Organizing Multidimensional Scaling I/O and Computation

Jan de Leeuw - University of California Los Angeles

Started October 10 2023, Version of October 10, 2023

Abstract

We develop data structures suitable for multidimensional scaling in the smacof framework. Code in both R and C is included.

Contents

1 Introduction **1**

2 Example **2**

3 Appendix: Code **3**
3.1 smacofSort.R 3
3.2 smacofSort.c 4

4 References **6**

Note: This is a working paper which will be expanded/updated frequently. All suggestions for improvement are welcome. All Rmd, tex, html, pdf, R, and C files are in the public domain. Attribution will be appreciated, but is not required. The files can be found at https://github.com/deleeuw/mdsStruct.

1 Introduction

The loss function in (metric, least squares, Euclidean, symmetric) Multidimensional Scaling (MDS) is

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

This assume symmetry and it uses all elements below the diagonal of both W, Δ, and $D(X)$. For missing data we set $w_{ij}=0$.

2 Example

Here is a small input example.

```
n <- 5
row <- col <- NULL
set.seed(12345)
delta <- sample(1:5, 10, replace = TRUE)
w <- sample(1:5, 10, replace = TRUE)

for (j in 1:(n - 1)) {
  for (i in (j + 1):n) {
    row <- c(row, i)
    col <- c(col, j)
  }
}
data.frame(row = row, col = col, delta = delta, w = w)
```

```
##    row col delta w
## 1    2   1     3 1
## 2    3   1     2 4
## 3    4   1     4 4
## 4    5   1     2 2
## 5    3   2     5 4
## 6    4   2     3 3
## 7    5   2     2 1
## 8    4   3     3 5
## 9    5   3     2 4
## 10   5   4     1 2
```

We use this example as input for the R version of *smacofSort()*, which results in the *smacofStructure*

```
dyn.load("smacofSort.so")
smacofSortR(delta, w, row, col)
```

```
##    row col delta w block dist dhat
## 1    5   4     1 2     1    0    0
## 2    5   3     2 4     2    0    0
## 3    5   1     2 2     2    0    0
## 4    3   1     2 4     2    0    0
## 5    5   2     2 1     2    0    0
## 6    4   2     3 3     3    0    0
## 7    4   3     3 5     3    0    0
## 8    2   1     3 1     3    0    0
## 9    4   1     4 4     4    0    0
## 10   3   2     5 4     5    0    0
```

```
dyn.unload("smacofSort.so")
```

The *dist* element in the data frame is zero, because we have not computed distances yet. Given a configuration X the *row* and *col* elements in the data frame allow from straightforward computation of distances. In the case of nonmetric MDS, or more generally in MDS with transformed dissimilarities, the *dhat* column will be filled as well. The fact that preprocessing with *smacofSort()* gives the ordered dissimilarities, as well as the tie blocks, is especially useful in the ordinal case, both with monotone polynomials and monotone splines. In the metric (ratio) case there is no need for the *dhat* column.

It is obvious how this *smacofStructure* can be adapted if there are multiway data.If we have row-conditional or matrix-conditional data, then we use one of these smacofStructures for each row or each matrix.

3 Appendix: Code

We use *qsort* to sort the rows of the input data frame by increasing delta. The sorting is done in C, the R version is a wrapper around the compiled C code. The C code also contains a main which analyzes the same small example as we have used in the text. Compile with "clang -o runner -O2 smacofSort.c" and then start "runner" in the shell. The C code uses the .C() interface in R, which can be improved using .Call(), but probably in this case with little gain.

3.1 smacofSort.R

```
smacofSortR <- function(delta, w, row, col, eps = 1e-6) {
  n <- length(delta)
  h <- .C(
    "smacofSort",
    delta = as.double(delta),
    weight = as.double(w),
    row = as.integer(row),
    col = as.integer(col),
    index = as.integer(0:(n - 1)),
    n = as.integer(n)
  )
  g <- .C(
    "smacofTieBlocks",
    delta = as.double(h$delta),
    block = as.integer(rep(0, n)),
    eps = as.double(eps),
    n = as.integer(n)
  )
  return(data.frame(row = h$row, col = h$col, delta = h$delta, w = h$w, block = g$block
}
```

3.2 smacofSort.c

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

int smacofComparison(const void *px, const void *py);
void smacofSort(double *delta, double *weight, int *row, int *col, int *index,
                const int *n);
void smacofTieBlocks(const double *x, int *it, double *eps, const int *n);

struct smacofData {
    int index;
    int row;
    int col;
    double delta;
    double weight;
};

int smacofComparison(const void *px, const void *py) {
    double x = ((struct smacofData *)px)->delta;
    double y = ((struct smacofData *)py)->delta;
    return (int)copysign(1.0, x - y);
}

void smacofSort(double *delta, double *weight, int *row, int *col, int *index,
                const int *n) {
    int nn = *n;
    struct smacofData *xi = (struct smacofData *)calloc(
        (size_t)nn, (size_t)sizeof(struct smacofData));
    for (int i = 0; i < nn; i++) {
        xi[i].index = i;
        xi[i].row = row[i];
        xi[i].col = col[i];
        xi[i].delta = delta[i];
        xi[i].weight = weight[i];
    }
    (void)qsort(xi, (size_t)nn, (size_t)sizeof(struct smacofData),
                smacofComparison);
    for (int i = 0; i < nn; i++) {
        index[i] = xi[i].index;
        row[i] = xi[i].row;
        col[i] = xi[i].col;
        delta[i] = xi[i].delta;
```

```
        weight[i] = xi[i].weight;
    }
    free(xi);
    return;
}

void smacofTieBlocks(const double *delta, int *block, double *eps,
                     const int *n) {
    int nn = *n;
    block[0] = 1;
    for (int i = 1; i < nn; i++) {
        if (fabs(delta[i] - delta[i - 1]) < *eps) {
            block[i] = block[i - 1];
        } else {
            block[i] = block[i - 1] + 1;
        }
    }
    return;
}

int main(void) {
    int index[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
    double delta[10] = {3.0, 2.0, 4.0, 2.0, 5.0, 3.0, 2.0, 3.0, 2.0, 1.0};
    double weight[10] = {1.0, 4.0, 4.0, 2.0, 4.0, 3.0, 1.0, 5.0, 4.0, 2.0};
    double dist[10] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
    double dhat[10] = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
    int row[10] = {2, 3, 4, 5, 3, 4, 5, 4, 5, 5};
    int col[10] = {1, 1, 1, 1, 2, 2, 2, 3, 3, 4};
    int block[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
    int n = 10;
    double eps = 1e-6;
    for (int i = 0; i < n; i++) {
        printf(
            "row%2d col%2d delta %4.2f weight %4.2f block%2d dist %4.2f dhat "
            "%4.2f\n",
            row[i], col[i], delta[i], weight[i], block[i], dist[i], dhat[i]);
    }
    printf("\n\n");
    (void)smacofSort(delta, weight, row, col, index, &n);
    (void)smacofTieBlocks(delta, block, &eps, &n);
    for (int i = 0; i < n; i++) {
        printf(
            "row%2d col%2d delta %4.2f weight %4.2f block%2d dist %4.2f dhat "
            "%4.2f\n",
```

```
            row[i], col[i], delta[i], weight[i], block[i], dist[i], dhat[i]);
    }
    return (EXIT_SUCCESS);
}
```

4 References