Multidimensional Scaling in the City-Block Metric: L_1 and L_2-Norm Optimization Methods Using MATLAB

L. J. Hubert
University of Illinois

P. Arabie
Rutgers University

J. J. Meulman
Leiden University

The research reported here has been partially supported by the National Science Foundation through Grant No. SES-981407 (to Hubert) and by the Netherlands Organization for Scientific Research (NWO) through Grant No. 575-67-053 for the 'PIONEER' project 'Subject Oriented Multivariate Analysis' (to Meulman).

Authors' Addresses: L. J. Hubert, Department of Psychology, University of Illinois, 603 East Daniel Street, Champaign, Illinois 61820, USA; P. Arabie, Faculty of Management, Rutgers University, 180 University Avenue, Newark, NJ 07102-1895, USA; J. J. Meulman, Department of Education, Data Theory Group, Leiden University, PO Box 9555, 2300 RB Leiden, The Netherlands.

Abstract: This paper is a companion to Hubert, Arabie, and Meulman (2002; and published in this same journal). The latter provides a number of different least-squares (L_2) optimization strategies for linear unidimensional scaling (LUS) within a MATLAB environment. The current paper develops extensions, again within a MATLAB context, to multidimensional scaling in the city-block metric using both an L_2 and an L_1 (least sum-of-absolute-deviations) loss function. Although L_1 is an alternative to the use of L_2, it doesn't appear to give any salient advantages; also, it is very expensive computationally to implement when formulated through its linear programming subtasks. A final generalization is given in the L_2 context that incorporates optimal monotonic transformations of the original proximities, thus providing a readily available, computationally reasonable strategy for nonmetric multidimensional scaling in the city-block metric (in two and three dimensions) based on the given MATLAB functions provided.

Keywords: Multidimensional Scaling; Linear Unidimensional Scaling; City-Block Metric; L_1-Norm; L_2-Norm; MATLAB.

1 Introduction

In a previous paper in this journal (Hubert, Arabie, and Meulman, 2002; hereafter, referred to as HAM2002), a comparison is made among several different optimization strategies for the linear unidimensional scaling (LUS) task in the L_2-norm, with all implementations carried out within a MATLAB computational environment. The central LUS task involves arranging the n objects in a set $S = \{O_1, O_2, \ldots, O_n\}$ along a single dimension, defined by coordinates $x_1, x_2, \ldots, x_n$, based on an $n \times n$ symmetric proximity matrix $\mathbf{P}$ = $\{p_{ij}\}$, whose nonnegative entries are given a dissimilarity interpretation ($p_{ij} = p_{ji}$ for $1 \leq i, j \leq n$; $p_{ii} = 0$ for $1 \leq i \leq n$). The L_2 criterion

$$\sum_{i<j} (p_{ij} - |x_j - x_i|)^2, \qquad (1)$$

is minimized by the choice of the coordinates. The present paper can be considered a close companion to this earlier piece, with extensions now given to multidimensional scaling in the city-block metric for both the L_1 and L_2 norms. The computational routines to be discussed and illustrated are again freely available as MATLAB m-files.[1] We also note that most of the references given in this earlier paper would also be relevant here as background material on the basic LUS task, but that review will not be repeated here. Also, we will not discuss (in this paper) comparisons to other methods (or strategies) for multidimensional scaling in the city-block metric — for the development of some of these alternatives, see Brusco (2001), Brusco and Stahl (in press), Groenen, Heiser, and Meulman (1999), Hubert, Arabie, and Meulman (1997), and Hubert, Arabie, and Hesson-McInnis (1992).

In the extensions to city-block multidimensional scaling being pursued, a slight generalization to the basic unidimensional task that incorporates an additional additive constant will prove extremely convenient. So, in Section 2 we emphasize the more general least-squares loss function of the form

$$\sum_{i<j}(p_{ij} - \{|x_j - x_i| - c\})^2, \tag{2}$$

where c is some constant to be estimated along with the coordinates $x_1, \ldots, x_n$; also, in Section 2, the least sum-of-absolute-deviations (the L_1 loss function) is considered as a variant for estimating c and the coordinates:

$$\sum_{i<j}|p_{ij} - \{|x_j - x_i| - c\}|. \tag{3}$$

Sections 3 and 4 remove the restriction to fitting only a single unidimensional structure to a symmetric proximity matrix, and rely on the type of computational approaches developed in Section 2 that include the augmentation by estimated additive constants. Based on these latter strategies, extensions are given to the use of multiple unidimensional structures through a procedure of successive residualization of the original proximity matrix (even though in this process, negative residuals are encountered and need to be fitted). For example, the fitting of two LUS structures to a proximity matrix $\{p_{ij}\}$ could be rephrased as the minimization of an L_2 loss function generalizing (2) to the form

$$\sum_{i<j}(p_{ij} - [|x_{j1} - x_{i1}| - c_1] - [|x_{j2} - x_{i2}| - c_2])^2, \tag{4}$$

[1]In the presentation throughout the paper, reference is made to the m-files available in an Appendix A. This appendix (in pdf format) called cbs_appendix.pdf is available at the ftp site: (ftp://www.psych.uiuc.edu/pub/cda). The m-files themselves (plus all other files mentioned throughout the paper) are in the 'zipped' file, cbs_files.zip, at this same address. The 'cbs' acronym stands for 'city-block scaling'.

or by extending (3) to the use of an L_1 loss function:

$$\sum_{i<j} |p_{ij} - [|x_{j1} - x_{i1}| - c_1] - [|x_{j2} - x_{i2}| - c_2]|. \tag{5}$$

The attempt to minimize (4) or (5) could proceed with the fitting of a single LUS structure to $\{p_{ij}\}$, $[|x_{j1} - x_{i1}| - c_1]$, and once obtained, fitting a second LUS structure, $[|x_{j2} - x_{i2}| - c_2]$, to the residual matrix, $\{p_{ij} - [|x_{j1} - x_{i1}| - c_1]\}$. The process would then cycle by repetitively fitting the residuals from the second linear structure by the first, and the residuals from the first linear structure by the second, until the sequence converges. In any case, obvious extensions would also exist to (4) and (5) for the inclusion of more than two LUS structures.

The explicit inclusion of two constants, c_1 and c_2, in (4) and (5) rather than adding these two together and including a single additive constant c, deserves some additional introductory explanation. As would be the case in fitting a single LUS structure using the loss functions in (2) and (3), two interpretations exist for the role of the additive constant c. We could consider $\{|x_j - x_i|\}$ to be fitted to the translated proximities $\{p_{ij} + c\}$, or alternatively, $\{|x_j - x_i| - c\}$ to be fitted to the original proximities $\{p_{ij}\}$, where the constant c becomes part of the actual model. Although these two interpretations do not lead to any algorithmic differences in how we would proceed with minimizing the loss functions in (2) and (3), a consistent use of the second interpretation suggests that we frame extensions to the use of multiple LUS structures as we did in (4) and (5), where it is explicit that the constants c_1 and c_2 are part of the actual models to be fitted to the (untransformed) proximities $\{p_{ij}\}$. Once c_1 and c_2 are obtained, they could be summed as $c = c_1 + c_2$, and an interpretation made that we have attempted to fit a transformed set of proximities $\{p_{ij} + c\}$ by the sum $\{|x_{j1} - x_{i1}| + |x_{j2} - x_{i2}|\}$ (and in this latter case, a more usual terminology would be one of a two-dimensional scaling (MDS) based on the city-block distance function). However, such a further interpretation is unnecessary and could lead to at least some small terminological confusion in further extensions that we might wish to pursue. For instance, if some type of (optimal nonlinear) transformation, say $f(\cdot)$, of the proximities is also sought (e.g., a monotonic function of some form as we do in Section 5) in addition to fitting multiple LUS structures, and where p_{ij} in (4) or (5) is replaced by $f(p_{ij})$, and $f(\cdot)$ is to be constructed, the first interpretation would require the use of a 'doubly transformed' set of proximities $\{f(p_{ij}) + c\}$ to be fitted by the sum $\{|x_{j1} -$

$x_{i1}| + |x_{j2} - x_{i2}|\}$. In general, it seems best to avoid the need to incorporate the notion of a double transformation in this context, and instead merely consider the constants c_1 and c_2 to be part of the models being fitted to a transformed set of proximities $f(p_{ij})$.

Although L_1 norms are possible to use within both the unidimensional and multidimensional contexts, and we give MATLAB m-files to do so, our general conclusion after experimentation is that they might be best avoided because of their generally needed much greater computationally expensive implementation without any particularly clear advantage. Given this lack of any prominent reason to use L_1 as opposed to L_2, the extension to a nonmetric multidimensional scaling in the city-block metric given in Section 5 will be limited to the L_2 context.

2 The Incorporation of Additive Constants in LUS

In Section 2.1.1 below, we present and illustrate a MATLAB m-function, `linfitac.m`, that fits in L_2 a given single unidimensional scale (by providing the coordinates $x_1, \ldots, x_n$) and the additive constant (c) for some fixed input object ordering along the continuum defined by a permutation $\rho^{(0)}$. This parallels directly the m-function given in HAM2002 called `linfit.m`, but now with an included additive constant estimation. The computational mechanisms implemented in `linfitac.m` are reviewed in Section 2.1. Two additional MATLAB m-functions, `linfitl1.m` and `linfitl1ac.m`, are presented in Section 2.2.1 that fit within L_1 a given single unidimensional scale defined by a fixed object ordering $\rho^{(0)}$ without and with an additional additive constant respectively. The computational basis for these functions is reviewed in Section 2.2. Finally, Section 2.3 presents two m-functions, `uniscallp.m` and `uniscallpac.m`, that rely on an iterative linear programming strategy to actually identify best-fitting object orderings within the L_1 norm. For the L_2 context, a comparable general purpose method for finding an object order was made available in HAM2002 based on iterative quadratic assignment (the m-function `uniscalqa.m`).

2.1 The L_2 Fitting of a Single Unidimensional Scale (with an Additive Constant)

Given a fixed object permutation, $\rho^{(0)}$, we denote the set of all $n \times n$ matrices that are additive translations of the off-diagonal entries in the reordered symmetric proximity matrix $\{p_{\rho^{(0)}(i)\rho^{(0)}(j)}\}$ by $\Delta_{\rho^{(0)}}$, and let Ξ be the set of all $n \times n$ matrices that represent the interpoint distances between all pairs of n coordinate locations along a line. Explicitly,

$\Delta_{\rho^{(0)}} \equiv \{\{q_{ij}\}\}$, where $q_{ij} = p_{\rho^{(0)}(i)\rho^{(0)}(j)} + c$, for some constant c, $i \neq j; q_{ii} = 0, 1 \leq i, j \leq n$;

$\Xi \equiv \{\{r_{ij}\}\}$, where $r_{ij} = |x_j - x_i|$ for some set of n coordinates, $x_1 \leq \cdots \leq x_n$; $\sum_i x_i = 0$.

Alternatively, we could define Ξ through a set of linear inequality (for non-negativity restrictions) and equality constraints (to represent the additive nature of distances along a line – as we did in `linfit.m` in HAM2002). In any case, both $\Delta_{\rho^{(0)}}$ and Ξ are closed convex sets (in a Hilbert space), and thus, given any $n \times n$ symmetric matrix with a zero main diagonal, its projection onto either $\Delta_{\rho^{(0)}}$ or Ξ exists, i.e., there is a (unique) member of $\Delta_{\rho^{(0)}}$ or Ξ at a closest (Euclidean) distance to the given matrix (e.g., see Cheney and Goldstein, 1959). Moreover, if a procedure of alternating projections onto $\Delta_{\rho^{(0)}}$ and Ξ is carried out (where a given matrix is first projected onto one of the sets, and that result is then projected onto the second which result is in turn projected back onto the first, and so on), the process is convergent and generates members of $\Delta_{\rho^{(0)}}$ and Ξ that are closest to each other (again, this last statement is justified in Cheney and Goldstein, 1959, Theorems 2 and 4).

Given any $n \times n$ symmetric matrix with a main diagonal of all zeros, which we denote arbitrarily as $\mathbf{U} = \{u_{ij}\}$, its projection onto $\Delta_{\rho^{(0)}}$ may be obtained by a simple formula for the sought constant c. Explicitly, the minimum over c of

$$\sum_{i<j} (\{p_{\rho^{(0)}(i)\rho^{(0)}(j)}\} + c - u_{ij})^2,$$

is obtained for

$$\hat{c} = (2/n(n-1)) \sum_{i<j} (u_{ij} - p_{\rho^{(0)}(i)\rho^{(0)}(j)}),$$

and thus, this last value defines a constant translation of the proximities necessary to generate that member of $\Delta_{\rho^{(0)}}$ closest to $\mathbf{U} = \{u_{ij}\}$. For the second necessary projection and given any $n \times n$ symmetric matrix (again with a main diagonal of all zeros), that we denote arbitrarily as $\mathbf{V} = \{v_{ij}\}$ (but which in our applications will generally have the form $v_{ij} = p_{\rho^{(0)}(i)\rho^{(0)}(j)} + c$ for $i \neq j$ and some constant c), its projection onto Ξ is somewhat more involved and requires minimizing

$$\sum_{i<j}(v_{ij} - r_{ij})^2,$$

over r_{ij}, where $\{r_{ij}\}$ is subject to the linear inequality nonnegativity constraints, and the linear equality constraints of representing distances along a line (of the set Ξ). Although this is a (classic) quadratic programming problem for which a wide variety of optimization techniques has been published, we adopt (as we did in fitting a LUS without an additive constant in `linfit.m`), the Dykstra-Kaczmarz iterative projection strategy reviewed in the appendix to HAM2002.

2.1.1 The MATLAB function linfitac.m

As discussed above, the MATLAB m-function in Section A.1 of the Appendix A, `linfitac.m`, fits a set of coordinates to a given proximity matrix based on some given input permutation, say, $\rho^{(0)}$, plus an additive constant, c. The usage syntax of

```
[fit vaf coord addcon] = linfitac(prox,inperm)
```

is similar to that of `linfit.m` (of HAM2002) except for the inclusion (as output) of the additive constant `ADDCON`, and the replacement of the least-squares criterion of `DIFF` by the variance-accounted-for (`VAF`) given by the general formula

$$\text{vaf} = 1 - \frac{\sum_{i<j}(p_{\rho^{(0)}(i)\rho^{(0)}(j)} + c - |x_j - x_i|)^2}{\sum_{i<j}(p_{ij} - \bar{p})^2},$$

where $\bar{p}$ is the mean of the proximity values being used.

To illustrate the invariance of `VAF` to the use of linear transformations of the proximity matrix (although `COORD` and `ADDCON` obviously will change depending on the transformation used), we fit the permutation found optimal in the HAM2002 companion paper to two different matrices: the original proximity matrix for `number.dat`, and one standardized to mean zero and variance one. The latter matrix is obtained with the utility `proxstd.m`, given in Appendix A.11, with usage explained in its m-file header comments.

In the recording below, semicolons are placed after the invocation of the m-functions to initially suppress the output; transposes(') are then used on the output vectors to conserve space by only using row (as opposed to column) vectors in the listing; also, to conserve space, blank lines are always deleted in any output given throughout the paper. Note that for the two proximity matrices used, the vaf values are the same (.5612) but the coordinates and additive constants differ; a listing of the standardized proximity matrix is given in the output to explicitly show how negative proximities pose no problem for the fitting process that allows the incorporation of additive constants within the fitted model.

```
>> load number.dat
>> inperm = [1 2 3 5 4 6 7 9 10 8];
>> [fit vaf coord addcon] = linfitac(number,inperm);
>> vaf
vaf =
    0.5612
>> coord'
ans =
  Columns 1 through 5
   -0.3790   -0.2085   -0.1064   -0.0565   -0.0257
  Columns 6 through 10
    0.0533    0.1061    0.1714    0.1888    0.2565
>> addcon
addcon =
   -0.3089
>> numberstan = proxstd(number,0.0)
numberstan =
  Columns 1 through 5
         0   -0.5919    0.2105    0.8258    0.7027
   -0.5919         0   -1.2663   -0.9611    0.5157
    0.2105   -1.2663         0   -0.9217   -2.3739
    0.8258   -0.9611   -0.9217         0   -0.6313
    0.7027    0.5157   -2.3739   -0.6313         0
    1.2934    0.2302    0.6387   -0.5525   -0.6510
    1.2147    1.0670   -0.5919   -1.1876   -0.7544
    1.8103    0.4369    1.2541    0.2498    0.9882
    1.3771    1.2294   -0.8577    1.2934   -1.4534
    1.5199    0.4123    1.3131   -1.3697    0.6978
  Columns 6 through 10
    1.2934    1.2147    1.8103    1.3771    1.5199
    0.2302    1.0670    0.4369    1.2294    0.4123
    0.6387   -0.5919    1.2541   -0.8577    1.3131
   -0.5525   -1.1876    0.2498    1.2934   -1.3697
   -0.6510   -0.7544    0.9882   -1.4534    0.6978
         0   -0.7150   -0.6953    0.6387    0.2498
   -0.7150         0   -0.6116   -0.9414   -1.2072
   -0.6953   -0.6116         0   -0.6953   -0.4049
    0.6387   -0.9414   -0.6953         0   -0.7347
    0.2498   -1.2072   -0.4049   -0.7347         0
>> [fit vaf coord addcon] = linfitac(numberstan,inperm);
>> vaf
vaf =
    0.5612
>> coord'
ans =
  Columns 1 through 5
   -1.8656   -1.0262   -0.5235   -0.2783   -0.1266
  Columns 6 through 10
    0.2624    0.5224    0.8435    0.9292    1.2626
>> addcon
addcon =
    1.1437
```

2.2 The L_1 Fitting of a Single Unidimensional Scale (with an Additive Constant)

The linear unidimensional scaling task in the L_1 norm can be phrased as one of finding a set of coordinates $x_1, \ldots, x_n$ such that the L_1 criterion

$$\sum_{i<j} |p_{ij} - (|x_j - x_i| - c)| \qquad (6)$$

is minimized, where we now immediately include the possibility of an additive constant in the model (in what follows, c can just be set to 0 for the more elemental model without an additive constant). As an alternative reformulation of the optimization task in (6) that will prove convenient as a point of departure in our development of computational routines (much as what we did within the L_2 norm), we subdivide (6) into the two separate problems of finding a set of n numbers, $x_1 \leq \cdots \leq x_n$, and a permutation on the first n integers, $\rho(\cdot) \equiv \rho$, for which

$$\sum_{i<j} |p_{\rho(i)\rho(j)} - ((x_j - x_i) - c)| \qquad (7)$$

is minimized. Again, we can impose the additional constraint that $\sum_{i=1}^{n} x_i = 0$.

Assuming for now that the permutation ρ is given, the task of finding $x_1 \leq \cdots \leq x_n$ to minimize (7) is a linear programming problem. Without loss of generality, we let ρ be the identity permutation and first rewrite $\sum_{i<j} |p_{ij} - (|x_j - x_i| - c)|$ as the loss criterion $\sum_{i<j}(z_{ij}^{+} + z_{ij}^{-})$, where

$$z_{ij}^{+} = \frac{1}{2}\{|p_{ij} - (|x_j - x_i| - c)| - (p_{ij} - (|x_j - x_i| - c))\};$$

$$z_{ij}^{-} = \frac{1}{2}\{|p_{ij} - (|x_j - x_i| - c)| + (p_{ij} - (|x_j - x_i| - c))\},$$

for $1 \leq i < j \leq n$. The unknowns are c, $x_1, \ldots, x_n$, and for $1 \leq i < j \leq n$, z_{ij}^{+}, z_{ij}^{-}, and y_{ij} ($\equiv |x_j - x_i|$). The constraints of the linear program take the form:

$$-z_{ij}^{+} + z_{ij}^{-} + y_{ij} - c = p_{ij};$$

$$-x_j + x_i + y_{ij} = 0;$$

$$z_{ij}^{+} \geq 0, z_{ij}^{-} \geq 0, y_{ij} \geq 0,$$

for $1 \leq i < j \leq n$, and

$$x_1 + \cdots + x_n = 0.$$

2.2.1 The MATLAB functions linfitl1.m and linfitl1ac.m

Based on the linear programming reformulation just given for finding a set of ordered coordinates for a fixed object permutation, Appendices A.2 and A.3 give the m-functions, `linfitl1.m` and `linfitl1ac.m`, where the latter includes an additive constant in the model and the former does not. Both of these m-functions serve to setup the relevant (constraint) matrices for the associated linear programming task; the actual linear programming optimization is carried out by invoking `linprog.m` from the MATLAB Optimization Toolbox.

The syntax for `linfitl1.m` is

```
[fit diff coord exitflag] = linfitl1(prox,inperm)
```

where if we denote the given permutation as $\rho^0(\cdot)$ (`INPERM`), we seek a set of coordinates $x_1 \leq \cdots \leq x_n$ (`COORD`) to minimize (at a value of `DIFF`)

$$\sum_{i<j} |p_{\rho^0(i)\rho^0(j)} - |x_j - x_i||;$$

`FIT` refers to the matrix $\{|x_j - x_i|\}$, and `EXITFLAG` describes the exit condition of the linear program optimization (greater than 0 for convergence; 0 denotes the maximum number of function evaluations or iterations was exceeded; less than 0 indicates a failure of convergence to a solution). For using `linfitl1ac.m`, the syntax is

```
 [fit dev coord addcon exitflag] = linfitl1ac(prox,inperm)
```

Here, we minimize

$$\sum_{i<j} |p_{\rho^0(i)\rho^0(j)} - (|x_j - x_i| - c)|$$

where c is given by `ADDCON` and `DEV` refers to the deviance(-accounted-for) defined by the normalized L_1 loss value:

$$\text{DEV} = 1 - \frac{\sum_{i<j} |p_{\rho^0(i)\rho^0(j)} - (|x_j - x_i| - c)|}{\sum_{i<j} |p_{ij} - p_{med}|},$$

where p_{med} is the median of the off-diagonal proximity values.

We illustrate the use of `linfitl1.m` and `linfitl1ac.m` on the `number.dat` proximity matrix using the identity permutation as the input object order. To conserve space, the `FIT` matrices are not listed below.

```
>> load number.dat
>> inperm = 1:10;
>> [fit diff coord exitflag] = linfitl1(number,inperm);
Optimization terminated successfully.
>> diff
diff =
    8.2120
>> coord'
ans =
  Columns 1 through 6
   -0.7260   -0.4780   -0.2205   -0.1320   -0.1046    0.0780
  Columns 7 through 10
    0.2005    0.4022    0.4022    0.5784
>> exitflag
exitflag =
     1
>> [fit dev coord addcon exitflag] = linfitl1ac(number,inperm);
Optimization terminated successfully.
>> dev
dev =
    0.4252
>> coord'
ans =
  Columns 1 through 6
   -0.3511   -0.1956   -0.1129   -0.0809   -0.0137    0.0466
  Columns 7 through 10
    0.0930    0.1643    0.2020    0.2482
>> addcon
addcon =
   -0.3458
>> exitflag
exitflag =
     1
```

2.3 Iterative Linear Programming

Given the availability of the two linear programming based m-functions (discussed in the previous Section 2.2.1) for fitting given unidimensional scales defined by specific input object permutations, it is possible to embed these two routines in a search strategy for actually finding the (at least hopefully) best such permutations in the first place. This embedding is analogous to adopting iterative quadratic assignment in `uniscalqa.m` (as given in HAM2002) and attempting to locate good unidimensional scalings in the L_2 norm. Here, we have an iterative use of linear programming in `uniscallp.m` (in Appendix A.4) and `uniscallpac.m` (in Appendix A.5) to identifying the good unidimensional scales in the L_1 norm, without and with, respectively, an additive constant in the fitted model. The usage syntax of both m-functions are as follows:

```
[outperm coord diff fit] = uniscallp(prox,inperm)
[outperm coord dev fit addcon] = uniscallpac(prox,inperm)
```

Both m-functions begin with a given object ordering (`INPERM`) and evaluate the effect of pairwise object interchanges on the current permutation carried forward to that point. If an object interchange is identified that improves the L_1 loss value, that interchange is made and the changed permutation becomes the current one. When no pairwise object interchange can reduce `DIFF` in `uniscallp.m`, or increase `DEV` in `uniscallpac.m` over its current value, that ending permutation is provided as `OUTPERM` along with its coordinates (`COORD`) and the matrix `FIT` (the absolute differences of the ordered coordinates). In `uniscallpac.m`, the additive constant (`ADDCON`) is also given.

The numerical example that follows relies on `number.dat` to provide the proximity matrix, and initializes both the m-functions with the identity permutation (1:10). From other random starts that we have tried in addition to this very rational starting permutation, the resulting scales we give below are (almost undoubtedly) L_1-norm optimal. We might note that the (optimal) object orderings differ depending on whether or not an additive constant is included in the model.

```
>> load number.dat
>> [outperm coord diff fit] = uniscallp(number,1:10);
>> outperm
outperm =
     1     2     3     5     4     9     7     6    10     8
>> coord'
ans =
  Columns 1 through 6
   -0.7129   -0.3762   -0.2451   -0.1861   -0.0302    0.1081
  Columns 7 through 10
    0.1838    0.2229    0.4798    0.5559
>> diff
diff =
    7.0430
```

```
>> [outperm coord dev fit addcon] = uniscallpac(number,1:10);
>> outperm
outperm =
     1     2     3     4     5     7     6     9    10     8
>> coord'
ans =
  Columns 1 through 6
   -0.3807   -0.1647   -0.1087   -0.0637    0.0143    0.0903
  Columns 7 through 10
    0.1113    0.1283    0.1573    0.2163
>> dev
dev =
    0.4479
>> addcon
addcon =
   -0.3120
```

Although the m-functions are provided to either fit or find the best unidimensional scalings in the L_1 norm, we do not suggest their routine use. The finding of the best unidimensional scales in L_1 is extremely expensive computationally (given the use of the repetitive linear programming subtasks), and without any obvious advantage over L_2, it is not clear why the L_1 approach should be pursued. This same set of conclusions exist as well for the L_1 finding and fitting of multidimensional unidimensional scales illustrated in Section 4. (We might also mention a possible issue with ill-conditioning for some uses of `linfitl1.m` and `linfitl1ac.m` if the simplex option is chosen [and not the default interior-point algorithm] for the optimization method implemented in `linprog.m` (from the Optimization Toolbox for MATLAB 6.5). The end results appear generally to be fine, but the intermediate warnings in either the finding or fitting of these unidimensional scales within L_1 is disconcerting. So the use of the default interior-point strategy is recommended whenever `linprog.m` is called.)

3 The L_2 Finding and Fitting of Multiple Unidimensional Scales

As reviewed in the Introduction, the fitting of multiple unidimensional structures will be done by (repetitive) successive residualization, along with a reliance on the m-function, `linfitac.m`, to fit each separate unidimensional structure, including its associated additive constant. The m-function in Appendix Section A.6, `biscalqa.m`, is a two-(or bi-)dimensional scaling strategy for the L_2 loss function of (4). It has the syntax

```
[outpermone outpermtwo coordone coordtwo fitone fittwo ...
addconone addcontwo vaf] = biscalqa(prox,...
targone,targtwo,inpermone,inpermtwo,kblock,nopt)
```

where the variables are similar to `linfitac.m`, but with a suffix of `ONE` or `TWO` to indicate which one of the two unidimensional structures is being referenced. The new variable `NOPT` controls the confirmatory or exploratory fitting of the two unidimensional scales; a value of `NOPT = 0` will fit in a confirmatory manner the two scales indicated by `INPERMONE` and `INPERMTWO`; if `NOPT = 1`, iterative quadratic assignment (QA) (as discussed in HAM2002) is used to locate the better permutations to fit.

In the example given below, the input `PROX` is the standardized (to a mean of zero and a standard deviation of one) 10×10 proximity matrix based on `number.dat` (referred to as `STANNUMBER`); `TARGONE` and `TARGTWO` are identical 10×10 equally-spaced target matrices; `INPERMONE` and `INPERMTWO` are different random permutations of the first 10 integers; `KBLOCK` is set at 2 (for the iterative QA subfunctions). In the output, `OUTPERMONE` and `OUTPERMTWO` refer to the object orders; `COORDONE` and `COORDTWO` give the coordinates; `FITONE` and `FITTWO` are based on the absolute coordinate differences for the two unidimensional structures; `ADDCONONE` and `ADDCONTWO` are the two associated additive constraints; and finally, `VAF` is the variance-accounted-for in `PROX` by the two-dimensional structure.

```
>> load number.dat
>> stannumber = proxstd(number,0.0);
>> inpermone = randperm(10);
>> inpermtwo = randperm(10);
>> kblock = 2;
>> nopt = 1;
>> [prox10,targone,targcir] = ransymat(10);
>> targtwo = targone;
>> [outpermone outpermtwo coordone coordtwo fitone fittwo...
addconone addcontwo vaf] = biscalqa(stannumber,targone,...
targtwo,inpermone,inpermtwo,kblock,nopt);
>> outpermone
outpermone =
  Columns 1 through 10
     1     2     3     4     5     6     7     9     8    10
>> outpermtwo
outpermtwo =
  Columns 1 through 10
     5     9     3     1     7     4    10     2     8     6
>> coordone'
ans =
  Columns 1 through 5
   -2.1786   -1.2363   -0.5418   -0.2915    0.0009
  Columns 6 through 10
    0.0858    0.6805    1.0310    1.0310    1.4191
>> coordtwo'
ans =
  Columns 1 through 5
   -0.8791   -0.8791   -0.8791   -0.2629   -0.1151
  Columns 6 through 10
    0.2472    0.2472    0.3639    0.9885    1.1688
>> addconone
addconone =
    1.3137
>> addcontwo
addcontwo =
    0.8803
>> vaf
vaf =
    0.8243
```

Although we have used the proximity matrix in `number.dat` primarily as a convenient numerical example to illustrate the various m-functions provided in the appendix, the substantive interpretation (as noted in more detail in HAM2002) for this particular two-dimensional structure is rather remarkable and worth pointing out. The first dimension reflects number magnitude perfectly (in its coordinate order) with two objects (the actual digits 7 8) at the same (tied) coordinate value. The second axis reflects the structural characteristics perfectly, with the coordinates split into the odd and even numbers (the digits 4 8 2 0 6 in the first five positions; 3 9 1 7 5 in the second five); there is a grouping of 4 8 2 at the same coordinates (reflecting powers of 2); a grouping of 6 3 9 (reflecting multiples of three) and of 3 9 at the same coordinates (reflecting the powers of 3); the odd numbers 7 5 that are not powers of 3 are at the extreme two coordinates of this second dimension.

Although we will not explicitly illustrate its use here, a tridimensional m-function, `triscalqa.m`, is given in Appendix A.7 that is an obvious generalization of `biscalqa.m`. The pattern of programming that this shows could be used directly as a pattern for extensions beyond three unidimensional structures.

4 The L_1 Finding and Fitting of Multiple Unidimensional Scales

In analogy to the L_2 fitting of multiple unidimensional structures, the use of the L_1 norm can again be done by (repetitive) successive residualization, but now with a reliance on the m-function, `linfitl1ac.m`, to fit each separate unidimensional structure with its additive constant. The m-function, `biscallp.m`, given in the Appendix Section A.8 is a two-(or bi-)dimensional scaling strategy for the L_1 loss-function

$$\sum_{i<j} |p_{ij} - [|x_{j1} - x_{i1}| - c_1] - [|x_{j2} - x_{i2}| - c_2]|, \qquad (8)$$

with syntax (and all variables) similar to `biscalqa.m` of Section 3, including a provision for the confirmatory fitting of two given input orders (by setting `NOPT = 0`).

In the example given below, the two beginning permutations used were the ones identified with the L_2 norm. Compared to the input permutations, there are no changes in the output permuations. The final deviance-accounted-for (`DEV`) of this solution is .6787.

```
>>load number.dat
>>stannumber = proxstd(number,0.0);
>>nopt = 1;
>>inpermone = [1 2 3 4 5 6 7 9 8 10];
>>inpermtwo = [5 3 9 1 7 4 10 2 8 6];
>>[outpermone outpermtwo coordone coordtwo fitone fittwo ...
addconone addcontwo dev] = biscallp(stannumber,inpermone,inpermtwo,nopt);
>> outpermone
outpermone =
     1     2     3     4     5     6     7     9     8    10
>> outpermtwo
outpermtwo =
     5     3     9     1     7     4    10     2     8     6
>> coordone'
ans =
  Columns 1 through 6
   -1.9927   -1.1377   -0.4443   -0.3343    0.0611    0.1170
  Columns 7 through 10
    0.4919    0.7986    1.1162    1.3243
>> coordtwo'
ans =
  Columns 1 through 6
   -0.9351   -0.9351   -0.8721   -0.1714   -0.0055    0.1462
  Columns 7 through 10
    0.1462    0.4965    0.8732    1.2571
>> addconone
addconone =
    1.2366
>> addcontwo
addcontwo =
    0.8781
>> dev
dev =
    0.6787
```

As noted earlier, in principle one can use the L_1 norm in city-block scaling, but given the increased computational expense of doing so with no apparent advantage over L_2, it is suggested that the use of L_1 generally be avoided in favor of L_2.

5 Incorporating Monotonic Transformation of a Proximity Matrix in Fitting Multiple Unidimensional Scales: L_2 Nonmetric Multidimensional Scaling in the City-Block Metric

As a direct extension of the m-function `biscalqa.m` discussed in the last section, Appendix A.9 gives `bimonscalqa.m` which provides an optimal monotonic transformation (by incorporating the use of `proxmon.m` discussed in HAM2002) of the original proximity matrix given as input in addition to the later's bidimensional scaling. To prevent degeneracy, the sum-of-squares value for the initial input proximity matrix is maintained in the optimally transformed proximities; the overall strategy is iterative with termination again dependent on a change in the variance-accounted-for being less than 1.0e-005. The usage syntax is almost identical to that of `biscalqa.m` except for the inclusion of the monotonically transformed proximity matrix `MONPROX` as an output matrix:

```
[ ... monprox] = bimonscalqa( ... )
```

The ellipses indicate that the same items should be used as in `biscalqa.m`. If `bimonscalqa` would have been used in the numerical example of the previous section, the same results given would have been provided initially plus the results for the optimally transformed proximity matrix. We give this additional output below, which shows that the incorporation of an optimal monotonic transformation provides an increase in the `VAF` from .8243 to .9362; the orderings on the two dimensions remain the same as well as the nice substantive explanation of the previous section.

```
>> outpermone
outpermone =
     1     2     3     4     5     6     7     9     8    10
>> outpermtwo
outpermtwo =
     9     5     3     1     7     4    10     2     8     6
>> coordone'
```

```
ans =
  Columns 1 through 6
   -2.3514   -1.3290   -0.6409   -0.3565    0.0775    0.1216
  Columns 7 through 10
    0.5857    1.1342    1.1342    1.6247
>> coordtwo'
ans =
  Columns 1 through 6
   -0.7793   -0.7793   -0.7793   -0.3891   -0.1196    0.3242
  Columns 7 through 10
    0.3242    0.3480    0.8467    1.0035
>> addconone
addconone =
    1.4394
>> addcontwo
addcontwo =
    0.7922
>> vaf
vaf =
    0.9362
>> monprox
monprox =
  Columns 1 through 6
         0   -0.7387   -0.1667    0.5067    0.5067    1.4791
   -0.7387         0   -0.8218   -0.8218    0.5067   -0.1667
   -0.1667   -0.8218         0   -0.8218   -1.6174    0.5067
    0.5067   -0.8218   -0.8218         0   -0.7387   -0.7387
    0.5067    0.5067   -1.6174   -0.7387         0   -0.7387
    1.4791   -0.1667    0.5067   -0.7387   -0.7387         0
    1.0321    0.5067   -0.7387   -0.8218   -0.8218   -0.8218
    2.6590    0.5067    1.0321   -0.1667    0.5067   -0.8218
    1.7609    1.0321   -0.8218    1.0321   -1.2541    0.5067
    2.6231    0.5067    1.4791   -0.8218    0.5067   -0.0534
  Columns 7 through 10
    1.0321    2.6590    1.7609    2.6231
    0.5067    0.5067    1.0321    0.5067
   -0.7387    1.0321   -0.8218    1.4791
   -0.8218   -0.1667    1.0321   -0.8218
  -0.8218    0.5067   -1.2541    0.5067
  -0.8218   -0.8218    0.5067   -0.0534
        0   -0.7387   -0.8218   -0.8218
  -0.7387         0   -0.7387   -0.7387
  -0.8218   -0.7387         0   -0.8218
  -0.8218   -0.7387   -0.8218         0
```

Although we will not provide an example of its use here, Appendix A.10 gives `trimonscalqa.m`, which extends `triscalqa.m` (listed in A.7) to include an optimal monotonic transformation of whatever is given as the original input proximity matrix.

6 Concluding Comments

An obvious conclusion to this manuscript is that if one is interested in (nonmetric) city-block scaling in two or three dimensions within L_2, the MATLAB routines, `biscalqa.m` and `bimonscalqa.m`; `triscalqa.m` and `trimonscalqa.m`, would be natural alternatives to consider. The L_1 metric is not a computationally competitive alternative, and appears to be without any particular discernible advantage to make up for its greater expense. In any case, one aspect of all of these given m-files that we have not emphasized but will in these concluding comments, is their possible usage in the confirmatory context (by setting the `NOPT` switch to `0`), and fitting various fixed object orderings in multiple dimensions. One possible application of this type of confirmatory fitting would be in an individual differences scaling context. Explicitly, we begin with a collection of, say, K proximity matrices, $\mathbf{P}_1, \ldots, \mathbf{P}_K$, obtained from K separate sources, and through some weighting and averaging process construct a single aggregate proximity matrix, $\mathbf{P}_A$. On the basis of $\mathbf{P}_A$, suppose a two-dimensional city-block scaling is constructed (using, say, `biscalqa.m`); we label the latter the "common space" in consistency with what is usually done in the weighted Euclidean model (e.g., see the INDSCAL model of Carroll and Chang, 1970, or the PROXSCAL program in the Categories Module of SPSS — Busing, Commandeur, and Heiser, 1997). Each of the K proximity matrices then can be used in a confirmatory fitting of the object orders along the two axes. Thus, a very general "private space" is generated for each source and where the actually coordinates along both axes are unique to that source, subject only to the object order constraints of the group space. This strategy provides an individual differences model generalization over the usual weighted Euclidean model where the latter allows only differential axes scaling (stretching or shrinking) in generating the private spaces. The authors are continuing to work on these kinds of individual difference generalizations both for multiple unidimensional scalings in L_2 and for other types of proximity matrix representations such as ultrametrics or additive trees.

References

[1] Brusco, M. J. (2001). A simulated annealing heuristic for unidimensional and multidimensional (city-block) scaling of symmetric proximity matrices. *Journal of Classification*, *18*, 3–33.

[2] Brusco, M. J., & Stahl, S. (in press). Optimal least-squares unidimensional scaling: Improved branch-and-bound procedures and comparison to dynamic programming. *Psychometrika*, in press.

[3] Busing, F. M. T. A., Commandeur, J. J. F., & Heiser, W. J. (1997). PROXSCAL: A multidimensional scaling program for individual differences scaling with constraints. In W. Bandilla & F. Faulbaum (Eds.), *Softstat '97: Advances in Statistical Software, Volume 6* (pp. 67–74). Stuttgart: Lucius & Lucius.

[4] Carroll, J. D., & Chang, J. J. (1970). Analysis of individual differences in multidimensional scaling via an N-way generalization of Eckart-Young decomposition. *Psychometrika*, *35*, 283–319.

[5] Cheney, W., & Goldstein, A. (1959). Proximity maps for convex sets. *Proceedings of the American Mathematical Society*, *10*, 448–450.

[6] Groenen, P. J. F., Heiser, W. J., & Meulman, J. J. (1999). Global optimization in least-squares multidimensional scaling by distance smoothing. *Journal of Classification*, *16*, 225–254.

[7] Hubert, L. J., Arabie, R., & Hesson-McInnis, M. (1992). Multidimensional scaling in the city-block metric: A combinatorial approach. *Journal of Classification*, *9*, 211–236.

[8] Hubert, L. J., Arabie, R., & Meulman, J. J. (1997). Linear and circular unidimensional scaling for symmetric proximity matrices. *British Journal of Mathematical and Statitical Psychology*, *50*, 253–284.

[9] Hubert, L. J., Arabie, R., & Meulman, J. J. (2002). Linear unidimensional scaling in the L_2-norm: Basic optimization methods using MATLAB. *Journal of Classification*, *19*, 303–328.