Multivariate Experimental Clinical Research, Volume 11, No. 2, 1996, 103-107.

AN ALGORITHM FOR THE ENTRY OF MULTIDIMENSIONAL SCALING DATA

John M. Martz*

ABSTRACT

This paper presents a computer algorithm to aid in the entry of asymmetric three-way data (cf. Young, 1987). It is appropriate to procedures where subjects sort stimuli into piles, write descriptions of each pile, and then employ these descriptions to make similarity judgments between piles.

AN ALGORITHM FOR THE ENTRY OF MULTIDIMENSIONAL SCALING DATA

The translation of raw multidimensional scaling data into usable form (i.e., from paper to disk) can be daunting — a small project where 80 subjects make similarity comparisons among 10 stimuli requires typing at least 8000 numbers (assuming asymmetric data). Such a task involves a large time drain and creates great potential for error. The current paper presents a computer algorithm to aid in the entry of asymmetric three-way data (cf. Young, 1987). It is appropriate to procedures where subjects sort stimuli into piles, write descriptions of each pile, and then employ these descriptions to make similarity judgments between piles. For pedagogical purposes, I will employ imaginary data from one subject who has sorted six stimuli into three piles as the basis of all examples.

The first step of the procedure required the subject to sort the stimuli into piles. In this example, the subject judged stimuli number two and three to be similar to each other on some trait, placing them in pile one. Similarly, she grouped stimuli one, five, and six into a second pile. Finally, she placed stimuli four alone in pile three. From this data, a 1 x n array (tpiles[n]), where n equals the total number of stimuli, is created (Table 1). Each stimulus is represented by a unique cell within the array; the value within a cell is the pile in which the corresponding stimulus appeared (e.g., since stimulus 5 was placed in pile 2, the value in tpiles[5] is 2).

* At the time this article was prepared and accepted for publication, John M. Martz was affiliated with the University of North Carolina at Chapel Hill. He was killed in an automobile accident in the summer of 1995.

TABLE 1
EXAMPLE OF THE 1 x n TPILES ARRAY WHEN n = 6

	Stimulus Number 1	2	3	4	5	6
Pile Number	2	1	1	3	2	2

After sorting the stimuli, the subject wrote a description of each pile and made similarity judgements among the piles. For example, when pile 1 was the target, she ranked pile 3 as being most similar to pile 1; pile 2 was seen as least similar. This data is entered into a (p - 1) x p array (order[p - 1, p]), where p is the number of piles created by the subject (Table 2). Each of the p columns in this array coincides with a target pile; the values within the (p - 1) rows represent ordered similarity rankings between the other piles and the target pile (e.g., since pile 1 was seen as most similar and pile 3 as least similar to target pile 2, the values in order[1,2] and order[2,2] are 1 and 3, respectively). In the current example, it is the asymmetrical similarity ratings among the columns that produces the asymmetry in the final data matrix.

TABLE 2
EXAMPLE OF THE (p - 1) x p ORDER ARRAY WHEN p = 3.

	Target Pile 1	2	3
Most similar	3	1	2
Least similar	2	3	1

Given these two arrays, the subject's complete 6 x 6 matrix of similarity data (Table 3) can be generated using the algorithm presented in the Appendix. Essentially, two nested loops (see lines 5 & 7) compare every cell of tpiles[n] to every other cell within that array (i.e., tpiles[1] vs. tpiles[1], tpiles[1] vs. tpiles[2], . . ., tpiles[6] vs. tpiles[6]), filling the data matrix (data[n,n]) left to right, top to bottom, one cell at a time. During this comparison process, three distinct conditions exist (lines 9, 13, & 21).

The first condition (lines 9-10) occurs when the target and stimulus are identical. Since the target and stimulus are identical (e.g., tpiles[2] with itself), the distance between them is zero; a zero is entered in the data matrix (e.g., data[2,2] = 0). This condition only applies to the diagonal components of the data matrix (Table 3).

TABLE 3
COMPLETE n x n MATRIX OF SIMILARITY DATA
FOR EXAMPLE

	Stimulus Number					
Target Number	1	2	3	4	5	6
1	0	2	2	3	1	1
2	3	0	1	2	3	3
3	3	1	0	2	3	3
4	2	3	3	0	2	2
5	1	2	2	3	0	1
6	1	2	2	3	1	0

The next condition (lines 13-14) occurs when the target and stimulus are not identical, but they were grouped in the same pile. Since the target and stimulus are not identical but grouped together (e.g., tpiles[1] = tpiles[5]), the distance between them is one; a one is entered in the data matrix (e.g., data[1,5] = 1).

The final condition (lines 21-27) occurs when the target and stimulus are not identical and not grouped together (e.g., tpiles[1] ≠ tpiles[2]; tpiles[1] ≠ tpiles[4]). Since the target and stimuli were not grouped together, the distance between them depends on the similarity of the piles in which they were located. As the similarity between the piles decreases, the distance between the stimuli increases. Thus, if the stimulus appears in the pile rated as most similar to the target's pile (e.g., when the stimulus is two, and the target is one as seen by the equality tpiles[2] = order[1,tpiles[1]]), the distance between them is two (one more than if they had appeared in the same pile); a two is entered in the data matrix (e.g., data[1,2] = 2). If the stimulus does not appear in the most similar pile but appears in the next pile (e.g., when the stimulus is four, and the target is one as seen by the equality tpiles[4] = order[2,tpiles[1]]), the distance between them is three; a three is entered in the data matrix (e.g., data[1,4] = 3).

Obviously, the speed of this algorithm provides a clear advantage over conventional data-entry techniques (i.e., entering every datum by hand), especially when applied to larger designs.[1] A second advantage depends on the accuracy of the tpiles[n] and orders [p - 1,p] arrays; given that these arrays have been entered faithfully and without missing data, all distances in data matrix will be correct.

Although the code presented in the Appendix assumes that tpiles[n] and order[p - 1,p] are error-free, its current inability to handle missing data represents a disadvantage that could be solved in several (not mutually exclusive) ways: delete subjects with missing data, enter these subjects by hand, or modify the code. However, such considerations are beyond the scope of this paper. Probably the best method is to design the procedure to reduce the possibility of missing data and handle any problems that do occur on an individual basis.

In conclusion, the simple algorithm presented here provides a useful tool. This tool will increase the speed and accuracy of MDS data entry.

REFERENCES

Lipkus, I., Drigotas, S. M., Martz, J. M., & Panter, A. T. (1993). *Dimensions of vulnerability: A multidimensional scaling analysis*. Manuscript in preparation.

Young, F. W. (1987). *Multidimensional scaling: History, theory, and applications*. Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers.

Author Notes

The Appendix contains only the Turbo Pascal code necessary for the comprehension of the algorithm presented here. Although the complete program is available from the author, the reader is free to adapt this algorithm for his/her purposes.

The author thanks Abigail Panter and Forest Young for their comments. The author also thanks James Kennedy for the crash course in Turbo Pascal.

Address requests for reprints to: Psychology Department, University of North Carolina at Chapel Hill, CB# 3270, Davie Hall, Chapel Hill, NC 27599.

Footnote

1. Indeed, this algorithm was originally written to handle 30 stimuli sorted into a maximum of 12 piles — the asymmetric nature of this data would have required 900 keystrokes per subject (Lipkus, Drigotas, Martz, & Panter, 1993).

ALGORITHM

APPENDIX

This code creates a data matrix (data[target,stimulus]) given the arrays presented in Tables 1 (tpiles[n]) and 2 (order[p - 1,p]). See text for details.

```
PROCEDURE DISTANCE;

BEGIN

FOR target := 1 TO numtargets DO
 BEGIN
  FOR stimulus := 1 TO numstimuli DO
   BEGIN
    IF target = stimulus THEN
     data[target,stimulus] := 0
    ELSE
     BEGIN
      IF tpiles[target] = tpiles[stimulus] THEN
       data[target,stimulus] := 1
      ELSE
       BEGIN
        quit := 'N';
        dist := 1;
        WHILE quit = 'N' DO
         BEGIN
          IF tpiles[stimulus] = order[dist,tpiles[target]] THEN
           BEGIN
            data[target,stimulus] := dist + 1;
            quit := 'Y';
           END {if begin}
          ELSE
           dist := dist + 1
         END {while begin}
       END {else begin}
     END {else begin}
   END {for begin}
 END; {for begin}

END; { PROC }
```