\min_X \sigma(X) = \min_X \max_Y \theta(X,Y) \geq \max_Y \min_X \theta(X,Y)$. Now $\min_X \theta(X,Y)$ is $-\infty$, unless $B(Y) \lesssim V$, in which case $\min_X \theta(X,Y) = 0$. Thus

$$\max_Y \min_X \theta(X,Y) = \max_{B(Y)\lesssim V}(1 - \rho(Y)) = 1 - \min_{B(Y)\lesssim V} \rho(Y)$$

5.6 Smacof in Coefficient Space

5.7 Newton in MDS

5.7.1 Regions of Attraction

```
delta <- as.matrix (dist (diag (4)))
delta <- delta * sqrt (2 / sum (delta ^ 2))
```

5.7.1.1 Smacof

We use the smacof() function from the code in the appendix with 100 different starting points of θ, equally spaced on the circle. Figure 5.3 is a histogram of the number of smacof iterations to convergence within 1e-15. In all cases smacof converges to a local minimum in coefficient space, never

to a saddle point. Figure 5.4 shows which local minima are reached from the different starting points. This shows, more or less contrary to what Trosset and Mathar (1997) suggests, that non-global minima can indeed be points of attraction for smacof iterations.

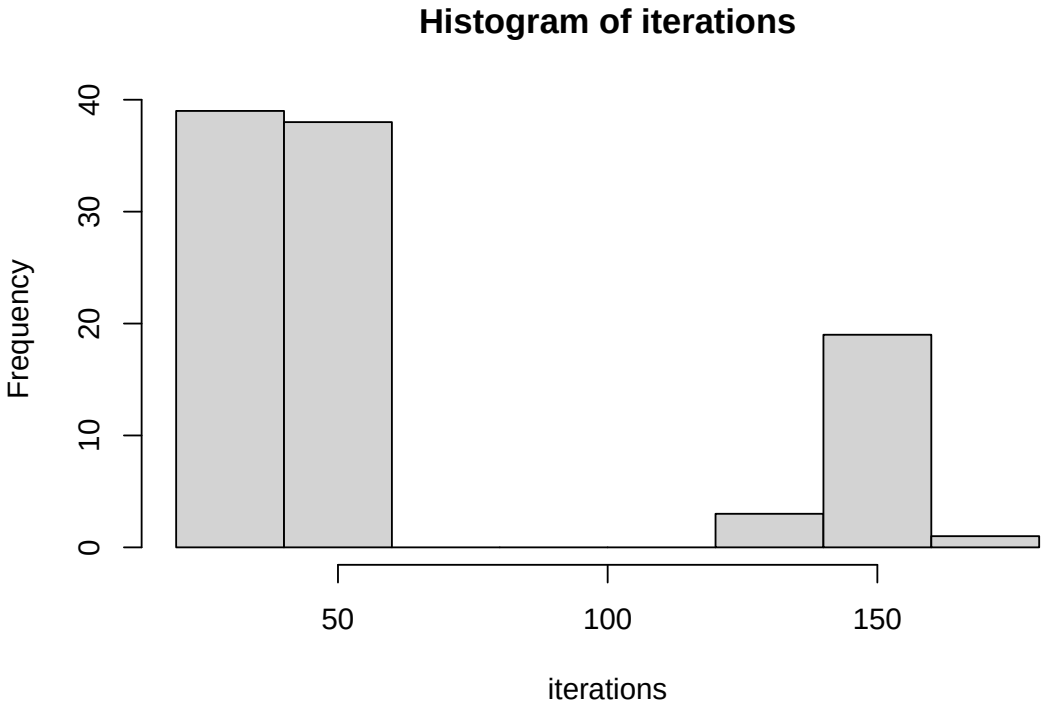


Figure 5.3: Histogram Number of Smacof Iterations

5.7.1.2 Newton

We repeat the same exercise with Newton's method, which also converges from all 100 starting points in our example. In higher dimensions we may not be so lucky.

The histogram of iteration counts is in figure 5.5. It shows in this example that `smacof` needs about 10 times the number of iterations that Newton needs. Because `smacof` iterations are much less expensive than Newton ones, this does not really say much about computing times. If we look at figure 5.6 we see the problem with non-safeguarded Newton. Although we have fast

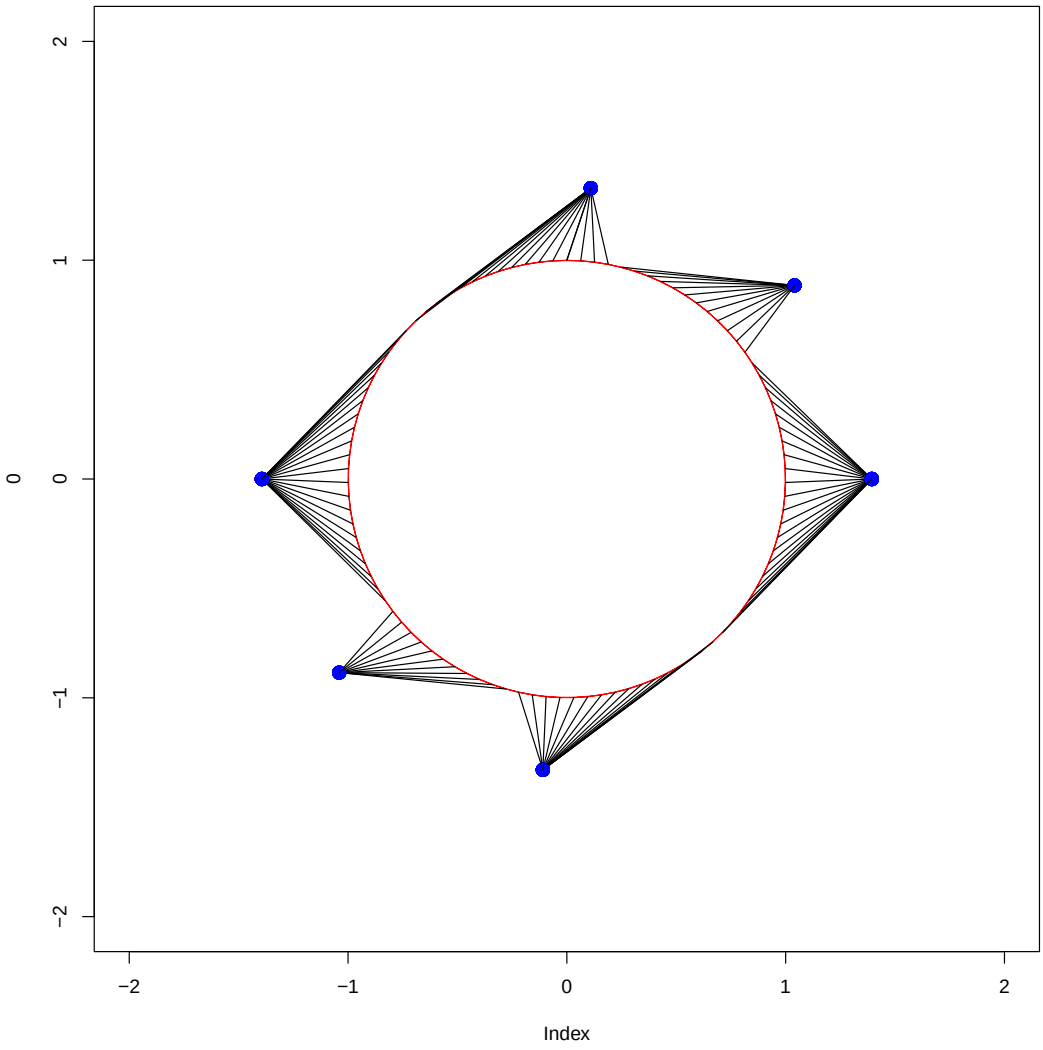


Figure 5.4: Path Endpoints of Smacof Iterations

convergence from all 100 starting points, Newton converges to a saddle point in 45 cases.

Figure 5.5: Histogram Number of Newton Iterations

5.8 Distance Smoothing

In sections 2.4 and 2.5 we show the lack of differentiability in basic MDS is not a serious problem in the actual computation of local minima.

There is another rather straightforward way to circumvent the differentiabily issue, which actually may have additional benefits. The idea is to use an approximation of the Euclidean distance that is as close as possible on the positive real axis, but smooth at zero. This was first applied in unidimensional MDS by Pliner (Pliner (1986), Pliner (1996)) and later taken up and generalized to pMDS for arbitrary p, and even for arbitrary Minkovski metrics, by Groenen, Heiser, and Meulman (1998) and Groenen, Heiser, and

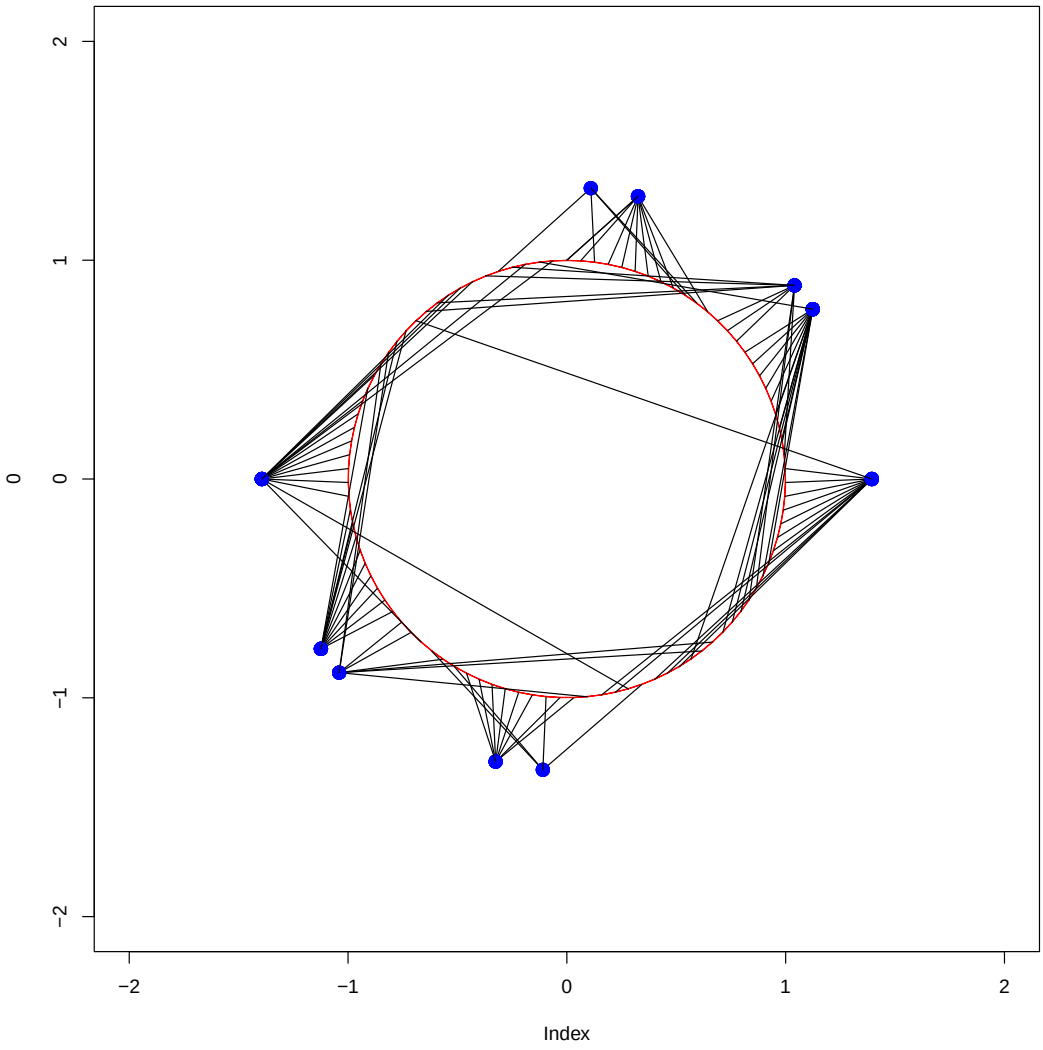


Figure 5.6: Path Endpoints of Newton Iterations

Meulman (1999). They coined the term *distance smoothing* for this variation of the smacof framework for MDS.

Pliner (1986) uses a smooth approximation of the sign function, while Groenen, Heiser, and Meulman (1998) borrow the smooth Huber approximation of the absolute value function from robust regression. We use another classical and efficient approximation $|x| \approx \sqrt{x^2+\epsilon^2}$ to the absolute value function, used in image analysis, location analysis, and computational geometry (De Leeuw (2018b), Ramirez et al. (2014)). In our context that becomes $d_{ij}(X) \approx d_{ij}(X,\epsilon) := \sqrt{d_{ij}^2(X)+\epsilon^2}$. Note that on the non-negative reals

$$\max(\epsilon, d_{ij}(X)) \leq d_{ij}(X,\epsilon) \leq d_{ij}(X) + \epsilon. \tag{5.45}$$

Figures 5.7 and 5.8 show the absolute value function and its derivative are approximated for ϵ equal to 0, 0.01, 0.05, 0.1, 0.5.

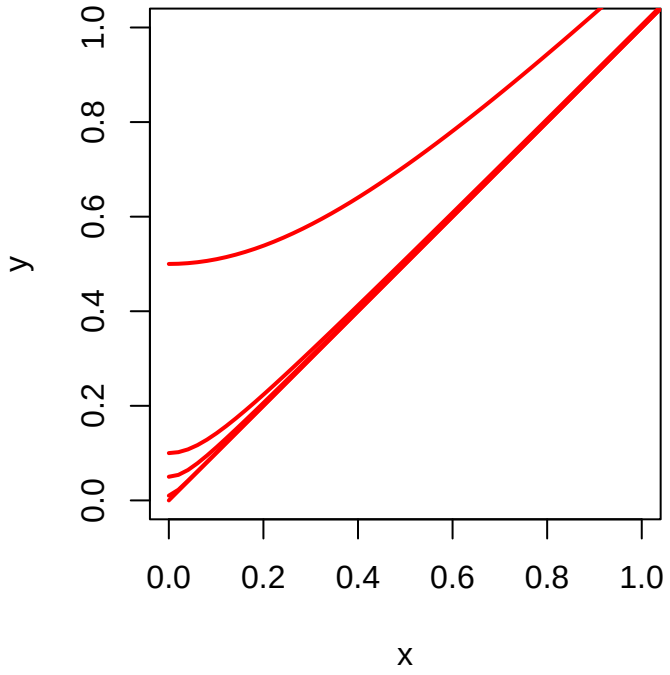


Figure 5.7: Function for Various Epsilon

The distance smoother we use fits nicely into smacof. Define $X_\epsilon := \begin{bmatrix} X & | & \epsilon I \end{bmatrix}$. Then $d_{ij}(X_\epsilon) = \sqrt{d_{ij}^2(X)+\epsilon^2}$. Thus we can define

$$\sigma_\epsilon(X) := \sigma(X_\epsilon) = \sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij} - d_{ij}(X_\epsilon))^2, \tag{5.46}$$

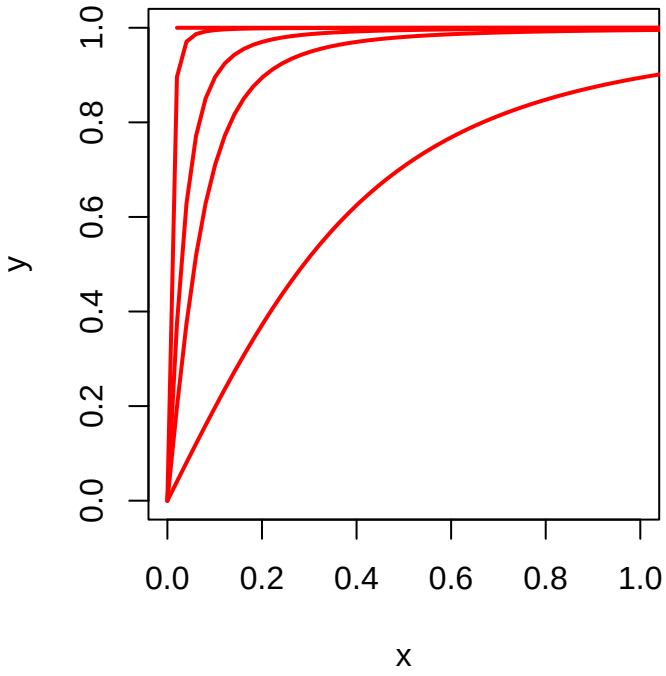


Figure 5.8: Derivative for Various Epsilon

with ρ_ϵ and η^2_ϵ defined in the same way.

For a fixed $\epsilon > 0$ now $d_{ij}(X_\epsilon)$, and thus stress, is (infinitely many times) differentiable on all of $\mathbb{R}^{n\times p}$. Moreover $d_{ij}(X,\epsilon)$ is convex in X for fixed ϵ and jointly convex in X and ϵ, and as a consequence so are ρ_ϵ and η^2_ϵ.

Figure 5.9: Jan de Leeuw, Gilbert Saporta, Yutaka Kanaka in Kolkata, December 1985

Chapter 6

Acceleration of Convergence

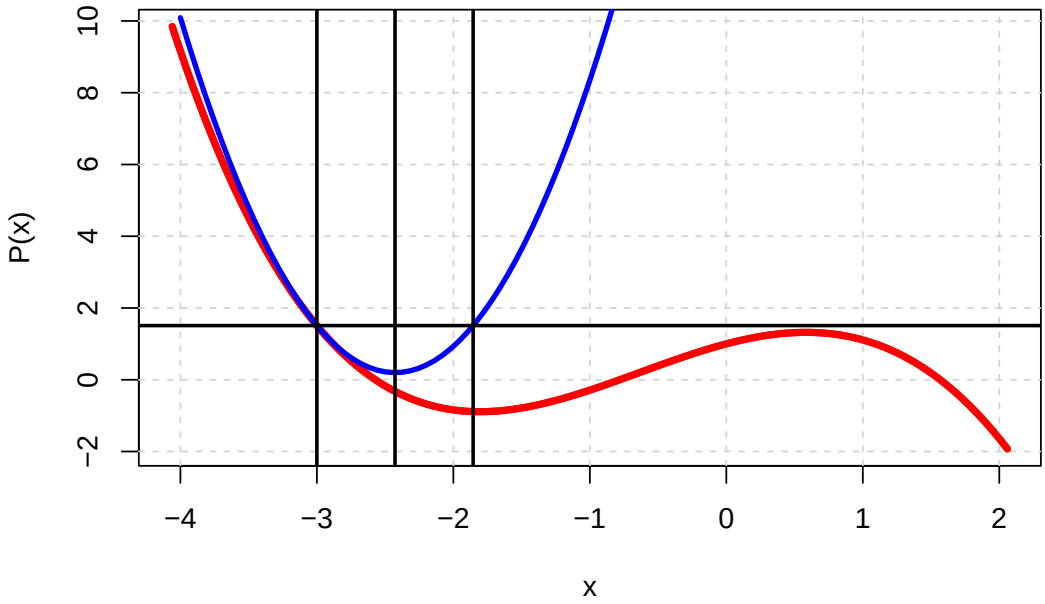
6.1 Simple Acceleration

A simple and inexpensive way to accelerate smacof iterations was proposed by
De Leeuw and Heiser (1980).

On the other hand, if we choose $X = 2\Gamma(Y) - Y$ then again $X \neq Y$, but $\eta^2(X - \Gamma(Y)) = \eta^2(Y - \Gamma(Y))$. Thus

$$\sigma(X) \leq 1+\eta^2(X-\Gamma(Y))-\eta^2(\Gamma(Y)) = 1+\eta^2(Y-\Gamma(Y))-\eta^2(\Gamma(Y)) = \sigma(Y). \tag{6.1}$$

Let's define the two update rules $\text{up}_A(X) := \Gamma(X)$ and $\text{up}_B(X) = 2\Gamma(X)-X$.



This is illustrated in figure .... We want to locate a local minimum of f, in red, in the interval $(-4, 2)$. In this case we happen to know that f is a quartic polynomial, with minimum -0.8894476 at -1.8044048. In the interval we are looking at we have $f''(x) \leq 8$. Suppose our initial guess for the location of the minimum is $x = -3$, the first vertical line from the left, with $f(-3)$ equal to 1.51. The upper bound on the second derivative allows us to construct a quadratic majorizer g, in blue, touching f at -3. Update rule up_A tells us to go to the minimum of g, which is at -2.4275, the second vertical line. Here g is equal to 0.198975 and f is -0.3245082.

Rule up_B "overrelaxes" and goes all the way to -1.855, the third vertical line from the left, where g is equal to both $g(-3)$ and $f(-3)$, and where f is -0.8862781, indeed much closer to the minimum. Examples such as this make up_B look good.

De Leeuw and Heiser give a rather informal theoretical justification of up_B as well. Suppose the sequence $X^+ = \Gamma(X)$ generated by up_A has slow linear convergence with ACR $1 - \epsilon$, where ϵ is positive and small. Then choosing the up_B will change the ACR of $1 - \epsilon$ to $2(1 - \epsilon) - 1 = 1 - 2\epsilon \approx (1 - \epsilon)^2$, and will approximately halve the number of iterations to convergence. This argument is supported by numerical experiments which seem to show that indeed about half the number of iterations are needed. It seems that up_B will get you something for almost nothing, and thus it has been implemented in

various versions of the smacof programs as the default update. Unfortunately this may mean that many users have obtained, and presumably reported, MDS results that are incorrect.

What is ignored in De Leeuw and Heiser (1980) is that majorization only guarantees that the sequence of loss function values converges for both update methods. The general convergence theory discussed earlier in this chapter shows that for both up_A and up_B the sequence $\{X^{(k)}\}$ has at least one accumulation point, and that the accumulation points of the sequence $\{X^{(k)}\}$ are fixed points of the update rule, which means for both up_A and up_B that at accumulation points X we have $X = \Gamma(X)$. But it does **not** say that $\{X^{(k)}\}$ converges.

The argument also ignores that at any X the derivative of up_A has a zero eigenvalue, with eigenvector X. For up_B the eigenvector X has eigenvalue equal to -1, which is the largest one in modulus near any local minimum. And so ...

Suppose we have a configuration of the form αX with $X = \Gamma(X)$. Then $\text{up}_B(\alpha X) = 2\Gamma(\alpha X) - \alpha X = (2-\alpha)X$ and $\text{up}_B((2-\alpha)X) = \alpha X$. Thus starting with $X^{(1)} = \alpha X$ up_B generates a sequence with even members $(2-\alpha)X$ and odd members αX. Thus there are two convergent subsequences with accumulation points αX and $(2-\alpha)X$. And never the twain shall meet.

As far as stress is concerned, note that if $X = \Gamma(X)$ then $\sigma(\alpha X) = \sigma((2-\alpha)X)$. Thus the stress values never change, and consequently form a convergent sequence.

We also see that $\text{up}_B^{(2)}(\alpha X) := \text{up}_B(\text{up}_B(\alpha X)) = \alpha X$, which means that αX is a fixed point of $\text{up}_B^{(2)}$ for any fixed point X of up_A and any α.

Another way to express the difference between the two update rule is that up_A is *self-scaling*, i.e. $\Gamma(\alpha X) = \Gamma(X)$, while up_B is not. Self-scaling implies $\mathcal{D}\Gamma(X)(X) = 0$, while for up_B $\mathcal{D}(2\Gamma(X) - X)(X) = -X$.

Let's now look at a real example. We use the Ekman color similarity data again, this time transformed by $\delta_{ij} = (1 - s_{ij})^3$, The analysis is in two dimensions, with no weights. We run four analyses, by crossing update rules up_A and up_B with stopping criteria $\sigma(X^{(k)}) - \sigma(X^{(k+1)}) < \epsilon$ and $\max_{i,s} |x_{is}^{(k)} - x_{is}^{(k+1)}| < \epsilon$. Let's call these stopping criteria *stop_s* and *stop_x*. In all cases we allow a maximum of 1000 iterations and we set ϵ to 1e-10.

	stop_f	stop_x
rule A	17	32
rule B	15	1000

	stop_f	stop_x
rule A	0.4696867	0.4696867
rule B	0.6176456	0.6176456

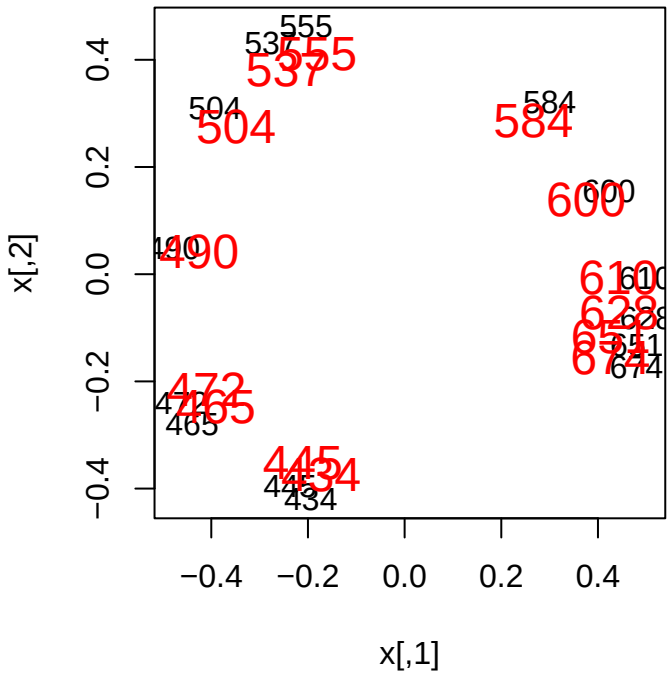
The results are in table ... The first subtable gives the number of iterations, the second the final stress value. We see that generally stop_x requires more iterations than stop_s, because it is a stricter criterion. If we use stop_x then up_B does not converge at all. Both with stop_s and stop_x up_B gves a higher stress value than up_A. And yes, with stop_s (which is the default stop criterion in the smacof programs so far) up_B use fewer iterations than up_A.

To verify that something is seriously wrong with running up_B, we compute the maximum absolute value of the gradient at convergence for both rules and stop_s. For up_A it is 0.0000000010 and for up_B it is 0.7834335001. Once again, with up_B both loss function and configuration converge to an incorrect value.

This can also be illustrated graphically. We see from table ... that up_B with stop_x ends after 1000 iteration. We perform an extra iteration, number 1001, and see how the configuration changes. In figure ... iteration 1000 is in black, iteration 1001 in red with slightly bigger characters. Except for a scaling factor the two configurations are the same. Elementwise dividing the up_B by the up_A final configuration gives a shrinkage factor α of 1.0595315. This shrinkage factor can also be computed from the final stress values. Using $\rho(X) = \eta^2(X)$ and $\sigma(X) = 1 - \eta^2(X)$ we find $\sigma(\alpha X) - \sigma(X) = (\alpha - 1)\eta^2(X)$, and thus

$$\alpha = 1 \pm \sqrt{\frac{\sigma(\alpha X) - \sigma(X)}{1 - \sigma(X))}}. \tag{6.2}$$

There are two values α and $2 - \alpha$, equal to 0.9595728 and 1.0404272, because the sequence has two accumulation points.



Things do not look good for up_B but simple remedies are available. The first one is renormalization. After the iterations, with say stop_s, have converged, we scale the configuration such that $\rho(X) = \eta^2(X)$ and recompute stress. This corrects both stress and the confguration to the correct outcome. Another way to normalize is to do another single up_A step after convergence of up_B. This has the same effect. We tried up_B with both renormalization approaches and both stop_s and stop_b. The number of up_B iterations is still the same as in table ... because we just compute something additional at the end. All stress values for the four combinations are now the correct 0.4696867. It seems that using up_B with stop_s and renormalization at the end gives us the best of both worlds. It accelerates convergence and it gives the correct loss function values.

Of course up_B with stop_x still does not converge, and probably the best way to deal with that unfortunate fact is to avoid the combination alltogether.

We can still use stop_x and get acceleration by define a single interation as $\text{up}_{AB}(x) := \text{up}_A(\text{up}_B(X))$. For comparison purposes we also run $\text{up}_{AA}(x) := \text{up}_A(\text{up}_A(X))$. Both converge to the correct values, up_{AA} in 17 and $\text{up}_{AB}(x)$ in 10 iterations.

Again up_{AB} is an attractive strategy. It works with both stop_s and stop_x

and it accelerates. Less so than up_B, however. If the ACR of up_A is $1-\epsilon$, then, by the same reasoning as before, the ACR of up_{AB} is $(1-\epsilon)^{\frac{3}{2}}$.

6.2 One-Parameter Methods

In psychometrics, and perhaps in multivariate analysis, Ramsay (1975) was the first to apply a general acceleration methods to sequences in $\mathbb{R}^n$ of the form $x^{(k+1)} = f(x^{(k)})$.

De Leeuw (2006)

6.3 SQUAREM

6.4 Vector Extrapolation Methods

De Leeuw (2008a)

De Leeuw (2008c)

Sidi (2017)

Chapter 7

Nonmetric MDS

7.1 Generalities

In non-metric MDS the dissimilarities are not a vector of known non-negative numbers, but they are only known up to a transformation or quantification. Ever since Kruskal (1964a) the approach for dealing with this aspect of the MDS problem is to define stress as a function of both X and Δ, and to minimize

$$\sigma(X,\Delta) := \sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij}-d_{ij}(X))^2 \tag{7.1}$$

over both configurations X and feasible *disparities* (i.e. transformed dissimilarities). The name *disparities* was coined, as far as I know, by Forrest Young and used in our joint ALS work from the seventies (Takane, Young, and De Leeuw (1977)). Kruskal's name for the transformed or quantified dissimilarities is *pseudo-distances*.

To work with a general notion of the feasability of a matrix of disparities we use the notation $\Delta \in \mathfrak{D}$. Typically, although not necessarily, $\mathfrak{D}$ is a convex set in disparity space. In interval, polynomial, splinical, and ordinal MDS it usually is a convex cone with apex at the origin. This implies that $0 \in \mathfrak{D}$, and consequently that

$$\min_{X\in\mathbb{R}^{n\times p}}\min_{\Delta\in\mathfrak{D}} = 0, \tag{7.2}$$

with the minimum attained at $X = 0$ and $\Delta = 0$. Of course this is a *trivial solution*, which is completely independent of the data. Thus we cannot

formulate the NMDS problem as the minimization of stress from equation (7.1) over unconstrained X and over Δ in its cone. We need some way to exclude either $X = 0$ or $\Delta = 0$, or both, from the feasible solutions. This we can do either by normalization of the loss function, or by using constraints that explicitly exclude one or both zero solutions. The commonly used options will be discussed in section 7.4.1 of this chapter.

7.1.1 Kruskal's Stress

... we shall find ourselves doing arithmetic with dissimilarities. This we must not do, because we are committed to using only the rank ordering of the dissimilarities. (Kruskal (1964a), p 6-7)

section 7.4.1

7.2 Single and Double Phase

The distinction between *single phase* and *double phase* NMDS algorithms, introduced by Guttman (1968), has caused a great deal of confusion in the early stages of non-metric MDS (say between 1960 and 1970).

> This beguiling complex distinction has given rise to an almost endless debate (among > Guttman, Kruskal, Lingoes, Roskam, and Shepard – for all permutations of five things taken two at a time) and has caused anguish and despair (accompanied by an imprecation or two by at least four of the five) extending over a three year period – only occasionally alleviated by evanescent flashes of partial insight (Lingoes and Roskam (1973))

I was an active, although late-arriving, participant in discussing, and perhaps perpetuating, this confusion (De Leeuw (1973b)). For raking up this debate at this late stage I was sternly spoken to by Jim Lingoes, who pointed me to the discussion in Lingoes and Roskam (1973).

> I would hate to believe that after this heroic attempt on our part that "we all" would once more be engaged in a "correspondence musical chairs" on these issues. (Lingoes in De Leeuw (1973b)).

Nevertheless, even in later discussions of the distinction between single-phase and double-phase (such as Roskam (1979)) I still get the feeling that there are some unresolved misunderstandings. Thus I will pay some more attention to the musical chairs here.

By the very definition of the minimum of a function we have the mathematical truism

$$\min_{(X,\Delta)\in\mathfrak{X}\otimes\mathfrak{D}}\sigma(X,\Delta)=\min_{X\in\mathfrak{X}}\min_{\Delta\in\mathfrak{D}}\sigma(X,\Delta)=\min_{X\in\mathfrak{X}}\left\{\min_{\Delta\in\mathfrak{D}}\sigma(X,\Delta)\right\}=\min_{\Delta\in\mathfrak{D}}\left\{\min_{X\in\mathfrak{X}}\sigma(X,\Delta)\right\}, \tag{7.3}$$

provided all minima exist. This is true no matter what the subsets $\mathfrak{X}$ of configuration space and $\mathfrak{D}$ of disparity space are.

7.2.1 Double Phase

In a *double phase algorithm* we alternate the minimization of stress over X and Δ. Thus

$$X^{(k+1)}=\operatorname*{argmin}_{X\in\mathfrak{X}}\sigma(X,\Delta^{(k)}), \tag{7.4}$$

$$\Delta^{(k+1)}=\operatorname*{argmin}_{\Delta\in\mathfrak{D}}\sigma(X^{(k+1)},\Delta).. \tag{7.5}$$

Thus double phase algorithms are *alternating least squares* or ALS algorithms. The designation "alternating least squares" was first used, AFAIK, by De Leeuw (1968b), and of course it was widely disseminated by the series of ALS algorithms of Young, Takane, and De Leeuw in the seventies (see F. W. Young (1981) for a retrospective summary).

There are some possible variations in the ALS scheme. In equation (7.4) we update X first, and then in equation (7.4) we update Δ. That order can be reversed without any essential changes. More importantly, we have to realize that minimizing over X in equation (7.4) is a basic metric MDS problem, which will generally take an infinite number of iterations for an exact solution. This means we have to truncate the minimization, and stop at some point. And, in addition, equation (7.4) implies we have to find the global minimum over X, which is generally infeasible as well.Thus the ALS scheme as defined cannot really be implemented.

We remedy this situations by switching from minimization in each substep to a decrease, or, notationwise, from argmin to arglower. The resulting update sequence

$$X^{(k+1)} = \underset{X\in\mathfrak{X}}{\text{arglower}}\,\sigma(X,\Delta^{(k)}), \tag{7.6}$$

$$\Delta^{(k+1)} = \underset{\Delta\in\mathfrak{D}}{\text{arglower}}\,\sigma(X^{(k+1)},\Delta).. \tag{7.7}$$

is much more loosely defined than the previous one, because arglower can be implemented in many different ways. More about that later. But at least the new scheme can actually be implemented.

Algorithm #ref(eq:nmslte1) and #ref(eq:nmslte2) is still considered to be ALS, but it is also firmly in the class of *block relaxation* algorithms. General block relaxation, which has alternating least squares, coordinate relaxation, augmentation, EM, and majorization as special cases, was used to describe many different data analysis algorithms in De Leeuw (1994). As with ALS, special cases of block relaxation have been around for a long time.

7.2.2 Single Phase

From equation (7.3)

$$\min_{X\in\mathfrak{X}}\min_{\Delta\in\mathfrak{D}}\sigma(X,\Delta) = \min_{X\in\mathfrak{X}}\left\{\min_{\Delta\in\mathfrak{D}}\sigma(X,\Delta)\right\}. \tag{7.8}$$

So if we define

$$\sigma_\star(X) := \min_{\Delta\in\mathfrak{D}}\sigma(X,\Delta), \tag{7.9}$$

the NMDS problem is to minimize $\sigma_\star$ from (7.9) over X. Note there is a σ defined by equation (7.1) on $\mathfrak{X}\otimes\mathfrak{D}$, and a $\sigma_\star$, defined by equation (7.9), which is a function only of X. It is sometimes said that that Δ is *projected* when going from (7.1) to (7.9), or that $\sigma_\star$ is a *marginal* function.

Once more with feeling. The two-phase σ is a function of two matrix variables X and Δ, the one-phase $\sigma_\star$is a function of the single matrix variable X. To make this even more clear we can write $\sigma_\star(X) = \sigma(X,\Delta(X))$, where

$$\Delta(X) := \underset{\Delta\in\mathfrak{D}}{\text{argmin}}\,\sigma(X,\Delta). \tag{7.10}$$

Of course by projecting out X instead of Δ we could also have defined a loss function which is a function of Δ only, but typically we do not use the alternative projection because it is complicated and heavily nonlinear. Projecting out X is, in fact, solving a standard basic MDS problem. Projecting out Δ is usually much simpler. In most applications $\mathfrak{D}$ is convex, so computing $\Delta(X)$ is computing the projection on a convex set, and projections on convex sets always exist and are unique and continuous.

As an aside, projection creates a function of one variable out of a function of two variables. The inverse of projection is called *augmentation*, which starts with a function f of one variable on $\mathfrak{X}$ and tries to find a function of two variables g on $\mathfrak{X}\otimes\mathfrak{Y}$ such that $f(x) = \min_{y\in\mathfrak{Y}} g(x,y)$. If we have found such a g then we can minimize f over $\mathfrak{X}$ by minimizing g over $\mathfrak{X}\otimes\mathfrak{Y}$, for example by block relaxation (De Leeuw (1994)).

One reason there was some confusion, and some disagreement between Kruskal and Guttman, was a result on differentiation of the minimum function, which was not known in the psychometric community at the time. Guttman thought that $\sigma_\star$ was not differentiable at X, because Δ from (7.10) is a step function. Kruskal proved in Kruskal (1971) that $\sigma_\star$ is differentiable, and saw that the result is basically one in convex analysis, not in classical linear analysis. The result follows easily from directional differentiability in Danskin's theorem (Danskin (1967)) or from the minimax theorems of, for example, Demyanov and Malozemov (1990), using the fact that the projection is unique. More directly, deleeuw_R_73g refers to discussion on page 255 of Rockafellar (1970), following his corollary 26.3.2. We will go into more detail about differentiability, and the differences between Kruskal's and Guttman's loss functions, in the next chapter 10. For now it suffices to note that

$$\mathcal{D}\sigma_\star(X) = \mathcal{D}_1\sigma(X,\Delta(X)), \tag{7.11}$$

or, in words, that the derivative of $\sigma_\star$ at X is the partial derivative of σ at $(X,\Delta(X))$.

7.3 Affine NMDS

Basic MDS can now be interpreted as the special case of NMDS in which $\mathfrak{D} = \{\Delta\}$ is a singleton, a set with only one element. Thus $0 \notin \mathfrak{D}$ and we do

not have to worry about trivial zero solutions for X.

This extends to basic MDS with missing data. We have so far dealt with missing data by setting the corresponding w_{ij} equal to zero. But for the non-missing part we still have fixed numbers in Δ, and thus again $0 \notin \mathfrak{D}$ (unless all dissimilarities are missing). In a sense missing data are our first example of non-metric MDS, because $\mathfrak{D}$ can also be defined as the set

$$\mathfrak{D} = \left\{ \Delta \mid \Delta_0 + \sum_{1\leq i<j\leq n}\sum \{\alpha_{ij}(E_{ij}+E_{ji}) \mid \delta_{ij} \text{ is missing}\} \right\}, \tag{7.12}$$

where the E_{ij} are the unit matrices defined in section @ref(#propmatrix) and Δ_0 is the non-missing part (which has zeroes for the missing elements).

Single/double phase

Another example in which $0 \notin \mathfrak{D}$ is the additive constant problem, which we will discuss in detail in section 8.1. Here $\mathfrak{D}$ is the set of all hollow and symmetric matrices of the form $\Delta + \alpha(E - I)$, where the dissimilarities in Δ_0 are known real numbers and where α is the unknown additive constant.

Affine MDS problems also have single phase and double phase algorithms. For missing data single phase stress is

$$\sigma_\star(X) = \min_{\Delta\in\mathfrak{D}} \sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij} - d_{ij}(X))^2 = \sum_{1\leq i<j\leq n}\sum \tilde{w}_{ij}(\delta_{ij} - d_{ij}(X))^2, \tag{7.13}$$

where $\tilde{w}_{ij} = 0$ if δ_{ij} is missing, and $\tilde{w}_{ij} = w_{ij}$ otherwise. In this case $\sigma_\star(X) = \sigma(X)$, the sigma of basic MDS with zero weights for missing data.

For the additive constant problem single phase stress is

$$\sigma_\star(X) = \min_{\alpha} \sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij}+\alpha-d_{ij}(X))^2 = \sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij}-d_{ij}(X))^2-(\overline{\delta}-\overline{d}(X))^2 \sum_{1\leq i<j\leq n}\sum w_{ij}, \tag{7.14}$$

where $\overline{\delta}$ and $\overline{d}(X)$ are the weighted means of the dissimilarities and distances.

7.4 Conic NMDS

7.4.1 Normalization

In "wide-sense" non-metric MDS $\mathfrak{D}$ can be any set of hollow, non-negative and symmetric matrices. In "narrow-sense" non-metric MDS $\mathfrak{D}$ is defined by

homogeneous linear inequality constraints of the form $\delta_{ij} \leq \delta_{kl}$ (in addition to hollow, non-negative, and symmetric). These constraints, taken together, define a polyhedral convex cone in disparity space. This just means that if Δ_1 and Δ_2 are in $\mathfrak{D}$ then so is $\alpha\Delta_1 + \beta\Delta_2$ for all non-negative α and β.

The disparities define a cone, and thus $0 \in \mathfrak{D}$. This implies that always $\min_X \min_{\Delta\in\mathfrak{D}} \sigma(X, \Delta) = 0$, independently of the data. This is our first example of a *trivial solution*, which have plagued non-metric scaling from the start. Note that $\mathfrak{D}$ for missing data and for the additive constant are not convex cones, and do not contain the zero matrix.

In our versions of *non-metric* MDS we actually require that the transformed dissimilarities satisfy $\eta_\delta = 1$, so that formula (2.6) is still valid. We call this **explicit normalization of the dissimilarities**.

To explain the different forms of normalization of stress that are needed whenever $\mathfrak{D}$ is a cone we look at some general properties of least squares loss functions. More details are in Kruskal and Carroll (1969) and in De Leeuw (1975a), De Leeuw (2019).

Suppose K and L are cones in $\mathbb{R}^n$, nor necessarily convex. Our problem is to minimize $\|x - y\|^2$ over both $x \in K$ and $y \in L$. Here $\|x\|^2 = x'Wx$ for some positive definite W. In the MDS context, for x think disparities, for y think distances.

Of course minimizing $\|x - y\|^2$ is too easy, because $x = y = 0$ is the (trivial, and useless) solution. So we need some form of normalization. We distinguish six different ones.

1. implicit x-normalization
$$\min_{x\in K}\min_{y\in L}\frac{\|x-y\|^2}{\|x\|^2}$$
2. implicit y-normalization
$$\min_{x\in K}\min_{y\in L}\frac{\|x-y\|^2}{\|y\|^2}$$
3. implicit xy-normalization
$$\min_{x\in K}\min_{y\in L}\frac{\|x-y\|^2}{\|x\|^2\|y\|^2}$$

4. explicit x-normalization

$$\min_{x\in K\cap S}\min_{y\in L}\|x-y\|^2$$

5. explicit y-normalization

$$\min_{x\in K}\min_{y\in L\cap S}\|x-y\|^2$$

6. explicit xy-normalization

$$\min_{x\in K\cap S}\min_{y\in L\cap S}\|x-y\|^2$$

   If we use a positive definite W to define our inner products and norms, then implicit normalization of x means

$$\min_{x\in X}\min_{y\in Y}\frac{(x-y)'W(x-y)}{x'Wx}.$$

   Let $\mathcal{S}_x$ and $\mathcal{S}_y$ be the ellipsoids of all x with $x'Wx=1$ and of all y with $y'Wy=1$. Then our implicit normalization problem is equivalent to

$$\min_{\alpha\geq 0}\min_{\beta\geq 0}\min_{x\in X\cap\mathcal{S}_x}\min_{y\in Y\cap\mathcal{S}_y}\frac{(\alpha x-\beta y)'W(\alpha x-\beta y)}{\alpha^2x'Wx}=\min_{x\in X\cap\mathcal{S}_x}\min_{y\in Y\cap\mathcal{S}_y}\min_{\alpha\geq 0}\min_{\beta\geq 0}\frac{\alpha^2+\beta^2-2\alpha\beta x'}{\alpha^2}$$

   Thus implicit normalization of x means maximizing $(x'Wy)^2$ over $x\in X\cap\mathcal{S}_x$ and $y\in Y\cap\mathcal{S}_y$.

In the same way implicit normalization of y minimizes

$$\min_{x\in X}\min_{y\in Y}\frac{(x-y)'W(x-y)}{y'Wy},$$

and in the same way it also leads to maximization of $(x'Wy)^2$ over $x\in X\cap\mathcal{S}_x$ and $y\in Y\cap\mathcal{S}_y$. In terms of normalized stress it does not matter if we use the distances or the dissimilarities in the denominator for implicit normalization.

In explicit normalization of x we solve

$$\min_{x\in X\cap\mathcal{S}_x}\min_{y\in Y}\{1+y'Wy-2y'Wx\}=\min_{\beta\geq 0}\min_{x\in X\cap\mathcal{S}_x}\min_{y\in Y\cap\mathcal{S}_y}\{1+\beta^2-2\beta x'Wy\}=\min_{x\in X\cap\mathcal{S}_x}\min_{y\in Y\cap\mathcal{S}_y}\{1-(x'$$

and the same thing is true for explicit normalization of y, which is

$$\min_{x\in X}\min_{y\in Y\cap\mathcal{S}_y}\{1+x'Wx-2y'Wx\}$$

So, again, it does not matter which one of the four normalizations we use, explicit/implicit on disparities/distances, the solutions will all be proportional to each other, i.e. the same except for scale factors.

7.4.2 Normalized Cone Regression

7.4.3 Hard Squeeze and Soft Squeeze

7.4.4 Inner Iterations

7.4.5 Stress-1 and Stress-2

In his original papers Kruskal (1964a) and Kruskal (1964b) defined two versions of normalized stress for nonmetric MDS. The first was

$$\sigma_{JBK1}(X) := \sqrt{\frac{\sum\sum_{1\leq i<j\leq n}(\hat{d}_{ij} - d_{ij}(X))^2}{\sum\sum_{1\leq i<j\leq n} d_{ij}^2(X)}}$$

$$\sigma_{JBK2}(X) := \sqrt{\frac{\sum\sum_{1\leq i<j\leq n}(\hat{d}_{ij} - d_{ij}(X))^2}{\sum\sum_{1\leq i<j\leq n}(d_{ij}(X) - \overline{d}(X))^2}}$$

where the $hatd_{ij}$ (the d-hats) are the pseudo-distances obtained by projecting the $d_{ij}(X)$ on the isocone defined by the order of the dissimilarities, i.e. by monotone regression (see section 10.1.1). The $\overline{d}(X)$ in the denominator of σ_{JBK2} is the average of the distances.

There are some differences with the definition of stress in this book.

1. We do not use the square root.
2. We use explicit and not implicit normalization.
3. In NMDS we think of stress as a function of both X and Δ, not of X only (see section 10.3).

Chapter 8

Interval MDS

intro: additive vs interval basic vs ratio

8.1 The Additive Constant

8.1.1 Early

In the early history of MDS dissimilarities were computed from comparative judgments in the Thurstonian tradition.

triads paired comparisons etc positive orthant

These early techniques only gave numbers on an interval scale, i.e. dissimilarities known only up to a linear transformation. In order to get positive dissimilarities a rational origin needed to be found in some way. This is the *additive constant problem*. It can be seen as the first example of nonmetric MDS, in which we have only partially known dissimilarities (up to an additive constant).

$$\begin{aligned}(\delta_{ij} + \alpha) &\approx d_{ij}(X), \\ \delta_{ij} &\approx d_{ij}(X) + \alpha.\end{aligned} \tag{8.1}$$

The additive constant techniques were more important in the fifties and sixties than they are these days, because they have largely been replaced by iterative nonmetric MDS techniques.

An early algorithm to fit the additive constant based on Schoenberg's theorem was given by Messick and Abelson (1956). Ii was Torgerson based, i.e. it used the eigenvalues of $\tau(\Delta^{(2)})$. It was a somewhat hopeful iterative technique, without a convergence proof, designed to make the sum of the $n-p$ smallest eigenvalues equal to zero. This is of course only a necessary condition for best approximation, not a sufficient one.

In addition, the Messick-Abelson algorithm sometimes yielded solutionsin which the Torgerson transform of the squared dissimilarities had negative eigenvalues, which could even be quite large. That is also somewhat of a problem.

8.1.2 Cooper

Consequently Cooper (1972) proposed an alternative additive constant algorithm, taking his clue from the work of Kruskal.

The solution was to redefine stress as a function of both the configuration and the additive constant. Thus

$$\sigma(X,\alpha) := \sum_{1\leq j<i\leq n}\sum w_{ij}(\delta_{ij}+\alpha-d_{ij}(X))^2, \tag{8.2}$$

and we minimize this stress over both X and α.

Double phase (ALS)

$\delta_{ij}+\alpha\geq 0$

Single Phase (Cooper)

$$\sigma(X) := \min_{\alpha}\sum_{1\leq j<i\leq n}\sum w_{ij}(\delta_{ij}+\alpha-d_{ij}(X))^2, \tag{8.3}$$

8.2 Algebra

The additive constant problem is to find $X \in \mathbb{R}^{n\times p}$ and α such that $\Delta + \alpha(E-I) \approx D(X)$. In this section we look for all α such that $\Delta + \alpha(E-I)$ is Euclidean, i.e. such that there is a configuration X with $\Delta + \alpha(E-I) = D(X)$. This is a one-parameter generalization of Schoenberg's theorem.

It makes sense to require $\alpha \geq 0$, because a negative α would more appropriately be called a subtractive constant. Also, we may want to make sure that the off-diagonal elements of $\Delta+\alpha(E-I)$ are non-negative, i.e. that $\alpha \geq -\delta_{ij}$ for all $i > j$. Note that if we allow a negative α then if all off-diagonal δ_{ij} are equal, say to $\delta > 0$, we have the trivial solution $\alpha = -\delta$ and $X = 0$.

8.2.1 Existence

We start with a simple construction.

Theorem 8.1. *For all Δ there is an $\alpha_0 \geq 0$ such that for all $\alpha \geq \alpha_0$ we have $\Delta + \alpha(E-I))$ Euclidean of dimension $r \leq n-1$.*

Proof. We have, using $\Delta \times (E-I) = \Delta$ and $(E-I) \times (E-I) = E-I$,

$$\tau((\Delta + \alpha(E-I)) \times (\Delta + \alpha(E-I))) = \tau(\Delta \times \Delta) + 2\alpha\tau(\Delta) + \frac{1}{2}\alpha^2 J. \quad (8.4)$$

Thus each off-diagonal element is a concave quadratic in α, which is negative for α big enough. Choose $\alpha_0 \geq 0$ to make all off-diagonal elements negative (and all dissimilarities non-negative). A doubly-centered matrix with all off-diagonal elements negative is positive semi-definite of rank $n-1$ (Taussky (1949)). □

Note that by the same argument we can also find a negative α_0 that makes all off-diagonal elements negative and thus $\Delta + \alpha(E-I))$ is again Euclidean of dimension $r \leq n-1$. But this α_0 will usually result in negative dissimilarities.

Theorem 8.1 can be sharpened for non-Euclidean Δ. Define the following function of α:

$$\lambda_\star(\alpha) := \min_{x'x=1, x'e=0} x'\{\tau(\Delta \times \Delta) + 2\alpha\tau(\Delta) + \frac{1}{2}\alpha^2 J\}x. \tag{8.5}$$

This is the smallest non-trivial eigenvalue of the Torgerson transform in (8.4). The matrix $\Delta + \alpha(E - I)$ is Euclidean if and only if $\lambda_\star(\alpha) \geq 0$. Note that $\lambda_\star$ is continuous, by a simple special case of the Maximum Theorem (Berge (1963), Chapter VI, section 3), and coercive, i.e. $\lambda_\star(\alpha) \to +\infty$ if $|\alpha| \to +\infty$.

Theorem 8.2. *For all non-Euclidean Δ there is an $\alpha_1 > 0$ such that for all $\alpha \geq \alpha_1$ we have that $\Delta + \alpha(E - I))$ Euclidean of dimension $r \leq n - 2$.*

Proof. Because Δ is non-Euclidean we have $\lambda_\star(0) < 0$. By the construction in theorem 8.1 there is an α_0 such that $\lambda_\star(\alpha) > 0$ for all $\alpha > \alpha_0$. By the Maximum Theorem the function $\lambda_\star$ is continuous, and thus, by Bolzano's theorem, there is an α_1 between 0 and α_0 such that $\lambda_\star(\alpha_1) = 0$. If there is more than one zero between 0 and α_0 we take the largest one as α_1. □

The problem with extending theorem 8.2 to Euclidean Δ is that the equation $\lambda_\star(\alpha) = 0$ may have only negative roots, or, even more seriously, no roots at all. This may not be too important from the practical point of view, because observed dissimilarities will usually not be exactly Euclidean. Nevertheless I feel compelled to address it.

Theorem 8.3. *If Δ is Euclidean then $\lambda_\star(\alpha)$ is non-negative and non-decreasing on $[0, +\infty)$.*

Proof. If Δ is Euclidean, then $\sqrt{\Delta}$, which is short for the matrix with the square roots of the dissimilarities, is Euclidean as well. This follows because the square root is a Schoenberg transform (Schoenberg (1937), Bavaud (2011)), and it implies that $\tau(\Delta) = \tau(\sqrt{\Delta} \times \sqrt{\Delta})$ is positive semi-definite. Thus the matrix (8.4) is positive semi-definite for all $\alpha \geq 0$. By Danskin's Theorem the one-sided directional derivative of $\lambda_\star$ at α is $2x(\alpha)'\tau(\Delta)x(\alpha)+\alpha$, where $x(\alpha)$ is one of the minimizing eigenvectors. Because the one-sided derivative is non-negative, the function is non-decreasing (in fact increasing if $\alpha > 0$). □

Of course $\lambda_\star(\alpha) = 0$ can still have negative solutions, and in particular it will have at least one negative solution if $\lambda_\star(\alpha) \leq 0$ for any α. There can even be negative solutions with $\Delta + \alpha(E - I)$ non-negative.

8.2.2 Solution

The solutions of $\lambda_\star(\alpha) = 0$ can be computed and studied in more detail, using results first presented in the psychometric literature by Cailliez (1983). We reproduce his analysis here, with a somewhat different discussion that relies more on existing mathematical results.

In order to find the smallest α we solve the quadratic eigenvalue problem (Tisseur and Meerbergen (2001)). WHY ??

$$\{\tau(\Delta \times \Delta) + 2\alpha\tau(\Delta) + \frac{1}{2}\alpha^2 J\}y = 0. \tag{8.6}$$

A solution (y, α) of #ref(eq:qep1) is an eigen pair, in which y is an eigenvector, and α the corresponding eigenvalue. The trivial solution $y = e$ satisfies #ref(eq:qep1) for any α. We are not really interested in the non-trivial eigenvectors here, but we will look at the relationship between the eigenvalues and the solutions of $\lambda_\star(\alpha) = 0$.

The eigenvalues can be complex, in which case they do not interest us. If α is a non-trivial real eigenvalue, then the rank of the Torgerson transform of the matrix in #ref(eq:qep1) is $n - 2$, but

To get rid of the annoying trivial solution $y = e$ we use a square orthonormal matrix whose first column is proportional to e. Suppose L contains the remaining $n - 1$ columns. Now solve

$$\{L'\tau(\Delta \times \Delta)L + 2\alpha L'\tau(\Delta)L + \frac{1}{2}\alpha^2 I\}y = 0. \tag{8.7}$$

Note that the determinant of the polynomial matrix in (8.7) is a polynomial of degree $2(n-1)$ in α, which has $2(n-1)$ real or complex roots.

The next step is linearization (Gohberg, Lancaster, and Rodman (2009), chapter 1), which means finding a linear or generalized linear eigen problem with the same roots as (8.7). In our case this is the eigenvalue problem for the matrix

$$\begin{bmatrix} 0 & I \\ -2L'\tau(\Delta \times \Delta)L & -4L'\tau(\Delta)L \end{bmatrix} \tag{8.8}$$

8.2.3 Examples

8.2.3.1 Small Example

Here is a small artificial dissimilarity matrix.

```
##   1  2  3  4
## 1 +0 +1 +2 +5
## 2 +1 +0 +4 +2
## 3 +2 +4 +0 +1
## 4 +5 +2 +1 +0
```

It is constructed such that $\delta_{14} > \delta_{12} + \delta_{24}$ and that $\delta_{23} > \delta_{21} + \delta_{13}$. Because the triangle inequality is violated the dissimilarities are not distances in any metric space, and certainly not in a Euclidean one. Because the minimum dissimilarity is $+1$, we require that the additive constant α is at least -1.

The R function treq() in appendix A.1.10 finds the smallest additive constant such that all triangle inequalities are satisfied. For this example it is $\alpha = 2$.

The Torgerson transform of $\Delta \times \Delta$ is

```
##   1      2      3      4
## 1 +4.312 +2.688 +1.188 -8.188
## 2 +2.688 +2.062 -5.938 +1.188
## 3 +1.188 -5.938 +2.062 +2.688
## 4 -8.188 +1.188 +2.688 +4.312
```

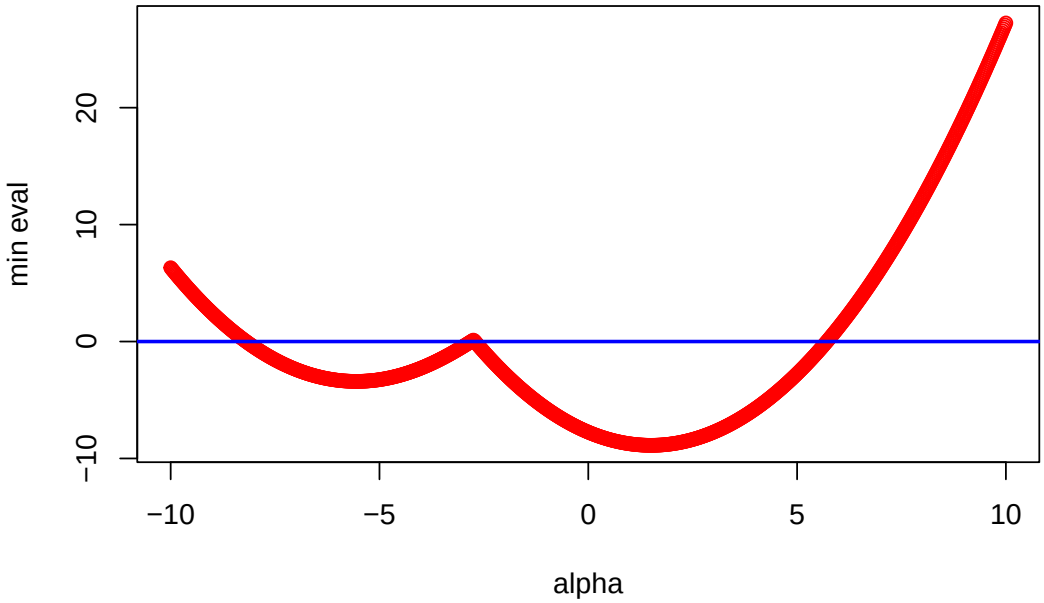
with eigenvalues

```
## [1] +12.954 +7.546  +0.000  -7.750
```

The smallest eigenvalue -7.75 is appropriately negative, and theorem 8.2 shows that $\Delta \times \Delta + 7.75(E - I)$ are squared distances between four points in the plane.

The upper bound for the smallest α from theorem 8.1, computed by the R function acbound(), is 9.309475.

It is useful to look at a graphical representation of the minimum non-trivial eigenvalue of $\tau((\Delta+\alpha(E-I))\times(\Delta+\alpha(E-I)))$ as a function of α. The R function aceval() generates the data for the plot.



We see that the minimum non-trivial eigenvalue is a continuous function of α,but one which certainly is not convex or concave or differentiable. The graph crosses the horizontal axes near -8, -3, and +6.

To make this precise we apply the theory of section xxx. The R function acqep() finds the six non-trivial eigenvalues

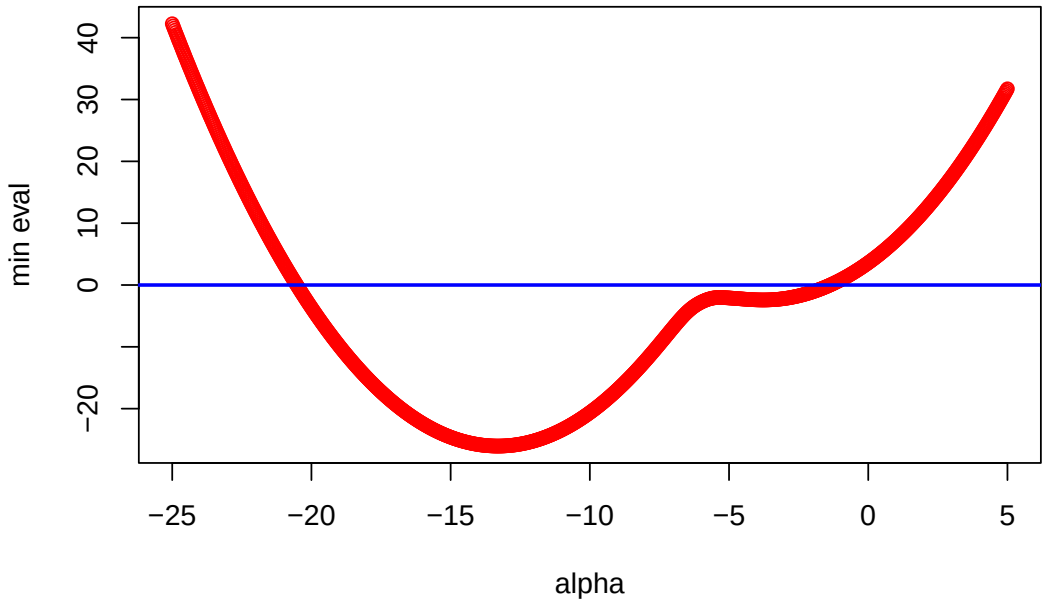
```
## [1] -8.192582+0.000000i  5.713075+0.000000i -3.500000+2.179449i
## [4] -3.500000-2.179449i -2.807418+0.000000i -2.713075+0.000000i
```

Two of the eigenvalues are complex conjugates, four are real. Of the real eigenvalues three are negative, and only one is positive, equal to +5.713. The table above gives the eigenvalues of the Torgerson transform, using all four real eigenvalues for α. The three negative ones do result in a positive semi-definite matrix with rank equal to $n-2$, but they also create negative dissimilarities.

```
##  -8.193  ******  +38.098 +13.885  +0.000  -0.000
```

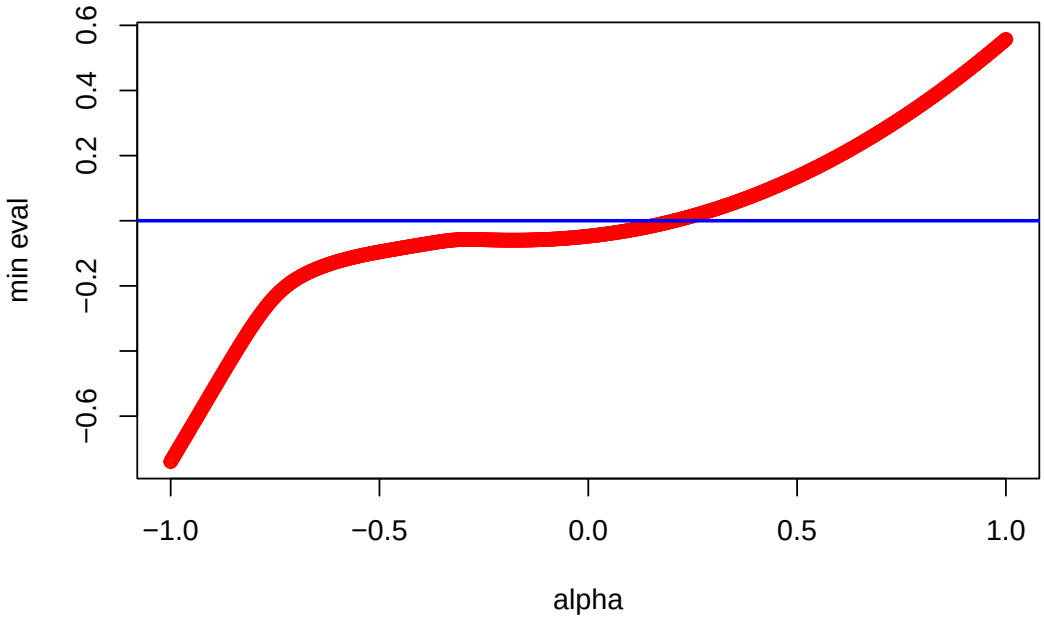
```
##  +5.713  ******  +61.116 +43.441  +0.000  -0.000
##  -2.807  ******   +3.115  +0.402  -0.000  -0.000
##  -2.713  ******   +3.228  +0.215  +0.000  +0.000
```

8.2.3.2 De Gruijter Example

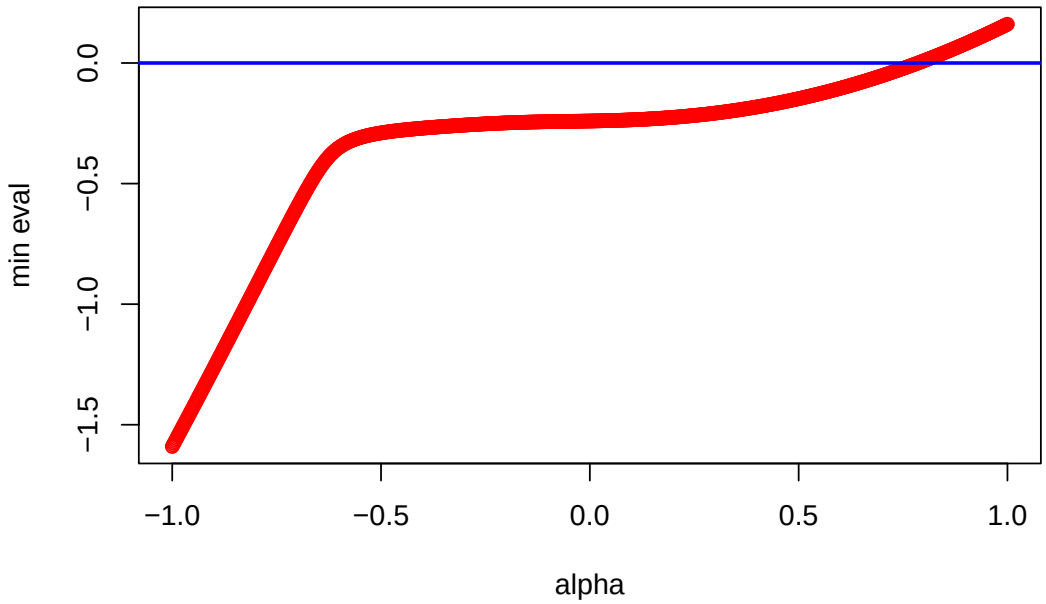



```
##  [1] -20.527411+0.000000i -10.174103+0.000000i  -9.472504+0.000000i
##  [4]  -6.622263+0.352619i  -6.622263-0.352619i  -5.885691+0.287544i
##  [7]  -5.885691-0.287544i  -5.640580+0.366889i  -5.640580-0.366889i
## [10]  -4.391289+0.253248i  -4.391289-0.253248i  -3.708911+0.386844i
## [13]  -3.708911-0.386844i  -3.238930+0.000000i  -2.311379+0.000000i
## [16]  -1.369315+0.000000i
```

8.2.3.3 Ekman Example




```
##  [1] -5.713009655+0.0000000i -3.782729083+0.0000000i -1.791313475+0.0000000i
##  [4] -1.628964140+0.0000000i -0.976213035+0.0000000i -0.744289350+0.0495939i
##  [7] -0.744289350-0.0495939i -0.682321433+0.0000000i -0.534849034+0.0000000i
## [10] -0.513033529+0.0000000i -0.497908376+0.0248145i -0.497908376-0.0248145i
## [13] -0.372321687+0.1313892i -0.372321687-0.1313892i -0.388308013+0.0000000i
## [16] -0.229813135+0.1825985i -0.229813135-0.1825985i -0.286712033+0.0000000i
## [19] -0.212601059+0.1185199i -0.212601059-0.1185199i  0.206312577+0.0000000i
## [22] -0.194299448+0.0000000i  0.132767430+0.0000000i -0.079646956+0.0000000i
## [25] -0.024193535+0.0000000i -0.006762279+0.0000000i
```




```
##  [1] -7.9740652+0.0000000i -4.8679294+0.0000000i -1.2242342+0.0000000i
##  [4] -0.9822376+0.0000000i  0.7856448+0.0000000i  0.6486775+0.0000000i
##  [7] -0.5540332+0.0000000i -0.5426006+0.0129703i -0.5426006-0.0129703i
## [10]  0.4864182+0.0000000i -0.1118921+0.3923586i -0.1118921-0.3923586i
## [13]  0.3829746+0.0000000i  0.3530897+0.0000000i -0.3516103+0.0000000i
## [16] -0.3073607+0.0000000i -0.1265941+0.2630789i -0.1265941-0.2630789i
## [19] -0.0735448+0.2698631i -0.0735448-0.2698631i -0.2333027+0.0000000i
## [22] -0.0081376+0.1948006i -0.0081376-0.1948006i -0.1380254+0.1175071i
## [25] -0.1380254-0.1175071i  0.1205026+0.0000000i
```

8.2.4 A Variation

Alternatively, we could define our approximation problem as finding $X \in \mathbb{R}^{n\times p}$ and α such that $\sqrt{\delta_{ij}^2 + \alpha} \approx d_{ij}(X)$, or, equivalently, $\Delta\times\Delta+\alpha(E-I) \approx D(X) \times D(X)$.

Theorem 8.4. *For any $X \in \mathbb{R}^{n\times p}$ with $p = n - 2$ there is an α such that $\sqrt{\delta_{ij}^2 + \alpha} = d_{ij}(X)$.*

Proof. Now we have

$$\tau(\Delta \times \Delta + \alpha(E - I))) = \tau(\Delta \times \Delta) + \frac{1}{2}\alpha J. \tag{8.9}$$

The eigenvalues of $\tau(\Delta \times \Delta) + \frac{1}{2}\alpha J$ are zero and $\lambda_s + \frac{1}{2}\alpha$, where the λ_s are the $n-1$ non-trivial eigenvalues of $\tau(\Delta \times \Delta)$. If $\underline{\lambda}$ is smallest eigenvalue we choose $\alpha = -2\underline{\lambda}$, and $\tau(\Delta \times \Delta) + \frac{1}{2}\alpha J$ is positive semi-definite of rank $r \leq n-2$. □

Note that theorem 8.2 implies that for any Δ there is a strictly increasing differentiable transformation to the space of Euclidean distance matrices in $n-2$ dimensions. This is a version of what is sometimes described as *Guttman's n-2 theorem* (Lingoes (1971)). The proof we have given is that from De Leeuw (1970), Appendix B.

8.3 Interval smacof

In this section we introduce a double-phase alternating least squares algorithm that fits better into the smacof framework than the single-phase method proposed by Cooper (1972). We also restrict our linear transformations to be to be increasing and non-negative on the positive real axes.

To avoid various kinds of trivialities, assume not all $d_{ij}(X)$ are zero.

In the optimal scaling phase we must minimize

$$\sigma(X, \alpha, \beta) = \sum_{1 \leq i < j \leq n} \sum w_{ij}(\alpha\delta_{ij} + \beta - d_{ij}(X))^2 \tag{8.10}$$

The constraints are $\alpha\delta_{ij} + \beta \geq 0$ and $\alpha\delta_{ij} + \beta \geq \alpha\delta_{kl} + \beta$ if $\delta_{ij} \geq \delta_{kl}$. These define pointed convex cone in the space of disparities. We need to project $D(X)$ on that cone, in the metric defined by W. But it is easy to see that and equivalent set of constraints in $\mathbb{R}^2$ is $\alpha \geq 0$ and $\alpha\delta_{\min} + \beta \geq 0$. Again these two constraints define a pointed cone in two-dimensional (α, β) space, where proje ction is much easier to handle thanin the generally much larger disparity space. Of course the projection metric in (α, β) is different from the one in disparity space.

In addition to the inequality constraints we have the normalization constraint

$$\sum_{1\leq i<j\leq n}\sum w_{ij}(\alpha\delta_{ij}+\beta)^2=1, \tag{8.11}$$

but as we have seen in chapter 7 we can initially ignore that constraint, project on the cone, and then normalize the projection.

In order to simplify the notation we collect the $d_{ij}(X)$ in a vector d, the δ_{ij} in a vector δ and the w_{ij} in a diagonal matrix W.

Let's first get the trivial case where all δ_{ij} are equal out of the way. In that case the linear regression is singular, and we simply choose all $\alpha\delta+\beta$ equal to the constant $e'Wd$, for example by setting $\alpha=0$ and $\beta=e'Wd$. Applying the normalization condition (8.11) then sets $\beta=1$. From now on we assume in this section that not all δ_{ij} are equal.

Projecting on the cone gives us four possibilities. We can have $\alpha=0$ or $\alpha\delta_{\min}+\beta=0$, or both, or neither. We first analyze the case in which the unconstrained minimum of (8.10) is in the cone, which will be the most common case, especially in later smacof iterations. Using the fact that $\delta'W\delta=e'We=1$ we find that

$$\begin{bmatrix}\tilde{\alpha}\\ \tilde{\beta}\end{bmatrix}=\frac{1}{1-(e'W\delta)^2}\begin{bmatrix}\delta'(W-Wee'W)d\\ e'(W-W\delta\delta'W)d\end{bmatrix}. \tag{8.12}$$

If $\tilde{\alpha}\geq 0$ and $\tilde{\beta}\geq-\alpha\delta_{\min}$ we are done. If not, we know the projection is on the line $\alpha=0$ or on the line $\tilde{\beta}=-\alpha\delta_{\min}$, or on their intersection, which is the origin.

First suppose the projection is on $\alpha=0$. We find the minimizing β equal to $\overline{\beta}:=e'Wd$, which strictly satisfies the second constraint because $\overline{\beta}>-\alpha\delta_{\min}=0$, and thus $(0,e'Wd)$ is on the boundary of the cone. This also show that the origin, which has $\sigma(X,0,0)=d'Wd$, can never be the projection. The minimum at $(0,e'Wd)$ is

$$\sigma(X,0,e'Wd)=d'Wd-(d'We)^2 \tag{8.13}$$

Or, alternatively, we can assume that the projection is on the vertex $\beta=-\alpha\delta_{\min}$, in which case the minimizing α is

$$\overline{\alpha}:=\frac{(\delta-\delta_{\min})'Wd}{(\delta-\delta_{\min})'W(\delta-\delta_{\min})}, \tag{8.14}$$

which is always positive, and thus $(\overline{\alpha}, -\overline{\alpha}\delta_{\min})$ is on the boundary of the cone. The minimum is

$$\sigma(X, \overline{\alpha}, -\overline{\alpha}\delta_{\min}) = d'Wd - \frac{((\delta - \delta_{\min})'Wd)^2}{(\delta - \delta_{\min})'W(\delta - \delta_{\min})} \tag{8.15}$$

If the unconstrained solution is not in the cone, then we choose the projection as the solution corresponding with the smallest of (8.13) and ((8.15).

8.3.1 Example

We illustrate finding the optimal linear transformation with a small example. We choose some arbitrary w, δ, and d and normalize them in the usual way.

```
w <- c(rep(1,5),rep(2,5))
w <- w / sum(w)
delta <- 1:10
s <- sum(w * delta ^ 2)
delta <- delta / sqrt (s)
d <- c(1, 2, 3, 4, 4, 3, 3, 3, 1, 1)
s <- sum (w * d ^ 2)
t <- sum (w * d * delta)
d <- d * (t / s)
```

After normalization the $\delta_{\min}$ is 0.1448414. The pink region in figure 8.1 is the cone formed by the intersection of the half-spaces $\alpha \geq 0$ and $\alpha\delta_{\min} + \beta \geq 0$.

The unconstrained minimum is attained at -0.2541855, 0.9484652, the red point in figure 8.1, with stress equal to 0.0939827. That is clearly outside the cone, so we now consider projection on the two one-dimensional boundary rays. The blue point is 0, 0.7152935, the projection on $\alpha \geq 0$. It is fairly close to the unconstrained minimum, with stress 0.1042239. The green point 0.6782764, -0.0982425is the projection on $\beta = -\alpha\delta_{\min}$, which has stress 0.2684117. Thus the blue point 0, 0.7152935is the actual projection on the cone in (α, β) space, and the best fitting line has slope zero (which, in smacof, would make all disparities equal for the next iteration).

This is illustrated in a different way (with Shepard plots) in figure 8.2, where we see the red, blue, and green lines corresponding with the red, blue, and

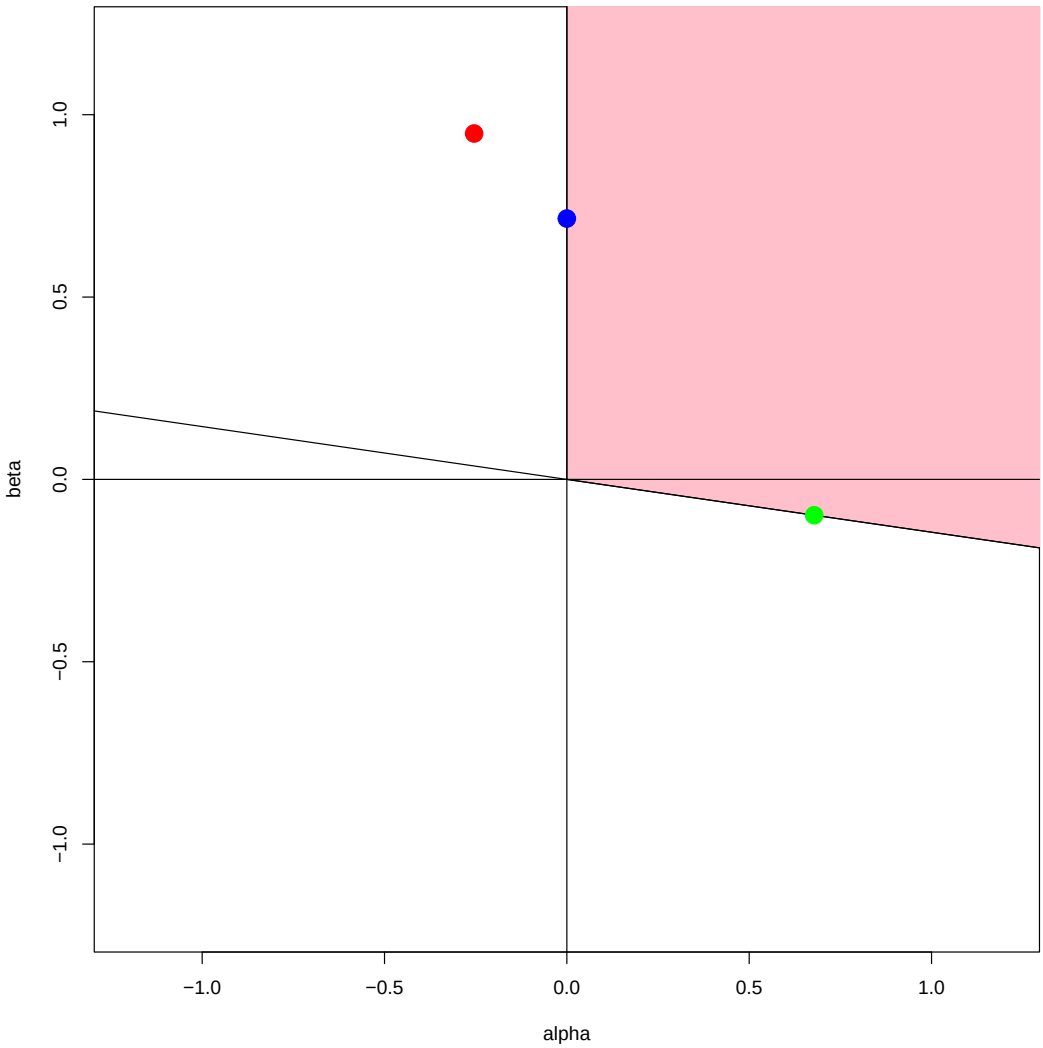


Figure 8.1: Cone Projection

green points in figure 8.1. Note that the green line goes through the point $(\delta_{\min}, 0)$. The horizontal blue line is the best fitting one under the constraints.

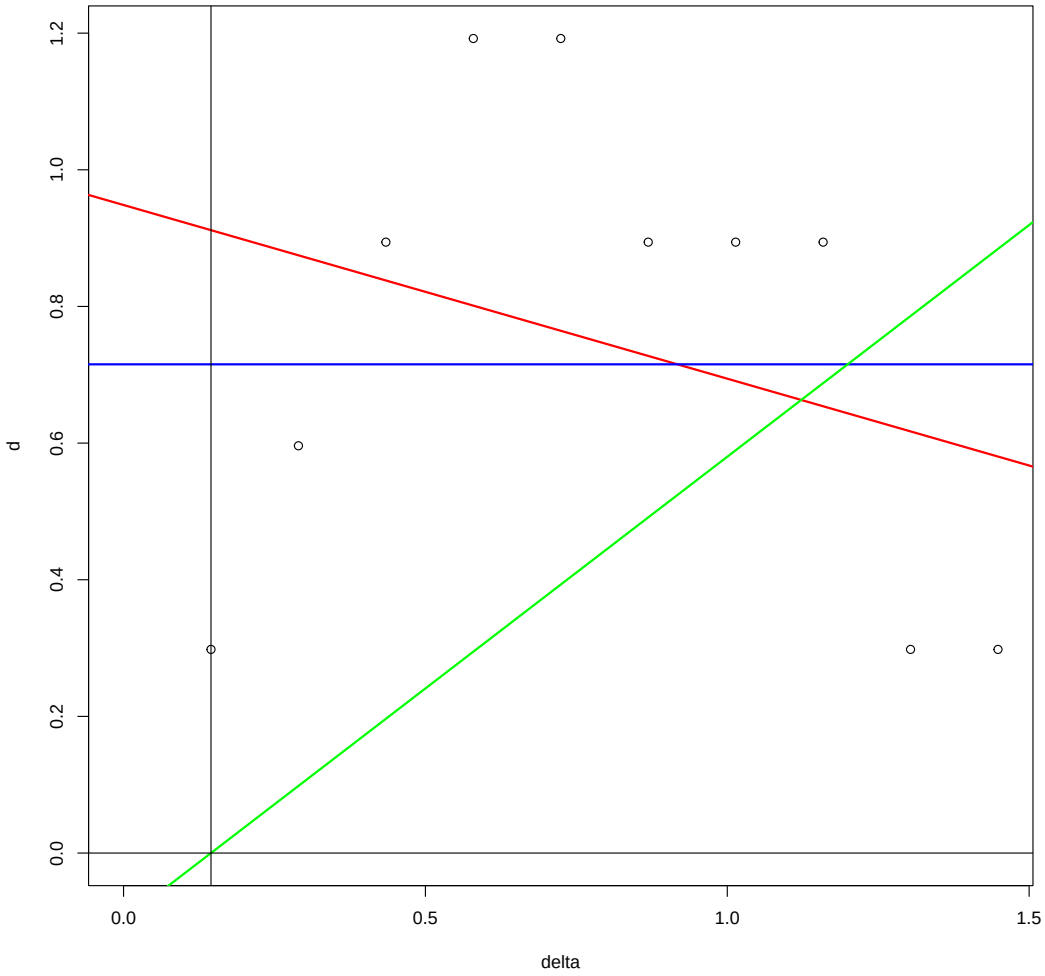


Figure 8.2: Fitted Lines

Chapter 9

Polynomial MDS

9.1 Introduction

9.2 Fitting Polynomials

$$\sigma(X) = \sum_{1 \le i < j \le n} \sum w_{ij}(P_r(\delta_{ij}) - d_{ij}(X))^2$$

$$P_r(\delta_{ij}) := \sum_{s=0}^{r} \alpha_s \delta_{ij}^s.$$

The polynomial P_r is *tied down* if $\alpha_0 = 0$, and thus $P_r(0) = 0$.

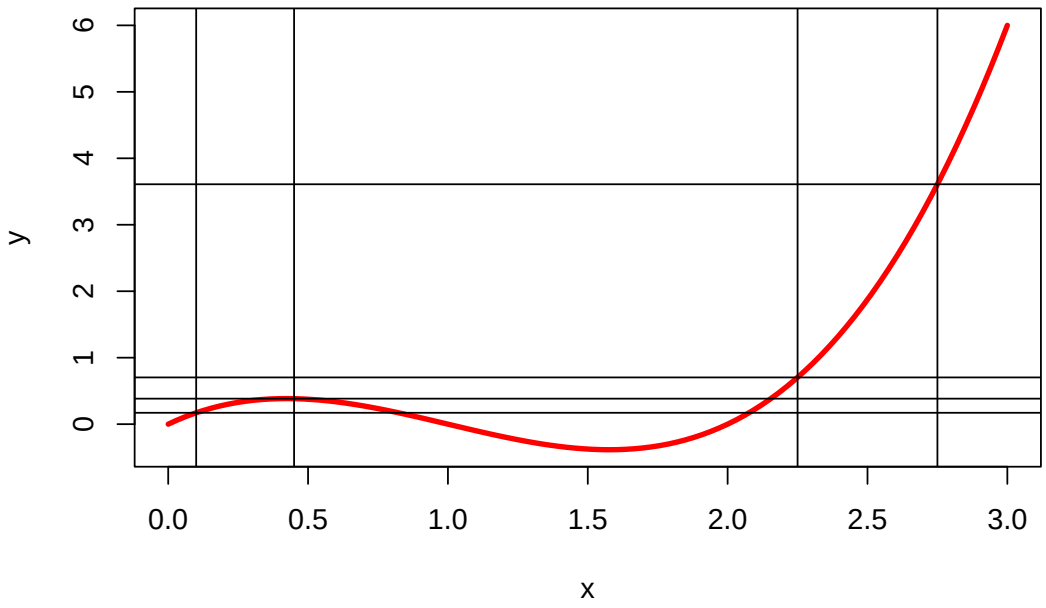
Vandermonde matrix

9.3 Positive and Convex, Monotone Polynomials

9.3.1 Introduction

Constraints on values, constraints on coefficients

```
f <- function(x) return(x * (x - 1) * (x - 2))
x <- seq(0, 3, length = 100)
y <- f(x)
plot(x, y, type="l", lwd = 3, col = "RED")
x <- c(.10, .45, 2.25, 2.75)
y <- f(x)
abline(h = y[1])
abline(h = y[2])
abline(h = y[3])
abline(h = y[4])
abline(v = x[1])
abline(v = x[2])
abline(v = x[3])
abline(v = x[4])
```



Tied-down increasing non-negative cubic.

Write the tied-down cubic in the form $f(x) = ax(x^2+2bx+c)$. Since we must have $f(x) \to +\infty$ of $x \to +\infty$ we have $a > 0$. Since f must be increasing at zero, we must have $f'(0) = c > 0$. No real roots on the positive reals. Case 1: no real roots at all $b^2 - c < 0$. Case 2: two real roots, both negative $b^2 - c \geq 0$ and $b \geq 0$. Since $c > 0$ the product of roots is positive. If the sum

is negative, i.e. if $b > 0$ both roots are negative. Since $f''(x) = 6x + b$ we see that f is convex on the positive real axis if $b \geq 0$.

9.3.2 A QP Algorithm

In this section we construct an algorithm for a general weighted linear least squares projection problem with equality and/or inequality constraints. It uses duality and unweighting majorization. The section takes the form of a small diversion, with examples. This may seem somewhat excessive, but it provides an easy reference for both you and me and it serves as a manual for the corresponding R code.

We start with the *primal problem*, say problem $\mathcal{P}$, which is minimizing

$$f(x) = \frac{1}{2}(Hx - z)'V(Hx - z) \tag{9.1}$$

over all x satisfying equalities $Ax \geq b$ and equations $Cx = d$. We suppose the *Slater condition* is satisfied, i.e. there is an x such that $Ax > b$. And, in addition, we suppose the system of inequalities and equations is *consistent*, i.e. has at least one solution.

We first reduce the primal problem to a simpler, and usually smaller, one by partitioning the loss function. Define

$$\begin{aligned} W &:= H'VH, \\ y &:= W^{-1}H'Vz, \\ Q &:= (I - H(H'VH)^{-1}H'V). \end{aligned} \tag{9.2}$$

Then

$$(Hx - y)'V(Hx - y) = (x - y)'W(x - y) + y'Q'VQy, \tag{9.3}$$

The simplified primal problem $\mathcal{P}'$ is to minimize $(x-y)'W(x-y)$ over $Ax \geq b$ and $Cx = d$, where W is assumed to be positive definite. Obviously the solutions to $\mathcal{P}$ and $\mathcal{P}'$ are the same. The two loss function values only differ by the constant term $y'Q'VQy$.

We do not solve $\mathcal{P}'$ drectly, but we use Lagrangian duality and solve the dual quadratic programmng problem. The Lagrangian for $\mathcal{P}'$ is

$$\mathcal{L}(x, \lambda, \mu) = \frac{1}{2}(x - y)'W(x - y) - \lambda'(Ax - b) - \mu'(Cx - d), \tag{9.4}$$

where $\lambda \geq 0$ and μ are the Lagrange multipliers.

Now

$$\max_{\lambda \geq 0} \max_{\mu} \mathcal{L}(x, \lambda, \mu) = \\ = \begin{cases} \frac{1}{2}(x - y)'W(x - y) - \lambda'(Ax - b) - \mu'(Cx - d) & \text{if } Ax \geq b, \\ +\infty & \text{otherwise,} \end{cases} \tag{9.5}$$

and thus

$$\min_{x} \max_{\lambda \geq 0} \max_{\mu} \mathcal{L}(x, \lambda, \mu) = \min_{Ax \geq b} \min_{Cx = d} \frac{1}{2}(x - y)'W(x - y), \tag{9.6}$$

which is our original simplified primal problem $\mathcal{P}'$.

We now look at the *dual problem* $\mathcal{D}'$ (of $\mathcal{P}'$), which means solving

$$\max_{\lambda \geq 0} \max_{\mu} \min_{x} \mathcal{L}(x, \lambda, \mu). \tag{9.7}$$

The inner minimum over x for given λ and μ is attained at

$$x = y + W^{-1}(A' \mid C') \begin{bmatrix} \lambda \\ \mu \end{bmatrix}, \tag{9.8}$$

and is equal to $-g(\lambda, \mu)$, where

$$\frac{1}{2} \begin{bmatrix} \lambda & \mu \end{bmatrix} \begin{bmatrix} AW^{-1}A' & AW^{-1}C' \\ CW^{-1}A' & CW^{-1}C' \end{bmatrix} \begin{bmatrix} \lambda \\ \mu \end{bmatrix} + \ + \lambda'(Ay - b) + \mu'(Cy - d) \tag{9.9}$$

Our strategy is to solve $\mathcal{D}'$ for $\lambda \geq 0$ and/or μ. Because of our biases we do not maximize $-g$, we minimize g. Then compute the solution of both $\mathcal{P}'$ and $\mathcal{P}$ from (9.8). The duality theorem for quadratic programming tells us the values of f at the optimum of $\mathcal{P}'$ and $-g$ at the optimum of $\mathcal{D}'$ are equal, and of course the value at the optimum of $\mathcal{P}$ is that of $\mathcal{P}'$ plus the constant $y'QVQy$.

From here on we can proceed with unweighting in various ways. We could, for instance, minimize out μ and then unweight the resulting quadratic form. Instead, we go the easy way. Majorize the partitioned matrix K in the quadratic part of (9.9) by a similarly partitioned diagonal positive matrix E.

$$E := \begin{bmatrix} F & \emptyset \\ \emptyset & G \end{bmatrix} \gtrsim K := \begin{bmatrix} AW^{-1}A' & AW^{-1}C' \\ CW^{-1}A' & CW^{-1}C' \end{bmatrix} \tag{9.10}$$

Suppose $\tilde{\lambda} \geq 0$ and $\tilde{\mu}$ are the current best solutions of the dual problem. Put them on top of each other to define $\tilde{\gamma}$, and do the same with λ and μ to get γ. Then $g(\lambda, \mu)$ becomes

$$\frac{1}{2}(\tilde{\gamma}+(\gamma-\tilde{\gamma}))'E(\tilde{\gamma}+(\gamma-\tilde{\gamma}))+\gamma'(Ry-e) = \ = \frac{1}{2}(\gamma-\tilde{\gamma})'E(\gamma-\tilde{\gamma})+(\gamma-\tilde{\gamma})'E(\tilde{\gamma}+(Ry-e))++\frac{1}{2}\tilde{\gamma}'E\tilde{\gamma}+\tilde{\gamma}'(Ry- \tag{9.11}$$

The last two terms do not depend on γ, so for the majorization algorithm is suffices to minimize

$$\frac{1}{2}(\gamma - \tilde{\gamma})'F(\gamma - \tilde{\gamma}) + (\gamma - \tilde{\gamma})'E(\tilde{\gamma} + (Ry - e)) \tag{9.12}$$

Let

$$\xi := \tilde{\gamma} - F^{-1}E(\tilde{\gamma} + (Ry - e)) \tag{9.13}$$

then (9.12) becomes

$$\frac{1}{2}(\gamma - \xi)'F(\gamma - \xi) - \frac{1}{2}\xi'F\xi \tag{9.14}$$

Because F is diagonal $\lambda_i = \max(0, \xi_i)$ for $i = 1, \cdots m_1$ and and $\mu_i = \xi_{i+m_1}$ for $i = 1, \cdots m_2$.

Section A.1.19 has the R code for qpmaj(). The defaults are set to do a simple isotone regression, but of course the function has a much larger scope. It can handle equality constraints, linear convexity constraints, partial orders, and much more general linear inequalities. It can fit polynomials, monotone polynomials, splnes, and monotone splines of various sorts. It is possible to have only inequality constraints, only equality constraints, or both. The matrix H of predictors in (9.1) can either be there or not be there.

The function qpmaj() returns both x and λ, and the values of $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{D}'$. And also the *predicted values* Hx, and the *constraint values* $Ax - b$ and $Cx - d$, if applicable. It's always nice to check *complimentary slackness* $\lambda'(Ax - b) = 0$, and another check is provided because the values of $\mathcal{P}'$ and $\mathcal{D}'$ must be equal. Finally qpmaj() returns the number of iterations for the dual problem.

The function qpmaqj() does not have the pretense to compete in efficiency with the sophisticated pivoting and active set strategies for quadratic programming discussed for example by Best (2017). But it seems to do a reliable job on our small examples, and it is an interesting example of majorization and unweighting.

9.3.2.1 Example 1: Simple Monotone Regression

Here are the two simple monotone regression examples from section 10.1.1, the first one without weights and the second one with a diagonal matrix of weights.

```
y<-c(1,2,1,3,2,-1,3)
qpmaj(y)
```

```
## $x
## [1] 1.0 1.4 1.4 1.4 1.4 1.4 3.0
##
## $fprimal
## [1] 4.6
```

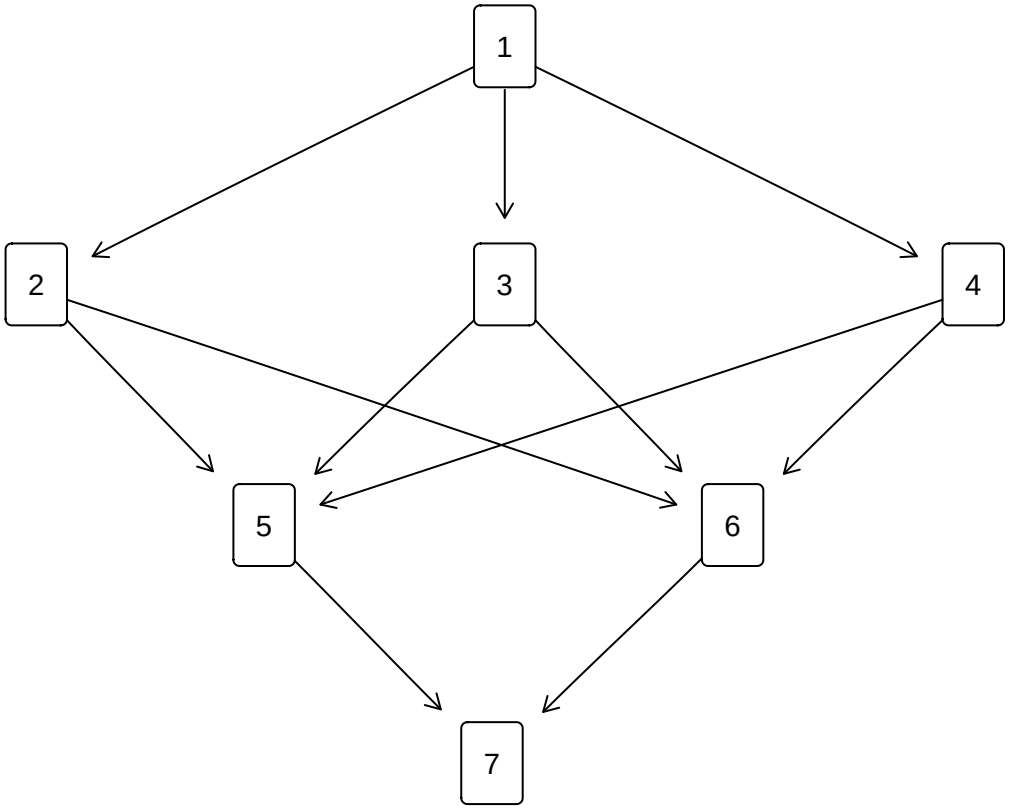
```
##
## $fdual
## [1] 4.6
##
## $lambda
## [1] 0.0000000 0.5999999 0.1999999 1.7999999 2.3999999 0.0000000
##
## $inequalities
## [1]  4.000001e-01 -2.760349e-08 -4.466339e-08 -4.466339e-08 -2.760349e-08
## [6]  1.600000e+00
##
## $itel
## [1] 146
```

```
qpmaj(y, v = diag(c(1,2,3,4,3,2,1)))
```

```
## $x
## [1] 1.000000 1.400000 1.400000 1.777778 1.777778 1.777778 3.000000
##
## $fprimal
## [1] 11.37778
##
## $fdual
## [1] 11.37778
##
## $lambda
## [1] 0.000000 1.200000 0.000000 4.888889 5.555555 0.000000
##
## $inequalities
## [1]  4.000000e-01  0.000000e+00  3.777778e-01 -3.451610e-08 -2.391967e-08
## [6]  1.222222e+00
##
## $itel
## [1] 82
```

9.3.2.2 Example 2: Monotone Regression with Ties

Now suppose the data have tie-blocks, which we indicate with $\{1\} \leq \{2,3,4\} \leq \{5,6\} \leq \{7\}$. The Hasse diagram of the partial order (courtesy of Ciomek (2017)) is



In the primary approach to ties the inequality constraints $Ax \geq 0$ are coded with A equal to

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7]
##  [1,]   -1   +1   +0   +0   +0   +0   +0
##  [2,]   -1   +0   +1   +0   +0   +0   +0
##  [3,]   -1   +0   +0   +1   +0   +0   +0
##  [4,]   +0   -1   +0   +0   +1   +0   +0
##  [5,]   +0   -1   +0   +0   +0   +1   +0
```

```
##  [6,]   +0   +0   -1   +0   +1   +0   +0
##  [7,]   +0   +0   -1   +0   +0   +1   +0
##  [8,]   +0   +0   +0   -1   +1   +0   +0
##  [9,]   +0   +0   +0   -1   +0   +1   +0
## [10,]   +0   +0   +0   +0   -1   +0   +1
## [11,]   +0   +0   +0   +0   +0   -1   +1
```

Applying our algorithm gives

```
qpmaj(y, a = a)
```

```
## $x
## [1] 1.000000 1.333333 1.000000 1.333333 2.000000 1.333333 3.000000
##
## $fprimal
## [1] 4.333333
##
## $fdual
## [1] 4.333333
##
## $lambda
##  [1] 0.000000e+00 2.069906e-15 0.000000e+00 0.000000e+00 6.666667e-01
##  [6] 0.000000e+00 0.000000e+00 0.000000e+00 1.666667e+00 0.000000e+00
## [11] 0.000000e+00
##
## $inequalities
##  [1]  3.333333e-01  4.107825e-15  3.333334e-01  6.666667e-01  8.182895e-08
##  [6]  1.000000e+00  3.333333e-01  6.666666e-01 -8.182895e-08  1.000000e+00
## [11]  1.666667e+00
##
## $itel
## [1] 96
```

In the secondary approach we require $Cx = 0$, with C equal to

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
```

```
## [1,]   +0   +1   -1   +0   +0   +0   +0
## [2,]   +0   +1   +0   -1   +0   +0   +0
## [3,]   +0   +0   +0   +0   +1   -1   +0
```

In addition we construct A to require $x_1 \leq x_2 \leq x_5 \leq x_7$. This gives

```
qpmaj(y, a = a, c = c)
```

```
## $x
## [1] 1.0 1.4 1.4 1.4 1.4 1.4 3.0
##
## $fprimal
## [1] 4.6
##
## $fdual
## [1] 4.6
##
## $lambda
## [1] 0.0 1.8 0.0
##
## $inequalities
## [1]  4.000000e-01 -5.100425e-08  1.600000e+00
##
## $mu
## [1] -0.4  1.6 -2.4
##
## $equations
## [1] -2.055633e-08 -2.055633e-08  3.443454e-08
##
## $itel
## [1] 163
```

In the tertiary approach, without weights, we require $x_1 \leq \frac{x_2+x_3+x_4}{3} \leq \frac{x_5+x_6}{2} \leq x_7$ which means

```
a <- matrix(c(-1,1/3,1/3,1/3,0,0,0,
              0,-1/3,-1/3,-1/3,1/2,1/2,0,
              0,0,0,0,-1/2,-1/2,1),
              3,7,byrow = TRUE)
matrixPrint(a, d = 2, w = 5)
```

```
##      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]
## [1,] -1.00 +0.33 +0.33 +0.33 +0.00 +0.00 +0.00
## [2,] +0.00 -0.33 -0.33 -0.33 +0.50 +0.50 +0.00
## [3,] +0.00 +0.00 +0.00 +0.00 -0.50 -0.50 +1.00
```

This gives

```
qpmaj(y, a = a)
```

```
## $x
## [1]  1.0  1.4  0.4  2.4  2.9 -0.1  3.0
##
## $fprimal
## [1] 1.35
##
## $fdual
## [1] 1.35
##
## $lambda
## [1] 0.0 1.8 0.0
##
## $inequalities
## [1]  4.000000e-01 -1.716935e-08  1.600000e+00
##
## $itel
## [1] 30
```

9.3.2.3 Example 3: Weighted Rounding

This is a silly example in which a vector y = 0.5855288, 0.709466, -0.1093033, -0.4534972, 0.6058875, -1.817956, 0.6300986, -0.2761841, -

0.2841597, -0.919322 is "rounded" so that its elements are between -1 and $+1$. The weights $V = W$ are a banded positive definite matrix.

```
a<-rbind(-diag(10),diag(10))
b<-rep(-1, 20)
w<-ifelse(outer(1:10,1:10,function(x,y) abs(x-y) < 4), -1, 0)+7*diag(10)
qpmaj(y, v = w, a = a, b = b)
```

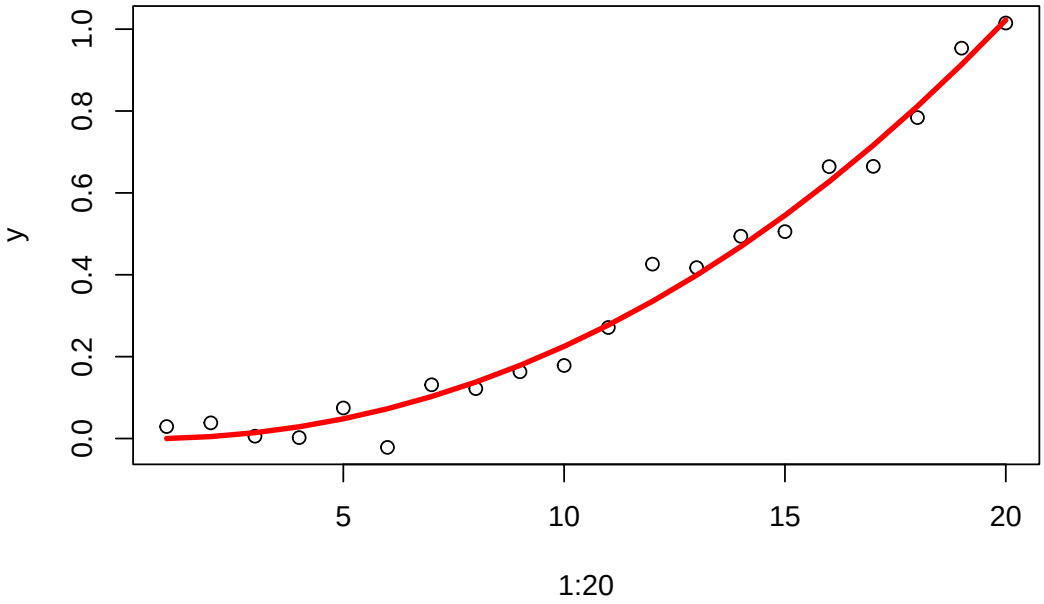
```
## $x
##  [1]  0.737345533  0.917584527  0.215666042 -0.075684750  1.000000017
##  [6] -1.000000039  1.000000023  0.061944776 -0.002332657 -0.754345762
##
## $fprimal
## [1] 1.110748
##
## $fdual
## [1] 1.110749
##
## $lambda
##  [1] 0.0000000 0.0000000 0.0000000 0.0000000 0.0722112 0.0000000 0.1554043
##  [8] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [15] 0.0000000 2.8209838 0.0000000 0.0000000 0.0000000 0.0000000
##
## $inequalities
##  [1]  2.626545e-01  8.241547e-02  7.843340e-01  1.075685e+00 -1.735205e-08
##  [6]  2.000000e+00 -2.303727e-08  9.380552e-01  1.002333e+00  1.754346e+00
## [11]  1.737346e+00  1.917585e+00  1.215666e+00  9.243153e-01  2.000000e+00
## [16] -3.922610e-08  2.000000e+00  1.061945e+00  9.976673e-01  2.456542e-01
##
## $itel
## [1] 224
```

9.3.2.4 Example 4: Monotone Polynomials

This example has a matrix H with the monomials of degree $1, 2, 3$ on the 20 points $1, \cdots 20$. We want to fit a third-degree polynomial which is monotone, non-negative, and anchored at zero (which is why we do not have a monomial

of degree zero, i.e. an intercept). Monotonicity is imposed by $(h_{i+1}-h_i)'x \geq 0$ and non-negativity by $h_1'x \geq 0$. Thus there are $19+1$ inequality restrictions. For y we choose points on the quadratic curve $y = x^2$, perturbed with random error.

```
set.seed(12345)
h <- cbind(1:20,(1:20)^2,(1:20)^3)
a <- rbind (h[1,],diff(diag(20)) %*% h)
y<-seq(0,1,length=20)^2+rnorm(20)/20
plot(1:20, y)
out<-qpmaj(y,a=a,h=h,verbose=FALSE,itmax=1000, eps = 1e-15)
lines(1:20,out$pred,type="l",lwd=3,col="RED")
```



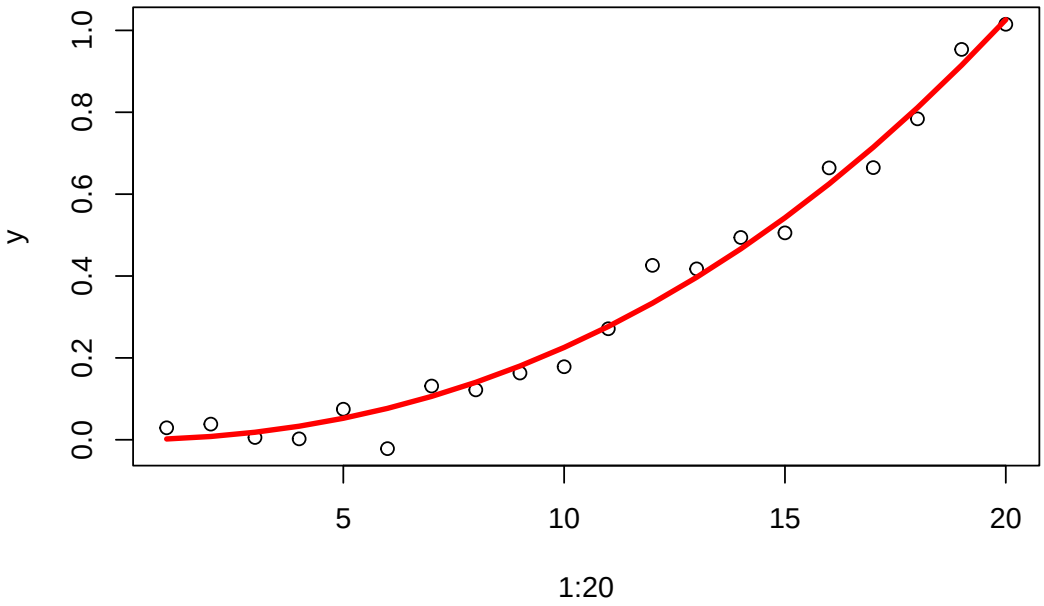
The plot above and the output below shows what qpmaj() does in this case.

```
## $x
## [1] -2.311264e-03  2.292295e-03  1.895686e-05
##
## $fprimal
## [1] 0.0003265426
##
```

```
## $fdual
## [1] 0.0003265446
##
## $ftotal
## [1] 0.01655943
##
## $predict
##  [1] -1.255287e-08  4.698305e-03  1.420869e-02  2.864490e-02  4.812065e-02
##  [6]  7.274970e-02  1.026458e-01  1.379226e-01  1.786940e-01  2.250737e-01
## [11]  2.771753e-01  3.351127e-01  3.989996e-01  4.689497e-01  5.450767e-01
## [16]  6.274945e-01  7.163167e-01  8.116571e-01  9.136294e-01  1.022347e+00
##
## $lambda
##  [1] 0.1587839 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##  [8] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [15] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##
## $inequalities
##  [1] -1.255287e-08  4.698318e-03  9.510389e-03  1.443620e-02  1.947576e-02
##  [6]  2.462905e-02  2.989609e-02  3.527686e-02  4.077138e-02  4.637964e-02
## [11]  5.210164e-02  5.793738e-02  6.388687e-02  6.995009e-02  7.612706e-02
## [16]  8.241776e-02  8.882221e-02  9.534040e-02  1.019723e-01  1.087180e-01
##
## $itel
## [1] 97
```

We now want to accomplish more or less the same thing, but using a cubic of the form $f(x) = x(c + bx + ax^2)$. Choosing a, b and c to be nonnegative guarantees monotonicity (and convexity) on the positive axis, with a root at zero. If $b^2 \geq 4ac$ then the cubic has two additional real roots, and by AM/GM we can guarantee this by $b \geq a + c$. So $a \geq 0$, $c \geq 0$, and $b \geq a + c$ are our three inequalities.

```
h <- cbind(1:20,(1:20)^2,(1:20)^3)
a <- matrix(c(1,0,0,0,0,1,-1,1,-1), 3, 3, byrow = TRUE)
plot(1:20, y)
out<-qpmaj(y,a=a,h=h,verbose=FALSE,itmax=10000, eps = 1e-15)
lines(1:20,out$pred,type="l",lwd=3,col="RED")
```



The results of this alternative way of fitting the cubic are more or less indistinguishable from the earlier results, although this second approach is quite a bit faster (having only three inequalities instead of 21).

```
## $x
## [1] -5.673813e-09  1.945808e-03  3.098195e-05
##
## $fprimal
## [1] 0.0007170899
##
## $fdual
## [1] 0.000717091
##
## $ftotal
## [1] 0.01694997
##
## $predict
##  [1] 0.001976784 0.008031075 0.018348765 0.033115745 0.052517907 0.076741142
##  [7] 0.105971343 0.140394402 0.180196208 0.225562656 0.276679635 0.333733038
## [13] 0.396908757 0.466392682 0.542370707 0.625028722 0.714552619 0.811128290
## [19] 0.914941626 1.026178519
##
```

```
## $lambda
## [1] 0.2021458 0.0000000 0.0000000
##
## $inequalities
## [1] -5.673813e-09  3.098195e-05  1.914831e-03
##
## $itel
## [1] 25
```

9.3.3 Examples

Chapter 10

Ordinal MDS

10.1 Monotone Regression

Is it really what we want

10.1.1 Simple Monotone Regression

Ever since Kruskal (1964a) and Kruskal (1964b) monotone regression has played an important part in non-metric MDS. Too important, perhaps. Initially there was some competition with the rank images of Guttman (1968), but that competition has largely faded over time.

We only give the barest outline in this section. More details are in De Leeuw, Hornik, and Mair (2009). In (simple least squares) monotone (or isotone) regression we minimize $(x-y)'W(x-y)$, where $W \gtrsim 0$ is diagonal, over x satisfying $x_1 \leq \cdots \leq x_n$. The vector y is the *target* or the *data*.

The algorithm, which is extremely fast and of order n, is based on the simple rule that if elements are out of order, then you compute their weighted average, forming blocks, keeping track of the block sizes and block weights. This reduces the MR problem to a smaller MR problem, and following the rule systematically leads to a finite algorithm. A particularly efficient implementation is in Busing (2021).

A simple illustration. The first column are the value that we compute the best monotone fit for, the second columns are the size of the blocks after merging. In this case there are no weights, in fact the block sizes serve as weights.

$$(1,2,1,3,2,-1,3) \qquad (1,1,1,1,1,1,1) \tag{10.1}$$
$$(1,\frac{3}{2},3,2,-1,3) \qquad (1,2,1,1,1,1) \tag{10.2}$$
$$(1,\frac{3}{2},\frac{5}{2},-1,3) \qquad (1,2,2,1,1) \tag{10.3}$$
$$(1,\frac{3}{2},\frac{4}{3},3) \qquad (1,2,3,1) \tag{10.4}$$
$$(1,\frac{7}{5},3) \qquad (1,5,1) \tag{10.5}$$

Expanding using the block size gives the solution $(1,\frac{7}{5},\frac{7}{5},\frac{7}{5},\frac{7}{5},\frac{7}{5},3)$.

In the second example we do have weights, in the second column, and we use a third column for blocks size.

$$(1,2,1,3,2,-1,3) \qquad (1,2,3,4,3,2,1) \qquad (1,1,1,1,1,1,1) \tag{10.6}$$
$$(1,\frac{7}{5},3,2,-1,3) \qquad (1,5,4,3,2,1) \qquad (1,2,1,1,1,1) \tag{10.7}$$
$$(1,\frac{7}{5},\frac{18}{7},-1,3) \qquad (1,5,7,2,1) \qquad (1,2,2,1,1) \tag{10.8}$$
$$(1,\frac{7}{5},\frac{16}{9},3) \qquad (1,5,9,1) \qquad (1,2,3,1) \tag{10.9}$$

Expansion gives the solution $(1,\frac{7}{5},\frac{7}{5},\frac{16}{9},\frac{16}{9},\frac{16}{9},3)$.

The usual monotone regression algorithms used in MDS allow for slightly more complicated orders to handle ties in the data . There are basically three approaches to ties implemented. In what Kruskal calls the *primary approach*, only order relations between tie blocks are maintained. Within blocks no order s mposed. In the *secondary approach* we require equality in tie blocks. Ties in the data means we impose ties in the isotone regression.

Both approaches can be incorporated in simple monotone regresson by preprocessing. The secondary approach starts with the weighted averages of the tie blocks, the primary approach orders the data within tie blocks so they are non-decreasing. De Leeuw (1977b) showed that this preprocessing does indeed give the least squares solution for both approaches. In the same paper he also introduces a less restrictive *tertiary approach*, which merely requires that the averages of the tie blocks are in the required order.

10.1.2 Weighted Monotone Regression

$$(x-y)'V(x-y)$$

$$V(x-y)=A'\lambda$$

$$Ax\geq 0$$

$$\lambda\geq 0$$

$$\lambda'Ax=0$$

$$x=y+V^{-1}A'\lambda$$

go to the dual if $\lambda_i>0$ then $a_i'x=0$

unweighting actually proves weighted is unweighted for something else

$MR(x+\epsilon y)=MR(x)+\epsilon B(y)$ if $MR(x)=Bx$

10.1.3 Normalized Cone Regression

De Leeuw (1975a)

Bauschke, Bui, and Wang (2018)

10.1.4 Iterative MR

primal-dual: MR is dual Dykstra vertices of cone plus CCA one iteration only

10.2 Alternating Least Squares

smacof: hard squeeze double phase

10.3 Kruskal's Approach

10.3.1 Kruskal's Stress

Remember that Kruskal's definition of stress was intended for ordinal multidimensional scaling only. Thus dissimilarities are not necessarily numerical, as in basic MDS, only their rank order is known. He first defined *raw stress* as

$$\sigma_K^\star(X) := \sum_{1\leq i<j\leq n}\sum (\hat{d}_{ij} - d_{ij}(X))^2, \tag{10.10}$$

where the $\hat{d}_{ij}$ is some set of numbers monotone with the dissimilarities.

> To simplify the discussion, we delay the precise definition of $\hat{d}$, for a little while. (Kruskal (1964a), p. 8)

Kruskal then mentions that raw stress satisfies $\sigma_K^\star(\alpha X) = \alpha^2\sigma_K^\star(X)$, which is clearly undesirable because the size of the configuration should not influence the quality of the fit.

> An obvious way to cure this defect in the raw stress is to divide it by a scaling factor, that is, a quantity which has the same quadratic dependence on the scale of the configuration that raw stress does. (Kruskal (1964a), p. 8).

By the way, although the precise definition of $\hat{D}$ has been delayed, the uniform stretching/shrinking argument already assumes that if we multiply D by α then $\hat{D}$ also gets multiplied by α. Thus it sort of gives away that $\hat{D}$ is also a function of X, at least of the scale of X.

For the normalization of raw stress Kruskal chooses

$$\tau_K^\star(X) := \sum_{1\leq i<j\leq n}\sum d_{ij}^2(X), \tag{10.11}$$

> Finally, it is desirable to use the square root of this expression, which is analogous to choosing the standard deviation in place of the variance. (Kruskal (1964a), p. 9)

Thus Kruskal's normalized loss function for ordinal MDS becomes

$$\sigma_K(X) := \sqrt{\frac{\sigma_K^\star(X)}{\tau_K^\star(X)}} = \sqrt{\frac{\sum\sum_{1\leq i<j\leq n}(\hat{d}_{ij}-d_{ij}(X))^2}{\sum\sum_{1\leq i<j\leq n} d_{ij}^2(X)}}. \tag{10.12}$$

At this point in Kruskal (1964a) the definition of $\hat{D}$ still hangs in the air, although we know that the $\hat{D}$ are monotone with Δ, and that multiplying X by a constant will multiply both $D(X)$ and $\hat{D}$ by the same constant. Matters are clarified right after the definition of stress.

> Now it is easy to define the $\hat{d}_{ij}$. They are the numbers which minimize σ (or equivalently, $\sigma^\star$) subject to the monotonicity constraints. (Kruskal (1964a), p. 9)

Thus, actually, raw stress is the minimum over the pseudo-distance matrices Ω in $\mathfrak{D}$, the set of all monotone transformations of the dissimilarities.

$$\sigma^\star(X) := \min_{\Omega\in\mathfrak{D}} \sum_{1\leq i<j\leq n}\sum (\omega_{ij}-d_{ij}(X))^2, \tag{10.13}$$

and $\hat{D}$ is the minimizer, which is now clearly a function of X,

$$\hat{D}(X) := \operatorname*{argmin}_{\Omega\in\mathfrak{D}} \sum_{1\leq i<j\leq n}\sum (\omega_{ij}-d_{ij}(X))^2. \tag{10.14}$$

So, finally,

$$\sigma(X) := \min_{\Omega\in\mathfrak{D}} \sqrt{\frac{\sigma^\star(X)}{\tau^\star(X)}} = \sqrt{\frac{\sum\sum_{1\leq i<j\leq n}(\omega_{ij}-d_{ij}(X))^2}{\sum\sum_{1\leq i<j\leq n} d_{ij}^2(X)}}. \tag{10.15}$$

In Guttman's terminology Kruskal's approach is hard squeeze single phase. Thus what is minimized is

$$\sigma_{JBK}(X) := \min_{\Delta\in\mathfrak{D}} \sqrt{\frac{\sum\sum_{1\leq i<j\leq n}(\delta_{ij}-d_{ij}(X))^2}{\sum\sum_{1\leq i<j\leq n} d_{ij}^2(X)}}$$

10.3.2 Stress1 and Stress2

10.4 Guttman's Approach

The main alternative to the Kruskal approach to MDS, besides smacof, is the Smallest Space Analysis (SSA) of Guttman and Lingoes. I have mixed feelings about the fundamental SSA paper of Guttman (1968). It is, no doubt, a milestone MDS paper, and some of the distinctions it makes (which we will discuss later in this section) are clearly important. Its use of matrix algebra, wherever possible, is an improvement over Kruskal (1964a), and the correction matrix algorithm for SSA is an immediate predecessor of smacof. But it seems to me the derivation of the correction matrix algorithm is incomplete and could even be called incorrect. The rank images used by Guttman and Lingoes in SSA seem an ad-hoc solution invented by someone who did not yet know about monotone regression. And, above all, the paper exudes a personality cult-like atmosphere that is somewhat repellent to me. There are no gurus in science. Or at least there should not be. It is true that between 1930 and 1960 Guttman invented and elucidated about 75% of the psychometrics of his time, but 75% is still less than 100%. This book you are reading now may set a record in self-citation, but that makes sense because it is supposed to document my work in MDS and to give access to the pdf's of my unpublished work. I try to be careful not to take credit for results that did not originate with me, and to give appropriate attributions in all cases.

Table 10.1: Semi-strong Rank Images

	1	2	3	4	5	6	7
Δ	1	2	2	3	3	4	5
$D(X)$	1	3	1	3	4	3	4
$D(X)\ \text{ordered}$	1	1	3	3	3	4	4
$D^\star$	1	1	3	3	3	4	4

10.4.0.1 Rank Images

The rank image transformation, which replaces the monotone regression in Kruskal's approach, has a rather complicated definition. It is simple enough when both Δ and $D(X)$ have no ties. In that case the rank image $D^\star$ is just the unique permutation of $D(X)$ that is monotone with Δ. Thus

$$\delta_{ij} < \delta_{kl} \Leftrightarrow d^\star_{ij} < d^\star_{kl}. \tag{10.16}$$

If there are ties in Δ and/or $D(X)$ then some of the uniqueness and simplicity will get lost. Guttman (1968) introduces an elaborate notation for rank images with ties, but that notation does neither him nor the reader any favors.

If there are ties in $D(X)$ you use the rank order of the corresponding elements of Δ to order $D(X)$ within tie blocks. If two elements are tied both in $D(X)$ and Δ, then their order in the tie block is arbitrary.

Suppose the Δ have R tie-blocks, in increasing order, with $m_1, \cdots, m_R$ elements. The smallest m_1 elements of the vector of distances become the first m_1 elements of $D^\star$, the next m_2 elements of $D^\star$ are the next smallest m^2 elements of distance vector, and so on for all tie blocks. Thus tied elements in Δ can become untied in $D^\star$ and untied elements in Δ can becomes tied in $D^\star$. We require

$$\delta_{ij} < \delta_{kl} \Rightarrow d^\star_{ij} \leq d^\star_{kl} \tag{10.17}$$

This corresponds with Kruskal's primary approach to ties. Guttman calls it *semi-strong monotonicity*. There is a small numerical example in table 10.1.

Table 10.2: Strong Rank Images

	1	2	3	4	5	6	7
Δ	1	2	2	3	3	4	5
$D(X)$	1	3	1	3	4	3	4
$D(X)\ \text{ordered}$	1	1	3	3	3	4	4
$D^\star$	1	2	2	3	3	4	4

The sum of the squared differences between $D(X)$ and $D^\star$ is 6.

Alternatively, we can require that tied elements in Δ correspond with tied elements in $D^\star$. Guttman calls this *strong monotonicity*, and requires in addition that $D^\star$ has the same number of blocks, with the same block sizes, as Δ. Instead of copying ordered blocks from the sorted distances, we compute averages of blocks, and copy those into D^star. This corresponds with Kruskal's secondary approach to ties. Thus the elements of $D^\star$ are no longer a permutation of those in $D(X)$. We have @eq:nmrankimage1, and also

$$\delta_{ij} = \delta_{kl} \Rightarrow d^\star_{ij} = d^\star_{kl} \tag{10.18}$$

Our numerical example is now in table 10.2.

Now the sum of squared differences between $D(X)$ and $D^\star$ is 4, which means, surprisingly, that strong monotonicity gives a better fit than semi-strong monotonicity. This cannot happen with monotone regression, where the primary approch to ties always has a better fit than the secondary approach.

10.4.0.2 Single and Double Phase

Note: suppose $\|D_1^\star - D(X_2)\|^2 < \|D_1^\star - D(X_1)\|^2$ but $\|D_2^\star - D(X_2)\|^2 > \|D_1^\star - D(X_2)\|^2$

$$\sigma_G(X) = \frac{\sum\sum_{1\leq i<j\leq n} w_{ij}(d^\star_{ij}(X) - d_{ij}(X))^2}{\sum\sum_{1\leq i<j\leq n} w_{ij} d^2_{ij}(X)}.$$

$$\sigma_G(X, D^\star) = \frac{\sum\sum_{1\leq i<j\leq n}(d^\star_{ij} - d_{ij}(X))^2}{\sum\sum_{1\leq i<j\leq n} d^2_{ij}(X)}.$$

10.4.0.3 Hard and Soft Squeeze

$$\sigma(X) := \min_{\delta \in \mathbb{K} \cap \mathbb{S}} \sum_{k \in \mathcal{K}} w_k (\delta_k - d_k(X))^2$$

question: in double phase do rank images decrease stress ? My guess is yes. Are they continuous ?

$$\rho_G(X) = \max_{P \in \Pi} \delta' P d(X)$$

is a continuous function of X. Also (Shepard)

$$D_+ \rho_G(X) = \max_{P \in \Pi(X)} \delta' P \mathcal{D} d(X)$$

rank-images Pd are not continuous

10.4.1 Smoothness of Ordinal Loss Functions

Kruskal

$$\min_{\hat{D} \in \mathfrak{D}} \sum \sum w_{ij} (\hat{d}_{ij} - d_{ij}(X))^2$$

is a differentiable function of X

Double phase rank image

$$\sigma_{LG}(X) = \min_P \| Pd(X) - d(X) \|^2$$

P in the Birkhoff polytope and satisfying inequalities, equalities.

P given by $\max_P d(X)' P d(X)$

De Leeuw (1973b)

10.5 Scaling with Distance Bounds

$$\alpha_{ij} \leq d_{ij}(X) \leq \beta_{ij}$$

10.6 Bounds on Stress

De Leeuw and Stoop (1984)

Stress1 and Stress2

Chapter 11

Splinical MDS

11.1 Splines

In this section we give a short introduction, with examples, to (univariate) splines, B-splines, and I-splines. It is taken from De Leeuw (2017a), with some edits to make it fit into the book. The report it was taken from has more detail and more examples.

To define *spline functions* we first define a finite sequence of *knots* $T = \{t_j\}$ on the real line, with $t_1 \leq \cdots \leq t_p$, and an *order* m. In addition each knot t_j has a *multiplicity* m_j, the number of knots equal to t_j. We suppose throughout that $m_j \leq m$ for all j.

A function f is a *spline function of order* m for a knot sequence $\{t_j\}$ if

1. f is a polynomial π_j of degree at most $m-1$ on each half-open interval $I_j = [t_j, t_{j+1})$ for $j = 1, \cdots, p$,
2. the polynomial pieces are joined in such a way that $\mathcal{D}_-^{(s)} f(t_j) = \mathcal{D}_+^{(s)} f(t_j)$ for $s = 0, 1, \cdots, m - m_j - 1$ and $j = 1, 2, \cdots, p$.

Here we use $\mathcal{D}_-^{(s)}$ and $\mathcal{D}_+^{(s)}$ for the left and right s^{th}-derivative operator. If $m_j = m$ for some j, then the second requirement is empty, if $m_j = m-1$ then the second requirement means $\pi_j(t_j) = \pi_{j+1}(t_j)$, i.e. we require continuity of f at t_j. If $1 \leq m_j < m - 1$ then f must be $m - m_j - 1$ times differentiable, and thus continuously differentiable, at t_j.

In the case of simple knots (with multiplicity one) a spline function of order one is a *step function* which steps from one level to the next at each knot. A spline of order two is piecewise linear, with the pieces joined at the knots so that the spline function is continuous. Order three means a piecewise quadratic function which is continuously differentiable at the knots. And so on.

11.1.1 B-splines

Alternatively, a spline function of order m can be defined as a linear combination of *B-splines* (or *basic splines*) of order m on the same knot sequence. A B-spline of order m is a spline function consisting of at most m non-zero polynomial pieces. A B-spline $\mathcal{B}_{j,m}$ is determined by the $m+1$ knots $t_j \leq \cdots \leq t_{j+m}$, is zero outside the interval $[t_j, t_{j+m})$, and positive in the interior of that interval. Thus if $t_j = t_{j+m}$ then $\mathcal{B}_{j,m}$ is identically zero.

For an arbitrary finite knot sequence $t_1, \cdots, t_p$, there are $p-m$ B-splines to of order m to be considered, although some may be identically zero. Each of the splines covers at most m consecutive intervals, and at most $m-1$ different B-splines are non-zero at each point.

11.1.1.1 Boundaries

B-splines are most naturally and simply defined for doubly infinite sequences of knots, that go to $\pm\infty$ in both directions. In that case we do not have to worry about boundary effects, and each subsequence of $m+1$ knots defines a B-spline. For splines on finite sequences of p knots we have to decide what happens at the boundary points.

There are B-splines for $t_j, \cdots, t_{j+m}$ for all $j = 1, \cdots, p-m$. This means that the first $m-1$ and the last $m-1$ intervals have fewer than m splines defined on them. They are not part of what De Boor (2001), page 94, calls the *basic interval*. For doubly infinite sequences of knots there is not need to consider such a basic interval.

If we had m additional knots on both sides of our knot sequence we would also have m additional B-splines for $j = 1-m, \cdots, 0$ and m additional B-splines for $j = p-m+1, \cdots, p$. By adding these additional knots we make

sure each interval $[t_j, t_{j+1})$ for $j = 1, \cdots, p-1$ has m B-splines associated with it. There is stil some ambiguity on what to do at t_p, but we can decide to set the value of the spline there equal to the limit from the left, thus making the B-spline left-continuous there.

In our software we will use the convention to define our splines on a closed interval $[a, b]$ with r *interior knots* $a < t_1 < \cdots < t_r < b$, where interior knot t_j has multiplicity m_j. We extend this to a series of $p = M + 2m$ knots, with $M = \sum_{j=1}^{r} m_j$, by starting with m copies of a, appending m_j copies of t_j for each $j = 1, \cdots, r$, and finishing with m copies of b. Thus a and b are both knots with multiplicity m. This defines the *extended partition* (Schumaker (2007), p 116), which is just handled as any knot sequence would normally be.

11.1.1.2 Normalization

The conditions we have mentioned only determine the B-spline up to a normalization. There are two popular ways of normalizing B-splines. The N-splines $N_{j,m}$, a.k.a. the *normalized B-splines* j or order m, satisfies

$$\sum_j N_{j,m}(t) = 1. \tag{11.1}$$

Note that in general this is not true for all t, but only for all t in the *basic interval*.

Alternatively we can normalize to M-splines, for which

$$\int_{-\infty}^{+\infty} M_{j,m}(t)dt = \int_{t_j}^{t_{j+k}} M_{j,m}(t)dt = 1. \tag{11.2}$$

There is the simple relationship

$$N_{j,m}(t) = \frac{t_{j+m} - t_j}{m}\ M_{j,m}(t). \tag{11.3}$$

11.1.1.3 Recursion

B-splines can be defined in various ways, using piecewise polynomials, divided differences, or recursion. The recursive definition, first used as the preferred

definition of B-splines by De Boor and Höllig (1985), is the most convenient one for computational purposes, and that is the one we use.

The recursion is due independently to M. G. Cox (1972) for simple knots and to De Boor (1972) in the general case, is

$$M_{j,m}(t) = \frac{t - t_j}{t_{j+m} - t_j} M_{j,m-1}(t) + \frac{t_{j+m} - t}{t_{j+m} - t_j} M_{j+1,m-1}(t), \tag{11.4}$$

or

$$N_{j,m}(t) = \frac{t - t_j}{t_{m+j-1} - t_j} N_{j,m-1}(t) + \frac{t_{j+m} - t}{t_{j+m} - t_{j+1}} N_{j+1,m-1}(t). \tag{11.5}$$

A basic result in the theory of B-splines is that the different B-splines are linearly independent and form a basis for the linear space of spline functions (of a given order and knot sequence).

In section A.1.19 the basic BSPLVB algorithm from De Boor (2001), page 111, for normalized B-splines is translated to R and C. There are two auxiliary routines, one to create the extended partition, and one that uses bisection to locate the knot interval in which a particular value is located (Schumaker (2007), p 191). The R function bsplineBasis() takes an arbitrary knot sequence. It can be combined with extendPartition(), which uses inner knots and boundary points to create the extended partion.

11.1.1.4 Illustrations

For our example, which is the same as the one from figure 1 in Ramsay (1988), we choose $a = 0$, $b = 1$, with simple interior knots 0.3, 0.5, 0.6. First the *step functions*, which have order 1.

Now the *hat functions*, which have order 2, again with simple knots.

Next piecewise quadratics, with simple knots, which implies continuous differentiability at the knots. This are the N-splines corresponding with the M-splines in figure 1 of Ramsay (1988).

If we change the multiplicities to 1, 2, 3, then we lose some of the smoothness.

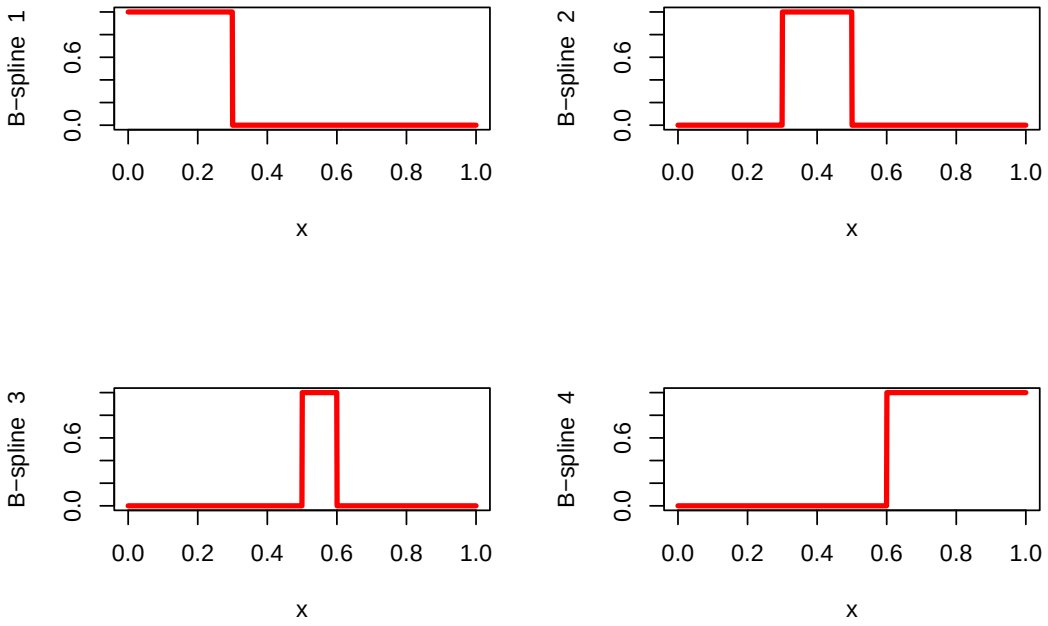


Figure 11.1: Zero Degree Splines with Simple Knots

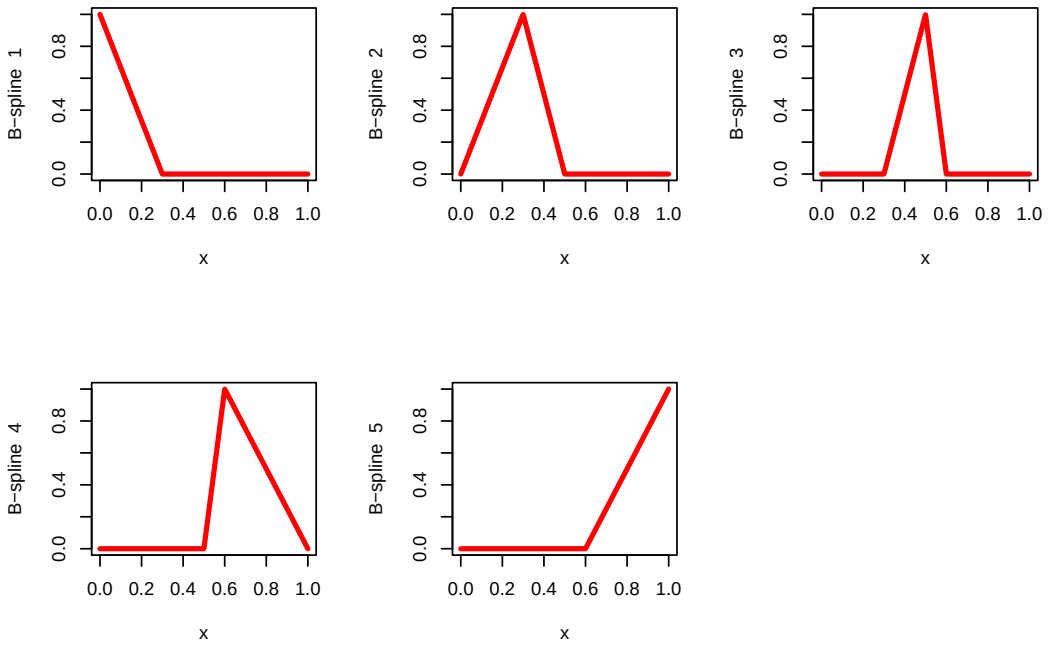


Figure 11.2: Piecewise Linear Splines with Simple Knots

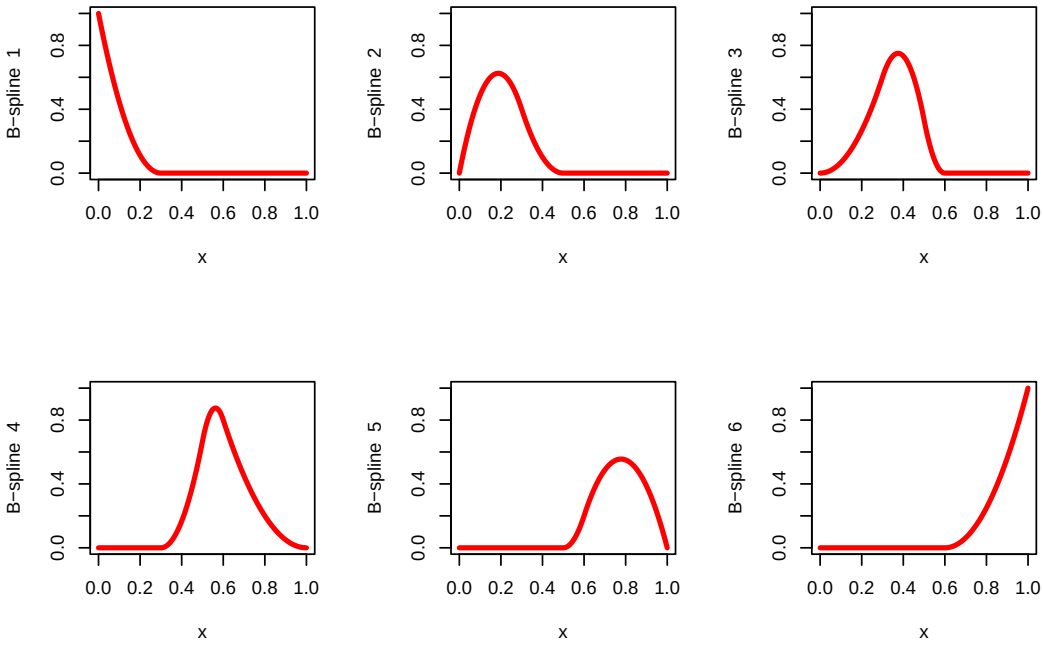


Figure 11.3: Piecwise Quadratic Splines with Simple Knots

11.1.2 I-splines

There are several ways to require splines to be monotone increasing. Since B-splines are non-negative, the definite integral of a B-spline of order m from the beginning of the interval to a value x in the interval is an increasing spline of order $m+1$. Integrated B-splines are known as *I-splines* (Ramsay (1988)). Non-negative linear combinations I-splines can be used as a basis for the convex cone of increasing splines. Note, however, that if we use an extended partition, then all I-splines start at value zero and end at value one, which means their convex combinations are those splines that are also probability distributions on the interval. To get a basis for the increasing splines we need to add the constant function to the I-splines and allow it to enter the linear combination with either sign.

I-splines are most economically computed by using the formula first given by Gaffney (1976). If ℓ is defined by $t_{j+\ell-1} \leq x < t_{j+\ell}$ then

$$\int_{x_j}^{x} M_{j,m}(t)dt = \frac{1}{m}\sum_{r=0}^{\ell-1}(x - x_{j+r})M_{j+r,m-r}(x)$$

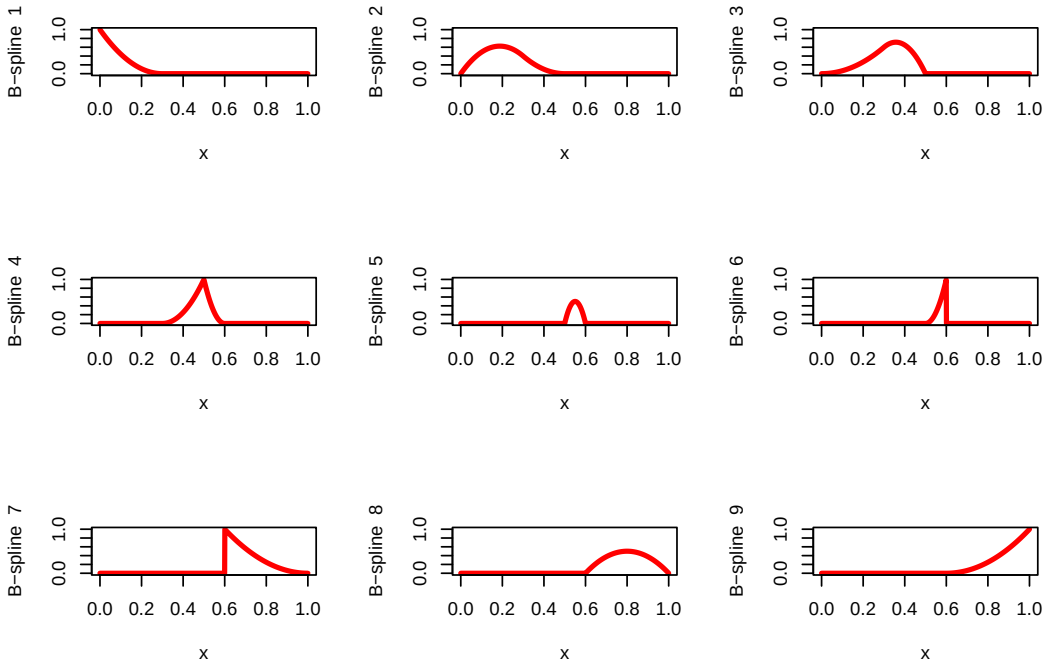


Figure 11.4: Piecewise Quadratic Splines with Multiple Knots

It is somewhat simpler, however, to use lemma 2.1 of De Boor, Lyche, and Schumaker (1976). This says

$$\int_a^x M_{j,m}(t)dt = \sum_{\ell \geq j} N_{\ell,m+1}(x) - \sum_{\ell \geq j} N_{\ell,m+1}(a),$$

If we specialize this to I-splines, we find , as in De Boor (1976), formula 4.11,

$$\int_{-\infty}^x M_{j,m}(t)dt = \sum_{\ell=j}^{j+r} N_{\ell,m+1}(x)$$

for $x \leq t_{j+r+1}$. This shows that I-splines can be computed by using cumulative sums of B-spline values.

Note that using the definition using integration does not give a natural way to define increasing splines of degree one, i.e. increasing step functions. There is no such problem with the cumulative sum approach.

11.1.2.1 Increasing Coefficients

As we know, a spline is a linear combination of B-splines. The formula for the derivative of a spline, for example in De Boor (2001), p 116, shows that a spline is increasing if the coefficients of the linear combination of B-splines are increasing. Thus we can fit an increasing spline by restricting the coefficients of the linear combination to be increasing, again using the B-spline basis.

It turns out this is in fact identical to using I-splines. If the B-spline values at n points are in an $n \times r$ matrix H, then non-decreasing coefficients β are of the form $\beta = S\alpha + \gamma e_r$, where S is lower-diagonal with all elements on and below the diagonal equal to one, where $\alpha \geq 0$, where e_r has all elements equal to one, and where γ can be of any sign. So $H\beta = (HS)\alpha + \gamma e_n$. Thus non-decreasing coefficients is the same thing as using cumnulative sums of the B-spline basis.

11.1.2.2 Increasing Values

Finally, we can simply require that the n elements of $H\beta$ are increasing. This is a less restrictive requirement, because it allows for the possibility that the spline is decreasing between data values. It has the rather serious disadvantage, however, that it does its computations in n-dimensional space, and not in r-dimensional space, where $r = M + m$, which is usually much smaller than n. Software for the increasing-value restrictions has been written by De Leeuw (2015). In our software, however, we prefer the `cumsum()` approach. It is less general, but considerably more efficient.

We use the same Ramsay example as before, but now cumulatively. First we integrate step functions with simple knots, which have order one, using `isplineBasis()`. The corresponding I-splines are piecewise linear with order two.

Now we integrate the hat functions, which have order 2, again with simple knots, to find piecewise quadratic I-splines of order 3. These are the functions in the example of Ramsay (1988).

Finally, we change the multiplicities to 1, 2, 3, and compute the corresponding piecewise quadratic I-splines.

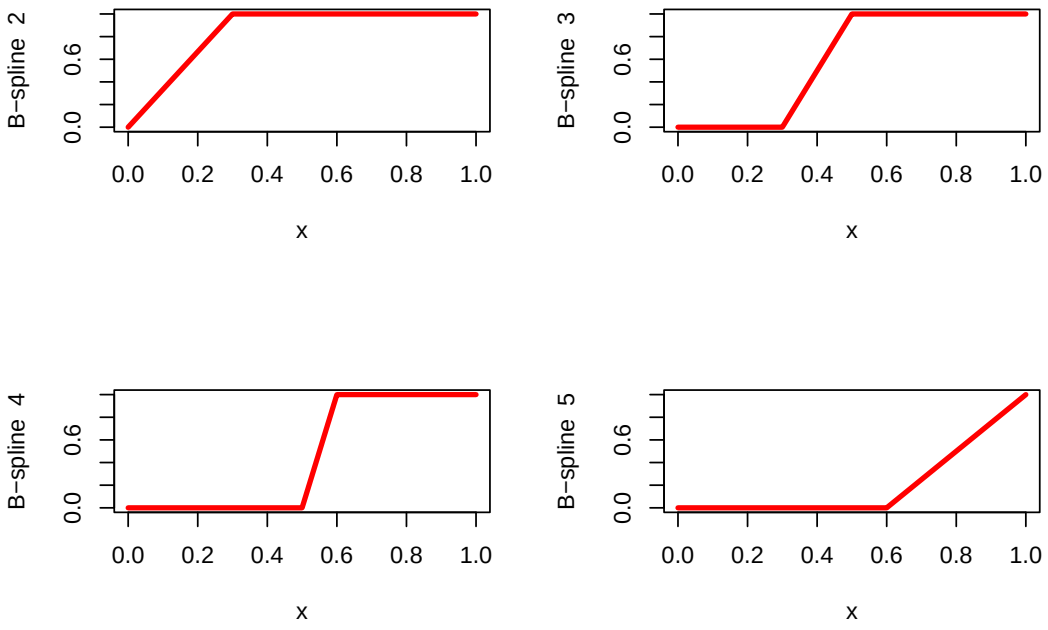


Figure 11.5: Not Sure

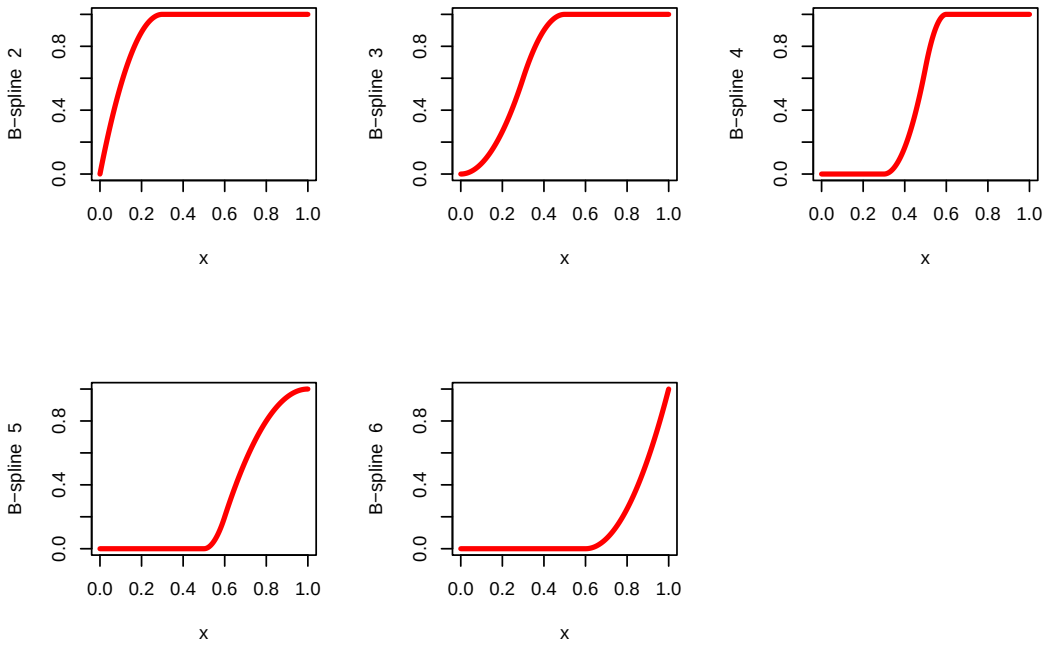


Figure 11.6: Monotone Piecewise Linear Splines with Simple Knots

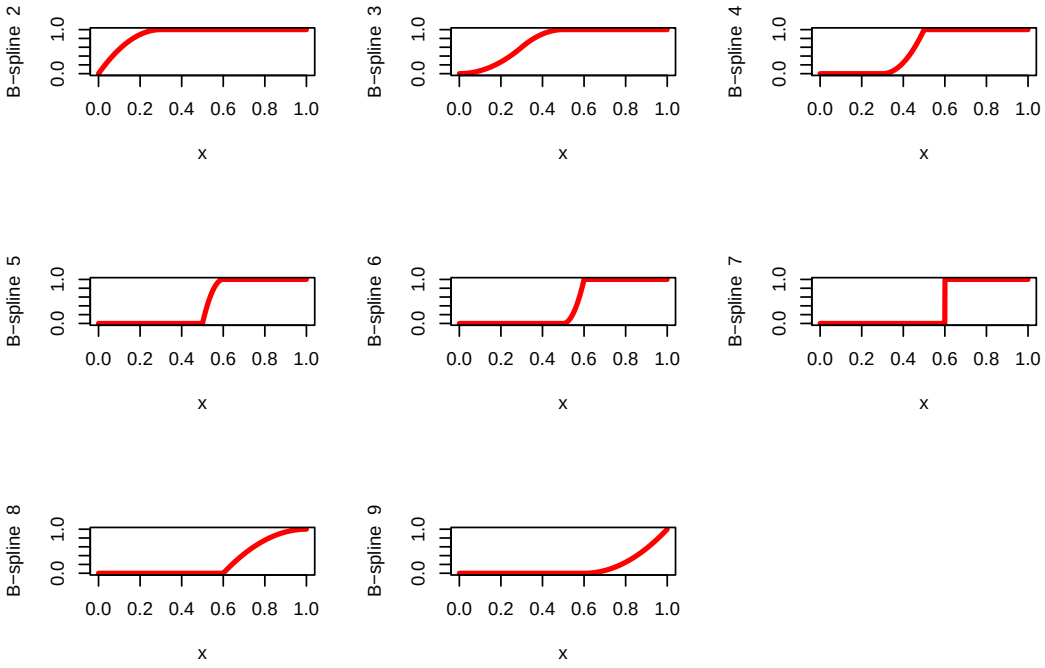


Figure 11.7: Monotone Piecewise Quadratic Splines with Multiple Knots

11.1.3 Time Series Example

Our first example smoothes a time series by fitting a spline. We use the number of births in New York from 1946 to 1959 (on an unknown scale), from Rob Hyndman's time series archive.

11.1.3.1 B-splines

First we fit B-splines of order three. The basis matrix uses x equal to $1:168$, with inner knots 12, 24, 36, 48, 60, 72, 84, 96, 108, 120, 132, 144, 156, and interval $[1, 168]$.

```
innerknots <- 12 * 1:13
multiplicities <- rep(1, 13)
lowend <- 1
highend <- 168
order <- 3
```

```
x <- 1:168
knots <-
  extendPartition (innerknots, multiplicities, order, lowend, highend)$knots
h <- bsplineBasis (x, knots, order)
u <- lm.fit(h, births)
res <- sum ((births - h %*% u$coefficients) ^ 2) / 2
```

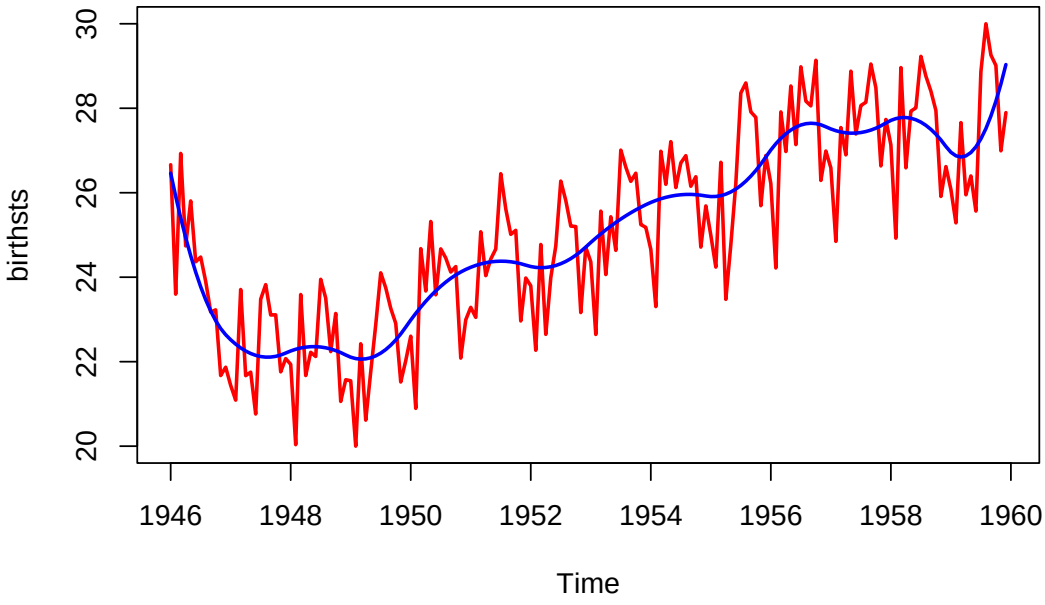


Figure 11.8: Monotone Piecewise Quadratic Splines with Simple Knots

The residual sum of squares is 114.6917709.

11.1.3.2 I-splines

We now fit the I-spline using the B-spline basis. Compute $Z = HS$ using `cumsum()`, and then $\overline{y}$ and $\overline{Z}$ by centering (substracting the column means). The formula is

$$\min_{\alpha \geq 0, \gamma} \textbf{SSQ}\ (y - Z\alpha - \gamma e_n) = \min_{\alpha \geq 0} \textbf{SSQ}\ (\overline{y} - \overline{Z}\alpha).$$

We use `pnnls()` from Wang, Lawson, and Hanson (2015).

```
knots <- extendPartition (innerknots, multiplicities, order, lowend, highend)$
h <- isplineBasis (x, knots, order)
g <- cbind (1, h[,-1])
u <- pnnls (g, births, 1)$x
v <- g%*%u
```

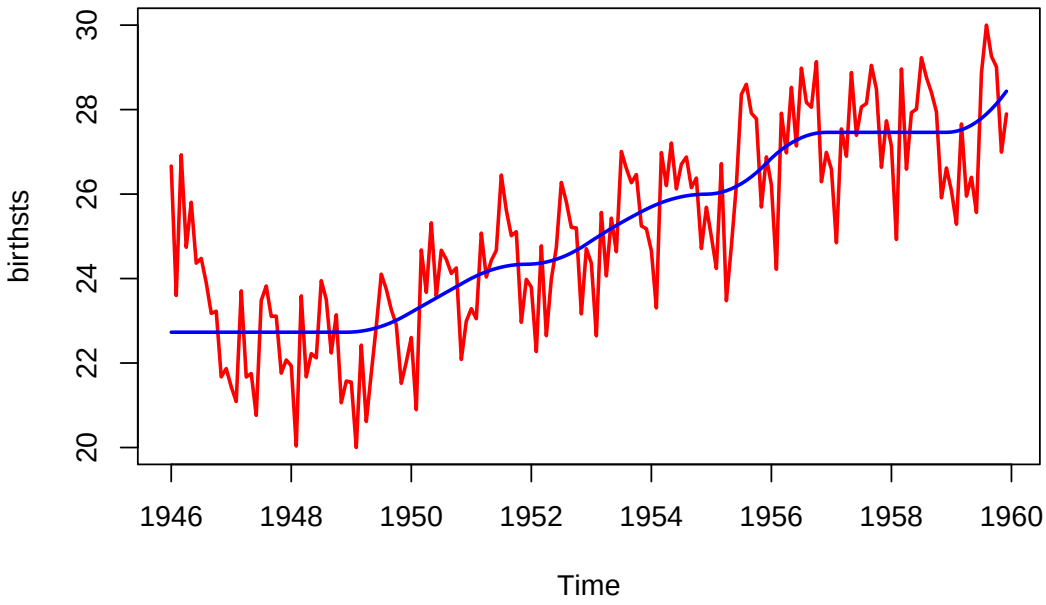


Figure 11.9: Monotone Piecewise Linear Splines with Simple Knots

The residual sum of squares is 144.2027491.

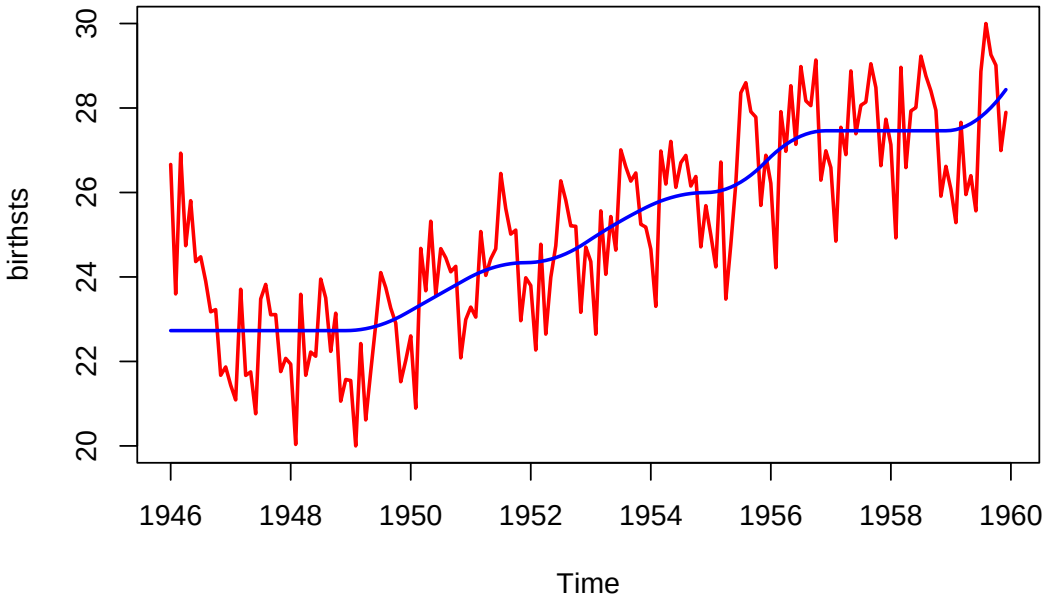
11.1.3.3 B-Splines with monotone weights

Just to make sure, we also solve the problem

$$\min_{\beta_1 \leq \beta_2 \leq \cdots \leq \beta_p} \mathbf{SSQ}(y - X\beta),$$

which should give the same solution, and the same loss function value, because it is just another way to fit I-splines. We use the `lsi()` function from Wang, Lawson, and Hanson (2015).

```
knots <-
  extendPartition (innerknots, multiplicities, order, lowend, highend)$knots
h <- bsplineBasis (x, knots, order)
nb <- ncol (h)
d <- matrix(0, nb - 1, nb)
diag(d) = -1
d[outer(1:(nb - 1), 1:nb, function(i, j)
  (j - i) == 1)] <- 1
u <- lsi(h, births, e = d, f = rep(0, nb - 1))
v <- h %*% u
```



The residual sum of squares is 144.2027491, indeed the same as before.

11.1.3.4 B-Splines with monotone values

Finally we solve

$$\min_{x_1'\beta \leq \cdots \leq x_n'\beta} \mathbf{SSQ}\ (y - X\beta)$$

using qpmaj() from section ???.

```
knots <-
  extendPartition (innerknots, multiplicities, order, lowend, highend)$knots
h <- bsplineBasis (x, knots, order)
a <- diff(diag(nrow(h))) %*% h
u <- qpmaj(births, h = h, a = a)
```

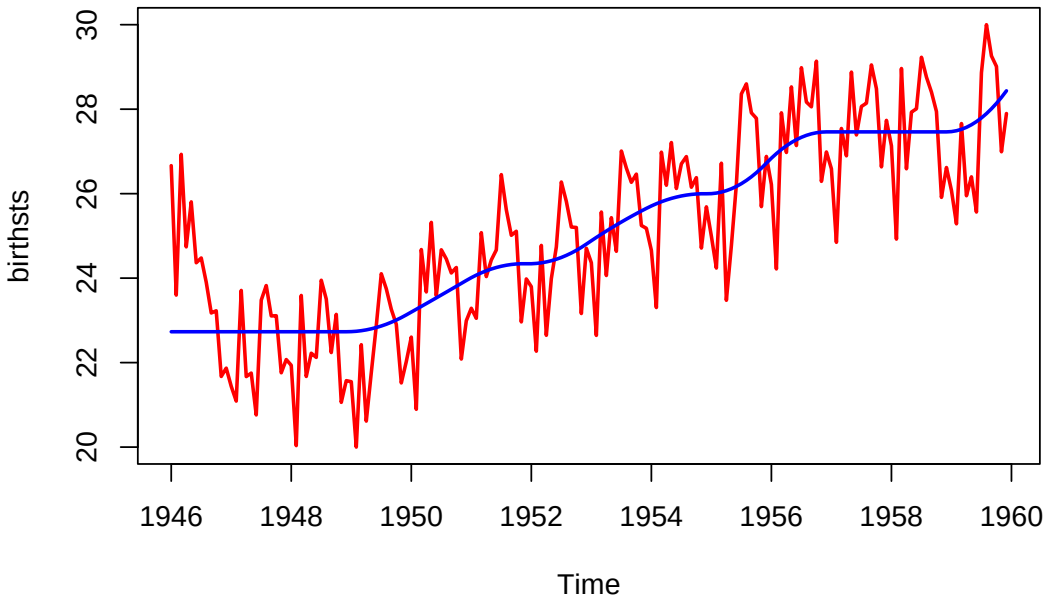


Figure 11.10: Monotone Piecewise Quadratic Splines with Multiple Knots

The residual sum of squares is 144.1574541 , which is indeed smaller than the I-splines value, although only very slightly so.

11.1.4 Local positivity, monotonicity, convexity

Chapter 12

Unidimensional Scaling

For Unidimensional Scaling (UDS or 1MDS) the configuration is a matrix $X \in \mathbb{R}^{n\times 1}$. We can trivially identify this single-column matrix X with the vector x of coordinates of n points on the real line. All our previous general MDS results remain valid for UDS, but we will use the additional structure that comes with $p = 1$ to discuss a number of special results.

Unidimensional scaling appears under different names in the literature such as *seriation* in archeology and *sequencing* in genetics. Often the algorithms permute the rows and columns of a matrix of dissimilarities to approximate some special structure. In this book seriation and sequencing are always understood to be the minimization of stress over x, i.e. minimization of

$$\sigma(x) = \sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij} - |x_i - x_j|)^2 \tag{12.1}$$

12.1 An example

We start the chapter with some pictures, similar to the ones in chapter 2. There are four objects. Dissimilarities are again chosen to be all equal, in this case to $\frac{1}{6}\sqrt{6}$. Weights are all equal to one.

We look at stress on the two-dimensional subspace spanned by the two vectors $y = (0, -1, +1, 0)$ and $z = (-1.5, -.5., +.5, +1.5)$. First we normalize both y and z by $\rho = \eta^2$. This gives $y = (0, -\frac{1}{8}\sqrt{6}, +\frac{1}{8}\sqrt{6}, 0)$ and

$z = (-\frac{1}{8}\sqrt{6}, -\frac{1}{24}\sqrt{6}, +\frac{1}{24}\sqrt{6}, +\frac{1}{8}\sqrt{6})$. We know from previous results (for example, De Leeuw and Stoop (1984)) that the equally spaced configuration z is the global minimizer of stress over $\mathbb{R}^4$. Of course it is far from unique, because all 24 permutations of z have the same function value, and are consequently also global minima. In fact, there are 24 local minima, which are all global minima as well. (paired)

In the example, we do not minimize over all of $\mathbb{R}^4$, but only over the subspace of linear combinations of y and z. These linear combinations, with coefficients α and β, are given by

$$x = \alpha y + \beta z = \frac{1}{24}\sqrt{6}\begin{bmatrix} -3\beta \\ -3\alpha - \beta \\ 3\alpha + \beta \\ 3\beta \end{bmatrix}, \tag{12.2}$$

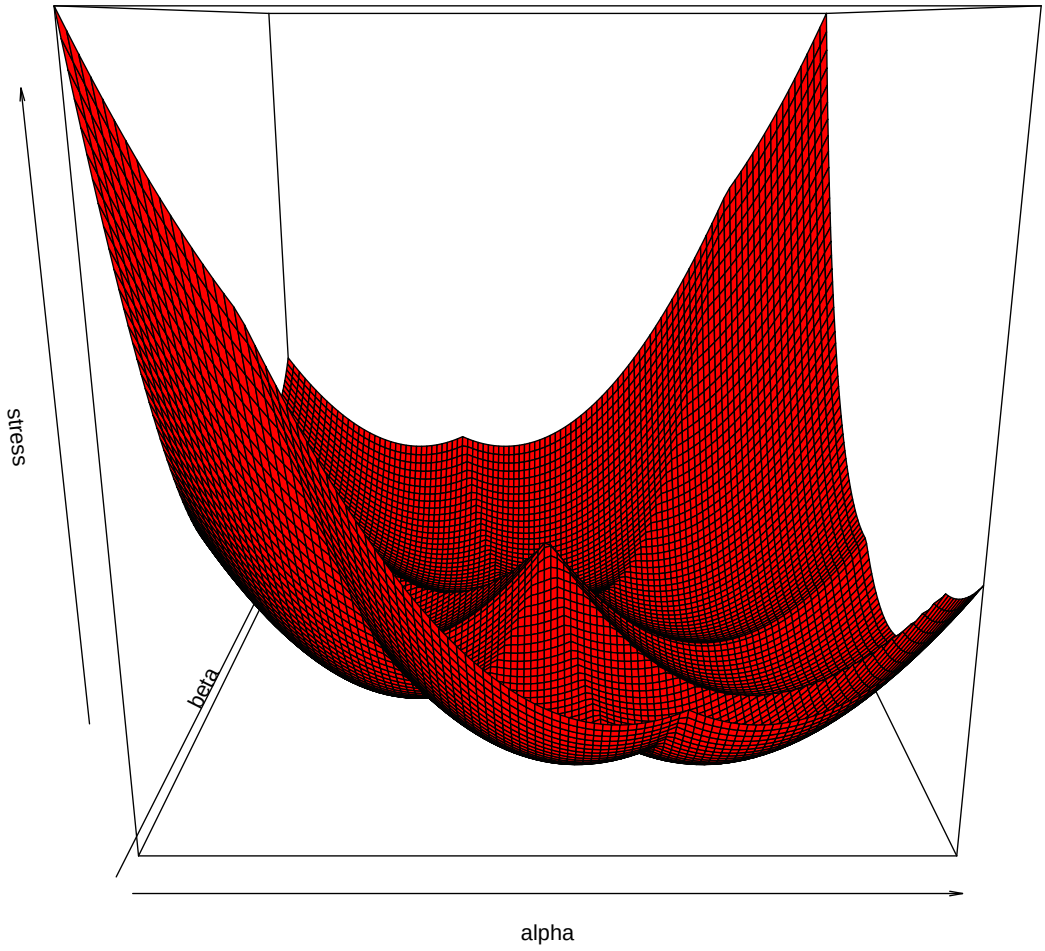
with distances

$$D(x) = \frac{1}{24}\sqrt{6}\begin{bmatrix} 0 & & & \\ |3\alpha - 2\beta| & 0 & & \\ |3\alpha + 4\beta| & |6\alpha + 2\beta| & 0 & \\ |6\beta| & |3\alpha + 4\beta| & |3\alpha - 2\beta| & 0 \end{bmatrix}. \tag{12.3}$$

We see that on the line $\beta = \frac{3}{2}\alpha$ both $d_{12}(x)$ and $d_{34}(x)$ are zero, on $\beta = -3\alpha$ we have $d_{23}(x) = 0$, on $\beta = 0$ we have $d_{14}(x) = 0$, and finally $d_{13}(x) = d_{24}(x) = 0$ on $\beta = -\frac{3}{4}\alpha$. On those lines, through the origin, stress is not differentiable.

12.1.1 Perspective

We first make a global perspective plot, with both α and β in the range $(-2.0, +2.0)$.

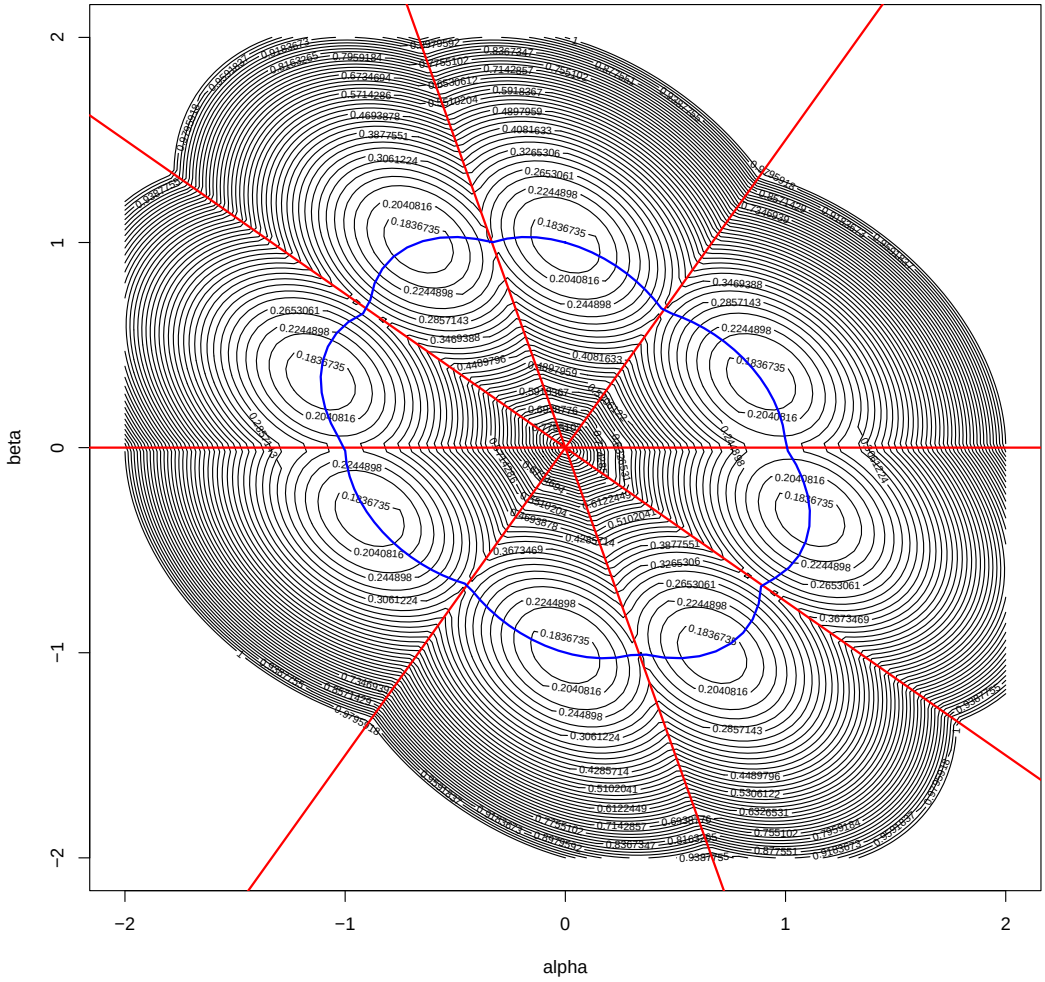


What do we see ? Definitely more ridges and valleys than in the two-dimensional example of chapter 2. In the one-dimensional case there is a ridge wherever two coordinates are equal, and thus one or more distances are zero. It is clear that at the bottom of each of the valleys there sits a local minimum.

12.1.2 Contour

A contour plot gives some additional details. In the plot we have drawn the four lines through the origin where one or more distances are zero (in red), and we have drawn the curve where $\eta^2(x) = \rho(x)$ (in blue). Thus all local

minima are on the blue line. The intersections of the red and the blue lines are the local minima of stress restricted to the red line. In those points there are both directions of ascent (along the red lines, in both directions) and of descent (into the adjoining valleys, in all directions).



We see once more the importance of the local minimum result from De Leeuw (1984c) that we discussed in section 2.5.2. The special relevbance of this result for UMDS was already pointed out by Pliner (1996). At a local minimum all distances are positive, and thus local minima must be in the interior of the eight cones defined by the four zero-distance lines. There are no saddle points, and only a single local maximum at the origin.

12.2 Order Formulation

Define an *isocone* as a closed convex cone of isotone vectors, and int(K) as its interior. Thus

$$K := \{x \in \mathbb{R}^n \mid x_{i_1} \leq \cdots \leq x_{i_n}\}, \tag{12.4}$$

and

$$\text{int}(K) = \{x \in \mathbb{R}^n \mid x_{i_1} < \cdots < x_{i_n}\}. \tag{12.5}$$

where $(i_1, \cdots, i_n)$ is a permutation of $(1, \cdots, n)$. There are $n!$ such closed isocones, and their union is all of $\mathbb{R}^n$. Thus $\min_x \sigma(x) = \min_{K \in \mathcal{K}} \min_{x \in K} \sigma(x) =$ where $\mathcal{K}$ are the $n!$ isocones.

For UMDS purposes the isocones are paired, because the negative of each configuration has the same distances between the n points, and thus the same stress. Thus each isocone and its negative cone are equivalent for UMDS, and we only have to consider $(n!)/2$ distinct orders.

Let us consider the problem of minimizing σ over a fixed $K \in \mathcal{K}$. Now

$$\rho(x) = \sum_{1 \leq i < j \leq n} \sum w_{ij} \delta_{ij} s_{ij} (x_i - x_j),$$

and $s_{ij} = \text{sign}(x_i - x_j)$ is the sign matrix of x.

$S(x)$ is the *sign matrix* of $x \in \mathbb{R}^n$ if $s_{ij}(x) = \text{sign}(x_i - x_j)$ for all i and j, i.e.

$$s_{ij}(x) := \begin{cases} +1 & \text{if } x_i > x_j, \\ -1 & \text{if } x_i < x_j, \\ 0 & \text{if } x_i = x_j. \end{cases} \tag{12.6}$$

The set of all sign matrices is $\mathcal{S}$.

Sign matrices are hollow and anti-symmetric. A sign matrix S is *strict* if its only zeroes are on the diagonal, i.e. $S = S(P\iota)$ for some permutation matrix P. The set of strict sign matrices is $\mathcal{S}_+$. Since there is a 1:1 correspondence between strict sign matrices and permutations, there are $n!$ strict sign

matrices. The row sums and column sums of a strict sign matrix are some permutation of the numbers $n - 2\iota + 1$.

For all $x \in \text{int}(K)$ the matrix S is the same strict sign matrix. Now

$$\rho(x) = \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n w_{ij}\delta_{ij}s_{ij}(x_i - x_j) = x't_K,$$

where t_K is the vector of row sums of the Hadamard product $W \times \Delta \times S$, or

$$\{t_K\}_i := \sum_{j=1}^n w_{ij}\delta_{ij}s_{ij}.$$

Again t_K only depends of K, not on x as long as $x \in \text{int}(K)$.

Thus on K

$$\sigma(x) = 1 - 2x't_K + x'Vx = 1 + (x - V^{-1}t_K)'V(x - V^{-1}t_K) - t_K'V^{-1}t_K.$$

If there are no weights the t_K were first defined using isocones in De Leeuw and Heiser (1977). They point out that minimizing $(x - V^{-1}t)'V(x - V^{-1}t)$ over $x \in K$ is a monotone regression problem (see 10.1).

A crucial next step is in De Leeuw (2005b), using the basic result in De Leeuw (1984c). De Leeuw (2005b) does use weights. We know if x is a local minimum then it must be in the interior of the isocone. If $V^{-1}t_K$ is not in interior, then monotone regression will creates ties, and thus x will not be in the interior either. In fact for local minima of UMDS it is necessary and sufficient that $V^{-1}t_K$ is in the interior of K. This result, without weights and in somewhat different language, is also in Pliner (1984). Thus we can limit our search to those isocones for which $V^{-1}t_K \in \text{int}(K)$. For those isocones, say the set $\mathcal{K}^\circ$, the local minimum is at $x = V^{-1}t_K$.

Thus

$$\min_{K\in\mathcal{K}}\min_{x\in K}\sigma(x) = 1 - \max_{K\in\mathcal{K}^\circ} t_K'V^{-1}t_K.$$

There is also an early short but excellent paper by Defays (1978), which derives basically the same result in a non-geometrical way. Defays does not use weights, so in his paper V^{-1} is $n^{-1}I$.

In the two-dimensional subspace of the example some of the $n!$ cones are empty.

12.3 Permutation Formulation

12.4 Sign Matrix Formulation

$$\rho(x) = \max_{S \in \mathcal{S}} \sum_{1 \leq i < j \leq n} \sum w_{ij} \delta_{ij} s_{ij} (x_i - x_j), \tag{12.7}$$

with the maximum attained for $S = S(x)$. If we define

$$t_i(y) := \sum_{j=1}^{n} w_{ij} \delta_{ij} s_{ij}(y), \tag{12.8}$$

then

$$\sigma(x) = \min_y \{1 + (x - V^{-1}t(y))'V(x - V^{-1}t(y)) - t(y)'V^{-1}t(y)\}. \tag{12.9}$$

This implies

$$\min_x \sigma(x) = 1 - \max_y \ t(y)'V^{-1}t(y) \tag{12.10}$$

12.5 Algorithms for UMDS

12.5.1 SMACOF

12.5.2 SMACOF (smoothed)

Now local minimum $x_i \neq x_j$

$$\min_{x \in K} \sigma(x) = \sum_{1 \leq i < j \leq n} \sum w_{ij} (\delta_{ij} - s_{ij}(x))(x_i - x_j))^2$$

Each isocone has a sign matrix (hollow, antisymmetric)

$$s_{ij}(x) = \begin{cases} +1 & \text{if } x_i > x_j, \\ -1 & \text{if } x_i < x_j, \\ 0 & \text{if } x_i = x_j. \end{cases}$$

$$\rho(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\delta_{ij}s_{ij}(x)(x_i - x_j) \geq \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\delta_{ij}s_{ij}(y)(x_i - x_j) = 2\sum_{i=1}^{n} x_i \sum_{j=1}^{n} w_{ij}\delta_{ij}s_{ij}(y)$$

Now

$$\rho(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\delta_{ij}s_{ij}(x)(x_i - x_j), \tag{12.11}$$

and for all $y \in \mathbb{R}^n$

$$\rho(x) \geq \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\delta_{ij}s_{ij}(y)(x_i - x_j) = 2\sum_{i=1}^{n} x_i \sum_{j=1}^{n} w_{ij}\delta_{ij}s_{ij}(y). \tag{12.12}$$

Stress is the maximum of a finite number of quadratics.

12.5.3 Branch-and-Bound

12.5.4 Dynamic Programming

12.5.5 Simulated Annealing

12.5.6 Penalty Methods

Chapter 13

Full-dimensional Scaling

13.1 Convexity

13.2 Optimality

13.3 Iteration

13.4 Cross Product Space

So far we have formulated the MDS problem in *configuration space*. Stress is a function of X, the $n \times p$ configuration matrix. We now consider an alternative formulation, where stress is a function of a positive semi-definite C or order n. The relevant definitions are

$$\sigma(C) := 1 - 2\rho(C) + \eta(C), \tag{13.1}$$

where

$$\begin{aligned} \rho(C) &:= \mathbf{tr}\ B(C)C, \\ \eta(C) &:= \mathbf{tr}\ VC, \end{aligned}$$

with

$$B(C) := \sum_{1\leq i<j\leq n}\sum \begin{cases} w_{ij}\frac{\delta_{ij}}{d_{ij}(C)}A_{ij} & \text{if } d_{ij}(C) > 0, \\ 0 & \text{if } d_{ij}(C) = 0. \end{cases}$$

and $d_{ij}^2(C) := \mathbf{tr}\ A_{ij}C$.

We call the space of all positive semi-definite $n \times n$ matrices *cross product space*. The problem of minimizing σ over $n \times p$-dimensional configuration space is equivalent to the problem of minimizing σ over the set of matrices C in $n \times n$-dimensional cross product space that have rank less than or equal to p. The corresponding solutions are related by the simple relationship $C = XX'$.

Theorem 13.1. *Stress is convex on cross product space.*

Proof. First, η is linear in C. Second,

$$\rho(C) = \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}\sqrt{\mathbf{tr}\ A_{ij}C}.$$

This is the weighted sum of square roots of non-negative functions that are linear in C, and it is consequently concave. Thus σ is convex. □

Unfortunately the subset of cross product space of all matrices with rank less than or equal to p is far from simple (see Datorro (2015)), so computational approaches to MDS prefer to work in configuration space.

13.5 Full-dimensional Scaling

Cross product space, the set of all positive semi-definite matrices, is a closed convex cone $\mathcal{K}$ in the linear space of all $n \times n$ symmetric matrices. This has an interesting consequence.

Theorem 13.2. *Full-dimensional scaling, i.e. minimizing σ over $\mathcal{K}$, is a convex programming problem. Thus in FDS all local minima are global. If $w_{ij}\delta_{ij} > 0$ for all i, j then the minimum is unique.*

This result has been around since about 1985. De Leeuw (1993) gives a proof, but the report it appeared in remained unpublished. A published proof is in "Inverse Multidimensional Scaling" (2007). Another treatment of FDS, with a somewhat different emphasis, is in De Leeuw (2014a).

Now, by a familiar theorem (Theorem 31.4 in Rockafellar (1970)), a matrix C minimizes σ over $\mathcal{K}$ if and only if

$$C \in \mathcal{K}, \tag{13.2}$$
$$V - B(C) \in \mathcal{K}, \tag{13.3}$$
$$\mathbf{tr}\ C(V - B(C)) = 0. \tag{13.4}$$

We give a computational proof of this result for FDS that actually yields a bit more.

Theorem 13.3. *For $\Delta \in \mathcal{K}$ we have*

$$\sigma(C + \epsilon\Delta) = \sigma(C) - 2\epsilon^{\frac{1}{2}} \sum_{\mathbf{tr}\ A_i C = 0} w_i \delta_i \sqrt{\mathbf{tr}\ A_i \Delta} + \epsilon\ \mathbf{tr}\ (V - B(C))\Delta + o(\epsilon). \tag{13.5}$$

::: {.proof} Simple expansion. :::

Theorem 13.4. *Suppose C is a solution to the problem of minimizing σ over $\mathcal{K}$. Then*

- $\mathbf{tr}\ A_{ij} C > 0$ *for all i, j for which $w_{ij}\delta_{ij} > 0$.*
- $V - B(C)$ *is positive semi-definite.*
- $\mathbf{tr}\ C(V - B(C)) = 0$.
- *If C is positive definite then $V = B(C)$ and $\sigma(C) = 0$.*

Proof. The $\epsilon^{\frac{1}{2}}$ term in (13.5) needs to vanish at a local minimum. This proves the first part. It follows that at a local minimum

$$\sigma(C + \epsilon\Delta) = \sigma(C) + \epsilon\ \mathbf{tr}\ (V - B(C))\Delta + o(\epsilon).$$

If $V - B(C)$ is not positive semi-definite, then there is a $\Delta \in \mathcal{K}$ such that $\mathbf{tr}\ (V - B(C))\Delta < 0$. Thus C cannot be the minimum, which proves the second part. If we choose $\Delta = C$ we find

$$\sigma((1+\epsilon)C) = \sigma(C) + \epsilon\ \mathbf{tr}\ (V - B(C))C + o(\epsilon).$$

and choosing ϵ small and negative shows we must have $\mathbf{tr}\ (V - B(C))C = 0$ for C to be a minimum. This proves the third part. Finally, if σ has a minimum at C, and C is positive definite, then from parts 2 and 3 we have $V = B(C)$. Comparing off-diagonal elements shows $\Delta = D(C)$, and thus $\sigma(C) = 0$. □

If C is the solution of the FDS problem, then $\mathbf{rank}(C)$ defines the *Gower rank* of the dissimilarities. The number of positive eigenvalues of the negative of the doubly-centered matrix of squared dissimilarities, the matrix factored in classical MDS, defines the *Torgerson rank* of the dissimilarities. The *Gower conjecture* is that the Gower rank is less than or equal to the Torgerson rank. No proof and no counter examples have been found.

We compute the FDS solution using the smacof algorithm

$$X^{(k+1)} = V^+ B(X^{(k)}) \tag{13.6}$$

in the space of all $n \times n$ configurations, using the identity matrix as a default starting point. Since we work in configuration space, not in crossproduct space, this does not guarantee convergence to the unique FDS solution, but after convergence we can easily check the necessary and sufficient conditions of theorem 13.4.

As a small example, consider four points with all dissimilarities equal to one, except δ_{14} which is equal to three. Clearly the triangle inequality is violated, and thus there certainly is no perfect fit mapping into Euclidean space.

The FDS solution turns out to have rank two, thus the Gower rank is two. The singular values of the FDS solution are

```
## [1] 0.4508464709 0.2125310645 0.0000001303
```

Gower rank two also follows from the eigenvalues of the matrix $B(C)$, which are

```
## [1] 1.0000000000 1.0000000000 0.9205543464
```

13.6 Ekman example

The Ekman (1954) color data give similarities between 14 colors.

```
##      434  445  465  472  490  504  537  555  584  600  610  628  651
## 445 0.86
## 465 0.42 0.50
## 472 0.42 0.44 0.81
## 490 0.18 0.22 0.47 0.54
## 504 0.06 0.09 0.17 0.25 0.61
## 537 0.07 0.07 0.10 0.10 0.31 0.62
## 555 0.04 0.07 0.08 0.09 0.26 0.45 0.73
## 584 0.02 0.02 0.02 0.02 0.07 0.14 0.22 0.33
## 600 0.07 0.04 0.01 0.01 0.02 0.08 0.14 0.19 0.58
## 610 0.09 0.07 0.02 0.00 0.02 0.02 0.05 0.04 0.37 0.74
## 628 0.12 0.11 0.01 0.01 0.01 0.02 0.02 0.03 0.27 0.50 0.76
## 651 0.13 0.13 0.05 0.02 0.02 0.02 0.02 0.02 0.20 0.41 0.62 0.85
## 674 0.16 0.14 0.03 0.04 0.00 0.01 0.00 0.02 0.23 0.28 0.55 0.68 0.76
```

We use three different transformations of the similarities to dissimilarities. The first is $1 - x$, the second $(1 - x)^3$ and the third $\sqrt[3]{1 - x}$. We need the following iterations to find the FDS solution (up to a change in loss of 1e-15).

```
## power =  1.00  itel =    6936  stress =  0.0000875293
## power =  3.00  itel =     171  stress =  0.0110248119
## power =  0.33  itel =     423  stress =  0.0000000000
```

For the same three solutions we compute singular values of the thirteen-dimensional FDS solution.

```
##  [1] 0.1797609824 0.1454675297 0.0843865491 0.0777136109 0.0486123551
##  [6] 0.0393576522 0.0236290817 0.0162344515 0.0072756171 0.0000031164
## [11] 0.0000000009 0.0000000000 0.0000000000
##
##  [1] 0.2159661347 0.1549184093 0.0000000727 0.0000000041 0.0000000000
##  [6] 0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000
```

```
## [11] 0.0000000000 0.0000000000 0.0000000000
##
##  [1] 0.1336126813 0.1139019875 0.0880453752 0.0851609618 0.0710424935
##  [6] 0.0664988952 0.0561005006 0.0535112029 0.0492295395 0.0479964575
## [11] 0.0468628701 0.0410193579 0.0388896490
```

Thus the Gower ranks of the transformed dissimilarities are, repectively, nine (or ten), two, and thirteen. Note that for the second set of dissimilarities, with Gower rank two, the first two principal components of the thirteen-dimensional solution are the global minimizer in two dimensions. To illustrate the Gower rank in yet another way we give the thirteen non-zero eigenvalues of $V^+B(X)$, so that the Gower rank is the number of eigenvalues equal to one. All three solutions satisfy the necessary and sufficient conditions for a global FDS solution.

```
##  [1] 1.0000000432 1.0000000222 1.0000000012 1.0000000005 1.0000000002
##  [6] 1.0000000001 1.0000000000 1.0000000000 0.9999993553 0.9989115116
## [11] 0.9976821885 0.9942484083 0.9825147154
##
##  [1] 1.0000000000 1.0000000000 0.9234970864 0.9079012130 0.8629365849
##  [6] 0.8526920031 0.8298036209 0.8145561677 0.7932385763 0.7916517225
## [11] 0.7864426781 0.7476794757 0.7282682474
##
##  [1] 1.0000000820 1.0000000241 1.0000000047 1.0000000009 1.0000000004
##  [6] 1.0000000003 1.0000000001 1.0000000001 1.0000000001 1.0000000000
## [11] 0.9999999999 0.9999999689 0.9999999005
```

We also plot the first two principal components of the thirteen-dimensional FDS solution. Not surprisingly, they look most circular and regular for the solution with power three, because this actually is the global minimum over two-dimensional solutions. The other configurations still have quite a lot of variation in the remaining dimensions.

Figure @ref{fig:ekmantrans} illustrates that the FDS solution with power 3 is quite different from power 1 and power one 1/3 Basically the transformations with lower powers result in dissimilarity measures that are very similar to Euclidean distances in a high-dimensional configuration, while power equal to 3 makes the dissimilarties less Euclidean. This follows from metric transform

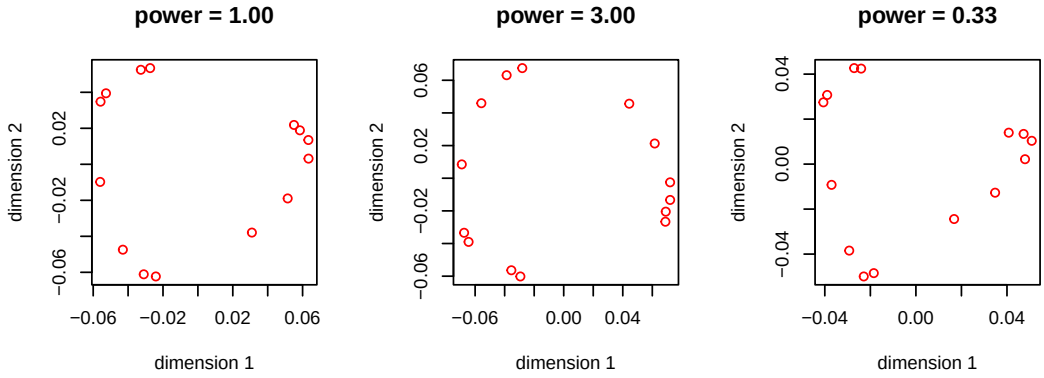


Figure 13.1: Ekman data, configurations for three powers

theory, where concave increasing transforms of finite metric spaces tend to be Euclidean. In particular the square root transformation of a finite metric space has the Euclidean four-point property, and there is a $c > 0$ such that the metric transform $f(t) = ct/(1+ct)$ makes a finite metric space Euclidean (Maehara (1986)).

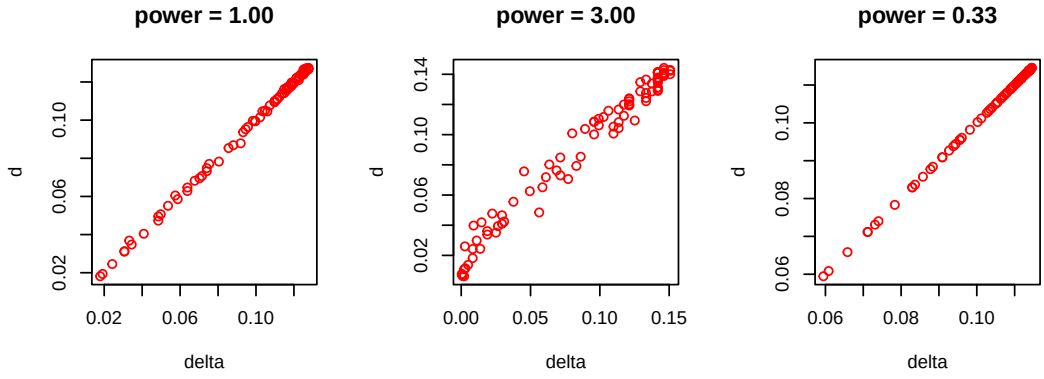


Figure 13.2: Ekman data, fit plots for three powers

Chapter 14

Unfolding

In unfolding the objects are partitioned into two sets of, say, n and m objects and only the nm between-set dissimilarities are observed. The within-set weights are zero. Thus we minimize

$$\sigma(X,Y) = \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}(\delta_{ij} - d(x_i, y_j))^2 \tag{14.1}$$

over $X \in \mathbb{R}^{n\times p}$ and $Y \in \mathbb{R}^{m\times p}$.

$\frac{1}{2}\{(n+m)(n+m-1) - 2nm\}$ are missing

note that we can have $d_{ij}(X) = 0$ at a local minimum.

14.1 Algebra

The missing data in unfolding complicate the MDS problem, in the same way as the singular value decomposition of a rectangular matrix is more complicated than the eigen decomposition of a symmetric matrix.

The problem we want to solve in this section is recovering X and Y (up to a translation and rotation) from $D(X,Y)$.

The first matrix algebra results in metric unfolding were due to Ross and Cliff (1964). An actual algorithm for the "ignore-errors" case was proposed by Schönemann (1970). Schönemann's technique was studied in more detail by Gold (1973) and Heiser and De Leeuw (1979).

This sectiom discusses a slightly modified version of Schönemann (1970).

First, we compute the Torgerson transform $E(X,Y) = -\frac{1}{2}JD^{(2)}(X,Y)J$ It was observed for the first time by Ross and Cliff (1964) that $E(X,Y) = JXY'J$.

Assume that $E(X,Y) = JXY'J$ is a full-rank decomposition, and that the rank of $E(X,Y)$ is r. Note that there are cases in which the rank of JX or JY is strictly smaller than the rank of X or Y. If X, for example, has columns x and $e-x$, with x and e linearly independent, then its rank is two, while JX with columns Jx and $-Jx$ has rank one.

Suppose $E(X,Y) = GH'$ is another full-rank decomposition. Then there exist vectors u and v with r elements and a non-singular T of order r such that

$$\begin{aligned} X &= GT + eu', \\ Y &= HT^{-t} + ev'. \end{aligned} \tag{14.2}$$

We can assume without loss of generality that the centroid of the X configuration is in the origin, so that $JX = X$, and $u = 0$ in the first equation of (14.2).

We use the QR decomposition to compute the rank r of $E(X,Y)$, and the factors G and H.

We now use (14.2) to show that $F = D^{(2)}(X,Y) + 2GH'$ is of the form $F = \gamma + \alpha e' + e\beta'$, with $\gamma = v'v$ and $M = TT'$.

$$\begin{aligned} \alpha_i &= g_i'Mg_i - 2g_i'Tv, \\ \beta_j &= h_j'M^{-1}h_j + 2h_j'T^{-t}v. \end{aligned} \tag{14.3}$$

It follows that $JF = J\alpha e'$ and $FJ = e\beta'J$. Thus $J\alpha$ is any column of JF and $J\beta$ is any row of FJ.

Consider the first equation of (14.3). For the time being, we ignore the second one. Suppose M_k is a basis for the space of real symmetric matrices of order p with the $\frac{1}{2}p(p+1)$ elements $e_se_t' + e_te_s'$ for $s \neq t$ and e_se_s' for the diagonal. Define $q_{ik} := g_i'M_kg_i$. Then

$$J\alpha = J\begin{bmatrix} Q & -2G \end{bmatrix}\begin{bmatrix} \mu \\ Tv \end{bmatrix}, \tag{14.4}$$

with μ the coordinates of M for the basis M_k.

Equations (14.4) are n linear equations in the $\frac{1}{2}p(p+1)+p = \frac{1}{2}p(p+3)$ unknowns μ and Tv. Assume they have a unique solution. Then $M = \sum \mu_k M_k$ is PSD, and can be eigen-decomposed as $M = K\Lambda^2 K'$. Set $T = K\Lambda$

```
set.seed(12345)
x <- matrix (rnorm(16), 8, 2)
x <- apply (x, 2, function (x) x - mean (x))
y <- matrix (rnorm(10), 5, 2)
a <- rowSums (x ^ 2)
b <- rowSums (y ^ 2)
d <- sqrt (outer(a, b, "+") - 2 * tcrossprod (x, y))
```

14.1.1 One-dimensional

The one-dimensional case is of special interest, because it allows us to construct an single joint metric scale for row objects and column objects from metric dissimilarities. We have to find a solution to $\delta_{ij} = |x_i - y_j|$, without making assumptions about the order of the projections on the dimension. Compute any solution for Jg and Jh from $\tau(\Delta^{(2)}) = Jgh'J$. For data with errors we would probably use the SVD. Assume without loss of generality that $Jg = g$. Then the general solution is $x = \tau g$ and $y = \tau^{-1}h + \nu e$ for some real τ and ν.

Now

$$\Delta^2 = \tau^2 g^{(2)} e' + \tau^{-2} e (h'_j)^{(2)} + \nu^2 E - 2gh' - 2\tau\nu g_i e' \tag{14.5}$$

are nm equations in the two unknowns (τ, ν). They can be solved by many methods, but we go the Schönemann way. Column-centering gives

$$J(\Delta^{(2)} + 2g_j h_j) = \tau^2 J g^{(2)} - 2\tau\nu g, \tag{14.6}$$

while row-centering gives

$$(\Delta^{(2)} + 2g_j h_j)J = \tau^{-2} e (h^{(2)})' J. \tag{14.7}$$

14.2 Classical Unfolding

Multidimensional unfolding as a data analysis technique was introduced by Coombs (1964).

bennett-hays hays-bennett bennett

SMACOF - Heiser and De Leeuw (1979)

Form of V

What happens to nonzero theorem ? within-set distances can be zero

$\Delta = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$

$x_1 = x_2 = 0$ $y_1 = 1, y_2 = 2, y_3 = 3$

14.3 Nonmetric Unfolding

row-conditional busing van deun deleeuw_R_06a

Stress3 – Roskam

14.3.1 Degenerate Solutions

What are they

14.3.1.1 Which Stress

$$\sigma(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(\delta_{ij} - d_{ij}(X))^2$$

Weak order, plus normalization. Two-point solution.

14.3.1.2 l'Hôpital's Rule

We all know that 0/0 is not defined and should be avoided at all cost. But then again we have

$$\lim_{x \to 0} \frac{\sin(x)}{x} = \cos(0) = 1,$$

and in fact $\sup_x \frac{sin(x)}{x} = 1$. Or, for that matter, if f is differentiable at x then

$$\lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon} = f'(x)$$

If $f : \mathbb{R} \Rightarrow \mathbb{R}$ and $g : \mathbb{R} \Rightarrow \mathbb{R}$ are two functions

- differentiable in an interval $\mathcal{I}$, except possibly at $c \in \mathcal{I}$,
- $g'(x) \neq 0$ for all $x \in \mathcal{I}$,
- $\lim_{x \to c} f(x) = \lim_{x \to c} g(x) = 0$ or $\lim_{x \to c} f(x) = \lim_{x \to c} g(x) = \pm\infty$, then

$$\lim_{x \to c} \frac{f(x)}{g(x)} = \lim_{x \to c} \frac{f'(x)}{g'(x)}$$

We use l'Hôpital's rule in chapter 14, section 14.3.1 on degeneracies in nonmetric unfolding. We have not explored the multivariate versions of l'Hôpitals rule, discussed for example by Lawlor (2020).

Illustration.

$\delta_{12} > \delta_{13} = \delta_{23}$.

$d_{12}(X_\epsilon) = 1$

$d_{13}(X_\epsilon) = d_{23}(X_\epsilon) = 1 + \frac{1}{2}\epsilon$.

Then

$\lim_{\epsilon \to 0} D(X_\epsilon) = \Delta$.

euclidean for $\epsilon \geq -1$

14.3.1.3 Penalizing

14.3.1.4 Restricting Regression

Busing

Van Deun

Chapter 15

Constrained Multidimensional Scaling

As we have seen in section 1.4 Constrained Multidimensional Scaling or CMDS is defined as the generalization of basic MDS in which we want to solve

$$\min_{X\in\Omega} \sum\sum_{1\leq i<j\leq n} w_{ij}(\delta_{ij} - d_{ij}(X))^2,$$

where Ω is a subset of $\mathbb{R}^{n\times p}$. Of course there are also versions of CMDS in which the dissimilarities are nonmetric and must be quantified or transformed accordingly. But in this chapter we concentrate on the ways in which we constrain the configuration and on the ways to incorporate this into the smacof framework.

Constraints on X can be defined in many different ways. They can be in *parametric form*, using a map $F : \Theta \Rightarrow \mathbb{R}^{n\times p}$. Thus the constraints are $X = F(\theta)$ and θ varies in some subset Θ of real parameter space. Alternatively we can haveconstraints in *dual form*, i.e, have a maps F on $\mathbb{R}^{n\times p}$ and define Ω by $F(X) = 0$ (or $F_1(X) = 0$ and $F_2(X) \geq 0$). If F is smooth both parametric and dual forms define manifolds in $\mathbb{R}^{n\times p}$, and often constraints can be equivalently expressed in both forms. Requiring the points in the configuration to be on the unit circle, for example, has the parametric form $x_i = (\sin\theta_i, \cos\theta_i)$ and the dual form $\|x_i\|^2 = 1$.

Bentler-Weeks

$$\sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij}-(\alpha\ d_{ij}(X)+\beta))^2.$$

with $x_{is}=K$ or $x_{is}=wy_{is}$ for some set of (i,s).

Configuration-Distances $F(D(X))\geq 0$ and $G(D(X))=0$ Borg-Lingoes

$$\min_{D\in\mathcal{D}}\sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij}-d_{ij})^2,$$

De Leeuw-Heiser

15.1 Primal-Dual (note: the base partitioning has dual aspects)

Least squares

$$\sigma_\lambda(X):=\sigma(X)+\lambda\min_{Y\in\Omega}\eta^2(X-Y)$$
$$\sigma_\lambda(X):=\sigma(X)+\lambda\min_{\Delta\in\mathcal{D}}\sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij}-d_{ij}(X)^2$$

15.2 Basic Partitioning

A comprehensive smacof approach to constrained MDS was developed in De Leeuw and Heiser (1980). It is a primal method that does not involve penalty parameters, and it defines the constraints directly on the configuration.

The starting point is the *majorization partitioning*

$$\sigma(X)\leq 1+\eta^2(X-\gamma(Y))-\eta^2(\gamma(Y)),\tag{15.1}$$

with equality, of course, if $X=Y$.

Note similarity with dual approach

The smacof algorithm for constrained MDS has consequently two steps. In the first we compute the Guttman transform of the current configuration,

and in the second we find the metric projection of this Guttman transform on the constraint set (in the metric defined by V). Thus, in shorthand,

$$X^{(k+1)} \in \underset{Y\in\Omega}{\text{Argmin}}\ \eta^2(Y - \Gamma(X^{(k)})). \tag{15.2}$$

To emphasize we look for a fixed point of the composition of two maps, the Guttman transform and the projection operator Π_Ω, we can write in even shorter hand

$$X^{(k+1)} \in \Pi_\Omega(\Gamma(X^{(k)}))$$

The smacof formulation of the CMDS problem is elegant, if I say so myself, but it is not always simple from the computational point of view. The Guttman transform is easy enough to compute, but projecting on Ω in the V metric may be complicated, depending on how Ω is defined. In this chapter we will discuss a number of examples with varying degrees of difficulty in computing the *smacof projection*.

15.3 Unweigthing

For some types of constraints, for example the circular and elliptical MDS discussed in section 15.6, unweighted least squares is computationally simpler than weighted least squares. In those cases it generally pays to use majorization to go from a weighted to an unweighted problem (see also Groenen, Giaquinto, and Kiers (2003)). This will tend to increase the number of iterations of smacof, but the computation within each iteration will be considerably faster.

From equation (15.2), the projection problem in constrained MDS is to minimize the weighted least squares loss function $\phi(X) := \text{tr}\ (Z-X)'V(Z-X)$ over $X \in \Omega$. Now suppose θ is the largest eigenvalue of V, so that $V \lesssim \theta I$, and suppose $Y \in \Omega$. Then

$$\begin{aligned}\phi(X) = \text{tr}\ ((Z-Y)-(X-Y))'V((Z-Y)-(X-Y)) \leq \\ \phi(Y) - 2\ \text{tr}\ (Z-Y)'V(X-Y) + \theta\ \text{tr}\ (X-Y)'(X-Y).\end{aligned} \tag{15.3}$$

Completing the square gives the majorization

$$\phi(X) \leq \phi(Y) + \theta \text{ tr } (X - \overline{Y})'(X - \overline{Y}) - \theta \text{ tr } \overline{Y}'\overline{Y}, \tag{15.4}$$

with $\overline{Y}$ the matrix-convex combination

$$\overline{Y} := (I - \frac{1}{\theta}V)Y + \frac{1}{\theta}VZ. \tag{15.5}$$

The weighted projection problem from equation (15.2) is replaced by one or more inner iterations of an unweighted projection problem. Set $X^{(k,1)} = X^{(k)}$ and

$$X^{(k,l+1)} \in \underset{Y \in \Omega}{\text{Argmin}} \text{ tr } (Y - \overline{X}^{k,l})'(Y - \overline{X}^{(k,l)}). \tag{15.6}$$

After stopping the inner iterations at $X^{(k,l+s)}$ we set $X^{(k+1)} = X^{(k,l+s)}$. All $X^{(k,l)}$ remain feasible, loss decreases in each inner iteration, and as long at the metric projections are continuous the map from $X^{(k)}$ to $X^{(k+1)}$ is continuous as well.

15.4 Constraints on the Distances

15.4.1 Rectangles

15.5 Linear Constraints

15.5.1 Uniqueness

$X = (Z \mid D)$

$X = (Z \mid \alpha I)$

Distance smoothing

15.5.2 Combinations

$X = \sum \alpha_r Z_r$

15.5.3 Step Size

$X = Z + \alpha G$

15.5.4 Single Design Matrix

$X = ZU$

15.5.5 Multiple Design Matrices

$x_s = G_s u_s$

$$d_{ij}^2(X) = \sum_{s=1}^{p} x_s' A_{ij} x_s = \sum_{s=1}^{p} u_s' G_s' A_{ij} G_s u_s$$

15.6 Circular MDS

De Leeuw (2007c), De Leeuw (2007a), De Leeuw (2005a)

There are situations in which it is desirable to have a configuration with points that are restricted to lie on some surface or manifold in $\mathbb{R}^p$. Simple examples are the circle in $\mathbb{R}^2$ or the sphere in $\mathbb{R}^3$. Some applications are discussed in T. F. Cox and Cox (1991) (also see T. F. Cox and Cox (2001), section 4.6), in Borg and Lingoes (1980), in Papazoglou and Mylonas (2017), and in De Leeuw and Mair (2009), section 5. The most prominent actual examples are probably the color circle and the spherical surface of the earth, but there are many other cases in which MDS solutions show some sort of "horseshoe" (De Leeuw (2007a)).

Figure 15.1: John van de Geer

15.6.1 Some History

Permit me to insert some personal history here. Around 1965 I got to work at the Psychological Institute. At the time Experimental Psychology and Methodology were in the same department, with John van de Geer as its chair. John had a long-running project with Pim Levelt and Reinier Plomp at the Institute for Perception RVO/TNO on perceptual and cognitive aspects of musical intervals. In Van de Geer, Levelt, and Plomp (1962), for example, they used various cutting-edge techniques at the time, the semantic differential for data collection, the centroid method for factor analysis, and oblique simple structure rotation. A couple of years later the cutting edge had moved to triadic comparisons for data collection and nonmetric multidimensional scaling (Levelt, Van De Geer, and Plomp (1966)). The analysis in Levelt, Van De Geer, and Plomp (1966) revealed a parabolic horseshoe structure of the musical intervals.

This inspired John to find a technique to fit quadratic (and higher order, if necessary) structures to scatterplots. If X is a two-dimensional configuration of n points, then form the $n \times 6$ matrix Z with columns $1, x_1, x_2, x_i^2, x_2^2, x_1x_2$. Now find α with $\alpha'\alpha = 1$ such that $\alpha'Z'Z\alpha$ is as small as possible. This gives the normalized eigenvector corresponding with the smallest eigenvalue of $Z'Z$, or, equivalently, the right singular vector corresponding with the smallest singular value of Z. It is easy to see how this approach generalizes to more dimensions and higher order algebraic surfaces. I remember with how much awe this technique was received by the staff of the Psychological Institute. It probably motivated me in 1966 to develop similar techniques and get my portion of awe.

Levelt, Van De Geer, and Plomp (1966) used the curve fitting technique to draw the best fitting parabola in the two-dimensional scatterplot of musical intervals. In their discussion they suggested that a similar quadratic structure could be found if similarities between political parties were analyzed, because for people in the middle of the left-right scale extreme-left and extreme-right parties would tend to be similar. If the effect of extremity was strong enough, the two extreme might even bend towards each other, leading to an ellipse rather than a parabola. In 1966 John asked student-researcher Dato de Gruijter to figure out if this curving back actually happened, which lead to De Gruijter (1967). Dato collected triadic comparisons between nine Dutch political parties, cumulated over 100 psychology students. The curve fitting

technique indeed found best fitting ellipses.

15.6.2 Primal Methods

We follow De Leeuw and Mair (2009) in distinguishing *primal* and *dual* methods. In a primal method the surface we fit is specified in parametric form. The points on the circle, for example, have $(x_{i1}, x_{i2}) = (sin(\xi_i), cos(\xi_i))$. Rotational invariance of MDS means we can assume the center of the circle is in the origin. This is the approach of T. F. Cox and Cox (1991). They substitute the parametrix expression for the circle in the formula for stress and minimize over the sperical coordinates ξ_i using gradient methods. They develop a similar method for the sphere in $\mathbb{R}^3$. For those who want to go to higher dimensions we illustrate a parametric representation for $\mathbb{R}^4$.

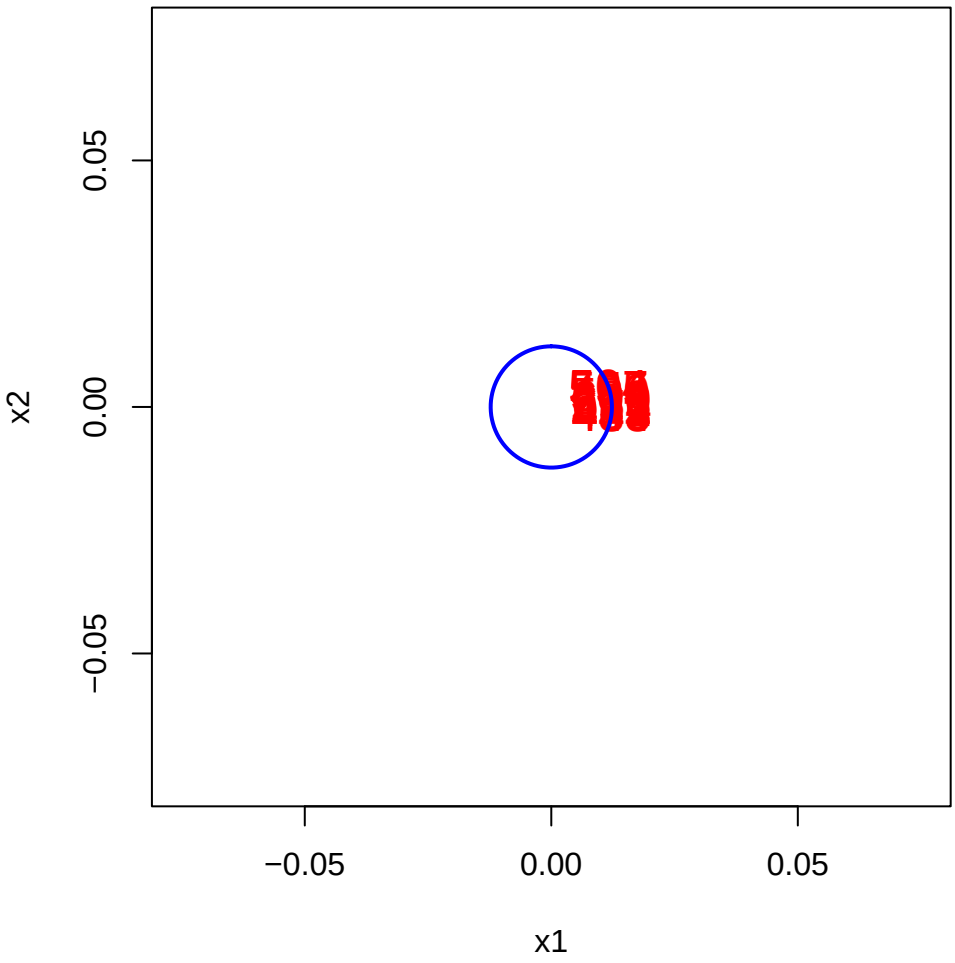
$$(x_{i1}, x_{i2}, x_{i3}, x_{i4}) = (\sin(\xi_i)\cos(\theta_i)\sin(\mu_i),\ \sin(\xi_i)\cos(\theta_i)\cos(\mu_i),\ \sin(\xi_i)\sin(\theta_i),\ \cos(\xi_i)).$$

Spherical coordinates soon get tedious, and De Leeuw and Mair (2009) simply require the distances of all points to the origin to be the same constant. Note that this puts the center of the fitted sphere in the origin, which means that in general the center of the point cloud cannot be taken to be in the origin as well. In smacof we use the

$$\phi(X, \lambda) := \text{tr}\ (Z - \lambda X)'V(Z - \lambda X)$$

over the radius λ and the configuration X, which is constrained to have diag $XX' = I$. De Leeuw and Mair (2009) project out λ and minimize $\phi(X, \star) := \min_\lambda \phi(X, \lambda)$ over X, using Dinkelbach majorization (Dinkelbach (1967)), a block relaxation that cycles over rows of X. Solving for each p vector of coordinates requires solving a secular equation.

Here we proceed slightly differently. Our first step is to get rid of V using the formulas in 15.3.



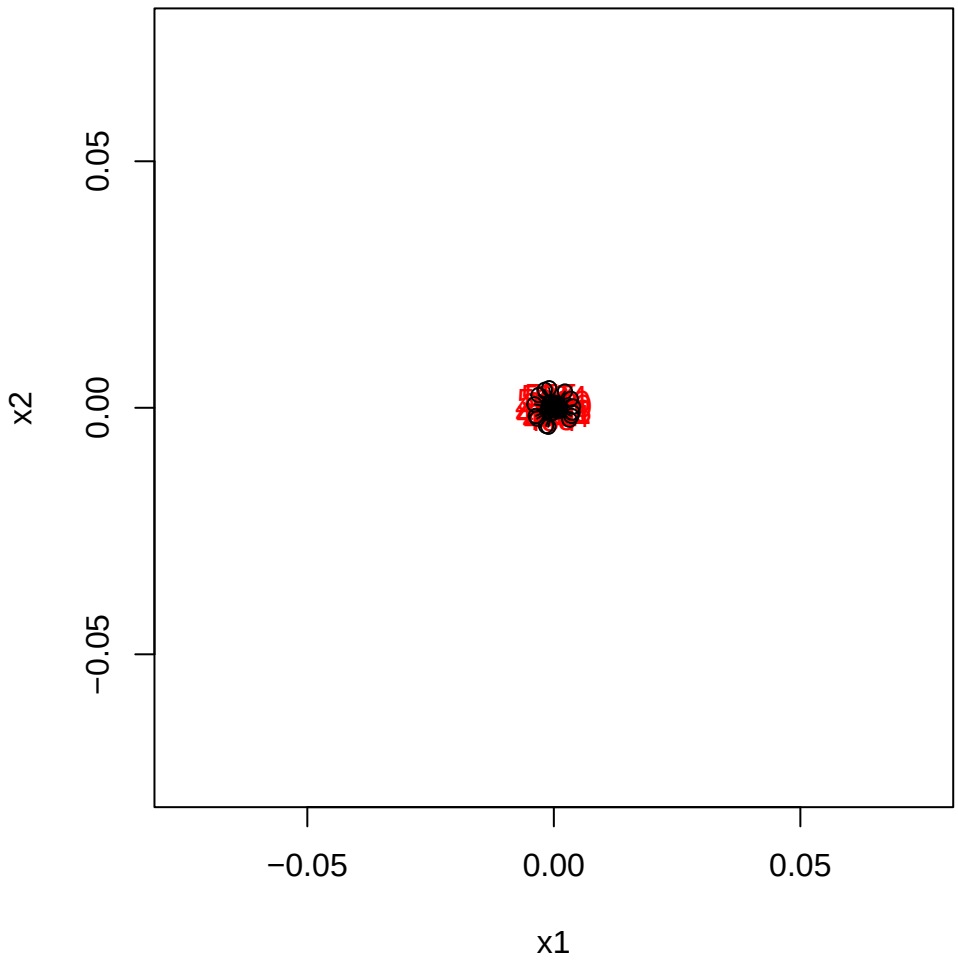
After 2 iterations the primal method converges to a stress value of 0.4654367. The circle has radius 0.0122974.

15.6.3 Dual Methods

In a dual method we use unrestricted smacof, but we add a penalty to the loss if the configurations do not satisfy the constraints. We use a quadratic penalty, mainly because that fits seamlessly into the smacof approach.

We add one point, the center of the circle, with coordinates x_0 to the configuration, and we require that all n other points have an equal distance from the center. The n dissimilarities $\delta_{0,i}$ are unknown, so we use alternating

least squares and estimate the missing dissimilarities by minimizing stress over them, requiring them to be all equal. All weights $w_{0,i}$ are chosen equal to the penalty parameter ω. The solution for the common $\delta_{0,i}$ is obviously the average of the n distances $d_{0,i}$. In this case it is not necessary to use majorization to transform to unweighted least squares.



15.7 Elliptical MDS

15.7.1 Primal

The smacof projection problem for a p-axial ellipsoid minimizes

$$\phi(Y,\Lambda) := \text{tr } (Z - Y\Lambda)'V(Z - Y\Lambda)$$

with diag $YY' = I$ and with Λ diagonal and PSD.

ALS

Minimizing ϕ over Λ for fixed Y is easy. For dimension s we have

$$\lambda_s = \frac{y_s'Vz_s}{y_s'Vy_s}.$$

To minimize ϕ over Y for fixed Λ we use $Z - Y\Lambda = (Z\Lambda^{-1} - Y)\Lambda$ so that

$$\phi(Y,\Lambda) = \text{tr } \Lambda^2(\tilde{Z} - Y)'V(\tilde{Z} - Y)$$

with $\tilde{Z} = Z\Lambda^{-1}$. We now use a slight modification of the majorization technique in section 15.3. Set $Y = Y_{\text{old}} + (Y - Y_{\text{old}})$. Then

$$\phi(Y,\Lambda) = \text{tr } \Lambda^2((\tilde{Z}-Y_{\text{old}})-(Y-Y_{\text{old}}))'V((\tilde{Z}-Y_{\text{old}})-(Y-Y_{\text{old}})) = \phi(Y_{\text{old}},\Lambda)-2\,\text{tr } \Lambda^2(\tilde{Z}-Y_{\text{old}})'V(Y-Y_{\text{old}})+$$

$$\text{tr } \Lambda^2(Y - Y_{\text{old}})'V(Y - Y_{\text{old}}) \leq \theta\lambda_{\max}^2 \text{ tr } (Y - Y_{\text{old}})'(Y - Y_{\text{old}})$$

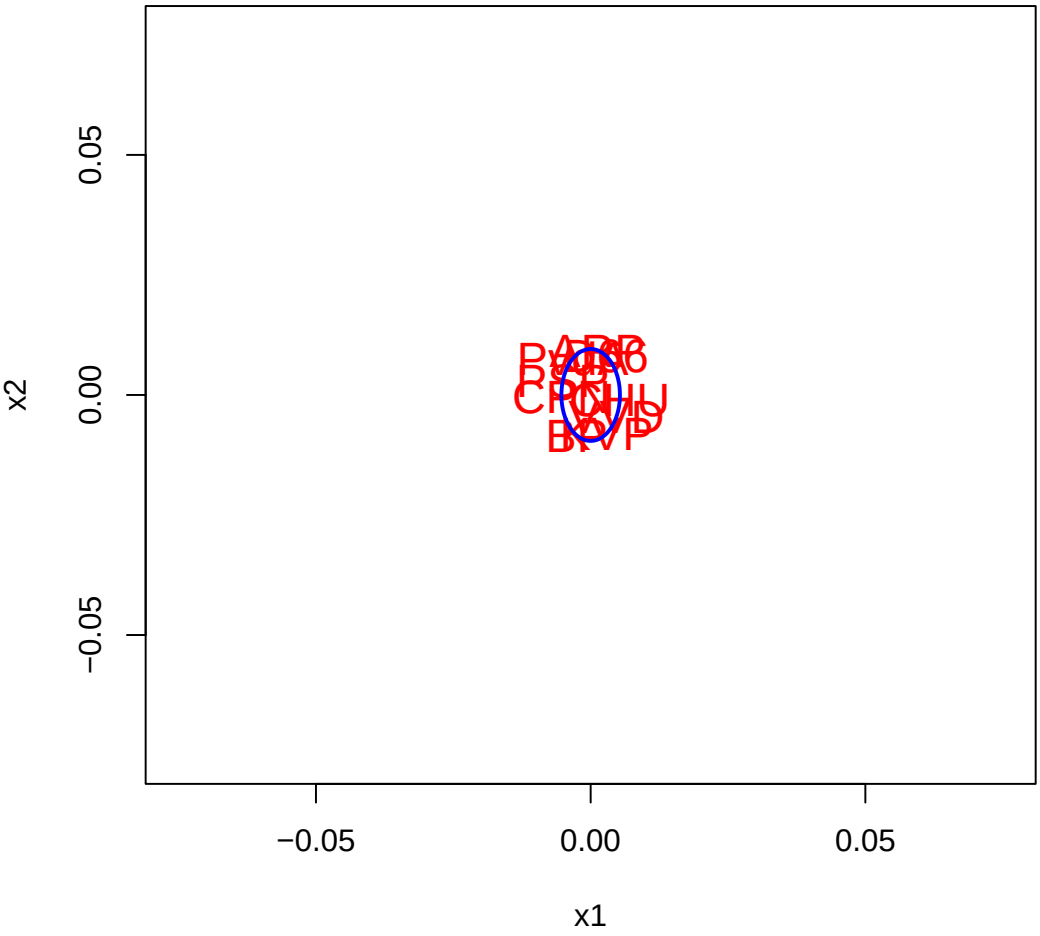
where, as before, θ is the largest eigenvalue of V.

$$\theta\lambda_{\max}^2 (Y - Y_{\text{old}}) = V(\tilde{Z} - Y_{\text{old}})\Lambda^2$$

(abadir_magnus_05, p 283)

Normalize the rows of

$$Y_{\text{old}} + \frac{1}{\theta\lambda_{\max}^2}V(Z\Lambda^{-1} - Y_{\text{old}})\Lambda^2$$



0.0053277, 0.0095667

2

0.415335

15.7.2 Dual

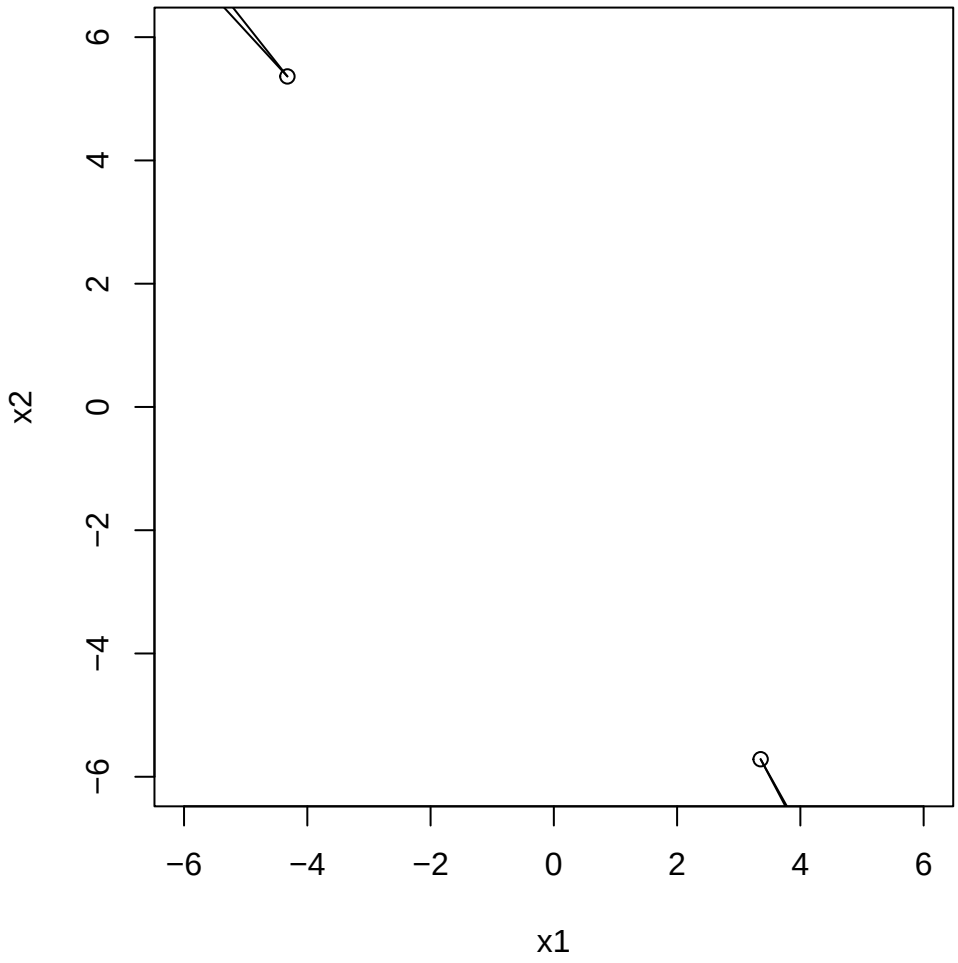
We will only develop a dual method for ellipses in two dimensions, because there is no easy characterization in terms of distances in higher dimensions (that I know of). But in two dimensions the famous pin-and-string construction uses the fact that for all points on the ellipse the sum of the distances to

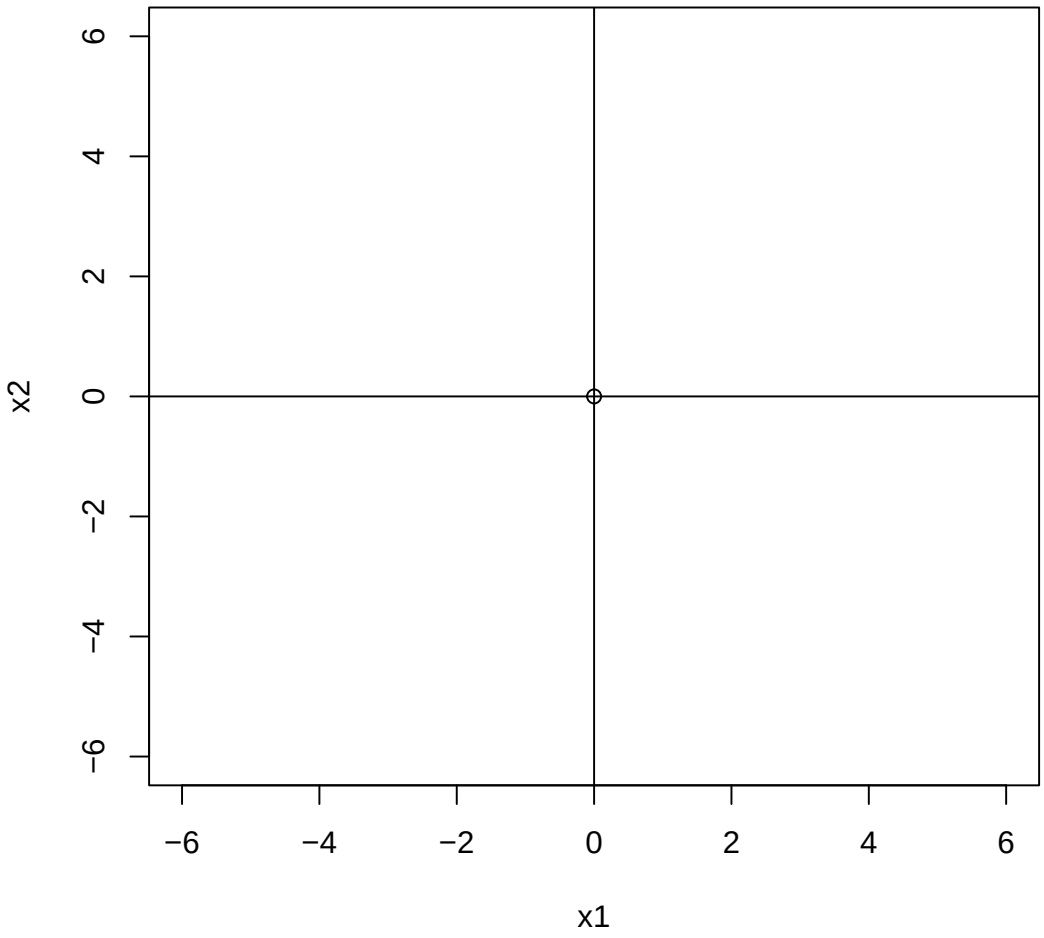
two focal points is constant. Thus our dual method now adds two points to the n points in the configuration, chooses the weights for the $2n$ components of stress to be the penalty parameter w, and finds the $2n$ unknown dissimilarities between the two focal points and the n points on the ellipse to add up to a constant.

This means we have to minimize tr $(\Delta - D)'(\Delta - D)$ over Δ satisfying $\Delta e = \gamma e$, where for the time being Δ and D are $n \times 2$ submatrices. The Lagrangian is tr $(\Delta - D)'(\Delta - D) - 2\mu'(\Delta e - \gamma e)$, and thus we must have $\Delta = D + \mu e'$. Taking row sums gives $\gamma e = De + p\mu$ and thus $\mu = \frac{1}{2}(\gamma e - De)$. This implies $\Delta - D = \mu e' = \frac{1}{2}(\gamma E_{np} - DE_{pp})$, and to minimize loss over γ we choose $\gamma = \frac{1}{n} e_n' D e_p$. This gives

$$\Delta = DJ + \frac{e'De}{2n} ee'.$$

Thus we take D, transform its n rows to deviations from the mean, and then add the overall mean to all elements.


x2
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
x1



hyperbola: difference of distances constant $|d((x_i, x_2), (f_1, 0)) - d((x_i, x_2), (g_1, 0))| = c$

parabola: equal distance from the focus point and the directrix (horizontal axis) $d((x_i, x_2), (f_1, f_2)) = d((x_1, x_2), d(x_1, 0))$

15.8 Distance Bounds

De Leeuw (2017d) De Leeuw (2017c) De Leeuw (2017b)

15.9 Localized MDS

15.10 MDS as MVA

Q methodology

http://qmethod.org

Stephenson (1953)

De Leeuw and Meulman (1986) Meulman (1986) Meulman (1992)

15.11 Horseshoes

Chapter 16

Individual Differences

This chapter deals with the situation in which we observe more than one set of dissimilarities. We need an extra index $k = 1, \cdots, m$ for Δ_k, W_k, and for X_k. The definition of stress becomes

$$\sigma(X_1, \cdots, X_m) := \frac{1}{2} \sum_{k=1}^{m} \sum_{1 \leq i < j \leq n} \sum w_{ijk}(\delta_{ijk} - d_{ij}(X_k))^2 \tag{16.1}$$

In order make it interesting we have to constrain the X_k in some way or other, preferable one in which the different X_k are linked, so they have something in common and something in which they differ.

MDS with linking constraints on the configurations is known in the psychometric literature as MDS with individual differences. This does not imply that index k necessarily refers to individuals, it can refer to replications, points in time, points of view, experimental conditions, and so on. The essential component is that we have m sets of dissimilarities between the same n objects. In order not to prejudge where the m different sets of dissimilarities come from, we shall refer to them with the neutral term *slices*, just as the dissimilarities are defined on pairs of neutral *objects*. But to honor the tradition of differential psychology we will continue to use the chapter title "individual differences".

16.1 Stress Majorization

The CS majorizaton in this case is the same as our treatment in chapter 15. For all $(X_1, \cdots, X_m)$ and $(Y_1, \cdots, Y_m)$ we ahve

$$\sigma_k(X_k) \leq \frac{1}{2}\sum_{k=1}^m \eta_k^2(X_k - \Gamma(Y_k)) - \frac{1}{2}\sum_{k=1}^m \eta_k^2(\Gamma(Y_k)), \tag{16.2}$$

and thus the smacof approach tells us to minimize, or at least decrease, the first term on the right of (16.2) over the X_k.

As in CMDS, within an iteration we have to (approximately) solve a projection problem on some linear or nonlinear manifold, in the metric defined by the weights. Minimize over X, project X, conforming with chapter 15.

16.2 Types of Constraints

16.2.1 Unconstrained

If there are no constraints on the configurations of stress the minimization over $X_1, \cdots, X_m$ simply means solving m separate MDS problems, one for each k. Thus it does not bring anything new.

16.2.2 Replications

The first constraint that comes to mind is $X_k = X$ for all $k = 1, \cdots, m$. Thus the configuration is the same for all slices, and there really are no individual differences in this case. The computational aspects are discussed in sufficient detail in section 5.4.7.

16.2.3 INDSCAL/PARAFAC

$$x_{ijk} = \sum_{s=1}^S a_{is} b_{js} \lambda_{ks}$$

$$X_k = A\Lambda_k B'$$

16.2.4 IDIOSCAL/TUCKALS

$$x_{ijk} = \sum_{s=1}^{S} \sum_{t=1}^{T} \sum_{u=1}^{U} a_{is} b_{jt} c_{ku} d_{stu}$$

$$X_k = A V_k B'$$

$$V_k = \sum_{u=1}^{U} c_{ku} D_u$$

16.2.5 PARAFAC2

$$x_{ijk} = \sum_{s=1}^{S} a_{is} b_{kjs} \lambda_{ks}$$

$$X_k = K \Lambda_k L_k'$$

16.2.6 Factor Models

Q and R techniques

$$X_k = \begin{bmatrix} X & | & Y_k \end{bmatrix}$$

$$X_k = \begin{bmatrix} K & | & L \end{bmatrix} \begin{bmatrix} A_k \\ -- \\ Y_k \end{bmatrix},$$

$$X_k' = \begin{bmatrix} K & | & L \end{bmatrix} \begin{bmatrix} A_k \\ -- \\ Y_k \end{bmatrix}$$

16.3 Nonmetric Individual Differences

16.3.1 Conditionality

Chapter 17

Asymmetry in MDS

17.1 Conditional Rankings

Two sets (= unfolding), one set (much tighter) solution Young_75

17.2 Confusion Matrices

Choice theory

17.3 The Slide Vector

17.4 DEDICOM

17.5 Constantine-Gower

Chapter 18

Nominal MDS

So far we have assumed there was some direct information about dissimilarities between objects. The information could be either numerical or ordinal, but we have not talked about nominal or categorical information yet. By nominal information we mean that the n objects are partitioned in K categories, where we usually assume that K is much smaller than n. Categories are not necessarily ordered.

Where does MDS come in ? Our starting point is that objects in the same category are more similar than objects in different categories. But this requirement can be formalized in different ways.

We discuss some of the ways in which we can express the nominal information in terms of distances and apply techniques in the smacof family to computed optimal least squares configurations.

For historical and other reasons this topic is of great interest to me. Some of my first rambling red reports (De Leeuw (1968a), De Leeuw (1969)) were on the analysis of relational or categorical data. The first one discussed mainly the quantification methods of Guttman (1941), the second one explored using the topological notion of separation. My dissertation (De Leeuw (1973a)) systematized that early research. Making it available to a wider audience, and as software, was the main motivation for starting the Gifi project (Gifi (1990), (**deleeuw_B_20?**)).

18.1 Binary Dissimilarities

Suppose we have and equivalence relation $\simeq$ on n objects $\mathcal{O}$. Let $d_{ij} = 0$ if $i \simeq j$ and $d_{ij} = 1$ if $i \not\simeq j$. Suppose the quotient set $\mathcal{O}/\simeq$ has K equivalence classes. Thus the objects can be ordered in such a way that Δ has K zero matrices with size $n_1 \cdots, n_K$ in the diagonal blocks and ones everywhere else. The n_k are the sizes of the equivalence classes, and they add up on n.

$[x]$ canonical projection

18.2 Indicator matrix

Q technique

```
##     1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
## 1   0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1
## 2   0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1
## 3   0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1
## 4   0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1
## 5   0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1
## 6   0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1
## 7   0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1
## 8   0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1
## 9   0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1
## 10  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1  1
## 11  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  1  1  1  1  1
## 12  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  1  1  1  1  1
## 13  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  1  1  1  1  1
## 14  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  1  1  1  1  1
## 15  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0  1  1  1  1  1
## 16  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0
## 17  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0
## 18  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0
## 19  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0
## 20  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0
```

Table 18.1: Example Indicator Matrix

	A	B	C
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	1	0	0
6	1	0	0
7	1	0	0
8	1	0	0
9	1	0	0
10	1	0	0
11	0	1	0
12	0	1	0
13	0	1	0
14	0	1	0
15	0	1	0
16	0	0	1
17	0	0	1
18	0	0	1
19	0	0	1
20	0	0	1

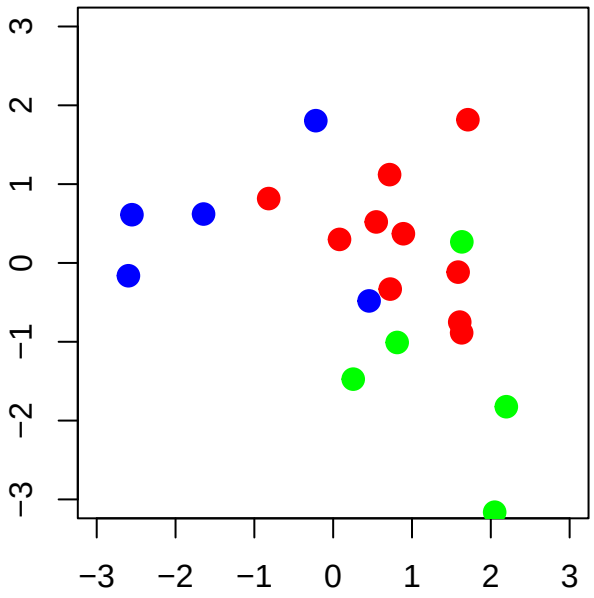


Figure 18.1: Example Object Scores

18.3 Unfolding Indicator Matrices

all within category distances smaller than all between category distances (for each variable separately) as in MCA: joint persons and categories, primary approach to ties

These order constraints are rather strict. They do not only imply that the category clouds are separated, they also mean the clouds must be small and rather far apart.

Think of the situation where the two categories are balls in $\mathbb{R}^p$ with centers x and y and \$radius r. The largest within-category distance is $2r$. The smallest between-category distance is $\max(0, d(x, y) - 2r)$. Thus all within-category distances are all smaller than all between-category distances if and only if $d(x, y) \geq 4r$.

We can make the requirements less strict by

For all k

$$\max_{i \in I_k} \max_{j \in I_k} d(x_i, x_j) \leq \min_{i \in I_k} \min_{j \in I \setminus I_k} d(x_i, x_j).$$

$$\max(kk) \leq \min(k\overline{k})$$

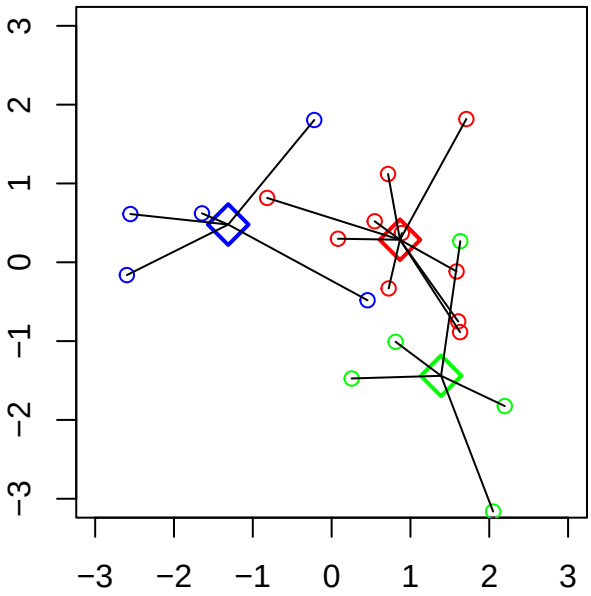


Figure 18.2: Distance Based MCA

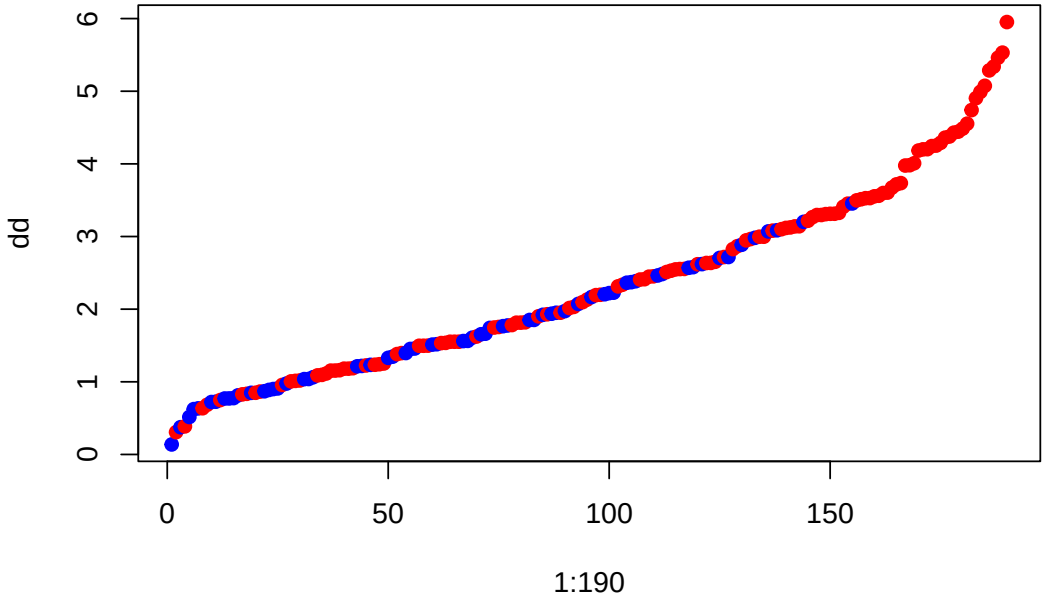


Figure 18.3: Within and Between Distances

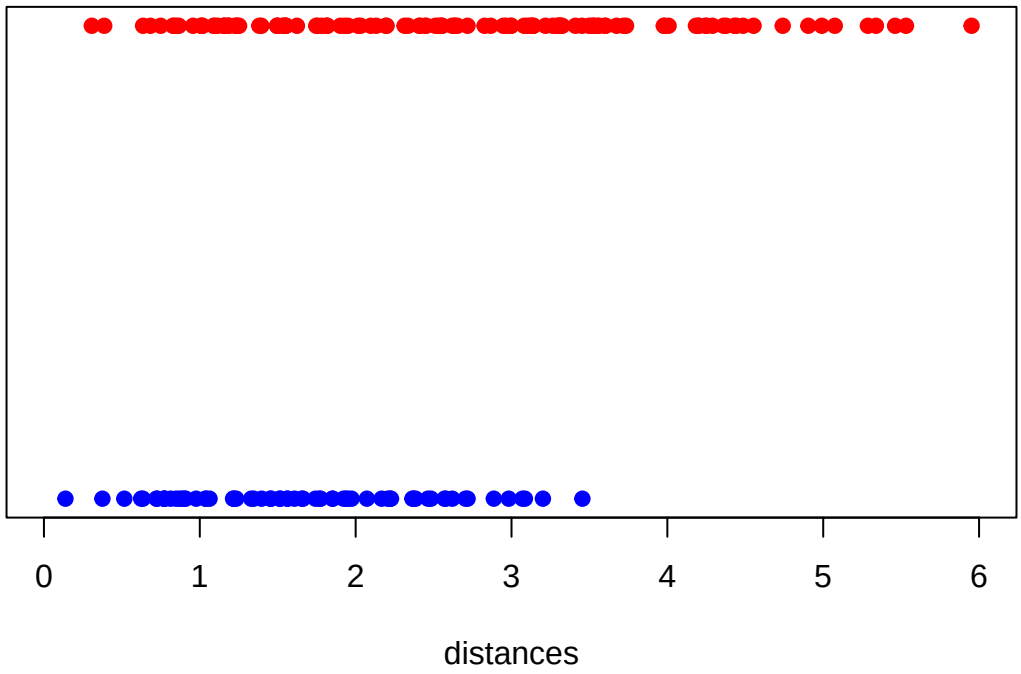


Figure 18.4: Within and Between Distances

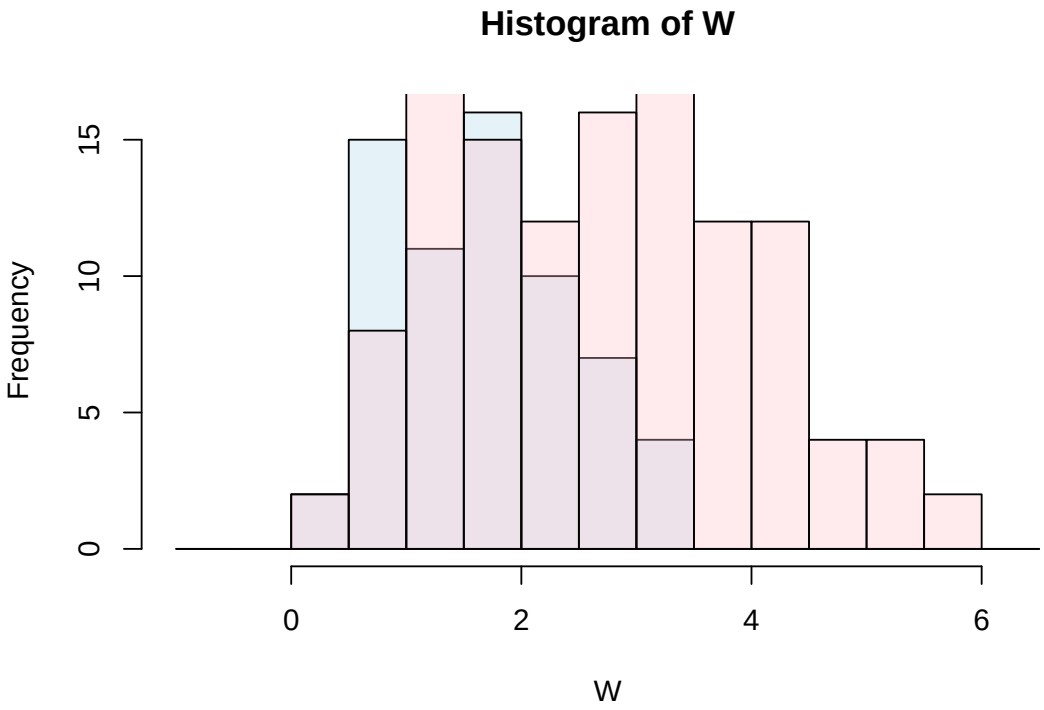


Figure 18.5: Within and Between Distances

$$\max_k \max(kk) \leq \min_{i \neq j} \min(ij)$$

the within category distances of category 1 are less than the smallest between-category distance

18.4 Linear Separation

line perpendicular to line connecting category points separates categories

all closer to their star center than to other star centers: primary monotone regression over all rows of g

suppose the star centers are y and z. The plane is $(x - \frac{1}{2}(y+z))'(y-z) = 0$ If u is in the y category we must have $(u - \frac{1}{2}(y+z))'(y-z) \geq 0$

Just in terms of distances

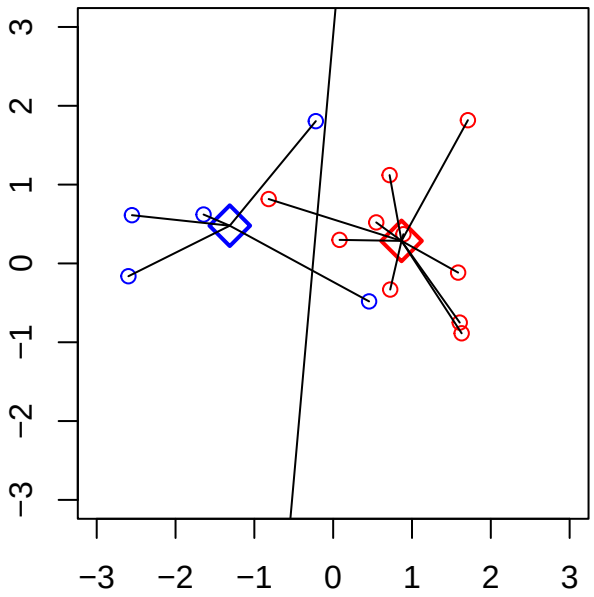


Figure 18.6: Distance Based MCA

What if the star centers are on a straight line

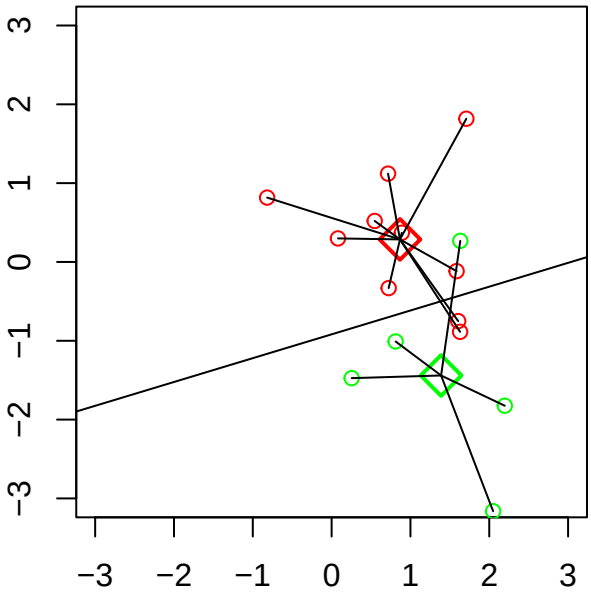


Figure 18.7: Distance Based MCA

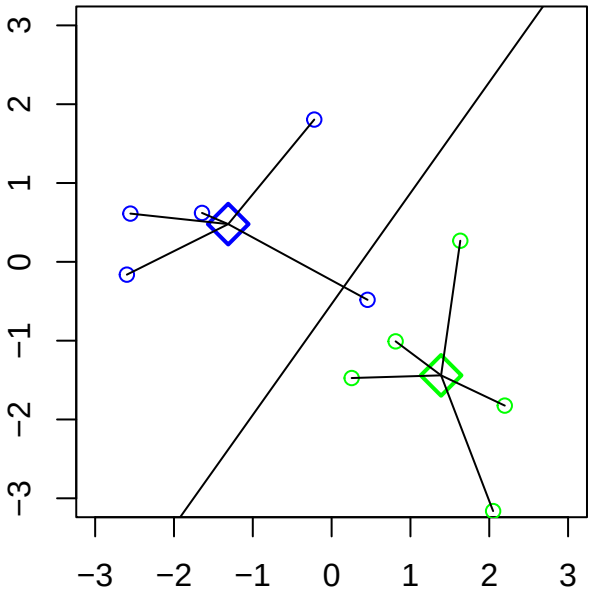


Figure 18.8: Distance Based MCA

18.5 Circular Separation

monotone regression on each column of g

k within balls must be disjoint => they can be separated by straight lines

18.6 Convex Hull Scaling

18.7 Voronoi Scaling

18.8 Multidimensional Scalogram Analysis

Guttman's MSA: Inner points, outer points

Suitably mysterious

Lingoes (1968b) Lingoes (1968a) Guttman (1967)

Chapter 19

Nonmonotonic MDS

19.1 Filler

This chapter will discuss techniques in which the relation between dissimilarities and distances is not necessarily monotone (in as far as these techniques fit into the smacof framework). I have in mind the mapping of high-dimensional manifolds into low-dimensional Euclidean ones, in the spirit of Shepard and Carroll (1966). The prime examples are still cutting and unrolling the circle, the sphere, or the torus.

The dissimilarities define the high-dimensional space, the distances the low-dimensional space. The relation between distances and dissimilarities may not be functional, i.e. we can have $F(D(X), \Delta) = 0$ or Δ could be a function of $D(X)$.

In general, small distances in low-dimensional space are small distances in high-dimensional space, but large distances in low-dimensional space can be small distances in high-dimensional space. This suggests an inverse Shepard plot, with dissimilarity as a function of distance. Or

$$\sigma(X) = \sum\sum(delta_{ij} - f(d_{ij}(X)))^2$$

Euclidean

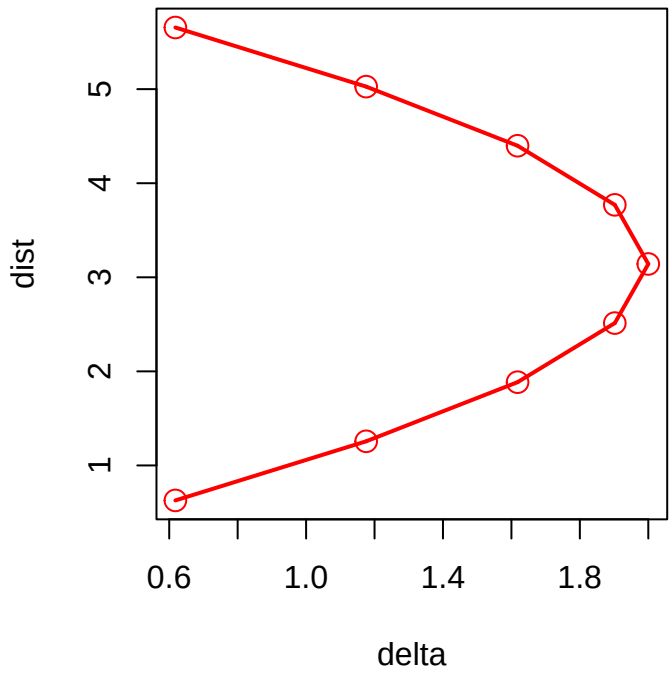
$$\sqrt{2 - 2\ \cos|i - j|\theta}$$

Circular

$$|i-j|\frac{2\pi}{n}$$

Linear

$$|i-j|$$



Quadratic

$F(\Delta, D(X)) = 0$

$$a_1\delta_{ij}^2 + a_2\delta_{ij}d_{ij}(X) + a_3d_{ij}^2(X) + a_4\delta_{ij} + a_5d_{ij}(X) + a_6 = 0$$

$$pd_{ij}^2(X) + 2qd_{ij}(X) + r = p(d_{ij}(X) - q/p)^2 + r - (q/p)^2$$

$f(\Delta) = g(D(X))$

Chapter 20

Compound Objects

20.1 Filler

A compound object is a set of $m > 1$ objects. This chapter treats the analysis of dissimilarities between compound objects, again as far as they fit into the smacof framework. Thus we do not, for example, look at the pairwise dissimilarities within a triad of objects, but at the dissimilarities of the triads themselves.

Chapter 21

Sstress and strain

21.1 sstress

Takane, Young, and De Leeuw (1977) gave σ_2 the name *sstress*. Thus sstress is defined as

$$\sigma_2(X) := \sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij}^2 - d_{ij}^2(X))^2$$

On the space of configurations sstress is mathematically much better behaved as stress. It is a non-negative multivariate polynomial of degree four, actually a sum-of-squares or SOS polynomial (ref).

It is everywhere infinitely many times differentiable everywhere and, in principle at least, we can compute all real-valued configurations where the derivatives of sstress vanish by algebraic methods. That includes all local minima, and thus also the global minimum. Unfortunately in almost all MDS applications the number of variables is too large to apply the usual algebraic methods.

nonnegative polynomials of degree four – general algebra

21.1.1 sstress and stress

Clearly configurations with small stress will tend to have small sstress, and vice versa.

weighting residuals – fitting large dissimilarities

$$\sum d_{ij}^2(\delta_{ij}^2 - d_{ij}^2) = 0$$

We can write sstress as

$$\sigma_2(X) = \sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij} + d_{ij}(X))^2(\delta_{ij} - d_{ij}(X))^2. \tag{21.1}$$

Thus if the $d_{ij}(X)$ provide a good fit to the δ_{ij} we have the approximation

$$\sigma_2(X) \approx 4\sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}^2(\delta_{ij} - d_{ij}(X))^2 \tag{21.2}$$

Since $\mathcal{D}f(\delta) = 2\delta$ in this case, (21.2) also follows from (22.3).

Now

$$\frac{\sigma_2(X)}{\sigma(X)} = \frac{\sum\sum_{1\leq i<j\leq n} w_{ij}(\delta_{ij} + d_{ij}(X))^2(\delta_{ij} - d_{ij}(X))^2}{\sum\sum_{1\leq i<j\leq n} w_{ij}(\delta_{ij} - d_{ij}(X))^2}. \tag{21.3}$$

This is a weighted average of the quantities $(\delta_{ij} + d_{ij}(X))^2$, and thus

$$4\{\min(\delta_{ij} + d_{ij}(X))\}^2\sigma(X) \leq \sigma_2(X) \leq 4\{\max(\delta_{ij} + d_{ij}(X))\}^2\sigma(X). \tag{21.4}$$

21.1.2 Decomposition

$$\sigma_2(X) = 1 - 2\rho_2(X) + \eta_2^2(X)$$
$$\sigma_2(\alpha X) = 1 - 2\alpha^2\rho_2(X) + \alpha^4\eta_2^2(X)$$

At a minimum $\rho_2(X) = \eta_2^2(X)$ and thus $\sigma(X) = 1 - \eta_2^2(X)$, which implies $\eta_2(X) \leq 1$.

Thus minimizing σ_2 means maximizing ρ_2 over $\eta_2(X) \leq 1$, which is the same thing as minimizing η_2 over $\rho_2(X) \geq 1$. Both are reverse convex problems.

$$\eta_2(X) = \sqrt{\sum\sum w_{ij} d_{ij}^4(X)} \geq \frac{1}{\sqrt{\sum\sum w_{ij} d_{ij}^4(Y)}} \sum\sum w_{ij} d_{ij}^2(Y) d_{ij}^2(X)$$

Thus maximizing ρ_2 with $\rho_2(X) = \text{tr } X'B_0X$ over tr $X'B_2(Y)X \leq 1$ for all Y.

At a local minimum partitioning

21.1.3 Full-dimensional sstress

Theorem: The set of squared Euclidean distance matrices between n points $\mathfrak{D} := \{D \mid D = D^{(2)}(X)\}$ is a closed convex cone.

Proof. It suffices to observe that $\alpha D^{(2)}(X) = D^{(2)}(\sqrt{(}\alpha)X)$ and $D^{(2)}(X) + D^{(2)}(Y) = D^{(2)}(X \mid Y)$. Alternatively, $\mathfrak{D}$ is the intersection of two convex cones, and thus convex. The first cone are the hollow non-negative symmetric matrices and the second cone are the symmetric matrices D for which $-JDJ \gtrsim 0$. □

Corollary:

$$\min_{D \in \mathfrak{D}} \sum_{1 \leq i < j \leq n} \sum w_{ij} (\delta_{ij}^2 - d_{ij})^2$$

is a convex problem with a unique minimum.

Polar cone

$$\sum_{1 \leq i < j \leq n} \sum w_{ij} e_{ij} \text{tr } A_{ij} C \leq 0$$

for all $C \gtrsim 0$. Thus we must have $\sum \sum_{1 \leq i < j \leq n} w_{ij} e_{ij} A_{ij} \lesssim 0$

$$D \in \mathfrak{D}, \Delta - D \in \mathfrak{D}^o \text{tr } W \times D(\Delta - D) = 0$$

What is the polar cone $\mathfrak{D}^o$

Since $d_{ij}^2 = c_{ii} + c_{jj} - 2cij$ it follows that σ_2 is a convex quadratic in C and that minimizing σ_2 over $C \gtrsim 0$ is a convex problem, just as minimizing σ_1 over $C \gtrsim 0$ is.

But difference –

$$\sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij}^2 - \text{tr } A_{ij}C)^2$$

$$\mathcal{D}\sigma_2(C) = -2\sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij}^2 - \text{tr } A_{ij}C)A_{ij}$$

$$B_2(C) := \sum_{1\leq i<j\leq n}\sum w_{ij}d_{ij}^2(C)A_{ij}$$

Weights !! Polar cone !

$$\begin{aligned} C &\gtrsim 0, \\ V_2 - B_2(C) &\gtrsim 0, \\ \text{tr } C(V_2 - B_2(C)) &= 0. \end{aligned} \tag{21.5}$$

gower2 rank

21.1.4 Minimizing sstress

21.1.4.1 ALSCAL

The first published paper on sstress minimization, with detailed algorithm and computer program, was Takane, Young, and De Leeuw (1977). There are some historical precursors, but they are mostly in internal memos, and they usually did not come with software. The ALSCAL program (F. W. Young, Takane, and Lewyckyj (1978b)) was widely distributed through SPSS and SAS and is still used regularly in various areas of research.

In this section of the book we will discuss basic ALSCAL, i.e. the sstress version of basic MDS scaling. As usual, we generalize to weighted least squares, but for now we ignore the individual differences and the non-metric parts.

It is also worth noting that in the original version of ALSCAL, from Takane, Young, and De Leeuw (1977), the configuration is fitted by an alternating least squares algorithm that changes all p coordinates of each point simultaneously, and then cycles through the points. The minimization over coordinates is done with a safeguarded Newton-Raphson method. At the time I forcefully objected to this. Sstress is a p-dimensional quartic, and because p is usually small fiding the global minimum with algebraic methods is at least conceivable. But in terms of simplicity, it is much better to change a single coordinate at the time, meaning that one cycle consists of finding the global minimum of np univariate quadrics. There is some acknowledgement of this in section 5 of Takane, Young, and De Leeuw (1977), but the paper is quite verbose and it can easily be overlooked. My understanding, based on F. W. Young, Takane, and Lewyckyj (1978a), is that later versions of ALSCAL did indeed adopt one-dimensional cyclic coordinate descent (CCD). And this is what we will discuss here.

First note that sstress can be decomposed in the same way as stress. We have

$$\sigma_2(X) = 1 - 2\rho_2(X) + \eta_2^2(X) \tag{21.6}$$

with

$$\begin{aligned}\rho_2(X) &:= \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}^2 d_{ij}^2(X),\\ \eta_2^2(X) &:= \sum_{1\leq i<j\leq n}\sum w_{ij} d_{ij}^4(X).\end{aligned} \tag{21.7}$$

Both ρ_2 and η_2^2 are convex, ρ_2 is quadratic and η_2^2 is a quartic (in fact, a sum of squares of quadratics).

In CCD we replace x_{ks} by $\tilde{x}_{ks} := x_{ks} + \epsilon$. Or, in matrices, $\tilde{X} := X + \epsilon\ e_k e_s'$. Only the $d_{ij}^2(\tilde{X})$ with $i = k$ or $j = k$ differ from the corresponding $d_{ij}(X)$. All these $d_{ik}^2(\tilde{X})$ are now just a function of ϵ and thus we write

$$d_{ik}^2(\epsilon) := d_{ik}^2(\tilde{X}) = d_{ik}^2(X) - 2\epsilon u_i + \epsilon^2. \tag{21.8}$$

where $u_i := x_{is} - x_{ks}$. Also

$$d_{ik}^4(\epsilon) = d_{ij}^4(X) - 4\epsilon u_i d_{ik}^2(X) + 2\epsilon^2(d_{ik}^2(X) + 2u_i^2) - 4\epsilon^3 u_i + \epsilon^4. \tag{21.9}$$

Combining (21.8) and (21.9) with (21.6) gives

$$\rho_2(\epsilon) = \rho_2(0) - 2\epsilon \sum_{i=1}^{n} w_{ik}\delta_{ik}^2 u_i + \epsilon^2 \sum_{i=1}^{n} w_{ik}\delta_{ik}^2 \tag{21.10}$$

and

$$\eta_2^2(\epsilon) = \eta_2^2(0) - 4\epsilon \sum_{i=1}^{n} w_{ik} d_{ik}^2(X) u_i + 2\epsilon^2 \sum_{i=1}^{n} w_{ik}(d_{ik}^2(X) + 2u_i^2) - 4\epsilon^3 \sum_{i=1}^{n} w_{ik} u_i + \epsilon^4 \sum_{i=1}^{n} w_{ik} \tag{21.11}$$

and finally

$$\sigma_2(\epsilon) = \sigma_2(0) - 4\epsilon \sum_{i=1}^{n} w_{ik}(d_{ik}^2(X) - \delta_{ik}^2) u_i + 2\epsilon^2 \sum_{i=1}^{n} w_{ik}((d_{ik}^2(X) - \delta_{ik}^2) + 2u_i^2) - 4\epsilon^3 \sum_{i=1}^{n} w_{ik} u_i + \epsilon^4 \sum_{i=1}^{n} w_{ik} \tag{21.12}$$

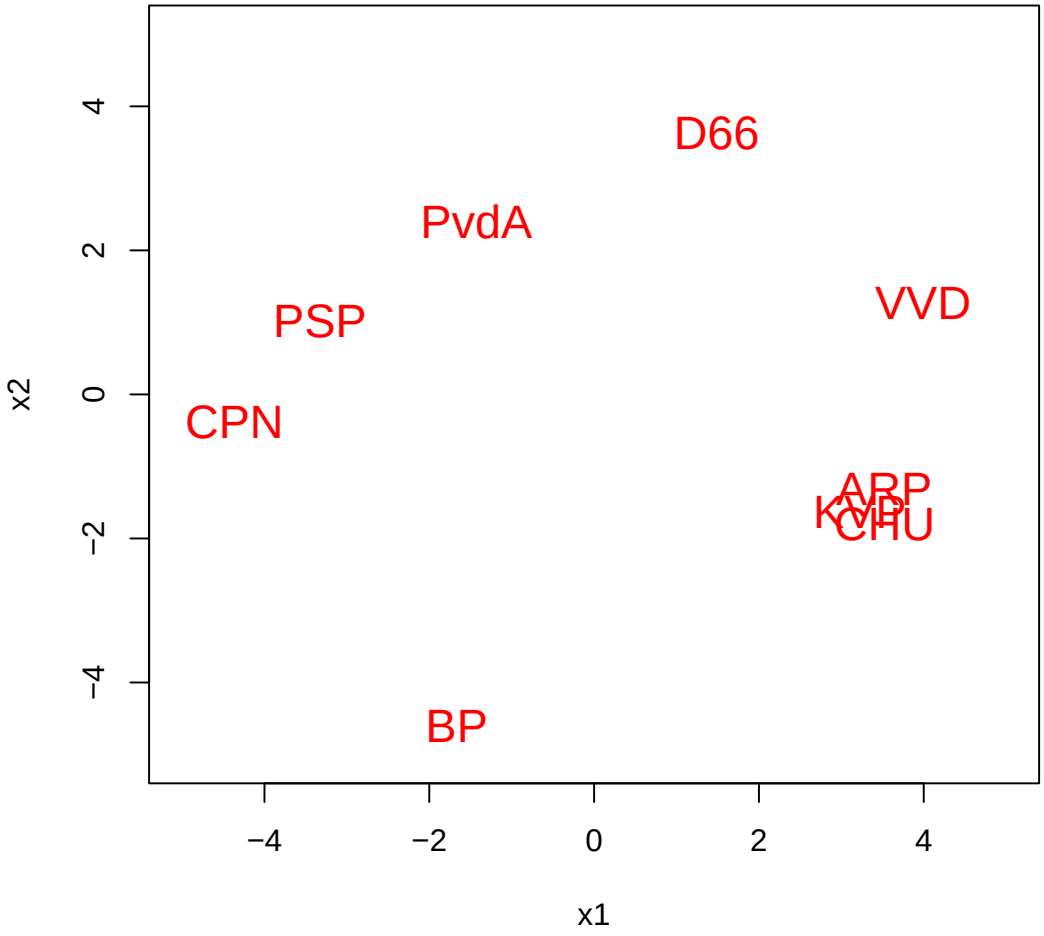
Convex, DC, derivative

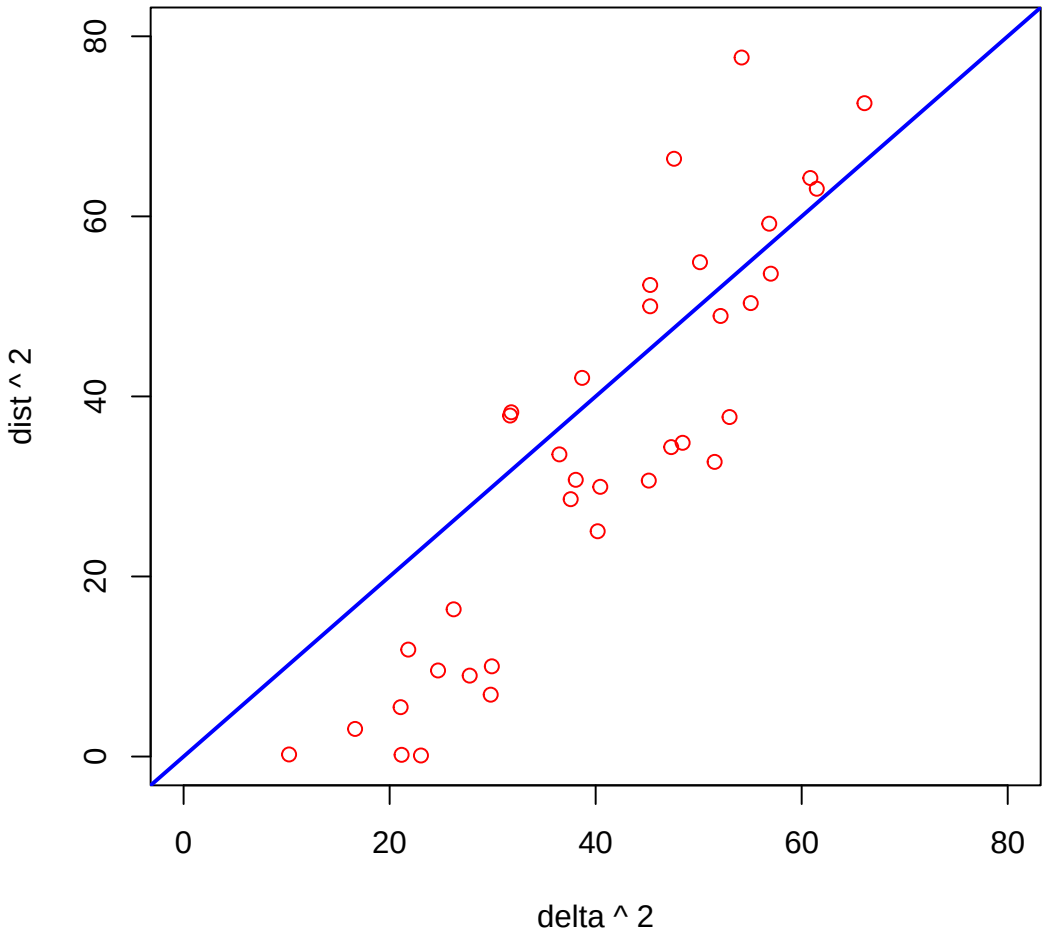
Jeffrey

Code

Example

Query: is σ_2 a convex quartic in ϵ ? Does $\mathcal{D}\sigma_2(\epsilon) = 0$ always have a single root.


D66
PvdA
VVD
PSP
CPN
ARP
KVP
CHU
BP
x1
x2
−4
−2
0
2
4
−4
−2
0
2
4



1.7957662×10^4 after 55 iterations

Try global minimum over points (p-dimensonal quartic)

21.1.4.2 Majorization

$$\sum\sum w_{ij}\{\text{tr}A_{ij}C\}^2 \leq \lambda \text{ tr } C^2.$$

Better maybe (check this)

$$\sum\sum w_{ij}\{\text{tr } A_{ij}C\}^2 \leq \lambda \sum\sum w_{ij}c_{ij}^2.$$

$$\sigma(C)=\sigma(\tilde{C}+(C-\tilde{C}))=1-2\sum\sum w_{ij}\delta_{ij}\text{tr}\,A_{ij}(\tilde{C}+(C-\tilde{C}))+\sum\sum w_{ij}\{\text{tr}\,A_{ij}(\tilde{C}+(C-\tilde{C})\}^2=1-2\sum$$

$$B(C):=\nabla\sigma_2(C)=\sum\sum w_{ij}(d_{ij}^2(C)-\delta_{ij}^2)A_{ij}$$

Necessary condition: C is the projection of $\overline{C}:=C-\lambda^{-1}B(C)$ on the cone of psd matrices. Thus

$$C\gtrsim 0B(C)\gtrsim 0\text{tr}\;CB(C)=0$$

De Leeuw (1975b)

De Leeuw, Groenen, and Pietersz (2016)

takane

brown

Augmentation

Functions of Squared Distances

$$f(X)=F(D^2(X))$$
$$\mathcal{D}f(X)=2\sum\sum\mathcal{D}_{ij}F(D^2(X))A_{ij}X$$
$$f(C)=F(D^2(C))$$
$$\mathcal{D}f(C)=\sum\sum\mathcal{D}_{ij}F(D^2(C))A_{ij}$$

$$\mathcal{D}^2f(C)=\sum\sum\mathcal{D}_{ij,kl}F(D^2(C))A_{ij}$$

21.1.4.3 SOS

Normal Fourth degree tensor

$$\sum\sum w_{ij}(\delta_{ij}-x'A_{ij}x)^2$$
$$\sum_k\sum_l\left\{\sum\sum w_{ij}\delta_{ij}A_{ij}\right\}x_kx_l$$

$$\sum_k\sum_l\sum_p\sum_q\left\{\sum\sum w_{ij}\delta_{ij}\{A_{ij}\otimes A_{ij}\}\right\}_{klpq}x_kx_lx_px_q$$

21.1.4.4 Duality

!! David Gao

21.1.4.5 ALS Unfolding

In the heady days around 1968, when the Department of Data Theory was founded in Leiden, the focus was very much on unfolding. Coombs had just visited and the non-metric revolution was starting up. Alternating least squares was in the air. I was supposed to work on a program for metric unfolding, starting from the ideas of Ross and Cliff (1964) and the machinery provoded by Torgerson's classical scaling.

The idea, mainly due to John van de Geer, was to complete the $n \times m$ matrix of off-diagonal dissimilarities to a symmetric matrix of order $n + m$, starting with initial estimates of the distances in the two diagonal blocks. Then apply Torgerson, and use the results to improve the estimates of the distances in the diagonal blocks, then use Torgerson again, and so on. Alternating least squares with imputation of the diagonal blocks. Multidimensional scaling of $n + m$ objects, with zero weights for the diagonal blocks. But it only worked to a certtain point. After decreasing stress for a while and approaching convergence the loss started to increase. We were deflated and gave up the approach, without really being able to understand about why it did not work (De Leeuw (1968c)).

In hindsight, it is clear what was wrong. We imputed the diagonal blocks minimizing stress, and then adjusted the configuration using strain. Two different loss functions, which obviously violated the basic idea of alternating least squares and the guaranteed convergence of either of the two loss functions.

What is worth preserving from this approach is the initial estimate for the diagonal blocks, again due to John van de Geer. It cleverly uses the two triangle inequalities, assuming the dissimilarities are really distances. For $1 \leq i < j \leq n$

$$\delta_{ij} = \frac{1}{2}\left\{\min_{k=1}^{m}(\delta_{ik} + \delta_{jk}) + \max_{k=1}^{m}|\delta_{ik} - \delta_{jk}|\right\},$$

and for $n+1 \leq k < l \leq n+m$

$$\delta_{kl} = \frac{1}{2}\left\{\min_{v=1}^{n}(\delta_{vk}+\delta_{vl}) + \max_{k=1}^{m}|\delta_{vk}-\delta_{vl}|\right\}.$$

Greenacre and Browne (1986)

augmentation

21.1.5 Bounds for sstress

De Leeuw and Bettonvil (1986)

21.2 strain

Classical scaling as formulated by W. S. Torgerson (1958) computes the dominant non-negative eigenvalues, with corresponding eigenvectors, of the Torgerson transform of the squared dissimilarities. This is usually presented as an "ignore-errors" technique. It is clear what it does in the case of perfect fit of distances to dissimilarities, it is not so obvious how it measures approximation errors in the case of imperfect fit. Or, to put it differently, MDS lore has it that in classical scaling loss is defined on the scalar products, which are a transformation of the dissimilarity data, and not on the dissimilarities themselves. This is presented as somehow being a disadvantage (see, for example, Takane, Young, and De Leeuw (1977), Browne (1987)).

If we agree to use weights, then *strain* is defined straightforwardly as

$$\sigma_\tau(X) := \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(\tau_{ij}(\Delta^{(2)}) - x_i'x_j)^2. \tag{21.13}$$

Note that this summation includes the diagonal elements, so in general we cannot expect W to be hollow. In fact, @(eq:straindef) adds diagonal elements only once, while the off-diagonal elements are added twice. This observation is the basis of the excellent paper by Bailey and Gower (1990).

Because of the weights, minimizing strain in this form does not lead to an eigenvalue problem, unless there is a non-negative vector u such that $w_{ij} = u_i u_j$. In that case

$$\sigma_\tau(X) := \text{tr } (U\tau(\Delta^{(2)})U - UXX'U)^2, \tag{21.14}$$

where $U = \text{diag}(u)$, and we can find UX, and thus X, by eigen decomposition of $U\tau_{ij}(\Delta^{(2)})U$.

We can use our general results on unweighting, as in section 5.4.10, to get rid of the weights, but this leads to a sequence of eigenvalue problems. Nevertheless, if the weights are important, this is an option.

21.2.1 Unweighted

For column-centered configurations

$$J(\Delta^{(2)} - D^{(2)}(X))J = -2(\tau(\Delta^{(2)}) - XX')$$

Thus if all weights are equal to one then

$$4\sigma_\tau(X) = \text{tr } J(\Delta^{(2)} - D^{(2)}(X))J(\Delta^{(2)} - D^{(2)}(X)),$$

which shows that strain is a matrix-weighted version of sstress. It also shows (De Leeuw and Heiser (1982), theorem 21) that $\sigma_\tau(X) \leq \frac{1}{4}\sigma_2(X)$.

21.2.2 Bailey-Gower

$$\min_{C \gtrsim 0} \sigma(C) := \sum_{i=1}^n \sum_{j=1}^n w_{ij}(s_{ij} - c_{ij})^2$$

If E is psd then

$$\sum_{i=1}^n \sum_{j=1}^n w_{ij}((s_{ij}-c_{ij})-\alpha e_{ij})^2 = \sigma(C)-2\alpha \sum_{i=1}^n \sum_{j=1}^n w_{ij}(s_{ij}-c_{ij})e_{ij}+\alpha^2 \sum_{i=1}^n \sum_{j=1}^n w_{ij}e_{ij}^2 \geq \sigma(C)$$

$$\sum_{i=1}^n \sum_{j=1}^n w_{ij}(c_{ij} - s_{ij})e_{ij} \geq 0, \tag{21.15}$$

$$\sum_{i=1}^n \sum_{j=1}^n w_{ij}(c_{ij} - s_{ij})c_{ij} = 0. \tag{21.16}$$

Using Additivity

$$\sigma_\tau(X) = \min_\alpha \sum\sum_{1\leq i<j\leq n} w_{ij}(\delta_{ij}^2 - (\alpha_i + \alpha_j) - 2x_i'x_j))^2$$

$$\sigma_2(X) = \min_{\alpha\geq 0} \min_{\text{diag}XX'=I} \sum\sum_{1\leq i<j\leq n} w_{ij}(\delta_{ij}^2 - (\alpha_i^2 + \alpha_j^2 - 2\alpha_i\alpha_j\ x_i'x_j))^2$$

projection

If there are no weights, or if we unweight the weighted loss function

Chapter 22

fstress and rstress

22.1 fstress

Fstress is a straightforward generalization of stress. Suppose f is any non-decreasing real-valued function, and define

$$\sigma_f(X) := \sum_{1 \leq i < j \leq n} \sum w_{ij}(f(\delta_{ij}) - f(d_{ij}(X)))^2 \tag{22.1}$$

We discuss various specific examples in this chapter, such as the square and the logarithm, but let's first mention some general results.

22.1.1 Use of Weights

Suppose the $d_{ij}(X)$ are close to the δ_{ij}, so that we have a good fit and a low stress. Then the approximation

$$f(d_{ij}(X)) \approx f(\delta_{ij}) + \mathcal{D}f(\delta_{ij})(d_{ij}(X) - \delta_{ij}) \tag{22.2}$$

will be close. Thus

$$\sigma_f(X) \approx \sum_{1 \leq i < j \leq n} \sum w_{ij}(\mathcal{D}f(\delta_{ij}))^2(\delta_{ij} - d_{ij}(X))^2. \tag{22.3}$$

Thus we can approximately minimize fstress by minimizing stress with weights $w_{ij}(\mathcal{D}f(\delta_{ij}))^2$. If the fit is good, we can expect to be close. If the fit is perfect, the approximation is perfect too. Note that we do not assume that f is increasing, i.e. that $f' \geq 0$.

22.1.2 Convexity

$$f(g(\lambda x + (1-\lambda)y)) \leq f(\lambda g(x) + (1-\lambda)g(y)) \leq \lambda f(g(x)) + (1-\lambda)f(g(y))$$

Thus if g is convex (for instance distance) and f is convex and increasing then $f \circ g$ is convex (and thus stress is DC). Unfortunately a concave f is more interesting.

22.2 rStress

$$\sigma_r(X) := \sum_{1\leq j<i\leq n}\sum w_{ij}(\delta_{ij}^r - d_{ij}^r(X))^2. \tag{22.4}$$

In definition (22.4) we approximate the r-th power of the dissimilarities by the r-th power of the distances. Alternatively, we could have defined

$$\sigma_r(X) := \sum_{1\leq j<i\leq n}\sum w_{ij}(\delta_{ij} - d_{ij}^r(X))^2 \tag{22.5}$$

In definition (22.4) we are still approximating the dissimilarities by the distances, as in basic MDS, but we are defining errors of approximation as the differences between the r-th powers. In definition (22.5)

I am not sure which of the two formulations is the more natural one. I am sure, however, that for basic MDS the two formulations are effectively the same, because we can just define dissimilarities in (22.5) as the r-th power of the ones in (22.4). And in ordinal MDS the two formulations are the same as well, because the rank orders of the unpowered and powered dissimilarities are the same.

22.2.1 Using Weights

$$d_{ij}^r(X) - \delta_{ij}^r = (d_{ij}^r(X) + \delta_{ij}^r)(d_{ij}^r(X) - \delta_{ij}^r)$$

If $\delta_{ij} \approx d_{ij}(X)$ then

If r is a power of 2 ==>

22.2.2 Minimizing rstress

rstress, qstress, power stress

Groenen and De Leeuw (2010) De Leeuw (2014b) De Leeuw, Groenen, and Mair (2016c) De Leeuw, Groenen, and Mair (2016e) De Leeuw, Groenen, and Mair (2016b)

22.3 mstress

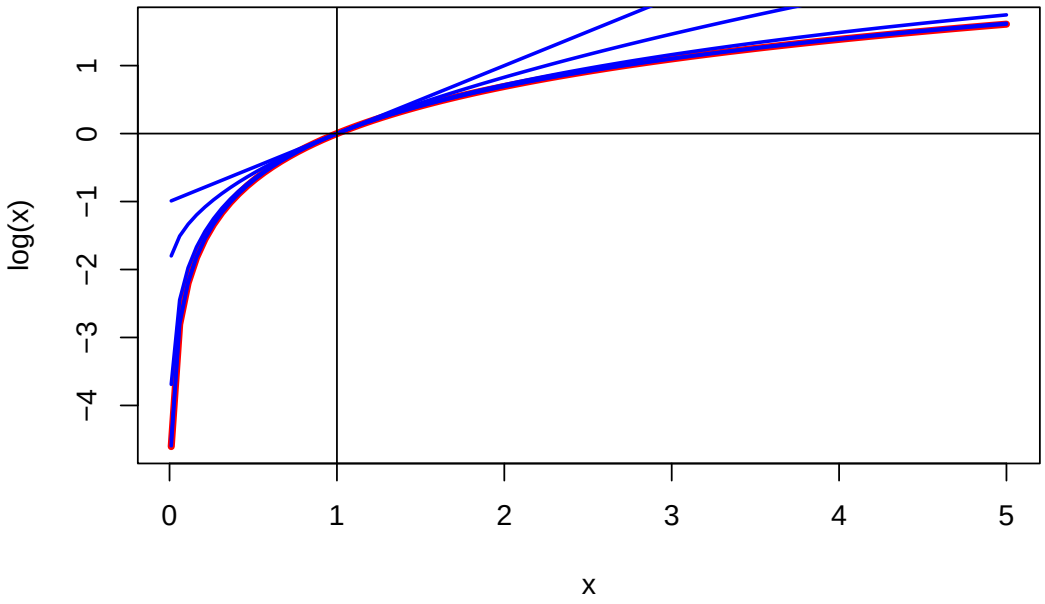
The loss function used by by Ramsay (1977) in his MULTISCAL program for MDS can be written as

$$\sigma_0(X) := \sum_{1\leq i<j\leq n}\sum w_{ij}(\log\delta_{ij} - \log(d_{ij}(X)))^2. \tag{22.6}$$

To justify the notation σ_0 we define f_r, for all $x > 0$ and $r < 1$, by $f_r(x) := r^{-1}\frac{x^r-1}{r}$. f_r is concave for all r, it majorizes the log because $f_r(x) \geq \log(x)$ for all x, with equality iff $x = 1$, and

$$\lim_{r\to 0}\frac{x^r-1}{r} = \log x. \tag{22.7}$$

We have drawn the logarithm, in red, and f_r for r equal to 0.001, 0.01, 0.1, 0.5, 1 over the interval $[.01, 5]$ in the figure that follows.



For $r = .001$ and $r = .01$ the logarithm and f_r are practically indistinguishable, and even for $r = .1$ we have an approximation which is probably good enough (in the given range) for most practical purposes.

Using the approximation of the logarithm with small r gives

$$\begin{aligned}\sigma_0(X) &\approx \sum_{1\leq i<j\leq n}\sum w_{ij}(\delta_{ij} - \frac{d_{ij}^r(X)-1}{r})^2\\ &= r^{-2}\sum_{1\leq i<j\leq w_{ij}}\sum((r\delta_{ij}+1) - d_{ij}^r(X))^2.\end{aligned} \tag{22.8}$$

If r is really small both $r\delta_{ij} + 1$ and $d_{ij}^r(X)$ will be very close to one, which will make minimization of the approximation difficult. A simple suggestion is to start with the SMACOF solution for $r = 1$, then use that solution for $r = 1$ as a starting point for $r = \frac{1}{2}$, and so on.

Alternative ?

$$\log d_{ij}(X) - \log \delta_{ij} \leq \frac{\{\frac{d_{ij}(X)}{\delta_{ij}}\}^r - 1}{r} = \frac{d_{ij}^r(X) - \delta_{ij}^r}{r\delta ij^r}$$

$$\log d_{ij}(X) - \log d_{ij}(Y) \leq \frac{\{\frac{d_{ij}(X)}{d_{ij}(Y)}\}^r - 1}{r} = \frac{d_{ij}^r(X) - d_{ij}^r(Y)}{r d_{ij}^r(Y)}$$

22.4 astress

robust MDS (zho@u_xu_li_19) LAR (Heiser (1988))

$$\sigma_{11}(X) = \sum_{1\leq i<j\leq n}\sum w_{ij}|\delta_{ij} - d_{ij}(X)|$$

$$|(\delta_{ij} - d_{ij}(X)) + \epsilon| \leq \frac{1}{2}\frac{((\delta_{ij} - d_{ij}(X)) + \epsilon)^2 + ((\delta_{ij} - d_{ij}(Y)) + \epsilon)^2}{|(\delta_{ij} - d_{ij}(Y)) + \epsilon|}$$

$$\sigma_{rs}(X) = \sum_{1\leq i<j\leq n}\sum w_{ij}|\delta_{ij}^r - d_{ij}^r(X)|^s$$

22.5 pstress

The p in pstress stands for panic. We define

$$\sigma(X) := \sum\sum_{(i<j)\leq(k<l)}\sum\sum w_{ijkl}(\delta_{ij} - d_{ij}(X))(\delta_{kl} - d_{kl}(X)),$$

where $(i < j) \leq (k < l)$ means that index pair (i, j) is lexicographically not larger than pair (k, l).

Covariances/variances

How many of these weights w_{ijkl} are there ?

$$\frac{1}{2}\binom{n}{2}(\binom{n}{2} + 1) = \frac{1}{8}n(n-1)(n^2 - n + 2)$$

No reason to panic. Again, majorization comes to the rescue (Groenen, Giaquinto, and Kiers (2003)). Suppose there is a $K > 0$ and a hollow, symmetric, non-negative Ω such that

$$\sum\sum_{(i<j)\leq(k<l)}\sum\sum w_{ijkl}z_{ij}z_{kl} \leq K\sum_{1\leq i<j\leq n}\sum \omega_{ij}z_{ij}^2.$$

Often the form of the weights w_{ijkl} will suggest how to choose K. In the worst case scenario we choose $\Omega = E - I$ and compute K by the power method. If all w_{ijkl} are equal to one, then ref becomes

$$\frac{1}{2}\left\{\sum\sum_{1\leq i<j\leq n} z_{ij}\right\}^2 + \sum\sum_{1\leq i<j\leq n} z_{ij}^2 \leq K \sum\sum_{1\leq i<j\leq n} \omega_{ij} z_{ij}^2.$$

$$\sigma(X) \leq \sigma(Y) + 2\sum \theta_{ij}(d_{ij}(Y) - d_{ij}(X)) + K\sum \omega_{ij}(d_{ij}(Y) - d_{ij}(X))^2$$

with (modify slightly !!)

$$\theta_{ij} = \sum\sum_{1\leq k<l\leq n} w_{ijkl}(\delta_{kl} - d_k lY)$$

$$\Phi(\Delta, D(X))$$

Chapter 23

Alternative Least Squares Loss

Kruskal/Carroll in Krishnaiah

23.1 Sammon's MDS

23.2 Kamade-Kawai Spring

23.3 McGee's Work

23.4 Shepard's Nonmetric MDS

23.5 Guttman's Nonmetric MDS

23.6 Positive Orthant Nonmetric MDS

!! Richard Johnson 1973

!! Guttman Absolute Value

!! Hartmann

23.7 Role Reversal

Kruskal, arithmetic with dissimilarities

$$\sigma(X) = \sum_{1 \leq i < j \leq n} \sum w_{ij}(\delta_{ij} - P_r(d_{ij}(X)))^2$$

Chapter 24

Inverse Multidimensional Scaling

In MDS we start with dissimilarities and we find a configuration that locally minimizes stress. We know that the equation we have to solve is $B(X)X = VX$. MDS maps dissimilarities into configurations by finding one (or, ideally, all) solutions of this equation. In Inverse MDS (IMDS) we start wth the same equation $B(X)X = VX$, but now we find all dissimilarity matrices for which a given configuration is a local optimum, or at least a stationary point. Thus we solve for Δ for given X, and we study the inverse of the MDS map.

Inverse MDS was first described in "Inverse Multidimensional Scaling" (2007), R code was provided in De Leeuw (2012), and some elaborations are in De Leeuw, Groenen, and Mair (2016d). This chapter leans heavily on De Leeuw, Groenen, and Mair (2016d), but we have reformulated some results and pruned some of the examples.

In studying the IMDS mapping we limit ourselves, unless explicitly stated otherwise, to configurations X that are *regular*, in the sense that $d_{ij}(X) > 0$ for all $i \neq j$. This can be done without loss of generality. If some of the distances are zero, then the corresponding IMDS problem can be reduced to a regular problem with a smaller number of points (De Leeuw, Groenen, and Mair (2016f)). Also, an $n \times p$ configuration X is *normalized* if it is column-centered and has rank p. For such X there exist $n \times (n - p - 1)$ centered orthonormal matrix K such that $K'X = 0$. In IMDS we will always assume that X is both regular and normalized. We also assume, unless it is explicitly

stated otherwise, that all off-diagonal weights w_{ij} are non-zero.

In "Inverse Multidimensional Scaling" (2007) two different basic IMDS versions are discussed. Both versions start with the stationary equation $B(X)X = VX$. The first finds all W and Δ for which a given X is stationary. The second finds all Δ for a given X and W for which X is stationary. In this chapter we only look at the second form of IMDS. As we shall see, the first version turns out introduce too many unknowns in W and Δ to be useful. The second form also reflects the point of view that Δ are the data, while the W are part of the definition of the loss function.

24.1 Basic IMDS

Suppose X is a regular and normalized configuration satisfying the stationary equation $(V - B(X))X = 0$. Our first IMDS step is to describe the set $\mathfrak{D}(X)$ of all ΔE for which X is stationary.

Lemma 24.1. *Suppose X is an $n \times p$ matrix of rank $p < n$. Suppose K is an $n \times (n-p)$ orthonormal matrix with $K'X = 0$. Then a symmetric matrix A satisfies $AX = 0$ if and only if there is a symmetric S such that $A = KSK'$. If rank$(K'AK) = r$ then S can be chosento be of order r.*

Proof. Suppose $X = LT$ with L an orthonormal basis for the column space of X and T non-singular. Write A as

$$A = \begin{bmatrix} L & K \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} L' \\ K' \end{bmatrix}. \tag{24.1}$$

Then $AX = LA_{11}T + KA_{21} = 0$ which is true if and only if $A_{11} = 0$ and $A_{21} = 0$, and by symmetry $A_{12} = 0$. Thus $A = KA_{22}K'$. □

Theorem 24.1. *$\Delta \in \mathfrak{D}(X)$ if and only if there is a symmetric S of order $n - p - 1$ such that for all $i \neq j$*

$$\delta_{ij} = d_{ij}(X) \left\{ 1 - \frac{k_i' S k_j}{w_{ij}} \right\}. \tag{24.2}$$

Proof. By lemma XXX we have $(V - B(X))X = 0$ if and only if there is a symmetric S such that $V - B(X) = KSK'$. This can be rearranged to yield (24.2). Since the vector e satisfies both $X'e = 0$ and $(V - B(X))e = 0$ we can choose S to be of order $n - p - 1$. □

Thus $\mathfrak{D}(X)$ is a non-empty affine space, a translation of a linear subspace, closed under all linear combinations with coefficients that add up to one. Since S is symmetric of order $n - p - 1$, equation (24.2) defines an affine subspace of dimension $\frac{1}{2}(n - p)(n - p - 1)$. If $n = 3$ and $p = 1$, or if $n = 4$ and $p = 2$, the dimension is one. If $n = 4$ and $p = 1$ the dimension is three.

If $\Delta_1, \cdots, \Delta_m$ are in $\mathfrak{D}(X)$, then so is the affine subspace spanned by the Δ_j. For all configurations X we have $D(X) \in \mathfrak{D}(X)$. Specifically, if we compute a solution to the stationary equations X with some MDS algorithm such as smacof, then the whole line through the data Δ and $D(X)$ is in $\mathfrak{D}(X)$.

The next result is corollary 6.3 in "Inverse Multidimensional Scaling" (2007).

Corollary 24.1. `r corollary_nums("full_result", display = "f")` *If $p = n - 1$ then $\Delta \in \mathfrak{D}(X)$ if and only if $\Delta = D(X)$ if and only if $\sigma(X) = 0$.*

Proof. If $p = n - 1$ then S in theorem 24.1 is of order zero. □

For any two elements of $\mathfrak{D}(X)$, one cannot be elementwise larger (or smaller) than the other. This is corollary 3.3 in "Inverse Multidimensional Scaling" (2007).

Corollary 24.2. *If Δ_1 and Δ_2 are both in $\mathfrak{DG}(X)$ and $\Delta_1 \leq \Delta_2$, then $\Delta_1 = \Delta_2$.*

Proof. With obvious notation **tr** $X'(B_1(X) - B_2(X))X = 0$, which can be written as

$$\sum_{1 \leq i < j \leq n} \sum w_{ij} \left(\delta_{1ij} - \delta_{2ij}\right) d_{ij}(X) = 0, \tag{24.3}$$

and thus $\Delta_1 \leq \Delta_2$ implies $\Delta_1 = \Delta_2$. □

24.2 Non-negative Dissimilarities

From equation (24.2) it follows δ_{ij} is a decreasing function of τ_{ij}, and $\delta_{ij} \geq 0$ if and only if $\tau_{ij} \leq w_{ij}$.

Convex cone, affine convex cone

Lemma 24.2. *A non-vacuous polyhedral convex set $C = \{x \mid Ax \leq b\}$ is bounded if and only if $Q = \{x \mid Ax \leq 0\} = \{0\}$.*

Proof. See Goldman (1956), corollary 1B. □

If C satisfies the conditions of lemma 24.2 then it is a bounded convex polyhedron and is the convex hull of its finite set of extreme vectors.

There are, of course, affine combinations Δ with negative elements. We could decide that we are only interested in non-negative dissimilarities. In order to deal with non-negativity we define Δ_+ as the polyhedral convex cone of all symmetric, hollow, and non-negative matrices.

Theorem 24.2. *We have $\Delta \in \Delta(W, X) \cap \Delta_+$ if and only if there is a symmetric S such that @(eq:invalldelta) holds and such that*

$$\textbf{low}\ (KSK') \leq \textbf{low}\ (W). \tag{24.4}$$

Thus $\Delta(W, X) \cap \Delta_+$ is a convex polyhedron, closed under non-negative linear combinations with coefficients that add up to one.

Proof. Follows easily from the representation in theorem 24.1} display = "n")'. □

Of course the minimum of $\sigma(X, W, \Delta)$ over $\Delta \in \Delta(W, X) \cap \Delta_+$ is zero, attained at $D(X)$. The maximum of stress, which is a convex quadratic in Δ, is attained at one of the vertices of $\Delta(W, X) \cap \Delta_+$.

Theorem 24.3. *$\mathfrak{D}(X) \cap \mathfrak{D}_+$ is bounded, i.e. it is a convex polygon.*

Proof. This is corollary 3.2 in "Inverse Multidimensional Scaling" (2007), but the proof given there is incorrect. A hopefully correct proof goes as follows. A polyhedron is bounded if and only if its recession cone is the zero vector.

If the polyhedron is defined by $Ax \leq b$ then the recession cone is the solution set of $Ax \leq 0$. Thus, in our case, the recession cone consists of all matrices S for which low $(KSK') \leq 0$. Since $U := KSK'$ is doubly-centered, and K is orthogonal to X, we have

$$0 = \text{tr } X'UX = 2 \sum_{1 \leq i < j \leq n} \sum u_{ij} d_{ij}^2(X). \tag{24.5}$$

\end{equation}

This implies $U = 0$, and the recession cone is the zero vector. □

We can compute the vertices of $\Delta(W, X) \cap \Delta_+$ using the complete description method of Fukuda (2015), with an `R` implementation in the `rcdd` package by Geyer and Meeden (2015). Alternatively, as a check on our computations, we also use the `lrs` method of Avis (2015), with an `R` implementation in the `vertexenum` package by Robere (2015). Both methods convert the *H-representation* of the polygon, as the solution set of a number of linear inequalities, to the *V-representation*, as the convex combinations of a number of vertices.

There is also a brute-force method of converting H to V that is somewhat wasteful, but still practical for small examples. Start with the H-representation $Ax \leq b$, where A is $n \times m$ with $n \geq m$. Then look at all $\binom{n}{m}$ choices of m rows of A. Each choice partitions A into the $m \times m$ matrix A_1 and the $(n-m) \times m$ matrix A_2 and b into $B-1$ and b_2. If $rank(A_1) < m$ there is no extreme point associated with this partitioning. If $rank(A_1) = m$ we compute $\hat{v} = A_1^{-1} b_1$ and if $A_2 \hat{v} \leq b_2$ then we add $\hat{v}$ to the V representation.

24.3 Zero Weights and/or Distances

If a distance is zero then the corresponding element of $B(X)$ must be zero as well. If a weight is zero, then the corresponding elements of both V and $B(X)$ are zero. It is still true that $(V - B)X = 0$ if and only if there is an S such that $B = V - KSK'$, but it may no longer be possible to find the Δ corresponding with some $V + KSK'$. In other words, not all solutions B to $(V - B)X = 0$ correspond with a proper $B(X)$. Specifically, zero weights and/or distances imply that one or more elements of B are required to be

zero. If these zero requirements are taken into account then not all matrices S are allowed.

If, for example, X is

```
##      [,1]
## [1,] -0.5
## [2,] -0.5
## [3,]  0.5
## [4,]  0.5
```

and the weights are all one, then $V - B$ must be a linear combination of the three matrices, say P_{11}, P_{22} and P_{12},

```
##      [,1] [,2] [,3] [,4]
## [1,]    1   -1    1   -1
## [2,]   -1    1   -1    1
## [3,]    1   -1    1   -1
## [4,]   -1    1   -1    1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1   -1   -1    1
## [2,]   -1    1    1   -1
## [3,]   -1    1    1   -1
## [4,]    1   -1   -1    1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]   -2    2    0    0
## [2,]    2   -2    0    0
## [3,]    0    0    2   -2
## [4,]    0    0   -2    2
```

and B is V minus the linear combination. For any B computed this way we have $BX = VX$, but we have $b_{12} = b_{34} = 0$ if and only if $B = V - \alpha P_{11} + (1 - \alpha)P_{22}$.

24.4 Examples

24.4.1 First Example

As our first example we take X equal to four points in the corners of a square. This example is also used in "Inverse Multidimensional Scaling" (2007) and De Leeuw (2012). Here X is

```
##      [,1] [,2]
## [1,] -0.5 -0.5
## [2,] -0.5  0.5
## [3,]  0.5  0.5
## [4,]  0.5 -0.5
```

with distances

```
##          1        2        3
## 2 1.000000
## 3 1.414214 1.000000
## 4 1.000000 1.414214 1.000000
```

and K is the vector

```
## [1] -0.5  0.5 -0.5  0.5
```

For unit weights we have $\Delta \in \Delta(X, W)$ if and only if $\Delta = D(X)\{W - \lambda kk'\}$ for some real λ. This means that $\Delta \in \Delta(X, W) \cap \Delta_+$ if and only if $-4 \leq \lambda \leq 4$. The endpoints of this interval correspond with the two dissimilarity matrices

$$\Delta_1 := 2\sqrt{2}\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

and

$$\Delta_2 := 2\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

Thus $\Delta(X,W) \cap \Delta_+$ are the convex combinations

$$\Delta(\alpha) := \alpha\Delta_1 + (1-\alpha)\Delta_2 = \begin{bmatrix} 0 & 2(1-\alpha) & 2\alpha\sqrt{2} & 2(1-\alpha) \\ 2(1-\alpha) & 0 & 2(1-\alpha) & 2\alpha\sqrt{2} \\ 2\alpha\sqrt{2} & 2(1-\alpha) & 0 & 2(1-\alpha) \\ 2(1-\alpha) & 2\alpha\sqrt{2} & 2(1-\alpha) & 0 \end{bmatrix}.$$

This can be thought of as the distances between points on a (generally non-Euclidean) square with sides $2(1-\alpha)$ and diagonal $2\alpha\sqrt{2}$. The triangle inequalities are satisfied if the length of the diagonal is less than twice the length of the sides, i.e. if $\alpha \leq \frac{2}{2+\sqrt{2}} \approx .585786$.

The distances are certainly Euclidean if Pythagoras is satisfied, i.e. if the square of the length of the diagonal is twice the square of the length of the sides. This gives $\alpha = \frac{1}{2}$, for which $\Delta = D(X)$. For a more precise analysis, observe that the two binary matrices, say E_1 and E_2, in the definition of Δ_1 and Δ_2 commute, and are both diagonalized by

$$L := \begin{bmatrix} \frac{1}{2} & \frac{1}{2}\sqrt{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2}\sqrt{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2}\sqrt{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & -\frac{1}{2}\sqrt{2} & -\frac{1}{2} \end{bmatrix}$$

The diagonal elements of $L'E_1L$ are $1, -1, -1, 1$ and those of $L'E_2L$ are $2, 0, 0, -2$. Because L diagonalizes E_1 and E_2, it also diagonalizes Δ_1 and Δ_2, as well as the elementwise squares Δ_1^2 and Δ_2^2. And consequently also the Torgerson transform $-\frac{1}{2}J\Delta^2(\alpha)J$, which has eigenvalues $0, 4\alpha^2, 4\alpha^2, 4(1-2\alpha)$. All eigenvalues are non-negative for $\alpha \leq \frac{1}{2}$.

We see that $\Delta(\alpha)$ is two-dimensional Euclidean for $\alpha = \frac{1}{2}$, one-dimensional Euclidean for $\alpha = 0$, and three-dimensional Euclidean for $0 < \alpha < \frac{1}{2}$. In particular for $\alpha = \frac{1}{1+\sqrt{2}}$ all dissimilarities are equal and $\Delta(\alpha)$ is the distance matrix of a regular simplex.

On the unit interval stress is the quadratic $32(\alpha - \frac{1}{2})^2$, which attains its maximum equal to 8 at the endpoints.

24.4.2 Second Example

We next give another small example with four equally spaced points on the line, normalized to sum of squares one, and unit weights. Thus X is

```
##             [,1]
## [1,] -0.6708204
## [2,] -0.2236068
## [3,]  0.2236068
## [4,]  0.6708204
```

and $D(X)$ is

```
##          1        2        3
## 2 3.464102
## 3 4.472136 2.828427
## 4 6.324555 4.472136 3.464102
```

For S we use the basis $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$, and $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$. Thus $\Delta(X, W)$ are the affine linear combinations of $D(X)$ and the three matrices

```
##          1        2        3
## 2 4.330127
## 3 5.590170 2.121320
## 4 4.743416 5.590170 4.330127
```

```
##          1        2        3
## 2 3.983717
## 3 3.801316 4.101219
## 4 6.640783 3.801316 3.983717
```

```
##          1        2        3
## 2 1.914908
## 3 5.472136 2.828427
## 4 6.324555 3.472136 5.013295
```

Both `scdd()` from `rcdd` and `enumerate.vertices()` from `vertexenum` find the same seven vertices. All non-negative dissimilarity matrices for which x is stationary are convex combinations of these seven matrices.

```
##           1         2         3
## 2 20.784610
## 3  0.000000  5.656854
## 4  0.000000 13.416408  0.000000
##           1         2         3
## 2 13.856406
## 3  4.472136  0.000000
## 4  0.000000 13.416408  0.000000
##          1        2        3
## 2 20.78461
## 3  0.00000 22.62742
## 4  0.00000  0.00000 20.78461
##           1         2         3
## 2  0.000000
## 3 13.416408  5.656854
## 4  0.000000  0.000000 20.784610
##           1         2         3
## 2  0.000000
## 3 13.416408  0.000000
## 4  0.000000  4.472136 13.856406
##          1        2        3
## 2 0.000000
## 3 4.472136 0.000000
## 4 8.432738 4.472136 0.000000
##           1         2         3
## 2  0.000000
## 3  0.000000  5.656854
## 4 12.649111  0.000000  0.000000
```

The stress values for these seven vertices are

```
## [1]  460.00000  248.00000 1072.00000  460.00000  248.00000   36.44444  112.
```

and we know that 1072 is the maximum of stress over $\Delta \in \Delta(X, W) \cap \Delta_+$.

In general the vanishing of the stationary equations does not imply that X corresponds with a local minimum. It can also give a local maximum or a saddle point. We know, however, that the only local maximum of stress is the origin (De Leeuw, Groenen, and Mair (2016a)), and that in the one-dimensional case all solutions of the stationary equations that do not have tied coordinates are local minima (De Leeuw (2005b)).

24.4.3 Third Example

The number of extreme points of the polytope $\Delta(W, X) \cap \Delta_+$ grows very quickly if the problem becomes larger. In our next example we take six points equally spaced on the unit sphere. Due to the intricacies of floating point comparisons (testing for zero, testing for equality) it can be difficult to determine exactly how many extreme points there are.

"Inverse Multidimensional Scaling" (2007) analyzed this example and found 42 extreme points. We repeat their analysis with our `R` function `bruteForce()` from the code section. We select $\binom{15}{6} = 5005$ sets of six linear equations from our 15 linear inequalities, test them for non-singularity, and solve them to see if they satisfy the remaining nine inequalities. This gives 1394 extreme points of the polytope, but many of them are duplicates. We use our function `cleanUp()` to remove duplicates, which leaves 42 vertices, same number as found by "Inverse Multidimensional Scaling" (2007). The 42 stress values are

```
##  [1] 24.00000 24.00000 24.00000 21.75000 12.00000 13.33333  6.66667 13.33333
##  [9] 17.33333 17.33333 19.68000 13.33333  6.66667 17.33333  6.66667 21.75000
## [17]  6.66667 60.00000 24.00000 21.75000 19.68000 17.33333 21.75000 13.33333
## [25] 19.68000 17.33333 17.33333 21.75000 17.33333 17.33333 13.33333  6.66667
## [33] 21.75000 17.33333 19.68000 13.33333 17.33333 19.68000 19.68000 17.33333
## [41]  6.66667 17.33333
```

and their maximum is 60.

If we perform the calculations more efficiently in `rcdd`, using rational arithmetic, we come up with a list of 153 extreme points. Using `cleanUp()` to

remove what seem to be duplicates leaves 43. The `vertexenum` package, which uses a different conversion from float to rational and back, finds 147 extreme points, and removing what seem to be duplicates leaves 51.

The fact that we get different numbers of vertices with different methods is somewhat disconcerting. We test the vertices found by `rcdd` and `vertexenum` that are not found with the brute force method by using our function `rankTest()`. This test is based on the fact that a vector x satisfying the $n \times m$ system $Ax \leq b$ is an extreme point if and only if matrix with all rows a_i for which $a_i'x = b_i$ is of rank m. It turns out all the additional vertices found by `rcdd` and `vertexenum` do not satisfy this rank test, because the matrix of active constraints (satisfied as equalities) is of rank 5.

24.4.4 Fourth Example

This is an unfolding example with $n = 3 + 3$ points, configuration

```
##            [,1]       [,2]
## [1,]  0.0000000  0.0000000
## [2,]  0.0000000  0.0000000
## [3,]  0.0000000 -0.8164966
## [4,]  0.0000000  0.4082483
## [5,]  0.7071068  0.0000000
## [6,] -0.7071068  0.4082483
```

and weight matrix

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    0    0    0    1    1    1
## [2,]    0    0    0    1    1    1
## [3,]    0    0    0    1    1    1
## [4,]    1    1    1    0    0    0
## [5,]    1    1    1    0    0    0
## [6,]    1    1    1    0    0    0
```

Note that row-points one and two in X are equal, and thus $d_{12}(X) = 0$. The example has both zero weights and zero distances. We now require that $(V - B(X))X = 0$, but also that the elements of $B(X)$ in the two 3×3 principal submatrices corresponding with the rows and columns are zero. This means that for the elements of $B(X)$ we require $k_i'Sk_j \leq 1$ for $w_{ij} = 1$ and both $k_i'Sk_j \leq 0$ and $-k_i'Sk_j \leq 0$ for $w_{ij} = 0$. We can then solve for edges of the off-diagonal block of dissimilarities. There are no constraints on the dissimilarities in the diagonal blocks, because they are not part of the MDS problem.

Using our brute force method, we find the two edges

```
##           4        5        6
## 1 0.0000000 1.414214 1.632993
## 2 0.8164966 0.000000 0.000000
## 3 1.2247449 1.080123 1.414214
```

```
##           4        5        6
## 1 0.8164966 0.000000 0.000000
## 2 0.0000000 1.414214 1.632993
## 3 1.2247449 1.080123 1.414214
```

and the off-diagonal blocks for which X is an unfolding solution are convex combinations of these two.

24.5 MDS Sensitivity

Suppose X is a solution to the MDS problem with dissimilarities Δ, found by some iterative algorithm such as `smacof`. We can then compute $\mathfrak{D}(X) \cap \mathfrak{D}_+$, which is a convex neighborhood of the data Δ, and consists of all non-negative dissimilarity matrices that have X as a solution to the stationary equations. The size of this convex neighborhood can be thought of as a measure of *stability* or *sensitivity*.

For typical MDS examples there is no hope of computing all vertices of $\mathfrak{D}(X) \cap \mathfrak{D}_+$. Consider the data from De Gruijter (1967), for example, with $n = 9$ objects, to be scaled in $p = 2$ dimensions. We have $\frac{1}{2}n(n-1) = 36$

dissimilarities, and because $m = n - p - 1 = 6$ there are $\frac{1}{2}m(m+1) = 21$ variables. It suffices to consider that there are 5567902560 ways in which we can pick 21 rows from 36 rows to understand the number of potential vertices.

What we can do, however, is to optimize linear (or quadratic functions) over $\mathfrak{D}(X) \cap \mathfrak{D}_+$, because 36 linear inequalities in 21 variables define an easily manageable LP (or QP) problem. As an example, not necessarily a very sensible one, we solve 36 linear programs to maximize and minimize each of the δ_{ij} in $\mathfrak{D}(X) \cap \mathfrak{D}_+$ separately. We use the `lpSolve` package (Berkelaar, M. and others (2015)), and collect the maximum and minimum δ_{ij} in a matrix. The range from the smallest possible δ_{ij} to the largest possible δ_{ij} turns out to be quite large.

The data are

```
##       KVP PvdA  VVD  ARP  CHU  CPN  PSP   BP  D66
## KVP  0.00 5.63 5.27 4.60 4.80 7.54 6.73 7.18 6.17
## PvdA 5.63 0.00 6.72 5.64 6.22 5.12 4.59 7.22 5.47
## VVD  5.27 6.72 0.00 5.46 4.97 8.13 7.55 6.90 4.67
## ARP  4.60 5.64 5.46 0.00 3.20 7.84 6.73 7.28 6.13
## CHU  4.80 6.22 4.97 3.20 0.00 7.80 7.08 6.96 6.04
## CPN  7.54 5.12 8.13 7.84 7.80 0.00 4.08 6.34 7.42
## PSP  6.73 4.59 7.55 6.73 7.08 4.08 0.00 6.88 6.36
## BP   7.18 7.22 6.90 7.28 6.96 6.34 6.88 0.00 7.36
## D66  6.17 5.47 4.67 6.13 6.04 7.42 6.36 7.36 0.00
```

The optimal configuration found by `smacof` is

```
##        [,1]   [,2]
##  [1,]  1.78  3.579
##  [2,] -1.46  2.297
##  [3,]  3.42 -2.776
##  [4,]  3.30  1.837
##  [5,]  3.84  0.308
##  [6,] -5.09 -0.044
##  [7,] -3.79  2.132
##  [8,] -2.86 -4.318
##  [9,]  0.86 -3.015
```

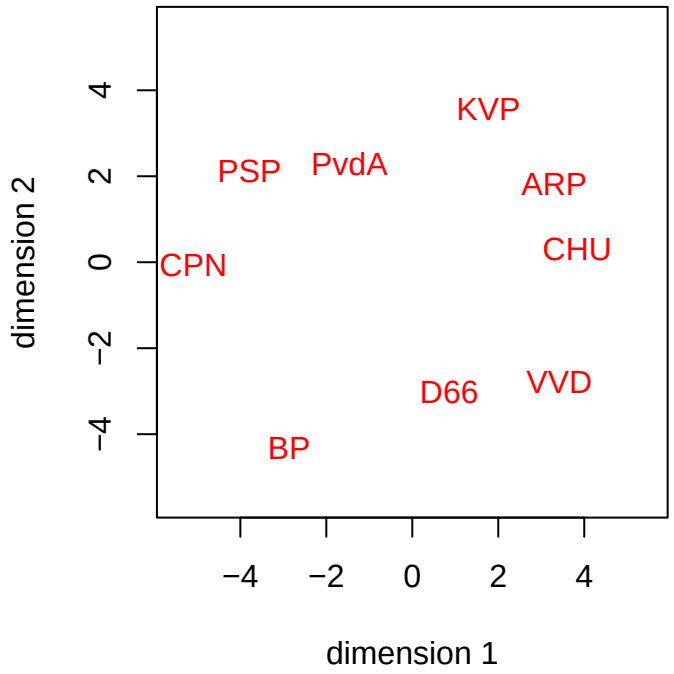


Figure 24.1: De Gruijter Configuration

The maximum and minimum dissimilarities in $\Delta(X, W) \cap \Delta_+$ are

```
##       KVP   PvdA  VVD   ARP   CHU   CPN   PSP   BP    D66
## KVP    0.0  32.6   9.7   9.3  24.6  24.3  11.7  18.9  15.2
## PvdA  32.6   0.0  25.7  36.9  20.7  34.1  36.1  29.7  22.2
## VVD    9.7  25.7   0.0  17.4  32.0  34.4  22.3  23.5  20.4
## ARP    9.3  36.9  17.4   0.0  28.0  33.7  25.6  28.9  23.0
## CHU   24.6  20.7  32.0  28.0   0.0  20.8  31.9  33.3  26.3
## CPN   24.3  34.1  34.4  33.7  20.8   0.0  24.9  30.7  31.8
## PSP   11.7  36.1  22.3  25.6  31.9  24.9   0.0  23.7  27.7
## BP    18.9  29.7  23.5  28.9  33.3  30.7  23.7   0.0  35.0
## D66   15.2  22.2  20.4  23.0  26.3  31.8  27.7  35.0   0.0
```

```
##       KVP   PvdA  VVD   ARP   CHU   CPN   PSP   BP    D66
## KVP   0.00  3.48  0.00  0.00  2.34  0.00  0.00  0.00  0.00
## PvdA  3.48  0.00  0.00  0.00  0.00  0.00  0.93  0.00  0.00
## VVD   0.00  0.00  0.00  0.00  0.83  0.00  0.00  0.00  0.00
## ARP   0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
## CHU   2.34  0.00  0.83  0.00  0.00  0.00  0.00  0.00  0.00
## CPN   0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
## PSP   0.00  0.93  0.00  0.00  0.00  0.00  0.00  0.00  0.00
## BP    0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
## D66   0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
```

24.6 Second Order Inverse MDS

As we mentioned many times before, stationary values are not necessarily local minima. It is necessary for a local minimum that the stationary equations are satisfied, but it is also necessary that at a solution of the stationary equations the Hessian is positive semi-definite. Thus it becomes interesting to find the dissimilarities in $\Delta(W, X) \cap \Delta_+$ for which the Hessian is positive semi-definite. Define

$$\Delta_H(W, X) := \{\Delta \mid \mathcal{D}^2\sigma(X) \gtrsim 0\}.$$

We can now study $\Delta(W, X) \cap \Delta_H(W, X)$ or $\Delta(W, X) \cap \Delta_H(W, X) \cap \Delta_+$.

Theorem 24.4. *$\Delta_H(W,X)$ is a convex non-polyhedral set.*

Proof. In MDS the Hessian is $2(V - H(x,W,\Delta))$, where

$$H(x,W,\Delta) := \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}(x)}\left\{A_{ij} - \frac{A_{ij}xx'A_{ij}}{x'A_{ij}x}\right\}. \tag{24.6}$$

Here we use $x := \mathbf{vec}(X)$, and both A_{ij} and V are direct sums of p copies of our previous A_{ij} and V (De Leeuw, Groenen, and Mair (2016e)). The Hessian is linear in Δ, which means that requiring it to be positive semi-definite defines a convex non-polyhedral set. The convex set is defined by the infinite system of linear inequalities $y'H(x,W,\Delta)y \geq 0$. □

In example 3 the smallest eigenvalues of the Hessian at the 42 vertices are

```
##  [1]   0.00000   0.00000   0.00000  -5.86238   0.00000  -2.96688   0.00000
##  [8]  -2.96688  -4.47894  -4.47894  -5.71312  -2.96688   0.00000  -5.00453
## [15]   0.00000  -5.86238   0.00000 -12.00000   0.00000  -5.86238  -5.71312
## [22]  -5.00453  -5.86238  -2.96688  -5.71312  -4.47894  -5.00453  -5.86238
## [29]  -5.00453  -5.00453  -2.96688   0.00000  -5.86238  -4.47894  -5.71312
## [36]  -2.96688  -5.00453  -5.71312  -5.71312  -4.47894   0.00000  -4.47894
```

and thus there are at most 11 vertices corresponding with local minima.

The next step is to refine the polyhedral approximation to $\Delta(W,X) \cap \Delta_H(W,X) \cap \Delta_+$ by using cutting planes. We add linear inequalities to the H-representation by using the eigenvectors corresponding to the smallest eigenvalues of all those vertices for which this smallest eigenvalue is negative. Thus, if the eigenvector is y, we would add the inequality

$$\sum_{1\leq i<j\leq n}\sum \zeta_{ij}(w_{ij} - k_i'Sk_j) \geq 0, \tag{24.7}$$

where

$$\zeta_{ij} := y'\left(A_{ij} - \frac{A_{ij}xx'A_{ij}}{x'A_{ij}x}\right)y.$$

It is clear, however, that adding a substantial number of linear inequalities will inevitably lead to a very large number of potential extreme points. We proceed conservatively by cutting off only the solution with the smallest negative eigenvalue in computing the new V representation, using our function `bruteForceOne()`.

24.7 Inverse FDS

A configuration X is a *full dimensional scaling* or *FDS* solution (De Leeuw, Groenen, and Mair (2016a)) if $(V - B(X))X = 0$ and $V - B(X)$ is positive semi-definite. In that case X actually provides the global minimum of stress. The *inverse FDS* or *iFDS* problem is to find all Δ such that a given X is the FDS solution.

Theorem 24.5. *$\Delta \in \Delta_F(W, X)$ if and only if there is a positive semi-definite S such that for all $i \neq j$*

$$\delta_{ij} = d_{ij}(X)(1 - \frac{1}{w_{ij}}k_i'Sk_j), \tag{24.8}$$

Thus $\Delta_F(W, X)$ is a non-polyhedral convex set, closed under linear combinations with coefficients that add up to one.

Proof. Of course $\Delta_F(W, X) \subseteq \Delta(W, X)$. Thus $V - B(X) = KSK'$ for some S, and XX' and $V - B(X)$ must both be positive semi-definite, with complementary null spaces. □

We reanalyze our second example, with the four points equally spaced on the line, requiring a positive semi-definite S. We start with the original 7 vertices, for which the minimum eigenvalues of S are

```
## [1] -4.000 -2.899 -1.708 -3.333  8.000 -1.708 -2.899
```

If the minimum eigenvalue is negative, with eigenvector y, we add the constraints $y'Sy \geq 0$. This leads to 11 vertices with minimum eigenvalues

```
##  [1]  8.0000 -0.0355  0.0000 -0.7911 -0.3834 -0.3834 -0.0355 -0.0853 -0.085
## [10] -0.7911  0.0000
```

We see that the negative eigenvalues are getting smaller. Repeating the procedure of adding constraints based on negative eigenvalues three more times gives 19, 37, and 79 vertices, with corresponding minimum eigenvalues

```
##  [1]  8.000000 -0.000021  0.000000 -0.209852 -0.202322 -0.085642 -0.085642
##  [8] -0.000021 -0.106132 -0.018912 -0.008051 -0.023967 -0.008051 -0.023967
## [15] -0.106132 -0.018912 -0.209852 -0.202322  0.000000
```

```
##  [1]  8.000000 -0.000021  0.000000 -0.056872 -0.053562 -0.045307 -0.062028
##  [8] -0.020853 -0.020853 -0.000021 -0.028793 -0.004490 -0.001925 -0.006410
## [15] -0.001925 -0.006410 -0.028793 -0.004490 -0.000005 -0.002104 -0.021934
## [22] -0.024355 -0.021934 -0.024355 -0.000005 -0.002104 -0.004975 -0.005596
## [29] -0.005596 -0.004975 -0.002318 -0.002318 -0.056872 -0.053562 -0.045307
## [36] -0.062028  0.000000
```

```
##  [1]  8.000000 -0.000021  0.000000 -0.015026 -0.013698 -0.010861 -0.017838
##  [8] -0.013830 -0.013426 -0.012084 -0.013998 -0.018140 -0.005180 -0.005180
## [15] -0.000021 -0.007568 -0.001096 -0.000471 -0.001662 -0.000471 -0.001662
## [22] -0.007568 -0.001096 -0.000005 -0.000001 -0.000538 -0.000488 -0.005588
## [29] -0.005886 -0.005588 -0.005886 -0.000005 -0.000001 -0.000538 -0.000488
## [36] -0.001278 -0.001355 -0.001355 -0.001278 -0.000001 -0.000649 -0.000594
## [43] -0.005244 -0.005378 -0.005244 -0.005378 -0.000001 -0.000649 -0.000594
## [50] -0.006838 -0.006293 -0.001150 -0.001210 -0.000492 -0.000514 -0.000566
## [57] -0.001545 -0.001444 -0.000492 -0.000514 -0.000566 -0.001545 -0.001444
## [64] -0.006838 -0.006293 -0.001150 -0.001210 -0.000001 -0.000001 -0.015026
## [71] -0.013698 -0.010861 -0.017838 -0.013830 -0.013426 -0.012084 -0.013998
## [78] -0.018140  0.000000
```

It should be noted that the last step already takes an uncomfortable number of minutes to compute. Although the number of vertices goes up quickly, the diameter of the polygon (the maximum distance between two vertices) slowly goes down and will eventually converge to the diameter of $\Delta_F(W, X) \cap \Delta_+$. Diameters in subsequent steps are

```
## [1] 5.657 5.086 5.065 5.061 5.060
```

24.8 Multiple Solutions

If $X_1, \cdots, X_s$ are configurations with the same number of points, then the intersection $\{\bigcap_{r=1}^{s} \Delta(W, X_r)\} \cap \Delta_+$ is again a polygon, i.e. a closed and bounded

convex polyhedron (which may be empty). If Δ is in this intersection then $X_1, \cdots, X_s$ are all solutions of the stationary equations for this Δ and W.

Let's look at the case of two configurations X_1 and X_2. We must find vectors t_1 and t_2 such that

$$d_1 \circ (e - w^\dagger \circ G_1 t_1) = d_2 \circ (e - w^\dagger \circ G_2 t_2).$$

If $H_1 := \mathbf{diag}(d_1 \circ w^\dagger) G_1$ and $H_2 := \mathbf{diag}(d_2 \circ w^\dagger) G_2$ then this can be written as

$$\begin{bmatrix} H_1 & -H_2 \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = d_1 - d_2$$

This is a system of linear equations in t_1 and t_2. If it is solvable we can intersect it with the convex sets $G_1 t_1 \leq w$ and $G_2 t_2 \leq w$ to find the non-negative dissimilarity matrices Δ for which both X_1 and X_2 are stationary.

```
## $delta1
##          [,1]
## [1,] 1.464102
## [2,] 1.464102
## [3,] 1.464102
## [4,] 1.464102
## [5,] 1.464102
## [6,] 1.464102
##
## $delta2
##          [,1]
## [1,] 1.464102
## [2,] 1.464102
## [3,] 1.464102
## [4,] 1.464102
## [5,] 1.464102
## [6,] 1.464102
##
## $res
## [1] 1.024137e-15
##
## $rank
## [1] 2
```

As a real simple example, suppose X and Y are four by two. They both have three points in the corners of an equilateral triangle, and one point in the centroid of the other three. In X the fourth point is in the middle, in Y the first point. The only solution to the linear equations is the matrix with all dissimilarities equal.

For a much more complicated example we can choose the De Gruijter data. We use `smacof` to find two stationary points in two dimensions. The matrices G_1 and G_2 are 36×21, and thus $H := (H_1 \mid -H_2)$ is 36×42. The solutions of the linear system $Ht = d_1 - d_2$ are of the form $t_0 - Lt$, with t_0 an arbitrary solution and L a 42×6 basis for the space of $Ht = 0$. To find the non-negative solutions we can use the H representation $Lv \leq t_0$, and then compute the V representation, realizing of course that we can choose 6 rows from 42 rows in 5245786 ways.

24.9 Minimizing iStress

The IMDS approach can also be used to construct an alternative MDS loss function. We call it *iStress*, defined as

$$\sigma_\iota(X, W, \Delta) := \min_{\tilde{\Delta} \in \Delta(W,X) \cap \Delta_+} \sum\sum_{1 \leq i < j \leq n} w_{ij}(\delta_{ij} - \tilde{\delta}_{ij})^2.$$

Minimizing iStress means minimizing the projection distance between the observed dissimilarity matrix and the moving convex set of non-negative dissimilarity matrices for which X satisfies the stationary equations. The convex set is moving, because it depends on X. For each X we have to solve the IMDS problem of finding $\Delta(X, W) \cap \Delta_+$, and then solve the quadratic programming problem that computes the projection.

Theorem 24.6. $\min_X \sigma_\iota(X, W, \Delta) = 0$ *and the minimum is attained at all* X *with* $(V - B(X))X = 0$.

Proof. If X is a stationary point of stress then $\Delta \in \mathfrak{D}(X) \cap \mathfrak{D}_+$ and thus iStress is zero. Conversely, if iStress is zero then $\Delta \in \mathfrak{D}(X) \cap \mathfrak{D}_+$ and X is a stationary point of stress. □

Minimizing iStress may not be a actual practical MDS method, but it has some conceptual interest, because it provides another way of looking at the relationship of MDS and IMDS.

We use the De Gruijter data for an example of iStress minimization. We use `optim` from base `R`, and the `quadprog` package of Turlach and Weingessel (2013). Two different solutions are presented, the first with iterations starting at the classical MDS solution, the second starting at the `smacof` solution. In both cases iStress converges to zero, i.e. the configurations converge to a solution of the stationary equations for stress, and in the `smacof` case the initial solution already has stress equal to zero.

```
## initial  value 106.624678
## iter  10 value 0.205734
## iter  20 value 0.025163
## iter  30 value 0.018427
## iter  40 value 0.000116
## iter  50 value 0.000007
## iter  60 value 0.000000
## final  value 0.000000
## converged
```

```
## initial  value 0.000000
## iter  10 value 0.000000
## iter  20 value 0.000000
## final  value 0.000000
## converged
```

24.10 Order three

We will now consider, in some detail, MDS of a dissimilarity matrix of order three.. The plan is as follows. Our MDS problem is of order three, i.e. there are only three objects, and three dissimilarities between them. Suppose, for simplicity, that all weights are one. At a stationary point of we have $B(X)X = 3X$. Thus the B matrix has an eigenvalue equal to 3 (in addition to having an eigenvalue equal to zero).

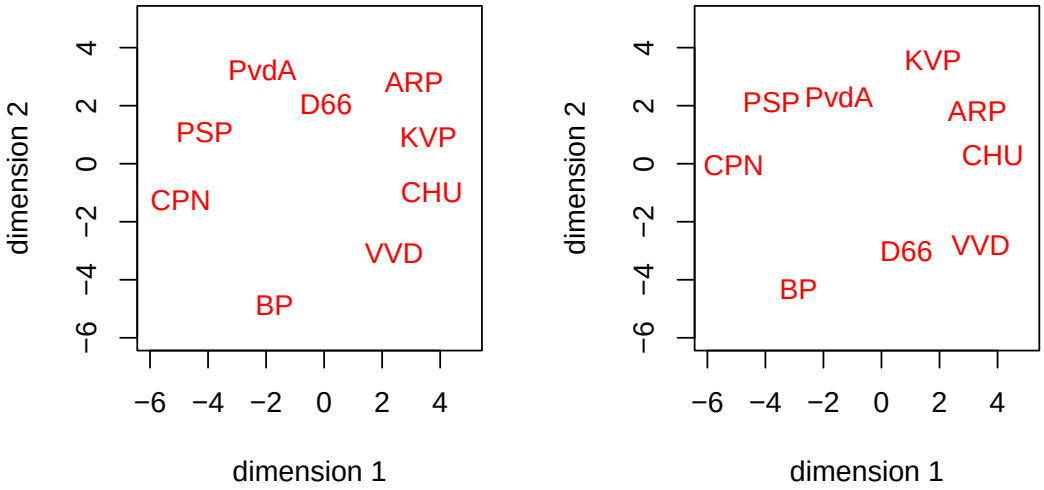


Figure 24.2: De Gruijter Minimim iStress Configuration

!! Not just eval 3, but also evecs X

If there are only three objects we can look at the set of doubly-centered matrices of order three with an eigenvalue equal to three. If three is the largest eigenvalue then the local minimum is global, if it is the second largest eigenvalue then it may or may not be global. If B has two eigenvalues equal to three, then $B = 3J$ and the two eigenvectors give the unique global minimum in two dimensions (an equilateriaL triangle).

A symmetric doubly-centered matrix of order three is a linear combination $B = \alpha A_{12} + \beta A_{13} + \gamma A_{23}$ of three diff matrices (@ref(#propmatrix)), with α, β, γ all non-negative. Thus

$$B = \begin{bmatrix} \alpha+\beta & -\alpha & -\beta \\ -\alpha & \alpha+\gamma & -\gamma \\ -\beta & -\gamma & \beta+\gamma \end{bmatrix}. \tag{24.9}$$

B has one eigenvalue equal to zero, and two real non-negative eigenvalues λ_1 and λ_2. The characteristic equation is $f(\lambda) := \lambda(\lambda^2 - 2\tau\lambda + 3\eta) = 0$, where $\tau := \alpha + \beta + \gamma$ and $\eta := \alpha\beta + \alpha\gamma + \beta\gamma$.

Thus we have at least one eigenvalue equal to three if $f(3) = 0$, i.e. $2\tau - \eta = 3$. Both eigenvalues are equal to three if $\eta = \tau = 3$, which means $\alpha = \beta = \gamma = 1$ and $B = 3J$.

The two eigenvalues are $\tau \pm \sqrt{\tau^2 - 3\eta}$ Note that $\tau^2 - 3\eta \geq 0$ with equality if and only if $\tau = \eta = 3$ if and only if $\alpha = \beta = \gamma = 1$. This easily follows from $\tau^2 - 3\eta = \alpha^2 + \beta^2 + \gamma^2 - \eta$, while $\eta = \alpha\beta + \alpha\gamma + \beta\gamma \leq \alpha^2 + \beta^2 + \gamma^2$ by applying AM/GM three times. Also note that if two out of the three of α, β, γ are zero then $\eta = 0$ and thus $\lambda_1 = 2\tau$ and $\lambda_2 = 0$.

Since $\lambda_1 \geq \lambda_2$ we have the two possibilities $\lambda_2 \leq \lambda_1 = 3$ and $3 \geq \lambda_1 \geq \lambda_2 = 3$ Now $\lambda_1 = 3$ iff $\sqrt{\tau^2 - 3\eta} = 3 - \tau$ iff $\tau \leq 3$ and $2\tau - \eta = 3$. And $\lambda_2 = 3$ iff $\sqrt{\tau^2 - 3\eta} = \tau - 3$ iff $\tau \geq 3$ and $2\tau - \eta = 3$. It also follows that it is necessary for $\lambda = 3$ that $\tau \geq \frac{3}{2}$.

If we have (τ, η) we can solve for α, β and γ by

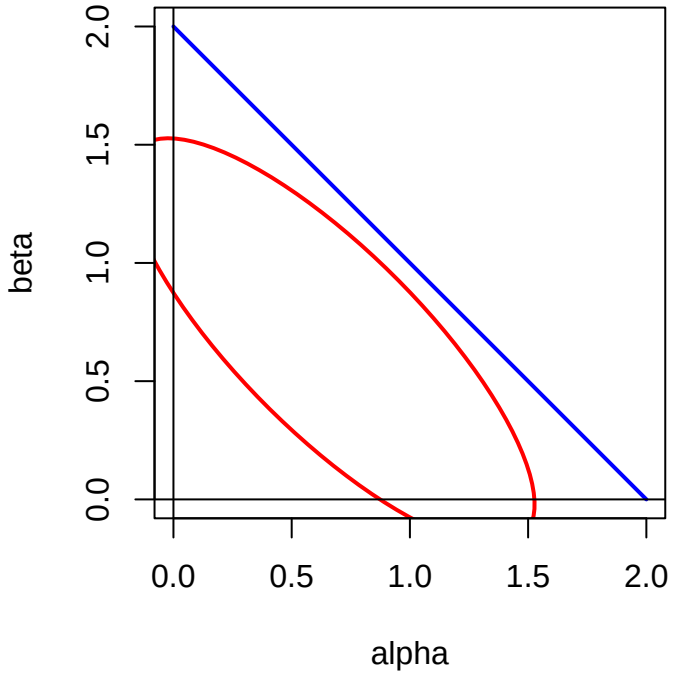
$$\alpha + \beta + \gamma = \tau, \alpha^2 + \beta^2 + \gamma^2 = \omega$$

Thus the set of (α, β, γ) corresponding with τ, η is the intersection of a sphere and an equilateral triangle in the non-negative orthant of $\mathbb{R}^3$.

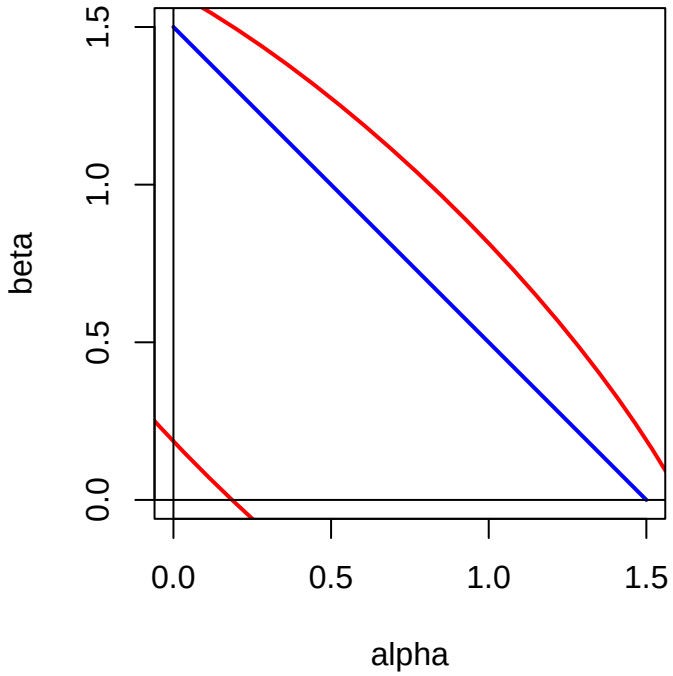
$$\gamma = \tau - \alpha - \beta \alpha^2 + \beta^2 - \tau\alpha - \tau\beta + \alpha\beta = -\eta$$

ellipse with center $\frac{1}{3}\tau e$ and radius $\frac{1}{3}(\tau - 3)^2$

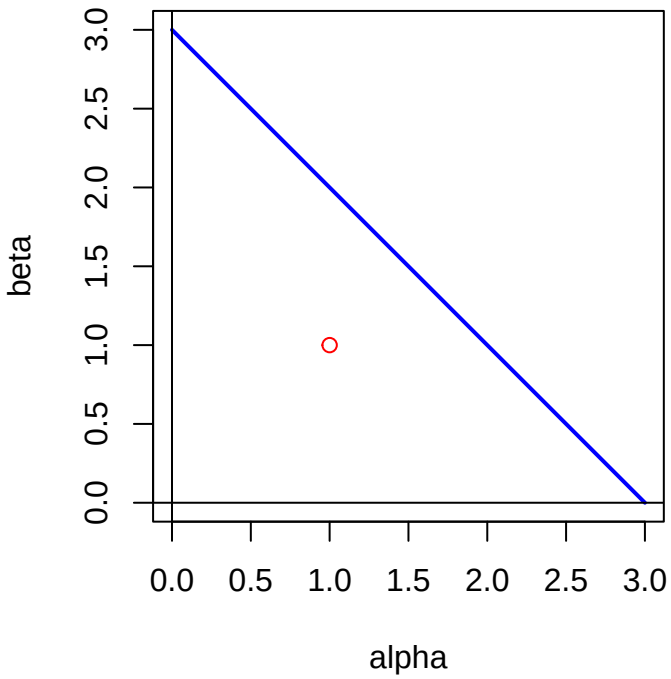
```
imdsSolver(2)
```



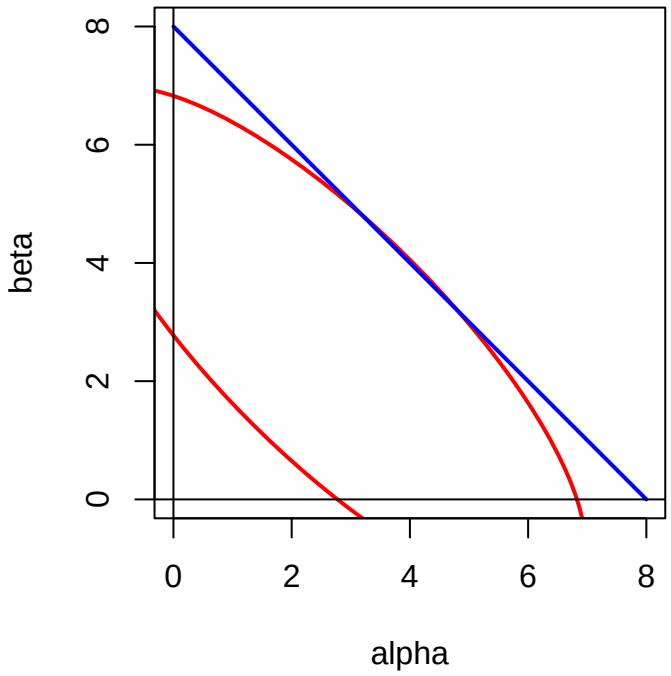

```
imdsSolver(3/2)
```



```
imdsSolver(3)
points (1, 1, col = "RED")
```




```
imdsSolver(8)
```



24.11 Order Four

$$B = \alpha A_{12} + \beta A_{13} + \gamma A_{14} + \delta A_{23} + \epsilon A_{24} + \phi A_{34}$$

$$B = \begin{bmatrix} \alpha+\beta+\gamma & -\alpha & -\beta & -\gamma \\ -\alpha & \alpha+\delta+\epsilon & -\delta & -\epsilon \\ -\beta & -\delta & \beta+\delta+\phi & -\phi \\ -\gamma & -\epsilon & -\phi & \gamma+\epsilon+\phi \end{bmatrix}.$$

The characteristic equation is

$$f(\lambda) = \lambda(\lambda^3 - 2\tau\lambda^2 + (3\eta + (\alpha\phi + \beta\epsilon + \gamma\delta))\lambda - Y).$$

$$Z = \frac{1}{2}\begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & +1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}.$$

24.12 Sensitivity

$$\sigma(W, \Delta) = \min_X \sigma(X, W, \Delta)$$

$$X(W, \Delta) = \operatorname*{Argmin}_X \sigma(X, W, D)$$

$$\mathcal{D}_1\sigma(W, \Delta) = (\Delta - D(X(W, D)))^{(2)}$$

$$\mathcal{D}_2\sigma(W, \Delta) = W * (\Delta - D(X(W, D)))$$

Chapter 25

Stability of MDS Solutions

25.1 Null Distribution

25.2 Pseudo-confidence Ellipsoids

25.3 A Pseudo-Bootstrap

25.4 How Large is My Stress ?

Users of MDS, influenced no doubt by the tyranny of significance tests, often ask if their stress level from a smacof analysis is low or high. The appropriate answer, in many cases, is: "It's as low as I could get it."

Kruskal has muddied the waters.

Normalization. Remember

$$\sigma(X) = \frac{\sum\sum_{1\leq i<j\leq n} w_{ij}(\delta_{ij} - d_{ij}(X))^2}{\sum\sum_{1\leq i<j\leq n} w_{ij}\delta_{ij}^2}$$

so it is the weighted average of the squared residuals, relative to the weighted average of the squared dissimilarities. If we take the square root then weighted averages of squares become root-mean-squares.

Thurstonian

Ramsay

Monte Carlo: De Leeuw and Stoop (1984), Spence (1983)

$$\mathbb{E}(\chi_p) = \sqrt{2}\frac{\Gamma(\frac{p+1}{2})}{\Gamma(\frac{p}{2})}$$

Chapter 26

In Search of Global Minima

We have already discussed the problem of finding the global minimum, instead of merely one or more local minima, in chapters 13 and 12. In the unidimensional case the basic MDS problem becomes combinatorial, we have to minimize over all $n!$ permutations of ι_n, and there usually are very many local minima, all of them strict. The case in which all δ_{ij} are the same shows there can be $n!$ local minima, all global. All these minima are strict and isolated, and thus a small perturbation of the equal-dissimilarity case still has $n!$ local minima (Pliner (1996)). In the full-dimensional case there is only one minimum, which is by definition the global minimum. For $1 < p < n-1$ we can expect to be somewhere between these two extremes, with in addition the possibility that some of the critical points are saddle points and not local minima. But note that if all dissimilarities are equal all permutations of the points in the global minimum configuration also give global minima, which means that even in higher dimensional cases we may have $n!$ local minima.

One way to protect against non-global minima is to start with a really good initial configuration. Generally, the Torgerson and Guttman initial configurations are helpful, and so are the first p principal components of the full-dimensional solution. Another important tool is to stop iterations using the size of the gradient (or the difference between X and $\gamma(X)$), with a cut-off value of at least $1e-7$, but preferably $1e-10$ or even $1e-15$. Earlier implementations of smacof may have stopped too soon if the target local minimum is in a flat region of the configuration space, or if the iterations stray too close to a saddle point and must flex their muscle to get away from

it.

In this chapter we will discuss some methods to find the global minimum, or, more modestly, to move from one local minimum to another better local minimum. The global optimization battlefield is in constant flux. New methods for general of specific global optimization problems seem to be invented every day, all struggling with the curse of dimensionality. The field is riddled with the remains of methods that died in infancy. So I am not saying I will discuss the best, or even the most promising, global optimization methods for MDS. I simply choose the ones I like best, and the ones that fit nicely into the smacof framework.

26.1 Random Starts

The simplest way to get an idea about the local minima of stress in any specific example is to run smacof with multiple random initial configurations. The implementation is simple. Put the smacof runs in a loop from 1 to N, start each run with a random initial configuration, and collect the results in some data structure. It is true that our analysis will take N times as long to finish, but just start up your PC and go and do something else while it runs. It seems to me that this ought to be standard practice for actual MDS applications. Not only do we find the best local minimum in terms of stress, but we get valuable information about the stability of our result. If we take, for example, $N = 1000$ and we find the same local minimum in all runs, we can be reasonably confident that we have found the global minimum. A small change in the dissimilarities will probably find the same global minimum. If we find multiple local minima, all with about the same frequency and with stress values that are close, then a small change may very well switch to another local minimum with the smallest stress value.

There is some freedom in the choice of method to generate random initial configurations. In the examples in this chapter we sample from the standard multivariate normal, but since we know that at local minima $\eta(X) \leq 1$ it may be more appropriate to sample from the unit ball in $\mathbb{R}^{n\times p}$ (Harman and Lacko (2010)). In fact, since we also know that at a local minimum $\rho(X) = \eta^2(X)$, we may project our intial configurations on that surface.

metric - nonmetric

Table 26.1: Local Minima in Ekman Example

no	minimum	frequency
1	0.0086066	824
2	0.0182714	136
3	0.0300985	19
4	0.0309922	7
5	0.0317395	8
6	0.0331013	4
7	0.0337281	1
8	0.0379281	1

26.1.1 Examples

26.1.1.1 Ekman

We use the Ekman data to illustrate this. We use 1000 random starts, and we stop iterating when the decrease in stress is less than 1e-15. We find 8 local minima, listed in table 26.1.

The results are encouraging, because they indicate that, at least in the Ekman example, the lower the local minimum, the more attractive it is for the smacof iterations. The lowest local minima seem to have the largest regions of attraction. Specifically, we find what is presumably the global minimum in more than 80% of the cases. Nevertheless, if our clients were so unwise to start their MDS analysis with a random initial configuration then about 20% of them will get the wrong answer.

In the example we also see a number of local minima, found only in a small number of cases, whose stress values are very close to each other. It seems likely that others of roughly the same stress level will be found if we continue sampling additional random initial configurations.

The configurations corresponding with the two dominant local minima are in 26.3 and 26.4. if we compare them we see that the color circle has two separate circular segments. In the second local minimum the order of the colors on one of these two segments is reversed.

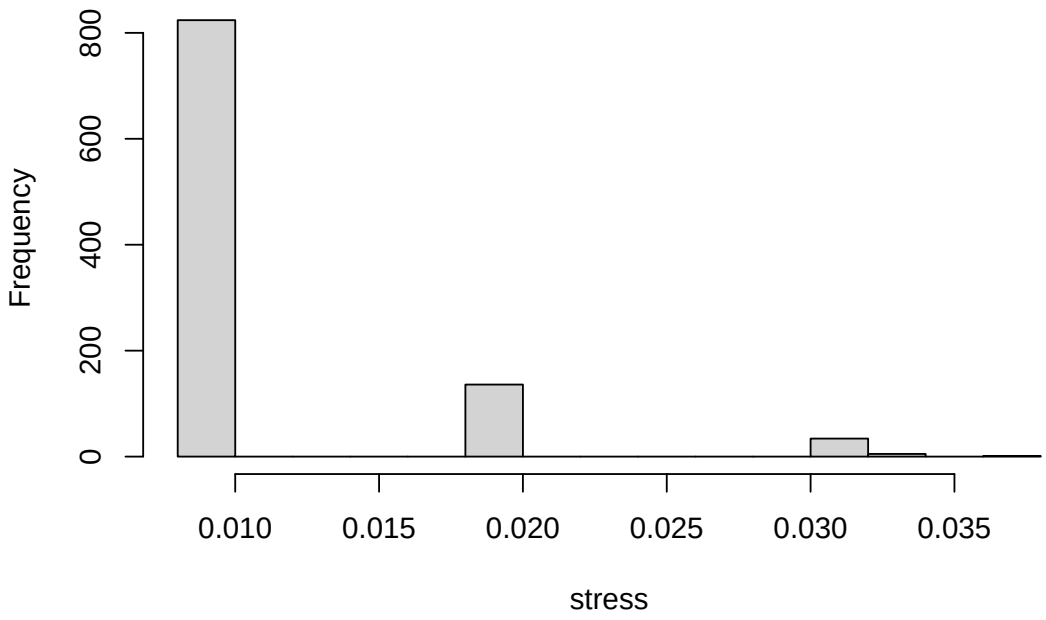


Figure 26.1: Ekman Stress Values

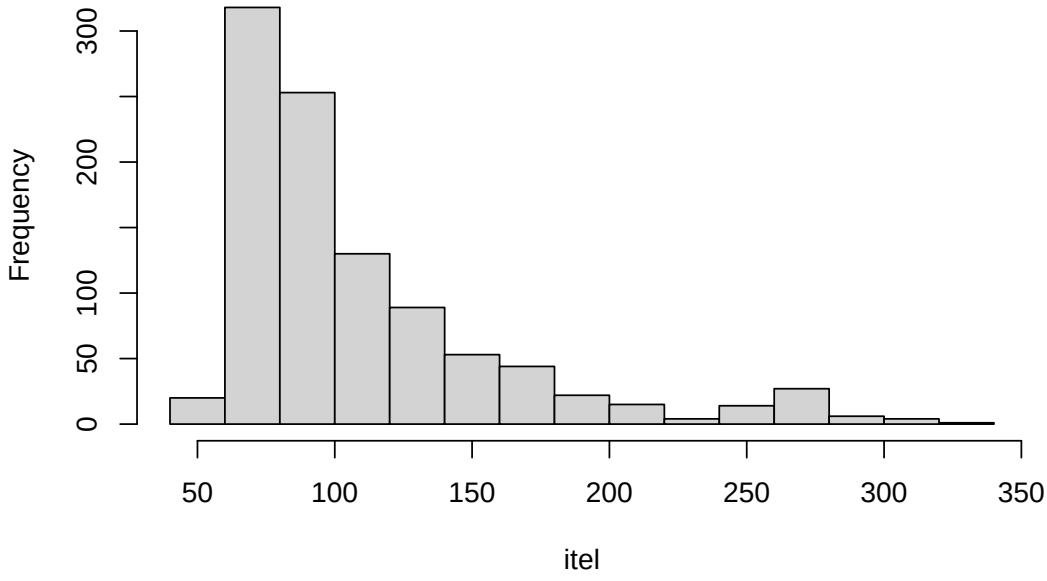


Figure 26.2: Ekman Iteration Counts

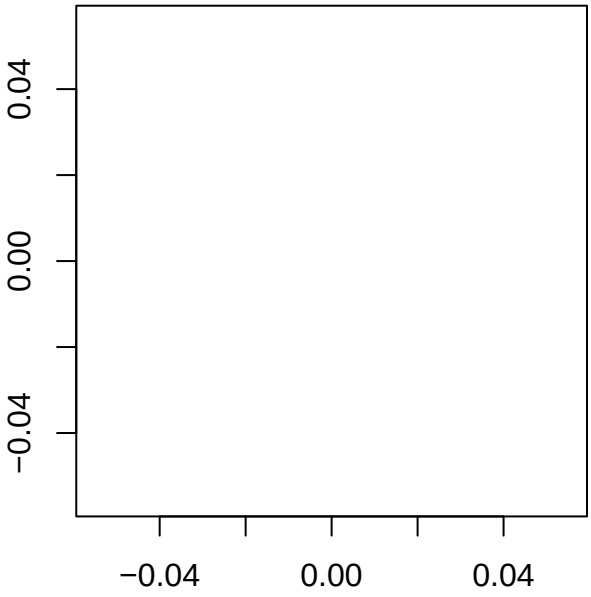


Figure 26.3: Global (?) Minimum

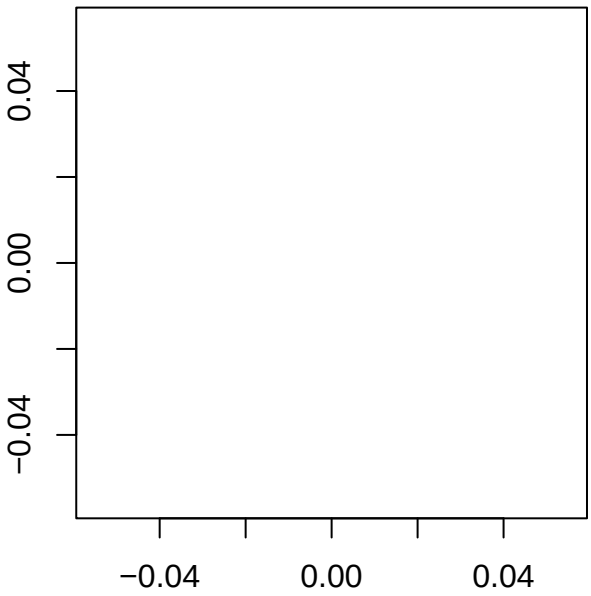


Figure 26.4: Largest (?) Non-global Minimum

26.1.1.2 De Gruijter

The Ekman example is somewhat atypical, because it has an exceptionally good fit. We perform the same computations for the De Gruijter example, still using a cut-off at 1e-15, but now allowing for up to 10,000 iterations.

The data in this example are averages of preference rankings for nine Dutch political parties by 100 students. Due to the heterogeneity of the population there is a considerable regression to the mean. Typically this would suggest splitting the students into more homogeneous groups (which is what De Gruijter (1967) did), and/or using a form of non-metric scaling (which is what De Gruijter did as well).

Our 1000 runs produce 21 local minima, in table @ref{tab:gruijterlocalminima}, with stress values that are close to each other. In this case it is easy to imagine that more runs will produce many more local minima, with low frequencies, mainly permuting the political parties on the horseshoe. In other words, the situation is somewhat like the case in which all dissimilarities between the nine objects are equal, in which case we have 9! =3.6288×10^5 local minima, all with the same stress value.

Another comparison may be useful. The Ekman example is like a matrix with two dominant eigenvalues, the De Gruijter example is like a matrix will all eigenvalues approximately equal to each other. Since smacof is somewhat like the power method, we expect poor convergence in the De Gruijter example, and histogram 26.6 shows exactly that. There is even one random start from which there is no convergence in 10,000 iterations. In the Ekman example the frequency of the local minima seems closely related to the stress value at the local minimum, in the De Gruijter example the frequency of the local minims seems more random. In the Ekman case we can be pretty sure we have found the global minimum, in the De Gruijter case we are far from sure.

```
x1 <- hgruijter[[14]]$x
x3 <- hgruijter[[1]]$x
```

Table 26.2: Local Minima in De Gruijter Example

no	minimum	frequency
1	0.0222149	155
2	0.0222262	85
3	0.0222631	156
4	0.0223017	248
5	0.0232310	130
6	0.0244955	88
7	0.0245003	35
8	0.0246216	2
9	0.0254775	26
10	0.0254982	18
11	0.0261988	18
12	0.0273897	1
13	0.0274695	7
14	0.0297550	4
15	0.0303408	4
16	0.0315674	1
17	0.0315679	1
18	0.0356327	14
19	0.0362333	3
20	0.0364538	3
21	0.0375689	1

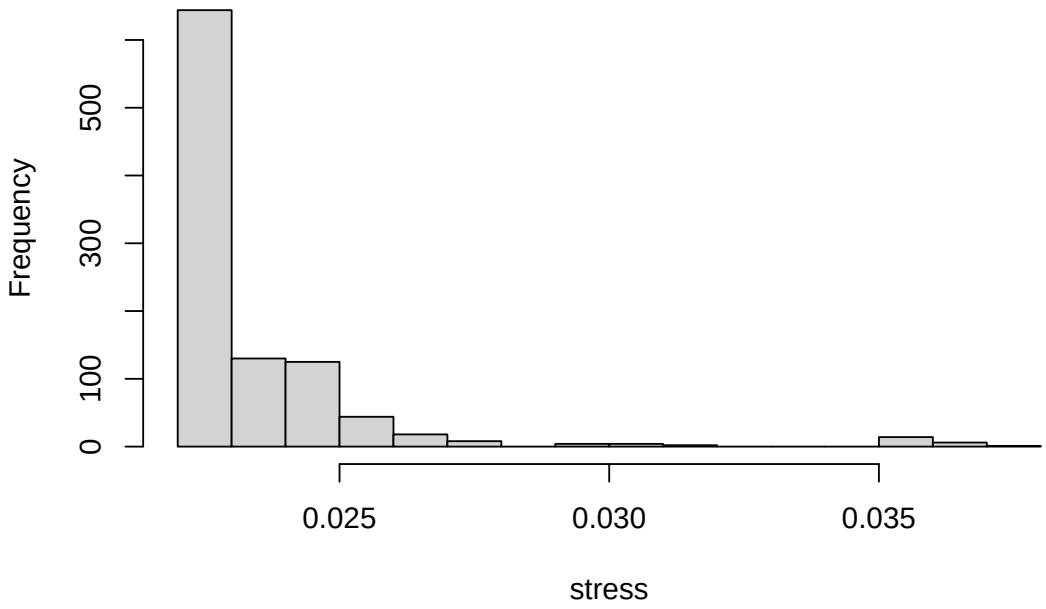


Figure 26.5: De Gruijter Stress Values

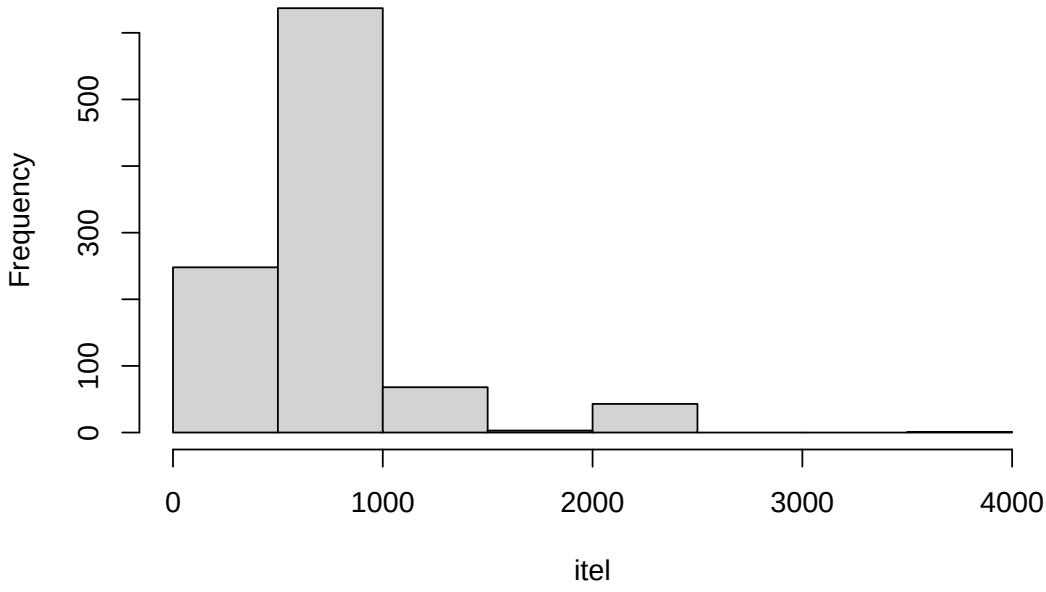


Figure 26.6: De Gruijter Iteration Count

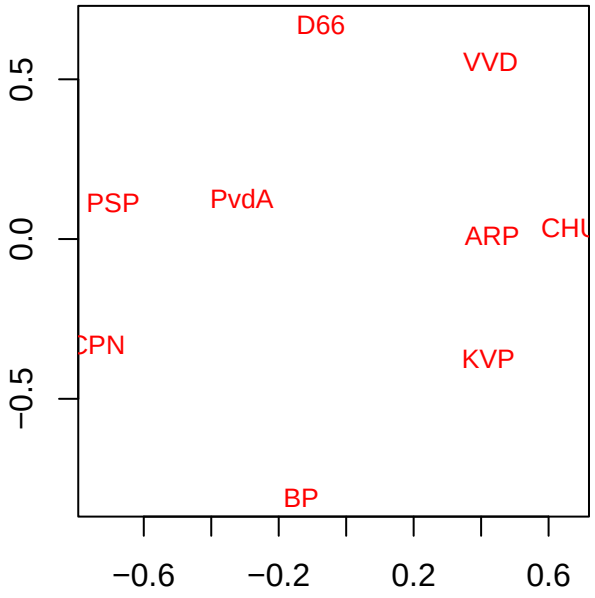


Figure 26.7: Largest Local Minimum

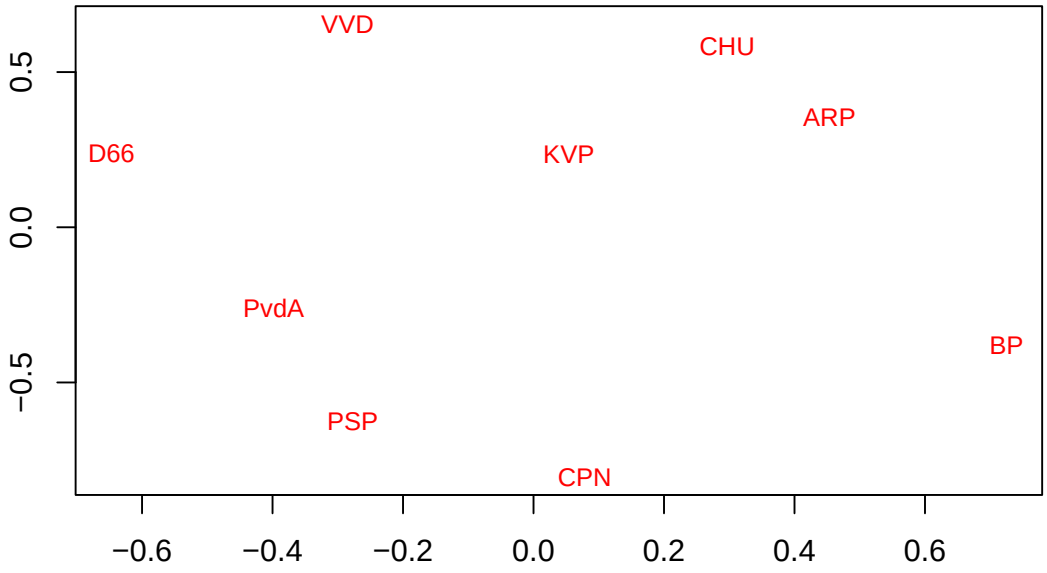


Figure 26.8: Another Local Minimum

26.2 Tunneling, Filling, Annealing, etc.

26.3 Cutting Planes

In cutting plane methods we approximate a non-polyhedral compact convex set $\mathcal{C}$ by a sequence $\mathcal{P}_n$ of convex polyhedra. Approximation is from the outside, i.e. $\mathcal{C} \subset \mathcal{P}_n$, strictly monotonic, i.e $\mathcal{P}_{n+1} \subset \mathcal{P}_n$, and convergent, i.e. $\lim_{n\rightarrow\infty} \mathcal{P}_\setminus = \mathcal{C}$. Under suitable conditions the maximum/minimum $f_n^\star$ of a function f on $\mathcal{P}_n$ will converge to $f^\star$, the maximum/minimum of f on $\mathcal{C}$.

If we are maximizing a convex function on $\mathcal{C}$ then the maximum on the approximating polyhedron $\mathcal{P}_n$ will be at one of the vertices $x^\star$. If $x^\star \notin \mathcal{C}$ we cut it off by finding a hyperplane that separates x and $\mathcal{C}$. We can, for example, project $x^\star$ on $\mathcal{C}$ and use the tangent hyperplane to $\mathcal{C}$ in the projection $\hat{x}$. Define $\mathcal{P}_{n+1} \cap \{x \mid a'x \leq b\}$, with $a'x = b$ the separating hyperplane that has $a'x \leq b$ for each $x \in \mathcal{C}$ and $a'x^\star > b$.

The basic MDS problem can be reformulated as maximization of $\rho(x)$ on the unit ball $\eta^2(x) = x'x \leq 1$. See sections 2.1.3 and 3.2 for the reformulation tools. The cutting plane method in the case of a ball is particulary simple, at least conceptually. If $x^\star \notin \mathcal{C}$ then its projection on the ball is $\hat{x} = x^\star/\|x^\star\|$ and the tangent hyperplane in $\hat{x}$ is $\hat{x}'x \leq 1$.

Computationally, however, matters are not so simple. The polyhedron $\mathcal{P}_n$ is described by an increasing number of linear inequalities. Finding all vertices requires a non-trivial effort, and in the general case the method seems practical only for small or moderately small examples. In an actual implementation we would have to have a scheme for dropping or not adding redundant inequalities (that are implied by earlier inequalities) and a scheme for dropping inequalities generating vertices that can never be the global maximum. Such strategfies will depend on the nature of the function f and on the convex set $\mathcal{S}$.

26.3.1 On the Circle

For the circle the cutting plane method can be made very simple. Suppose we start with a n distinct inner points $x_1, \cdots, x_n$ on the unit circle $\mathcal{S}$, ordered

clockwise so that x_1 is at the top of the circle (high noon). Let $\mathcal{Q}_n$ be their convex hull. Then $\mathcal{Q}_n \subset \mathcal{S}$. The tangent lines at x_i and x_{i+1} intersect outside the circle in an outer point y_i, with y_n the intersection of the tangents at x_n and x_1. Let $\mathcal{P}_n$ be the convex hull of the y_i. Then $\mathcal{Q}_n \subset \mathcal{S} \subset \mathcal{P}_n$ and thus

$$\max_{i=1}^{n} \rho(x_i) = \max_{x \in \mathcal{Q}_n} \rho(x) \leq \max_{x \in \mathcal{S}} \rho(x) \leq \max_{x \in \mathcal{P}_n} \rho(x) = \max_{i=1}^{n} \rho(y_i).$$

Thus we have easily computable lower and upper bounds for the global maximum of ρ on $\mathcal{S}$. The next step is to add the n projections $y_i/\|y_i\|$ to the n inner points to have a new set of $2n$ inner points. Then compute the corresponding $2n$ outer points, and so on. After k steps we have $2^k k_0$ inner and outer points, where k_0 is the number of inner points we started with.

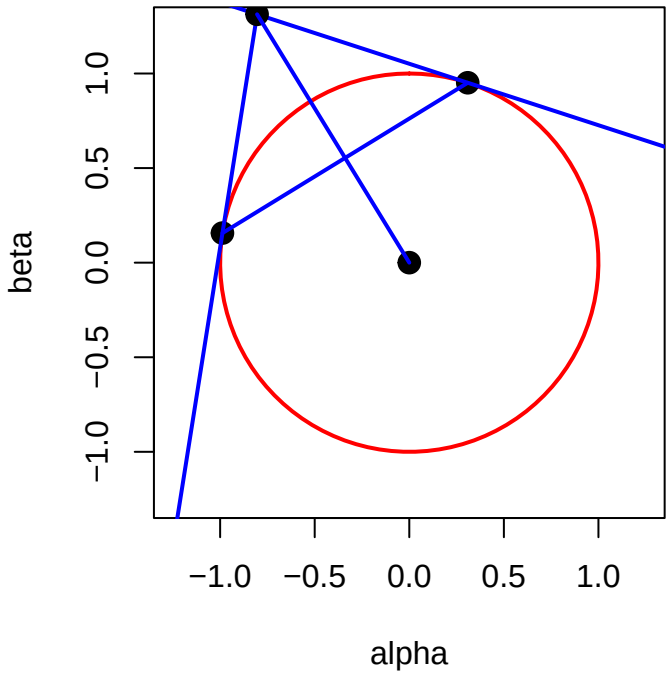


Figure 26.9: Circle Segment

As figure 26.9 shows, the outer point corresponding with two consecutive points on the circle lies on the perpendicular bisector of the line connecting the points. Using non-consecutive points will produce tangent lines which intersect farther away from the circle, and which consequently leads to a larger convex hull, and a worse approximation.

The next three figures illustrate the first iterations of the algorithm. We always start with n inner points equally distributed on the unit circle, in this case $n = 4$. The circle is in red, the convex hulls of the outer and inner points are in blue.

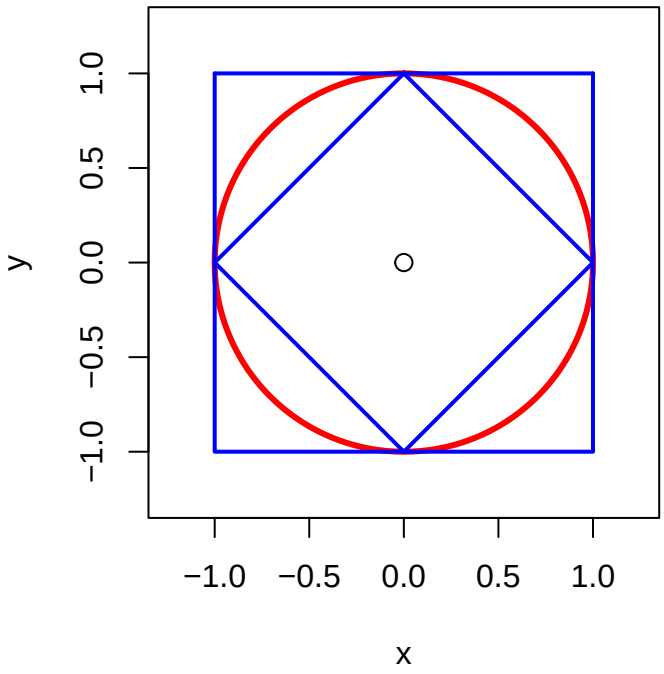


Figure 26.10: Starting Point

We see rapid convergence of the convex hulls to the circle. The figures also suggest an improvement of the method. Suppose ρ_0 is a lower bound of the global minimum $\rho_\star$ equal to the largest ρ value of the inner points. Suppose an outer point has a ρ value less than or equal to ρ_0, consider the triangle with the outer point and the two inner points. All three vertices have a ρ value less than or equal to ρ_0, and because ρ is convex so have all points in the triangle, including a segment of the circle. Thus we can *phantom* that segment of the circle and create no new inner points there. If ρ_0 get closer to $\rho_\star$ more and more segments of the circle are eliminated, which will presumably lead to faster computation. It seems advantageous to start with a value of ρ_0 that is as large as possible, for example by using the value the smacof algorithm converges to. We can then use the inner and outer points to check if the ρ value is a global minimum.

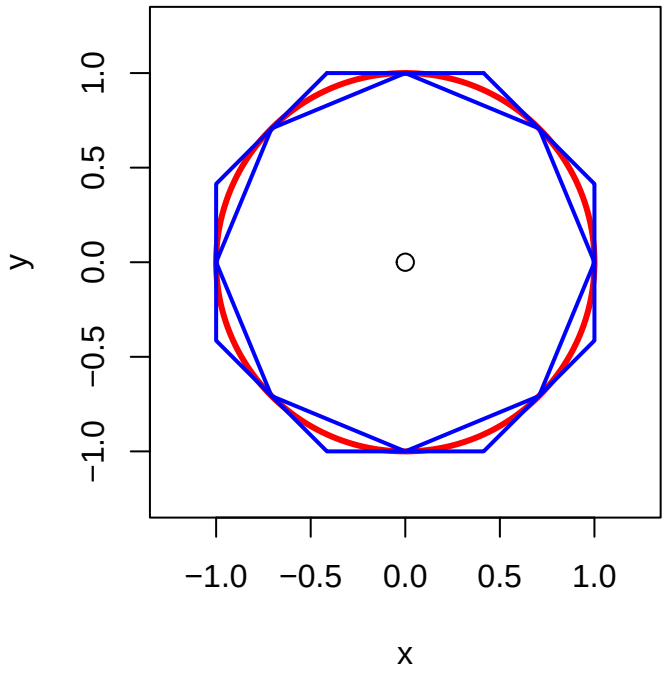


Figure 26.11: After First Iteration

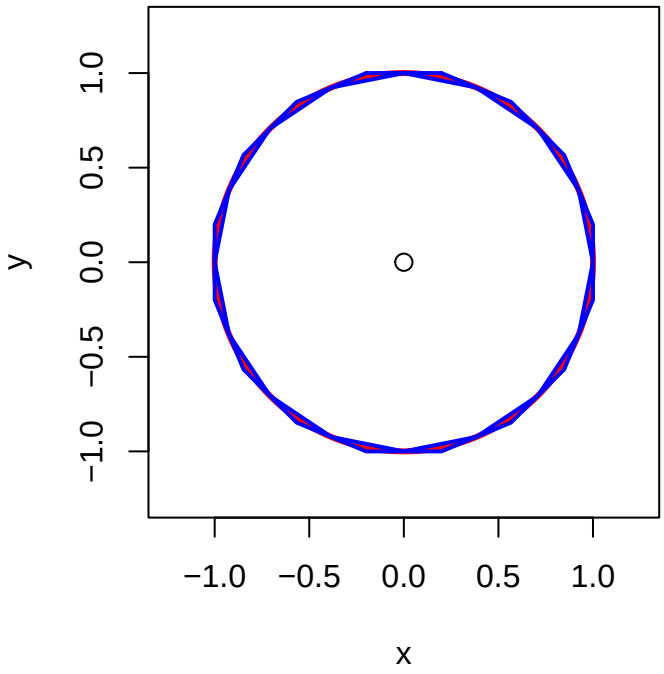


Figure 26.12: After Second Iteration

26.3.2 Cauchy Step Size

The standard smacof update of X (update method up_A of section 6.1) is $\gamma(X) = V^+B(X)X$. The relaxed update is $X(\lambda) := \lambda\gamma(X) + (1-\lambda)X$. We usually choose $\lambda = 2$, which gives update method up_B of section 6.1.

The Cauchy or steepest descent update is $X(\hat{\lambda})$, with

$$\hat{\lambda} = \underset{\lambda\geq 0}{\text{argmin}}\, \sigma(X(\lambda)). \tag{26.1}$$

There are some examples of the use of $\hat{\lambda}$ in De Leeuw and Heiser (1980), but there the optimal step-size is computed by using constrained smacof iterations, which may actually take us to just a local minimum along the line. In this example we use our circle methodology to compute the global minimum.

Now tr $X'V\gamma(X) = \rho(X)$ and thus if tr $X'VX = 1$ then $Y = \gamma(X) - \rho(X)X$ satisfies tr $X'VY = 0$ and $\eta^2(Y) = \eta^2(\gamma(X)) - \rho^2(X)$. Normalize Y such that tr $Y'VY = 1$, and now maximize ρ over α and β, i.e. in configuration space maximize $\rho(\alpha X + \beta Y)$, where $\alpha^2 + \beta^2 = 1$.

In the example we choose X to be a 10×2 matrix filled with random standard normals, and we start with 10 inner points on the circle. The iterations until convergence are as follows.

```
## itel       1 vertices      10 innermax       0.96683250 outermax       1.01451883
## itel       2 vertices      20 innermax       0.96683250 outermax       0.97744943
## itel       3 vertices      40 innermax       0.96683250 outermax       0.97008052
## itel       4 vertices      80 innermax       0.96709009 outermax       0.96794626
## itel       5 vertices     160 innermax       0.96720000 outermax       0.96739328
## itel       6 vertices     320 innermax       0.96720681 outermax       0.96726518
## itel       7 vertices     640 innermax       0.96721856 outermax       0.96722816
## itel       8 vertices    1280 innermax       0.96721856 outermax       0.96722140
## itel       9 vertices    2560 innermax       0.96721856 outermax       0.96721949
## itel      10 vertices    5120 innermax       0.96721876 outermax       0.96721890
## itel      11 vertices   10240 innermax       0.96721876 outermax       0.96721880
## itel      12 vertices   20480 innermax       0.96721876 outermax       0.96721877
## itel      13 vertices   40960 innermax       0.96721876 outermax       0.96721877
```

The optimal α and β are -0.5464817, -0.837471. In configuration space this translates to 1.7919402 X+-2.6770034 $\gamma(X)$.

26.3.3 Balls

In dimension $p > 2$, where ρ must be maximized on the unit ball in $\mathbb{R}^p$, matters are not so simple any more. There is no single compelling natural ordering of the points on the sphere or hypersphere, and thus we have to improvise more. We would like to maintain both upper and lower bounds for the global minimum that both keep improving in every iteration.

26.3.3.1 Outer Approximation

Let's first discuss a possible initial set of inner and outer points that are more or less regularly spaced inside or outside the unit sphere. For the inner points we can use the vertices of the cross-polytope (or the ℓ_1-ball), which is the set of all x in $\mathbb{R}^n$ with $\sum_{i=1}^n |x_i| \leq 1$. If $|x_i| \leq 1$ then $x_i^2 \leq |x_i|$ with equality iff $x_i \in \{-1, 0, 1\}$. Thus $x'x \leq \sum_{i=1}^n |x_i|$ and $x'x = \sum_{i=1}^n |x_i| = 1$ iff exactly one of x_i is ± 1, i.e. there are $2n$ inner points on the sphere.

For the outer points we choose the vertices of $\max_{i=1}^n |x_i| \leq 1$, or equivalently $-1 \leq x_i \leq +1$ for all i. Thus there are 2^n vertices which have $x_i = \pm 1$ for all i.

26.4 Distance Smoothing

$$d_{ij}(X, \epsilon) := \sqrt{d_{ij}^2(X) + \epsilon^2}$$

$$d_{ij}^\epsilon(X) := \sqrt{d_{ij}^2(X) + \epsilon^2}$$

$$\mathcal{D} d_{ij}^\epsilon(X) = \frac{1}{d_{ij}^\epsilon(X)} A_{ij} X$$

$$\nabla \sigma_\epsilon(X) = 2(V - B_\epsilon(X))X$$

$$\mathcal{D}^2 \sigma_\epsilon(X) =$$

Theorem 26.1. *If $\epsilon \geq \max_{i,j} \delta_{ij}$ then $B(X_\epsilon) \lesssim V$ and thus $\mathcal{D}^2(X_\epsilon) \gtrsim 0$ for all X and σ_ϵ is convex.*

$$\mathcal{D}_1 d_{ij}(X,\epsilon) = \frac{1}{d_{ij}(X,\epsilon)} A_{ij} X$$

$$\nabla \sigma_\epsilon(X) = 2\left(V - \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}(X_\epsilon)}A_{ij}\right)X$$

$$\nabla^2 \sigma_\epsilon(X) = 2V - \sum_{1\leq i<j\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}(X_\epsilon)}A_{ij}X$$

26.5 Penalizing Dimensions

In Shepard (1962a) and Shepard (1962b) an NMDS technique is developed that minimizes a loss function over configurations in full dimensionality $n-1$. In that sense the technique is similar to FDS. Shepard's iterative process, however, aims to maintain monotonicity between distances and dissimilarities and at the same time concentrate as much of the variation as possible in a small number of dimensions (De Leeuw (2017e)).

Let us explore the idea of concentrating variation in $p < n-1$ dimensions, but use an approach which is quite different from the one used by Shepard. We remain in the FDS framework, but we aim for solutions in $p < n-1$ dimensions by penalizing $n-p$ dimensions of the full configuration, using the classical Courant quadratic penalty function.

Partition a full configuration $Z = \begin{bmatrix} X & | & Y \end{bmatrix}$, with X of dimension $n \times p$ and Y of dimension $n \times (n-p)$. Then

$$\sigma(Z) = 1 - \mathbf{tr}\ X'B(Z)X - \mathbf{tr}\ Y'B(Z)Y + \frac{1}{2}\mathbf{tr}\ X'VX + \frac{1}{2}\mathbf{tr}\ Y'VY. \tag{26.2}$$

Also define the *penalty term*

$$\tau(Y) = \frac{1}{2}\mathbf{tr}\ Y'VY, \tag{26.3}$$

and *penalized stress*

$$\pi(Z,\lambda) = \sigma(Z) + \lambda\ \tau(Y). \tag{26.4}$$

Our proposed method is to minimize penalized stress over Z for a sequence of values $0 = \lambda_1 < \lambda_2 < \cdots \lambda_m$. For $\lambda = 0$ this is simply the FDS problem, for

which we know we can compute the global minimum. For fixed $0 < \lambda < +\infty$ this is a Penalized FDS or PFDS problem. PFDS problems with increasing values of λ generate a *trajectory* $Z(\lambda)$ in configuration space.

The general theory of exterior penalty functions, which we review in section XX of this paper, shows that increasing λ leads to an increasing sequence of stress values σ and a decreasing sequence of penalty terms τ. If $\lambda \to +\infty$ we approximate the global minimum of the FDS problem with Z of the form $Z = \begin{bmatrix} X & | & 0 \end{bmatrix}$, i.e. of the pMDS problem. This assumes we do actually compute the global minimum for each value of λ, which we hope we can do because we start at the FDS global minimum, and we slowly increase λ. There is also a local version of the exterior penalty result, which implies that $\lambda \to \infty$ takes us to a local minimum of pMDS, so there is always the possibility of taking the wrong trajectory to a local minimum of pMDS.

26.5.1 Local Minima

The stationary equations of the PFDS problem are solutions to the equations

$$(V - B(Z))X = 0, \tag{26.5}$$
$$((1+\lambda)V - B(Z))Y = 0. \tag{26.6}$$

We can easily related stationary points and local minima of the FDS and PFDS problem.

Theorem 26.2. *1: If X is a stationary point of the pMDS problem then $Z = [X \mid 0]$ is a stationary point of the PFDS problem, no matter what λ is.*

2: If $Z = [X \mid 0]$ is a local minimum of the PFDS problem then X is a local minimum of pMDS and $(1+\lambda)V - B(X) \gtrsim 0$, or $\lambda \geq \|V^+ B(X)\|_\infty - 1$, with $\|\bullet\|_\infty$ the spectral radius (largest eigenvalue).

Proof. Part 1 follows by simple substitution in the stationary equations.

Part 2 follows from the expansion for $Z = [X + \epsilon P \mid \epsilon Q]$.

$$\begin{aligned}\pi(Z) = \pi(X) + \epsilon \text{ tr } P'\mathcal{D}\sigma(X) + \\ + \frac{1}{2}\epsilon^2 \mathcal{D}^2\sigma(X)(P,P) + \frac{1}{2}\epsilon^2 \text{ tr } Q'((1+\lambda)V - B(X))Q + o(\epsilon^2).\end{aligned} \tag{26.7}$$

At a local minimum we must have $\mathcal{D}\sigma(X) = 0$ and $\mathcal{D}^2\sigma(X)(P,P) \gtrsim 0$, which are the necessary conditions for a local minimum of pMDS. We also must have $((1+\lambda)V - B(X)) \gtrsim 0$. □

Note that the conditions in part 2 of theorem 26.2 are also sufficient for PFDS to have a local minimum at $[X \mid 0]$, provided we eliminate translational and rotational indeterminacy by a suitable reparametrization, as in De Leeuw (1993).

26.5.2 Algorithm

The smacof algorithm for penalized stress is a small modification of the unpenalized FDS algorithm (ref). We start our iterations for λ_j with the solution for λ_{j-1} (the starting solution for $\lambda_1 = 0$ can be completely arbitrary). The update rules for fixed λ are

$$X^{(k+1)} = V^+ B(Z^{(k)}) X^{(k)}, \tag{26.8}$$

$$Y^{(k+1)} = \frac{1}{1+\lambda} V^+ B(Z^{(k)}) Y^{(k)}. \tag{26.9}$$

Thus we compute the FDS update $Z^{(k+1)} = V^+ B(Z^{(k)}) Z^{(k)}$ and then divide the last $n-p$ columns by $1+\lambda$.

Code is in the appendix. Let us analyze a number of examples.

26.5.3 Examples

This section has a number of two-dimensional and a number of one-dimensional examples. The one-dimensional examples are of interest, because of the documented large number of local minima of stress in the one-dimensional case, and the fact that for small and medium n exact solutions are available (for example, De Leeuw (2005b)). By default we use `seq(0, 1, length = 101)` for λ in most examples, but for some of them we dig a bit deeper and use longer sequences with smaller increments.

If for some value of λ the penalty term drops below the small cutoff γ, for example 10^{-10}, then there is not need to try larger values of λ, because they

will just repeat the same result. We hope that result is the global minimum of the 2MDS problem.

The output for each example is a table in which we give, the minimum value of stress, the value of the penalty term at the minimum, the value of λ, and the number of iterations needed for convergence. Typically we print for the first three, the last three, and some regularly spaced intermediate values of λ. Remember that the stress values increase with increasing λ, and the penalty values decrease.

For two-dimensional examples we plot all two-dimensional configurations, after rotating to optimum match (using the function `matchMe()` from the appendix). We connect corresponding points for different values of λ. Points corresponding to the highest value of λ are labeled and have a different plot symbol. For one-dimensional examples we put `1:n` on the horizontal axes and plot the single dimension on the vertical axis, again connecting corresponding points. We label the points corresponding with the highest value of λ, and draw horizontal lines through them to more clearly show their order on the dimension.

The appendix also has code for the function `checkUni()`, which we have used to check the solutions in the one dimensional case are indeed local minima. The function checks the necessary condition for a local minimum $x = V^+u$, with

$$u_i = \sum_{j=1}^{n} w_{ij}\delta_{ij} \ \textbf{sign}\ (x_i - x_j).$$

It should be emphasized that all examples are just meant to study convergence of penalized FDS. There is no interpretation of the MDS results

26.5.3.1 Chi Squares

In this example, of order 10, the δ_{ij} are independent draws from a chi-square distribution with two degrees of freedom. There is no structure in this example, everything is random.

```
## itel  198 lambda   0.000000 stress 0.175144 penalty 0.321138
## itel    5 lambda   0.010000 stress 0.175156 penalty 0.027580
## itel    3 lambda   0.020000 stress 0.175187 penalty 0.025895
```

```
## itel    1 lambda   0.100000 stress 0.175914 penalty 0.015172
## itel    1 lambda   0.200000 stress 0.177666 penalty 0.004941
## itel    4 lambda   0.300000 stress 0.178912 penalty 0.000088
## itel    6 lambda   0.310000 stress 0.178933 penalty 0.000020
## itel   20 lambda   0.320000 stress 0.178939 penalty 0.000000
```

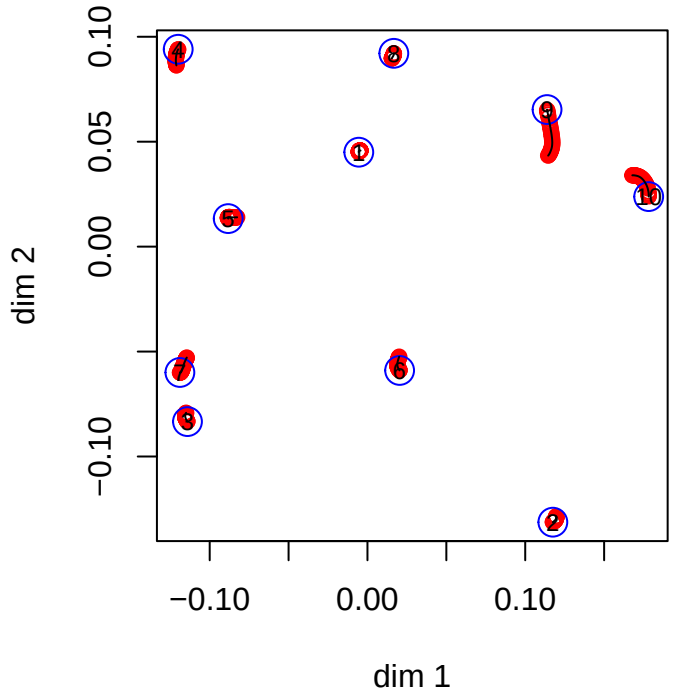


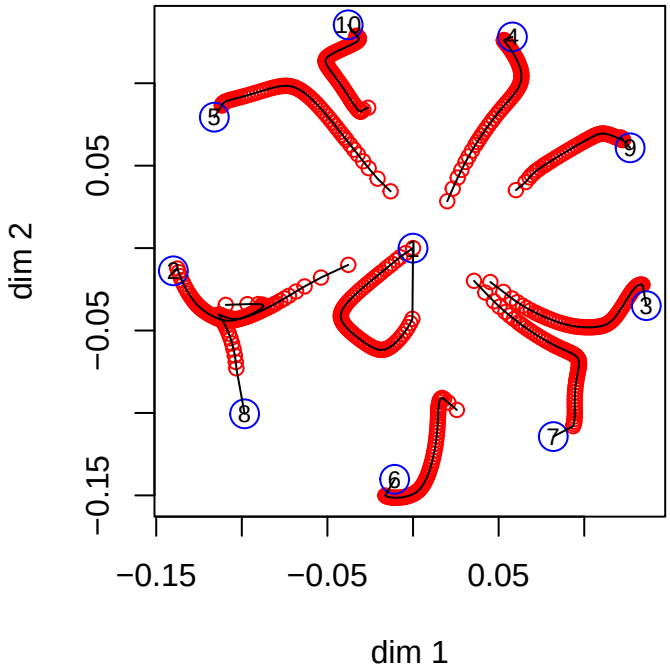
Figure 26.13: 10 Chi Squares

It seems that in this example the first two dimensions of FDS are already close to optimal for 2MDS. This is because the Gower rank of the dissimilarities is only three (or maybe four, the fourth singular value of the FDS solution Z is very small).

26.5.3.2 Regular Simplex

The regular simplex has all dissimilarities equal to one. We use an example with $n = 10$, for which the global minimum (as far as we know) of pMDS with $p = 2$ is a configuration with nine points equally spaced on a circle and one point in the center.

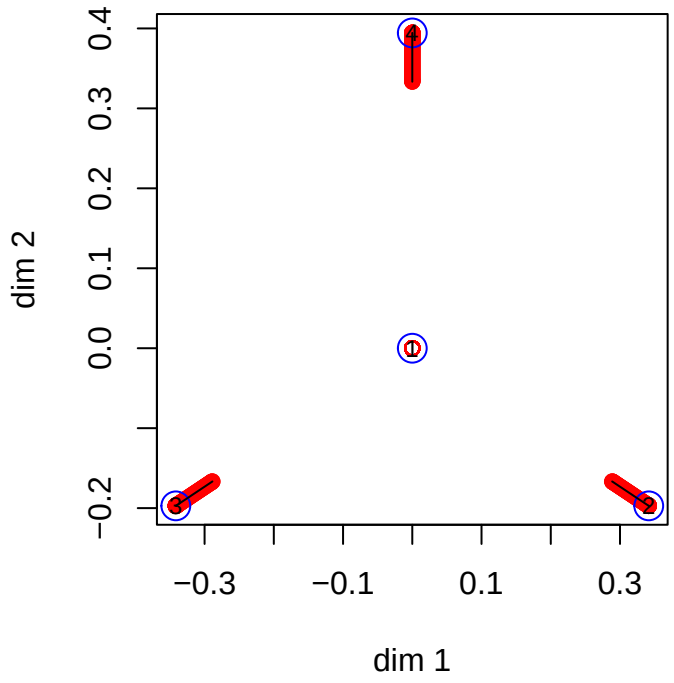
```
## itel    1 lambda   0.000000 stress 0.000000 penalty 0.400000
## itel    7 lambda   0.010000 stress 0.000101 penalty 0.375653
## itel    5 lambda   0.020000 stress 0.000422 penalty 0.360483
## itel    1 lambda   0.100000 stress 0.008849 penalty 0.258090
## itel    1 lambda   0.200000 stress 0.032243 penalty 0.149427
## itel    1 lambda   0.300000 stress 0.059088 penalty 0.079592
## itel    1 lambda   0.400000 stress 0.079534 penalty 0.043250
## itel    1 lambda   0.500000 stress 0.095361 penalty 0.020895
## itel    1 lambda   0.600000 stress 0.105667 penalty 0.006921
## itel    1 lambda   0.610000 stress 0.106337 penalty 0.005862
## itel    1 lambda   0.620000 stress 0.106951 penalty 0.004879
## itel  105 lambda   0.630000 stress 0.109880 penalty 0.000000
```



Next, we look at the regular simplex with $n = 4$, for which the global minimum has four points equally spaced on a circle (i.e. in the corners of a square). We use `seq(0, 1, length = 101)` for the λ sequence.

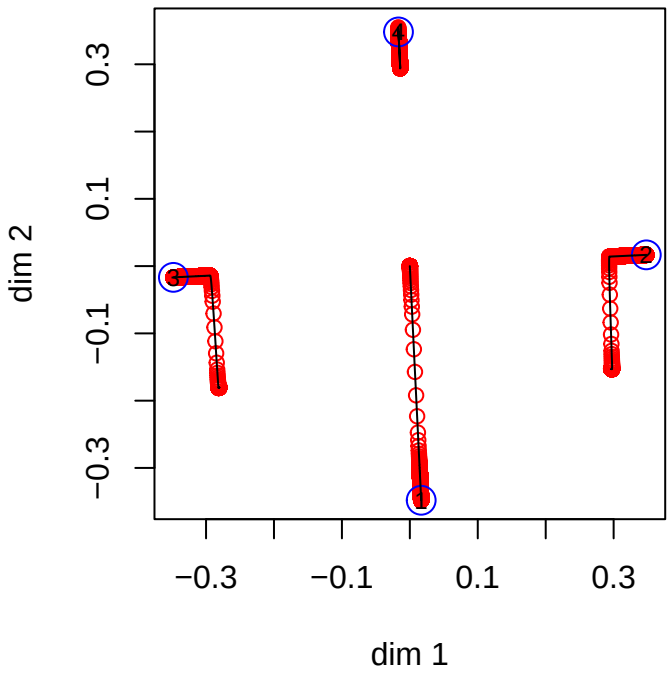
```
## itel    1 lambda   0.000000 stress 0.000000 penalty 0.250000
## itel    2 lambda   0.010000 stress 0.000035 penalty 0.162295
```

```
## itel    1 lambda   0.020000 stress 0.000122 penalty 0.158591
## itel    1 lambda   0.100000 stress 0.003808 penalty 0.122344
## itel    1 lambda   0.200000 stress 0.014089 penalty 0.084111
## itel    1 lambda   0.300000 stress 0.028044 penalty 0.053366
## itel    1 lambda   0.400000 stress 0.043331 penalty 0.028965
## itel    1 lambda   0.500000 stress 0.056851 penalty 0.011482
## itel    1 lambda   0.600000 stress 0.064718 penalty 0.002471
## itel    1 lambda   0.700000 stress 0.066799 penalty 0.000203
## itel    1 lambda   0.800000 stress 0.066982 penalty 0.000005
## itel    1 lambda   0.820000 stress 0.066985 penalty 0.000002
## itel    1 lambda   0.830000 stress 0.066986 penalty 0.000001
## itel    1 lambda   0.840000 stress 0.066986 penalty 0.000001
```



The solution converges to an equilateral triangle with the fourth point in the centroid. This is a local minimum. What basically happens is that the first two dimensions of the FDS solution are too close to the local minimum. Or, what amounts to the same thing, the Gower rank is too large (it is $n-1$ for a regular simplex) , there is too much variation in the higher dimensions, and as a consequence the first two dimensions of FDS are a bad 2MDS solution. We try to repair this by refining the trajectory, using `seq(0, 1, 10001)`.

```
## itel    1 lambda   0.000000 stress 0.000000 penalty 0.250000
## itel    2 lambda   0.000100 stress 0.000000 penalty 0.166622
## itel    1 lambda   0.000200 stress 0.000000 penalty 0.166583
## itel    1 lambda   0.202200 stress 0.028595 penalty 0.000001
## itel    1 lambda   0.202300 stress 0.028595 penalty 0.000001
## itel    1 lambda   0.202400 stress 0.028595 penalty 0.000001
```

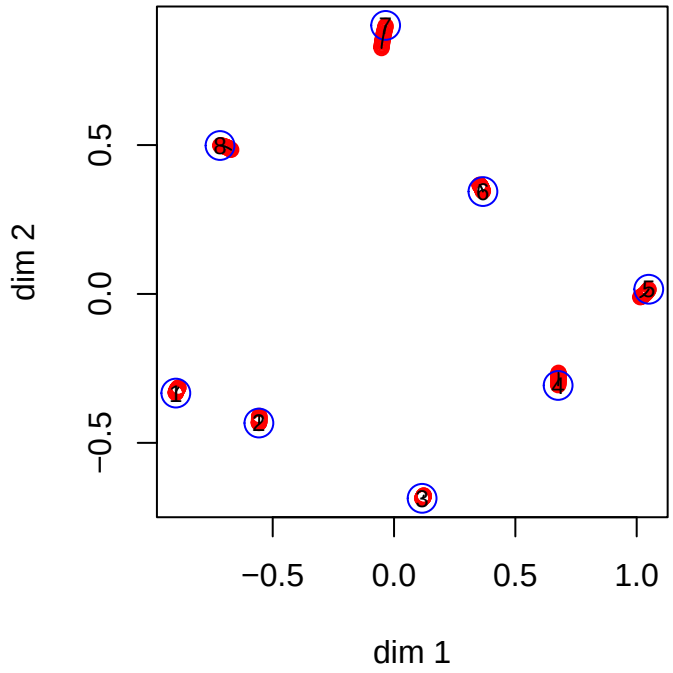


Now the trajectories move us from what starts out similar to an equilateral triangle to the corners of the square, and thus we do find the global minimum in this way. It is remarkable that we manage to find the square even when we start closer to the triangle with midpoint.

26.5.3.3 Intelligence

These are correlations between eight intelligence tests, taken from the smacof package. We convert to dissimilarities by taking the negative logarithm of the correlations. As in the chi-square example, the FDS and the 2MDS solution are very similar and the PMDS trajectories are short.

```
## itel 2951 lambda   0.000000 stress 0.107184 penalty 7.988384
## itel    7 lambda   0.010000 stress 0.107560 penalty 0.685654
## itel    4 lambda   0.020000 stress 0.108528 penalty 0.628538
## itel    3 lambda   0.030000 stress 0.110045 penalty 0.573208
## itel    3 lambda   0.040000 stress 0.112449 penalty 0.510730
## itel    2 lambda   0.050000 stress 0.114714 penalty 0.464650
## itel    2 lambda   0.060000 stress 0.117623 penalty 0.415037
## itel    2 lambda   0.070000 stress 0.121095 penalty 0.364536
## itel    2 lambda   0.080000 stress 0.125010 penalty 0.315023
## itel    2 lambda   0.090000 stress 0.129226 penalty 0.267831
## itel    2 lambda   0.100000 stress 0.133589 penalty 0.223898
## itel    2 lambda   0.110000 stress 0.137944 penalty 0.183868
## itel    3 lambda   0.120000 stress 0.143921 penalty 0.133739
## itel    2 lambda   0.130000 stress 0.147473 penalty 0.106166
## itel    4 lambda   0.140000 stress 0.153215 penalty 0.064499
## itel    4 lambda   0.150000 stress 0.157159 penalty 0.037735
## itel    9 lambda   0.160000 stress 0.161434 penalty 0.010337
## itel   72 lambda   0.170000 stress 0.163122 penalty 0.000000
```



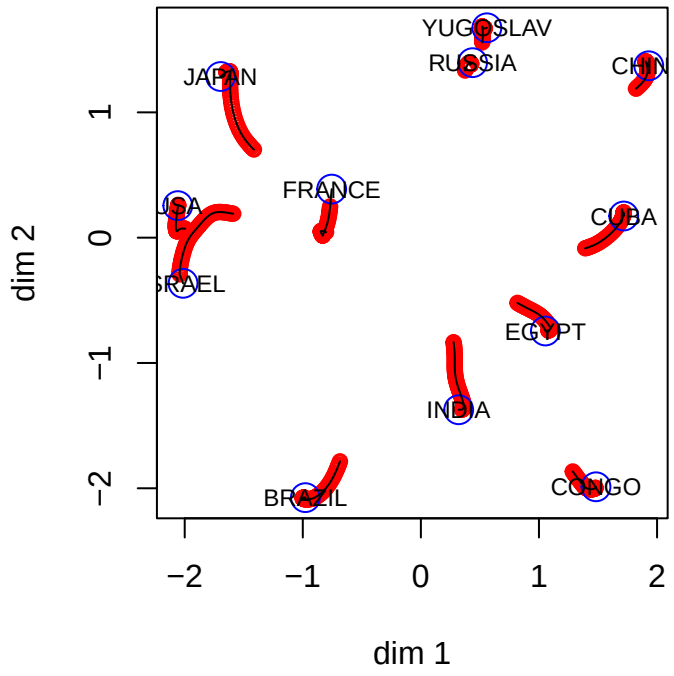
The singular values of the FDS solution are 1.78e+00, 1.36e+00, 5.94e-01,

1.47e-01, 3.29e-03, 2.39e-16, 1.15e-16, 9.35e-18, which shows that the Gower rank is probably five, but approximately two.

26.5.3.4 Countries

This is the wish dataset from the smacof package, with similarities between 12 countries. They are converted to dissimilarties by subtracting each of them from seven.

```
## itel 1381 lambda   0.000000 stress 4.290534 penalty 98.617909
## itel    4 lambda   0.010000 stress 4.301341 penalty 36.137074
## itel    3 lambda   0.020000 stress 4.336243 penalty 34.389851
## itel    1 lambda   0.100000 stress 5.187917 penalty 23.300775
## itel    1 lambda   0.200000 stress 7.539228 penalty 11.543635
## itel    1 lambda   0.300000 stress 9.901995 penalty 4.963372
## itel    1 lambda   0.400000 stress 11.523357 penalty 1.859569
## itel    1 lambda   0.500000 stress 12.391692 penalty 0.556411
## itel    1 lambda   0.590000 stress 12.696493 penalty 0.080144
## itel    1 lambda   0.600000 stress 12.708627 penalty 0.060113
## itel  100 lambda   0.610000 stress 12.738355 penalty 0.000000
```



The singular values of the FDS solution are 4.20e+00, 3.71e+00, 2.67e+00, 1.80e+00, 1.33e+00, 6.64e-01, 5.97e-04, 6.75e-16, 4.70e-16, 2.38e-16, 1.14e-16, 1.98e-17, and the Gower rank is six or seven.

26.5.3.5 Dutch Political Parties

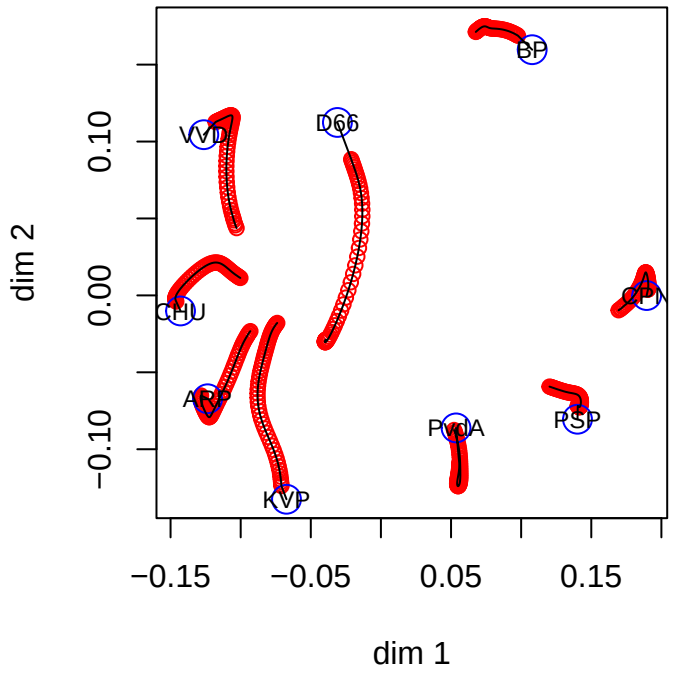
In 1967 one hundred psychology students at Leiden University judged the similarity of nine Dutch political parties, using the complete method of triads (De Gruijter (1967)). Data were aggregated and converted to dissimilarities. We first print the matrix of dissimilarities.

```
##       KVP    PvdA   VVD    ARP    CHU    CPN    PSP    BP     D66
## KVP  +0.000 +0.209 +0.196 +0.171 +0.179 +0.281 +0.250 +0.267 +0.230
## PvdA +0.209 +0.000 +0.250 +0.210 +0.231 +0.190 +0.171 +0.269 +0.204
## VVD  +0.196 +0.250 +0.000 +0.203 +0.185 +0.302 +0.281 +0.257 +0.174
## ARP  +0.171 +0.210 +0.203 +0.000 +0.119 +0.292 +0.250 +0.271 +0.228
## CHU  +0.179 +0.231 +0.185 +0.119 +0.000 +0.290 +0.263 +0.259 +0.225
## CPN  +0.281 +0.190 +0.302 +0.292 +0.290 +0.000 +0.152 +0.236 +0.276
```

```
## PSP  +0.250 +0.171 +0.281 +0.250 +0.263 +0.152 +0.000 +0.256 +0.237
## BP   +0.267 +0.269 +0.257 +0.271 +0.259 +0.236 +0.256 +0.000 +0.274
## D66  +0.230 +0.204 +0.174 +0.228 +0.225 +0.276 +0.237 +0.274 +0.000
```

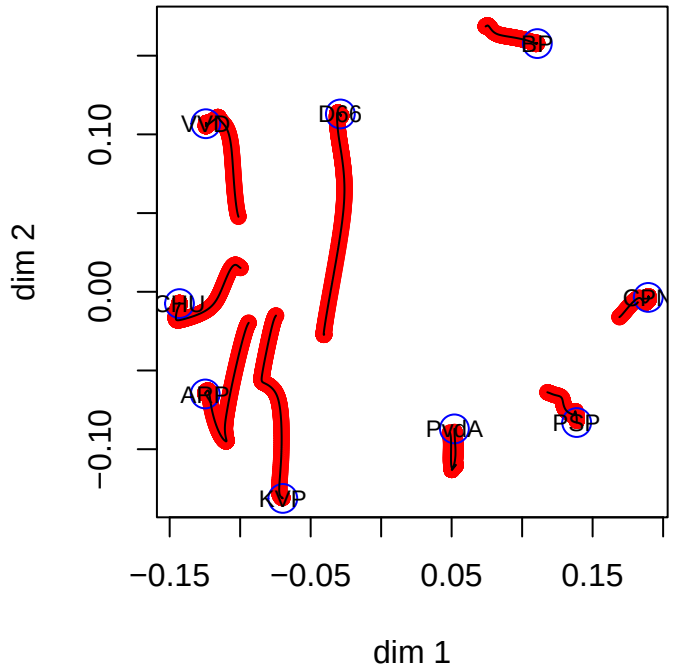
The trajectories from FDS to 2MDS show some clear movement, especially of the D'66 party, which was new at the time.

```
## itel  223 lambda   0.000000 stress 0.000000 penalty 0.414526
## itel    5 lambda   0.010000 stress 0.000061 penalty 0.196788
## itel    2 lambda   0.020000 stress 0.000199 penalty 0.190472
## itel    1 lambda   0.100000 stress 0.004399 penalty 0.136576
## itel    1 lambda   0.200000 stress 0.015811 penalty 0.075466
## itel    1 lambda   0.300000 stress 0.028235 penalty 0.036636
## itel    1 lambda   0.400000 stress 0.038275 penalty 0.012608
## itel    1 lambda   0.500000 stress 0.043644 penalty 0.002156
## itel    1 lambda   0.520000 stress 0.044091 penalty 0.001324
## itel    1 lambda   0.530000 stress 0.044253 penalty 0.001019
## itel  277 lambda   0.540000 stress 0.044603 penalty 0.000000
```



There seems to be some bifurcation going on at the end, so we repeat the analysis using `seq(0, 1, length = 1001)` for λ. The results turn out to be basically the same.

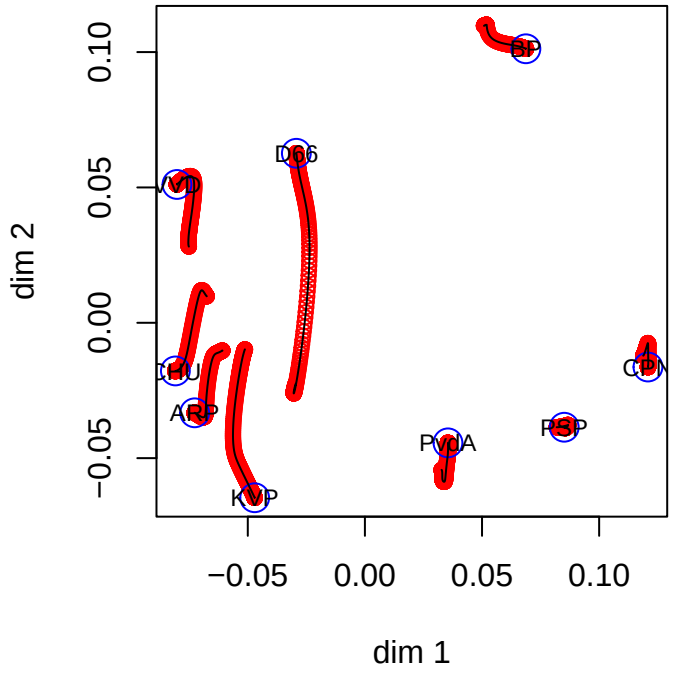
```
## itel  223 lambda   0.000000 stress 0.000000 penalty 0.414526
## itel    4 lambda   0.001000 stress 0.000001 penalty 0.204225
## itel    2 lambda   0.002000 stress 0.000002 penalty 0.203535
## itel    1 lambda   0.468000 stress 0.044604 penalty 0.000001
## itel    1 lambda   0.469000 stress 0.044604 penalty 0.000000
## itel  166 lambda   0.470000 stress 0.044603 penalty 0.000000
```



The singular values of the FDS solution are 2.95e-01, 2.10e-01, 1.89e-01, 1.34e-01, 1.16e-01, 1.06e-01, 8.61e-02, 7.06e-02, 3.98e-18, and the Gower rank is probably eight. This is mainly because these data, being averages, regress to the mean and thus have a substantial additive constant. If we repeat the analysis after subtracting .1 from all dissimilarities we get basically the same solution, but with somewhat smoother trajectories.

```
## itel  511 lambda   0.000000 stress 0.000176 penalty 0.150789
```

```
## itel    2 lambda   0.001000 stress 0.000176 penalty 0.037759
## itel    2 lambda   0.002000 stress 0.000176 penalty 0.037619
## itel    1 lambda   0.370000 stress 0.007642 penalty 0.000000
## itel    1 lambda   0.371000 stress 0.007642 penalty 0.000000
## itel    1 lambda   0.372000 stress 0.007642 penalty 0.000000
```



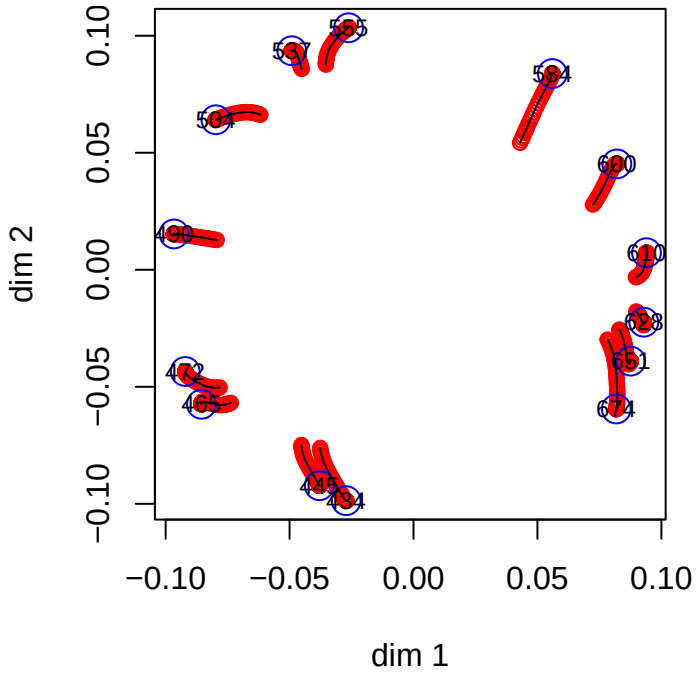
Now the singular values of the FDS solution are 2.05e-01, 1.34e-01, 1.11e-01, 6.03e-02, 3.11e-02, 3.97e-04, 1.18e-07, 4.55e-12, 8.75e-18, and the approximate Gower rank is more like five or six.

26.5.3.6 Ekman

The next example analyzes dissimilarities between 14 colors, taken from Ekman (1954). The original similarities s_{ij}, averaged over 31 subjects, were transformed to dissimilarities by $\delta_{ij} = 1 - s_{ij}$.

```
## itel 1482 lambda   0.000000 stress 0.000088 penalty 0.426110
## itel    5 lambda   0.010000 stress 0.000132 penalty 0.118988
```

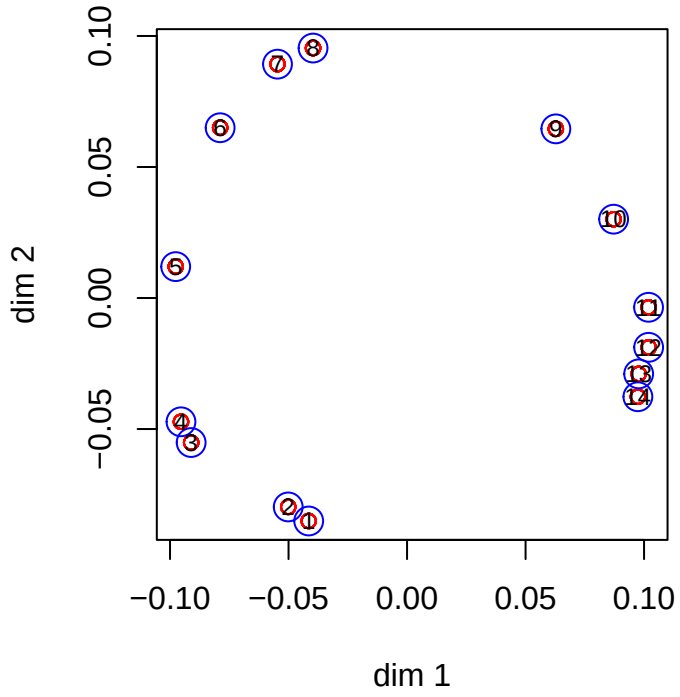
```
## itel    3 lambda   0.020000 stress 0.000253 penalty 0.112777
## itel    1 lambda   0.100000 stress 0.003195 penalty 0.070791
## itel    1 lambda   0.200000 stress 0.010778 penalty 0.024407
## itel    1 lambda   0.300000 stress 0.016125 penalty 0.003230
## itel    1 lambda   0.400000 stress 0.017142 penalty 0.000165
## itel    4 lambda   0.500000 stress 0.017213 penalty 0.000000
## itel    1 lambda   0.600000 stress 0.017213 penalty 0.000000
## itel    1 lambda   0.610000 stress 0.017213 penalty 0.000000
## itel    1 lambda   0.620000 stress 0.017213 penalty 0.000000
## itel    1 lambda   0.630000 stress 0.017213 penalty 0.000000
```



If we transform the Ekman similarities by $\delta_{ij} = (1 - s_{ij})^3$ then its is known (De Leeuw (2016b)) that the Gower rank is equal to two. Thus the FDS solution has rank 2, and the 2MDS solution is the global minimum.

```
## itel   99 lambda   0.000000 stress 0.011025 penalty 0.433456
## itel    1 lambda   0.010000 stress 0.011025 penalty 0.000000
## itel    1 lambda   0.020000 stress 0.011025 penalty 0.000000
## itel    1 lambda   0.100000 stress 0.011025 penalty 0.000000
```

```
## itel    1 lambda   0.110000 stress 0.011025 penalty 0.000000
## itel    1 lambda   0.120000 stress 0.011025 penalty 0.000000
## itel    1 lambda   0.130000 stress 0.011025 penalty 0.000000
```

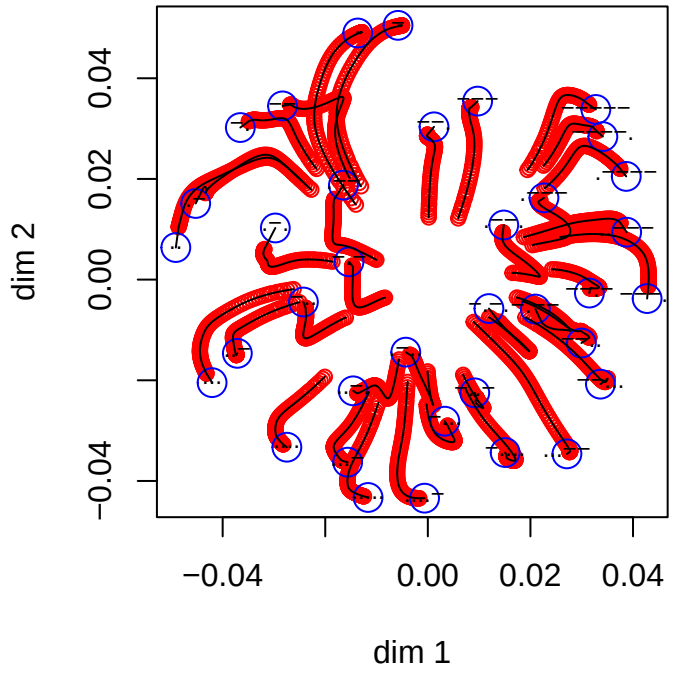


26.5.3.7 Morse in Two

Next, we use dissimilarities between 36 Morse code signals (Rothkopf (1957)). We used the symmetrized version `morse` from the `smacof` package (De Leeuw and Mair (2009)).

```
## itel 1461 lambda   0.000000 stress 0.000763 penalty 0.472254
## itel    6 lambda   0.010000 stress 0.000858 penalty 0.322181
## itel    4 lambda   0.020000 stress 0.001147 penalty 0.308335
## itel    1 lambda   0.100000 stress 0.008576 penalty 0.216089
## itel    1 lambda   0.200000 stress 0.028903 penalty 0.119364
## itel    1 lambda   0.300000 stress 0.051285 penalty 0.060060
## itel    1 lambda   0.400000 stress 0.068653 penalty 0.028190
## itel    1 lambda   0.500000 stress 0.080258 penalty 0.011356
```

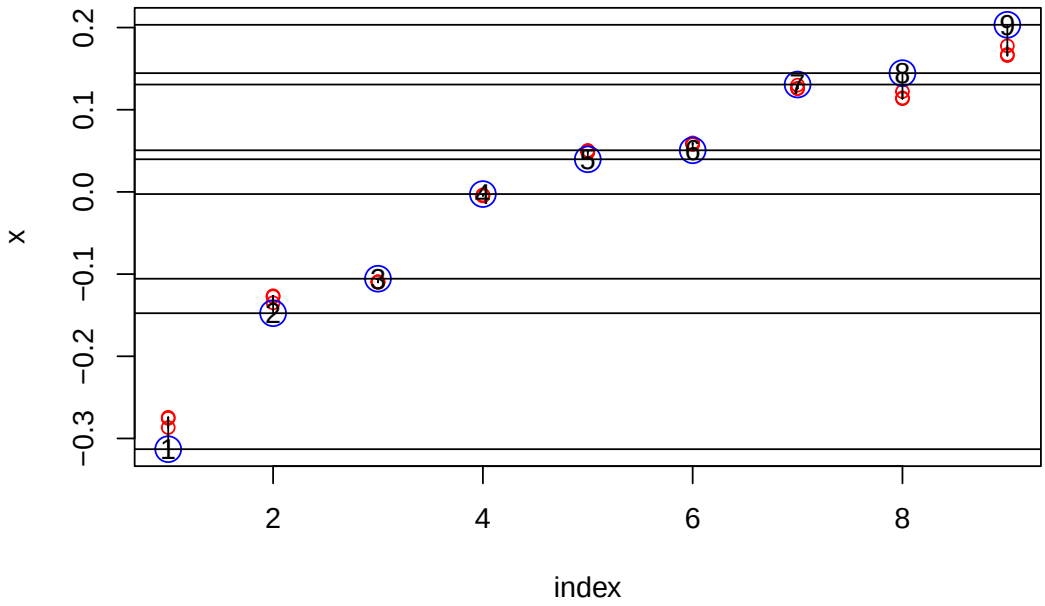
```
## itel    1 lambda   0.600000 stress 0.086572 penalty 0.003578
## itel    1 lambda   0.700000 stress 0.089140 penalty 0.000854
## itel    1 lambda   0.800000 stress 0.089898 penalty 0.000116
## itel    1 lambda   0.830000 stress 0.089958 penalty 0.000053
## itel    1 lambda   0.840000 stress 0.089970 penalty 0.000040
## itel  197 lambda   0.850000 stress 0.089949 penalty 0.000000
```



26.5.3.8 Vegetables

Our first one-dimensional example uses paired comparisons of 9 vegetables, originating with Guilford (1954). The proportions are transformed to dissimilarities by using the absolute values of the normal quantile function, i.e. $\delta_{ij} = |\Phi^{-1}(p_{ij})|$. We use a short sequence for λ.

```
## itel 1412 lambda   0.000000 stress 0.013675 penalty 0.269308
## itel    5 lambda   0.010000 stress 0.013716 penalty 0.114786
## itel    5 lambda   0.100000 stress 0.016719 penalty 0.069309
## itel   23 lambda   1.000000 stress 0.035301 penalty 0.000000
```

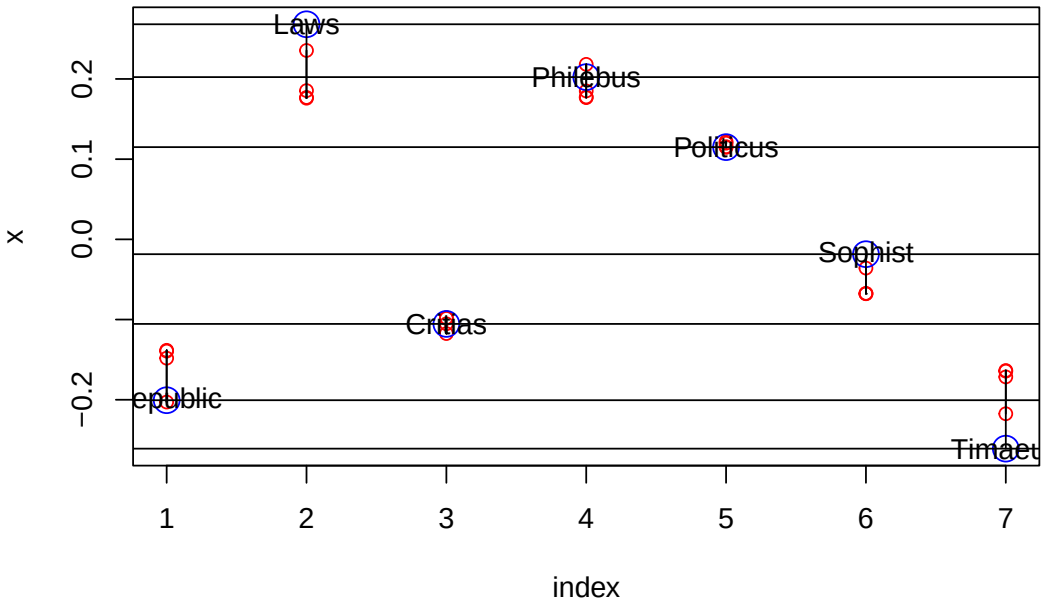


This example was previously analyzed by me in De Leeuw (2005b) using enumeration of all permutations. I found 14354 isolated local minima, and a global minimum equal to the one we computed here.

26.5.3.9 Plato

Mair, Groenen, and De Leeuw (2019) use seriation of the works of Plato, from the data collected by D. R. Cox and Brandwood (1959), as an example of unidimensional scaling. We first run this example with our usual sequence of five λ values.

```
## itel  169 lambda   0.000000 stress 0.000000 penalty 0.410927
## itel    3 lambda   0.010000 stress 0.000062 penalty 0.255246
## itel    3 lambda   0.100000 stress 0.005117 penalty 0.194993
## itel    4 lambda   1.000000 stress 0.106058 penalty 0.019675
## itel    9 lambda  10.000000 stress 0.139462 penalty 0.000000
```



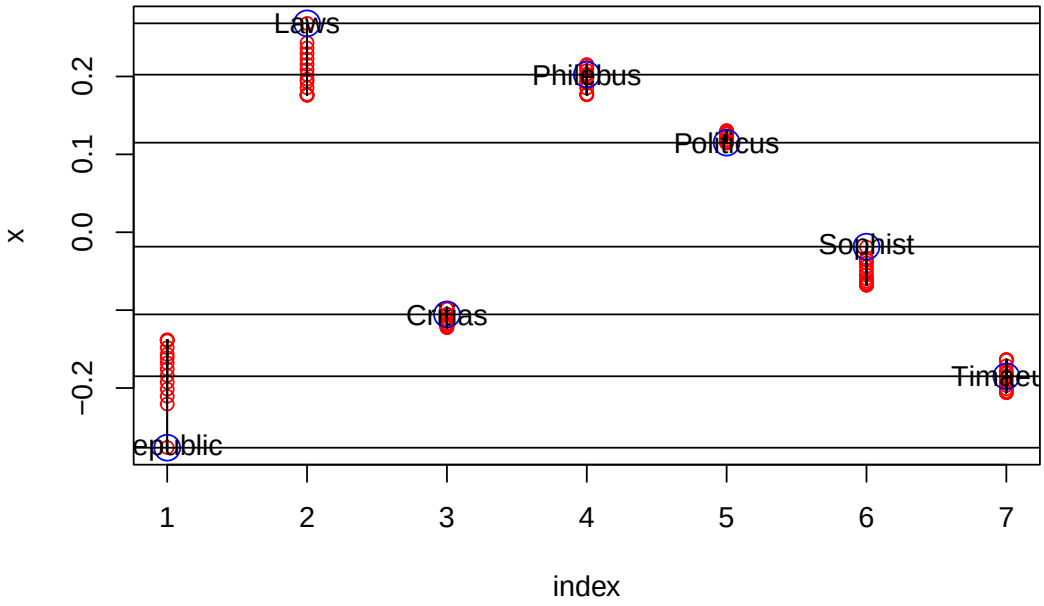
This gives the order

```
##      [,1]
## [1,] "Timaeus"
## [2,] "Republic"
## [3,] "Critias"
## [4,] "Sophist"
## [5,] "Politicus"
## [6,] "Philebus"
## [7,] "Laws"
```

which is different from the order at the global minimum that has Republic before Timaeus. Thus we have recovered a local minimum, and it seems our sequence of λ values was not fine enough to do the job properly. So we try a longer and finer sequence.

```
## itel  169 lambda   0.000000 stress 0.000000 penalty 0.410927
## itel    3 lambda   0.000100 stress 0.000000 penalty 0.263015
## itel    3 lambda   0.001000 stress 0.000001 penalty 0.262280
## itel    3 lambda   0.010000 stress 0.000064 penalty 0.255078
```

```
## itel    3 lambda   0.100000 stress 0.005123 penalty 0.194945
## itel    2 lambda   0.200000 stress 0.016184 penalty 0.147493
## itel    1 lambda   0.300000 stress 0.026997 penalty 0.119323
## itel    1 lambda   0.400000 stress 0.040023 penalty 0.093615
## itel    1 lambda   0.500000 stress 0.053688 penalty 0.072330
## itel    1 lambda   0.600000 stress 0.066833 penalty 0.055452
## itel    1 lambda   0.700000 stress 0.078832 penalty 0.042269
## itel    1 lambda   0.800000 stress 0.089439 penalty 0.032019
## itel    1 lambda   0.900000 stress 0.098557 penalty 0.024079
## itel    1 lambda   1.000000 stress 0.106135 penalty 0.017940
## itel    6 lambda   2.000000 stress 0.130789 penalty 0.000148
## itel   13 lambda   3.000000 stress 0.131135 penalty 0.000000
```



Now the order is

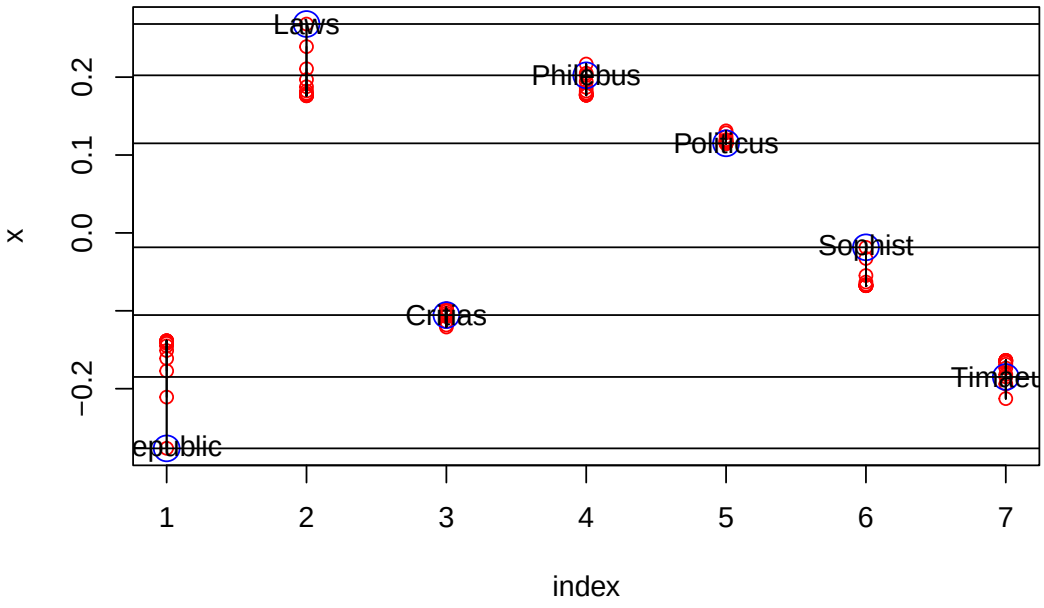
```
##      [,1]
## [1,] "Republic"
## [2,] "Timaeus"
## [3,] "Critias"
## [4,] "Sophist"
```

```
## [5,] "Politicus"
## [6,] "Philebus"
## [7,] "Laws"
```

which does indeed correspond to the global minimum.

With a different λ sequence we find the same solution.

```
## itel  169 lambda   0.000000 stress 0.000000 penalty 0.410927
## itel    3 lambda   0.001000 stress 0.000001 penalty 0.262296
## itel    2 lambda   0.002000 stress 0.000003 penalty 0.261483
## itel    2 lambda   0.004000 stress 0.000010 penalty 0.259872
## itel    2 lambda   0.008000 stress 0.000041 penalty 0.256690
## itel    2 lambda   0.016000 stress 0.000159 penalty 0.250470
## itel    2 lambda   0.032000 stress 0.000613 penalty 0.238574
## itel    2 lambda   0.064000 stress 0.002266 penalty 0.216785
## itel    2 lambda   0.128000 stress 0.007791 penalty 0.180067
## itel    2 lambda   0.256000 stress 0.023483 penalty 0.127006
## itel    2 lambda   0.512000 stress 0.056940 penalty 0.067948
## itel    3 lambda   1.024000 stress 0.107743 penalty 0.017937
## itel    8 lambda   2.048000 stress 0.131059 penalty 0.000032
## itel    9 lambda   4.096000 stress 0.131135 penalty 0.000000
```



The order is

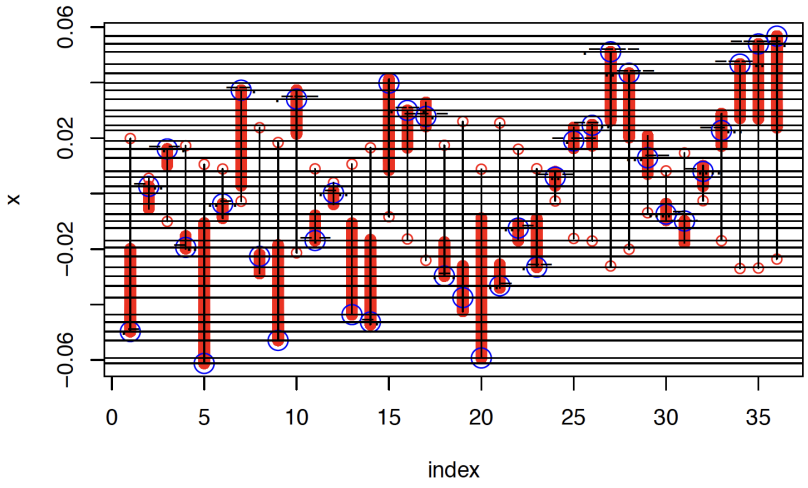
```
##      [,1]
## [1,] "Republic"
## [2,] "Timaeus"
## [3,] "Critias"
## [4,] "Sophist"
## [5,] "Politicus"
## [6,] "Philebus"
## [7,] "Laws"
```

26.5.3.10 Morse in One

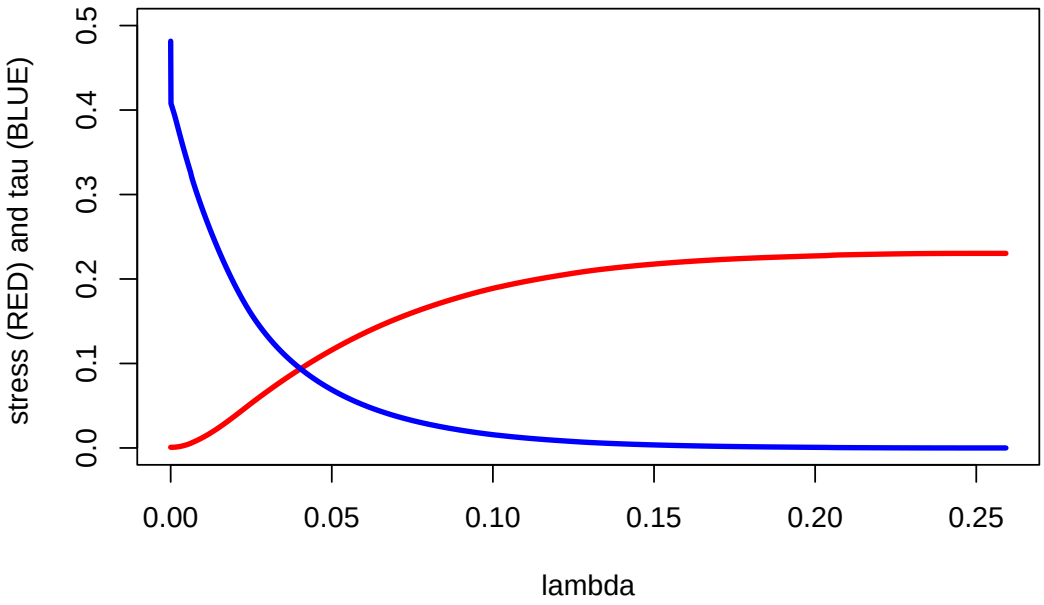
Now for a more challenging example. The Morse code data have been used to try out exact unidimensional MDS techniques, for example by Palubeckis (2013). We will enter the global minimum contest by using 10,000 values of λ, in an equally spaced sequence from 0 to 10. This is not as bad as it sounds. For the 10,000 FDS solutions `system.time()` tells us

```
##    user  system elapsed
##  14.092   0.457  14.610
```

The one-dimensional plot show quite a bit of movement, but much of it seems to be contained in the very first change of λ.



We can also plot stress and the penalty term as functions of λ. Again, note the big change in the penalty term when λ goes from zero to 0.001.



After the first 2593 values of λ the penalty term is zero and we stop, i.e. we estimate λ_+ is 2.593. At that point we have run a total of 5013 FDS iterations, and thus on average about two iterations per λ value. Stress has increased from 0.0007634501 to 0.2303106976 and the penalty value has decreased from 0.4815136419 to 0.0000000001. We find the following order of the points on the dimension.

```
##       [,1]
##  [1,] "."
##  [2,] "-"
##  [3,] ".."
##  [4,] ".-"
##  [5,] "-."
##  [6,] "--"
##  [7,] "..."
##  [8,] "..-"
##  [9,] ".-."
## [10,] ".--"
```

```
## [11,] "...."
## [12,] "-.."
## [13,] "-.-"
## [14,] "...-"
## [15,] "....."
## [16,] "....-"
## [17,] "..-."
## [18,] ".-.."
## [19,] "-..."
## [20,] "-..-"
## [21,] "-...."
## [22,] "...--"
## [23,] "-.-."
## [24,] "-.--"
## [25,] "--..."
## [26,] "--.."
## [27,] "--.-"
## [28,] ".--."
## [29,] ".---"
## [30,] "--."
## [31,] "---"
## [32,] "..---"
## [33,] "---.."
## [34,] ".----"
## [35,] "----."
## [36,] "-----"
```

Our order, and consequently our solution, is the same as the exact global solution given by Palubeckis (2013). See his table 4, reproduced below. The difference is that computing our solution takes 10 seconds, while his takes 494 seconds. But of course we would not have known we actually found the global mimimum if the exact exhaustive methods had not analyzed the same data as well.

Chapter 27

Software

In actual computer output using the scaling in formula (1.3) and (1.3) has some disadvantages. There are, say, M non-zero weights. The summation in #ref(eq:stressall) is really over M terms only. If n is at all large the scaled dissimilarities, and consequently the distances and the configuration, will become very small. Thus, in actual computation, or at least in the computer output, we scale our dissimilarities as $\frac{1}{2}\sum\sum_{1\leq j<i\leq n} w_{ij}\delta_{ij}^2 = M$. So, we scale our dissimilarities to one in formulas and to M in computations. Thus the computed stress will b

rcode

ccode

lib

Appendix A

Code

A.1 R Code

The MDS functions in R throughout work with square matrices of weights, dissimilarities, and distances. More efficient versions, that have their computations done in C, will be added along the way.

A.1.1 utilities.R

```
source ("common/indexing.R")
source ("common/io.R")
source ("common/linear.R")
source ("common/nextPC.R")
source ("common/decode.R")
source ("common/smacof.R")
```

A.1.2 common/indexing.r

```
kron <- function (i, j) {
  return (ifelse (i == j, 1, 0))
```

```
}

ein <- function (i, n) {
  return (ifelse (i == 1:n, 1, 0))
}

aijn <- function (i, j, n) {
  dif <- ein (i, n) - ein (j, n)
  return (outer (dif, dif))
}

jmat <- function (n) {
  return (diag(n) - 1 / n)
}

ccen <- function (x) {
  return (apply (x, 2, function (y)
    y - mean (y)))
}

repList <- function(x, n) {
  z <- list()
  for (i in 1:n)
    z <- c(z, list(x))
  return(z)
}

rcen <- function (x) {
  return (t (apply (x, 1, function (y)
    y - mean (y))))
}

dcen <- function (x) {
  return (ccen (rcen (x)))
}

wdef <- function (n) {
```

```
  return (1 - diag (n))
}


lower_triangle <- function (x) {
  n <- nrow (x)
  return (x[outer(1:n, 1:n, ">")])
}

fill_symmetric <- function (x) {
  m <- length (x)
  n <- (1 + sqrt (1 + 8 * m)) / 2
  d <- matrix (0, n, n)
  d[outer(1:n, 1:n, ">")] <- x
  return (d + t(d))
}
```

A.1.3 common/io.r

```
matrixPrint <- function (x,
                    digits = 6,
                    width = 8,
                    format = "f",
                    flag = "+") {
  print (noquote (formatC (
    x,
    digits = digits,
    width = width,
    format = format,
    flag = flag
  )))
}

iterationWrite <- function (labels, values, digits, width, format) {
  for (i in 1:length(labels)) {
```

```
    cat (labels[i],
         formatC(
           values[i],
           di = digits[i],
           wi = width[i],
           fo = format[i]
         ),
         " ")
  }
  cat("\n")
}

rotateEllipse <- function (x) {
  z <- (x[1,] + x[2,]) / 2
  x <- x - matrix (z, nrow(x), 2, byrow = TRUE)
  s <- sqrt (sum (x[1,] ^ 2))
  r <- matrix(c(x[1, 1], x[1, 2],-x[1, 2], x[1, 1]), 2, 2) / s
  x <- x %*% r
  e <- as.matrix (dist (x))
  d <- mean (rowSums(e[-(1:2), 1:2]))
  a <- d / 2
  c <- abs (x[1, 1])
  b <- sqrt (a ^ 2 - c ^ 2)
  return (list(
    x = x,
    a = a,
    b = b,
    c = c
  ))
}

plotEllipse <- function (x) {
  r <- rotateEllipse (x)
  f <- seq (0, 2 * pi, length = 100)
  z <- cbind(sin (f), cos (f))
  z[, 1] <- z[, 1] * r$a
  z[, 2] <- z[, 2] * r$b
```

```
  plot(z,
       type = "l",
       col = "RED",
       lwd = 2)
  text (r$x, as.character (1:nrow(r$x)))
  abline(h = 0)
  abline(v = 0)
}

draw_ellipse <- function (center,
                          radius,
                          a = diag (2),
                          np = 100,
                          ...) {
  par (pty = "s")
  e <- eigen(a)
  k <- e$vectors
  lbd <- e$values
  seq <- seq(0, 2 * pi, length = np)
  scos <- (radius * sin (seq)) / lbd[1]
  ccos <- (radius * cos (seq)) / lbd[2]
  sico <- k %*% rbind(scos, ccos) + center
  plot (sico[1,], sico[2,], type = "l", ...)
}
```

A.1.4 common/linear.r

```
## invert with pivot
## invert with bordering
## solve wth bordering
## solve with pivoting
## jacobi
## QR

gramy <- function (y, v) {
```

```
  r <- length (y)
  s <- sum (y[[1]] * (v %*% y[[1]]))
  y[[1]] <- y[[1]] / sqrt (s)
  for (j in 2:r) {
    for (i in 1:(j - 1)) {
      s <- sum (y[[i]] * (v %*% y[[j]]))
      y[[j]] <- y[[j]] - s * y[[i]]
    }
    s <- sum (y[[j]] * v %*% y[[j]])
    y[[j]] <- y[[j]] / sqrt (s)
  }
  return (y)
}



hinv <- function(x) {
  return (apply (x, c(1, 2), function (a)
    ifelse (a == 0, 0, 1 / a)))
}


circular <- function (n) {
  x <- seq (0, 2 * pi, length = n + 1)
  z <- matrix (0, n + 1, 2)
  z[, 1] <- sin (x)
  z[, 2] <- cos (x)
  return (z[-1, ])
}

direct_sum <- function (x) {
  n <- length (x)
  nr <- sapply (x, nrow)
  nc <- sapply (x, ncol)
  s <- matrix (0, sum (nr), sum (nc))
  k <- 0
  l <- 0
```

```
  for (j in 1:n) {
    s[k + (1:nr[j]), l + (1:nc [j])] <- x[[j]]
    k <- k + nr[j]
    l <- l + nc[j]
  }
  return (s)
}
```

A.1.5 common/nextPC.r

```
nextPermutation <- function (x) {
  if (all (x == (length(x):1)))
    return (NULL)
  z <- .C("nextPermutation", as.integer(x), as.integer(length(x)))
  return (z[[1]])
}

nextCombination <- function (x, n) {
  m <- length (x)
  if (all (x == ((n - m) + 1:m)))
    return (NULL)
  z <-
    .C("nextCombination",
       as.integer(n),
       as.integer (m),
       as.integer(x))
  return (z[[3]])
}
```

A.1.6 common/smacof.r

```
smacofNormW <- function(w) {
  return (w / sum(w))
```

```
}

smacofNormDelta <- function(w, delta) {
  return (delta / sqrt(sum (w * delta ^ 2)))
}

smacofNormXD <- function(w, delta, xold) {
  x <- apply (xold, 2, function(x)
    x - mean(x))
  d <- as.matrix(dist(x))
  s <- sum (w * delta * d) / sum (w * d ^ 2)
  return (list(x = x * s, d = d * s))
}

smacofLossR <- function (d, w, delta) {
  return (sum (w * (delta - d) ^ 2) / 2)
}

smacofBmatR <- function (d, w, delta) {
  dd <- ifelse (d == 0, 0, 1 / d)
  b <- -dd * w * delta
  diag (b) <- -rowSums (b)
  return(b)
}

smacofVmatR <- function (w) {
  v <- -w
  diag(v) <- -rowSums(v)
  return (v)
}

smacofGuttmanR <- function (x, b, vinv) {
  return (vinv %*% b %*% x)
}

smacofGradientR <- function (x, b, v) {
  return (2 * ((v - b) %*% x))
```

```
}

smacofHmatR <- function (x, b, v, d, w, delta) {
  n <- nrow (x)
  p <- ncol (x)
  r <- n * p
  h <- matrix (0, r, r)
  dd <- ifelse (d == 0, 0, 1 / d)
  cc <- w * delta * (dd ^ 3)
  for (s in 1:p) {
    ns <- (s - 1) * n + 1:n
    for (t in 1:s) {
      nt <- (t - 1) * n + 1:n
      cst <- matrix (0, n, n)
      for (i in 1:n) {
        for (j in 1:n) {
          cst[i, j] <- cc[i, j] * (x[i, s] - x[j, s]) * (x[i, t] - x[j, t])
        }
      }
      cst <- -cst
      diag(cst) <- -rowSums(cst)
      if (s == t) {
        h[ns, ns] <- b - cst
      } else {
        h[ns, nt] <- -cst
        h[nt, ns] <- -cst
      }
    }
  }
  return (h)
}

smacofHessianR <- function (x, b, v, d, w, delta) {
  n <- nrow (x)
  p <- ncol (x)
  h <- -smacofHmatR (x, b, v, d, w, delta)
  for (s in 1:p) {
```

```
    nn <- (s - 1) * n + 1:n
    h[nn, nn] <- h[nn, nn] + v
  }
  return(h)
}

smacofDerGuttmanR <- function(x, b, vinv, d, w, delta) {
  n <- nrow (x)
  p <- ncol (x)
  h <- smacofHmatR (x, b, v, d, w, delta)
  for (s in 1:p) {
    ns <- (s - 1) * n + 1:n
    for (t in 1:s) {
      nt <- (t - 1) * n + 1:n
      h[ns, nt] <- vinv %*% h[ns, nt]
    }
  }
  return(h)
}

smacofInitialR <- function (delta, p) {
  n <- nrow(delta)
  delta <- delta ^ 2
  rw <- rowSums (delta) / n
  sw <- sum (delta) / (n ^ 2)
  h <- -(delta - outer (rw, rw, "+") + sw) / 2
  e <- eigen (h)
  ea <- e$values
  ev <- e$vector
  ea <- ifelse (ea > 0, sqrt (abs(ea)), 0)[1:p]
  return (ev[, 1:p] %*% diag (ea))
}

smacofRandomStart <- function (w, delta, n, p) {
  x <- matrix(rnorm(n * p), n, p)
  x <- apply (x, 2, function(x)
    x - mean(x))
```

```
  d <- as.matrix(dist (x))
  a <- sum (w * delta * d) / sum (w * d ^ 2)
  return (a * x)
}

smacofVinvR <- function (v) {
  e <- 1 / nrow(v)
  return (solve (v + e) - e)
}

smacofR <-
  function (w,
            delta,
            p,
            xold = smacofInitialR(delta, p),
            xstop = FALSE,
            itmax = 1000,
            eps = 1e-10,
            verbose = TRUE) {
    labels = c("itel", "eiff", "sold", "snew")
    digits = c(4, 10, 10, 10)
    widths = c(6, 15, 15, 15)
    format = c("d", "f", "f", "f")
    n <- dim(delta)[1]
    itel <- 1
    w <- smacofNormW(w)
    delta <- smacofNormDelta(w, delta)
    xdold <- smacofNormXD(w, delta, xold)
    xold <- xdold$x
    dold <- xdold$d
    sold <- smacofLossR (dold, w, delta)
    bold <- smacofBmatR (dold, w, delta)
    vmat <- smacofVmatR (w)
    vinv <- smacofVinvR (vmat)
    repeat {
      xnew <- smacofGuttmanR (xold, bold, vinv)
      dnew <- as.matrix (dist (xnew))
```

```
    bnew <- smacofBmatR (dnew, w, delta)
    snew <- smacofLossR (dnew, w, delta)
    if (xstop) {
      eiff <- max (abs (xold - xnew))
    } else {
      eiff <- sold - snew
    }
    if (verbose) {
      values = c(itel, eiff, sold, snew)
      iterationWrite (labels, values, digits, widths, format)
    }
    if ((eiff < eps) || (itel == itmax)) {
      break
    }
    itel <- itel + 1
    xold <- xnew
    bold <- bnew
    dold <- dnew
    sold <- snew
  }
  return (
    list (
      x = xnew,
      d = dnew,
      b = bnew,
      g = smacofGradientR(xnew, bnew, vmat),
      h = smacofHessianR(xnew, bnew, vmat, dnew, w, delta),
      s = snew,
      itel = itel
    )
  )
}
```

A.1.7 properties.R

```
csupper <- function (lbd, mbd) {
  n <- length (lbd)
  m <- length (mbd)
  mad <- nad <- matrix (0, m, n)
  sad <- rep(0, n)
  plot(lbd,
       lbd,
       type = "n",
       ylab = "",
       ylim = c(0, 2))
  for (k in 1:m) {
    ly <- mbd[k]
    yy <- ly * z1 + (1 - ly) * z2
    dy <- dist (yy)
    by <- as.matrix(-delta / dy)
    diag (by) <- -rowSums(by)
    for (l in 1:n) {
      xx <- lbd[l] * z1 + (1 - lbd[l]) * z2
      dx <- dist (xx)
      rx <- sum (xx * (by %*% yy))
      nx <- sum (dx ^ 2)
      mad[k, l] <- 1 - 2 * rx + nx
    }
    lines (lbd, mad [k,])
    abline (v = ly)
  }
  lines(lbd,
        apply(mad, 2, min),
        type = "l",
        col = "BLUE",
        lwd = 3)
  for (l in 1:n) {
    xx <- lbd[l] * z1 + (1 - lbd[l]) * z2
    dx <- dist (xx)
    sad[l] <- 1 - 2 * sum (dx * delta) + sum (dx ^ 2)
```

```
  }
  lines (lbd,
         sad,
         type = "l",
         col = "RED",
         lwd = 3)
}

aglower <- function (lbd, mbd) {
  n <- length (lbd)
  m <- length (mbd)
  mad <- matrix (0, m, n)
  sad <- rep(0, n)
  plot(lbd,
       lbd,
       type = "n",
       ylab = "",
       ylim = c(0, 2))
  for (k in 1:m) {
    ly <- mbd[k]
    yy <- ly * z1 + (1 - ly) * z2
    dy <- dist (yy)
    ry <- sum (delta * dy)
    by <- as.matrix(-delta / dy)
    diag (by) <- -rowSums(by)
    for (l in 1:n) {
      xx <- lbd[l] * z1 + (1 - lbd[l]) * z2
      dx <- dist (xx)
      sx <- sum (xx * (by %*% xx))
      nx <- sum (dx ^ 2)
      mad[k, l] <- 1 - ry + nx - sx
    }
    lines (lbd, mad [k,])
    abline (v = ly)
  }
  lines(lbd,
        apply(mad, 2, max),
```

```
        type = "l",
        col = "BLUE",
        lwd = 3)
  for (l in 1:n) {
    xx <- lbd[l] * z1 + (1 - lbd[l]) * z2
    dx <- dist (xx)
    sad[l] <- 1 - 2 * sum (dx * delta) + sum (dx ^ 2)
  }
  lines (lbd,
         sad,
         type = "l",
         col = "RED",
         lwd = 3)
}
```

A.1.8 expandOneDim.R

```
expandRho <- function (delta, w = wdef(nrow(delta)), x, y) {
  n <- nrow(delta)
  s0 <- s1 <- s2 <- s3 <- 0
  for (i in 1:n) {
    for (j in 1:n) {
      if (i == j)
        next
      del <- delta[i, j]
      www <- w[i, j]
      dxx <- sum((x[i, ] - x[j, ]) ^ 2)
      dxr <- ifelse (dxx < 1e-15, 1, dxx)
      dsx <- sqrt(dxr)
      dyy <- sum((y[i, ] - y[j, ]) ^ 2)
      dxy <- sum((x[i, ] - x[j, ]) * (y[i, ] - y[j, ]))
      vxy <- dyy - ((dxy) ^ 2) / dxr
      s0 <- s0 + www * del * dsx
      s1 <- s1 + www * (del / dsx) * dxy
      s2 <- s2 + www * (del / dsx) * vxy
```

```
      s3 <- s3 + www * dxy * (del / (dsx ^ 3)) * vxy
    }
  }
  return(c(s0, s1, s2 / 2, -s3 / 2))
}

expandEta2 <- function (delta, w = wdef(nrow(delta)), x, y) {
  n <- nrow(delta)
  s0 <- s1 <- s2 <- s3 <- 0
  for (i in 1:n) {
    for (j in 1:n) {
      dxx <- sum((x[i, ] - x[j, ]) ^ 2)
      dsx <- sqrt(dxx)
      dyy <- sum((y[i, ] - y[j, ]) ^ 2)
      dxy <- sum((x[i, ] - x[j, ]) * (y[i, ] - y[j, ]))
      s0 <- s0 + w[i, j] * dxx
      s1 <- s1 + w[i, j] * dxy
      s2 <- s2 + w[i, j] * dyy
    }
  }
  return(c(s0, 2 * s1, s2, s3))
}

expandStress <- function (delta, w = wdef(nrow(delta)), x, y) {
  return (c(1, 0, 0, 0) + expandEta2(delta, w, x, y) - 2 * expandRho(delta, w,
}

expandTester <- function (delta,
                          w = wdef(nrow(delta)),
                          x,
                          y,
                          left = -1,
                          right = 1,
                          length = 1001,
                          order = 3) {
  w <- w / sum (w)
  delta <- delta / sqrt(sum(w * delta ^ 2))
```

```
  x <- apply(x, 2, function (x)
    x - mean(x))
  y <- apply(y, 2, function (x)
    x - mean(x))
  d <- as.matrix(dist(x))
  s <- sum(w * d * delta) / sum(w * d * d)
  d <- s * d
  x <- s * x
  h <- expandStress (delta, w, x, y)
  SEQ <- seq(left, right, length = length)
  sig <- rep(0, length)
  sag <- rep(h[1], length)
  for (i in 1:length) {
    z <- x + SEQ[i] * y
    d <- as.matrix (dist(z))
    eta2 <- sum(w * d ^ 2)
    rho <- sum(w * delta * d)
    sig[i] <- 1 - 2 * rho + eta2
    if (order > 0) {
      sag[i] <- sag[i] + SEQ[i] * h[2]
    }
    if (order > 1) {
      sag[i] <- sag[i] + (SEQ[i] ^ 2) * h[3]
    }
    if (order > 2) {
      sag[i] <- sag[i] + (SEQ[i] ^ 3) * h[4]
    }
  }
  return(cbind(sig, sag))
}
```

A.1.9 pictures.R

```
dommy <- function () {
  ya <- matrix(c(1, -1, 1, -1, 0, 1, 1, -1, -1, 0), 5, 2)
```

```
  yy <- seq (0, 2 * pi, length = 6)[1:5]
  yb <- cbind (sin (yy), cos (yy))
  ya <- apply(ya, 2, function (x)
    x - mean(x))
  yb <- apply(yb, 2, function (x)
    x - mean(x))
  y1 <- ya / sqrt (5 * sum (ya ^ 2))
  y2 <- yb / sqrt (5 * sum (yb ^ 2))
  deq <- as.dist (1 - diag(5))
  deq <- deq / sqrt (sum (deq ^ 2))
  y1 <- sum (dist (y1) * deq) * y1
  y2 <- sum (dist (y2) * deq) * y2
}

twostress <- function (deq, y1, y2, a, b) {
  d <- dist (a * y1 + b * y2)
  eta2 <- sum (d ^ 2)
  rho <- sum (d * deq)
  stress <- 1 - 2 * rho + eta2
  return(list(
    eta2 = eta2,
    rho = rho,
    stress = stress
  ))
}

zeroes <- function (y1, y2) {
  n <- nrow (y1)
  for (i in 2:n) {
    for (j in 1:(i - 1)) {
      yy <- rbind (y1[i, ] - y1[j, ], y2[i, ] - y2[j, ])
      ee <- eigen(tcrossprod(yy))$values
      print (c(i, j, ee))
    }
  }
}
```

```
pairme <- function (x, y) {
  n <- nrow(x)
  m <- ncol(x)
  z <- matrix (0, 2, m)
  for (i in 2:n) {
    for (j in 1:(i - 1)) {
      z[1,] <- x[i,] - x[j,]
      z[2,] <- y[i,] - y[j,]
      s <- svd (z)
      a <- s$d[2]
      b <- s$u[, 2]
      if (a < 1e-10) {
        cat (
          formatC(
            i,
            format = "d",
            digits = 2,
            width = 4
          ),
          formatC(
            j,
            format = "d",
            digits = 2,
            width = 4
          ),
          formatC(
            c(a, b),
            format = "f",
            digits = 6,
            width = 10
          ),
          "\n"
        )
      }
    }
  }
}
```

```
dummy <- function () {
  set.seed(12345)
  x <- matrix(rnorm(10), 5, 2)
  x <- apply (x, 2, function(x)
    x - mean(x))
  delta <- dist(x)
  d <- dist (x)
  eps <- (-500:500) / 100
  sy <- rep (0, 1001)
  plot (
    0,
    0,
    xlim = c(-5, 5),
    ylim = c(0, 20),
    xlab = "epsilon",
    ylab = "stress",
    type = "n"
  )
  for (i in 1:5) {
    for (j in 1:2) {
      for (k in 1:1001) {
        y <- x
        y[i, j] <- x[i, j] + eps[k]
        dy <- dist (y)
        sy[k] <- sum ((delta - dy) ^ 2)
      }
      lines (eps, sy, lwd = 2, col = "RED")
    }
  }
}


bmat2 <- function (a, b, x, y, delta) {
  bm <- matrix (0, 2, 2)
  hm <- matrix (0, 2, 2)
  z <- c(a, b)
  for (i in 1:4) {
```

```
    for (j in 1:4) {
      if (i == j)
        next
      uij <- uu (i, j, x, y)
      uz <- drop (uij %*% z)
      dij <- sqrt (sum (uij * outer (z, z)))
      bm <- bm + (delta[i, j] / dij) * uij
      hm <-
        hm + (delta[i, j] / dij) * (uij - outer (uz, uz) / sum (z * uz))
    }
  }
  return (list (b = bm, h = hm))
}

stress2 <- function (a, b, x, y, delta) {
  z <- c (a, b)
  bm <- bmat2 (a, b, x, y, delta)$b
  return (1 + sum(z ^ 2) / 2 - sum (z * bm %*% z))
}

rho2 <- function (a, b, x, y, delta) {
  z <- c (a, b)
  bm <- bmat2 (a, b, x, y, delta)$b
  return (sum (z * bm %*% z))
}

vv <- function (i, j, x, y) {
  a <- matrix (0, 2, 2)
  a[1, 1] <- sum ((x[i,] - x[j, ]) ^ 2)
  a[2, 2] <- sum ((y[i,] - y[j, ]) ^ 2)
  a[1, 2] <- a[2, 1] <- sum ((x[i,] - x[j, ]) * (y[i,] - y[j,]))
  return (a)
}

uu <- function (i, j, x, y) {
  n <- nrow (x)
  asum <-
```

```
    2 * n * matrix (c (sum(x ^ 2), sum (x * y), sum (x * y), sum (y ^ 2)), 2,
  csum <- solve (chol (asum))
  return (t(csum) %*% vv (i, j, x, y) %*% csum)
}

smacof2 <-
  function (a,
            b,
            x,
            y,
            delta,
            eps = 1e-10,
            itmax = 1000,
            verbose = TRUE) {
    zold <- c(a, b)
    bold <- bmat2 (a, b, x, y, delta)$b
    fold <- 1 + sum(zold ^ 2) / 2 - sum (zold * bold %*% zold)
    itel <- 1
    repeat {
      znew <- drop (bold %*% zold)
      bhmt <- bmat2 (znew[1], znew[2], x, y, delta)
      bnew <- bhmt$b
      fnew <- 1 + sum(znew ^ 2) / 2 - sum (znew * bnew %*% znew)
      if (verbose) {
        cat (
          formatC (itel, width = 4, format = "d"),
          formatC (
            fold,
            digits = 10,
            width = 13,
            format = "f"
          ),
          formatC (
            fnew,
            digits = 10,
            width = 13,
            format = "f"
```

```
          ),
          "\n"
        )
      }
      if ((itel == itmax) || (fold - fnew) < eps)
        break ()
      itel <- itel + 1
      fold <- fnew
      zold <- znew
      bold <- bnew
    }
    return (
      list (
        stress = fnew,
        theta = znew,
        itel = itel,
        b = bnew,
        g = znew - bnew %*% znew,
        h = diag(2) - bhmt$h
      )
    )
  }


newton2 <-
  function (a,
            b,
            x,
            y,
            delta,
            eps = 1e-10,
            itmax = 1000,
            verbose = TRUE) {
    zold <- c(a, b)
    bhmt <- bmat2 (a, b, x, y, delta)
    bold <- bhmt$b
    hold <- diag(2) - bhmt$h
```

```
  fold <- 1 + sum(zold ^ 2) / 2 - sum (zold * bold %*% zold)
  itel <- 1
  repeat {
    znew <- drop (solve (hold, bold %*% zold))
    bhmt <- bmat2 (znew[1], znew[2], x, y, delta)
    bnew <- bhmt$b
    hnew <- diag(nrow(bnew)) - bhmt$h
    fnew <- 1 + sum(znew ^ 2) / 2 - sum (znew * bnew %*% znew)
    if (verbose) {
      cat (
        formatC (itel, width = 4, format = "d"),
        formatC (
          fold,
          digits = 10,
          width = 13,
          format = "f"
        ),
        formatC (
          fnew,
          digits = 10,
          width = 13,
          format = "f"
        ),
        "\n"
      )
    }
    if ((itel == itmax) || abs (fold - fnew) < eps)
      break ()
    itel <- itel + 1
    fold <- fnew
    zold <- znew
    bold <- bnew
    hold <- hnew
  }
  return (list (
    stress = fnew,
    theta = znew,
```

```
      itel = itel,
      b = bnew,
      g = znew - bnew %*% znew,
      h = hnew
    ))
  }
```

A.1.10 classical.R

```
tau <- function (x) {
  return (- 0.5 * dcen (x))
}

kappa <- function (x) {
  return (outer (diag (x), diag (x), "+") - 2 * x)
}

fcmds <-
  function (delta,
            xold,
            ninner = 1,
            itmax = 100,
            eps = 1e-6,
            verbose = TRUE) {
    itel <- 0
    p <- ncol (xold)
    xold <- apply (xold, 2, function (x)
      x - mean(x))
    xold <- qr.Q (qr (xold))
    repeat {
      xinn <- xold
      for (i in 1:ninner) {
        xnew <- delta %*% xinn
        xnew <- -apply (xnew, 2, function (x)
          x - mean (x)) / 2
```

```
        xinn <- xnew
        itel <- itel + 1
      }
      qnew <- qr (xnew)
      xnew <- qr.Q (qnew)
      rnew <- qr.R (qnew)
      epsi <- 2 * p - 2 * sum (svd (crossprod (xold, xnew))$d)
      if (verbose) {
        cat(
          "itel ",
          formatC (
            itel,
            digits = 4,
            width = 6,
            format = "d"
          ),
          "epsi ",
          formatC (
            epsi,
            digits = 10,
            width = 15,
            format = "f"
          ),
          "\n"
        )
      }
      if ((epsi < eps) || (itel == itmax))
        break
      xold <- xnew
    }
    return (list (x = xnew, r = rnew, itel = itel))
  }

treq <- function (x) {
  n <- nrow (d)
  m <- -Inf
  for (i in 2:n) {
```

```
    for (j in 1:(i - 1)) {
      for (k in 1:n) {
        if ((k == i) || (k == j))
          next
        m <- max (m, x[i, j] - (x[i, k] + x[k, j]))
      }
    }
  }
  return (m)
}

acbound <- function (d) {
  n <- nrow (d)
  s <- qr.Q (qr (cbind (1, matrix (rnorm (
    n * (n - 1)
  ), n, n - 1))))
  k <- tau (d * d)
  l <- 2 * tau (d)
  m <- jmat (n) / 2
  ma <- -Inf
  for (i in 2:n)
    for (j in 1:(i - 1)) {
      v <- solve(polynomial(c(k[i, j], l[i, j], m[i, j])))
      ma <- max(ma, max(v))
    }
  return (list(ma = ma, mw = k + ma * l + m * ma ^ 2))
}

aceval <- function (d, bnd = c(-10, 10)) {
  n <- nrow (d)
  k <- tau (d * d)
  l <- 2 * tau (d)
  m <- jmat (n) / 2
  s <- qr.Q (qr (cbind (1, matrix (rnorm (
    n * (n - 1)
  ), n, n - 1))))
  kc <- (crossprod (s, k) %*% s)[-1,-1]
```

```
  lc <- (crossprod (s, l) %*% s)[-1,-1]
  mc <- (crossprod (s, m) %*% s)[-1,-1]
  a <- seq(bnd[1], bnd[2], length = 1000)
  b <- rep(0, 1000)
  for (i in 1:1000) {
    ww <- kc + lc * a[i] + mc * (a[i] ^ 2)
    b[i] <- min (eigen(ww)$values)
  }
  return (list(a = a, b = b))
}

acqep <- function(d) {
  n <- nrow (d)
  k <- tau (d * d)
  l <- 2 * tau (d)
  m <- jmat (n) / 2
  s <- qr.Q (qr (cbind (1, matrix (rnorm (
    n * (n - 1)
  ), n, n - 1))))
  nn <- n - 1
  ns <- 1:nn
  kc <- (crossprod (s, k) %*% s)[-1,-1]
  lc <- (crossprod (s, l) %*% s)[-1,-1]
  mc <- (crossprod (s, m) %*% s)[-1,-1]
  ma <- matrix(0, 2 * nn, 2 * nn)
  ma[ns, nn + ns] <- diag (n - 1)
  ma[nn + ns, ns] <- -2 * kc
  ma[nn + ns, nn + ns] <- -2 * lc
  return (list (ma = ma, me = eigen(ma)$values))
}
```

A.1.11 minimization.R

A.1.12 full.R

```
library(MASS)

torgerson <- function(delta, p = 2) {
  doubleCenter <- function(x) {
    n <- dim(x)[1]
    m <- dim(x)[2]
    s <- sum(x) / (n * m)
    xr <- rowSums(x) / m
    xc <- colSums(x) / n
    return((x - outer(xr, xc, "+")) + s)
  }
  z <-
    eigen(-doubleCenter((as.matrix (delta) ^ 2) / 2), symmetric = TRUE)
  v <- pmax(z$values, 0)
  return(z$vectors[, 1:p] %*% diag(sqrt(v[1:p])))
}


makeA <- function (n) {
  m <- n * (n - 1) / 2
  a <- list()
  for (j in 1:(n - 1))
    for (i in (j + 1):n) {
      d <- ein (i, n) -ein (j, n)
      e <- outer (d, d)
      a <- c(a, list (e))
    }
  return (a)
}

makeD <- function (a, x) {
  return (sapply (a, function (z)
    sqrt (sum (x * (
```

```
      z %*% x
    )))))
}

makeB <- function (w, delta, d, a) {
  n <- length (a)
  m <- nrow (a[[1]])
  b <- matrix (0, m , m)
  for (i in 1:n)
    b <- b + w[i] * (delta[i] / d[i]) * a[[i]]
  return (b)
}

makeV <- function (w, a) {
  n <- length (a)
  m <- nrow (a[[1]])
  v <- matrix (0, m, m)
  for (i in 1:n)
    v <- v + w[i] * a[[i]]
  return (v)
}

inBetween <- function (alpha, beta, x, y, w, delta, a) {
  z <- alpha * x + beta * y
  d <- makeD (a, z)
  return (sum (w * (delta - d) ^ 2))
}

biBase <- function (x, y, a) {
  biBi <- function (x, y, v) {
    a11 <- sum (x * (v %*% x))
    a12 <- sum (x * (v %*% y))
    a22 <- sum (y * (v %*% y))
    return (matrix (c(a11, a12, a12, a22), 2, 2))
  }
  return (lapply (a, function (u)
    biBi (x, y, u)))
```

```
}

fullMDS <-
  function (delta,
            w = rep (1, length (delta)),
            xini,
            a,
            itmax = 100,
            eps = 1e-6,
            verbose = TRUE) {
    m <- length (a)
    v <- makeV (w, a)
    vv <- ginv (v)
    xold <- xini
    dold <- makeD (a, xini)
    sold <- sum ((delta - dold) ^ 2)
    bold <- makeB (w, delta, dold, a)
    itel <- 1
    repeat {
      xnew <- vv %*% bold %*% xold
      dnew <- makeD (a, xnew)
      bnew <- makeB (w, delta, dnew, a)
      snew <- sum ((delta - dnew) ^ 2)
      if (verbose) {
        cat (
          formatC (itel, width = 4, format = "d"),
          formatC (
            sold,
            digits = 10,
            width = 13,
            format = "f"
          ),
          formatC (
            snew,
            digits = 10,
            width = 13,
            format = "f"
```

```
          ),
          "\n"
        )
      }
      if ((itel == itmax) || (abs(sold - snew) < eps))
        break
      itel <- itel + 1
      xold <- xnew
      dold <- dnew
      sold <- snew
      bold <- bnew
    }
    return (list (
      x = xnew,
      d = dnew,
      delta = delta,
      s = snew,
      b = bnew,
      v = v,
      itel = itel
    ))
  }
```

A.1.13 unfolding.R

```
dummy <- function () {
  set.seed(12345)

  x <- matrix (rnorm(16), 8, 2)
  x <- apply (x, 2, function (x)
    x - mean (x))
  y <- matrix (rnorm(10), 5, 2)
  a <- rowSums (x ^ 2)
  b <- rowSums (y ^ 2)
  d <- sqrt (outer(a, b, "+") - 2 * tcrossprod (x, y))
```

```
  set.seed (12345)
  x <- matrix(rnorm(10), 5, 2)
  x <- apply (x, 2, function (x)
    x - mean (x))
  x <- qr.Q(qr(x))
  y <- matrix(rnorm(12), 6, 2)
  v <- apply (y, 2, mean)
  print (v)
  dx <- diag(tcrossprod(x))
  dy <- diag(tcrossprod(y))
  xy <- tcrossprod(x, y)
  dd <- outer(dx, dy, "+") - 2 * xy
  j5 <- diag(5) - 1 / 5
  j6 <- diag(6) - 1 / 6
  dc <- -(j5 %*% dd %*% j6) / 2
  sv <- svd (dc, nu = 2, nv = 2)
  xs <- sv$u
  ys <- sv$v %*% diag (sv$d[1:2])
  tt <- crossprod (x, xs)
  dk <- diag (tcrossprod(xs))
  dl <- diag (tcrossprod(ys))
  dr <- dd - outer (dk, dl, "+") + 2 * tcrossprod (xs, ys)
  print (dr)
}

schoenemann <- function (delta, p) {
  n <- nrow (delta)
  m <- ncol (delta)
  l <- p * (p + 1) / 2
  d <- delta ^ 2
  e <- torgerson (d)
  q <- svd (e, nu = p, nv = p)
  g <- q$u
  h <- q$v %*% diag (q$d[1:p])
  f <- d + 2 * tcrossprod(g, h)
  a <- apply (ccen (f), 1, mean)
  r <- matrix (0, n, l)
```

```
  k <- 1
  for (i in 1:p) {
    for (j in 1:i) {
      if (i == j) {
        r[, k] <- g[, i] ^ 2
      } else {
        r[, k] <- 2 * g[, i] * g[, j]
      }
      k <- k + 1
    }
  }
  lhs <- cbind (ccen(r), ccen (-2 * g))
  b <- lm.fit (lhs, a)$coefficients
  k <- 1
  s <- matrix (0, p, p)
  for (i in 1:p) {
    for (j in 1:i) {
      if (i == j) {
        s[i, i] = b[k]
      } else {
        s[i, j] <- s[j, i] <- b[k]
      }
      k <- k + 1
    }
  }
  e <- eigen (s)
  f <- e$values
  if (min(f) < 0) {
    stop ("Negative eigenvalue, cannot proceed")
  }
  t <- e$vectors %*% diag (sqrt (f))
  v <- solve (t, b[-(1:l)])
  x <- g %*% t
  y <-
    h %*% (e$vectors %*% diag (1 / sqrt (f))) + matrix (v, m, p, byrow = TRUE)
  return (list (x = x, y = y))
}
```

```
unfoldals <- function (offdiag) {
  n <- nrow (offdiag)
  m <- ncol (offdiag)
  dd <- offdiag ^ 2
  delta <- matrix (0, n + m, n + m)
  delta[1:n, n + (1:m)] <- dd
  delta <- pmax(delta, t(delta))
  cc <-
    dd - outer (rowSums(dd) / m, colSums (dd) / n, "+") + sum(dd) / (n * m)
  sc <- svd (-cc / 2)
  lb <- diag (sqrt(sc$d))
  xold <- sc$u %*% lb
  yold <- sc$v %*% lb
  zold <- rbind (xold, yold)
  lold <- rowSums (zold ^ 2)
  dold <- outer (lold, lold, "+") - 2 * tcrossprod (zold)

}

teqbounds <- function (offdiag) {
  n <- nrow (offdiag)
  m <- ncol (offdiag)
  a <- matrix (0, n, n)
  b <- matrix (0, m, m)
  for (i in 2:n) {
    for (j in 1:(i - 1)) {
      smin = Inf
      smax = -Inf
      for (k in 1:m) {
        smin = min (smin, offdiag[i, k] + offdiag[j, k])
        smax = max (smax, abs (offdiag[i, k] - offdiag[j, k]))
      }
      a[i, j] <- a[j, i] <- (smin + smax) / 2
    }
  }
  for (i in 2:m) {
    for (j in 1:(i - 1)) {
```

```
      smin = Inf
      smax = -Inf
      for (k in 1:n) {
        smin = min (smin, offdiag[k, i] + offdiag[k, j])
        smax = max (smax, abs (offdiag[k, i] - offdiag[k, j]))
      }
      b[i, j] <- b[j, i] <- (smin + smax) / 2
    }
  }
  return (list (a = a, b = b))
}
```

A.1.14 constrained.R

```
pcircsmacof <-
  function (delta,
            w = wdef (nrow (delta)),
            p = 2,
            x = smacofInitialR (delta, p),
            itmax = 1000,
            eps = 1e-6,
            verbose = TRUE) {
    labels = c("itel", "sold", "snew")
    digits = c(4, 10, 10)
    widths = c(6, 15, 15)
    format = c("d", "f", "f")
    n <- nrow (x)
    p <- ncol (x)
    xold <- x / sqrt (rowSums (x ^ 2))
    dold <- as.matrix (dist (xold))
    v <- smacofVmatR (w)
    e <- max (eigen (v, only.values = TRUE)$values)
    vinv <- ginv(v)
    itel <- 1
    sold <- smacofLossR (dold, w, delta)
```

```
    repeat {
      b <- smacofBmatR (dold, w, delta)
      xgut <- smacofGuttmanR(xold, b, vinv)
      xtar <- xold + v %*% (xgut - xold) / e
      xlen <- sqrt (rowSums (xtar ^ 2))
      xrad <- mean (xlen)
      xnew <- (xtar / xlen) * xrad
      dnew <- as.matrix (dist(xnew))
      snew <- smacofLossR (dnew, w, delta)
        if (verbose) {
          values = c(itel, sold, snew)
          iterationWrite (labels, values, digits, width, format)
        }
        if (((sold - snew) < eps) || (itel == itmax)) {
          break
        }
        itel <- itel + 1
        xold <- xnew
        sold <- snew
      }
      return (list (
        x = xnew,
        d = dnew,
        stress = snew,
        radius = xrad,
        itel = itel
      ))
  }


pellipsmacof <-
  function (delta,
            w = wdef (nrow (delta)),
            p = 2,
            x = smacofInitialR (delta, p),
            itmax = 1000,
            eps = 1e-6,
```

```
          verbose = TRUE) {
  labels = c("itel", "sold", "smid", "snew")
  digits = c(4, 10, 10, 10)
  widths = c(6, 15, 15, 15)
  format = c("d", "f", "f", "f")
  n <- nrow (x)
  p <- ncol (x)
  yold <- x / sqrt (rowSums (x ^ 2))
  xlbd <- rep (1, p)
  xold <- yold %*% diag (xlbd)
  dold <- as.matrix (dist (xold))
  v <- smacofVmatR (w)
  e <- max (eigen (v, only.values = TRUE)$values)
  vinv <- ginv(v)
  itel <- 1
  sold <- smacofLoss(dold, w, delta)
  repeat {
    b <- smacofBmatR (dold, w, delta)
    xgut <- smacofGuttmanR(xold, b, vinv)
    for (s in 1:p) {
      xlbd[s] <-
        sum (xgut[, s] * (v %*% yold[, s])) / sum (yold[, s] * (v %*% yold[,
    }
    xmid <- yold %*% diag (xlbd)
    dmid <- as.matrix (dist (xmid))
    smid <- sum (w * (delta - dmid) ^ 2) / 2
    mlbd <- max (xlbd ^ 2)
    ytar <-
      yold + v %*% ((xgut %*% diag (1 / xlbd)) - yold) %*% diag (xlbd ^ 2) /
    ylen <- sqrt (rowSums (ytar ^ 2))
    ynew <- ytar / ylen
    xnew <- ynew %*% diag (xlbd)
    dnew <- as.matrix (dist(xnew))
    snew <- sum (w * (delta - dnew) ^ 2) / 2
    if (verbose) {
      values = c(itel, sold, smid, snew)
      iterationWrite (labels, values, digits, width, format)
```

```
      }
      if (((sold - snew) < eps) || (itel == itmax)) {
        break
      }
      itel <- itel + 1
      xold <- xnew
      yold <- ynew
      sold <- snew
    }
    return (list (
      x = xnew,
      d = dnew,
      stress = snew,
      axes = xlbd,
      itel = itel
    ))
  }

dcircsmacof <-
  function (delta,
            w = wdef (nrow (delta)),
            p = 2,
            x = smacofInitialR (delta, p),
            pen = 1,
            itmax = 1000,
            eps = 1e-6,
            verbose = TRUE) {
    labels = c("itel", "sold", "smid", "snew")
    digits = c(4, 10, 10, 10)
    widths = c(6, 15, 15, 15)
    format = c("d", "f", "f", "f")
    n <- nrow (x)
    xold <-
      rbind (0, x / sqrt (rowSums(x ^ 2)))
    dold <- as.matrix (dist (xold))
    w <- rbind (pen, cbind (pen, w))
    delta <- rbind (1, cbind (1, delta))
```

```
    w[1, 1] <- delta [1, 1] <- 0
    v <- smacofVmatR (w)
    vinv <- ginv(v)
    itel <- 1
    sold <- smacofLossR(dold, w, delta)
    repeat {
      b <- smacofBmatR(dold, w, delta)
      xnew <- smacofGuttmanR(xold, b, vinv)
      dnew <- as.matrix (dist (xnew))
      smid <- smacofLossR(dnew, w, delta)
      a <- sum (dnew[1,]) / n
      delta[1,] <- delta[, 1] <- a
      delta[1, 1] <- 0
      snew <- smacofLossR(dnew, w, delta)
      if (verbose) {
        values = c(itel, sold, smid, snew)
        iterationWrite (labels, values, digits, width, format)
      }
      if (((sold - snew) < eps) || (itel == itmax)) {
        break
      }
      itel <- itel + 1
      xold <- xnew
      sold <- snew
    }
    return (list (
      x = xnew,
      d = dnew,
      a = a,
      stress = snew,
      itel = itel
    ))
  }

dellipsmacof <-
  function (delta,
            w = wdef (nrow (delta)),
```

```
          p = 2,
          x = smacofInitialR (delta, p),
          pen = 1,
          itmax = 1000,
          eps = 1e-6,
          verbose = TRUE) {
  labels = c("itel", "sold", "smid", "snew")
  digits = c(4, 10, 10, 10)
  widths = c(6, 15, 15, 15)
  format = c("d", "f", "f", "f")
  n <- nrow (x)
  set.seed(12345)
  focal <- rnorm(2)
  xold <-
    rbind (focal, -focal, x / sqrt (rowSums(x ^ 2)))
  dold <- as.matrix (dist (xold))
  w <- rbind (pen, pen, cbind (pen, pen, w))
  delta <- rbind (1, 1, cbind (1, 1, delta))
  w[1:2, 1:2] <- delta [1:2, 1:2] <- 0
  v <- smacofVmatR (w)
  vinv <- ginv(v)
  itel <- 1
  sold <- smacofLossR(dold, w, delta)
  repeat {
    b <- smacofBmatR (dold, w, delta)
    xnew <- smacofGuttmanR (xold, b, vinv)
    dnew <- as.matrix (dist (xnew))
    smid <- smacofLossR(dnew, w, delta)
    dsub <- dnew[1:2, 2 + (1:n)]
    asub <- sum (dsub) / (2 * n)
    dsub <- ccen (dsub) + asub
    delta[1:2, 2 + (1:n)] <- dsub
    delta[2 + (1:n), 1:2] <- t(dsub)
    snew <- smacofLossR(dnew, w, delta)
    if (verbose) {
      values = c(itel, sold, smid, snew)
      iterationWrite (labels, values, digits, width, format)
```

```
      }
      if (((sold - snew) < eps) || (itel == itmax)) {
        break
      }
      itel <- itel + 1
      xold <- xnew
      sold <- snew
    }
    return (list (
      pen = pen,
      x = xnew,
      d = dnew,
      stress = snew,
      itel = itel
    ))
  }
```

A.1.15 nominal.R

```
baseplot <- function (x,
                      y,
                      z,
                      wx = TRUE,
                      wy = TRUE,
                      wz = TRUE) {
  par(pty="s")
  plot(
    x,
    xlim = c(-3, 3),
    ylim = c(-3, 3),
    xlab = "",
    ylab = "",
    type  = "n"
  )
  if (wx)
```

```
  points(x, col = "RED", cex = 1)
if (wy)
  points(y, col = "BLUE", cex = 1)
if (wz)
  points(z, col = "GREEN", cex = 1)
mx <- apply(x, 2, mean)
my <- apply(y, 2, mean)
mz <- apply(z, 2, mean)
if (wx)
  points(
    matrix(mx, 1, 2),
    col = "RED",
    pch = 5,
    cex = 2,
    lwd = 2
  )
if (wy)
  points(
    matrix(my, 1, 2),
    col = "BLUE",
    pch = 5,
    cex = 2,
    lwd = 2
  )
if (wz)
  points(
    matrix(mz, 1, 2),
    col = "GREEN",
    pch = 5,
    cex = 2,
    lwd = 2
  )
if (wx)
  for (i in 1:10) {
    lines(rbind(x[i,], mx))
  }
if (wy)
```

```
    for (i in 1:5) {
      lines(rbind(y[i,], my))
    }
  if (wz)
    for (i in 1:5) {
      lines(rbind(z[i,], mz))
    }
}
```

A.1.16 sstress.R

```
strainAdd <-
  function (delta,
            w = rep (1, length (delta)),
            p = 2,
            itmax = 100,
            eps = 1e-6,
            verbose = TRUE) {
    delta <- as.matrix (delta ^ 2)
    n <- nrow(delta)

  }

strainWeight <-
  function (delta,
            w = rep (1, length (delta)),
            p = 2,
            itmax = 100,
            eps = 1e-6,
            verbose = TRUE) {

  }

alscal <-
  function (delta,
```

```
          p,
          x = torgerson (delta, p),
          w = wdef (nrow (x)),
          itmax = 1000,
          eps = 1e-6,
          verbose = TRUE,
          check = TRUE) {
  n <- nrow (x)
  delta <- delta ^ 2
  d <- as.matrix (dist (x)) ^ 2
  sold <- sum (w * (delta - d) ^ 2)
  wsum <- rowSums (w)
  itel <- 1
  snew <- sold
  repeat {
    for (k in 1:n) {
      t4 <- wsum[k]
      for (s in 1:p) {
        u <- x[, s] - x[k, s]
        t0 <- snew
        t1 <- t2 <- t3 <- 0
        for (i in 1:n) {
          t1 <- t1 + 4 * w[i, k] * (d[i, k] - delta[i, k]) * u[i]
          t2 <-
            t2 + 2 * w[i, k] * ((d[i, k] - delta[i, k]) + 2 * u[i] ^ 2)
          t3 <- t3 + 4 * w[i, k] * u[i]
        }
        pp <- polynomial(c(t0,-t1, t2,-t3, t4))
        qq <- deriv (pp)
        ss <- solve (qq)
        ss <- Re (ss[which (abs (Im (ss)) < 1e-10)])
        tt <- predict (pp, ss)
        snew <- min (tt)
        root <- ss[which.min (tt)]
        x[k, s] <- x[k, s] + root
        for (i in (1:n)[-k]) {
          d[i, k] <- d[i, k] - 2 * root * u[i] + root ^ 2
```

```
            d[k, i] <- d[i, k]
          }
        }
      }
      if (verbose) {
        cat(
          "itel ",
          formatC(itel, width = 6, format = "d"),
          "sold ",
          formatC(
            sold,
            digits = 6,
            width = 15,
            format = "f"
          ),
          "snew ",
          formatC(
            snew,
            digits = 6,
            width = 15,
            format = "f"
          ),
          "\n"
        )
      }
      if (((sold - snew) < eps) || (itel == itmax)) {
        break
      }
      sold <- snew
      itel <- itel + 1
    }
    return (list (
      x = x,
      sstress = snew,
      itel = itel
    ))
  }
```

```
jeffrey <- function(a) {
  h <-
    .C("jeffreyC",
       a = as.double (a),
       minwhere = as.double (0),
       minvalue = as.double (0))
  return (list.remove (h, 1))
}
```

A.1.17 inverse.R

```
imdsSolver <- function (tau) {
  radius <- sqrt (((tau - 3) ^ 2) / 3)
  center <- c(tau, tau) / 3
  a <- matrix (c(1, .5, .5, 1), 2, 2)
  draw_ellipse (
    center,
    radius,
    a,
    col = "RED",
    lwd = 2,
    xlim = c(0, tau),
    ylim = c(0, tau),
    xlab = "alpha",
    ylab = "beta"
  )
  lines (matrix(c(0, tau, tau, 0), 2, 2), col = "BLUE", lwd = 2)
  abline(h = 0)
  abline(v = 0)
}

bs <- function () {
  z <- matrix(c(1, 1, 1, 1, 1, 1, -1, -1, 1, -1, 1, -1, 1, -1, -1, 1), 4 , 4) / 2
  a <- as.list (1:6)
  k <- 1
```

```
  for (i in 1:3) {
    for (j in (i + 1):4) {
      a[[k]] <- crossprod (z, aijn(i, j, 4) %*% z)
      k <- k + 1
    }
  }
  return(a)
}

imdsChecker <- function (a) {
  aa <- bs()
  bb <- matrix(0, 4, 4)
  for (i in 1:6) {
    bb <- bb + a[i] * aa[[i]]
  }
  return (bb)
}

inverseMDS <- function (x) {
  n <- nrow (x)
  m <- ncol (x)
  x <- apply (x, 2, function (y)
    y - mean (y))
  nm <- n - (m + 1)
  kk <- cbind (1, x, matrix (rnorm (n * nm), n , nm))
  kperp <- as.matrix (qr.Q (qr (kk))[,-(1:(m + 1))])
  dd <- as.matrix (dist (x))
  k <- 1
  base <- matrix (0, n * (n - 1) / 2, nm * (nm + 1) / 2)
  for (i in 1:nm) {
    for (j in 1:i) {
      oo <- outer (kperp[, i], kperp[, j])
      if (j != i) {
        oo <- oo + t(oo)
      }
      base[, k] <- lower_triangle (dd + (1 - oo))
      k <- k + 1
```

```
      print (c(i, j, k))
    }
  }
  return (base = cbind (lower_triangle (dd), base))
}

inversePlus <- function (base, affine = TRUE) {
  if (affine) {
    hrep <- makeH (
      a1 = d2q (-base),
      b1 = d2q (rep (0, nrow (base))),
      a2 = d2q (rep (1, ncol (base))),
      b2 = d2q (1)
    )
  } else {
    hrep <- makeH (a1 = d2q (-base), b1 = d2q (rep (0, nrow (base))))
  }
  vrep <- scdd (hrep)
  hrep <- q2d (hrep)
  vrep <- q2d (vrep$output)
  pr <- tcrossprod (hrep[, -c(1, 2)], vrep[, -c(1, 2)])[-1, ]
  return (list (
    base = base,
    hrep = hrep,
    vrep = vrep,
    pr = pr
  ))
}

twoPoints <- function (x, y, w = 1 - diag (nrow (x))) {
  dx <- lower_triangle (as.matrix (dist (x)))
  dy <- lower_triangle (as.matrix (dist (y)))
  w <- lower_triangle (w)
  gx <- makeG (x)
  gy <- makeG (y)
  hx <- (dx / w) * gx
  hy <- (dy / w) * gy
```

```
  lxy <- lm.fit (cbind (hx,-hy), dx - dy)
  lxx <- lxy$coefficients[1:ncol(hx)]
  lyy <- lxy$coefficients[-(1:ncol(hx))]
  return (list(
    delta1 = dx - hx %*% lxx,
    delta2 = dy - hy %*% lyy,
    res = sum (abs(lxy$residuals)),
    rank = lxy$rank
  ))
}

second_partials_stress <-
  function (x, delta, w = wdef (nrow (x))) {
    n <- nrow (x)
    p <- ncol (x)
    d <- as.matrix (dist (x))
    fac <- (w * delta) / (d + diag (n))
    dd <- d * d
    v <- smacofVmatR (w)
    deri <- direct_sum (repList (v, p))
    xx <- as.vector (x)
    for (i in 1:(n - 1)) {
      for (j in (i + 1):n) {
        aa <- direct_sum (repList (aijn (i, j, n), p))
        ax <- drop (aa %*% xx)
        deri <- deri - fac[i, j] * (aa - outer (ax, ax) / dd[i, j])
      }
    }
    return (deri)
  }

second_partials_numerical <-
  function (x, delta, w = wdef (nrow (x))) {
    stress <- function (x, delta, w) {
      n <- nrow (delta)
      p <- length (x) / n
      d <- as.matrix(dist(matrix (x, n, p)))
```

```
      res <- delta - d
      return (sum (w * res * res) / 2)
    }
    return (hessian (stress, x, delta = delta, w = w))
  }

cleanUp <- function (a, eps = 1e-3) {
  nv <- nrow (a)
  ind <- rep (TRUE, nv)
  for (i in 1:(nv - 1)) {
    xx <- a[i, ]
    for (j in (i + 1):nv) {
      if (!ind[j])
        next
      yy <- a[j, ]
      mm <- max (abs (xx - yy))
      if (mm < eps)
        ind[j] <- FALSE
    }
  }
  return (ind)
}

bruteForce <- function (a, b, eps = 1e-3) {
  n <- nrow (a)
  m <- ncol (a)
  cb <- combn (n, m)
  n1 <- ncol (cb)
  ind <- rep(TRUE, n1)
  ht <- numeric()
  for (i in 1:n1) {
    gg <- a[cb[, i],]
    bg <- b[cb[, i]]
    qg <- qr(gg)
    if (qg$rank < m) {
      ind[i] <- FALSE
      next
```

```
    }
    hh <- solve (qg, bg)
    hg <- drop (a %*% hh)
    if (min (b - hg) < -eps) {
      ind[i] <- FALSE
      next
    }
    ht <- c(ht, hh)
  }
  n2 <- sum (ind)
  ht <- matrix (ht, m, n2)
  ind <-
    .C (
      "cleanup",
      as.double(ht),
      as.integer(n2),
      as.integer(m),
      as.integer(rep(1, n2)),
      as.double (eps)
    )[[4]]
  n3 <- sum (ind)
  return (list (
    x = t(ht)[which(ind == 1),],
    n1 = n1,
    n2 = n2,
    n3 = n3
  ))
}

bruteForceOne <- function (a, b, p, q, v, eps = 1e-3) {
  n <- nrow (a)
  m <- ncol (a)
  ind <- which ((q - v %*% p) > -eps)
  v <- v[ind,]
  cb <- combn (n, m - 1)
  n1 <- ncol (cb)
  ind <- rep(TRUE, n1)
```

```
  ht <- numeric()
  for (i in 1:n1) {
    gg <- rbind (a[cb[, i],], p)
    bg <- c (b[cb[, i]], q)
    qg <- qr(gg)
    if (qg$rank < m) {
      ind[i] <- FALSE
      next
    }
    hh <- solve (qg, bg)
    hg <- drop (a %*% hh)
    if (min (b - hg) < -eps) {
      ind[i] <- FALSE
      next
    }
    ht <- c(ht, hh)
  }
  n2 <- sum (ind)
  ht <- t (matrix (ht, m, n2))
  ht <- rbind (v, ht)
  ind <- cleanUp (ht, eps)
  print (ind)
  n3 <- sum (ind)
  return (list (
    x = ht[ind,],
    n1 = n1,
    n2 = n2,
    n3 = n3
  ))
}

rankTest <- function (x, a, b, eps = 1e-3) {
  h <- drop (a %*% x)
  ind <- which (abs (h - b) < eps)
  r <- qr (a[ind, ])$rank
  f <- min (b - h) > -eps
  return (list (rank = r, feasibility = f))
```

```
}

makeDC <- function (x) {
  y <- -x
  diag(y) <- -rowSums (y)
  return (y)
}

bmat <- function (delta, w, d) {
  n <- nrow (w)
  dd <- ifelse (d == 0, 0, 1 / d)
  return (makeDC (w * delta * dd))
}

smacof <-
  function (delta,
            w,
            xini,
            eps = 1e-6,
            itmax = 100,
            verbose = TRUE) {
    n <- nrow (xini)
    xold <- xini
    dold <- as.matrix (dist (xold))
    sold <- sum (w * (delta - dold) ^ 2) / 2
    itel <- 1
    v <- ginv (makeDC (w))
    repeat {
      b <- bmat (delta, w, dold)
      xnew <- v %*% b %*% xold
      dnew <- as.matrix (dist (xnew))
      snew <- sum (w * (delta - dnew) ^ 2) / 2
      if (verbose) {
        cat (
          formatC (itel, width = 4, format = "d"),
          formatC (
            sold,
```

```
            digits = 10,
            width = 13,
            format = "f"
          ),
          formatC (
            snew,
            digits = 10,
            width = 13,
            format = "f"
          ),
          "\n"
        )
      }
      if ((itel == itmax) || (sold - snew) < eps)
        break ()
      itel <- itel + 1
      sold <- snew
      dold <- dnew
      xold <- xnew
    }
    return (list (
      x = xnew,
      d = dnew,
      s = snew,
      itel = itel
    ))
  }

oneMore <- function (g, u) {
  v <- bruteForce (g, u)$x
  nv <- nrow (v)
  s <- matrix (0, 2, 2)
  ev <- rep (0, nv)
  for (i in 1:nv) {
    s[1, 1] <- v[i, 1]
    s[2, 2] <- v[i, 2]
    s[1, 2] <- s[2, 1] <- v[i, 3]
```

```
    ee <- eigen (s)
    ev[i] <- min (ee$values)
    if (ev[i] < 0) {
      yy <- ee$vectors[, 2]
      hh <- c (yy[1] ^ 2, yy[2] ^ 2, 2 * yy[1] * yy [2])
      g <- rbind (g,-hh)
      u <- c (u, 0)
    }
  }
  return (list (
    v = v,
    g = g,
    u = u,
    e = ev
  ))
}

makeG <- function (x) {
  n <- nrow (x)
  p <- ncol (x)
  m <- n - p - 1
  k <- qr.Q(qr(cbind(1, x, diag (n))))[,-c(1:(p + 1))]
  g <- matrix (0, n * (n - 1) / 2, m * (m + 1) / 2)
  l <- 1
  if (m == 1) {
    g[, 1] <- lower_triangle (outer (k, k))
  }
  else {
    for (i in 1:m) {
      g[, l] <- lower_triangle (outer(k[, i], k[, i]))
      l <- l + 1
    }
    for (i in 1:(m - 1))
      for (j in (i + 1):m) {
        g[, l] <-
          lower_triangle (outer(k[, i], k[, j]) + outer(k[, j], k[, i]))
        l <- l + 1
```

```
      }
  }
  return (g)
}

iStress <-
  function (x,
            delta,
            w = rep (1, length (delta)),
            only = TRUE) {
    m <- length (delta)
    n <- (1 + sqrt (1 + 8 * m)) / 2
    x <- matrix (x, n, length (x) / n)
    d <- lower_triangle (as.matrix (dist (x)))
    g <- makeG (x)
    h <- (d / w) * makeG (x)
    u <- -colSums(w * (delta - d) * h)
    v <- crossprod (h, w * h)
    s <- solve.QP (
      Dmat = v,
      dvec = u,
      Amat = -t(h),
      bvec = -d
    )
    ds <- d  - h %*% s$solution
    is <- sum (w * (delta - ds) ^ 2)
    if (only)
      return (is)
    else
      return (list (istress = is, delta = fill_symmetric (ds)))
  }
```

A.1.18 global.R

```
checkUni <- function (w, delta, x) {
  x <- drop (x)
  n <- length (x)
  vinv <- solve (smacofVmat (w) + (1 / n)) - (1 / n)
  return (drop (vinv %*% rowSums (w * delta * sign (outer (
    x, x, "-"
  )))))
}

matchMe <- function (x,
                     itmax = 100,
                     eps = 1e-10,
                     verbose = FALSE) {
  m <- length (x)
  y <- sumList (x) / m
  itel <- 1
  fold <- sum (sapply (x, function (z)
    (z - y) ^ 2))
  repeat {
    for (j in 1:m) {
      u <- crossprod (x[[j]], y)
      s <- svd (u)
      r <- tcrossprod (s$u, s$v)
      x[[j]] <- x[[j]] %*% r
    }
    y <- sumList (x) / m
    fnew <- sum (sapply (x, function (z)
      (z - y) ^ 2))
    if (verbose) {

    }
    if (((fold - fnew) < eps) || (itel == itmax))
      break
    itel <- itel + 1
    fold <- fnew
  }
  return (x)
```

```
}

sumList <- function (x) {
  m <- length (x)
  y <- x[[1]]
  for (j in 2:m) {
    y <- y + x[[j]]
  }
  return (y)
}



smacofLoss <- function (d, w, delta) {
  return (sum (w * (delta - d) ^ 2) / 4)
}

smacofBmat <- function (d, w, delta) {
  dd <- ifelse (d == 0, 0, 1 / d)
  b <- -dd * w * delta
  diag (b) <- -rowSums (b)
  return(b)
}

smacofVmat <- function (w) {
  v <- -w
  diag(v) <- -rowSums(v)
  return (v)
}

smacofGuttman <- function (x, b, vinv) {
  return (vinv %*% b %*% x)
}

columnCenter <- function (x) {
  return (apply (x, 2, function (z)
    z - mean (z)))
}
```

```
smacofComplement <- function (y, v) {
  return (sum (v * tcrossprod (y)) / 4)
}

smacofPenalty <-
  function (w,
            delta,
            p = 2,
            lbd = 0,
            zold = columnCenter (diag (nrow (delta))),
            itmax = 10000,
            eps = 1e-10,
            verbose = FALSE) {
    itel <- 1
    n <- nrow (zold)
    vmat <- smacofVmat (w)
    vinv <- solve (vmat + (1 / n)) - (1 / n)
    dold <- as.matrix (dist (zold))
    mold <- sum (w * delta * dold) / sum (w * dold * dold)
    zold <- zold * mold
    dold <- dold * mold
    yold <- zold [, (p + 1):n]
    sold <- smacofLoss (dold, w, delta)
    bold <- smacofBmat (dold, w, delta)
    told <- smacofComplement (yold, vmat)
    uold <- sold + lbd * told
    repeat {
      znew <- smacofGuttman (zold, bold, vinv)
      ynew <- znew [, (p + 1):n] / (1 + lbd)
      znew [, (p + 1):n] <- ynew
      xnew <- znew [, 1:p]
      dnew <- as.matrix (dist (znew))
      bnew <- smacofBmat (dnew, w, delta)
      tnew <- smacofComplement (ynew, vmat)
      snew <- smacofLoss (dnew, w, delta)
      unew <- snew + lbd * tnew
      if (verbose) {
```

```
        cat(
          "itel ",
          formatC(itel, width = 4, format = "d"),
          "sold ",
          formatC(
            sold,
            width = 10,
            digits = 6,
            format = "f"
          ),
          "snew ",
          formatC(
            snew,
            width = 10,
            digits = 6,
            format = "f"
          ),
          "told ",
          formatC(
            told,
            width = 10,
            digits = 6,
            format = "f"
          ),
          "tnew ",
          formatC(
            tnew,
            width = 10,
            digits = 6,
            format = "f"
          ),
          "uold ",
          formatC(
            uold,
            width = 10,
            digits = 6,
            format = "f"
```

```
          ),
          "unew ",
          formatC(
            unew,
            width = 10,
            digits = 6,
            format = "f"
          ),
          "\n"
        )
      }
      if (((uold - unew) < eps) || (itel == itmax)) {
        break
      }
      itel <- itel + 1
      zold <- znew
      bold <- bnew
      sold <- snew
      told <- tnew
      uold <- unew
    }
    zpri <- znew %*% svd(znew)$v
    xpri <- zpri[, 1:p]
    return (list (
      x = xpri,
      z = zpri,
      b = bnew,
      l = lbd,
      s = snew,
      t = tnew,
      itel = itel
    ))
  }

plotMe2 <- function(hList, labels, s = 1, t = 2) {
  n <- nrow(hList[[1]]$x)
  m <- length (hList)
```

```
  par(pty = "s")
  hMatch <- matchMe (lapply (hList, function(r)
    r$x))
  hMat <- matrix (0, 0, 2)
  for (j in 1:m) {
    hMat <- rbind(hMat, hMatch[[j]][, c(s, t)])
  }
  plot(
    hMat,
    xlab = "dim 1",
    ylab = "dim 2",
    col = c(rep("RED", n * (m - 1)), rep("BLUE", n)),
    cex = c(rep(1, n * (m - 1)), rep(2, n))
  )
  for (i in 1:n) {
    hLine <- matrix (0, 0, 2)
    for (j in 1:m) {
      hLine <- rbind (hLine, hMatch[[j]][i, c(s, t)])
    }
    lines(hLine)
  }
  text(hMatch[[m]], labels, cex = .75)
}

plotMe1 <- function(hList, labels) {
  n <- length (hList[[1]]$x)
  m <- length (hList)
  blow <- function (x) {
    n <- length (x)
    return (matrix (c(1:n, x), n, 2))
  }
  hMat <- matrix (0, 0, 2)
  for (j in 1:m) {
    hMat <- rbind(hMat, blow(hList[[j]]$x))
  }
  plot(
    hMat,
```

```
    xlab = "index",
    ylab = "x",
    col = c(rep("RED", n * (m - 1)), rep("BLUE", n)),
    cex = c(rep(1, n * (m - 1)), rep(2, n))
  )
  for (i in 1:n) {
    hLine <- matrix (0, 0, 2)
    for (j in 1:m) {
      hLine <- rbind (hLine, blow(hList[[j]]$x)[i,])
      lines(hLine)
    }
  }
  text(blow(hList[[m]]$x), labels, cex = 1.00)
  for (i in 1:n) {
    abline(h = hList[[m]]$x[i])
  }
}




runPenalty <-
  function (w,
            delta,
            lbd,
            p = 2,
            itmax = 10000,
            eps = 1e-10,
            cut = 1e-6,
            write = TRUE,
            verbose = FALSE) {
    m <- length (lbd)
    hList <- as.list (1:m)
    hList[[1]] <-
      smacofPenalty(
        w,
        delta,
```

```
      p,
      lbd = lbd[1],
      itmax = itmax,
      eps = eps,
      verbose = verbose
    )
  for (j in 2:m) {
    hList[[j]] <-
      smacofPenalty(
        w,
        delta,
        p,
        zold = hList[[j - 1]]$z,
        lbd = lbd[j],
        itmax = itmax,
        eps = eps,
        verbose = verbose
      )
  }
  mm <- m
  for (i in 1:m) {
    if (write) {
      cat(
        "itel",
        formatC(hList[[i]]$itel, width = 4, format = "d"),
        "lambda",
        formatC(
          hList[[i]]$l,
          width = 10,
          digits = 6,
          format = "f"
        ),
        "stress",
        formatC(
          hList[[i]]$s,
          width = 8,
          digits = 6,
```

```
            format = "f"
          ),
          "penalty",
          formatC(
            hList[[i]]$t,
            width = 8,
            digits = 6,
            format = "f"
          ),
          "\n"
        )
      }
      if (hList[[i]]$t < cut) {
        mm <- i
        break
      }
    }
    return(hList[1:mm])
  }

writeSelected <- function (hList, ind) {
  m <- length (hList)
  n <- length (ind)
  mn <- sort (union (union (1:3, ind), m - (2:0)))
  for (i in mn) {
    if (i > m) {
      next
    }
    cat(
      "itel",
      formatC(hList[[i]]$itel, width = 4, format = "d"),
      "lambda",
      formatC(
        hList[[i]]$l,
        width = 10,
        digits = 6,
        format = "f"
```

```
      ),
      "stress",
      formatC(
        hList[[i]]$s,
        width = 8,
        digits = 6,
        format = "f"
      ),
      "penalty",
      formatC(
        hList[[i]]$t,
        width = 8,
        digits = 6,
        format = "f"
      ),
      "\n"
    )
  }
}

rhofun <- function (a) {
  rhomax <- 0
  rhoval <- c(0, 0)
  n <- nrow (a)
  for (i in 1:n) {
    z <- a[i, 1] * xbase + a[i, 2] * ybase
    rho <- sum (delta * dist (z))
    if (rho > rhomax) {
      rhomax <- rho
      rhoval <- a[i,]
    }
  }
  return(list(rhomax = rhomax, rhoval = rhoval))
}

rhomax2Plot <- function (inpoints) {
  par(pty = "s")
```

```
s <- (0:500) / 500 * 2 * pi
x <- sin(s)
y <- cos(s)
plot(
  x,
  y,
  type = "l",
  col = "RED",
  xlim = c(-1.25, 1.25),
  ylim = c(-1.25, 1.25),
  lwd = 3
)
points(0, 0, cex = 1.2)
n <- nrow(inpoints)
outpoints <- matrix(0, n, 2)
for (i in 1:n) {
  a <- inpoints[i, ]
  if (i == n) {
    b <- inpoints[1, ]
  } else {
    b <- inpoints[i + 1, ]
  }
  d <- a[1] * b[2] - a[2] * b[1]
  outpoints[i, 1] <- (b[2] - a[2]) / d
  outpoints[i, 2] <- (a[1] - b[1]) / d
}
lines (
  x = inpoints[, 1],
  y = inpoints[, 2],
  col = "BLUE",
  lwd = 2
)
lines (
  x = inpoints[c(1, n), 1],
  y = inpoints[c(1, n), 2],
  col = "BLUE",
  lwd = 2
```

```
  )
  lines (
    x = outpoints[, 1],
    y = outpoints[, 2],
    col = "BLUE",
    lwd = 2
  )
  lines (
    x = outpoints[c(1, n), 1],
    y = outpoints[c(1, n), 2],
    col = "BLUE",
    lwd = 2
  )
}


rhomax2Comp <-
  function (inpoints,
            itmax = 100,
            eps = 1e-8,
            verbose = TRUE) {
    itel = 1
    repeat {
      n <- nrow(inpoints)
      outpoints <- matrix(0, n, 2)
      for (i in 1:n) {
        a <- inpoints[i, ]
        if (i == n) {
          b <- inpoints[1, ]
        } else {
          b <- inpoints[i + 1, ]
        }
        d <- a[1] * b[2] - a[2] * b[1]
        outpoints[i, 1] <- (b[2] - a[2]) / d
        outpoints[i, 2] <- (a[1] - b[1]) / d
      }
      infun <- rhofun (inpoints)
```

```
    oufun <- rhofun (outpoints)
    inmax <- infun$rhomax
    oumax <- oufun$rhomax
    inval <- infun$rhoval
    ouval <- oufun$rhoval
    for (i in 1:n) {
      outpoints[i, ] <- outpoints[i, ] / sqrt (sum (outpoints[i, ] ^ 2))
    }
    impoints <- matrix (0, 2 * n, 2)
    impoints[seq.int(1, (2 * n) - 1, by = 2),] <- inpoints
    impoints[seq.int(2, 2 * n, by = 2),] <- outpoints
    if (verbose) {
      cat(
        "itel ",
        formatC(itel, width = 6, format = "d"),
        "vertices ",
        formatC(n, width = 6, format = "d"),
        "innermax ",
        formatC(
          inmax,
          digits = 8,
          width = 15,
          format = "f"
        ),
        "outermax ",
        formatC(
          oumax,
          digits = 8,
          width = 15,
          format = "f"
        ),
        "\n"
      )
    }
    if ((itel == itmax) || ((oumax - inmax) < eps)) {
      break
    }
```

```
      itel <- itel + 1
      inpoints <- impoints
    }
    return (list (
      itel = itel,
      vertices = n,
      inmax = inmax,
      oumax = oumax,
      inval = inval,
      ouval = ouval
    ))
  }
```

A.1.19 mathadd.R

```
lsuw <- function (y,
                  w,
                  proj,
                  xold = rep (1, length (y)),
                  v = max (eigen (w)$values)  * diag (length (y)),
                  itmax = 100,
                  eps = 1e-6,
                  verbose = FALSE,
                  add = 1e-6) {
  f <- function (x, y, w) {
    return (sum ((x - y) * w %*% (x - y)))
  }
  n <- length (y)
  labels <- c("itel", "fold", "fnew")
  digits <- c(0, 6, 6)
  widths <- c(3, 10, 10)
  formats <- c("d", "f", "f")
  fold <- f (xold, y, w)
  itel <- 1
  repeat {
```

```
    u <- drop (solve (v, w %*% (xold - y)))
    xnew <- proj (xold - u, v)
    fnew <- f (xnew, y, w)
    if (verbose) {
      values <- c(itel, fold, fnew)
      iterWrite (labels, values, digits, widths, formats)
    }
    if ((itel == itmax) || ((fold - fnew) < eps)) {
      break
    }
    fold <- fnew
    xold <- xnew
    itel <- itel + 1
  }
  return (list (x = xnew, f = fnew, itel = itel))
}

projeq <- function (x, v) {
  s <- sum (v)
  h <- sum (x * rowSums (v))
  return (rep (h / s, length (x)))
}

projplus <- function (x, v) {
  if (!all(v == diag(diag(v)))) {
    stop ("V must be diagonal")
  }
  if (min (diag (v)) < 0) {
    stop ("V must be positive semidefinite")
  }
  return (pmax(x, 0))
}

qpmaj <-
  function (z,
            v = diag (length (z)),
            a = diff (diag (length(z))),
```

```
            b = ifelse (!is.null(a), rep(0, nrow(a)), NULL),
            c = NULL,
            d = ifelse (!is.null(c), rep(0, nrow(c)), NULL),
            h = NULL,
            itmax = 1000,
            eps = 1e-15,
            verbose = FALSE) {
  labs <- c("itel", "fold", "fnew")
  digs <- c(0, 6, 6)
  wids <- c(3, 10, 10)
  fors <- c("d", "f", "f")
  if (is.null(h)) {
    w <- v
    y <- z
    rsum <- 0
  } else {
    w <- crossprod(h, v %*% h)
    y <- drop (solve (w, crossprod (h, v %*% z)))
    rsum <- sum (z * (v %*% (z - h %*% y))) / 2
  }
  winv <- solve (w)
  if (!is.null(a)) {
    nin <- nrow(a)
    dualaa <- a %*% winv %*% t(a)
    feasa <- drop ((a %*% y) - b)
  }
  if (!is.null(c)) {
    neq <- nrow (c)
    dualcc <- c %*% winv %*% t(c)
    feasc <- drop((c %*% y) - d)
  }
  if ((!is.null(a)) && (!is.null(c))) {
    dualac <- a %*% winv %*% t(c)
    feas <- c(feasa, feasc)
    dual <- rbind(cbind(dualaa, dualac), cbind(t(dualac), dualcc))
  }
  if ((!is.null(a)) && (is.null(c))) {
```

```
    feas <- feasa
    dual <- dualaa
  }
  if ((is.null(a)) && (!is.null(c))) {
    feas <- feasc
    dual <- dualcc
  }
  vmax <- max(eigen(dual)$values)
  itel <- 1
  lold <- rep (0, nrow (dual))
  fold <-
    -(sum (lold * drop (dual %*% lold)) / 2 +
        sum (lold * feas))
  repeat {
    lnew <- lold - (drop(dual %*% lold) + feas) / vmax
    if (!is.null(a)) {
      lnew[1:nin] <- pmax(lnew[1:nin], 0)
    }
    fnew <-
      -(sum (lnew * drop (dual %*% lnew)) / 2 + sum (lnew * feas))
    if (verbose) {
      vals <- c(itel, fold, fnew)
      iterWrite (labs, vals, digs, wids, fors)
    }
    if ((itel == itmax) || ((fnew - fold) < eps)) {
      break
    }
    fold <- fnew
    lold <- lnew
    itel <- itel + 1
  }
  fdua <- fnew
  if ((!is.null(c) && (!is.null(a)))) {
    x <- y + drop (winv %*% drop(cbind(t(a), t(c)) %*% lnew))
    lb <- lnew[1:nin]
    mu <- lnew[-(1:nin)]
  }
```

```
    if ((is.null(c) && (!is.null(a)))) {
      x <- y + drop (winv %*% drop (t(a) %*% lnew))
      lb <- lnew
    }
    if ((!is.null(c) && (is.null(a)))) {
      x <- y + drop (winv %*% drop (t(c) %*% lnew))
      mu <- lnew
    }
    fprs <- sum ((x - y) * drop (w %*% (x - y))) / 2
    out <- list(x = x,
                fprimal = fprs,
                fdual = fdua)
    if (!is.null(h)) {
      out <-
        list.append(out, ftotal = fprs + rsum, predict = drop (h %*% x))
    }
    if (!is.null(a)) {
      out <-
        list.append(out, lambda = lb, inequalities = drop (a %*% x - b))
    }
    if (!is.null(c)) {
      out <- list.append(out, mu = mu, equations = drop (c %*% x - d))
    }
    return (list.append(out, itel = itel))
  }


checkIncreasing <- function (innerknots, lowend, highend) {
  h <- .C(
    "checkIncreasing",
    as.double (innerknots),
    as.double (lowend),
    as.double (highend),
    as.integer (length (innerknots)),
    fail = as.integer (0)
  )
  return (h$fail)
```

```
}

extendPartition <-
  function (innerknots,
            multiplicities,
            order,
            lowend,
            highend) {
    ninner <- length (innerknots)
    kk <- sum(multiplicities)
    nextended <- kk + 2 * order
    if (max (multiplicities) > order)
      stop ("multiplicities too high")
    if (min (multiplicities) < 1)
      stop ("multiplicities too low")
    if (checkIncreasing (innerknots, lowend, highend))
      stop ("knot sequence not increasing")
    h <-
      .C(
        "extendPartition",
        as.double (innerknots),
        as.integer (multiplicities),
        as.integer (order),
        as.integer (ninner),
        as.double (lowend),
        as.double (highend),
        knots = as.double (rep (0, nextended))
      )
    return (h)
  }

bisect <-
  function (x,
            knots,
            lowindex = 1,
            highindex = length (knots)) {
    h <- .C(
```

```
      "bisect",
      as.double (x),
      as.double (knots),
      as.integer (lowindex),
      as.integer (highindex),
      index = as.integer (0)
    )
    return (h$index)
  }

bsplines <- function (x, knots, order) {
  if ((x > knots[length(knots)]) || (x < knots[1]))
    stop ("argument out of range")
  h <- .C(
    "bsplines",
    as.double (x),
    as.double (knots),
    as.integer (order),
    as.integer (length (knots)),
    index = as.integer (0),
    q = as.double (rep(0, order))
  )
  return (list (q = h$q, index = h$ind))
}


bsplineBasis <- function (x, knots, order) {
  n <- length (x)
  k <- length (knots)
  m <- k - order
  result <- rep (0, n * m)
  h <- .C(
    "bsplineBasis",
    as.double (x),
    as.double (knots),
    as.integer (order),
    as.integer (k),
```

```
    as.integer (n),
    result = as.double (result)
  )
  return (matrix (h$result, n, m))
}

isplineBasis <-  function (x, knots, order) {
  n <- length (x)
  k <- length (knots)
  m <- k - order
  result <- rep (0, n * m)
  h <- .C(
    "isplineBasis",
    as.double (x),
    as.double (knots),
    as.integer (order),
    as.integer (k),
    as.integer (n),
    result = as.double (result)
  )
  return (matrix (h$result, n, m))
}
```

A.2 C code

A.2.1 deboor.c

```
#include <math.h>
#include <stdbool.h>
#include <stdlib.h>

inline int VINDEX(const int);
inline int MINDEX(const int, const int, const int);
inline int IMIN(const int, const int);
```

```
inline int IMIN(const int, const int);

void checkIncreasing(const double *, const double *, const double *,
                     const int *, bool *);
void extendPartition(const double *, const int *, const int *, const int *,
                     const double *, const double *, double *);
void bisect(const double *, const double *, const int *, const int *, int *);
void bsplines(const double *, const double *, const int *, const int *, int *,
              double *);
void bsplineBasis(const double *, const double *, const int *, const int *,
                  const int *, double *);
void isplineBasis(const double *, const double *, const int *, const int *,
                  const int *, double *);
void bsplineSparse(const double *, const double *, const int *, const int *,
                   const int *, int *, double *);
void isplineSparse(const double *, const double *, const int *, const int *,
                   const int *, int *, double *);

inline int VINDEX(const int i) { return i - 1; }

inline int MINDEX(const int i, const int j, const int n) {
    return (i - 1) + (j - 1) * n;
}

inline int IMIN(const int a, const int b) {
    if (a > b) return b;
    return a;
}

inline int IMAX(const int a, const int b) {
    if (a < b) return b;
    return a;
}

void checkIncreasing(const double *innerknots, const double *lowend,
                     const double *highend, const int *ninner, bool *fail) {
    *fail = false;
```

```
    if (*lowend >= innerknots[VINDEX(1)]) {
        *fail = true;
        return;
    }
    if (*highend <= innerknots[VINDEX(*ninner)]) {
        *fail = true;
        return;
    }
    for (int i = 1; i < *ninner; i++) {
        if (innerknots[i] <= innerknots[i - 1]) {
            *fail = true;
            return;
        }
    }
}

void extendPartition(const double *innerknots, const int *multiplicities,
                     const int *order, const int *ninner, const double *lowend
                     const double *highend, double *extended) {
    int k = 1;
    for (int i = 1; i <= *order; i++) {
        extended[VINDEX(k)] = *lowend;
        k++;
    }
    for (int j = 1; j <= *ninner; j++)
        for (int i = 1; i <= multiplicities[VINDEX(j)]; i++) {
            extended[VINDEX(k)] = innerknots[VINDEX(j)];
            k++;
        }
    for (int i = 1; i <= *order; i++) {
        extended[VINDEX(k)] = *highend;
        k++;
    }
}

void bisect(const double *x, const double *knots, const int *lowindex,
            const int *highindex, int *index) {
```

```
    int l = *lowindex, u = *highindex, mid = 0;
    while ((u - l) > 1) {
        mid = (int)floor((u + l) / 2);
        if (*x < knots[VINDEX(mid)])
            u = mid;
        else
            l = mid;
    }
    *index = l;
    return;
}

void bsplines(const double *x, const double *knots, const int *order,
              const int *nknots, int *index, double *q) {
    int lowindex = 1, highindex = *nknots, m = *order, j, jp1;
    double drr, dll, saved, term;
    double *dr = (double *)calloc((size_t)m, sizeof(double));
    double *dl = (double *)calloc((size_t)m, sizeof(double));
    (void)bisect(x, knots, &lowindex, &highindex, index);
    int l = *index;
    for (j = 1; j <= m; j++) {
        q[VINDEX(j)] = 0.0;
    }
    if (*x == knots[VINDEX(*nknots)]) {
        q[VINDEX(m)] = 1.0;
        return;
    }
    q[VINDEX(1)] = 1.0;
    j = 1;
    if (j >= m) return;
    while (j < m) {
        dr[VINDEX(j)] = knots[VINDEX(l + j)] - *x;
        dl[VINDEX(j)] = *x - knots[VINDEX(l + 1 - j)];
        jp1 = j + 1;
        saved = 0.0;
        for (int r = 1; r <= j; r++) {
            drr = dr[VINDEX(r)];
```

```
            dll = dl[VINDEX(jp1 - r)];
            term = q[VINDEX(r)] / (drr + dll);
            q[VINDEX(r)] = saved + drr * term;
            saved = dll * term;
        }
        q[VINDEX(jp1)] = saved;
        j = jp1;
    }
    free(dr);
    free(dl);
    return;
}

void bsplineBasis(const double *x, const double *knots, const int *order,
                  const int *nknots, const int *nvalues, double *result) {
    int m = *order, l = 0;
    double *q = (double *)calloc((size_t)m + 1, sizeof(double));
    for (int i = 1; i <= *nvalues; i++) {
        (void)bsplines(x + VINDEX(i), knots, order, nknots, &l, q);
        for (int j = 1; j <= m; j++) {
            int r = IMIN(l - m + j, *nknots - m);
            result[MINDEX(i, r, *nvalues)] = q[VINDEX(j)];
        }
    }
    free(q);
    return;
}

void isplineBasis(const double *x, const double *knots, const int *order,
                  const int *nknots, const int *nvalues, double *result) {
    int m = *nknots - *order, n = *nvalues;
    (void)bsplineBasis(x, knots, order, nknots, nvalues, result);
    for (int i = 1; i <= n; i++) {
        for (int j = m - 1; j >= 1; j--) {
            result[MINDEX(i, j, n)] += result[MINDEX(i, j + 1, n)];
        }
    }
```

```
    return;
}

void bsplineSparse(const double *x, const double *knots, const int *order,
                   const int *nknots, const int *nvalues, int *columns,
                   double *result) {
    int m = *order, l = 0;
    double *q = (double *)calloc((size_t)m + 1, sizeof(double));
    for (int i = 1; i <= *nvalues; i++) {
        (void)bsplines(x + VINDEX(i), knots, order, nknots, &l, q);
        columns[VINDEX(i)] = l;
        for (int j = 1; j <= m; j++) {
            result[MINDEX(i, j, *nvalues)] = q[VINDEX(j)];
        }
    }
    free(q);
    return;
}

void isplineSparse(const double *x, const double *knots, const int *order,
                   const int *nknots, const int *nvalues, int *columns,
                   double *result) {
    int m = *nknots - *order, n = *nvalues;
    for (int i = 1; i <= *nvalues; i++) {
        (void)bsplineSparse(x + VINDEX(i), knots, order, nknots, nvalues,
                            columns, result);
        for (int j = m - 1; j >= 1; j--) {
            result[MINDEX(i, j, n)] += result[MINDEX(i, j + 1, n)];
        }
    }
    return;
}
```

A.2.2 cleanup.c

```c
#include <math.h>

#define MAX(a, b) (((a) > (b)) ? (a) : (b))

void cleanup(double *a, int *n, int *m, int *ind, double *eps) {
    int i, j, l, ii, jj, nn = *n, mm = *m;
    double s;
    for (i = 0; i < (nn - 1); i++) {
        ii = i * mm;
        for (j = (i + 1); j < nn; j++) {
            s = 0.0;
            jj = j * mm;
            if (ind[j] == 0) continue;
            for (l = 0; l < mm; l++) {
                s = MAX(s, fabs(*(a + ii + l) - *(a + jj + l)));
            }
            if (s < *eps) {
              ind[j] = 0;
            }
        }
    }
}
```

A.2.3 jacobi.c

```c
#include "jacobi.h"

void jacobiC(const int *nn, double *a, double *evec, double *oldi, double *old
             int *itmax, double *eps) {
    int n = *nn, itel = 1;
    double d = 0.0, s = 0.0, t = 0.0, u = 0.0, v = 0.0, p = 0.0, q = 0.0,
           r = 0.0;
    double fold = 0.0, fnew = 0.0;
```

```
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            evec[MINDEX(i, j, n)] = (i == j) ? 1.0 : 0.0;
        }
    }
    for (int i = 1; i <= n; i++) {
        fold += SQUARE(a[TINDEX(i, i, n)]);
    }
    while (true) {
        for (int j = 1; j <= n - 1; j++) {
            for (int i = j + 1; i <= n; i++) {
                p = a[TINDEX(i, j, n)];
                q = a[TINDEX(i, i, n)];
                r = a[TINDEX(j, j, n)];
                if (fabs(p) < 1e-10) continue;
                d = (q - r) / 2.0;
                s = (p < 0) ? -1.0 : 1.0;
                t = -d / sqrt(SQUARE(d) + SQUARE(p));
                u = sqrt((1 + t) / 2);
                v = s * sqrt((1 - t) / 2);
                for (int k = 1; k <= n; k++) {
                    int ik = IMIN(i, k);
                    int ki = IMAX(i, k);
                    int jk = IMIN(j, k);
                    int kj = IMAX(j, k);
                    oldi[VINDEX(k)] = a[TINDEX(ki, ik, n)];
                    oldj[VINDEX(k)] = a[TINDEX(kj, jk, n)];
                }
                for (int k = 1; k <= n; k++) {
                    int ik = IMIN(i, k);
                    int ki = IMAX(i, k);
                    int jk = IMIN(j, k);
                    int kj = IMAX(j, k);
                    a[TINDEX(ki, ik, n)] =
                        u * oldi[VINDEX(k)] - v * oldj[VINDEX(k)];
                    a[TINDEX(kj, jk, n)] =
                        v * oldi[VINDEX(k)] + u * oldj[VINDEX(k)];
```

```
                }
                for (int k = 1; k <= n; k++) {
                    oldi[VINDEX(k)] = evec[MINDEX(k, i, n)];
                    oldj[VINDEX(k)] = evec[MINDEX(k, j, n)];
                    evec[MINDEX(k, i, n)] =
                        u * oldi[VINDEX(k)] - v * oldj[VINDEX(k)];
                    evec[MINDEX(k, j, n)] =
                        v * oldi[VINDEX(k)] + u * oldj[VINDEX(k)];
                }
                a[TINDEX(i, i, n)] =
                    SQUARE(u) * q + SQUARE(v) * r - 2 * u * v * p;
                a[TINDEX(j, j, n)] =
                    SQUARE(v) * q + SQUARE(u) * r + 2 * u * v * p;
                a[TINDEX(i, j, n)] =
                    u * v * (q - r) + (SQUARE(u) - SQUARE(v)) * p;
            }
        }
        fnew = 0.0;
        for (int i = 1; i <= n; i++) {
            fnew += SQUARE(a[TINDEX(i, i, n)]);
        }
        if (((fnew - fold) < *eps) || (itel == *itmax)) break;
        fold = fnew;
        itel++;
    }
    return;
}

void primat(const int *n, const int *m, const int *w, const int *p,
            const double *x) {
    for (int i = 1; i <= *n; i++) {
        for (int j = 1; j <= *m; j++) {
            printf(" %*.*f ", *w, *p, x[MINDEX(i, j, *n)]);
        }
        printf("\n");
    }
    printf("\n\n");
```

```
    return;
}

void pritru(const int *n, const int *w, const int *p, const double *x) {
    for (int i = 1; i <= *n; i++) {
        for (int j = 1; j <= i; j++) {
            printf(" %*.*f ", *w, *p, x[TINDEX(i, j, *n)]);
        }
        printf("\n");
    }
    printf("\n\n");
    return;
}

void trimat(const int *n, const double *x, double *y) {
    int nn = *n;
    for (int i = 1; i <= nn; i++) {
        for (int j = 1; j <= nn; j++) {
            y[MINDEX(i, j, nn)] =
                (i >= j) ? x[TINDEX(i, j, nn)] : x[TINDEX(j, i, nn)];
        }
    }
    return;
}

void mattri(const int *n, const double *x, double *y) {
    int nn = *n;
    for (int j = 1; j <= nn; j++) {
        for (int i = j; i <= nn; i++) {
            y[TINDEX(i, j, nn)] = x[MINDEX(i, j, nn)];
        }
    }
    return;
}
```

A.2.4 jbkTies.c

```c
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define DEBUG false

struct block {
    double value;
    double weight;
    int size;
    int previous;
    int next;
};

struct quadruple {
    double value;
    double result;
    double weight;
    int index;
};

struct triple {
    double value;
    double weight;
    int index;
};

struct pair {
    int value;
    int index;
};

int myCompDouble(const void *, const void *);
```

```
int myCompInteger(const void *, const void *);
void mySortDouble(double *, double *, double *, int *, const int *);
void mySortInteger(int *, int *, const int *);
void mySortInBlock(double *, double *, int *, int *);
void tieBlock(double *, int *, const int *, int *);
void makeBlocks(double *, double *, double *, double *, int *, const int *,
                const int *);
void sortBlocks(double *, double *, int *, const int *, const int *,
                const int *);
void jbkPava(double *, double *, const int *);
void primary(double *, double *, int *, int *, const int *, const int *);
void secondary(double *, double *, int *, const int *, const int *);
void tertiary(double *, double *, int *, const int *, const int *);
void monreg(double *, double *, double *, const int *, const int *);

int myCompDouble(const void *px, const void *py) {
    double x = ((struct quadruple *)px)->value;
    double y = ((struct quadruple *)py)->value;
    return (int)copysign(1.0, x - y);
}

int myCompInteger(const void *px, const void *py) {
    int x = ((struct pair *)px)->value;
    int y = ((struct pair *)py)->value;
    return (int)copysign(1.0, x - y);
}

void mySortInBlock(double *x, double *w, int *xind, int *n) {
    int nn = *n;
    struct triple *xi =
        (struct triple *)calloc((size_t)nn, (size_t)sizeof(struct triple));
    for (int i = 0; i < nn; i++) {
        xi[i].value = x[i];
        xi[i].weight = w[i];
        xi[i].index = xind[i];
    }
    (void)qsort(xi, (size_t)nn, (size_t)sizeof(struct triple), myCompDouble);
```

```
    for (int i = 0; i < nn; i++) {
        x[i] = xi[i].value;
        w[i] = xi[i].weight;
        xind[i] = xi[i].index;
    }
    free(xi);
    return;
}

void mySortDouble(double *x, double *y, double *w, int *xind, const int *n) {
    int nn = *n;
    struct quadruple *xi = (struct quadruple *)calloc(
        (size_t)nn, (size_t)sizeof(struct quadruple));
    for (int i = 0; i < nn; i++) {
        xi[i].value = x[i];
        xi[i].result = y[i];
        xi[i].weight = w[i];
        xi[i].index = i + 1;
    }
    (void)qsort(xi, (size_t)nn, (size_t)sizeof(struct quadruple), myCompDouble
    for (int i = 0; i < nn; i++) {
        x[i] = xi[i].value;
        y[i] = xi[i].result;
        w[i] = xi[i].weight;
        xind[i] = xi[i].index;
    }
    free(xi);
    return;
}

void mySortInteger(int *x, int *k, const int *n) {
    int nn = *n;
    struct pair *xi =
        (struct pair *)calloc((size_t)nn, (size_t)sizeof(struct pair));
    for (int i = 0; i < nn; i++) {
        xi[i].value = x[i];
        xi[i].index = i + 1;
```

```
    }
    (void)qsort(xi, (size_t)nn, (size_t)sizeof(struct pair), myCompInteger);
    for (int i = 0; i < nn; i++) {
        x[i] = xi[i].value;
        k[i] = xi[i].index;
    }
    free(xi);
    return;
}

void tieBlock(double *x, int *iblks, const int *n, int *nblk) {
    iblks[0] = 1;
    for (int i = 1; i < *n; i++) {
        if (x[i - 1] == x[i]) {
            iblks[i] = iblks[i - 1];
        } else {
            iblks[i] = iblks[i - 1] + 1;
        }
    }
    *nblk = iblks[*n - 1];
    return;
}

void makeBlocks(double *x, double *w, double *xblks, double *wblks, int *iblks,
                const int *n, const int *nblk) {
    for (int i = 0; i < *nblk; i++) {
        xblks[i] = 0.0;
        wblks[i] = 0.0;
    }
    for (int i = 0; i < *n; i++) {
        xblks[iblks[i] - 1] += w[i] * x[i];
        wblks[iblks[i] - 1] += w[i];
    }
    for (int i = 0; i < *nblk; i++) {
        xblks[i] = xblks[i] / wblks[i];
    }
    return;
```

```
}

void sortBlocks(double *y, double *w, int *xind, const int *iblks, const int *
                const int *nblk) {
    int *nblks = (int *)calloc((size_t)*nblk, sizeof(int));
    for (int i = 0; i < *n; i++) {
        nblks[iblks[i] - 1]++;
    }
    int k = 0;
    for (int i = 0; i < *nblk; i++) {
        int nn = nblks[i];
        (void)mySortInBlock(y + k, w + k, xind + k, &nn);
        k += nn;
    }
    free(nblks);
    return;
}

void jbkPava(double *x, double *w, const int *n) {
    struct block *blocks = calloc((size_t)*n, sizeof(struct block));
    for (int i = 0; i < *n; i++) {
        blocks[i].value = x[i];
        blocks[i].weight = w[i];
        blocks[i].size = 1;
        blocks[i].previous = i - 1;  // index first element previous block
        blocks[i].next = i + 1;      // index first element next block
    }
    int active = 0;
    do {
        bool upsatisfied = false;
        int next = blocks[active].next;
        if (next == *n)
            upsatisfied = true;
        else if (blocks[next].value > blocks[active].value)
            upsatisfied = true;
        if (!upsatisfied) {
            double ww = blocks[active].weight + blocks[next].weight;
```

```
            int nextnext = blocks[next].next;
            blocks[active].value =
                (blocks[active].weight * blocks[active].value +
                 blocks[next].weight * blocks[next].value) /
                ww;
            blocks[active].weight = ww;
            blocks[active].size += blocks[next].size;
            blocks[active].next = nextnext;
            if (nextnext < *n) blocks[nextnext].previous = active;
            blocks[next].size = 0;
        }
        bool downsatisfied = false;
        int previous = blocks[active].previous;
        if (previous == -1)
            downsatisfied = true;
        else if (blocks[previous].value < blocks[active].value)
            downsatisfied = true;
        if (!downsatisfied) {
            double ww = blocks[active].weight + blocks[previous].weight;
            int previousprevious = blocks[previous].previous;
            blocks[active].value =
                (blocks[active].weight * blocks[active].value +
                 blocks[previous].weight * blocks[previous].value) /
                ww;
            blocks[active].weight = ww;
            blocks[active].size += blocks[previous].size;
            blocks[active].previous = previousprevious;
            if (previousprevious > -1) blocks[previousprevious].next = active;
            blocks[previous].size = 0;
        }
        if ((blocks[active].next == *n) && downsatisfied) break;
        if (upsatisfied && downsatisfied) active = next;
    } while (true);
    int k = 0;
    for (int i = 0; i < *n; i++) {
        int blksize = blocks[i].size;
        if (blksize > 0.0) {
```

```
            for (int j = 0; j < blksize; j++) {
                x[k] = blocks[i].value;
                k++;
            }
        }
    }
    free(blocks);
}
```

A.2.5 matrix.c

```
#include "smacof.h"

void gsC(double *x, double *r, int *n, int *m, int *rank, int *pivot,
         double *eps) {
    int i, j, ip, nn = *n, mm = *m, rk = *m, jwork = 1;
    double s = 0.0, p;
    for (j = 1; j <= mm; j++) {
        pivot[j - 1] = j;
    }
    while (jwork <= rk) {
        for (j = 1; j < jwork; j++) {
            s = 0.0;
            for (i = 1; i <= nn; i++) {
                s += x[MINDEX(i, jwork, nn)] * x[MINDEX(i, j, nn)];
            }
            r[MINDEX(j, jwork, mm)] = s;
            for (i = 1; i <= nn; i++) {
                x[MINDEX(i, jwork, nn)] -= s * x[MINDEX(i, j, nn)];
            }
        }
        s = 0.0;
        for (i = 1; i <= nn; i++) {
            s += x[MINDEX(i, jwork, nn)] * x[MINDEX(i, jwork, nn)];
        }
```

```
        if (s > *eps) {
            s = sqrt(s);
            r[MINDEX(jwork, jwork, mm)] = s;
            for (i = 1; i <= nn; i++) {
                x[MINDEX(i, jwork, nn)] /= s;
            }
            jwork += 1;
        }
        if (s <= *eps) {
            ip = pivot[rk - 1];
            pivot[rk - 1] = pivot[jwork - 1];
            pivot[jwork - 1] = ip;
            for (i = 1; i <= nn; i++) {
                p = x[MINDEX(i, rk, nn)];
                x[MINDEX(i, rk, nn)] = x[MINDEX(i, jwork, nn)];
                x[MINDEX(i, jwork, nn)] = p;
            }
            for (j = 1; j <= mm; j++) {
                p = r[MINDEX(j, rk, mm)];
                r[MINDEX(j, rk, mm)] = r[MINDEX(j, jwork, mm)];
                r[MINDEX(j, jwork, mm)] = p;
            }
            rk -= 1;
        }
    }
    *rank = rk;
}
```

A.2.6 jeffrey.c

```
// This implements the formulas in
//
// D.J. Jeffrey
// Formulae, Algorithms, and Quartic Extrema
// Mathematics Magazine, 1997, 70(5), 341-348
```

```c
//
// to find the minimum of a quartic polynomial.

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define SQ(x) ((x) * (x))
#define CU(x) ((x) * (x) * (x))
#define QU(x) ((x) * (x) * (x) * (x))

void smacofJeffrey(double *a, double *minwhere, double *minvalue) {
    if (a[4] <= 0.0) {
        printf("Quartic term must be positive. Exiting...\n");
        exit (EXIT_FAILURE);
    }
    double b0 = 0.0, b1 = 0.0, b2 = 0.0;
    b2 += a[2] / (3.0 * a[4]);
    b2 -= (SQ(a[3])) / (8.0 * SQ(a[4]));
    b1 += a[1] / (2.0 * a[4]);
    b1 += CU(a[3]) / (16.0 * CU(a[4]));
    b1 -= (a[2] * a[3]) / (4.0 * SQ(a[4]));
    b0 += (a[2] * SQ(a[3])) / (16.0 * SQ(a[4]));
    b0 -= (a[1] * a[3]) / (4.0 * a[4]);
    b0 -= (3.0 * QU(a[3])) / (256.0 * CU(a[4]));
    if (fabs(b1) > 1e-15) {
        double s = SQ(b1) + CU(b2) + sqrt(QU(b1) + 2 * SQ(b1) * CU(b2));
        double t = pow(s, (double) 1 / 3);
        double k = t + (SQ(b2) / t) + b2;
        double infp = -0.75 * (k - b2) * (k - 3.0 * b2);
        *minwhere = -b1 / k - 0.25 * a[3] / a[4];
        *minvalue = infp * a[4] + a[0] + b0;
    } else {
        double infp = -2.25 * SQ(fmin(0.0, b2));;
        *minwhere = sqrt(-fmin(0.0, 1.5 * b2));
        *minvalue = infp * a[4] + a[0] + b0;
    }
```

```
    return;
}
```

A.2.7 smacofBlockSort.c

```
/*

smacofBlockSort() is a routine to transform a matrix of dissimilarities (in
lower-triangular column-major storage) into a ordinal multidimensional scaling
structure (OMDS structure). An OMDS structure is an array of tie-blocks, where
each tie-block corresponds with a unique dissimilarity value. Tie-blocks have a
value, a size, a vector of weights, and a vector of indices. They are strictly
ordered by increasing value. The routine is written for the smacof project,
but it can be used as a preprocessor for any monotone regression problem.

*/

#include "../include/smacof.h"


int sortComp(const void *px, const void *py) {
    double x = ((struct triple *)px)->value;
    double y = ((struct triple *)py)->value;
    return (int)copysign(1.0, x - y);
}

void smacofBlockSort(const double *x, const double *w, const int n, int nblock,
                     block *yi) {
    triple *xi = (triple *)calloc((size_t)n, sizeof(triple));
    yi = (struct block *)calloc((size_t)n, (size_t)sizeof(block));
    for (int i = 0; i < n; i++) {
        xi[i].value = x[i];
        xi[i].weight = w[i];
        xi[i].index = i;
    }
```

```
    (void)qsort(xi, (size_t)n, (size_t)sizeof(triple), sortComp);
    int counter = 0;
    while (counter < n) {
        double value = xi[counter].value;
        int size = 0;
        for (int j = counter; j < n; j++) {
            if (xi[j].value == value) {
                size += 1;
            } else {
                break;
            }
        }
        yi[nblock].size = size;
        yi[nblock].value = value;
        yi[nblock].indices = (int *)calloc((size_t)size, sizeof(int));
        yi[nblock].weights = (double *)calloc((size_t)size, sizeof(double));
        for (int i = 0; i < size; i++) {
            yi[nblock].indices[i] = xi[counter + i].index;
            yi[nblock].weights[i] = xi[counter + i].weight;
        }
        counter += size;
        printf("nblock %4d value %4.1f size %4d counter %4d", nblock,
               yi[nblock].value, yi[nblock].size, counter);
        printf("\nindices");
        for (int i = 0; i < size; i++) {
            printf("%4d", yi[nblock].indices[i] + 1);
        }
        printf("\nweights");
        for (int i = 0; i < size; i++) {
            printf("%4.1f", yi[nblock].weights[i]);
        }
        printf("\n");
        if (counter == n) {
            break;
        } else {
            nblock++;
        }
```

```
    }
    yi = (block *)realloc(yi, (size_t)nblock * sizeof(block));
    free(xi);
    return;
}

/*
double x1[10] = {3.0, 1.0, 1.0, 5.0, 1.0, 5.0, 1.0, 2.0, 5.0, 2.0};
double x2[10] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
double x3[10] = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};
double ww[10] = {1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0};
int n = 10;

int main() {
    block *yi = NULL;
    int nblock = 1;
    (void)smacofBlockSort(x1, ww, n, nblock, yi);
    printf("**********************************\n");
    (void)smacofBlockSort(x2, ww, n, nblock, yi);
    printf("**********************************\n");
    (void)smacofBlockSort(x3, ww, n, nblock, yi);
    free(yi);
}
*/
```

A.2.8 smacofConvert.c

```
#include "../include/smacof.h"

void smacofEncode(const int *ip, const int *jp, const int *np, int *kp) {
    int i = *ip, j = *jp, n = *np;
    *kp = i + (j - 1) * n - j * (j + 1) / 2;
    return;
}
```

```
void smacofDecode(const int *kp, const int *np, int *ip, int *jp) {
    int j = 1, m = 1, k = *kp, n = *np;
    while (k >= ((j * n) - m + 1)) {
        j += 1;
        m += j;
    }
    *ip = k - (j - 1) * n + m;
    *jp = j;
    return;
}









/*
int main(void) {
    int i = 0, j = 0, k = 0, n = 6;
    printf("ENCODE\n\n");
    for (j = 1; j < n; j++) {
        for (i = (j + 1); i <= n; i++) {
            (void)dencode(&i, &j, &n, &k);
            printf(" %4d ", k);
        }
    }
    printf("\n\n");
    printf("DECODE\n\n");
    for (int k = 1; k <= n * (n - 1) / 2; k++) {
        (void)ddecode(&k, &n, &i, &j);
        printf("%4d %4d\n", i, j);
    }
    return EXIT_SUCCESS;
}
*/
```

A.2.9 nextPC.c

```c
void swap(int* x, int i, int j) {
    int temp;
    temp = x[i];
    x[i] = x[j];
    x[j] = temp;
}

void nextPermutation(int* x, int* nn) {
    int i, j, n = *nn;
    i = n - 1;
    while (x[i - 1] >= x[i]) i--;
    if (i == 0) return;
    j = n;
    while (x[j - 1] <= x[i - 1]) j--;
    swap(x, i - 1, j - 1);
    j = n;
    i++;
    while (i < j) {
        swap(x, i - 1, j - 1);
        j--;
        i++;
    }
}

void nextCombination(int* n, int* m, int* next) {
    int i, j, mm = *m - 1, nn = *n;
    for (i = mm; i >= 0; i--) {
        if (next[i] != nn - mm + i) {
            next[i]++;
            if (i < mm) {
                for (j = i + 1; j <= mm; j++) next[j] = next[j - 1] + 1;
            }
            return;
        }
```

```
    }
}
```

Appendix B

Data

B.1 Small

```
##   1 2 3
## 2 1
## 3 2 4
## 4 5 2 1
```

B.2 De Gruijter

```
##              KVP       PvdA        VVD        ARP        CHU        CPN
## KVP  0.00000000 0.10473553 0.09803841 0.08557432 0.08929495 0.14026748
## PvdA 0.10473553 0.00000000 0.12501293 0.10492156 0.11571137 0.09524794
## VVD  0.09803841 0.12501293 0.00000000 0.10157300 0.09245748 0.15124332
## ARP  0.08557432 0.10492156 0.10157300 0.00000000 0.05952996 0.14584841
## CHU  0.08929495 0.11571137 0.09245748 0.05952996 0.00000000 0.14510429
## CPN  0.14026748 0.09524794 0.15124332 0.14584841 0.14510429 0.00000000
## PSP  0.12519896 0.08538829 0.14045351 0.12519896 0.13171005 0.07590070
## BP   0.13357036 0.13431448 0.12836149 0.13543067 0.12947767 0.11794374
## D66  0.11478121 0.10175903 0.08687654 0.11403709 0.11236281 0.13803510
##              PSP        BP        D66
## KVP  0.12519896 0.1335704 0.11478121
## PvdA 0.08538829 0.1343145 0.10175903
```

```
## VVD  0.14045351 0.1283615 0.08687654
## ARP  0.12519896 0.1354307 0.11403709
## CHU  0.13171005 0.1294777 0.11236281
## CPN  0.07590070 0.1179437 0.13803510
## PSP  0.00000000 0.1279894 0.11831580
## BP   0.12798942 0.0000000 0.13691892
## D66  0.11831580 0.1369189 0.00000000
```

B.3 Ekman

B.4 Vegetables

```
veg <- abs (qnorm (matrix (c(.500,.818,.770,.811,.878,.892,.899,.892,.926,
.182,.500,.601,.723,.743,.736,.811,.845,.858,
.230,.399,.500,.561,.736,.676,.845,.797,.818,
.189,.277,.439,.500,.561,.588,.676,.601,.730,
.122,.257,.264,.439,.500,.493,.574,.709,.764,
.108,.264,.324,.412,.507,.500,.628,.682,.628,
.101,.189,.155,.324,.426,.372,.500,.527,.642,
.108,.155,.203,.399,.291,.318,.473,.500,.628,
.074,.142,.182,.270,.236,.372,.358,.372,.500), 9, 9)))
```

Appendix C

Unlicense

This book is free and unencumbered text and software released into the public domain.

Anyone is free to copy, modify, publish, use, compile, sell, or distribute this book, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means.

In jurisdictions that recognize copyright laws, the author of this book dedicates any and all copyright interest in the software to the public domain. I make this dedication for the benefit of the public at large and to the detriment of my heirs and successors. I intend this dedication to be an overt act of relinquishment in perpetuity of all present and future rights to this book under copyright law.

THE BOOK IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE BOOK OR THE USE OR OTHER DEALINGS IN THE BOOK.

For more information, please refer to http://unlicense.org/

References

Avis, D. 2015. *User's Guide for lrs - Version 6.1*. The Computational Geometry Lab at McGill. http://cgm.cs.mcgill.ca/~avis/C/lrslib/USERGUIDE.html.

Bacak, M., and J. M. Borwein. 2011. "On Difference Convexity of Locally Lipschitz Functions." *Optimization* 60 (8-9): 961–78.

Bailey, R. A., and J. C. Gower. 1990. "Approximating a Symmetric Matrix." *Psychometrika* 55 (4): 665–75.

Bauschke, H. H., M. N. Bui, and X. Wang. 2018. "Projecting onto the Intersection of a Cone and a Sphere." *SIAM Journal on Optimization* 28: 2158–88.

Bavaud, F. 2011. "On the Schoenberg Transformations in Data Analysis: Theory and Illustrations." *Journal of Classification* 28: 297–314.

Berge, C. 1963. *Topological Spaces*. Oliver & Boyd.

Berkelaar, M. and others. 2015. *lpSolve: Interface to 'Lp_solve' v. 5.5 to Solve Linear/Integer Programs*. %7Bhttps://CRAN.R-project.org/package=lpSolve%7D.

Best, M. J. 2017. *Quadratic Programming with Computer Programs*. Advances in Applied Mathematics. CRC Press.

Blumenthal, L. M. 1953. *Theory and Applications of Distance Geometry*. Oxford University Press.

Borg, I., and P. J. F. Groenen. 2005. *Modern Multidimensional Scaling*. Second Edition. Springer.

Borg, I., and D. Leutner. 1983. "Dimensional Models for the Perception of Rectangles." *Perception and Psychophysics* 34: 257–69.

Borg, I., and J. C. Lingoes. 1980. "A Model and Algorithm for Multidimensional Scaling with External Constraints on the Distances." *Psychometrika* 45 (1): 25–38.

Browne, M. W. 1987. "The Young-Householder Algorithm and the Least

Squares Multdimensional Scaling of Squared Distances." *Journal of Classification* 4: 175–90.

Busing, F. M. T. A. 2021. "Monotone Regression: A Simple and Fast O(n) PAVA Implementation." *Journal of Statiistical Software* Submitted.

Cailliez, F. 1983. "The Analytical Solution to the Additive Constant Problem." *Psychometrika* 48 (2): 305–8.

Carroll, J. D., and J. J. Chang. 1970. "Analysis of Individual Differences in Multidimensional scaling via an N-way generalization of "Eckart-Young" Decomposition." *Psychometrika* 35: 283–319.

Ciomek, K. 2017. *hasseDiagram: Drawing Hasse Diagram*. %7Bhttps://CRAN.R-project.org/package=hasseDiagram%7D.

Coombs, C. H. 1964. *A Theory of Data*. Wiley.

Cooper, L. G. 1972. "A New Solution to the Additive Constant Problem in Metric Multidimensional Scaling." *Psychometrika* 37 (3): 311–22.

Cox, D. R., and L. Brandwood. 1959. "On a Discriminatory Problem Connected with the Works of Plato." *Journal of the Royal Statistical Society, Series B* 21: 195–200.

Cox, M. G. 1972. "The Numerical Evaluation of B-splines." *Journal of the Institute of Mathematics and Its Applications* 10: 134–49.

Cox, T. F., and M. A. A. Cox. 1991. "Multidimensional Scaling on a Sphere." *Communications in Statistics* 20: 2943–53.

———. 2001. *Miltidimensional Scaling*. Second Edition. Monographs on Statistics and Applied Probability 88. Chapman & Hall.

Critchley, F. 1988. "On Certain Linear Mappings Between Inner Product and Squared Distance Matrices." *Linear Algebra and Its Applications* 105: 91–107.

Danskin, J. M. 1967. *The Theory of Max-Min and Its Application to Weapons Allocation Problems*. Springer.

Datorro, J. 2015. *Convex Optimization and Distance Geometry*. Second Edition. Palo Alto, CA: Meebo Publishing.

De Boor, C. 1972. "On Calculating with B-splines. II. Integration." *Journal of Approximation Theory* 6: 50–62.

———. 1976. "Splines as Linear Combination of B-splines. A Survey." In *Approximation Theory II*, edited by G. G. Lorentz, C. K. Chui, and L. L. Schumaker, 1–47. Academic Press.

———. 2001. *A Practical Guide to Splines*. Revised Edition. New York: Springer-Verlag.

De Boor, C., and K. Höllig. 1985. "B-splines without Divided Differences."

Technical Report 622. Department of Computer Science, University of Wisconsin-Madison.

De Boor, C., T. Lyche, and L. L. Schumaker. 1976. "On Calculating with B-splines. II. Integration." In *Numerische Methoden der Approximationstheorie*, edited by L. Collatz, G. Meinardus, and H. Werner, 123–46. Basel: Birkhauser.

De Gruijter, D. N. M. 1967. "The Cognitive Structure of Dutch Political Parties in 1966." Report E019-67. Psychological Institute, University of Leiden.

De Leeuw, J. 1968a. "Canonical Discriminant Analysis of Relational Data." Research Note 007-68. Department of Data Theory FSW/RUL.

———. 1968b. "Nonmetric Discriminant Analysis." Research Note 06-68. Department of Data Theory, University of Leiden.

———. 1968c. "Nonmetric Multidimensional Scaling." Research Note 010-68. Department of Data Theory FSW/RUL.

———. 1969. "Some Contributions to the Analysis of Categorical Data." Research Note 004-69. Leiden, The Netherlands: Department of Data Theory FSW/RUL.

———. 1970. "The Euclidean Distance Model." Research Note 002-70. Department of Data Theory FSW/RUL.

———. 1973a. "Canonical Analysis of Categorical Data." PhD thesis, University of Leiden, The Netherlands.

———. 1973b. "Smoothness Properties of Nonmetric Loss Functions." Technical Memorandum. Murray Hill, N.J.: Bell Telephone Laboratories.

———. 1974. "Approximation of a Real Symmetric Matrix by a Positive Semidefinite Matrix of Rank r." Technical Memorandum TM-74-1229-10. Murray Hill, N.J.: Bell Telephone Laboratories.

———. 1975a. "A Normalized Cone Regression Approach to Alternating Least Squares Algorithms." Department of Data Theory FSW/RUL.

———. 1975b. "An Alternating Least Squares Approach to Squared Distance Scaling." Department of Data Theory FSW/RUL.

———. 1977a. "Applications of Convex Analysis to Multidimensional Scaling." In *Recent Developments in Statistics*, edited by J. R. Barra, F. Brodeau, G. Romier, and B. Van Cutsem, 133–45. Amsterdam, The Netherlands: North Holland Publishing Company.

———. 1977b. "Correctness of Kruskal's Algorithms for Monotone Regression with Ties." *Psychometrika* 42: 141–44.

———. 1984a. *Canonical Analysis of Categorical Data.* Leiden, The Nether-

lands: DSWO Press.

———. 1984b. "Convergence of Majorization Algorithms for Multidimensional Scaling." Research Note RR-84-07. Leiden, The Netherlands: Department of Data Theory FSW/RUL.

———. 1984c. "Differentiability of Kruskal's Stress at a Local Minimum." *Psychometrika* 49: 111–13.

———. 1984d. "Fixed-Rank Approximation with Singular Weight Matrices." *Computational Statistics Quarterly* 1: 3–12.

———. 1988. "Convergence of the Majorization Method for Multidimensional Scaling." *Journal of Classification* 5: 163–80.

———. 1993. "Fitting Distances by Least Squares." Preprint Series 130. Los Angeles, CA: UCLA Department of Statistics.

———. 1994. "Block Relaxation Algorithms in Statistics." In *Information Systems and Data Analysis*, edited by H. H. Bock, W. Lenski, and M. M. Richter, 308–24. Berlin: Springer Verlag.

———. 2005a. "Fitting Ellipsoids by Least Squares." UCLA Department of Statistics.

———. 2005b. "Unidimensional Scaling." In *The Encyclopedia of Statistics in Behavioral Science*, edited by B. S. Everitt and D. C, 4:2095–97. New York, N.Y.: Wiley.

———. 2006. "Accelerated Least Squares Multidimensional Scaling." Preprint Series 493. Los Angeles, CA: UCLA Department of Statistics.

———. 2007a. "A Horseshoe for Multidimensional Scaling." Preprint Series 530. Los Angeles, CA: UCLA Department of Statistics.

———. 2007b. "Derivatives of Generalized Eigen Systems with Applications." Preprint Series 528. Los Angeles, CA: UCLA Department of Statistics.

———. 2007c. "Quadratic Surface Embedding." UCLA Department of Statistics.

———. 2008a. "Accelerating Majorization Algorithms." Preprint Series 543. Los Angeles, CA: UCLA Department of Statistics.

———. 2008b. "Derivatives of Fixed-Rank Approximations." Preprint Series 547. Los Angeles, CA: UCLA Department of Statistics.

———. 2008c. "Polynomial Extrapolation to Accelerate Fixed Point Algorithms." Preprint Series 542. Los Angeles, CA: UCLA Department of Statistics.

———. 2012. "Inverse Multidimensional Scaling." UCLA Department of Statistics.

———. 2014a. “Bounding, and Sometimes Finding, the Global Minimum in Multidimensional Scaling.” UCLA Department of Statistics.
———. 2014b. “Minimizing rStress Using Nested Majorization.” UCLA Department of Statistics.
———. 2015. “Regression with Linear Inequality Restrictions on Predicted Values.”
———. 2016a. “Derivatives of Low Rank PSD Approximation.” 2016.
———. 2016b. “Gower Rank.” 2016.
———. 2016c. “Pictures of Stress.” 2016.
———. 2017a. “Computing and Fitting Monotone Splines.” 2017.
———. 2017b. “Multidimensional Scaling with Distance Bounds.” 2017.
———. 2017c. “Multidimensional Scaling with Lower Bounds.” 2017.
———. 2017d. “Multidimensional Scaling with Upper Bounds.” 2017.
———. 2017e. “Shepard Non-metric Multidimensional Scaling.” 2017.
———. 2017f. “Tweaking the SMACOF Engine.” 2017.
———. 2018a. “Differentiability of Stress at Local Minima.” 2018.
———. 2018b. “MM Algorithms for Smoothed Absolute Values.” 2018.
———. 2019. “Normalized Cone Regression.” 2019.
———. 2021. *Block Relaxation Methods in Statistics*. Bookdown.
De Leeuw, J., and B. Bettonvil. 1986. “An Upper Bound for SSTRESS.” *Psychometrika* 51: 149–53.
De Leeuw, J., P. Groenen, and P. Mair. 2016a. “Full-Dimensional Scaling.” 2016.
———. 2016b. “Minimizing qStress for Small q.” 2016.
———. 2016c. “Minimizing rStress Using Majorization.” 2016.
———. 2016d. “More on Inverse Multidimensional Scaling.” 2016.
———. 2016e. “Second Derivatives of rStress, with Applications.” 2016.
———. 2016f. “Singularities and Zero Distances in Multidimensional Scaling.” 2016.
De Leeuw, J., P. Groenen, and R. Pietersz. 2016. “An Alternating Least Squares Approach to Squared Distance Scaling.” 2016.
De Leeuw, J., and W. J. Heiser. 1977. “Convergence of Correction Matrix Algorithms for Multidimensional Scaling.” In *Geometric Representations of Relational Data*, edited by J. C. Lingoes, 735–53. Ann Arbor, Michigan: Mathesis Press.
———. 1980. “Multidimensional Scaling with Restrictions on the Configuration.” In *Multivariate Analysis, Volume V*, edited by P. R. Krishnaiah, 501–22. Amsterdam, The Netherlands: North Holland Publishing Com-

pany.

———. 1982. "Theory of Multidimensional Scaling." In *Handbook of Statistics, Volume II*, edited by P. R. Krishnaiah and L. Kanal. Amsterdam, The Netherlands: North Holland Publishing Company.

De Leeuw, J., K. Hornik, and P. Mair. 2009. "Isotone Optimization in R: Pool-Adjacent-Violators Algorithm (PAVA) and Active Set Methods." *Journal of Statistical Software* 32 (5): 1–24.

De Leeuw, J., and P. Mair. 2009. "Multidimensional Scaling Using Majorization: SMACOF in R." *Journal of Statistical Software* 31 (3): 1–30.

De Leeuw, J., and J. J. Meulman. 1986. "Principal Component Analysis and Restricted Multidimensional Scaling." In *Classification as a Tool of Research*, edited by W. Gaul and M. Schader, 83–96. Amsterdam, London, New York, Tokyo: North-Holland.

De Leeuw, J., and K. Sorenson. 2012. "Derivatives of the Procrustus Transformation with Applications."

De Leeuw, J., and I. Stoop. 1984. "Upper Bounds for Kruskal's Stress." *Psychometrika* 49: 391–402.

Defays, D. 1978. "A Short Note on a Method of Seriation." *British Journal of Mathematical and Statistical Psychology* 31: 49–53.

Delfour, M. C. 2012. *Introduction to Optimization and Semidifferential Calculus*. SIAM.

Dempster, A. P., N. M. Laird, and D. B. Rubin. 1977. "Maximum Likelihood for Incomplete Data via the EM Algorithm." *Journal of the Royal Statistical Society* B39: 1–38.

Demyanov, V. F., and V. N. Malozemov. 1990. *Introduction to Minimax*. Dover.

Dinkelbach, W. 1967. "On Nonlinear Fractional Programming." *Management Science* 13: 492–98.

Du, S., C. Jin, J. D. Lee, M. I. Jordan, B. Póczos, and A. Singh. 2017. "Gradient Descent Can Take Exponential Time to Escape Saddle Points." In *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, 1067—1077.

Dür, M., R. Horst, and M. Locatelli. 1998. "Necessary and Sufficient Global Optimality Conditions for Convex Maximization Revisited." *Journal of Mathematical Analysis and Applications* 217: 637–49.

Eckart, C., and G. Young. 1936. "The Approximation of One Matrix by Another of Lower Rank." *Psychometrika* 1 (3): 211–18.

Ekman, G. 1954. "Dimensions of Color Vision." *Journal of Psychology* 38:

467–74.

Fukuda, K. 2015. *cdd and cddplus Homepage*. https://www.inf.ethz.ch/personal/fukudak/cdd_home/.

Gaffney, P. W. 1976. "The Calculation of Indefinite Integrals of B-splines." *Journal of the Institute of Mathematics and Its Applications* 17: 37–41.

Geyer, C. J., and G. D. Meeden. 2015. "rcdd: Computational Geometry." 2015. https://CRAN.R-project.org/package=rcdd.

Gifi, A. 1990. *Nonlinear Multivariate Analysis*. New York, N.Y.: Wiley.

Gilbert, P., and R. Varadhan. 2019. *numDeriv: Accurate Numerical Derivatives*. https://CRAN.R-project.org/package=numDeriv.

Gohberg, I., P. Lancaster, and L. Rodman. 2009. *Matrix Polynomials*. Classic in Applied Mathematics. SIAM.

Gold, E. Mark. 1973. "Metric Unfolding: Data Requirement for Unique Solution & Clarification of Schönemann's Algorithm." *Psychometrika* 38 (4): 555–69.

Goldman, A. J. 1956. "Resolution and Separation Theorems for Polyhedral Convex Sets." In *Linear Inequalities and Related Systems*, edited by Kuhn H. W. and A. W. Tucker, 41–51. Annals of Mathematics Studies 38. Princeton University Press.

Gower, J. C. 1966. "Some Distance Properties of Latent Root and Vector Methods Used in Multivariate Analysis." *Biometrika* 53: 325–38.

Gower, J. C., and G. B. Dijksterhuis. 2004. *Procrustus Problems*. Oxford University Press.

Greenacre, M. J., and M. W. Browne. 1986. "An Efficient Alternating Least Squares Algorithm to Perform Multidimensional Unfolding." *Psychometrika* 51 (2): 241–50.

Groenen, P. J. F., and J. De Leeuw. 2010. "Power-Stress for Multidimensional Scaling."

Groenen, P. J. F., P. Giaquinto, and H. A. L Kiers. 2003. "Weighted Majorization Algorithms for Weighted Least Squares Decomposition Models." Econometric Institute Report EI 2003-09. Econometric Institute, Erasmus University Rotterdam. http://repub.eur.nl/pub/1700/.

Groenen, P. J. F., W. J. Heiser, and J. J. Meulman. 1998. "City-Block Scaling: Smoothing Strategies for Avoiding Local Minima." In *Classification, Data Analysis, and Data Highways*, edited by I. Balderjahn, R. Mathar, and M. Schader. Springer.

———. 1999. "Global Optimization in Least-Squares Multidimensional Scaling by Distance Smoothing." *Journal of Classification* 16: 225–54.

Groenen, P. J. F., R. Mathar, and W. J. Heiser. 1995. "The Majorization Approach to Multidimensional Scaling for Minkowski Distances." *Journal of Classification* 12: 3–19.

Groenen, P. J. F., and M. Van de Velden. 2016. "Multidimensional Scaling by Majorization: A Review." *Journal of Statistical Software* 73 (8): 1–26. https://doi.org/10.18637/jss.v073.i08.

Guilford, J. P. 1954. *Psychometric Methods*. McGraw-Hill.

Guttman, L. 1941. "The Quantification of a Class of Attributes: A Theory and Method of Scale Construction." In *The Prediction of Personal Adjustment*, edited by P. Horst, 321–48. New York: Social Science Research Council.

———. 1967. "The Development of Nonmetric Space Analysis: A Letter to Professor John Ross." *Multivariate Behavioral Research* 2 (1): 71–82.

———. 1968. "A General Nonmetric Technique for Fitting the Smallest Coordinate Space for a Configuration of Points." *Psychometrika* 33: 469–506.

Harman, R., and V. Lacko. 2010. "On Decompositional aAgorithms for Uniform Sampling from n-Spheres and n-Balls." *Journal of Multivariate Analysis* 101: 2297–2304.

Harshman, R. A. 1970. "Foundations of the PARAFAC Procedure." Working Papers in Phonetics 16. UCLA. https://www.psychology.uwo.ca/faculty/harshman/wpppfac0.pdf.

Heiser, W. J. 1988. "Multidimensional Scaling with Least Absolute Residuals." In *Classification and Related Methods of Data Analysis*, edited by H. H. Bock, 455–62. North-Holland Publishing Co.

———. 1991. "A Generalized Majorization Method for Least Squares Multidimensional Scaling of Pseudodistances that May Be Negative." *Psychometrika* 56 (1): 7–27.

———. 1995. "Convergent Computing by Iterative Majorization: Theory and Applications in Multidimensional Data Analysis." In *Recent Advantages in Descriptive Multivariate Analysis*, edited by W. J. Krzanowski, 157–89. Oxford: Clarendon Press.

Heiser, W. J., and J. De Leeuw. 1977. "How to Use SMACOF-I." Department of Data Theory FSW/RUL.

———. 1979. "Metric Multidimensional Unfolding." *Methoden En Data Nieuwsbrief SWS/VVS* 4: 26–50.

Hiriart-Urruty, J.-B. 1988. "Generalized Differentiability / Duality and Optimization for Problems Dealing with Differences of Convex Functions."

In *Convexity and Duality in Optimization*, edited by Ponstein. J., 37–70. Lecture Notes in Economics and Mathematical Systems 256. Springer.

"Inverse Multidimensional Scaling." 2007. *Journal of Classification* 14: 3–21.

Kearsley, A. J., R. A. Tapia, and M. W. Trosset. 1995. "The Solution of the Metric STRESS and SSTRESS Problems in Multidimensional Scaling Using Newton's Method."

Keller, J. B. 1962. "Factorization of Matrices by Least Squares." *Biometrika* 49: 239–42.

Kiers, H. A. L. 1997. "Weighted Least Squares Fitting Using Iterative Ordinary Least Squares Algorithms." *Psychometrika* 62: 251–66.

Kruskal, J. B. 1964a. "Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis." *Psychometrika* 29: 1–27.

———. 1964b. "Nonmetric Multidimensional Scaling: a Numerical Method." *Psychometrika* 29: 115–29.

———. 1971. "Monotone Regression: Continuity and Differentiability Properties." *Psychometrika* 36 (1): 57–62.

Kruskal, J. B., and J. D. Carroll. 1969. "Geometrical Models and Badness of Fit Functions." In *Multivariate Analysis, Volume II*, edited by P. R. Krishnaiah, 639–71. North Holland Publishing Company.

Lange, K. 2016 (in press). *MM Optimization Algorithms*.

Lawlor, G. R. 2020. "l'Hôpital's Rule for Multivariable Functions." *American Mathematical Monthly* 127 (8): 717–25.

Lee, J. D., M. Simchowitz, M. I. Jordan, and B. Recht. 2016. "Gradient Descent Converges to Minimizers." https://arxiv.org/abs/1602.04915.

Levelt, W. J. M., J. P. Van De Geer, and R. Plomp. 1966. "Triadic Comparions of Musical Intervals." *British Journal of Mathematical and Statistical Psychology* 19: 163–79.

Lingoes, J. C. 1968a. "The Multivariate Analysis Of Qualitative Data." *Multivariate Behavioral Research* 3 (1): 61–94.

———. 1968b. "The Rationale of the Guttman-Lingoes Nonmetric Series: A Letter to Doctor Philip Runkel." *Multivariate Behavioral Research* 3 (4): 495–507.

———. 1971. "Some Boundary Conditions for a Monotone Analysis of Symmetric Matrices." *Psychometrika* 36 (2): 195–203.

Lingoes, J. C., and E. E. Roskam. 1973. "A Mathematical and Empirical Analysis of Two Multidimensional Scaling Algorithms." *Psychometrika* 38: Monograph Supplement.

Lyusternik, L., and L. Schnirelmann. 1934. *Méthodes Topologiues Dans Les*

Problèmes Variationelle. Hermann.

Maehara, H. 1986. "Metric Transforms of Finite Spaces and Connected Graphs." *Discrete Mathematics* 61: 235–46.

Mair, P., P. J. F. Groenen, and J. De Leeuw. 2019. "More on Multidimensional Scaling in R: smacof Version 2." *Journal of Statistical Software*.

Messick, S. J., and R. P. Abelson. 1956. "The Additive Constant Problem in Multidimensional Scaling." *Psychometrika* 21 (1–17).

Meulman, J. J. 1986. "A Distance Approach to Nonlinear Multivariate Analysis." PhD thesis, Leiden University.

———. 1992. "The Integration of Multidimensional Scaling and Multivariate Analysis with Optimal Transformations." *Psychometrika* 57 (4): 539–65.

Meulman, J. J., and W. J. Heiser. 2012. *IBM SPSS Categories 21*. IBM Corporation.

Ortega, J. M., and W. C. Rheinboldt. 1970. *Iterative Solution of Nonlinear Equations in Several Variables*. New York, N.Y.: Academic Press.

Palubeckis, G. 2013. "An Improved Exact Algorithm for Least-Squares Unidimensional Scaling." *Journal of Applied Mathematics*, 1–15.

Papazoglou, S., and K. Mylonas. 2017. "An Examination of Alternative Multidimensional Scaling Techniques." *Educational and Psychological Measurement* 77 (3): 429–48.

Pliner, V. 1984. "A Class of Metric Scaling Models." *Automation and Remote Control* 45: 789–94.

———. 1986. "The Problem of Multidimensional Metric Scaling." *Automation and Remote Control* 47: 560–67.

———. 1996. "Metric Unidimensional Scaling and Global Optimization." *Journal of Classification* 13: 3–18.

Ramirez, C., R. Sanchez, V. Kreinovich, and M. Argaez. 2014. "$\sqrt{x^2+\mu}$ is the Most Computationally Efficient Smooth Approximation to |x|." *Journal of Uncertain Systems* 8: 205–10.

Ramsay, J. O. 1975. "Solving Implicit Equations in Psychometric Data Analysis." *Psychometrika* 40: 337–60.

———. 1977. "Maximum Likelihood Estimation in Multidimensional Scaling." *Psychometrika* 42: 241–66.

———. 1988. "Monotone Regression Splines in Action." *Statistical Science* 3: 425–61.

Robere, R. 2015. *vertexenum: Vertex Enumeration of Polytopes*. https://CRAN.R-project.org/package=vertexenum.

Robert, F. 1967. "Calcul du Rapport Maximal de Deux Normes sur $\mathbb{R}^n$." *Revue Francaise d'Automatique, d'Informatique Et De Recherche Operationelle* 1: 97–118.

Rockafellar, R. T. 1970. *Convex Analysis*. Princeton University Press.

Roskam, E. E. 1968. "Metric Analysis of Ordinal Data in Psychology." PhD thesis, University of Leiden.

———. 1979. "A General System for Nonmetric Data Analysis." In *Geometric Representations of Relational Data*, edited by J. C. Lingoes, E. E. Roskam, and I. Borg, 313–47. Mathesis Press.

Ross, J., and N. Cliff. 1964. "A Generalization of the Interpoint Distance Model." *Psychometrika* 29 (167-176).

Rothkopf, E. Z. 1957. "A Measure of Stimulus Similarity and Errors in some Paired-associate Learning." *Journal of Experimental Psychology* 53: 94–101.

Schoenberg, I. J. 1935. "Remarks to Maurice Frechet's article: Sur la Definition Axiomatique d'une Classe d'Espaces Vectoriels Distancies Applicables Vectoriellement sur l'Espace de Hllbert." *Annals of Mathematics* 36: 724–32.

———. 1937. "On Certain Metric Spaces Arising From Euclidean Spaces by a Change of Metric and Their Imbedding in Hilbert Space." *Annals of Mathematics* 38 (4): 787–93.

Schönemann, P. H. 1970. "On Metric Multidimensional Unfolding." *Psychometrika* 35 (3): 349–66.

Schumaker, L. 2007. *Spline Functions: Basic Theory*. Third Edition. Cambridge University Press.

Shepard, R. N. 1962a. "The Analysis of Proximities: Multidimensional Scaling with an Unknown Distance Function. I." *Psychometrika* 27: 125–40.

———. 1962b. "The Analysis of Proximities: Multidimensional Scaling with an Unknown Distance Function. II." *Psychometrika* 27: 219–46.

Shepard, R. N., and J. D. Carroll. 1966. "Parametric Representation of Nonlinear Data Structures." In *Mulktivariate Analysis*, edited by P. R. Krishnaiah, 561–92. Academic Press.

Sidi, A. 2017. *Vector Extrapolation Methods with Applicatons*. SIAM.

Spang, H. A. 1962. "A Review of Minimization Techniques for Nonlinear Functions." *SIAM Review* 4 (4): 343–65.

Spence, I. 1983. "Monte Carlo Studies." *Applied Psychological Measurement* 7 (4): 405–25.

Stephenson, W. 1953. *The Study of Behavior: Q-technique and its Method-*

ology. University of Chicago Press.

Takane, Y., F. W. Young, and J. De Leeuw. 1977. "Nonmetric Individual Differences in Multidimensional Scaling: An Alternating Least Squares Method with Optimal Scaling Features." *Psychometrika* 42: 7–67. http://www.stat.ucla.edu/~deleeuw/janspubs/1977/articles/takane_young_deleeuw_A_77.pdf.

Taussky, O. 1949. "A Recurring Theorem on Determinants." *American Mathematical Monthly* 56: 672–76.

Tisseur, F., and K. Meerbergen. 2001. "The Quadratic Eigenvalue Problem." *SIAM Review* 43 (2): 235–86.

Torgerson, W. S. 1958. *Theory and Methods of Scaling*. New York: Wiley.

Torgerson, W. S. 1952. "Multidimensional Scaling: I. Theory and Method." *Psychometrika* 17 (4): 401–19.

———. 1965. "Multidimensional Scaling of Similarity." *Psychometrika* 30 (4): 379–93.

Trosset, M. W., and R. Mathar. 1997. "On the Existence on Nonglobal Minimizers of the STRESS Criterion for Metric Multidimensional Scaling." In *Proceedings of the Statistical Computing Section*, 158–62. Alexandria, VA: American Statistical Association.

Turlach, B. A., and A. Weingessel. 2013. *quadprog: Functions to solve Quadratic Programming Problems*. https://CRAN.R-project.org/package=quadprog.

Van de Geer, J. P. 1967. *Inleiding in de Multivariate Analyse*. Van Loghum Slaterus.

———. 1971. *Introduction to Multivariate Analysis for the Social Sciences*. San Francisco, CA: Freeman.

Van de Geer, J. P., W. J. M. Levelt, and R. Plomp. 1962. "The Connotation of Musical Consonance." *Acta Psychologica* 20: 308–19.

Wang, Y., C. L. Lawson, and R. J. Hanson. 2015. *lsei: Solving Least Squares Problems under Equality/Inequality Constraints*. http://CRAN.R-project.org/package=lsei.

Young, F. W. 1981. "Quantitative Analysis of Qualitative Data." *Psychometrika* 46: 357–88.

Young, F. W., Y. Takane, and R. Lewyckyj. 1978a. "ALSCAL: A Nonmetric Multidimensional Scaling Program with Several Individual-Differences Options." *Behavior Research Methods & Instrumentation* 10 (3): 451–53.

———. 1978b. "Three Notes on ALSCAL." *Psychometrika* 43 (3): 433–35.

Young, G., and A. S. Householder. 1938. "Discussion of a Set of Points in Terms of Their Mutual Distances." *Psychometrika* 3 (19-22).

Zilinskas, A., and A. Poslipskyte. 2003. "On Multimodality of the SSTRESS Criterion for Metric Multidimensional Scaling." *Informatica* 14 (1): 121–30.