A Combinatorial Data Analysis Toolbox for MATLAB

The HAM Team

June 2, 2003

Contents

I The Representation of Proximity Matrices by Structures Dependent on Order (Only) — the Order Structure Toolbox (OST) 3

1 Anti-Robinson (AR) Matrices for Symmetric Proximity Data 6
1.0.1 Incorporating Transformations . . . 7
1.0.2 Interpreting the Structure of an AR matrix . . . 8
1.1 Fitting a Given AR Matrix in the L_2-Norm . . . 10
1.1.1 Fitting the (In)-equality Constraints Implied by a Given Matrix in the L_2 Norm . . . 12
1.2 Finding an AR Matrix in the L_2-Norm . . . 13
1.3 Fitting and Finding a Strongly Anti-Robinson (SAR) Matrix in the L_2-Norm 17
1.4 The Use of Optimal Transformations and the m-function proxmon.m . . . 19
1.5 Representing SAR Structures (Graphically) . . . 25
1.6 Representation Through Multiple (Strongly) AR Matrices . . . 31

2 Circular-Anti-Robinson (CAR) Matrices for Symmetric Proximity Data 38
2.1 Fitting a Given CAR Matrix in the L_2-Norm . . . 40
2.1.1 The Circular Unidimensional Scaling Utilities for Constructing Circular Anti-Robinson Targets . . . 41
2.2 Finding a CAR Matrix in the L_2-Norm . . . 44
2.3 Fitting and Finding a Strongly Circular-Anti-Robinson (SCAR) Matrix in the L_2-Norm . . . 46
2.4 Representing SCAR Structures (Graphically) . . . 49
2.5 Representation Through Multiple (Strongly) CAR Matrices . . . 50

3 Order Structures for Two-Mode (Rectangular) Proximity Data 58

A main program files 61
A.1 arobfit.m . . . 61
A.2 targfit.m . . . 63
A.3 order.m . . . 66
A.4 arobfnd.m . . . 69
A.5 sarobfit.m . . . 70

A.6 sarobfnd.m 75
A.7 proxmon.m 76
A.8 biarobfnd.m 78
A.9 bisarobfnd.m 80
A.10 cirarobfit.m 81
A.11 cirfit.m 85
A.12 cirfitac.m 91
A.13 cirarobfnd.m 95
A.14 cirsarobfit.m 96
A.15 cirsarobfnd.m 105
A.16 bicirarobfnd.m 105
A.17 bicirsarobfnd.m 107

Part I

The Representation of Proximity Matrices by Structures Dependent on Order (Only) — the Order Structure Toolbox (OST)

Nonmetric multidimensional scaling (NMDS) as developed by Shepard (1962a,b) and Kruskal (1964a,b), has become a very familiar method in the psychological research literature for representing structure that may be inherent among a set of objects. Judging by the number of published substantive applications, whenever data are given in the form of a symmetric proximity matrix containing numerical relationship information between distinct object pairs, NMDS may have now become the default method of analysis. This routine use of NMDS, however, when faced with elucidating whatever pattern of relationships may underly a given set of proximities, does have interpretive implications and consequences. For one, there is an implicit choice made that whatever major generality will be allowed should reside primarily in the particular proximities being fitted by the explicitly parameterized (Euclidean) spatial structure. Thus, an optimal (usually monotonic) transformation of the proximities is sought in conjunction with the construction of a spatial representation. Second, the parameterized spatial structure implicitly involves fitting the (transformed) proximities by some function of the differences in object placement along a set of coordinate axes that may be best suited for representing object variation that could, at least in theory, be allowed to vary continuously. For instance, in the common Euclidean model we use the square root of the sum of squared coordinate differences along a set of axes (although the particular axis system selected is open to some arbitrariness). The tacit implication is that if the structure underlying the proximities is more classificatory (and discrete) in nature, we may not do very well in representing it by a spatial model that should do much better in the presence of more continuous variation (cf. Pruzansky, Tversky, and Carroll, 1982). In fact, in the limiting case where there exists a partition of the object set in which all proximities for object pairs within an object class are smaller than for object pairs between classes (and where proximities are keyed as dissimilarities so that larger values represent more dissimilar objects), NMDS will typically give a degenerate representation in which all objects within each class are located at the same spatial location and the optimally transformed proximities consist of just two values, one for the within-class proximities and one for the between-class proximities (cf. Shepard, 1974).

The present Toolbox concentrates on an alternative approach to understanding what a given proximity matrix may be depicting about the objects on which it was constructed, and one that does not require a prior commitment to the sole use of either some form of dimensional model (as in NMDS), or one that is strictly classificatory (as in the use of a partition hierarchy and the implicit fitting of an ultrametric that serves as the representational mechanism for the hierarchical clustering). The method of analysis is based on approximating a given proximity matrix additively by a sum of matrices, where each component in the sum is subject to specific patterning restrictions on its entries. The restrictions imposed on each component of the decomposition (to be referred to as matrices with anti-Robinson forms) are very general and encompass interpretations that might be dimensional, or classificatory, or some combination of both (e.g., through object classes that are themselves placed dimensionally in some space). Thus, as one special case — and particularly when an (optimal) transformation of the proximities is also permitted (as we will generally allow), proximity matrices that are well interpretable through NMDS should also be interpretable through an

additive decomposition of the (transformed) proximity matrix. Alternatively, when classificatory structures of various kinds might underlie a set of proximities (and the direct use of NMDS could possibly lead to a degeneracy), additive decompositions may still provide an analysis strategy for elucidating the structure.

The algorithmic details of fitting to a given proximity matrix a sum of matrices each having the desired general patterning to its entries (or even more explicitly parameterized forms that may be of help in providing a detailed interpretation, such as those given by partition hierarchies or unidimensional scales), are available in a series of papers that have appeared recently in the literature (i.e., Hubert and Arabie, 1994, 1995; Hubert, Arabie, and Meulman, 1997, 1998). Thus, in this sequel we can merely refer to these sources for the actual mechanics of carrying out the various decompositions. More unique aspects that will be incorporated in this Toolbox and in the documentation to follow are (a) the possible integration of (optimal) transformations for use with the originally given proximities to be fit by an additive matrix decomposition, and (b) the fitting of more restrictive parameterized forms to the various components of a decomposition in attempting to give a detailed substantive interpretation of what each separate matrix in the decomposition may be depicting. In this latter instance, one of our concerns might be directed toward the issue of whether a particular matrix as part of a decomposition is indicating primarily dimensional or classificatory aspects of the original proximities (or possibly and what may be more typical, some combination of the two). In these latter cases, the m-files discussed as part of the documentation for the LUS (Linear Unidimensional Scaling) and TS (Tree Structure) TOolboxes are particularly relevant.

Chapter 1

Anti-Robinson (AR) Matrices for Symmetric Proximity Data

Denoting an arbitrary symmetric $n \times n$ matrix by $\mathbf{A} = \{a_{ij}\}$, where the main diagonal entries are considered irrelevant and assumed to be zero (i.e., $a_{ii} = 0$ for $1 \leq i \leq n$), $\mathbf{A}$ is said to have an anti-Robinson (AR) form if after some reordering of the rows and columns of $\mathbf{A}$, the entries within each row and column have a distinctive pattern: moving away from the zero main diagonal entry within any row or any column, the entries never decrease. Generally, matrices having AR forms can appear both in spatial representations for a set of proximities as functions of the absolute differences in coordinate values along some axis, or for classificatory structures that are characterized through an ultrametric.

To illustrate, we first let $\mathbf{P} = \{p_{ij}\}$ be a given $n \times n$ proximity (dissimilarity) matrix among the n objects in a set $S = \{O_1, O_2, \ldots, O_n\}$ (where $p_{ii} = 0$ for $1 \leq i \leq n$). Then, suppose, for example, a two-dimensional Euclidean representation is possible for $\mathbf{P}$ and its entries are very well representable by the distances in this space, so

$$p_{ij} \approx \sqrt{(x_{1i} - x_{1j})^2 + (x_{2i} - x_{2j})^2} \ ,$$

where x_{ki} and x_{kj} are the coordinates on the k^{th} axis (for $k = 1$ and 2) for objects O_i and O_j (and the symbol $\approx$ is used to indicate approximation). Here, a simple monotonic transformation (squaring) of the proximities should then be fit well by the sum of two matrices both having AR forms, i.e.,

$$\{p_{ij}^2\} \approx \{(x_{1i} - x_{1j})^2\} + \{(x_{2i} - x_{2j})^2\}.$$

In a classificatory framework, if $\{p_{ij}\}$ were well representable, say, as a sum of two matrices, $\mathbf{A}_1 = \{a_{ij}^{(1)}\}$ and $\mathbf{A}_2 = \{a_{ij}^{(2)}\}$, each satisfying the ultrametric inequality, i.e., $a_{ij}^{(k)} \leq \max\{a_{ih}^{(k)}, a_{hj}^{(k)}\}$ for $k = 1$ and 2, then

$$\{p_{ij}\} \approx \{a_{ij}^{(1)}\} + \{a_{ij}^{(2)}\},$$

and each of the constituent matrices can be reordered to display an AR form. As can be seen in the TST (Tree Structure Toolbox), any matrix whose entries satisfy the ultrametric inequality can be represented by a sequence of partitions that are hierarchically related.

Given some proximity matrix $\mathbf{P}$, the task of approximating it as a sum of matrices each having an AR form is implemented through an iterative optimization strategy based on a least-squares loss criterion that is discussed in detail by Hubert and Arabie (1994). Given the manner in which the optimization process is carried out sequentially, each successive AR matrix in any decomposition generally accounts for less and less of the patterning of the original proximity information (and very analogous to what is typically observed in a principal component decomposition of a covariance matrix). In fact, it has been found empirically that for the many data sets we have analyzed, only a very small number of such AR matrices are ever necessary to represent almost all of the patterning in the given proximities. As a succinct summary that we could give to this empirical experience: no more than three AR matrices are ever necessary; the data analyst can usually get by with two; and sometimes one will suffice.

The substantive challenge that remains, once a well-fitting decomposition is found for a given proximity matrix, is to interpret substantively what each term in the decomposition might be depicting. The strategy that could be followed would approximate each separate AR matrix by ones having a more restrictive form, and usually those representing some type of unidimensional scale (from the LUS Toolbox) or partition hierarchy (from the TS Toolbox). As part of this interpretive process, an evaluation could be made of the degree to which classificatory or dimensional interpretations may best represent each AR matrix in the given decomposition.

1.0.1 Incorporating Transformations

One generalization that we will now allow to what has already been discussed in the literature for fitting sums of AR matrices to a proximity matrix $\mathbf{P}$, is the possible inclusion of an (optimal) transformation of the proximities. Thus, instead of just representing $\mathbf{P}$ as a sum of K matrices (and generally, for K very small) that we might denote as $\mathbf{A}_1 + \cdots + \mathbf{A}_K$, where each $\mathbf{A}_k$, $1 \leq k \leq K$, has an AR form, an (optimally) transformed matrix $\tilde{\mathbf{P}} = \{\tilde{p}_{ij}\}$ will be fitted by such a sum, say, $\tilde{\mathbf{A}}_1 + \cdots + \tilde{\mathbf{A}}_K$, where the entries in $\tilde{\mathbf{P}}$ are monotonic with respect to those in $\mathbf{P}$, i.e., for all $O_i, O_j, O_k, O_l \in S$, $p_{ij} < p_{kl} \Rightarrow \tilde{p}_{ij} \leq \tilde{p}_{kl}$. In the sequel we will rely on the m-file, `proxmon.m`, documented in the LUS Toolbox, which constructs optimal monotonic transformations by the same method of isotonic regression commonly used in NMDS (i.e., Kruskal's [1964a,b] primary approach to tied proximities in $\mathbf{P}$ that are allowed to be untied after transformation). Such transformations, for example, form the default option in the implementation of NMDS in the program KYST-2A (Kruskal, Young, and Seery, 1977) and in SYSTAT (Wilkinson, 1988).

The process of finding $\tilde{\mathbf{P}}$ and $\tilde{\mathbf{A}}_1 + \cdots + \tilde{\mathbf{A}}_K$ proceeds iteratively, with the original proximity matrix $\mathbf{P}$ first fit by $\mathbf{A}_1 + \cdots + \mathbf{A}_K$; a subsequent optimal (monotonic) transformation of $\mathbf{P}$ (through a least-squares approximation to $\mathbf{A}_1 + \cdots + \mathbf{A}_K$) is identified, which is then

refitted by the matrix sum. In many cases, this whole process can now be cycled through iteratively until convergence, i.e., through a sequential fitting and refitting of the optimally transformed proximities and its representation as a sum of matrices each having an AR form. In some contexts, however (particularly when fitting a single AR matrix [i.e., when $K = 1$]), it is probably best not to proceed to a complete convergence but instead to terminate the process after only a single optimal monotonic transformation of $\mathbf{P}$ is identified and then to refit by a matrix sum. This usage will be referred to as a single iteration optimal transformation (SIOT). If carried through to convergence, a perfect representation may be obtained but only at the expense of losing almost all the patterning contained within the original proximity matrix. For example, in fitting a single AR matrix, the optimal transformation identified after convergence might consist of just two values, with one corresponding to the smallest proximity in the original matrix and all others equal. Although technically permissible since this situation does reflect a perfect AR form, most of the detail present in the original proximity matrix is also lost. Difficulties with such so-called degeneracies have been pointed out by Carroll (1992), particularly when faced with fitting classificatory structures to a given proximity matrix.

1.0.2 Interpreting the Structure of an AR matrix

In representing a proximity matrix $\mathbf{P}$ as a sum, $\mathbf{A}_1 + \cdots + \mathbf{A}_K$ (or an optimal transformation $\tilde{\mathbf{P}}$ as $\tilde{\mathbf{A}}_1 + \cdots + \tilde{\mathbf{A}}_K$), the interpretive task remains to explain substantively what each term of the decomposition might be depicting. We suggest four possible strategies below, with the first two attempting to understand the structure of an AR matrix directly and without much loss of detail; the last two require the imposition of strictly parameterized approximations in the form of either an ultrametric or a unidimensional scale. In the discussion below, $\mathbf{A} = \{a_{ij}\}$ will be assumed to have an AR form that is displayed by the given row and column order.

(A) Complete representation and reconstruction through a collection of subsets and associated subset diameters:

The entries in any AR matrix $\mathbf{A}$ can be reconstructed exactly through a collection of M subsets of the original object set $S = \{O_1, \ldots, O_n\}$, denoted by $S_1, \ldots, S_M$, and where M is determined by the particular pattern of tied entries, if any, in $\mathbf{A}$. These M subsets have the following characteristics:

(i) each S_m, $1 \leq m \leq M$, consists of a sequence of (two or more) consecutive integers so that $M \leq n(n-1)/2$. (This bound holds because the number of different subsets having consecutive integers for any given fixed ordering is $n(n-1)/2$, and will be achieved if all the entries in the AR matrix $\mathbf{A}$ are distinct).

(ii) each S_m, $1 \leq m \leq M$, has a diameter, denoted by $d(S_m)$, so that for all object pairs within S_m, the corresponding entries in $\mathbf{A}$ are less than or equal to the diameter. The subsets, $S_1, \ldots, S_M$, can be assumed ordered as $d(S_1) \leq d(S_2) \leq \cdots \leq d(S_M)$, and if $S_m \subseteq S_{m'}$, $d(S_m) \leq d(S_{m'})$.

(iii) each entry in $\mathbf{A}$ can be reconstructed from $d(S_1), \ldots, d(S_M)$, i.e., for $1 \leq i, j \leq n$,

$$a_{ij} = \min_{1 \leq m \leq M} \{d(S_m) \mid O_i, O_j \in S_m\},$$

so the minimum diameter for subsets containing an object pair $O_i, O_j \in S$ is equal to a_{ij}. Given $\mathbf{A}$, the collection of subsets $S_1, \ldots, S_M$ and their diameters can be identified by inspection through the use of an increasing threshold that starts from the smallest entry in $\mathbf{A}$, and observing which subsets containing contiguous objects emerge from this process. The substantive interpretation of what $\mathbf{A}$ is depicting reduces to explaining why those subsets with the smallest diameters are so homogenous. For convenience of reference, the subsets $S_1, \ldots, S_M$ will be referred to as the set of AR reconstructive subsets.

(B) Representation by a strongly anti-Robinson matrix:

If the matrix $\mathbf{A}$ has a somewhat more restrictive form than just being AR, and is also *strongly* anti-Robinson, a convenient graphical representation can be given to the collection of AR reconstructive subsets $S_1, \ldots, S_M$ and their diameters, and how they can serve to retrieve $\mathbf{A}$. Specifically, $\mathbf{A}$ is said to be strongly anti-Robinson (SAR) if (considering the above-diagonal entries of $\mathbf{A}$) whenever two entries in adjacent columns are equal ($a_{ij} = a_{i(j+1)}$), those in the same two adjacent columns in the previous row are also equal ($a_{(i-1)j} = a_{(i-1)(j+1)}$ for $1 \leq i-1 < j \leq n-1$); also, whenever two entries in adjacent rows are equal ($a_{ij} = a_{(i+1)j}$), those in the same two adjacent rows in the succeeding column are also equal ($a_{i(j+1)} = a_{(i+1)(j+1)}$ for $2 \leq i+1 < j \leq n-1$).

When $\mathbf{A}$ is SAR, the collection of subsets, $S_1, \ldots, S_M$, and their diameters, and how these serve to reconstruct $\mathbf{A}$ can be modeled graphically as we will see in Section 1.5. The internal nodes (represented by solid circles) in each of these figures are at a height equal to the diameter of the respective subset; the consecutive objects forming that subset are identifiable by downward paths from the internal nodes to the terminal nodes corresponding to the objects in $S = \{O_1, \ldots, O_n\}$ (represented by labeled open circles). An entry a_{ij} in $\mathbf{A}$ can be reconstructed as the minimum node height of a subset for which a path can be constructed from O_i up to that internal node and then back down to O_j. (To prevent undue graphical "clutter", only the most homogenous subsets from $S_1, \ldots, S_M$ having the smallest diameters should actually be included in the graphical representation of an SAR matrix; each figure would explicitly show only how the smallest entries in $\mathbf{A}$ can be reconstructed, although each could be easily extended to include all of $\mathbf{A}$. The calibrated vertical axis in such figures could routinely include the heights at which the additional internal nodes would have to be placed to effect such a complete reconstruction.)

Given an arbitrary AR matrix $\mathbf{A}$, a least-squares SAR approximating matrix to $\mathbf{A}$ can be found using the heuristic optimization search strategy illustrated in Section 1.3 and developed in Hubert, Arabie, and Meulman (1998). This latter source also discusses in detail (through counterexample) why strongly AR conditions need to be imposed to obtain a consistent graphical representation.

(C) Representation by a unidimensional scale:

To obtain greater graphical simplicity for an eventual substantive interpretation than offered by an SAR matrix, one possibility is to use approximating unidimensional scales. To be explicit, one very simple form that an AR matrix $\mathbf{A}$ may assume is interpretable by a single dimension and through a unidimensional scale in which the entries have the parameterized form, $\mathbf{A} = \{a_{ij}\} = \{|\ x_j - x_i\ | + c\}$, where the coordinates are ordered as $x_1 \leq x_2 \leq \cdots \leq x_n$ and c is an estimated constant. Given any proximity matrix, a least-squares approximating unidimensional scale can be obtained through the optimization strategies of the LUS Toolbox, and would be one (dimensional) method that could be followed in attempting to interpret what a particular AR component of a decomposition might be revealing.

(D) Representation by an ultrametric:

A second simple form that an AR matrix $\mathbf{A}$ could have is strictly classificatory in which the entries in $\mathbf{A}$ satisfy the ultrametric condition: $a_{ij} \leq \max\{a_{ik}, a_{jk}\}$ for all $O_i, O_j, O_k \in S$. As a threshold is increased from the smallest entry in $\mathbf{A}$, a sequence of partitions of S is identified in which each partition is constructed from the previous one by uniting pairs of subsets from the latter. A partition identified at a given threshold level has equal values in $\mathbf{A}$ between each given pair of subsets, and all the within subset values are not greater than the between subset values. The reconstructive subsets $S_1, \ldots, S_M$ that would represent the AR matrix $\mathbf{A}$ are now the new subsets that are formed in the sequence of partitions, and have the property that if $d(S_m) \leq d(S_{m'})$, then $S_m \subseteq S_{m'}$ or $S_m \cap S_{m'} = \oslash$. Given any proximity matrix, a least-squares approximating ultrametric can be constructed by the heuristic optimization routines developed in the TS Toolbox, and would be another (classificatory) strategy for interpreting what a particular AR component of a decomposition might be depicting. As might be noted, there are generally $n-1$ subsets (each of size greater than one) in the collection of reconstructive subsets for any ultrametric, and thus $n-1$ values need to be estimated in finding the least-squares approximation (which is the same number needed for a least-squares approximating unidimensional scale, based on obtaining the $n-1$ nonnegative separation values between x_i and x_{i+1} for $1 \leq i \leq n-1$).

1.1 Fitting a Given AR Matrix in the L_2-Norm

The MATLAB function m-file given in Section A.1 of Appendix A, `arobfit.m`, fits an anti-Robinson matrix using iterative projection to a symmetric proximity matrix in the L_2-norm. The usage syntax is of the form

```
[fit,vaf] = arobfit(prox,inperm)
```

where `PROX` is the input proximity matrix ($n \times n$ with a zero main diagonal and a dissimilarity interpretation); `INPERM` is a given permutation of the first n integers; `FIT` is the least-squares optimal matrix (with variance-accounted-for of `VAF`) to `PROX` having an anti-Robinson form for the row and column object ordering given by `INPERM`. A recording of a MATLAB session using the `number.dat` data file and object ordering given by the identity permutation follows:

```
load number.dat
inperm = 1:10

inperm =

     1     2     3     4     5     6     7     8     9    10

[fit,vaf] = arobfit(number,inperm)

fit =

  Columns 1 through 7

         0    0.4210    0.5840    0.6965    0.6965    0.7960    0.7960
    0.4210         0    0.2840    0.3460    0.6170    0.6170    0.6940
    0.5840    0.2840         0    0.2753    0.2753    0.5460    0.5460
    0.6965    0.3460    0.2753         0    0.2753    0.3844    0.3844
    0.6965    0.6170    0.2753    0.2753         0    0.3844    0.3844
    0.7960    0.6170    0.5460    0.3844    0.3844         0    0.3844
    0.7960    0.6940    0.5460    0.3844    0.3844    0.3844         0
    0.8600    0.6940    0.5853    0.5853    0.5530    0.4000    0.3857
    0.8600    0.7413    0.5853    0.5853    0.5530    0.5530    0.3857
    0.8600    0.7413    0.7413    0.5853    0.5853    0.5853    0.3857

  Columns 8 through 10

    0.8600    0.8600    0.8600
    0.6940    0.7413    0.7413
    0.5853    0.5853    0.7413
    0.5853    0.5853    0.5853
    0.5530    0.5530    0.5853
    0.4000    0.5530    0.5853
    0.3857    0.3857    0.3857
         0    0.3857    0.3857
    0.3857         0    0.3857
    0.3857    0.3857         0


vaf =

    0.6979
```

1.1.1 Fitting the (In)-equality Constraints Implied by a Given Matrix in the L_2 Norm

At times it may be useful to fit through iterative projection a given set of equality and inequality constraints (as represented by the equalities and inequalities present among the entries in a given target matrix) to a symmetric proximity matrix in the L_2-norm. If the target matrix is AR in form already, the resulting fitted matrix would also AR in form, but the m-function, `targfit.m`, could be used more generally with any chosen target matrix (given in Section A.2). The usage follows the form

```
[fit,vaf] = targfit(prox,targ)
```

where, as usual, `PROX` is the input proximity matrix (with a zero main diagonal and a dissimilarity interpretation); `TARG` is a matrix of the same size as `PROX`; `FIT` is the least-squares optimal matrix (with variance-accounted-for of `VAF`) to `PROX` satisfying the equality and inequality constraints implicit among all the entries in `TARG`. An example follows in which the given target matrix is a distance matrix (having an AR form) between equally-spaced object placements along a line; the resulting fitted matrix obviously has an AR form as well:

```
load number.dat
[prox10 targlin targcir] = ransymat(10);
targlin

targlin =

     0     1     2     3     4     5     6     7     8     9
     1     0     1     2     3     4     5     6     7     8
     2     1     0     1     2     3     4     5     6     7
     3     2     1     0     1     2     3     4     5     6
     4     3     2     1     0     1     2     3     4     5
     5     4     3     2     1     0     1     2     3     4
     6     5     4     3     2     1     0     1     2     3
     7     6     5     4     3     2     1     0     1     2
     8     7     6     5     4     3     2     1     0     1
     9     8     7     6     5     4     3     2     1     0

[fit,vaf] = targfit(number,targlin)

fit =

  Columns 1 through 7
```

```
         0    0.3714    0.3714    0.5363    0.5363    0.6548    0.6548
    0.3714         0    0.3714    0.3714    0.5363    0.5363    0.6548
    0.3714    0.3714         0    0.3714    0.3714    0.5363    0.5363
    0.5363    0.3714    0.3714         0    0.3714    0.3714    0.5363
    0.5363    0.5363    0.3714    0.3714         0    0.3714    0.3714
    0.6548    0.5363    0.5363    0.3714    0.3714         0    0.3714
    0.6548    0.6548    0.5363    0.5363    0.3714    0.3714         0
    0.7908    0.6548    0.6548    0.5363    0.5363    0.3714    0.3714
    0.7908    0.7908    0.6548    0.6548    0.5363    0.5363    0.3714
    0.8500    0.7908    0.7908    0.6548    0.6548    0.5363    0.5363

  Columns 8 through 10

    0.7908    0.7908    0.8500
    0.6548    0.7908    0.7908
    0.6548    0.6548    0.7908
    0.5363    0.6548    0.6548
    0.5363    0.5363    0.6548
    0.3714    0.5363    0.5363
    0.3714    0.3714    0.5363
         0    0.3714    0.3714
    0.3714         0    0.3714
    0.3714    0.3714         0


vaf =

    0.5105
```

1.2 Finding an AR Matrix in the L_2-Norm

The *fitting* of a given AR matrix by the m-function of Section 1.1, `arobfit.m`, requires the presence of a beginning permutation to direct the optimization process. Thus, the *finding* of a best-fitting AR matrix reduces to the identification of an appropriate object permutation to use in the first place. We suggest the adoption of `order.m` in Appendix A.3, which carries out an iterative Quadratic Assignment maximization task using a given square $(n \times n)$ proximity matrix `PROX` (with a zero main diagonal and a dissimilarity interpretation). Three separate local operations are used to permute the rows and columns of the proximity matrix to maximize the cross-product index with respect to a given square target matrix `TARG`: pairwise interchanges of objects in the permutation defining the row and column order of the square proximity matrix; the insertion of from 1 to `KBLOCK` (which is less than or equal to $n-1$) consecutive objects in the permutation defining the row and column order of the data

matrix; the rotation of from 2 to KBLOCK (which is less than or equal to $n-1$) consecutive objects in the permutation defining the row and column order of the data matrix. The usage syntax has the form

```
[outperm,rawindex,allperms,index] =  order(prox,targ,inperm,kblock)
```

where INPERM is the input beginning permutation (a permutation of the first n integers); OUTPERM is the final permutation of PROX with the cross-product index RAWINDEX with respect to TARG. The cell array ALLPERMS contains INDEX entries corresponding to all the permutations identified in the optimization from ALLPERMS{1} = INPERM to ALLPERMS{INDEX} = OUTPERM.

A recording of a MATLAB session using order.m is listed below with the beginning INPERM given as the identity permutation, TARG by an equally-spaced object placement along a line, and KBLOCK set at 3. Based upon the generated OUTPERM, arobfit.m is then invoked to fit an AR form having final VAF of .7782.

```
load number.dat
[prox10,targlin,targcir] = ransymat(10);
[outperm,rawindex,allperms,index] =  order(number,targlin,1:10,3)

outperm =

     1     2     3     5     4     6     7     9    10     8


rawindex =

  206.4920


allperms =

    [1x10 double]    [1x10 double]    [1x10 double]    [1x10 double]


index =

     4

[fit, vaf] = arobfit(number, outperm)

fit =
```

```
  Columns 1 through 7

         0    0.4210    0.5840    0.6840    0.7090    0.7960    0.7960
    0.4210         0    0.2840    0.4960    0.4960    0.5880    0.7357
    0.5840    0.2840         0    0.0590    0.3835    0.4928    0.4928
    0.6840    0.4960    0.0590         0    0.3835    0.3985    0.3985
    0.7090    0.4960    0.3835    0.3835         0    0.3750    0.3750
    0.7960    0.5880    0.4928    0.3985    0.3750         0    0.3750
    0.7960    0.7357    0.4928    0.3985    0.3750    0.3750         0
    0.8210    0.7357    0.4928    0.4928    0.4928    0.4928    0.3460
    0.8500    0.7357    0.7357    0.6830    0.4928    0.4928    0.3460
    0.9090    0.7357    0.7357    0.7357    0.5920    0.4928    0.4253

  Columns 8 through 10

    0.8210    0.8500    0.9090
    0.7357    0.7357    0.7357
    0.4928    0.7357    0.7357
    0.4928    0.6830    0.7357
    0.4928    0.4928    0.5920
    0.4928    0.4928    0.4928
    0.3460    0.3460    0.4253
         0    0.3460    0.4253
    0.3460         0    0.4253
    0.4253    0.4253         0


vaf =

    0.7782
```

The m-file of Section A.4, `arobfnd.m` is our preferred method for actually identifying a single AR form. It incorporates an initial equally-spaced target and uses the iterative QA routine of `order.m` to generate better permutations; the obtained AR forms then are used as new targets against which possibly even better permutations might be identified, until convergence (i.e., the identified permutations remain the same). The syntax is as follows:

```
[fit, vaf, outperm] = arobfnd(prox, inperm, kblock)
```

where `PROX` is the input proximity matrix ($n \times n$ with a zero main diagonal and a dissimilarity interpretation); `INPERM` is a given starting permutation of the first n integers; `FIT` is the least-squares optimal matrix (with variance-accounted-for of `VAF`) to `PROX` having an anti-Robinson form for the row and column object ordering given by the ending permutation `OUTPERM`; `KBLOCK` defines the block size in the use the iterative quadratic assignment routine.

As seen from the example below, and starting from a random initial permutation, the same AR form is found as with just one application of `order.m` reported above.

```
[fit, vaf, outperm] = arobfnd(number, randperm(10), 1)

fit =

  Columns 1 through 7

         0    0.4253    0.4253    0.4253    0.4928    0.5920    0.7357
    0.4253         0    0.3460    0.3460    0.4928    0.4928    0.6830
    0.4253    0.3460         0    0.3460    0.4928    0.4928    0.4928
    0.4253    0.3460    0.3460         0    0.3750    0.3750    0.3985
    0.4928    0.4928    0.4928    0.3750         0    0.3750    0.3985
    0.5920    0.4928    0.4928    0.3750    0.3750         0    0.3835
    0.7357    0.6830    0.4928    0.3985    0.3985    0.3835         0
    0.7357    0.7357    0.4928    0.4928    0.4928    0.3835    0.0590
    0.7357    0.7357    0.7357    0.7357    0.5880    0.4960    0.4960
    0.9090    0.8500    0.8210    0.7960    0.7960    0.7090    0.6840

  Columns 8 through 10

    0.7357    0.7357    0.9090
    0.7357    0.7357    0.8500
    0.4928    0.7357    0.8210
    0.4928    0.7357    0.7960
    0.4928    0.5880    0.7960
    0.3835    0.4960    0.7090
    0.0590    0.4960    0.6840
         0    0.2840    0.5840
    0.2840         0    0.4210
    0.5840    0.4210         0


vaf =

    0.7782


outperm =

     8    10     9     7     6     4     5     3     2     1
```

1.3 Fitting and Finding a Strongly Anti-Robinson (SAR) Matrix in the L_2-Norm

The two m-functions in Sections A.5 (`sarobfit.m`) and A.6 (`sarobfnd.m`) are direct analogues of `arobfit.m` and `arobfnd.m`, respectively, but are concerned with fitting and finding *strongly* anti-Robinson forms. The syntax for `sarobfit.m`, which fits a strongly anti-Robinson matrix using iterative projection to a symmetric proximity matrix in the L_2-norm, is

```
[fit, vaf] = sarobfit(prox, inperm)
```

where, again, `PROX` is the input proximity matrix ($n \times n$ with a zero main diagonal and a dissimilarity interpretation); `INPERM` is a given permutation of the first n integers; `FIT` is the least-squares optimal matrix (with variance-accounted-for of `VAF`) to `PROX` having a strongly anti-Robinson form for the row and column object ordering given by `INPERM`.

An example follows using the same identity permutation as was done in fitting an AR form with `arobfit.m`; as might be expected from using the more restrictive strongly anti-Robinson form, the variance-accounted-for drops to .6128 from .6979.

```
load number.dat
[fit,vaf] = sarobfit(number,1:10)

fit =

  Columns 1 through 7

         0    0.4210    0.5840    0.6965    0.6965    0.7960    0.7960
    0.4210         0    0.2840    0.4960    0.4960    0.6730    0.6730
    0.5840    0.2840         0    0.2753    0.2753    0.4553    0.4553
    0.6965    0.4960    0.2753         0    0.2753    0.4553    0.4553
    0.6965    0.4960    0.2753    0.2753         0    0.3977    0.3977
    0.7960    0.6730    0.4553    0.4553    0.3977         0    0.3977
    0.7960    0.6730    0.4553    0.4553    0.3977    0.3977         0
    0.8600    0.6820    0.6050    0.6050    0.5557    0.5557    0.3857
    0.8600    0.6820    0.6050    0.6050    0.5557    0.5557    0.3857
    0.8600    0.6820    0.6050    0.6050    0.5557    0.5557    0.3857

  Columns 8 through 10

    0.8600    0.8600    0.8600
    0.6820    0.6820    0.6820
    0.6050    0.6050    0.6050
```

```
    0.6050    0.6050    0.6050
    0.5557    0.5557    0.5557
    0.5557    0.5557    0.5557
    0.3857    0.3857    0.3857
         0    0.3857    0.3857
    0.3857         0    0.3857
    0.3857    0.3857         0


vaf =

    0.6128
```

The m-function `sarobfnd.m`, which fits a strongly anti-Robinson matrix using iterative projection to a symmetric proximity matrix in the L_2-norm based on a permutation identified through the use of iterative quadratic assignment, has the expected syntax

```
[fit, vaf, outperm] = sarobfnd(prox, inperm, kblock)
```

where, again, `PROX` is the input proximity matrix ($n \times n$ with a zero main diagonal and a dissimilarity interpretation); `INPERM` is a given starting permutation of the first n integers; `FIT` is the least-squares optimal matrix (with variance-accounted-for of `VAF`) to `PROX` having a strongly anti-Robinson form for the row and column object ordering given by the ending permutation `OUTPERM`. As usual, `KBLOCK` defines the block size in the use the iterative quadratic assignment routine.

In the MATLAB recording below, and starting from a random permutation, a strongly anti-Robinson form is found with a variance-accounted-for of .7210 (and is an expected drop from the value of .7782 for the anti-Robinson form found using `arobfnd.m`).

```
[fit,vaf,outperm] = sarobfnd(number,randperm(10),1)

fit =

  Columns 1 through 7

         0    0.4210    0.5840    0.6965    0.6965    0.7960    0.7960
    0.4210         0    0.2840    0.4960    0.4960    0.6730    0.6730
    0.5840    0.2840         0    0.0590    0.3835    0.4723    0.4723
    0.6965    0.4960    0.0590         0    0.3835    0.4723    0.4723
    0.6965    0.4960    0.3835    0.3835         0    0.3750    0.3750
    0.7960    0.6730    0.4723    0.4723    0.3750         0    0.3750
    0.7960    0.6730    0.4723    0.4723    0.3750    0.3750         0
    0.8355    0.7080    0.5714    0.5714    0.4275    0.4275    0.2960
```

```
    0.8355    0.7080    0.5714    0.5714    0.5714    0.5714    0.3710
    0.9090    0.7227    0.7227    0.7227    0.5714    0.5714    0.4380

  Columns 8 through 10

    0.8355    0.8355    0.9090
    0.7080    0.7080    0.7227
    0.5714    0.5714    0.7227
    0.5714    0.5714    0.7227
    0.4275    0.5714    0.5714
    0.4275    0.5714    0.5714
    0.2960    0.3710    0.4380
         0    0.3710    0.4380
    0.3710         0    0.4000
    0.4380    0.4000         0


vaf =

    0.7210


outperm =

     1     2     3     5     4     6     7    10     9     8
```

1.4 The Use of Optimal Transformations and the m-function proxmon.m

As previously discussed within the LUS Toolbox, the MATLAB function, `proxmon.m`, given again here in Section A.7, provides a monotonically transformed proximity matrix that is close in a least-squares sense to a given input matrix. The syntax is

```
[monproxpermut vaf diff] = proxmon(proxpermut,fitted)
```

where `PROXPERMUT` is the input proximity matrix (which may have been subjected to an initial row/column permutation, hence the suffix '`PERMUT`') and `FITTED` is a given target matrix; the output matrix `MONPROXPERMUT` is closest to `FITTED` in a least-squares sense and obeys the order constraints obtained from each pair of entries in (the upper-triangular portion of) `PROXPERMUT` (and where the inequality constrained optimization is carried out using the Dykstra-Kaczmarz iterative projection strategy); `VAF` denotes 'variance-accounted-for' and indicates how much variance in `MONPROXPERMUT` can be accounted for by `FITTED`; finally `DIFF`

is the value of the least-squares loss function and is (one-half) the sum of squared differences between the entries in MONPROXPERMUT and FITTED.

In the notation of the introduction when fitting a given order, FITTED would correspond to the AR matrix $\mathbf{A} = \{a_{ij}\}$; the input PROXPERMUT would be $\{p_{\rho^0(i)\rho^0(j)}\}$; MONPROXPERMUT would be $\{f(p_{\rho^0(i)\rho^0(j)})\}$, where the function $f(\cdot)$ satisfies the monotonicity constraints, i.e., if $p_{\rho^0(i)\rho^0(j)} < p_{\rho^0(i')\rho^0(j')}$ for $1 \le i < j \le n$ and $1 \le i' < j' \le n$, then $f(p_{\rho^0(i)\rho^0(j)}) \le f(p_{\rho^0(i')\rho^0(j')})$. The transformed proximity matrix $\{f(p_{\rho^0(i)\rho^0(j)})\}$ minimizes the least-squares criterion (DIFF) of

$$\sum_{i<j} (f(p_{\rho^0(i)\rho^0(j)}) - a_{ij})^2,$$

over all functions $f(\cdot)$ that satisfy the monotonicity constraints. The VAF is a normalization of this loss value by the sum of squared deviations of the transformed proximities from their mean:

$$\text{VAF} = 1 - \frac{\sum_{i<j} (f(p_{\rho^0(i)\rho^0(j)}) - a_{ij})^2}{\sum_{i<j} (f(p_{\rho^0(i)\rho^0(j)}) - \bar{f})^2},$$

where $\bar{f}$ denotes the mean of the off-diagonal entries in $\{f(p_{\rho^0(i)\rho^0(j)})\}$.

The script m-file listed below gives an application of proxmon.m along with finding a best fitting AR form for our number.dat matrix. First, arobfnd.m is invoked to obtain a best-fitting AR matrix (fit); this is the same as found before based on the outperm of [1 2 3 5 4 6 7 9 10 8] with a vaf of .7782. The m-file, proxmon.m, is then used to generate the monotonically transformed proximity matrix (monproxpermut) with vaf of .8323. Given the SIOT (single-iteration-optimal-transformation) discussion of the introduction, it might now be best to fit once more an AR matrix to this now montonically transformed proximity matrix, but then stop. Otherwise as seen in the output below, if the strategy is repeated cyclically (i.e., finding a fitted matrix based on the monotonically transformed proximity matrix, finding a new monotonically transformed matrix, and so on), a perfect vaf of 1.0 can be achieved at the expense of losing most of the detail in the transformed proximities, i.e., only five distinct values remain that correspond to the three largest and single smallest of the original proximities with *all* the remaining now tied at a value of .5467. (To avoid another type of degeneracy (where all matrices would converge to zeros), the sum of squares of the fitted matrix is kept the same as it was initially; convergence is based on observing a minimal change (less than 1.0e-010) in the vaf.

```
load number.dat
[fit vaf outperm] = arobfnd(number,randperm(10),2)
[monproxpermut vaf diff] = ...
    proxmon(number(outperm,outperm),fit)
sumfitsq = sum(sum(fit.^2));
prevvaf = 2;
while (abs(prevvaf-vaf) >= 1.0e-010)
   prevvaf = vaf;
   [fit vaf] = arobfit(monproxpermut,1:10);
```

```
   sumnewfitsq = sum(sum(fit.^2));
   fit = sqrt(sumfitsq)*(fit/sqrt(sumnewfitsq));
    [monproxpermut vaf diff] = proxmon(number(outperm,outperm), fit);
end

outperm
fit


monproxpermut
number(outperm,outperm)
vaf
diff

fit =

  Columns 1 through 7

         0    0.4210    0.5840    0.6840    0.7090    0.7960    0.7960
    0.4210         0    0.2840    0.4960    0.4960    0.5880    0.7357
    0.5840    0.2840         0    0.0590    0.3835    0.4928    0.4928
    0.6840    0.4960    0.0590         0    0.3835    0.3985    0.3985
    0.7090    0.4960    0.3835    0.3835         0    0.3750    0.3750
    0.7960    0.5880    0.4928    0.3985    0.3750         0    0.3750
    0.7960    0.7357    0.4928    0.3985    0.3750    0.3750         0
    0.8210    0.7357    0.4928    0.4928    0.4928    0.4928    0.3460
    0.8500    0.7357    0.7357    0.6830    0.4928    0.4928    0.3460
    0.9090    0.7357    0.7357    0.7357    0.5920    0.4928    0.4253

  Columns 8 through 10

    0.8210    0.8500    0.9090
    0.7357    0.7357    0.7357
    0.4928    0.7357    0.7357
    0.4928    0.6830    0.7357
    0.4928    0.4928    0.5920
    0.4928    0.4928    0.4928
    0.3460    0.3460    0.4253
         0    0.3460    0.4253
    0.3460         0    0.4253
    0.4253    0.4253         0
```

```
vaf =

    0.7782


outperm =

     1     2     3     5     4     6     7     9    10     8


monproxpermut =

  Columns 1 through 7

         0    0.4244    0.5549    0.6840    0.7058    0.7659    0.7058
    0.4244         0    0.3981    0.5908    0.4054    0.5549    0.7058
    0.5549    0.3981         0    0.0590    0.4054    0.5908    0.4310
    0.6840    0.5908    0.0590         0    0.4244    0.4244    0.4054
    0.7058    0.4054    0.4054    0.4244         0    0.4310    0.3981
    0.7659    0.5549    0.5908    0.4244    0.4310         0    0.4054
    0.7058    0.7058    0.4310    0.4054    0.3981    0.4054         0
    0.8210    0.7058    0.4054    0.3981    0.7058    0.5908    0.4054
    0.8500    0.5908    0.7659    0.6830    0.3981    0.5549    0.3981
    0.9090    0.5908    0.7058    0.7058    0.5908    0.4244    0.4244

  Columns 8 through 10

    0.8210    0.8500    0.9090
    0.7058    0.5908    0.5908
    0.4054    0.7659    0.7058
    0.3981    0.6830    0.7058
    0.7058    0.3981    0.5908
    0.5908    0.5549    0.4244
    0.4054    0.3981    0.4244
         0    0.4054    0.4244
    0.4054         0    0.4310
    0.4244    0.4310         0


vaf =

    0.8323
```

```
diff =

    0.2075


outperm =

     1     2     3     5     4     6     7     9    10     8


fit =

  Columns 1 through 7

         0    0.5467    0.5467    0.5467    0.5467    0.5467    0.5467
    0.5467         0    0.5467    0.5467    0.5467    0.5467    0.5467
    0.5467    0.5467         0    0.0609    0.5467    0.5467    0.5467
    0.5467    0.5467    0.0609         0    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467    0.5467         0    0.5467    0.5467
    0.5467    0.5467    0.5467    0.5467    0.5467         0    0.5467
    0.5467    0.5467    0.5467    0.5467    0.5467    0.5467         0
    0.8474    0.5467    0.5467    0.5467    0.5467    0.5467    0.5467
    0.8774    0.5467    0.5467    0.5467    0.5467    0.5467    0.5467
    0.9383    0.5467    0.5467    0.5467    0.5467    0.5467    0.5467

  Columns 8 through 10

    0.8474    0.8774    0.9383
    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467
         0    0.5467    0.5467
    0.5467         0    0.5467
    0.5467    0.5467         0


monproxpermut =
```

```
  Columns 1 through 7

         0    0.5467    0.5467    0.5467    0.5467    0.5467    0.5467
    0.5467         0    0.5467    0.5467    0.5467    0.5467    0.5467
    0.5467    0.5467         0    0.0609    0.5467    0.5467    0.5467
    0.5467    0.5467    0.0609         0    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467    0.5467         0    0.5467    0.5467
    0.5467    0.5467    0.5467    0.5467    0.5467         0    0.5467
    0.5467    0.5467    0.5467    0.5467    0.5467    0.5467         0
    0.8474    0.5467    0.5467    0.5467    0.5467    0.5467    0.5467
    0.8774    0.5467    0.5467    0.5467    0.5467    0.5467    0.5467
    0.9383    0.5467    0.5467    0.5467    0.5467    0.5467    0.5467

  Columns 8 through 10

    0.8474    0.8774    0.9383
    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467
    0.5467    0.5467    0.5467
         0    0.5467    0.5467
    0.5467         0    0.5467
    0.5467    0.5467         0


ans =

  Columns 1 through 7

         0    0.4210    0.5840    0.6840    0.7090    0.8040    0.7880
    0.4210         0    0.2840    0.6460    0.3460    0.5880    0.7580
    0.5840    0.2840         0    0.0590    0.3540    0.6710    0.4210
    0.6840    0.6460    0.0590         0    0.4130    0.4090    0.3880
    0.7090    0.3460    0.3540    0.4130         0    0.4290    0.3000
    0.8040    0.5880    0.6710    0.4090    0.4290         0    0.3960
    0.7880    0.7580    0.4210    0.3880    0.3000    0.3960         0
    0.8210    0.7910    0.3670    0.2460    0.8040    0.6710    0.3500
    0.8500    0.6250    0.8080    0.6830    0.2630    0.5920    0.2960
    0.9090    0.6300    0.7960    0.7420    0.5920    0.4000    0.4170
```

```
  Columns 8 through 10

    0.8210    0.8500    0.9090
    0.7910    0.6250    0.6300
    0.3670    0.8080    0.7960
    0.2460    0.6830    0.7420
    0.8040    0.2630    0.5920
    0.6710    0.5920    0.4000
    0.3500    0.2960    0.4170
         0    0.3920    0.4000
    0.3920         0    0.4590
    0.4000    0.4590         0


vaf =

    1.0000


diff =

  8.3999e-011
```

1.5 Representing SAR Structures (Graphically)

The use of the very general form of representation offered by an AR matrix without the imposition of any further restrictions has one annoying interpretive difficulty. Specifically, it is usually necessary to interpret the fitted structures directly (and enumeratively) through a set of subsets or clusters that are all defined by objects contiguous in a specific object ordering; each such subset has an attached diameter that reflects its maximum within-class fitted value. More pointedly, it is generally *not* possible to use a more convenient graph-theoretic structure and the lengths of paths between objects in such a graph to represent visually a fitted AR matrix; this situation contrasts with opportunities resulting when the approximation matrix is more restricted and defined, say, by an ultrametric or an additive tree, or by a (linear or circular) unidimensional scaling (see Hubert, Arabie, & Meulman, 1997).

As noted in the introduction, the imposition of SAR conditions permits a representation of the fitted values in a (least-squares) SAR approximating matrix as lengths of paths in a graph, although this graph will not generally have the simplified form of a tree. A discussion of these latter SAR constriants is not new here, and a number of (theoretical) presentations of their usefulness exist in the literature (for example, see Critchley and Fichet, 1994; Critchley,

TABLE 1

Order-constrained least-squares approximations to the digit proximity data of Shepard *et al.* (1975) in (the lower-triangular portion of) Table 1; the upper-triangular portion is anti-Robinson and the lower-triangular portion is strongly-anti-Robinson.

digit	0	1	2	4	3	5	6	8	9	7
0	x	3.41	4.21	4.70	4.83	5.25	5.25	5.38	5.52	5.81
1	3.41	x	2.73	3.78	3.78	4.23	4.96	4.96	4.96	4.96
2	4.21	2.73	x	1.63	3.22	3.76	3.76	3.76	4.96	4.96
4	4.76	3.78	1.63	x	3.22	3.30	3.30	3.76	4.70	4.96
3	4.76	3.78	3.22	3.22	x	3.18	3.18	3.76	3.76	4.25
5	5.25	4.59	3.53	3.53	3.18	x	3.18	3.76	3.76	3.76
6	5.25	4.59	3.53	3.53	3.18	3.18	x	3.04	3.04	3.43
8	5.57	4.96	4.18	4.18	4.18	4.18	3.04	x	3.04	3.43
9	5.57	4.96	4.18	4.18	4.18	4.18	3.04	3.04	x	3.43
7	5.57	4.96	4.18	4.18	4.18	4.18	3.43	3.43	3.43	x

1994; Durand and Fichet, 1988; Mirkin, 1996, Chapter 7). Here, we give the example based on the number data from Hubert, Arabie, and Meulman (1998) for interpretative convenience. The latter data were transformed (in that reference) to a standard deviation of 1.0 and a mean of 4.0; thus, the numbers within the fitted matrices will differ from the examples given earlier. Approximating AR and SAR forms for the transformed number proximity data are given in the upper and lower-triangular portions, respectively, of the matrix in Table 1. For convenience below, we will denote the upper-trianglar AR matrix by $\mathbf{A}_{ut}$ and the lower-triangular SAR matrix by $\mathbf{A}_{lt}$.

The $10(10-1)/2 = 45$ subsets defined by objects contiguous in the object ordering used to display the upper-triangular portion of Table 1 are listed in Table 2 according to increasing diameter values. For purposes of our later discussion, 22 of the subsets are given in italics to indicate that they are proper subsets of another listed subset having the same diameter. Substantively, the dominant patterning of the entries in $\mathbf{A}_{ut}$ appears to reflect (primarily) digit magnitude except for the placement of digit 4 next to 2, and digit 7 being located in the last position. Both these latter deviations from an interpretation strictly according to digit magnitude show some of the salient structural properties of the digits. For example, the digit pair (2,4) has the absolute smallest dissimilarity in the data; besides being relatively close in magnitude, there are the possible (although redundant) similarity bases that $2 + 2 = 4$, $2 \times 2 = 4$, 4 is a power of 2, and both 2 and 4 are even numbers. Similarly, the placement of the digit 7 in the last position results from the salience of the triple $\{6, 8, 9\}$, which is the third to emerge according to its diameter. In addition to these three digits all being relatively close in magnitude, 6 and 8 are both even numbers, 6 and 9 are multiples of 3, and 8 is directly adjacent in size to 9. The three original dissimilarities within the set $\{6, 8, 9\}$

are all smaller than the dissimilarities digit 7 has to *any* other digit.

Given just the collection of subsets $S_1, \ldots, S_M$ listed in Table 2 and their associated diameters, it is possible (trivially) to reconstruct the original approximating matrix $\mathbf{A}_{ut}$ by identifying for each object pair the smallest diameter for a subset that contains that pair. (Explicitly, the smallest diameter for a subset that contains an object pair is equal to the value in $\mathbf{A}_{ut}$ associated with that pair, and the subset itself includes that object pair and all objects in between in the ordering that is used to display the AR form for $\mathbf{A}_{ut}$.) This type of reconstruction is generally possible for any matrix that can be row/column reordered to an AR form through the collection of subsets $S_1, \ldots, S_M$ and their diameters identified by increasing a threshold variable from the smallest fitted value. In fact, even if all the italicized subsets were removed (that are proper subsets of another having the same diameter), exactly the same reconstruction could be carried out because the italicized subsets are redundant with respect to identifying for each object pair the smallest diameter for a subset that contains the pair.

Without imposing further restrictions on the approximating matrix other than just being AR, a more convenient representation using a graph and path lengths in such a graph is generally not possible. We will select two small (AR) submatrices from the upper-triangular portion of Table 1 to make this point more convincingly, and in the process indicate by example how a graph representation is to be constructed and why further restrictions on the approximating matrix may be necessary to carry out the task.

First, consider the fitted values for the first four placed digits, 0, 1, 2, and 4, for which the desired type of graphical representation *is* possible without imposing any further constraints. This AR submatrix is given in Figure 1.1(a) along with the six corresponding subsets of contiguous objects and their diameters, and a graphical representation for the structure. The latter consists of four nodes corresponding to the original four objects that we represent by open circles (referred to as "terminal" nodes), plus six nodes represented by solid circles that denote the six subsets in the given listing (referred to as "internal nodes"). Based on this graph and the internal node heights provided by the calibrated scale on the left, a fitted value in the submatrix between any two terminal nodes can be obtained as one-half the length of the minimum path from one of the terminal nodes up to an internal node and back down to the other terminal node. All horizontal line segments are used here for display convenience only and are assumed not to contribute to the length of any path. Thus, if we changed the vertical scaling by a multiplier of 1/2, each of the fitted values in the submatrix would be exactly the length of the minimum path, between two terminal nodes, that proceeded upward from one such node to an internal node and then back down to the other. We might also note that from the topmost internal node, all paths down to the terminal nodes have exactly the same length; i.e., there is an internal node equidistant from all terminal nodes.

Now, consider the fitted values for the four objects placed respectively at the third through sixth positions: 2, 4, 3, and 5. This AR submatrix is given in Figure 1.1(b) along with the corresponding subsets of contiguous objects and their diameters (excluding the redundant subset {4,3} which is a proper subset of {2,4,3} having the same diameter), and the beginnings of a graphical representation for its structure. There is a difficulty encountered,

TABLE 2

The 45 subsets listed according to increasing diameter values that are contiguous in the object ordering used to display the upper-triangular portion of Table 1. The 22 subsets given in italics are redundant in the sense that they are proper subsets of another listed subset with the same diameter.

subset	*diameter*
{2,4}	1.63
{1,2}	2.73
{*6,8*},{*8,9*},{6,8,9}	3.04
{*3,5*},{*5,6*},{3,5,6}	3.18
{*4,3*},{2,4,3}	3.22
{*4,3,5*},{4,3,5,6}	3.30
{0,1}	3.41
{*9,7*},{*8,9,7*},{6,8,9,7}	3.43
{*5,6,8*},{*5,6,8,9*},{5,6,8,9,7}	3.76
{*3,5,6,8*},{3,5,6,8,9}	3.76
{*4,3,5,6,8*},{*2,4,3,5*},{*2,4,3,5,6*},{2,4,3,5,6,8}	3.76
{*1,2,4*},{1,2,4,3}	3.78
{0,1,2}	4.21
{1,2,4,3,5}	4.23
{3,5,6,8,9,7}	4.25
{0,1,2,4},{4,3,5,6,8,9}	4.70
{0,1,2,4,3}	4.83
{*1,2,4,3,5,6*},{*1,2,4,3,5,6,8*}	4.96
{*1,2,4,3,5,6,8,9*},{*2,4,3,5,6,8,9*}	4.96
{*2,4,3,5,6,8,9,7*},{*4,3,5,6,8,9,7*}	4.96
{1,2,4,3,5,6,8,9,7}	4.96
{*0,1,2,4,3,5*},{0,1,2,4,3,5,6}	5.25
{0,1,2,4,3,5,6,8}	5.38
{0,1,2,4,3,5,6,8,9}	5.52
{0,1,2,4,3,5,6,8,9,7}	5.81

TABLE 3

The fourteen (nonredundant) subsets listed according to increasing diameter values are contiguous in the linear object ordering used to display the lower-triangular SAR portion of Table 1.

For the lower-triangular SAR portion of Table 1:

subset	*diameter*	*subset*	*diameter*
{2,4}	1.63	{2,4,3,5,6}	3.53
{1,2}	2.73	{1,2,4,3}	3.78
{6,8,9}	3.04	{2,4,3,5,6,8,9,7}	4.18
{3,5,6}	3.18	{0,1,2}	4.21
{2,4,3}	3.22	{0,1,2,4,3}	4.76
{0,1}	3.41	{0,1,2,4,3,5,6}	5.25
{6,8,9,7}	3.43	{0,1,2,4,3,5,6,8,9,7}	5.57

however, in defining a graph that would be completely consistent with all the fitted values in the 4×4 submatrix; we indicate this anomaly by the dashed vertical and horizontal lines. If an internal node were to be placed at the level of 3.30 to represent the cluster $\{4, 3, 5\}$, by implication the fitted value for the digit pair (2,5) should also be 3.30 (and not its current value of 3.76). Because digit 3 was "joined" to *both* 2 and 4 at the threshold level 3.22, and thus, there are two fitted values tied at 3.22, a consistent graphical representation would be possible only if the fitted values for the pairs (2,5) and (4,5) were equal. This last observation, that when some fitted values are tied in an approximating matrix $\mathbf{A}_{ut}$, others must also be tied to allow for the construction of a consistent graphical representation, is the motivating basis for considering an additional set of SAR constraints.

When a graphical representation that permits their reconstruction through path lengths is desired for the collection of fitted values in an approximating matrix $\mathbf{A}$, the small illustration just provided serves as justification for imposing a stricter collection of constraints on the approximating matrix than just being row/column reorderable to an AR form. In particular, the additional restriction will be imposed that the approximating matrix $\mathbf{A}$ is row/column reorderable to one that is SAR, which will eliminate the type of graphical anomaly present in Figure 1.1(b).

For the SAR approximation given in the lower-triangular portion of Table 1, there are now only fourteen (nonredundant) subsets identifiable by increasing a threshold variable from the smallest fitted value; these are listed in Table 3 along with their diameters. The imposition of the more restrictive SAR constraints allows the graphical representation given in Figure 1.2. Although we might not change our substantive comments about the approximating matrix (i.e., mostly digit magnitude with some structural characteristics for the subsets $\{2, 4\}$ and $\{6, 8, 9\}$), a graphical representation makes these same observations visually clearer.

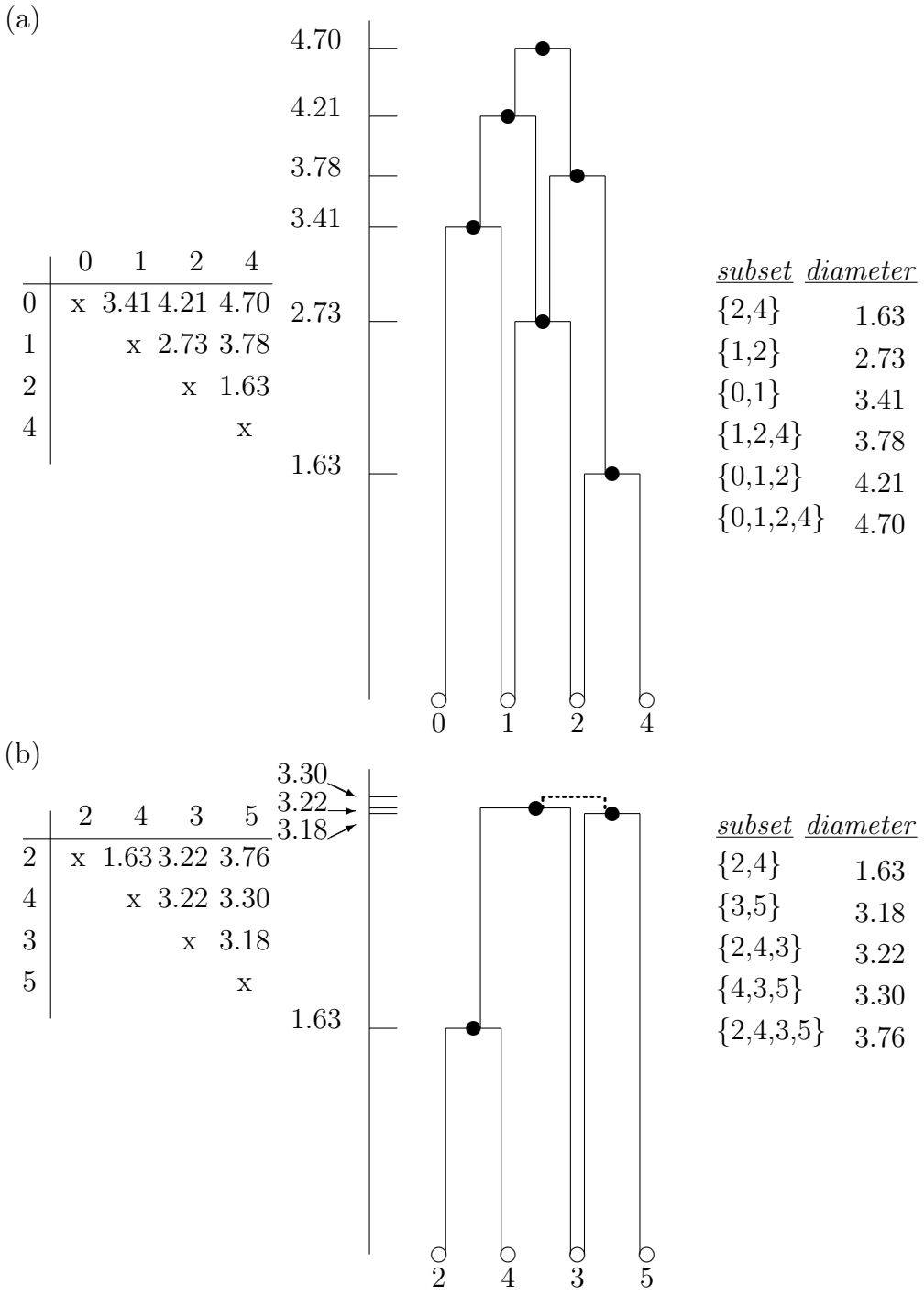

Figure 1.1: Two 4×4 submatrices and the object subsets they induce, taken from the anti-Robinson matrix in the upper-triangular portion of Table 1. For (a), a graphical representation of the fitted values is possible; for (b), the anomaly indicated by the dashed lines prevents a consistent graphical representation from being constructed.

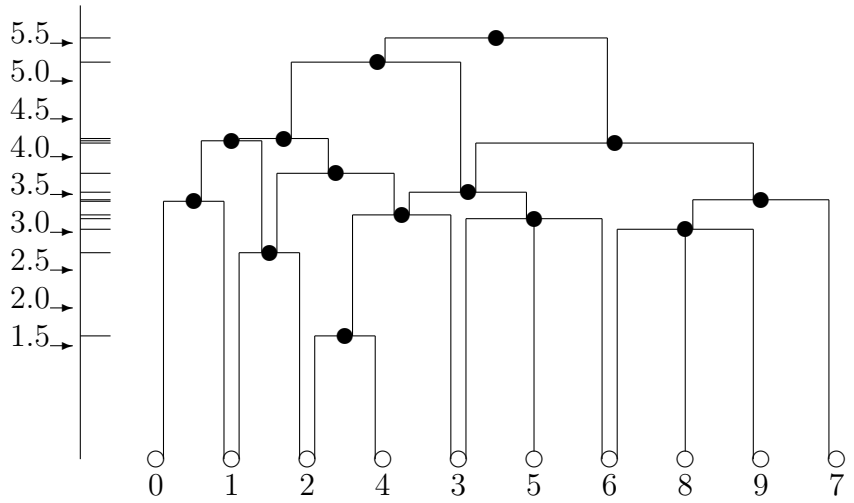

Figure 1.2: A graphical representation for the fitted values given by the strongly-anti-Robinson matrix in the lower-triangular portion of Table 1.

1.6 Representation Through Multiple (Strongly) AR Matrices

The representation of a proximity matrix by a single anti-Robinson structure extends easily to the additive use of multiple matrices. The m-function of Section A.8, `biarobfnd.m`, fits the sum of two anti-Robinson matrices using iterative projection to a symmetric proximity matrix in the L_2-norm based on permutations identified through the use of iterative quadratic assignment. The syntax usage is

```
[find,vaf,targone,targtwo,outpermone,outpermtwo] = ...
biarobfnd(prox,inperm,kblock)
```

where, as before, `PROX` is the input proximity matrix ($n \times n$ with a zero main diagonal and a dissimilarity interpretation); `INPERM` is a given starting permutation of the first n integers; `FIND` is the least-squares optimal matrix (with variance-accounted-for of `VAF`) to `PROX` and is the sum of the two anti-Robinson matrices `TARGONE` and `TARGTWO` based on the two row and column object orderings given by the ending permutations `OUTPERMONE` and `OUTPERMTWO`. As before, `KBLOCK` defines the block size in the use the iterative quadratic assignment routine.

In the example below, the two resulting AR forms are very clearly interpretable as number magnitude and digit structural properties; the variance-accounted-for is, in effect, 100%.

```
load number.dat
[find,vaf,targone,targtwo,outpermone,outpermtwo] = ...
biarobfnd(number,1:10,1)
```

```
find =

  Columns 1 through 7

         0    0.4209    0.5840    0.7090    0.6840    0.8040    0.7865
    0.4209         0    0.2840    0.3460    0.6460    0.5880    0.7568
    0.5840    0.2840         0    0.3540    0.0588    0.6702    0.4225
    0.7090    0.3460    0.3540         0    0.4130    0.4290    0.3000
    0.6840    0.6460    0.0588    0.4130         0    0.4094    0.3880
    0.8040    0.5880    0.6702    0.4290    0.4094         0    0.3960
    0.7865    0.7568    0.4225    0.3000    0.3880    0.3960         0
    0.9107    0.6300    0.7960    0.5920    0.7420    0.4000    0.4169
    0.8210    0.7975    0.3672    0.7975    0.2460    0.6714    0.3499
    0.8500    0.6250    0.8080    0.2630    0.6829    0.5920    0.2960

  Columns 8 through 10

    0.9107    0.8210    0.8500
    0.6300    0.7975    0.6250
    0.7960    0.3672    0.8080
    0.5920    0.7975    0.2630
    0.7420    0.2460    0.6829
    0.4000    0.6714    0.5920
    0.4169    0.3499    0.2960
         0    0.4000    0.4587
    0.4000         0    0.3922
    0.4587    0.3922         0


vaf =

    0.9999


targone =

  Columns 1 through 7

         0    0.3406    0.6710    0.6926    0.6956    0.6956    0.8303
    0.3406         0    0.2018    0.5421    0.5423    0.5880    0.6764
    0.6710    0.2018         0    0.3333    0.3680    0.4662    0.4662
```

```
    0.6926    0.5421    0.3333         0    0.3093    0.3206    0.3779
    0.6956    0.5423    0.3680    0.3093         0    0.2055    0.3779
    0.6956    0.5880    0.4662    0.3206    0.2055         0    0.2876
    0.8303    0.6764    0.4662    0.3779    0.3779    0.2876         0
    0.8303    0.6764    0.6764    0.6764    0.6383    0.4675    0.3360
    0.8303    0.7511    0.7138    0.6764    0.6383    0.4745    0.3366
    0.8611    0.7943    0.7943    0.6764    0.6690    0.4836    0.3849

  Columns 8 through 10

    0.8303    0.8303    0.8611
    0.6764    0.7511    0.7943
    0.6764    0.7138    0.7943
    0.6764    0.6764    0.6764
    0.6383    0.6383    0.6690
    0.4675    0.4745    0.4836
    0.3360    0.3366    0.3849
         0    0.2243    0.3783
    0.2243         0    0.3783
    0.3783    0.3783         0


targtwo =

  Columns 1 through 7

         0   -0.3923   -0.3092   -0.0093    0.0139    0.0139    0.1211
   -0.3923         0   -0.3092   -0.0116    0.0101    0.0139    0.1037
   -0.3092   -0.3092         0   -0.0870   -0.0438    0.0137    0.0207
   -0.0093   -0.0116   -0.0870         0   -0.0438   -0.0111    0.0164
    0.0139    0.0101   -0.0438   -0.0438         0   -0.0889   -0.0779
    0.0139    0.0139    0.0137   -0.0111   -0.0889         0   -0.4134
    0.1211    0.1037    0.0207    0.0164   -0.0779   -0.4134         0
    0.1211    0.1037    0.0822    0.0804    0.0804   -0.1693   -0.1961
    0.1757    0.1037    0.0822    0.0804    0.0804    0.0804   -0.0844
    0.2039    0.2039    0.2039    0.1084    0.1084    0.1084    0.1084

  Columns 8 through 10

    0.1211    0.1757    0.2039
    0.1037    0.1037    0.2039
    0.0822    0.0822    0.2039
```

```
    0.0804    0.0804    0.1084
    0.0804    0.0804    0.1084
   -0.1693    0.0804    0.1084
   -0.1961   -0.0844    0.1084
         0   -0.1211         0
   -0.1211         0   -0.0745
         0   -0.0745         0


outpermone =

     1     2     3     4     5     6     7     9     8    10


outpermtwo =

     9     5     3     1     7    10     4     2     8     6
```

For finding multiple SAR forms, appendix A.9 provides `bisarobfnd.m` with usage syntax

```
[find,vaf,targone,targtwo,outpermone,outpermtwo] = bisarobfnd(prox,inperm,kblock)
```

with all the various terms the same as for `biarobfnd.m` but now for strongly AR (SAR) structures. The example below finds essentially the same representation as above (involving digit magnitude and structure) with a slight drop in the variance-accounted-for of 99.06%.

```
[find,vaf,targone,targtwo,outpermone,outpermtwo] = bisarobfnd(number,randperm(10),1)

find =

  Columns 1 through 7

         0    0.4210    0.5840    0.7095    0.6838    0.8519    0.7260
    0.4210         0    0.2840    0.3460    0.6461    0.5892    0.7565
    0.5840    0.2840         0    0.3541    0.0590    0.6090    0.4830
    0.7095    0.3460    0.3541         0    0.4131    0.4278    0.3005
    0.6838    0.6461    0.0590    0.4131         0    0.4090    0.3882
    0.8519    0.5892    0.6090    0.4278    0.4090         0    0.3960
    0.7260    0.7565    0.4830    0.3005    0.3882    0.3960         0
    0.8998    0.6153    0.8059    0.6067    0.7286    0.4000    0.4168
    0.8208    0.8246    0.3670    0.7893    0.2460    0.6711    0.3502
    0.8736    0.6250    0.7797    0.2630    0.6965    0.5920    0.2955
```

```
  Columns 8 through 10

    0.8998    0.8208    0.8736
    0.6153    0.8246    0.6250
    0.8059    0.3670    0.7797
    0.6067    0.7893    0.2630
    0.7286    0.2460    0.6965
    0.4000    0.6711    0.5920
    0.4168    0.3502    0.2955
         0    0.4000    0.4590
    0.4000         0    0.3921
    0.4590    0.3921         0


vaf =

    0.9906


targone =

  Columns 1 through 7

         0    0.3148    0.6038    0.6296    0.6296    0.7457    0.7457
    0.3148         0    0.1778    0.5201    0.5201    0.6626    0.6626
    0.6038    0.1778         0    0.2742    0.3230    0.5028    0.5028
    0.6296    0.5201    0.2742         0    0.3192    0.5012    0.5012
    0.6296    0.5201    0.3230    0.3192         0    0.2831    0.3340
    0.7457    0.6626    0.5028    0.5012    0.2831         0    0.3021
    0.7457    0.6626    0.5028    0.5012    0.3340    0.3021         0
    0.7936    0.7061    0.6997    0.6974    0.6027    0.5526    0.3229
    0.7936    0.7061    0.6997    0.6974    0.6027    0.5526    0.3229
    0.7936    0.7061    0.6997    0.6974    0.6027    0.5527    0.4963

  Columns 8 through 10

    0.7936    0.7936    0.7936
    0.7061    0.7061    0.7061
    0.6997    0.6997    0.6997
    0.6974    0.6974    0.6974
    0.6027    0.6027    0.6027
    0.5526    0.5526    0.5527
```

```
    0.3229    0.3229    0.4963
         0    0.2815    0.4197
    0.2815         0    0.3001
    0.4197    0.3001         0


targtwo =

  Columns 1 through 7

         0   -0.3567   -0.2640    0.0542    0.0542    0.0938    0.0938
   -0.3567         0   -0.3327    0.0272    0.0272    0.0919    0.0919
   -0.2640   -0.3327         0   -0.0198   -0.0198    0.0799    0.0799
    0.0542    0.0272   -0.0198         0   -0.0198    0.0799    0.0799
    0.0542    0.0272   -0.0198   -0.0198         0   -0.2008   -0.2008
    0.0938    0.0919    0.0799    0.0799   -0.2008         0   -0.4344
    0.0938    0.0919    0.0799    0.0799   -0.2008   -0.4344         0
    0.1260    0.1185    0.1062    0.1062    0.0939   -0.0811   -0.1741
    0.1260    0.1185    0.1062    0.1062    0.0939    0.0393   -0.0907
    0.1260    0.1185    0.1062    0.1062    0.0939    0.0393   -0.0734

  Columns 8 through 10

    0.1260    0.1260    0.1260
    0.1185    0.1185    0.1185
    0.1062    0.1062    0.1062
    0.1062    0.1062    0.1062
    0.0939    0.0939    0.0939
   -0.0811    0.0393    0.0393
   -0.1741   -0.0907   -0.0734
         0   -0.0907   -0.0734
   -0.0907         0   -0.1526
   -0.0734   -0.1526         0


outpermone =

     1     2     3     4     5     6     7     8     9    10


outpermtwo =
```

5 9 3 1 7 10 4 2 8 6

Chapter 2

Circular-Anti-Robinson (CAR) Matrices for Symmetric Proximity Data

In the approximation of a proximity matrix $\mathbf{P}$ by one that is row/column reorderable to an AR form, the interpretation of the fitted matrix in general had to be carried out by identifying a set of subsets through an increasing threshold variable; each of the subsets contained objects that were contiguous with respect to a given *linear* ordering along a continuum, and had a diameter defined by the maximum fitted value within the subset. To provide a further representation depicting the fitted values as lengths of paths in a graph, an approximation was sought that satisfied the additional constraints of an SAR matrix; still, the subsets thus identified had to contain objects contiguous with respect to a linear ordering. As one possible generalization of both the AR and SAR constraints, we can define what will be called circular anti-Robinson (CAR) and circular strongly-anti-Robinson (CSAR) forms that allow the subsets identified from increasing a threshold variable to be contiguous with respect to a *circular* ordering of the objects around a closed continuum. Approximation matrices that are row/column reorderable to display an AR or SAR form respectively will also be (trivially) row/column reorderable to display what is formally characterized below as a CAR or a CSAR form, but not conversely. (Historically, there is a large literature on the possibility of circular structures emerging from and being identifiable in a given proximity matrix, with the CAR concept discussed most extensively under the term "circumplex". One of the earliest references is to Guttman [1954], but for a variety of others the reader is referred to the discussion of metric circular unidimensional scaling in Hubert, Arabie, and Meulman [1997]. The extension of CAR forms to those that are also CSAR, however, has apparently not been a topic discussed in the literature before the appearance of Hubert, Arabie, and Meulman [1998]; this latter source forms the basis for much of the present chapter.)

To be explicit, an arbitrary symmetric matrix $\mathbf{Q} = \{q_{ij}\}$, where $q_{ii} = 0$ for $1 \leq i, j \leq n$, is said to be row/column reorderable to a circular anti-Robinson form (or, for short, $\mathbf{Q}$ is a circular anti-Robinson (CAR) matrix) if there exists a permutation, $\rho(\cdot)$, on the first n

integers such that the reordered matrix $\mathbf{Q}_\rho = \{q_{\rho(i)\rho(j)}\}$ satisfies the conditions given in (II):

(II): for $1 \leq i \leq n-3$, and $i+1 < j \leq n-1$,

if $q_{\rho(i+1)\rho(j)} \leq q_{\rho(i)\rho(j+1)}$, then
$q_{\rho(i+1)\rho(j)} \leq q_{\rho(i)\rho(j)}$ and $q_{\rho(i+1)\rho(j)} \leq q_{\rho(i+1)\rho(j+1)}$;
if $q_{\rho(i+1)\rho(j)} \geq q_{\rho(i)\rho(j+1)}$, then
$q_{\rho(i)\rho(j)} \geq q_{\rho(i)\rho(j+1)}$ and $q_{\rho(i+1)\rho(j+1)} \geq q_{\rho(i)\rho(j+1)}$,
and, for $2 \leq i \leq n-2$,
if $q_{\rho(i+1)\rho(n)} \leq q_{\rho(i)\rho(1)}$, then
$q_{\rho(i+1)\rho(n)} \leq q_{\rho(i)\rho(n)}$ and $q_{\rho(i+1)\rho(n)} \leq q_{\rho(i+1)\rho(1)}$;
if $q_{\rho(i+1)\rho(n)} \geq q_{\rho(i)\rho(1)}$, then
$q_{\rho(i)\rho(n)} \geq q_{\rho(i)\rho(1)}$ and $q_{\rho(i+1)\rho(1)} \geq q_{\rho(i)\rho(1)}$.

Interpretatively, within each row of $\mathbf{Q}_\rho$ moving to the right from the main diagonal and then wrapping back around to re-enter the same row from the left, the entries never decrease until a maximum is reached and then never increase moving away from the maximum until the main diagonal is again reached. Given the symmetry of $\mathbf{P}$, a similar pattern of entries would be present within each column as well. As noted above, any AR matrix is CAR but not conversely.

In analogy to the SAR conditions that permit graphical representation, a symmetric matrix $\mathbf{Q}$ is said to be row/column reorderable to a circular strongly-anti-Robinson form (or, for short, $\mathbf{Q}$ is a circular strongly-anti-Robinson (CSAR) matrix) if there exists a permutation, $\rho(\cdot)$, on the first n integers such that the reordered matrix $\mathbf{Q}_\rho = \{q_{\rho(i)\rho(j)}\}$ satisfies the conditions given by (II), *and*

for $1 \leq i \leq n-3$, and $i+1 < j \leq n-1$,

if $q_{\rho(i+1)\rho(j)} \leq q_{\rho(i)\rho(j+1)}$, then $q_{\rho(i+1)\rho(j)} = q_{\rho(i)\rho(j)}$ implies $q_{\rho(i+1)\rho(j+1)} = q_{\rho(i)\rho(j+1)}$, and $q_{\rho(i+1)\rho(j)} = q_{\rho(i+1)\rho(j+1)}$ implies $q_{\rho(i)\rho(j)} = q_{\rho(i)\rho(j+1)}$;

if $q_{\rho(i+1)\rho(j)} \geq q_{\rho(i)\rho(j+1)}$, then $q_{\rho(i)\rho(j+1)} = q_{\rho(i+1)\rho(j+1)}$ implies $q_{\rho(i)\rho(j)} = q_{\rho(i+1)\rho(j)}$, and $q_{\rho(i)\rho(j)} = q_{\rho(i)\rho(j+1)}$ implies $q_{\rho(i+1)\rho(j)} = q_{\rho(i+1)\rho(j+1)}$,

and for $2 \leq i \leq n-2$,

if $q_{\rho(i+1)\rho(n)} \leq q_{\rho(i)\rho(1)}$, then $q_{\rho(i+1)\rho(n)} = q_{\rho(i)\rho(n)}$ implies $q_{\rho(i+1)\rho(1)} = q_{\rho(i)\rho(1)}$, and $q_{\rho(i+1)\rho(n)} = q_{\rho(i+1)\rho(1)}$ implies $q_{\rho(i)\rho(n)} = q_{\rho(i)\rho(1)}$;

if $q_{\rho(i+1)\rho(n)} \geq q_{\rho(i)\rho(1)}$, then $q_{\rho(i)\rho(1)} = q_{\rho(i+1)\rho(1)}$ implies $q_{\rho(i)\rho(n)} = q_{\rho(i+1)\rho(n)}$, and $q_{\rho(i)\rho(n)} = q_{\rho(i)\rho(1)}$ implies $q_{\rho(i+1)\rho(n)} = q_{\rho(i+1)\rho(1)}$.

Again, the imposition of the stronger CSAR conditions avoids the type of graphical anomaly present in Figure 1.1(b) but now in the context of a CAR matrix — when two fitted values that are adjacent within a row are equal, the fitted values in the same two adjacent columns must also be equal for a row that is either its immediate predecessor (if $q_{\rho(i+1)\rho(j)} \leq q_{\rho(i)\rho(j+1)}$), or successor (if $q_{\rho(i+1)\rho(j)} \geq q_{\rho(i)\rho(j+1)}$); a similar condition is imposed when two fitted values that are adjacent within a column are equal. As noted, any SAR matrix is CSAR but not conversely.

The computational strategy we suggest for identifying a best-fitting CAR or CSAR approximation matrix is based on an initial circular unidimensional scaling obtained through the optimization strategy developed by Hubert, Arabie, and Meulman (1997). Specifically, by a combination of combinatorial search for good matrix reorderings, and heuristic iterative projection to locate the points of inflection when minimum distance calculations change directionality around a closed circular structure, approximation matrices to $\mathbf{P}$ are found through a least-squares loss criterion, and they have the parameterized form

$$\mathbf{Q}_\rho = \{\min(\mid x_{\rho(j)} - x_{\rho(i)} \mid,\ x_0 - \mid x_{\rho(j)} - x_{\rho(i)} \mid)\ + c\},$$

where c is an estimated additive constant, $x_{\rho(1)} \leq x_{\rho(2)} \leq \cdots \leq x_{\rho(n)} \leq x_0$, and the last coordinate, x_0, is the circumference of the circular structure. Based on the inequality constraints implied by such a collection of coordinates, a CAR approximation matrix can be fitted to $\mathbf{P}$ directly; then, beginning with this latter CAR approximation, the identification and imposition of CSAR constraints proceeds through the heuristic use of iterative projection, directly analogous to the way SAR constraints in the linear ordering context were identified and fitted, beginning with a best approximation matrix satisfying just the AR restrictions.

2.1 Fitting a Given CAR Matrix in the L_2-Norm

The MATLAB function m-file given in Section A.10 of Appendix A, `cirarobfit.m`, fits a circular anti-Robinson (CAR) matrix using iterative projection to a symmetric proximity matrix in the L_2-norm. Usage syntax is

```
[fit, vaf] = cirarobfit(prox,inperm,targ)
```

where `PROX` is the input proximity matrix ($n \times n$ with a zero main diagonal and a dissimilarity interpretation); `INPERM` is a given permutation of the first n integers (around a circle); `TARG` is a given $n \times n$ matrix having the circular anti-Robinson form that guides the direction in which distances are taken around the circle. The matrix `FIT` is the least-squares optimal approximation (with variance-accounted-for of `VAF`) to `PROX` having an circular anti-Robinson form for the row and column object ordering given by `INPERM`.

A recording of a MATLAB session follows that uses the `number.dat` data file; an equally-spaced circular anti-Robinson matrix `targcir` obtained from the utility m-file `ransymat.m` first introduced in the LUS Toolbox; and the identity permutation for the objects around the circular structure. The fitted CAR matrix identified in this way has a `vaf` of 64.37%.

```
 load number.dat
 [prox10 targlin targcir] = ransymat(10);
 [fit vaf] = cirarobfit(number,1:10,targcir)

fit =
```

```
  Columns 1 through 7

         0    0.4210    0.5840    0.6510    0.6835    0.8040    0.7730
    0.4210         0    0.2840    0.3460    0.6170    0.6170    0.7730
    0.5840    0.2840         0    0.2753    0.2753    0.5460    0.5460
    0.6510    0.3460    0.2753         0    0.2753    0.3844    0.3844

    0.6835    0.6170    0.2753    0.2753         0    0.3844    0.3844
    0.8040    0.6170    0.5460    0.3844    0.3844         0    0.3844
    0.7730    0.7730    0.5460    0.3844    0.3844    0.3844         0
    0.7695    0.7695    0.7960    0.5920    0.5530    0.4000    0.3857
    0.6597    0.6597    0.6597    0.8040    0.5530    0.5530    0.3857
    0.6510    0.6510    0.6510    0.6510    0.6835    0.5920    0.3857

  Columns 8 through 10

    0.7695    0.6597    0.6510
    0.7695    0.6597    0.6510
    0.7960    0.6597    0.6510
    0.5920    0.8040    0.6510
    0.5530    0.5530    0.6835
    0.4000    0.5530    0.5920
    0.3857    0.3857    0.3857
         0    0.3857    0.3857
    0.3857         0    0.3857
    0.3857    0.3857         0


vaf =

    0.6437
```

2.1.1 The Circular Unidimensional Scaling Utilities for Constructing Circular Anti-Robinson Targets

Sections A.11 and A.12 in the Appendix provide two circular unidimensional scaling utilities, `cirfit.m` and `cirfitac.m`, that will prove useful in finding best-fitting CAR or CSAR approximation matrices by providing the type of anti-Robinson targets needed in the application of `cirarobfit.m` to guide the direction in which distances are to be taken around the circle. The m-file `cirfit.m` does a confirmatory fitting of a given order (assumed to reflect a circular anti-Robinson ordering around a closed unidimensional structure) using Dykstra's

(Kaczmarz's) iterative projection least-squares method. The usage syntax is

```
[fit, diff] = cirfit(prox,inperm)
```

where `INPERM` is the given order; `FIT` is an $n \times n$ matrix that is fitted to `PROX(INPERM,INPERM)` with least-squares value `DIFF`. The syntax for `cirfitac.m` is the same except for the inclusion of an additive constant `ADDCON`:

```
[fit,diff,addcon] = cirfitac(prox,inperm)
```

In brief, then, the type of matrix being fitted to the proximity matrix has the form

$$\mathbf{Q}_\rho = \{\min(\mid x_{\rho(j)} - x_{\rho(i)} \mid,\ x_0 - \mid x_{\rho(j)} - x_{\rho(i)} \mid)\ + c\},$$

where c is an estimated additive constant (assumed equal to zero in `cirfit.m`), $x_{\rho(1)} \leq x_{\rho(2)} \leq \cdots \leq x_{\rho(n)} \leq x_0$, and the last coordinate, x_0, is the circumference of the circular structure. We can obtain these latter coordinates from the adjacent spacings in the output matrix `fit`.

As an example, we applied `cirfit.m` to `number` with an assumed identity input permutation; the spacings around the circular structure between the placements for objects 1 and 2 is .3417; 2 and 3: .1639; 3 and 4: .1216; 4 and 5: .0726; 5 and 6: .1508; 6 and 7: .1146; 7 and 8: .1906; 8 and 9: .0607; 9 and 10: .0852; and back around between 10 and 1: .7830. For `cirfitac.m` the additive constant was estimated as -.3129 with a variance-accounted-for of .5422; here, the spacings around the circular structure between the placements for objects 1 and 2 is .1853; 2 and 3: .1013; 3 and 4: .0590; 4 and 5: .0100; 5 and 6: .0882; 6 and 7: .0520; 7 and 8: .0958; 8 and 9: .0000; 9 and 10: .0217; and back around between 10 and 1: .5327.

```
[fit,diff] = cirfit(number,1:10)

fit =

  Columns 1 through 6

         0    0.3417    0.5056    0.6272    0.6998    0.8506
    0.3417         0    0.1639    0.2855    0.3581    0.5089
    0.5056    0.1639         0    0.1216    0.1942    0.3450
    0.6272    0.2855    0.1216         0    0.0726    0.2234
    0.6998    0.3581    0.1942    0.0726         0    0.1508
    0.8506    0.5089    0.3450    0.2234    0.1508         0
    0.9652    0.6235    0.4596    0.3380    0.2654    0.1146
    0.9289    0.8141    0.6502    0.5286    0.4560    0.3052
    0.8682    0.8748    0.7109    0.5893    0.5167    0.3659
    0.7830    0.9600    0.7961    0.6745    0.6019    0.4511
```

```
  Columns 7 through 10

    0.9652    0.9289    0.8682    0.7830
    0.6235    0.8141    0.8748    0.9600
    0.4596    0.6502    0.7109    0.7961
    0.3380    0.5286    0.5893    0.6745
    0.2654    0.4560    0.5167    0.6019
    0.1146    0.3052    0.3659    0.4511
         0    0.1906    0.2513    0.3365
    0.1906         0    0.0607    0.1459
    0.2513    0.0607         0    0.0852
    0.3365    0.1459    0.0852         0


diff =

    1.8517

[fit,vaf,addcon] = cirfitac(number,1:10)

fit =

  Columns 1 through 6

         0    0.1853    0.2866    0.3457    0.3557    0.4439
    0.1853         0    0.1013    0.1604    0.1704    0.2587
    0.2866    0.1013         0    0.0590    0.0691    0.1573
    0.3457    0.1604    0.0590         0    0.0100    0.0983
    0.3557    0.1704    0.0691    0.0100         0    0.0882
    0.4439    0.2587    0.1573    0.0983    0.0882         0
    0.4960    0.3107    0.2094    0.1503    0.1403    0.0520
    0.5544    0.4065    0.3052    0.2461    0.2361    0.1478
    0.5544    0.4065    0.3052    0.2461    0.2361    0.1478
    0.5327    0.4282    0.3269    0.2678    0.2578    0.1695

  Columns 7 through 10

    0.4960    0.5544    0.5544    0.5327
    0.3107    0.4065    0.4065    0.4282
    0.2094    0.3052    0.3052    0.3269
    0.1503    0.2461    0.2461    0.2678
    0.1403    0.2361    0.2361    0.2578
    0.0520    0.1478    0.1478    0.1695
```

```
         0    0.0958    0.0958    0.1175
    0.0958         0    0.0000    0.0217
    0.0958    0.0000         0    0.0217
    0.1175    0.0217    0.0217         0


vaf =

    0.5422


addcon =

   -0.3129
```

2.2 Finding a CAR Matrix in the L_2-Norm

The m-file in Section A.13, `cirarobfnd.m`, is our suggested strategy for identifying a best-fitting CAR matrix for a symmetric proximity matrix in the L_2-norm based on a permutation that is initially identified through the use of iterative quadratic assignment. Based on an equally-spaced circular target matrix, `order.m` is first invoked to obtain a good (circular) permutation, which is then used to construct a new circular target matrix with `cirfit.m`. The final output is generated from `cirarobfit.m` once it is determined that no better permutation can be identified using the newer circular target matrix. The usage syntax for `cirarobfnd.m` is as follows:

```
[fit, vaf, outperm] = cirarobfnd(prox, inperm, kblock)
```

where `PROX` is the input proximity matrix ($n \times n$ with a zero main diagonal and a dissimilarity interpretation); `INPERM` is a given starting permutation (assumed to be around the circle) of the first n integers; `FIT` is the least-squares optimal matrix (with variance-accounted-for of `VAF`) to `PROX` having a circular anti-Robinson form for the row and column object ordering given by the ending permutation `OUTPERM`. Again, `KBLOCK` defines the block size in the use the iterative quadratic assignment routine.

An example of the use of `cirarobfnd` is given below that seems to lead to a circular ordering best interpreted according to the structural properties of the digits. This is only one of several local optima identifiable by repeated use of the routine from other random starting permutations. In general, the different local optima observed differ in the way the odd digits, $\{3, 5, 7, 9\}$, and the even digits, $\{2, 4, 6, 8\}$, are ordered within these sets when moving clockwise around a circular structure. Explicitly, all local optima had a general structure of $\rightarrow 0 \rightarrow 1 \rightarrow \{3, 5, 7, 9\} \rightarrow \{2, 4, 6, 8\} \rightarrow$, but with some variation in order within the odd and even digits. For example, the CAR matrix given below uses the odd digits as $\rightarrow 3 \rightarrow 5 \rightarrow 9 \rightarrow 7 \rightarrow$ and the even digits as $\rightarrow 6 \rightarrow 8 \rightarrow 4 \rightarrow 2. \rightarrow$.

```
[fit, vaf, outperm] = cirarobfnd(number, randperm(10), 3)

fit =

  Columns 1 through 7

         0    0.3460    0.5315    0.5315    0.6069    0.8040    0.4460
    0.3460         0    0.4210    0.4340    0.6069    0.7895    0.7895
    0.5315    0.4210         0    0.4340    0.6069    0.7895    0.7895
    0.5315    0.4340    0.4340         0    0.0590    0.3670    0.4210
    0.6069    0.6069    0.6069    0.0590         0    0.2460    0.3880
    0.8040    0.7895    0.7895    0.3670    0.2460         0    0.3500
    0.4460    0.7895    0.7895    0.4210    0.3880    0.3500         0
    0.4460    0.6300    0.9090    0.7697    0.6069    0.3960    0.3907
    0.4160    0.6250    0.8500    0.7698    0.6069    0.3960    0.3907
    0.4160    0.5880    0.7698    0.7698    0.6069    0.6069    0.4160

  Columns 8 through 10

    0.4460    0.4160    0.4160
    0.6300    0.6250    0.5880
    0.9090    0.8500    0.7698
    0.7697    0.7698    0.7698
    0.6069    0.6069    0.6069
    0.3960    0.3960    0.6069
    0.3907    0.3907    0.4160
         0    0.3907    0.4160
    0.3907         0    0.4160
    0.4160    0.4160         0


vaf =

    0.8128


outperm =

     4     2     1     3     5     9     7     8    10     6
```

2.3 Fitting and Finding a Strongly Circular-Anti-Robinson (SCAR) Matrix in the L_2-Norm

The two m-functions in Sections A.14 (`cirsarobfit.m`) and A.15 (`cirsarobfnd.m`) are direct analogues of `cirarobfit.m` and `cirarobfnd.m`, respectively, but are concerned with fitting and finding *strongly* circular-anti-Robinson forms. The syntax for `cirsarobfit.m`, which fits a strongly anti-Robinson matrix using iterative projection to a symmetric proximity matrix in the L_2-norm, is

```
[fit, vaf] = cirsarobfit(prox, inperm, targ)
```

where, again, `PROX` is the input proximity matrix ($n \times n$ with a zero main diagonal and a dissimilarity interpretation); `INPERM` is a given permutation of the first n integers; `TARG` is a given $n \times n$ matrix having the circular anti-Robinson form that guides the direction in which distances are taken around the circle. `FIT` is the least-squares optimal matrix (with variance-accounted-for of `VAF`) to `PROX` having a strongly circular-anti-Robinson form for the row and column object ordering given by `INPERM`.

An example follows using the same identity permutation as was done in fitting a CAR form with `cirarobfit.m`; as might be expected from using the more restrictive SCAR form, the variance-accounted-for drops to .4501 from .6437.

```
[fit, vaf] = cirsarobfit(number,1:10,targcir)

fit =

  Columns 1 through 7

         0    0.4210    0.5840    0.6505    0.6505    0.6505    0.6505
    0.4210         0    0.2840    0.6505    0.6505    0.6505    0.6505
    0.5840    0.2840         0    0.2753    0.2753    0.4306    0.4306
    0.6505    0.6505    0.2753         0    0.2753    0.4306    0.4306
    0.6505    0.6505    0.2753    0.2753         0    0.4306    0.4306
    0.6505    0.6505    0.4306    0.4306    0.4306         0    0.4306
    0.6505    0.6505    0.4306    0.4306    0.4306    0.4306         0
    0.6505    0.6505    0.6505    0.6505    0.6505    0.6505    0.3857
    0.6505    0.6505    0.6505    0.6505    0.6505    0.6505    0.3857
    0.6505    0.6505    0.6505    0.6505    0.6505    0.6505    0.3857

  Columns 8 through 10

    0.6505    0.6505    0.6505
    0.6505    0.6505    0.6505
```

```
    0.6505    0.6505    0.6505
    0.6505    0.6505    0.6505
    0.6505    0.6505    0.6505
    0.6505    0.6505    0.6505
    0.3857    0.3857    0.3857
         0    0.3857    0.3857
    0.3857         0    0.3857
    0.3857    0.3857         0


vaf =

    0.4501
```

The m-function `cirsarobfnd.m`, which finds and fits a SCAR matrix using iterative projection to a symmetric proximity matrix in the L_2-norm based on a permutation identified through the use of iterative quadratic assignment, has the expected syntax

```
[fit, vaf, outperm] = cirsarobfnd(prox, inperm, kblock)
```

where, again, `PROX` is the input proximity matrix ($n \times n$ with a zero main diagonal and a dissimilarity interpretation); `INPERM` is a given starting permutation of the first n integers; `FIT` is the least-squares optimal matrix (with variance-accounted-for of `VAF`) to `PROX` having a circular strongly anti-Robinson form for the row and column object ordering given by the ending permutation `OUTPERM`. As usual, `KBLOCK` defines the block size in the use the iterative quadratic assignment routine.

In the MATLAB recording below, and starting from a random permutation, a circular strongly anti-Robinson form was found with a variance-accounted-for of .7296 (again, this represents an expected drop from the value of .8119 for the CAR form — this is also listed below)).

```
[fit, vaf, outperm] = cirsarobfnd(number,randperm(10), 2)

target =

  Columns 1 through 6

         0    0.4160    0.4160    0.4160    0.6262    0.6262
    0.4160         0    0.3907    0.3907    0.3960    0.6263
    0.4160    0.3907         0    0.3907    0.3960    0.6263
    0.4160    0.3907    0.3907         0    0.3500    0.3880
    0.6262    0.3960    0.3960    0.3500         0    0.2460
    0.6262    0.6263    0.6263    0.3880    0.2460         0
```

```
    0.7858    0.7858    0.7858    0.4210    0.3670    0.0590
    0.7858    0.7858    0.9090    0.7895    0.7895    0.5810
    0.5880    0.6250    0.6300    0.7895    0.7895    0.5810
    0.4160    0.4160    0.4460    0.4460    0.8040    0.5810

  Columns 7 through 10

    0.7858    0.7858    0.5880    0.4160
    0.7858    0.7858    0.6250    0.4160
    0.7858    0.9090    0.6300    0.4460
    0.4210    0.7895    0.7895    0.4460
    0.3670    0.7895    0.7895    0.8040
    0.0590    0.5810    0.5810    0.5810
         0    0.4340    0.4340    0.5315
    0.4340         0    0.4210    0.5315
    0.4340    0.4210         0    0.3460
    0.5315    0.5315    0.3460         0


vaf =

    0.8119


outperm =

     6    10     8     7     9     5     3     1     2     4


fit =

  Columns 1 through 6

         0    0.4246    0.4246    0.4246    0.7304    0.7304
    0.4246         0    0.3907    0.3907    0.3960    0.7304
    0.4246    0.3907         0    0.3907    0.3960    0.7304
    0.4246    0.3907    0.3907         0    0.3500    0.3880
    0.7304    0.3960    0.3960    0.3500         0    0.2460
    0.7304    0.7304    0.7304    0.3880    0.2460         0
    0.7304    0.7304    0.7304    0.4210    0.3670    0.0590
    0.7304    0.7304    0.7304    0.7304    0.7304    0.5810
    0.7304    0.7304    0.7304    0.7304    0.7304    0.5810
```

```
    0.4246    0.4246    0.4246    0.4246    0.7304    0.5810

  Columns 7 through 10

    0.7304    0.7304    0.7304    0.4246
    0.7304    0.7304    0.7304    0.4246
    0.7304    0.7304    0.7304    0.4246
    0.4210    0.7304    0.7304    0.4246
    0.3670    0.7304    0.7304    0.7304
    0.0590    0.5810    0.5810    0.5810
         0    0.4340    0.4340    0.5315
    0.4340         0    0.4210    0.5315
    0.4340    0.4210         0    0.3460
    0.5315    0.5315    0.3460         0


vaf =

    0.7296


outperm =

     6    10     8     7     9     5     3     1     2     4
```

2.4 Representing SCAR Structures (Graphically)

As in the case of an AR or SAR matrix, the interpretation of the structure that may be represented by a CAR or CSAR matrix could proceed by first identifying those subsets and their diameters that emerge by increasing a threshold variable from the smallest fitted value. And in the case of a more restrictive CSAR matrix, this collection of subsets and their diameters can then be displayed by a graph where minimum length paths reconstruct the fitted values. To illustrate this graphical possibility on the transformed `number.dat` to mean 4.0 and variance 1.0 given in Hubert, Arabie, and Meulman (1978) — and used earlier to show the graphical representation of an SAR matrix — the fifteen (nonredundant) subsets identified from the CSAR matrix present in Table 5 are listed in Table 4 according to increasing diameter. Here, the structural properties of the digits are apparent (e.g., various subsets of the odd or even digits, or those that are multiples or powers of 2 or of 3), but some magnitude adjacencies can also be noted (e.g., $\{6, 7, 8, 9\}$, or subsets of $\{0, 1, 2, 3\}$). The graph adhering to the CSAR restrictions is given in Figure 2.1 and again minimum path lengths (that proceed up from a terminal node to an internal node and then back down to the other terminal node) can be used to reconstruct the fitted values in $\mathbf{Q}$.

TABLE 4

The fifteen (nonredundant) subsets listed according to increasing diameter values are contiguous in the circular object ordering used to display the CSAR enties in Table 5.

subset	*diameter*	*subset*	*diameter*
{4,2}	1.63	{6,8,4,2}	3.41
{8,4}	2.55	{0,1}	3.41
{1,3}	3.04	{3,5,9,7,6}	3.43
{6,8}	3.06	{2,0,1}	3.47
{8,4,2}	3.14	{2,0,1,3}	3.95
{6,8,4}	3.25	{4,2,0,1,3}	4.20
{9,7,6}	3.26	{0,1,3,5,9,7,6,8,4,2}	4.93
{9,7,6,8}	3.29		

In addition to searching for a best-fitting CSAR matrix directly, we might comment that the type of indirect approach mentioned in the introduction for the case of SAR approximations could also be considered, although we will not go into any of the details here. For example, based on a best-fitting CAR matrix, the additional constraints of a circular unidimensional scale could be identified and then imposed (in fact, this is our starting place in first obtaining the CAR approximation); or those of an ultrametric (which would lead to an SAR matrix that is trivially CSAR as well); or possibly, a collection of additive tree restrictions could be identified. In all cases, CSAR approximations would be automatically obtained.

2.5 Representation Through Multiple (Strongly) CAR Matrices

Just as we discussed in Section 1.6 on representing proximity matrices through multiple (strongly) AR matrices, representations of a proximity matrix by a single (strongly) circular-anti-Robinson structure extends easily to the additive use of multiple matrices. The m-function of Section A.16, `bicirarobfnd.m`, fits the sum of two circular-anti-Robinson matrices using iterative projection to a symmetric proximity matrix in the L_2-norm based on permutations identified through the use of iterative quadratic assignment. The syntax usage is

```
[find,vaf,targone,targtwo,outpermone,outpermtwo] = ...
bicirarobfnd(prox,inperm,kblock)
```

where, as before, `PROX` is the input proximity matrix ($n \times n$ with a zero main diagonal and a dissimilarity interpretation); `INPERM` is a given starting permutation of the first n integers; `FIND` is the least-squares optimal matrix (with variance-accounted-for of `VAF`) to

TABLE 5

A circular stongly-anti-Robinson order-constrained least-squares approximations to the digit proximity data of Shepard *et al.* (1975).

digit	0	1	3	5	9	7	6	8	4	2
0	x	3.41	3.95	4.93	4.93	4.93	4.93	4.93	4.20	3.47
1	3.41	x	3.04	4.93	4.93	4.93	4.93	4.93	4.20	3.47
3	3.95	3.04	x	3.43	3.43	3.43	3.43	4.93	4.20	3.95
5	4.93	4.93	3.43	x	3.43	3.43	3.43	4.93	4.93	4.93
9	4.93	4.93	3.43	3.43	x	3.26	3.26	3.29	4.93	4.93
7	4.93	4.93	3.43	3.43	3.26	x	3.26	3.29	4.93	4.93
6	4.93	4.93	3.43	3.43	3.26	3.26	x	3.06	3.25	3.41
8	4.93	4.93	4.93	4.93	3.29	3.29	3.06	x	2.55	3.14
4	4.20	4.20	4.20	4.93	4.93	4.93	3.25	2.55	x	1.63
2	3.47	3.47	3.95	4.93	4.93	4.93	3.41	3.14	1.63	x

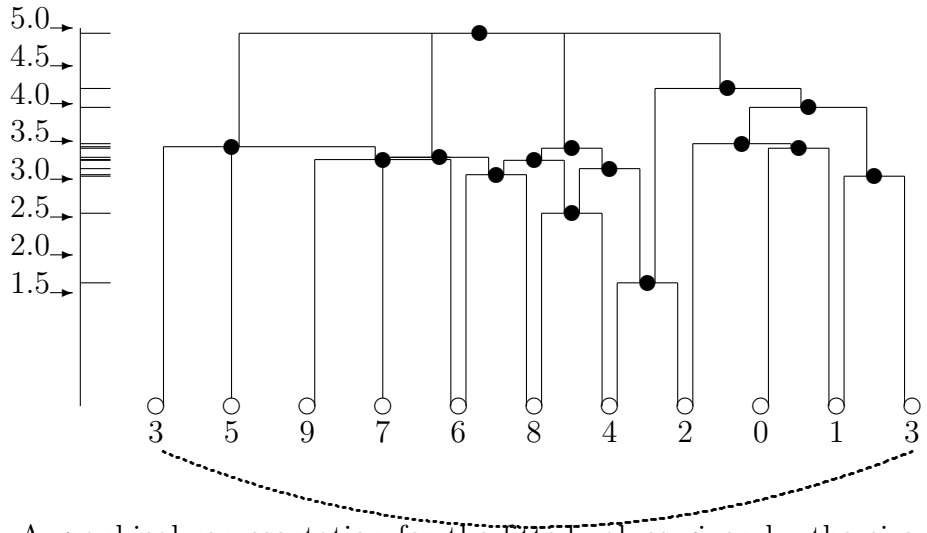

Figure 2.1: A graphical representation for the fitted values given by the circular strongly-anti-Robinson matrix in the lower-triangular portion of Table 5 (VAF = 72.96%). Note that digit 3 is placed both in the first and the last positions in the ordering of the objects with the implication that the sequence continues in a circular manner. This circularity is indicated by the curved dashed line in the figure.

PROX and is the sum of the two circular-anti-Robinson matrices TARGONE and TARGTWO based on the two row and column object orderings given by the ending permutations OUTPERMONE and OUTPERMTWO. As before, KBLOCK defines the block size in the use of iterative quadratic assignment routine.

```
[find,vaf,targone,targtwo,outpermone,outpermtwo] = bicirarobfnd(number,randperm(10),1

find =

  Columns 1 through 7

         0    0.3837    0.5644    0.7286    0.6958    0.8352    0.8003
    0.3837         0    0.3410    0.3461    0.6645    0.5880    0.7576
    0.5644    0.3410         0    0.3342    0.0588    0.6710    0.4211
    0.7286    0.3461    0.3342         0    0.4330    0.4257    0.3000
    0.6958    0.6645    0.0588    0.4330         0    0.4090    0.3881
    0.8352    0.5880    0.6710    0.4257    0.4090         0    0.4257
    0.8003    0.7576    0.4211    0.3000    0.3881    0.4257         0
    0.9091    0.6489    0.8293    0.5920    0.6900    0.4000    0.4537
    0.8003    0.8003    0.3669    0.8040    0.2459    0.6900    0.3501
    0.8501    0.6063    0.7434    0.3114    0.6645    0.5175    0.2476

  Columns 8 through 10

    0.9091    0.8003    0.8501
    0.6489    0.8003    0.6063
    0.8293    0.3669    0.7434
    0.5920    0.8040    0.3114
    0.6900    0.2459    0.6645
    0.4000    0.6900    0.5175
    0.4537    0.3501    0.2476
         0    0.4046    0.4537
    0.4046         0    0.4046
    0.4537    0.4046         0


vaf =

    0.9836


targone =
```

```
  Columns 1 through 7

         0    0.4520    0.4520    0.4520    0.4520    0.6303    0.6303
    0.4520         0    0.4520    0.4520    0.4520    0.6303    0.6303
    0.4520    0.4520         0    0.3882    0.3882    0.3882    0.6303
    0.4520    0.4520    0.3882         0    0.3882    0.3882    0.6303
    0.4520    0.4520    0.3882    0.3882         0    0.3337    0.4143
    0.6303    0.6303    0.3882    0.3882    0.3337         0    0.1862
    0.6303    0.6303    0.6303    0.6303    0.4143    0.1862         0
    0.5314    0.7696    0.7696    0.7696    0.4473    0.3072    0.2560
    0.5314    0.7696    0.8337    0.8927    0.7839    0.7839    0.6303
    0.3723    0.5224    0.6325    0.6325    0.7839    0.7839    0.6303

  Columns 8 through 10

    0.5314    0.5314    0.3723
    0.7696    0.7696    0.5224
    0.7696    0.8337    0.6325
    0.7696    0.8927    0.6325
    0.4473    0.7839    0.7839
    0.3072    0.7839    0.7839
    0.2560    0.6303    0.6303
         0    0.3673    0.3673
    0.3673         0    0.3673
    0.3673    0.3673         0


targtwo =

  Columns 1 through 7

         0   -0.0520    0.0597    0.0656    0.0656    0.0656   -0.0262
   -0.0520         0    0.0164    0.0164    0.0164    0.0656    0.0656
    0.0597    0.0164         0    0.0164    0.0164    0.0164    0.0164
    0.0656    0.0164    0.0164         0    0.0164    0.0164    0.0164
    0.0656    0.0164    0.0164    0.0164         0   -0.0262   -0.0262
    0.0656    0.0656    0.0164    0.0164   -0.0262         0   -0.1406
   -0.0262    0.0656    0.0164    0.0164   -0.0262   -0.1406         0
   -0.0262    0.1400    0.1737    0.1972   -0.0262   -0.1406   -0.1520
   -0.0986    0.0597    0.0597    0.1972   -0.0262   -0.0262   -0.0262
   -0.2213    0.0597    0.0597    0.0656    0.0342    0.0342   -0.0262
```

```
  Columns 8 through 10

   -0.0262   -0.0986   -0.2213
    0.1400    0.0597    0.0597
    0.1737    0.0597    0.0597
    0.1972    0.1972    0.0656
   -0.0262   -0.0262    0.0342
   -0.1406   -0.0262    0.0342
   -0.1520   -0.0262   -0.0262
         0   -0.1972   -0.1972
   -0.1972         0   -0.1972
   -0.1972   -0.1972         0


outpermone =

     4     6    10     8     7     9     5     3     1     2


outpermtwo =

     6     8     9     1     2    10     7     4     3     5
```

For finding multiple SCAR forms, appendix A.17 provides `bicirsarobfnd.m` with usage syntax

```
[find,vaf,targone,targtwo,outpermone,outpermtwo] = bicirsarobfnd(prox,inperm,kblock)
```

with all the various terms the same as for `biarobfnd.m` but now for strongly CAR (SCAR) structures. The example below finds essentially the same representation as above (involving digit magnitude and structure) with a slight drop in the variance-accounted-for of 99.06%.

```
>> [find,vaf,targone,targtwo,outpermone,outpermtwo] = bicirsarobfnd(number,randperm(1

find =

  Columns 1 through 7

         0    0.4210    0.4841    0.6812    0.6812    0.8082    0.8082
    0.4210         0    0.3839    0.3459    0.5831    0.7100    0.6466
    0.4841    0.3839         0    0.3540    0.0589    0.6330    0.4211
    0.6812    0.3459    0.3540         0    0.5061    0.3740    0.3000
    0.6812    0.5831    0.0589    0.5061         0    0.4090    0.3879
```

```
    0.8082    0.7100    0.6330    0.3740    0.4090         0    0.4510
    0.8082    0.6466    0.4211    0.3000    0.3879    0.4510         0
    0.9090    0.8082    0.8082    0.5492    0.7179    0.4589    0.4445
    0.8082    0.7797    0.3672    0.7797    0.2460    0.6710    0.3502
    0.8082    0.6250    0.7080    0.3876    0.7100    0.4510    0.2829

  Columns 8 through 10

    0.9090    0.8082    0.8082
    0.8082    0.7797    0.6250
    0.8082    0.3672    0.7080
    0.5492    0.7797    0.3876
    0.7179    0.2460    0.7100
    0.4589    0.6710    0.4510
    0.4445    0.3502    0.2829
         0    0.3652    0.4445
    0.3652         0    0.4271
    0.4445    0.4271         0


vaf =

    0.9145


targone =

  Columns 1 through 7

         0    0.1456    0.5927    0.5927    0.5927    0.7197    0.7197
    0.1456         0    0.3956    0.3956    0.5927    0.7197    0.7197
    0.5927    0.3956         0    0.3325    0.5927    0.7197    0.7197
    0.5927    0.3956    0.3325         0    0.4190    0.7197    0.7197
    0.5927    0.5927    0.5927    0.4190         0    0.4607    0.4607
    0.7197    0.7197    0.7197    0.7197    0.4607         0    0.4607
    0.7197    0.7197    0.7197    0.7197    0.4607    0.4607         0
    0.7197    0.7197    0.7197    0.7197    0.4607    0.4607    0.3560
    0.3976    0.4328    0.7197    0.7197    0.4607    0.4607    0.3560
    0.2947    0.3072    0.7197    0.7197    0.7197    0.7197    0.3670

  Columns 8 through 10
```

```
    0.7197    0.3976    0.2947
    0.7197    0.4328    0.3072
    0.7197    0.7197    0.7197
    0.7197    0.7197    0.7197
    0.4607    0.4607    0.7197
    0.4607    0.4607    0.7197
    0.3560    0.3560    0.3670
         0    0.3560    0.3670
    0.3560         0    0.2902
    0.3670    0.2902         0


targtwo =

  Columns 1 through 7

         0   -0.0731   -0.0731   -0.0117   -0.0097   -0.0097    0.0600
   -0.0731         0   -0.1607   -0.0117   -0.0097   -0.0097    0.0600
   -0.0731   -0.1607         0   -0.2387   -0.0867   -0.0867    0.0600
   -0.0117   -0.0117   -0.2387         0   -0.0867   -0.0867    0.0600
   -0.0097   -0.0097   -0.0867   -0.0867         0   -0.3107   -0.0487
   -0.0097   -0.0097   -0.0867   -0.0867   -0.3107         0   -0.0487
    0.0600    0.0600    0.0600    0.0600   -0.0487   -0.0487         0
    0.0885    0.0885    0.0885    0.0885   -0.0018   -0.0018   -0.0018
    0.0885    0.0885    0.0885    0.0885    0.0885    0.0885    0.0885
   -0.0947   -0.0731   -0.0731   -0.0117   -0.0097   -0.0097    0.0600

  Columns 8 through 10

    0.0885    0.0885   -0.0947
    0.0885    0.0885   -0.0731
    0.0885    0.0885   -0.0731
    0.0885    0.0885   -0.0117
   -0.0018    0.0885   -0.0097
   -0.0018    0.0885   -0.0097
   -0.0018    0.0885    0.0600
         0    0.1893    0.0885
    0.1893         0    0.0885
    0.0885    0.0885         0


outpermone =
```

```
     5     3     1     2     4     6    10     8     7     9

outpermtwo =

    10     7     4     3     5     6     9     8     1     2
```

Chapter 3

Order Structures for Two-Mode (Rectangular) Proximity Data

References

Carroll, J. D. (1992) Metric, nonmetric, and quasi-nonmetric analysis of psychological data. Division 5 Presidential Address, American Psychological Association, Washington, DC, August, 1992 (published in *Score*, Newsletter of Division 5, October, 1992, pp. 4–5).

Critchley, F. (1994). On exchangeability-based equivalence relations induced by strongly Robinson and, in particular, by quadripolar Robinson dissimilarity matrices. In B. van Cutsem (Ed.), *Classification and dissimilarity analysis*, Lecture Notes in Statistics (pp. 173–199). New York: Springer-Verlag.

Critchley, F., & Fichet, B. (1994). The partial order by inclusion of the principal classes of dissimilarity on a finite set, and some of their basic properties. In B. van Cutsem (Ed.), *Classification and dissimilarity analysis*, Lecture Notes in Statistics (pp. 5–65). New York: Springer-Verlag.

Durand, C., & Fichet, B. (1988). One-to-one correspondences in pyramidal representations: A unified approach. In H. H. Bock (Ed.), *Classification and related methods of data analysis* (pp. 85–90). Amsterdam: North-Holland.

Guttman, L. (1954). A new approach to factor analysis: The radex. In P. F. Lazarsfeld (Ed.), *Mathematical thinking in the social sciences* (pp. 258–348). Glencoe, IL: The Free Press.

Hubert, L. J., & Arabie, P. (1994). The analysis of proximity matrices through sums of matrices having (anti-)Robinson forms. *British Journal of Mathematical and Statistical Psychology*, *47*, 1–40.

Hubert, L. J., & Arabie, P. (1995). Iterative projection strategies for the least-squares fitting of tree structures to proximity data. *British Journal of Mathematical and Statistical Psychology*, *48*, 281–317.

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

Hubert, L. J., Arabie, P., & Meulman, J. (1998) Graph-theoretic representations for proximity matrices through strongly-anti-Robinson or circular strongly-anti-Robinson matrices. *Psychometrika*, *53*, xxx–xxx.

Kruskal, J. B. (1964a) Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, **29**, 1–27.

Kruskal, J. B. (1964b) Nonmetric multidimensional scaling: A numerical method. *Psychometrika*, **29**, 115–129.

Kruskal, J. B., Young, F. W., & Seery, J. B. (1977) *How to Use KYST2, a Very Flexible Program to Do Multidimensional Scaling and Unfolding.* AT&T Bell Laboratories, Murray Hill, NJ.

Pruzansky, S., Tversky, A., & Carroll, J. D. (1982) Spatial versus tree representations of proximity data. *Psychometrika*, **47**, 3–24.

Shepard, R. N. (1962a) Analysis of proximities: Multidimensional scaling with an unknown distance function I. *Psychometrika*, **27**, 125–140.

Shepard, R. N. (1962b) Analysis of proximities: Multidimensional scaling with an unknown distance function II. *Psychometrika*, **27**, 219–246.

Shepard, R. N. (1974) Representation of structure in similarity data: Problems and prospects. *Psychometrika*, **39**, 373–421.

Shepard, R. N., Kilpatric, D. W., & Cunningham, J. P. (1975). The internal representation of numbers. *Cognitive Psychology*, *7*, 82–138.

Wilkinson, L. (1988) *SYSTAT: The System for Statistics*. SYSTAT, Inc, Evanston, IL.
Mirkin, B. (1996). *Mathematical classification and clustering*. Dordrecht: Kluwer.

Appendix A

main program files

A.1 arobfit.m

```
function [fit,vaf] = arobfit(prox,inperm)

% AROBFIT fits an anti-Robinson matrix using iterative projection to
% a symmetric proximity matrix in the $L_{2}$-norm.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given permutation of the first $n$ integers;
% FIT is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX having an anti-Robinson form for the row and column
% object ordering given by INPERM.

n = size(prox,1);
work = zeros(n*(n-1)*(n-2),1);
fit = prox(inperm,inperm);
cr = 1.0;

while (cr >= 1.0e-006)

   cr = 0.0;
   indexll = 0;

   for jone = 1:(n-2)
      for jtwo = (jone+1):(n-1)

         p1 = fit(jone,jtwo);
         p2 = fit(jone,jtwo+1);

         fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);
         fit(jone,jtwo+1) = fit(jone,jtwo+1) - work(indexll+2);

         if(fit(jone,jtwo) <= fit(jone,jtwo+1))

            work(indexll+1) = 0.0;
```

```
         work(indexll+2) = 0.0;

      elseif(fit(jone,jtwo) > fit(jone,jtwo+1))

      ave= (fit(jone,jtwo) + fit(jone,jtwo+1))/2.0;
      work(indexll+1) = ave - fit(jone,jtwo);
      work(indexll+2) = ave - fit(jone,jtwo+1);

      fit(jone,jtwo) = ave;
      fit(jone,jtwo+1) = ave;

      end

      cr = cr + abs(p1-fit(jone,jtwo)) + ...
         abs(p2-fit(jone,jtwo+1));

      indexll = indexll + 2;
   end
end

for jone = 3:n
   for jtwo = 1:(jone-2);

      p1 = fit(jtwo,jone);
      p2 = fit(jtwo+1,jone);

      fit(jtwo,jone) = fit(jtwo,jone) - work(indexll+1);
      fit(jtwo+1,jone) = fit(jtwo+1,jone) - work(indexll+2);

      if(fit(jtwo+1,jone) <= fit(jtwo,jone))

         work(indexll+1) = 0.0;
         work(indexll+2) = 0.0;

      elseif(fit(jtwo+1,jone) > fit(jtwo,jone))

      ave = (fit(jtwo,jone) + fit(jtwo+1,jone))/2.0;
      work(indexll+1) = ave - fit(jtwo,jone);
      work(indexll+2) = ave - fit(jtwo+1,jone);

      fit(jtwo,jone) = ave;
      fit(jtwo+1,jone) = ave;

      end


      cr = cr + abs(p1-fit(jtwo,jone)) + ...
         abs(p2-fit(jtwo+1,jone));

      indexll = indexll + 2;
   end
```

```
   end
end


for jone = 1:(n-1)
   for jtwo = (jone+1):n

         fit(jtwo,jone) = fit(jone,jtwo);

   end
end

  aveprox = sum(sum(prox))/(n*(n-1));

for i = 1:n
   for j = 1:n
      if( i ~= j)
         proxave(i,j) = aveprox;
      else
         proxave(i,j) = 0;
      end
   end
end

diff = sum(sum((prox(inperm,inperm) - fit).^2));

denom = sum(sum((prox(inperm,inperm) - proxave).^2));

vaf = 1 - (diff/denom);
```

A.2 targfit.m

```
function [fit, vaf] = targfit(prox,targ)

% TARGFIT fits through iterative projection a given set of equality and
% inequality constraints (as represented by the equalities and
% inequalities present among the entries in a target matrix
% TARG) to a symmetric proximity matrix in the $L_{2}$-norm.
% PROX is the input proximity matrix (with a zero main diagonal
% and a dissimilarity interpretation);
% TARG is a matrix of the same size as PROX;
% FIT is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX satisfying the equality and
% inequality constraints implicit in TARG.

n = size(prox,1);
work = zeros(n*(n-1)*n*(n-1),1);
fit = prox;
cr = 1.0;
```

```
while (cr >= 1.0e-008)
   cr = 0.0;
   indexll = 0;

   for jone = 1:(n-1)
      for jtwo = (jone+1):n
         for jthree = jone:(n-1)
            for jfour = jtwo:n

               if((jone ~= jthree) | (jtwo ~= jfour))

                  p1 = fit(jone,jtwo);
                  p2 = fit(jthree,jfour);
                  fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);
                  fit(jthree,jfour) = fit(jthree,jfour) - work(indexll+2);

                  if(abs(targ(jone,jtwo) - targ(jthree,jfour)) <= ...
                        1.0e-006)

                     ave = (fit(jone,jtwo) + fit(jthree,jfour))/2.0;
                     work(indexll+1) = ave - fit(jone,jtwo);
                     work(indexll+2) = ave - fit(jthree,jfour);
                     fit(jone,jtwo) = ave;
                     fit(jthree,jfour) = ave;

                     indexll = indexll + 2;

                  elseif((abs(targ(jone,jtwo) - targ(jthree,jfour)) ...
                        > 1.0e-006) & (targ(jone,jtwo) < ...
                        targ(jthree,jfour)))

                     if(fit(jone,jtwo) < fit(jthree,jfour))

                        work(indexll+1) = 0;
                        work(indexll+2) = 0;

                        indexll = indexll + 2;

                     elseif(fit(jone,jtwo) >= fit(jthree,jfour))

                        ave = (fit(jone,jtwo) + fit(jthree,jfour))/2.0;
                        work(indexll+1) = ave - fit(jone,jtwo);
                        work(indexll+2) = ave - fit(jthree,jfour);

                        fit(jone,jtwo) = ave;
                        fit(jthree,jfour) = ave;

                        indexll = indexll + 2;

                     end
```

```
                elseif((abs(targ(jone,jtwo) - targ(jthree,jfour)) ...
                      > 1.0e-006) & (targ(jone,jtwo) > ...
                      targ(jthree,jfour)))

                   if(fit(jone,jtwo) > fit(jthree,jfour))

                      work(indexll+1) = 0;
                      work(indexll+2) = 0;

                      indexll = indexll + 2;

                   elseif(fit(jone,jtwo) <= fit(jthree,jfour))

                      ave = (fit(jone,jtwo) + fit(jthree,jfour))/2.0;
                      work(indexll+1) = ave - fit(jone,jtwo);
                      work(indexll+2) = ave - fit(jthree,jfour);
                      fit(jone,jtwo) = ave;
                      fit(jthree,jfour) = ave;


                      indexll = indexll + 2;

                   end
                end
             cr = cr + abs(p1-fit(jone,jtwo)) + ...
                abs(p2-fit(jthree,jfour));

          end


          end
       end
    end
 end
end




   for jone = 1:(n-1)
      for jtwo = (jone+1):n

         fit(jtwo,jone) = fit(jone,jtwo);

      end
   end


  aveprox = sum(sum(prox))/(n*(n-1));
```

```
for i = 1:n
   for j = 1:n
      if( i ~= j)
         proxave(i,j) = aveprox;
      else
         proxave(i,j) = 0;
      end
   end
end

diff = sum(sum((prox - fit).^2));

denom = sum(sum((prox - proxave).^2));

vaf = 1 - (diff/denom);
```

A.3 order.m

```
function [outperm,rawindex,allperms,index] =  order(prox,targ,inperm,kblock)

% ORDER carries out an iterative Quadratic Assignment maximization task using
% a given square ($n x n$) proximity matrix PROX (with a zero main diagonal and
% a dissimilarity interpretation).
% Three separate local operations are used to permute

% the rows and columns of the proximity matrix to maximize the cross-product
% index with respect to a given square target matrix TARG:
% pairwise interchanges of objects in the permutation defining the row and column
% order of the square proximity matrix; the insertion of from 1 to KBLOCK
% (which is less than or equal to $n-1$) consecutive objects in
% the permutation defining the row and column order of the data matrix; the
% rotation of from 2 to KBLOCK (which is less than or equal to $n-1$) consecutive objects in
% the permutation defining the row and column order of the data matrix.
% INPERM is the input beginning permutation (a permuation of the first $n$ integers).
% OUTPERM is the final permutation of PROX with the cross-product index RAWINDEX
% with respect to TARG. ALLPERMS is a cell array containing INDEX
% entries corresponding to all the
% permutations identified in the optimization from ALLPERMS{1} = INPERM to
% ALLPERMS{INDEX} = OUTPERM.


n = size(prox,1);
outperm = inperm;
index = 1;
allperms{index} = inperm;
begindex = sum(sum(prox(inperm,inperm).*targ));

for iterate = 1:2

   nchange = 1;
```

```
while (nchange == 1)

   nchange=0;

   for k = 1:(n-1)
      for j = (k+1):n

         intrperm = outperm;

         intrperm(k) = outperm(j);
         intrperm(j) = outperm(k);

         tryindex = sum(sum(prox(intrperm,intrperm).*targ));

         if(tryindex > (begindex + 1.0e-008))
            nchange = 1;
            begindex = tryindex;
            outperm = intrperm;
            index = index + 1;
            allperms{index} = intrperm;
         end

      end
   end
end

rawindex = begindex;
nchange = 1;

while (nchange == 1)

   nchange=0;

   for k = 1:kblock
      for insertpt = 1:(n+1)
         for nlimlow = 1:(n+1-k)

            intrperm = outperm;

            if (nlimlow > insertpt)

               jtwo = 0;
               for j = insertpt:(insertpt+k-1)
                  intrperm(j) =outperm(nlimlow+jtwo);
                  jtwo = jtwo + 1;
               end

               jone = 0;
               for j = (insertpt+k):(nlimlow+k-1);
                  intrperm(j) = outperm(insertpt+jone);
```

```
               jone = jone + 1;
            end

         elseif ((nlimlow+k) < insertpt)

            jtwo = 0;
            for j = (insertpt-k):(insertpt-1)
               intrperm(j) = outperm(nlimlow+jtwo);
               jtwo = jtwo + 1;
            end

            jone = 0;
            for j = nlimlow:(insertpt-k-1)
               intrperm(j) = outperm(nlimlow+k+jone);
               jone = jone + 1;
            end

         else

         end

         tryindex = sum(sum(prox(intrperm,intrperm).*targ));

         if(tryindex > (begindex + 1.0e-008))
            nchange = 1;
            begindex = tryindex;
            outperm = intrperm;
            index = index +1;
            allperms{index} = intrperm;
         end

      end
   end
end


rawindex = begindex;
nchange = 1;



while (nchange == 1)

   nchange=0;

   for k = 2:kblock
      for nlimlow = 1:(n+1-k)

         intrperm = outperm;
```

```
         for j = 1:k
            intrperm(nlimlow+j-1) = outperm(nlimlow+k-j);
         end

         tryindex = sum(sum(prox(intrperm,intrperm).*targ));

         if(tryindex > (begindex + 1.0e-008))
            nchange = 1;
            begindex = tryindex;
            outperm = intrperm;
            index = index + 1;
            allperms{index} = intrperm;
         end

      end
   end
end



rawindex = begindex;
nchange = 1;


end
```

A.4 arobfnd.m

```
function [fit, vaf, outperm] = arobfnd(prox, inperm, kblock)

% AROBFND fits an anti-Robinson matrix using iterative projection to
% a symmetric proximity matrix in the $L_{2}$-norm based on a permutation
% identified through the use of iterative quadratic assignment.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given starting permutation of the first $n$ integers;
% FIT is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX having an anti-Robinson form for the row and column
% object ordering given by the ending permutation OUTPERM. KBLOCK
% defines the block size in the use the iterative quadratic assignment
% routine.

n = size(prox,1);
[proxrandom,targlin,targcir] = ransymat(n);

[outperm,rawindex,allperms,index] =  order(prox,targlin,inperm,kblock);

[fit,vaf] = arobfit(prox,outperm);

nprevperm = 1;
```

```
while (nprevperm == 1)

    nprevperm = 0;
    prevperm = outperm;
    inperm = outperm;

    [outperm,rawindex,allperms,index] = order(prox,fit,inperm,kblock);
    [fit,vaf] = arobfit(prox,outperm);

    if (any(prevperm - outperm) == 1)
        nprevperm = 1;
    end
end
```

A.5 sarobfit.m

```
function [fit, vaf] = sarobfit(prox, inperm)

% SAROBFIT fits a strongly anti-Robinson matrix using iterative projection to
% a symmetric proximity matrix in the $L_{2}$-norm.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given permutation of the first $n$ integers;
% FIT is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX having a strongly anti-Robinson form for the row and column
% object ordering given by INPERM.

n = size(prox,1);
work = zeros(n*(n-1)*(n-2),1);
fit = prox(inperm,inperm);
cr = 1.0;

while (cr >= 1.0e-006)

   cr = 0.0;
   indexll = 0;

   for jone = 1:(n-2)
      for jtwo = (jone+1):(n-1)

         p1 = fit(jone,jtwo);
         p2 = fit(jone,jtwo+1);

         fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);
         fit(jone,jtwo+1) = fit(jone,jtwo+1) - work(indexll+2);

         if(fit(jone,jtwo) <= fit(jone,jtwo+1))
```

```
         work(indexll+1) = 0.0;
         work(indexll+2) = 0.0;

      elseif(fit(jone,jtwo) > fit(jone,jtwo+1))

      ave= (fit(jone,jtwo) + fit(jone,jtwo+1))/2.0;
      work(indexll+1) = ave - fit(jone,jtwo);
      work(indexll+2) = ave - fit(jone,jtwo+1);

      fit(jone,jtwo) = ave;
      fit(jone,jtwo+1) = ave;

      end

      cr = cr + abs(p1-fit(jone,jtwo)) + ...
         abs(p2-fit(jone,jtwo+1));

      indexll = indexll + 2;
   end
end

for jone = 3:n
   for jtwo = 1:(jone-2);

      p1 = fit(jtwo,jone);
      p2 = fit(jtwo+1,jone);

      fit(jtwo,jone) = fit(jtwo,jone) - work(indexll+1);
      fit(jtwo+1,jone) = fit(jtwo+1,jone) - work(indexll+2);

      if(fit(jtwo+1,jone) <= fit(jtwo,jone))

         work(indexll+1) = 0.0;
         work(indexll+2) = 0.0;

      elseif(fit(jtwo+1,jone) > fit(jtwo,jone))

      ave = (fit(jtwo,jone) + fit(jtwo+1,jone))/2.0;
      work(indexll+1) = ave - fit(jtwo,jone);
      work(indexll+2) = ave - fit(jtwo+1,jone);

      fit(jtwo,jone) = ave;
      fit(jtwo+1,jone) = ave;

      end


      cr = cr + abs(p1-fit(jtwo,jone)) + ...
         abs(p2-fit(jtwo+1,jone));

      indexll = indexll + 2;
```

```
      end
   end
end


   cr = 1.0;

while (cr >= 1.0e-008)

   cr = 0.0;

   for i = 1:(n-3)
      for j = (i+3):n

         p1 = fit(i,j);
         p2 = fit(i+1,j);

         if(abs(fit(i,j-1) - fit(i+1,j-1)) <= 1.0e-003)

            ave = (fit(i,j) + fit(i+1,j))/2.0;
            fit(i,j) = ave;
            fit(i+1,j) = ave;

         end

         cr = cr + abs(p1-fit(i,j)) + abs(p2-fit(i+1,j));

      end
   end

   for i = 1:(n-3)
      for j = 1:i

         p1 = fit(i+1-j,i+2);
         p2 = fit(i+1-j,i+3);

         if(abs(fit(i+2-j,i+2) - fit(i+2-j,i+3)) <= 1.0e-003)

            ave = (fit(i+1-j,i+2) + fit(i+1-j,i+3))/2.0;
            fit(i+1-j,i+2) = ave;
            fit(i+1-j,i+3) = ave;

         end


         cr = cr + abs(p1-fit(i+1-j,i+2)) + abs(p2-fit(i+1-j,i+3));

      end
   end
end
```

```
for jone = 1:(n-1)
   for jtwo = (jone+1):n

         fit(jtwo,jone) = fit(jone,jtwo);

   end
end

fittarg = fit;
work = zeros(n*(n-1)*(n-2),1);
fit = prox(inperm,inperm);
cr = 1.0;

while (cr >= 1.0e-006)

   cr = 0.0;
   indexll = 0;

   for jone = 1:(n-2)
      for jtwo = (jone+1):(n-1)

         p1 = fit(jone,jtwo);
         p2 = fit(jone,jtwo+1);

         fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);
         fit(jone,jtwo+1) = fit(jone,jtwo+1) - work(indexll+2);

         if((abs(fittarg(jone,jtwo) - fittarg(jone,jtwo+1)) > 1.0e-006) & ...
                 (fit(jone,jtwo) <= fit(jone,jtwo+1)))

            work(indexll+1) = 0.0;
            work(indexll+2) = 0.0;

        else

         ave= (fit(jone,jtwo) + fit(jone,jtwo+1))/2.0;
         work(indexll+1) = ave - fit(jone,jtwo);
         work(indexll+2) = ave - fit(jone,jtwo+1);

         fit(jone,jtwo) = ave;
         fit(jone,jtwo+1) = ave;

         end

         cr = cr + abs(p1-fit(jone,jtwo)) + ...
            abs(p2-fit(jone,jtwo+1));

         indexll = indexll + 2;
      end
   end
```

```
   for jone = 3:n
      for jtwo = 1:(jone-2);

         p1 = fit(jtwo,jone);
         p2 = fit(jtwo+1,jone);

         fit(jtwo,jone) = fit(jtwo,jone) - work(indexll+1);
         fit(jtwo+1,jone) = fit(jtwo+1,jone) - work(indexll+2);

         if((abs(fittarg(jtwo+1,jone) - fittarg(jtwo,jone)) > 1.0e-006 ) & ...
                 (fit(jtwo+1,jone) <= fit(jtwo,jone)))

            work(indexll+1) = 0.0;
            work(indexll+2) = 0.0;


        else

         ave = (fit(jtwo,jone) + fit(jtwo+1,jone))/2.0;
         work(indexll+1) = ave - fit(jtwo,jone);
         work(indexll+2) = ave - fit(jtwo+1,jone);

         fit(jtwo,jone) = ave;
         fit(jtwo+1,jone) = ave;

         end


         cr = cr + abs(p1-fit(jtwo,jone)) + ...
            abs(p2-fit(jtwo+1,jone));

         indexll = indexll + 2;
      end
   end
end


for jone = 1:(n-1)
   for jtwo = (jone+1):n

         fit(jtwo,jone) = fit(jone,jtwo);

   end
end



aveprox = sum(sum(prox))/(n*(n-1));

for i = 1:n
```

```
   for j = 1:n
      if( i ~= j)
         proxave(i,j) = aveprox;
      else
         proxave(i,j) = 0;
      end
   end
end

diff = sum(sum((prox(inperm,inperm) - fit).^2));

denom = sum(sum((prox(inperm,inperm) - proxave).^2));

vaf = 1 - (diff/denom);
```

A.6 sarobfnd.m

```
function [fit, vaf, outperm] = sarobfnd(prox, inperm, kblock)

% SAROBFND fits a strongly anti-Robinson matrix using iterative projection to
% a symmetric proximity matrix in the $L_{2}$-norm based on a permutation
% identified through the use of iterative quadratic assignment.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given starting permutation of the first $n$ integers;
% FIT is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX having a strongly anti-Robinson form for the row and column
% object ordering given by the ending permutation OUTPERM. KBLOCK
% defines the block size in the use the iterative quadratic assignment
% routine.

n = size(prox,1);
[proxrandom,targlin,targcir] = ransymat(n);

[outperm,rawindex,allperms,index] =  order(prox,targlin,inperm,kblock);

[fit,vaf] = sarobfit(prox,outperm);

nprevperm = 1;

while (nprevperm == 1)

    nprevperm = 0;
    prevperm = outperm;
    inperm = outperm;

    [outperm,rawindex,allperms,index] = order(prox,fit,inperm,kblock);
    [fit,vaf] = sarobfit(prox,outperm);

    if (any(prevperm - outperm) == 1)
```

```
        nprevperm = 1;
    end
end
```

A.7 proxmon.m

```
function [monproxpermut, vaf, diff] = proxmon(proxpermut, fitted)

%PROXMON produces a monotonically transformed proximity matrix (MONPROXPERMUT)
%  from the order constraints obtained from each pair of entries in the input
%  proximity matrix PROXPERMUT (symmetric with a zero main diagonal and a dissimilarity
%  interpretation).
%  MONPROXPERMUT is close to the $n \times n$ matrix FITTED in the least-squares sense;
%  The variance accounted for (VAF) is how much variance in MONPROXPERMUT can be accounted for by
%  FITTED; DIFF is the value of the least-squares criterion.

n = size(proxpermut,1);
work = zeros(n*(n-1)*n*(n-1),1);
targ = proxpermut;
fit = fitted;
cr = 1.0;

while (cr >= 1.0e-006)



   cr = 0.0;
   indexll = 0;

   for jone = 1:(n-1)
      for jtwo = (jone+1):n
         for jthree = 1:(n-1)
            for jfour = (jthree+1):n

               if((jone ~= jthree) | (jtwo ~= jfour))

                  p1 = fit(jone,jtwo);
                  p2 = fit(jthree,jfour);
                  fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);
                  fit(jthree,jfour) = fit(jthree,jfour) - work(indexll+2);


                  if((abs(targ(jone,jtwo) - targ(jthree,jfour)) ...
                        > 1.0e-006) & (targ(jone,jtwo) < ...
                        targ(jthree,jfour)))

                     if(fit(jone,jtwo) <= fit(jthree,jfour))

                        work(indexll+1) = 0;
                        work(indexll+2) = 0;
```

```
                elseif(fit(jone,jtwo) > fit(jthree,jfour))

                   ave = (fit(jone,jtwo) + fit(jthree,jfour))/2.0;
                   work(indexll+1) = ave - fit(jone,jtwo);
                   work(indexll+2) = ave - fit(jthree,jfour);

                   fit(jone,jtwo) = ave;
                   fit(jthree,jfour) = ave;



                end

             elseif((abs(targ(jone,jtwo) - targ(jthree,jfour)) ...
                   > 1.0e-006) & (targ(jone,jtwo) > ...
                   targ(jthree,jfour)))

                if(fit(jone,jtwo) >= fit(jthree,jfour))

                   work(indexll+1) = 0;
                   work(indexll+2) = 0;



                elseif(fit(jone,jtwo) < fit(jthree,jfour))

                   ave = (fit(jone,jtwo) + fit(jthree,jfour))/2.0;
                   work(indexll+1) = ave - fit(jone,jtwo);
                   work(indexll+2) = ave - fit(jthree,jfour);
                   fit(jone,jtwo) = ave;
                   fit(jthree,jfour) = ave;



                end
             end

          cr = cr + abs(p1-fit(jone,jtwo)) + ...
             abs(p2-fit(jthree,jfour));
       end

          indexll = indexll + 2;

       end
    end
 end
end
```

```
   for jone = 1:(n-1)
      for jtwo = (jone+1):n

         fit(jtwo,jone) = fit(jone,jtwo);

      end
   end

  avefit = sum(sum(fit))/(n*(n-1));

for i = 1:n
   for j = 1:n
      if( i ~= j)
         proxave(i,j) = avefit;
      else
         proxave(i,j) = 0;
      end
   end
end

diff = sum(sum((fit - fitted).^2));

denom = sum(sum((fit - proxave).^2));

vaf = 1 - (diff/denom);

monproxpermut = fit;

diff = (.5)*diff;
```

A.8 biarobfnd.m

```
function [find,vaf,targone,targtwo,outpermone,outpermtwo] = biarobfnd(prox,inperm,kblock)

% BIAROBFND fits the sum of two anti-Robinson matrices using iterative projection to
% a symmetric proximity matrix in the $L_{2}$-norm based on permutations
% identified through the use of iterative quadratic assignment.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given starting permutation of the first $n$ integers;
% FIND is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX and is the sum of the two anti-Robinson matrices
% TARGONE and TARGTWO based on the two row and column
% object orderings given by the ending permutations OUTPERMONE
% and OUTPERMTWO. KBLOCK defines the block size in the use the
```

```
% iterative quadratic assignment routine.

n = size(prox,1);

[targ1,vaftarg1,outperm1] = arobfnd(prox,inperm,kblock);

resprox1(outperm1,outperm1) = prox(outperm1,outperm1) - targ1;

[targ2,vaftarg2,outperm2] = arobfnd(resprox1,inperm,kblock);

resprox2(outperm2,outperm2) = resprox1(outperm2,outperm2) - targ2;

find = prox - resprox2;

aveprox = sum(sum(prox))/(n*(n-1));

for i = 1:n
    for j = 1:n
        if(i ~= j)
            proxave(i,j) = aveprox;
        else
            proxave(i,j) = 0.0;
        end
    end
end

diff = sum(sum((prox - find).^2));
denom = sum(sum((prox - proxave).^2));
vaf = 1 - (diff/denom);

targone = targ1;
targtwo = targ2;
outpermone = outperm1;
outpermtwo = outperm2;

vafdiff = 1.0;

while (vafdiff >= 1.0e-006)

    vafprev = vaf;


    resprox(outpermtwo,outpermtwo) = prox(outpermtwo,outpermtwo) - targtwo;
    [targone,vafone,outpermone] = arobfnd(resprox,outpermone,kblock);

    resprox(outpermone,outpermone) = prox(outpermone,outpermone) - targone;
    [targtwo,vaftwo,outpermtwo] = arobfnd(resprox,outpermtwo,kblock);

    find(outpermone,outpermone) = targone;
    find(outpermtwo,outpermtwo) = find(outpermtwo,outpermtwo) + targtwo;
```

```
    diff = sum(sum((prox - find).^2));
    denom = sum(sum((prox - proxave).^2));
    vaf = 1 - (diff/denom);

    vafdiff = abs(vaf - vafprev);
end
```

A.9 bisarobfnd.m

```
function [find,vaf,targone,targtwo,outpermone,outpermtwo] = bisarobfnd(prox,inperm,kblock)

% BISAROBFND fits the sum of two stongly anti-Robinson matrices using iterative
% projection to a symmetric proximity matrix in the $L_{2}$-norm based on permutations
% identified through the use of iterative quadratic assignment.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given starting permutation of the first $n$ integers;
% FIND is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX and is the sum of the two strongly anti-Robinson matrices
% TARGONE and TARGTWO based on the two row and column
% object orderings given by the ending permutations OUTPERMONE
% and OUTPERMTWO. KBLOCK defines the block size in the use the
% iterative quadratic assignment routine.

n = size(prox,1);

[targ1,vaftarg1,outperm1] = sarobfnd(prox,inperm,kblock);


resprox1(outperm1,outperm1) = prox(outperm1,outperm1) - targ1;

[targ2,vaftarg2,outperm2] = sarobfnd(resprox1,inperm,kblock);

resprox2(outperm2,outperm2) = resprox1(outperm2,outperm2) - targ2;

find = prox - resprox2;

aveprox = sum(sum(prox))/(n*(n-1));

for i = 1:n
    for j = 1:n
        if(i ~= j)
            proxave(i,j) = aveprox;
        else
            proxave(i,j) = 0.0;
        end
    end
end

diff = sum(sum((prox - find).^2));
```

```
denom = sum(sum((prox - proxave).^2));
vaf = 1 - (diff/denom);

targone = targ1;
targtwo = targ2;
outpermone = outperm1;
outpermtwo = outperm2;

vafdiff = 1.0;
iteration = 0;

while ((vafdiff >= 1.0e-006) & (iteration <= 100))

    vafprev = vaf;
    iteration = iteration + 1;

    resprox(outpermtwo,outpermtwo) = prox(outpermtwo,outpermtwo) - targtwo;
    [targone,vafone,outpermone] = sarobfnd(resprox,outpermone,kblock);

    resprox(outpermone,outpermone) = prox(outpermone,outpermone) - targone;
    [targtwo,vaftwo,outpermtwo] = sarobfnd(resprox,outpermtwo,kblock);

    find(outpermone,outpermone) = targone;
    find(outpermtwo,outpermtwo) = find(outpermtwo,outpermtwo) + targtwo;

    diff = sum(sum((prox - find).^2));
    denom = sum(sum((prox - proxave).^2));
    vaf = 1 - (diff/denom);

    vafdiff = abs(vaf - vafprev);
end
```

A.10 cirarobfit.m

```
function [fit, vaf] = cirarobfit(prox,inperm,targ)

% CIRAROBFIT fits a circular anti-Robinson matrix using iterative projection to
% a symmetric proximity matrix in the $L_{2}$-norm.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given permutation of the first $n$ integers (around a circle);
% TARG is a given $n \times n$ matrix having the circular anti-Robinson
% form that guides the direction in which distances are taken around the circle.
% FIT is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX having an circular anti-Robinson form for the row and column
% object ordering given by INPERM.

n = size(prox,1);
work = zeros(n*(n-1)*(n-2),1);
fit = prox(inperm,inperm);
```

```
cr = 1.0;

while (cr >= 1.0e-006)

   cr = 0.0;
   indexll = 0;

   for jone = 1:(n-2)
      for jtwo = (jone+1):(n-1)

         p1 = fit(jone,jtwo);
         p2 = fit(jone,jtwo+1);

         fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);
         fit(jone,jtwo+1) = fit(jone,jtwo+1) - work(indexll+2);

         if(((abs(targ(jone,jtwo) - targ(jone,jtwo+1)) < 1.0e-006) ...
               | ((targ(jone,jtwo) < targ(jone,jtwo+1)) & ...

               (fit(jone,jtwo) > fit(jone,jtwo+1)))) | ...
               ((targ(jone,jtwo) > targ(jone,jtwo+1)) & ...
               (fit(jone,jtwo) < fit(jone,jtwo+1))))

            ave = (fit(jone,jtwo) + fit(jone,jtwo+1))/2.0;
            work(indexll+1) = ave - fit(jone,jtwo);
            work(indexll+2) = ave - fit(jone,jtwo+1);
            fit(jone,jtwo) = ave;
            fit(jone,jtwo+1) = ave;

            cr = cr + abs(fit(jone,jtwo) - p1) + ...
               abs(fit(jone,jtwo+1) - p2);

         else
            work(indexll+1) = 0.0;
            work(indexll+2) = 0.0;

         end

         indexll = indexll + 2;

      end
   end




   for jone = 3:n
      for jtwo = 1:(jone-2)

         p1 = fit(jtwo,jone);
         p2 = fit(jtwo+1,jone);
```

```
      fit(jtwo,jone) = fit(jtwo,jone) - work(indexll+1);
      fit(jtwo+1,jone) = fit(jtwo+1,jone) - work(indexll+2);

      if(((abs(targ(jtwo+1,jone) - targ(jtwo,jone)) < 1.0e-006) ...
            | ((targ(jtwo+1,jone) < targ(jtwo,jone)) & ...
            (fit(jtwo+1,jone) > fit(jtwo,jone)))) | ...
            ((targ(jtwo+1,jone) > targ(jtwo,jone)) & ...
            (fit(jtwo+1,jone) < fit(jtwo,jone))))

         ave = (fit(jtwo+1,jone) + fit(jtwo,jone))/2.0;
         work(indexll+1) = ave - fit(jtwo,jone);
         work(indexll+2) = ave - fit(jtwo+1,jone);
         fit(jtwo+1,jone) = ave;
         fit(jtwo,jone) = ave;

         cr = cr + abs(fit(jtwo,jone) - p1) + ...
            abs(fit(jtwo+1,jone) - p2);

      else
         work(indexll+1) = 0.0;
         work(indexll+2) = 0.0;

      end

      indexll = indexll + 2;

   end
end




for jone = 2:(n-1)


      p1 = fit(jone,n);
      p2 = fit(1,jone);


      fit(jone,n) = fit(jone,n) - work(indexll+1);
      fit(1,jone) = fit(1,jone) - work(indexll+2);

      if(((abs(targ(jone,n) - targ(1,jone)) < 1.0e-006) ...
            | ((targ(jone,n) < targ(1,jone)) & ...
            (fit(jone,n) > fit(1,jone)))) | ...
            ((targ(jone,n) > targ(1,jone)) & ...
            (fit(jone,n) < fit(1,jone))))

         ave = (fit(jone,n) + fit(1,jone))/2.0;
         work(indexll+1) = ave - fit(jone,n);
```

```
            work(indexll+2) = ave - fit(1,jone);
            fit(jone,n) = ave;
            fit(1,jone) = ave;


            cr = cr + abs(fit(jone,n) - p1) + ...
               abs(fit(1,jone) - p2);

         else
            work(indexll+1) = 0.0;
            work(indexll+2) = 0.0;

         end

         indexll = indexll +2;

      end


   end




   for jone = 1:(n-1)
      for jtwo = (jone+1):n

         fit(jtwo,jone) = fit(jone,jtwo);

      end
   end

  aveprox = sum(sum(prox))/(n*(n-1));

for i = 1:n
   for j = 1:n
      if( i ~= j)
         proxave(i,j) = aveprox;
      else
         proxave(i,j) = 0;
      end
   end
end

diff = sum(sum((prox(inperm,inperm) - fit).^2));

denom = sum(sum((prox(inperm,inperm) - proxave).^2));

vaf = 1 - (diff/denom);
```

A.11 cirfit.m

```
function [fit, diff] = cirfit(prox,inperm)

%CIRFIT does a confirmatory fitting of a given order
%  (assumed to reflect a circular ordering around a closed
%  unidimensional structure) using Dykstra's
%  (Kaczmarz's) iterative projection least-squares method.
%  INPERM is the given order; FIT is an $n \times n$ matrix that
%  is fitted to PROX(INPERM,INPERM) with least-squares value DIFF.


n = size(prox,1);
work = zeros(n*(n-1)*(n-2),1);
fit = prox(inperm,inperm);
cr = 1.0;
iterate = 0;
coor = zeros(n,1);

while (cr >= 1.0e-006)


   cr = 0.0;
   indexll = 0;
   iterate = iterate +1;


   for jone = 1:(n-1)
      for jtwo = (jone+1):n

         p1 = fit(jone,jtwo);

         if(iterate <= 10)

            fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);

         end


         if(fit(jone,jtwo) < 0.0)

            work(indexll+1) = -fit(jone,jtwo);
            fit(jone,jtwo) = 0.0;

         elseif(fit(jone,jtwo) >= 0.0)

            work(indexll+1) = 0;

         end

         indexll = indexll + 1;
```

```
      cr = cr + abs(p1-fit(jone,jtwo));

   end
end


for jone = 1:(n-2)
   for jtwo = (jone+2):n

      if((jone ~= 1) | (jtwo ~= n))

         nhto = jtwo - jone;
         nhaf = n - nhto;

         for i = 1:(n-1)

            coor(i) = fit(i,i+1);

         end

         p1 = fit(jone,jtwo);
         coor(n) = fit(1,n);

         if(iterate <= 10)

            fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+3);

            for i = 1:(n-1)
               if(i < jone)
                  fit(i,i+1) = fit(i,i+1) - work(indexll+2);
               elseif((i >= jone) & (i < jtwo))
                  fit(i,i+1) = fit(i,i+1) - work(indexll+1);
               elseif(i >= jtwo)
                  fit(i,i+1) = fit(i,i+1) - work(indexll+2);
               end
            end

            fit(1,n) = fit(1,n) - work(indexll+2);
         end


            aa = 0.0;
            bb = 0.0;
            aac = 0.0;
            bbc = 0.0;

            for i = 1:(n-1)
               if(i < jone)
                  bb = bb + fit(i,i+1);
               elseif((i >= jone) & (i < jtwo))
                  aa = aa + fit(i,i+1);
```

```
            elseif(i >= jtwo)
               bb = bb + fit(i,i+1);
            end
         end
         bb = bb + fit(1,n);

         if(aa <= bb)
            del = (aa - fit(jone,jtwo))/(nhto+1);
            fit(jone,jtwo) = fit(jone,jtwo) + del;
            for i = 1:nhto
               fit(jone+i-1,jone+i) = ...
                  fit(jone+i-1,jone+i) - del;
            end

            work(indexll+1) = -del;
            work(indexll+2) = 0.0;
            work(indexll+3) = del;

            cr = cr + abs(p1-fit(jone,jtwo));
            indexll = indexll + 3;


         elseif(aa > bb)

            del = (bb - fit(jone,jtwo))/(nhaf+1);
            fit(jone,jtwo) = fit(jone,jtwo) + del;
            if(jone ~= 1)
               for i = 1:(jone-1)
                  fit(i,i+1) = fit(i,i+1) - del;
               end
            end
            if(jtwo ~= n)
               for i = jtwo:(n-1)
                     fit(i,i+1) = fit(i,i+1) - del;
               end
            end

                  fit(1,n) = fit(1,n) - del;

                  work(indexll+1) = 0.0;
                  work(indexll+2) = -del;
                  work(indexll+3) = del;

                  cr = cr + abs(p1 - fit(jone,jtwo));
                  indexll = indexll +3;

            end
         end
      end
   end
end
```

```
   targ = fit;

work = zeros(n*(n-1)*(n-2),1);
fit = prox(inperm,inperm);
cr = 1.0;
coor = zeros(n,1);

while (cr >= 1.0e-006)


   cr = 0.0;
   indexll = 0;


   for jone = 1:(n-1)
      for jtwo = (jone+1):n

         p1 = fit(jone,jtwo);
         fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);

         if(fit(jone,jtwo) < 0.0)

            work(indexll+1) = -fit(jone,jtwo);
            fit(jone,jtwo) = 0.0;

         elseif(fit(jone,jtwo) >= 0.0)

            work(indexll+1) = 0.0;

         end

         indexll = indexll + 1;
         cr = cr + abs(p1-fit(jone,jtwo));

      end
   end


   for jone = 1:(n-2)
      for jtwo = (jone+2):n

         if((jone ~= 1) | (jtwo ~= n))

            nhto = jtwo - jone;
            nhaf = n - nhto;

            for i = 1:(n-1)

               coor(i) = fit(i,i+1);
```

```
end

p1 = fit(jone,jtwo);
coor(n) = fit(1,n);


   fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+3);

   for i = 1:(n-1)
      if(i < jone)
         fit(i,i+1) = fit(i,i+1) - work(indexll+2);
      elseif((i >= jone) & (i < jtwo))
         fit(i,i+1) = fit(i,i+1) - work(indexll+1);
      elseif(i >= jtwo)
         fit(i,i+1) = fit(i,i+1) - work(indexll+2);
      end
   end

      fit(1,n) = fit(1,n) - work(indexll+2);




   aa = 0.0;
   bb = 0.0;
   aac = 0.0;
   bbc = 0.0;

   for i = 1:(n-1)
      if(i < jone)
         bbc = bbc + targ(i,i+1);
      elseif((i >= jone) & (i < jtwo))
         aac = aac + targ(i,i+1);
      elseif(i >= jtwo)
         bbc = bbc + targ(i,i+1);
      end
   end

   bbc = bbc + targ(1,n);


   for i = 1:(n-1)
      if(i < jone)
         bb = bb + fit(i,i+1);
      elseif((i >= jone) & (i < jtwo))
         aa = aa + fit(i,i+1);
      elseif(i >= jtwo)
         bb = bb + fit(i,i+1);
      end
   end
   bb = bb + fit(1,n);
```

```
            if(aac <= bbc)
               del = (aa - fit(jone,jtwo))/(nhto+1);
               fit(jone,jtwo) = fit(jone,jtwo) + del;
               for i = 1:nhto
                  fit(jone+i-1,jone+i) = ...
                     fit(jone+i-1,jone+i) - del;
               end

               work(indexll+1) = -del;
               work(indexll+2) = 0.0;
               work(indexll+3) = del;

               cr = cr + abs(p1-fit(jone,jtwo));
               indexll = indexll + 3;


            elseif(aac > bbc)
               del = (bb - fit(jone,jtwo))/(nhaf+1);
               fit(jone,jtwo) = fit(jone,jtwo) + del;
               if(jone ~= 1)
                  for i = 1:(jone-1)
                     fit(i,i+1) = fit(i,i+1) - del;
                  end
               end
               if(jtwo ~= n)
                  for i = jtwo:(n-1)
                     fit(i,i+1) = fit(i,i+1) - del;
                  end
               end

                     fit(1,n) = fit(1,n) - del;

                     work(indexll+1) = 0.0;
                     work(indexll+2) = -del;
                     work(indexll+3) = del;

                     cr = cr + abs(p1 - fit(jone,jtwo));
                     indexll = indexll +3;

                  end
            end
         end
      end


for jone = 1:(n-1)
   for jtwo = (jone+1):n

         fit(jtwo,jone) = fit(jone,jtwo);
```

```
   end
end

  aveprox = sum(sum(prox))/(n*(n-1));

for i = 1:n
   for j = 1:n
      if( i ~= j)
         proxave(i,j) = aveprox;
      else
         proxave(i,j) = 0;
      end
   end
end

diff = sum(sum((prox(inperm,inperm) - fit).^2));

denom = sum(sum((prox(inperm,inperm)).^2));

vaf = 1 - (diff/denom);

diff = (.5)*diff;
```

A.12 cirfitac.m

```
function [fit, vaf, addcon] = cirfitac(prox,inperm)

%CIRFITAC does a confirmatory fitting (including
%  the estimation of an additive constant) for a given order
%  (assumed to reflect a circular ordering around a closed
%  unidimensional structure) using Dykstra's
%  (Kaczmarz's) iterative projection least-squares method.
%  INPERM is the given order; FIT is an $n \times n$ matrix that
%  is fitted to PROX(INPERM,INPERM) with variance-accounted-for of
%  VAF; ADDCON is the estimated additive constant.

[targ,diff] = cirfit(prox,inperm);

n = size(prox,1);
acondiff = 1.0;
addcon = 0.0;
fit = prox(inperm,inperm);

while (acondiff >= 1.0e-004)

   for i = 1:n
      for j = 1:n
         if(i ~= j)
            fit(i,j) = prox(inperm(i),inperm(j)) + addcon;
```

```
         else
            fit(i,j) = 0.0;
         end
      end
   end

   addconpv = addcon;

   work = zeros(n*(n-1)*(n-2),1);
   cr = 1.0;
   coor = zeros(n,1);


while (cr >= 1.0e-006)


   cr = 0.0;
   indexll = 0;


   for jone = 1:(n-1)
      for jtwo = (jone+1):n

         p1 = fit(jone,jtwo);
         fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);

         if(fit(jone,jtwo) < 0.0)

            work(indexll+1) = -fit(jone,jtwo);
            fit(jone,jtwo) = 0.0;

         elseif(fit(jone,jtwo) >= 0.0)

            work(indexll+1) = 0.0;

         end

         indexll = indexll + 1;
         cr = cr + abs(p1-fit(jone,jtwo));

      end
   end


   for jone = 1:(n-2)
      for jtwo = (jone+2):n

         if((jone ~= 1) | (jtwo ~= n))

            nhto = jtwo - jone;
            nhaf = n - nhto;
```

```
for i = 1:(n-1)

   coor(i) = fit(i,i+1);

end

p1 = fit(jone,jtwo);
coor(n) = fit(1,n);


   fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+3);

   for i = 1:(n-1)
      if(i < jone)
         fit(i,i+1) = fit(i,i+1) - work(indexll+2);
      elseif((i >= jone) & (i < jtwo))
         fit(i,i+1) = fit(i,i+1) - work(indexll+1);
      elseif(i >= jtwo)
         fit(i,i+1) = fit(i,i+1) - work(indexll+2);
      end
   end

      fit(1,n) = fit(1,n) - work(indexll+2);




   aa = 0.0;
   bb = 0.0;
   aac = 0.0;
   bbc = 0.0;

   for i = 1:(n-1)
      if(i < jone)
         bbc = bbc + targ(i,i+1);
      elseif((i >= jone) & (i < jtwo))
         aac = aac + targ(i,i+1);
      elseif(i >= jtwo)
         bbc = bbc + targ(i,i+1);
      end
   end

   bbc = bbc + targ(1,n);


   for i = 1:(n-1)
      if(i < jone)
         bb = bb + fit(i,i+1);
      elseif((i >= jone) & (i < jtwo))
         aa = aa + fit(i,i+1);
```

```
            elseif(i >= jtwo)
               bb = bb + fit(i,i+1);
            end
         end
         bb = bb + fit(1,n);

         if(aac <= bbc)
            del = (aa - fit(jone,jtwo))/(nhto+1);
            fit(jone,jtwo) = fit(jone,jtwo) + del;
            for i = 1:nhto
               fit(jone+i-1,jone+i) = ...
                  fit(jone+i-1,jone+i) - del;
            end

            work(indexll+1) = -del;
            work(indexll+2) = 0.0;
            work(indexll+3) = del;

            cr = cr + abs(p1-fit(jone,jtwo));
            indexll = indexll + 3;


         elseif(aac > bbc)
            del = (bb - fit(jone,jtwo))/(nhaf+1);
            fit(jone,jtwo) = fit(jone,jtwo) + del;
            if(jone ~= 1)
               for i = 1:(jone-1)
                  fit(i,i+1) = fit(i,i+1) - del;
               end
            end
            if(jtwo ~= n)
               for i = jtwo:(n-1)
                  fit(i,i+1) = fit(i,i+1) - del;
               end
            end

                  fit(1,n) = fit(1,n) - del;

                  work(indexll+1) = 0.0;
                  work(indexll+2) = -del;
                  work(indexll+3) = del;

                  cr = cr + abs(p1 - fit(jone,jtwo));
                  indexll = indexll +3;

               end

      end
   end
end
```

```
end

for jone = 1:(n-1)
   for jtwo = (jone+1):n

         fit(jtwo,jone) = fit(jone,jtwo);

   end
end



addcon = -sum(sum(prox(inperm,inperm) - fit))/(n*(n-1));
acondiff = abs(addcon - addconpv);

end

  aveprox = sum(sum(prox))/(n*(n-1));

for i = 1:n
   for j = 1:n
      if( i ~= j)
         proxave(i,j) = aveprox;
         fmadd(i,j) = fit(i,j) - addcon;
      else
         proxave(i,j) = 0;
         fmadd(i,j) = 0.0;
      end
   end
end

diff = sum(sum((prox(inperm,inperm) - (fmadd)).^2));

denom = sum(sum((prox(inperm,inperm) - proxave).^2));

vaf = 1 - (diff/denom);
```

A.13 cirarobfnd.m

```
function [fit, vaf, outperm] = cirarobfnd(prox, inperm, kblock)

% CIRAROBFND fits a circular anti-Robinson matrix using iterative projection to
% a symmetric proximity matrix in the $L_{2}$-norm based on a permutation
% identified through the use of iterative quadratic assignment.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given starting permutation (assumed to be around the
% circle) of the first $n$ integers;
% FIT is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX having a circular anti-Robinson form for the row and column
```

```
% object ordering given by the ending permutation OUTPERM. KBLOCK
% defines the block size in the use the iterative quadratic assignment
% routine.

n = size(prox,1);
[proxrandom,targlin,targcir] = ransymat(n);

[outperm,rawindex,allperms,index] =  order(prox,targcir,inperm,kblock);

[target,diff] = cirfit(prox,outperm);

[fit,vaf] = cirarobfit(prox,outperm,target);

nprevperm = 1;

while (nprevperm == 1)

    nprevperm = 0;
    prevperm = outperm;
    inperm = outperm;

    [outperm,rawindex,allperms,index] = order(prox,fit,inperm,kblock);
    [fit,vaf] = cirarobfit(prox,outperm,target);

    if (any(prevperm - outperm) == 1)
        nprevperm = 1;
    end
end
```

A.14 cirsarobfit.m

```
function [fit, vaf] = cirsarobfit(prox,inperm,target)

% CIRSAROBFIT fits a strongly circular anti-Robinson matrix using iterative projection to
% a symmetric proximity matrix in the $L_{2}$-norm.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given permutation of the first $n$ integers (around a circle);
% TARGET is a given $n \times n$ matrix having the circular anti-Robinson
% form that guides the direction in which distances are taken around the circle.
% FIT is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX having a strongly circular anti-Robinson form for the row and column
% object ordering given by INPERM.

[targ,vaf] = cirarobfit(prox,inperm,target);

n = size(prox,1);
work = zeros(n*(n-1)*(n-2),1);
fit = prox(inperm,inperm);
```

```
cr = 1.0;

while (cr >= 1.0e-006)

   cr = 0.0;
   indexll = 0;

   for jone = 1:(n-2)
      for jtwo = (jone+1):(n-1)

         p1 = fit(jone,jtwo);
         p2 = fit(jone,jtwo+1);

         fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);
         fit(jone,jtwo+1) = fit(jone,jtwo+1) - work(indexll+2);

         if(((abs(targ(jone,jtwo) - targ(jone,jtwo+1)) < 1.0e-006) ...
               | ((targ(jone,jtwo) < targ(jone,jtwo+1)) & ...
               (fit(jone,jtwo) > fit(jone,jtwo+1)))) | ...
               ((targ(jone,jtwo) > targ(jone,jtwo+1)) & ...
               (fit(jone,jtwo) < fit(jone,jtwo+1))))

            ave = (fit(jone,jtwo) + fit(jone,jtwo+1))/2.0;
            work(indexll+1) = ave - fit(jone,jtwo);
            work(indexll+2) = ave - fit(jone,jtwo+1);
            fit(jone,jtwo) = ave;
            fit(jone,jtwo+1) = ave;

            cr = cr + abs(fit(jone,jtwo) - p1) + ...
               abs(fit(jone,jtwo+1) - p2);

         else
            work(indexll+1) = 0.0;
            work(indexll+2) = 0.0;

         end

         indexll = indexll + 2;

      end
   end


   for jone = 3:n
      for jtwo = 1:(jone-2)

         p1 = fit(jtwo,jone);
         p2 = fit(jtwo+1,jone);

         fit(jtwo,jone) = fit(jtwo,jone) - work(indexll+1);
         fit(jtwo+1,jone) = fit(jtwo+1,jone) - work(indexll+2);
```

```
      if(((abs(targ(jtwo+1,jone) - targ(jtwo,jone)) < 1.0e-006) ...
            | ((targ(jtwo+1,jone) < targ(jtwo,jone)) & ...
            (fit(jtwo+1,jone) > fit(jtwo,jone)))) | ...
            ((targ(jtwo+1,jone) > targ(jtwo,jone)) & ...
            (fit(jtwo+1,jone) < fit(jtwo,jone))))

         ave = (fit(jtwo+1,jone) + fit(jtwo,jone))/2.0;
         work(indexll+1) = ave - fit(jtwo,jone);
         work(indexll+2) = ave - fit(jtwo+1,jone);
         fit(jtwo+1,jone) = ave;
         fit(jtwo,jone) = ave;

         cr = cr + abs(fit(jtwo,jone) - p1) + ...
            abs(fit(jtwo+1,jone) - p2);

      else
         work(indexll+1) = 0.0;
         work(indexll+2) = 0.0;

      end

      indexll = indexll + 2;

   end
end


for jone = 2:(n-1)


      p1 = fit(jone,n);
      p2 = fit(1,jone);

      fit(jone,n) = fit(jone,n) - work(indexll+1);
      fit(1,jone) = fit(1,jone) - work(indexll+2);

      if(((abs(targ(jone,n) - targ(1,jone)) < 1.0e-006) ...
            | ((targ(jone,n) < targ(1,jone)) & ...
            (fit(jone,n) > fit(1,jone)))) | ...
            ((targ(jone,n) > targ(1,jone)) & ...
            (fit(jone,n) < fit(1,jone))))

         ave = (fit(jone,n) + fit(1,jone))/2.0;
         work(indexll+1) = ave - fit(jone,n);
         work(indexll+2) = ave - fit(1,jone);
         fit(jone,n) = ave;
         fit(1,jone) = ave;

         cr = cr + abs(fit(jone,n) - p1) + ...
            abs(fit(1,jone) - p2);
```

```
   else
      work(indexll+1) = 0.0;
      work(indexll+2) = 0.0;

   end

   indexll = indexll +2;

end


for jone = 1:(n-3)
   for jtwo = (jone+2):(n-1)

      p1 = fit(jone,jtwo);
      p2 = fit(jone+1,jtwo);
      p3 = fit(jone,jtwo+1);
      p4 = fit(jone+1,jtwo+1);

      if(targ(jone,jtwo+1) <= targ(jone+1,jtwo))

         fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);
         fit(jone+1,jtwo) = fit(jone+1,jtwo) - work(indexll+2);

         if(abs(targ(jone,jtwo+1) - targ(jone+1,jtwo+1)) ...
               <= 1.0e-006)

            ave = (fit(jone,jtwo) + fit(jone+1,jtwo))/2.0;

            work(indexll+1) = ave - fit(jone,jtwo);
            work(indexll+2) = ave - fit(jone+1,jtwo);

            fit(jone,jtwo) = ave;
            fit(jone+1,jtwo) = ave;

            cr = cr + abs(fit(jone,jtwo) - p1) + ...
               abs(fit(jone+1,jtwo) - p2);

         else

            work(indexll+1) = 0.0;
            work(indexll+2) = 0.0;

         end

         indexll = indexll + 2;

      else

         fit(jone,jtwo+1) = fit(jone,jtwo+1) - work(indexll+1);
```

```
   fit(jone+1,jtwo+1) = fit(jone+1,jtwo+1) - ...
      work(indexll+2);

   if(abs(targ(jone,jtwo) - targ(jone+1,jtwo)) ...
         <= 1.0e-006)

      ave = (fit(jone,jtwo+1) + fit(jone+1,jtwo+1))/2.0;

      work(indexll+1) = ave - fit(jone,jtwo+1);
      work(indexll+2) = ave - fit(jone+1,jtwo+1);

      fit(jone,jtwo+1) = ave;
      fit(jone+1,jtwo+1) = ave;

      cr = cr + abs(fit(jone,jtwo+1) - p3) ...
         + abs(fit(jone+1,jtwo+1) - p4);

   else

      work(indexll+1) = 0.0;
      work(indexll+2) = 0.0;

   end

   indexll = indexll + 2;

end


p1 = fit(jone,jtwo);
p2 = fit(jone+1,jtwo);
p3 = fit(jone,jtwo+1);
p4 = fit(jone+1,jtwo+1);

if(targ(jone,jtwo+1) <= targ(jone+1,jtwo))

   fit(jone+1,jtwo) = fit(jone+1,jtwo) - work(indexll+1);
   fit(jone+1,jtwo+1) = fit(jone+1,jtwo+1) - work(indexll+2);

   if(abs(targ(jone,jtwo) - targ(jone,jtwo+1)) ...
         <= 1.0e-006)

      ave = (fit(jone+1,jtwo) + fit(jone+1,jtwo+1))/2.0;

      work(indexll+1) = ave - fit(jone+1,jtwo);
      work(indexll+2) = ave - fit(jone+1,jtwo+1);

      fit(jone+1,jtwo) = ave;
      fit(jone+1,jtwo+1) = ave;

      cr = cr + abs(fit(jone+1,jtwo) - p2) + ...
```

```
            abs(fit(jone+1,jtwo+1) - p4);

         else

            work(indexll+1) = 0.0;
            work(indexll+2) = 0.0;

         end

         indexll = indexll + 2;

      else

         fit(jone,jtwo) = fit(jone,jtwo) - work(indexll+1);
         fit(jone,jtwo+1) = fit(jone,jtwo+1) - ...
            work(indexll+2);

         if(abs(targ(jone+1,jtwo) - targ(jone+1,jtwo+1)) ...
               <= 1.0e-006)

            ave = (fit(jone,jtwo) + fit(jone,jtwo+1))/2.0;

            work(indexll+1) = ave - fit(jone,jtwo);
            work(indexll+2) = ave - fit(jone,jtwo+1);

            fit(jone,jtwo) = ave;
            fit(jone,jtwo+1) = ave;

            cr = cr + abs(fit(jone,jtwo) - p1) ...
               + abs(fit(jone,jtwo+1) - p3);

         else

            work(indexll+1) = 0.0;
            work(indexll+2) = 0.0;

         end

         indexll = indexll + 2;

      end

  end
end


for jone = 2:(n-2)


      p1 = fit(jone,n);
      p2 = fit(jone+1,n);
```

```
p3 = fit(1,jone);
p4 = fit(1,jone+1);

if(targ(1,jone) <= targ(jone+1,n))

   fit(jone,n) = fit(jone,n) - work(indexll+1);
   fit(jone+1,n) = fit(jone+1,n) - work(indexll+2);

   if(abs(targ(1,jone) - targ(1,jone+1)) ...
         <= 1.0e-006)

      ave = (fit(jone,n) + fit(jone+1,n))/2.0;

      work(indexll+1) = ave - fit(jone,n);

      work(indexll+2) = ave - fit(jone+1,n);

      fit(jone,n) = ave;
      fit(jone+1,n) = ave;

      cr = cr + abs(fit(jone,n) - p1) + ...
         abs(fit(jone+1,n) - p2);

   else

      work(indexll+1) = 0.0;
      work(indexll+2) = 0.0;

   end

   indexll = indexll + 2;

else

   fit(1,jone) = fit(1,jone) - work(indexll+1);
   fit(1,jone+1) = fit(1,jone+1) - ...
      work(indexll+2);

   if(abs(targ(jone,n) - targ(jone+1,n)) ...
         <= 1.0e-006)

      ave = (fit(1,jone) + fit(1,jone+1))/2.0;

      work(indexll+1) = ave - fit(1,jone);
      work(indexll+2) = ave - fit(1,jone+1);

      fit(1,jone) = ave;
      fit(1,jone+1) = ave;

      cr = cr + abs(fit(1,jone) - p3) ...
         + abs(fit(1,jone+1) - p4);
```

```
   else

      work(indexll+1) = 0.0;
      work(indexll+2) = 0.0;

   end

   indexll = indexll + 2;

end


p1 = fit(jone,n);
p2 = fit(jone+1,n);
p3 = fit(1,jone);
p4 = fit(1,jone+1);

if(targ(1,jone) <= targ(jone+1,n))

   fit(jone+1,n) = fit(jone+1,n) - work(indexll+1);
   fit(1,jone+1) = fit(1,jone+1) - work(indexll+2);

   if(abs(targ(jone,n) - targ(1,jone)) ...
         <= 1.0e-006)

      ave = (fit(jone+1,n) + fit(1,jone+1))/2.0;

      work(indexll+1) = ave - fit(jone+1,n);
      work(indexll+2) = ave - fit(1,jone+1);

      fit(jone+1,n) = ave;
      fit(1,jone+1) = ave;

      cr = cr + abs(fit(jone+1,n) - p2) + ...
         abs(fit(1,jone+1) - p4);

   else

      work(indexll+1) = 0.0;
      work(indexll+2) = 0.0;

   end

   indexll = indexll + 2;

else

   fit(jone,n) = fit(jone,n) - work(indexll+1);
   fit(1,jone) = fit(1,jone) - ...
      work(indexll+2);
```

```
            if(abs(targ(jone+1,n) - targ(1,jone+1)) ...
                  <= 1.0e-006)

               ave = (fit(jone,n) + fit(1,jone))/2.0;

               work(indexll+1) = ave - fit(jone,n);
               work(indexll+2) = ave - fit(1,jone);

               fit(jone,n) = ave;
               fit(1,jone) = ave;

               cr = cr + abs(fit(jone,n) - p1) ...
                  + abs(fit(1,jone) - p3);

            else

               work(indexll+1) = 0.0;
               work(indexll+2) = 0.0;

            end

            indexll = indexll + 2;

         end
      end


   end



   for jone = 1:(n-1)
      for jtwo = (jone+1):n

         fit(jtwo,jone) = fit(jone,jtwo);

      end
   end

  aveprox = sum(sum(prox))/(n*(n-1));

for i = 1:n
   for j = 1:n
      if( i ~= j)
         proxave(i,j) = aveprox;
      else
         proxave(i,j) = 0;
      end
   end
end
```

```
diff = sum(sum((prox(inperm,inperm) - fit).^2));

denom = sum(sum((prox(inperm,inperm) - proxave).^2));

vaf = 1 - (diff/denom);


[newtarg,vaf] = csrobpolish(prox,inperm,fit);

[renewtarg,vaf] = csrobpolish(prox,inperm,newtarg);

[fit,vaf] = csrobpolish(prox,inperm,renewtarg);
```

A.15 cirsarobfnd.m

```
function [fit, vaf, outperm] = cirsarobfnd(prox, inperm, kblock)

% CIRSAROBFND fits a strongly circular anti-Robinson matrix using iterative projection to
% a symmetric proximity matrix in the $L_{2}$-norm based on a permutation
% identified through the use of iterative quadratic assignment.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given starting permutation (assumed to be around the
% circle) of the first $n$ integers;
% FIT is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX having a strongly circular anti-Robinson form for the row and column
% object ordering given by the ending permutation OUTPERM. KBLOCK
% defines the block size in the use the iterative quadratic assignment
% routine.

[target,vaf,outperm] = cirarobfnd(prox,inperm,kblock)

[fit,vaf] = cirsarobfit(prox,outperm,target);
```

A.16 bicirarobfnd.m

```
function [find,vaf,targone,targtwo,outpermone,outpermtwo] = bicirarobfnd(prox,inperm,kblock)

% BICIRAROBFND finds and fits the sum of two circular anti-Robinson matrices using iterative project
% a symmetric proximity matrix in the $L_{2}$-norm based on permutations
% identified through the use of iterative quadratic assignment.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given starting permutation of the first $n$ integers;
% FIND is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX and is the sum of the two circular anti-Robinson matrices
% TARGONE and TARGTWO based on the two row and column
```

```
% object orderings given by the ending permutations OUTPERMONE
% and OUTPERMTWO. KBLOCK defines the block size in the use the
% iterative quadratic assignment routine.

n = size(prox,1);

[targ1,vaftarg1,outperm1] = cirarobfnd_ac(prox,inperm,kblock);

resprox1(outperm1,outperm1) = prox(outperm1,outperm1) - targ1;

[targ2,vaftarg2,outperm2] = cirarobfnd_ac(resprox1,inperm,kblock);

resprox2(outperm2,outperm2) = resprox1(outperm2,outperm2) - targ2;

find = prox - resprox2;

aveprox = sum(sum(prox))/(n*(n-1));

for i = 1:n
    for j = 1:n
        if(i ~= j)
            proxave(i,j) = aveprox;
        else
            proxave(i,j) = 0.0;
        end
    end
end

diff = sum(sum((prox - find).^2));
denom = sum(sum((prox - proxave).^2));
vaf = 1 - (diff/denom);

targone = targ1;
targtwo = targ2;
outpermone = outperm1;
outpermtwo = outperm2;

vafdiff = 1.0;

while (vafdiff >= 1.0e-006)

    vafprev = vaf;


    resprox(outpermtwo,outpermtwo) = prox(outpermtwo,outpermtwo) - targtwo;
    [targone,vafone] = cirarobfit(resprox,outpermone,targone);

    resprox(outpermone,outpermone) = prox(outpermone,outpermone) - targone;
    [targtwo,vaftwo] = cirarobfit(resprox,outpermtwo,targtwo);

    find(outpermone,outpermone) = targone;
```

```
    find(outpermtwo,outpermtwo) = find(outpermtwo,outpermtwo) + targtwo;

    diff = sum(sum((prox - find).^2));
    denom = sum(sum((prox - proxave).^2));
    vaf = 1 - (diff/denom);

    vafdiff = abs(vaf - vafprev);
end
```

A.17 bicirsarobfnd.m

```
function [find,vaf,targone,targtwo,outpermone,outpermtwo] = bicirsarobfnd(prox,inperm,kblock)

% BICIRSAROBFND fits the sum of two stongly circular-anti-Robinson matrices using iterative
% projection to a symmetric proximity matrix in the $L_{2}$-norm based on permutations
% identified through the use of iterative quadratic assignment.
% PROX is the input proximity matrix ($n \times n$ with a zero main diagonal
% and a dissimilarity interpretation);
% INPERM is a given starting permutation of the first $n$ integers;
% FIND is the least-squares optimal matrix (with variance-accounted-for
% of VAF) to PROX and is the sum of the two strongly circular-anti-Robinson matrices
% TARGONE and TARGTWO based on the two row and column
% object orderings given by the ending permutations OUTPERMONE
% and OUTPERMTWO. KBLOCK defines the block size in the use the
% iterative quadratic assignment routine.

n = size(prox,1);

[targ1,vaftarg1,outperm1] = cirsarobfnd_ac(prox,inperm,kblock);

resprox1(outperm1,outperm1) = prox(outperm1,outperm1) - targ1;

[targ2,vaftarg2,outperm2] = cirsarobfnd_ac(resprox1,inperm,kblock);

resprox2(outperm2,outperm2) = resprox1(outperm2,outperm2) - targ2;

find = prox - resprox2;

aveprox = sum(sum(prox))/(n*(n-1));

for i = 1:n
    for j = 1:n
        if(i ~= j)
            proxave(i,j) = aveprox;
        else
            proxave(i,j) = 0.0;
        end
    end
end
```

```
diff = sum(sum((prox - find).^2));
denom = sum(sum((prox - proxave).^2));
vaf = 1 - (diff/denom);

targone = targ1;
targtwo = targ2;
outpermone = outperm1;
outpermtwo = outperm2;

vafdiff = 1.0;
iteration = 0;

while ((vafdiff >= 1.0e-006) & (iteration <= 100))

    vafprev = vaf;
    iteration = iteration + 1;

    resprox(outpermtwo,outpermtwo) = prox(outpermtwo,outpermtwo) - targtwo;
    [targone,vafone] = cirsarobfit(resprox,outpermone,targone);

    resprox(outpermone,outpermone) = prox(outpermone,outpermone) - targone;
    [targtwo,vaftwo] = cirsarobfit(resprox,outpermtwo,targtwo);

    find(outpermone,outpermone) = targone;
    find(outpermtwo,outpermtwo) = find(outpermtwo,outpermtwo) + targtwo;

    diff = sum(sum((prox - find).^2));
    denom = sum(sum((prox - proxave).^2));
    vaf = 1 - (diff/denom);

    vafdiff = abs(vaf - vafprev);
end
```