

Matrix Approximation with Kronecker and Precision Weighting

Jan de Leeuw

Jan Graffelman

April 23, 2026

Majorization (or MM) algorithms for matrix approximation, using minimization of triple-weighted least squares loss functions, are developed. Loss functions combine matrix variate weights (or Kronecker weights) with elementwise (or precision) weights. Implementations in R are provided.

Table of contents

1	Introduction	4
2	Algorithms	5
2.1	Simple Majorization	5
2.2	Nested Majorization	6
2.2.1	First Majorization	6
2.2.2	Second majorization	7
3	Software	8
3.1	Arguments	8
3.2	Values	9
3.3	Projection Routines	9
4	Examples	10
4.1	Artificial	10
4.1.1	Data	10
4.1.2	Triple	11
4.1.3	Nested, $\text{inmax} = 1$	11

4.1.4	Nested, inmax = 5	12
4.1.5	Nested, inmax = 10	12
4.1.6	Nested, inmax = 100	13
4.1.7	Conclusion	13
4.2	Correspondence Analysis	13
4.3	Canonical Analysis	14
4.4	Triple Weighting	14
5	Discussion	15
	References	16

Note: This is a working manuscript 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/triple>

1 Introduction

The problem addressed in this paper is to approximate a given $n \times m$ matrix Y by a $n \times m$ matrix X which must be in a given subset $\mathbb{X}$ of $\mathbb{R}^{n \times m}$. For example, $\mathbb{X}$ can be the $n \times m$ matrices of rank less than or equal to p , or a set of special matrices, such as the Hankel or Cauchy matrices.

We measure loss using a loss function¹ proposed in Graffelman (2026b).

$$\sigma(X) := \text{tr } R(W \circ (Y - X))C(W \circ (Y - X))'. \quad (1)$$

In definition (1) R and C are given positive semi-definite matrices of orders n and m , which define the *Kronecker weights*. The Kronecker weights can be, for example, inverse covariance matrices for row and column entries, as in the matrix variate normal distribution (Gupta and Nagar (2000)). The $n \times m$ matrix W of *precision weights* has positive entries, which can be, for example, inverse standard errors of the corresponding entries of Y .

Loss function (1) has many well-known special cases. If all weights are equal to one then we recover correspondence analysis, redundancy analysis, and various other forms of canonical analysis, as well as forms of principal component analysis with Kronecker weights (De Leeuw (1984), De Leeuw and Kukuyeva (2009), Graffelman (2026b), Graffelman (2026a)). If both R and C are equal to the identity we have principal components analysis with precision weights, as in Gabriel and Zamir (1979). If Y is symmetric the various approximations of the correlation matrix discussed in Graffelman and de Leeuw (2023) are special cases.

¹The symbol $:=$ is used for definitions. Symbol $\circ$ is used for Hadamard (elementwise) multiplication.

2 Algorithms

Our algorithms use majorization. See Lange (2016) or De Leeuw (2022) for comprehensive discussions of majorization (or MM). In this paper majorization is used to reduce the weighted least squares problem to an unweighted least squares problem, which is generally much easier to solve. Similar applications of MM are in Kiers (1997) and in Groenen et al. (2003).

2.1 Simple Majorization

We first *vectorize* the problem. This means replacing definition (1) that uses matrices by an equivalent definition in terms of vectors. Vectors are easier to work with. Define $x := \text{vec}(X)$ and $y := \text{vec}(Y)$. Loss becomes

$$\sigma(x) := (y - x)' W_D (C \otimes R) W_D (y - x). \quad (2)$$

In (2) matrix W_D is the diagonal matrix with the elements of $\text{vec}(W)$ on the diagonal.

Suppose λ is any number larger than or equal to the largest eigenvalue $\lambda_{\max}(H)$ of the matrix

$$H := W_D (C \otimes R) W_D. \quad (3)$$

Suppose $\tilde{x}$ is another vector of the same length as x . Use $x = \tilde{x} + (x - \tilde{x})$ in (2). For all x and $\tilde{x}$ we have²,

$$\begin{aligned} \sigma(x) &= ((y - \tilde{x}) - (x - \tilde{x}))' H ((y - \tilde{x}) - (x - \tilde{x})) \\ &= \sigma(\tilde{x}) - 2(x - \tilde{x})' H (y - \tilde{x}) + (x - \tilde{x})' H (x - \tilde{x}) \\ &\leq \sigma(\tilde{x}) - 2(x - \tilde{x})' H (y - \tilde{x}) + \lambda \|x - \tilde{x}\|^2 \\ &= \sigma(\tilde{x}) + \lambda \|x - \tilde{y}\|^2 - \lambda^{-1} \|H(y - \tilde{x})\|^2, \end{aligned} \quad (4)$$

with

$$\tilde{y} := \tilde{x} + \lambda^{-1} H (y - \tilde{x}). \quad (5)$$

A majorization (MM) step minimizes $\|x - \tilde{y}\|^2$ over the x satisfying the constraints. If we now *matricize* (the inverse of vectorize) definition (5) it follows that we have to minimize $\|X - \tilde{Y}\|^2$ over $\mathbb{X}$, with

$$\tilde{Y} := \tilde{X} + \lambda^{-1} W \circ \{R(W \circ (Y - \tilde{X}))C\}. \quad (6)$$

²We use $\|\cdot\|^2$ for the (unweighted) sum of squares.

Note that this expression for $\tilde{Y}$ does not use the (possibly very large) Kronecker product H . If computing the largest eigenvalue $\lambda_{\max}(H)$ (which only has to be done once before the iterations start) is a problem then we can use an upper bound.

$$\begin{aligned} \max_z \frac{z' W_D (C \otimes R) W_D z}{z' z} &= \max_z \frac{z' W_D (C \otimes R) W_D z}{z' W_D^2 z} \frac{z' W_D^2 z}{z' z} \\ &\leq \max_z \frac{z' W_D (C \otimes R) W_D z}{z' W_D^2 z} \max_z \frac{z' W_D^2 z}{z' z} = \lambda_{\max}(C) \lambda_{\max}(R) \max(W \circ W) \end{aligned} \quad (7)$$

2.2 Nested Majorization

There is a more circumspect way to avoid computing the largest eigenvalue of H . We employ two nested majorizations. In the first majorization we get rid of the Kronecker weighting and reduce each MM iteration to a weighted least squares problem that is separable, i.e. has a diagonal weight matrix. The second majorization is used to make a further reduction of this separable weighted least squares problem to a sequence of unweighted least squares problems. The majorizations are nested, in the sense that we need one or more iterations of the second majorization process within each iteration of the first majorization process.

2.2.1 First Majorization

Start with the same vectorization we used before in our simple majorization. But instead of majorizing by using $z' H z \leq \lambda_{\max}(H) z' z$ we use

$$z' H z \leq \lambda_{\max}(C \otimes R) z' W_D^2 z = \lambda_{\max}(C) \lambda_{\max}(R) z' W_D^2 z. \quad (8)$$

Let $\lambda := \lambda_{\max}(C) \lambda_{\max}(R)$. We now have

$$\begin{aligned} \sigma(x) &= ((y - \tilde{x}) - (x - \tilde{x}))' H ((y - \tilde{x}) - (x - \tilde{x})) \\ &= \sigma(\tilde{x}) - 2(x - \tilde{x})' H (y - \tilde{x}) + (x - \tilde{x})' H (x - \tilde{x}) \\ &\leq \sigma(\tilde{x}) - 2(x - \tilde{x})' H (y - \tilde{x}) + \lambda (x - \tilde{x})' W_D^2 (x - \tilde{x}) \\ &= \sigma(\tilde{x}) + \lambda (x - \tilde{y})' W_D^2 (x - \tilde{y}) + \lambda^{-1} (y - \tilde{x})' H W_D^{-2} H (y - \tilde{x}) \end{aligned} \quad (9)$$

The target for the projection becomes

$$\tilde{y} := \tilde{x} + \lambda^{-1} W_D^{-2} H (y - \tilde{x}), \quad (10)$$

which can be matricized to

$$\tilde{Y} := \tilde{X} + \lambda^{-1} W^{-1} \circ \{R(W \circ (Y - \tilde{X}))C\}. \quad (11)$$

Note that in definition (11) W^{-1} is the elementwise inverse of W .

2.2.2 Second majorization

In the “inner” majorization we need to minimize

$$S_W(X) := - \sum_{i=1}^n \sum_{j=1}^n w_{ij}^2 (\tilde{y}_{ij} - x_{ij})^2 \quad (12)$$

over $X \in \mathbb{X}$. Note that in the inner iterations the “target” $\tilde{Y}$ is the same in each iteration, because it only changes in the outer iterations. Minimizing (or decreasing) (12) can be done with the `wAddPCA()` function from the `Correplot` package (Graffelman and de Leeuw (2026)), but we for ease of reference we give the majorization for the inner iterations.

The same quadratic majorization reasoning as in the previous sections leads to the unweighted problem of minimizing

$$S(X) := \|\tilde{Y} - X\|^2, \quad (13)$$

with

$$\tilde{y}_{ij} := \tilde{x}_{ij} + \frac{w_{ij}^2}{w_{\max}^2} (\tilde{y}_{ij} - \tilde{x}_{ij}). \quad (14)$$

Inner iterations project $\tilde{Y}$ on $\mathbb{X}$ using unweighted least squares.

3 Software

The github repository at <https://github.com/deleeuw/triple> has the file `triple.R`, which contains the function `triple()`, for simple majorization, and the file `nested.R` with the R program `nested()`, for nested majorization. The file `auxiliary.R` has routines for different unweighted least squares projections in matrix space.

3.1 Arguments

Arguments are

```
args(triple)
```

```
function (ymat, wmat = array(1, dim(ymat)), rmat = diag(nrow(ymat)),  
          cmat = diag(ncol(ymat)), func = eckart_young, p = 2, xini = NULL,  
          itmax = 1e+05, eps = 1e-06, verbose = TRUE)  
NULL
```

```
args(nested)
```

```
function (ymat, wmat = array(1, dim(ymat)), rmat = diag(nrow(ymat)),  
          cmat = diag(ncol(ymat)), func = eckart_young, p = 2, xini = NULL,  
          itmax = 1e+05, inmax = 10, eps = 1e-06, ips = 1e-06, verbose = TRUE)  
NULL
```

The arguments are mostly self-explanatory. An exception is perhaps the `func` parameter, which is a separate R routine that performs the unweighted least squares projection on $\mathbb{X}$. Parameter p is a vector or list of additional parameters for `func`, if needed. The default `func` uses the truncated SVD for low-rank matrix approximation with $p = 2$. If `xini` is `NULL` (the default) then the initial estimate for both `triple()` and `nested()` is the unweighted projection of the data matrix, without taking any of the weight matrices into account. If `xini` is non-`NULL` it is used as the initial estimate. If `verbose` is `TRUE` then information for each iteration is written to `stdout`. Otherwise the programs are silent until they are done. Note that `nested()` has stopping parameters `itmax` and `eps` for the outer iterations and `inmax` and `ips` for the inner iterations.

3.2 Values

Both `triple()` and `nested` return a list with three elements.

```
list(x = xnew, loss = snw, itel = itel)
```

If the `verbose` parameter is true then `triple()` writes `itel`, `sold`, `snw`, and `epsi` to `stdout`. Here `epsi` is the sup-norm of the difference between the current and the previous solution of X . For `nested()` the iteration output also includes `ntel` (the number of inner iterations in the current outer iteration) and `ipsi` (the sup-norm of the difference between `xold` and `xnew` for the last inner iteration).

3.3 Projection Routines

The routines in `auxiliary.R` define possible values for the `func` parameter of `triple()` and `nested()`. Users can add their own functions as needed. Currently we have

- `eckart_young()` low-rank approximation.
- `column_adjust()` subtract column means and then `eckart_young`.
- `row_adjust()` subtract row means and then `eckart_young`
- `double_center()` and then `eckart_young`.
- `hankel()` finds the best fitting Hankel matrix (square matrices only).
- `toeplitz()` finds the best fitting Toeplitz matrix (square matrices only).
- `symmetric()` finds the best fitting symmetric matrix (square matrices only).
- `anti_symmetric()` finds the best fitting anti-symmetric matrix (square matrices only).
- `additive()` finds the best fitting additive matrix (i.e. with $x_{ij} = a_i + b_j$).

4 Examples

4.1 Artificial

We analyze a small artificial example, generated from random data.

4.1.1 Data

```
ymat
```

	[,1]	[,2]	[,3]
[1,]	0.5855288	0.6058875	-0.2841597
[2,]	0.7094660	-1.8179560	-0.9193220
[3,]	-0.1093033	0.6300986	-0.1162478
[4,]	-0.4534972	-0.2761841	1.8173120

```
rmat
```

	[,1]	[,2]	[,3]	[,4]
[1,]	0.6606540	0.6316750	0.2060831	0.4388240
[2,]	0.6316750	1.5141559	-0.7485539	0.5940988
[3,]	0.2060831	-0.7485539	2.0102613	-0.4442019
[4,]	0.4388240	0.5940988	-0.4442019	1.3318363

```
cmat
```

	[,1]	[,2]	[,3]
[1,]	1.3744581	0.7610956	-0.2692483
[2,]	0.7610956	1.2428719	-0.1845269
[3,]	-0.2692483	-0.1845269	1.3504256

```
wmat
```

	[,1]	[,2]	[,3]
[1,]	0.09264056	0.45164108	0.9291061
[2,]	6.13607849	0.09483528	0.7311661
[3,]	0.94326960	0.28785770	3.5605688
[4,]	3.48605937	0.68041062	0.1535224

4.1.2 Triple

```
h <- triple(yamat, rmat = rmat, cmat = cmat, wmat = wmat, verbose = FALSE)
h$itel
```

```
[1] 28251
```

```
h$loss
```

```
[1] 0.0006645135
```

```
h$x
```

	[,1]	[,2]	[,3]
[1,]	-0.07238101	0.7118148	-0.2867966
[2,]	0.71448655	-2.1034735	-0.9179862
[3,]	-0.08977028	0.5685101	-0.1163371
[4,]	-0.45002486	-0.2906866	1.8085130

4.1.3 Nested, inmax = 1

```
h <- nested(yamat, rmat = rmat, cmat = cmat, wmat = wmat, inmax = 1, verbose = FALSE)
h$itel
```

```
[1] 56285
```

```
h$loss
```

```
[1] 0.0006705183
```

```
h$x
```

	[,1]	[,2]	[,3]
[1,]	-0.07268466	0.7098735	-0.2856829
[2,]	0.71435241	-2.0877511	-0.9185953
[3,]	-0.09053565	0.5701966	-0.1165036
[4,]	-0.45001529	-0.2910992	1.8029899

4.1.4 Nested, inmax = 5

```
h <- nested(yamat, rmat = rmat, cmat = cmat, wmat = wmat, inmax = 5, verbose = FALSE)
h$itel
```

```
[1] 11292
```

```
h$loss
```

```
[1] 0.0006705229
```

```
h$x
```

	[,1]	[,2]	[,3]
[1,]	-0.07268506	0.7098702	-0.2856778
[2,]	0.71435237	-2.0877448	-0.9185993
[3,]	-0.09053604	0.5701999	-0.1165041
[4,]	-0.45001522	-0.2910987	1.8029850

4.1.5 Nested, inmax = 10

```
h <- nested(yamat, rmat = rmat, cmat = cmat, wmat = wmat, inmax = 10, verbose = FALSE)
h$itel
```

```
[1] 5674
```

```
h$loss
```

```
[1] 0.0006705318
```

```
h$x
```

	[,1]	[,2]	[,3]
[1,]	-0.07268565	0.7098653	-0.2856708
[2,]	0.71435231	-2.0877316	-0.9186048
[3,]	-0.09053672	0.5702046	-0.1165048
[4,]	-0.45001520	-0.2910982	1.8029762

4.1.6 Nested, inmax = 100

```
h <- nested(yamat, rmat = rmat, cmat = cmat, wmat = wmat, inmax = 100, verbose = FALSE)
h$itel
```

```
[1] 634
```

```
h$loss
```

```
[1] 0.0006707031
```

```
h$x
```

	[,1]	[,2]	[,3]
[1,]	-0.07269601	0.7097798	-0.2855324
[2,]	0.71435159	-2.0874979	-0.9187098
[3,]	-0.09054544	0.5702865	-0.1165185
[4,]	-0.45001542	-0.2910846	1.8027803

4.1.7 Conclusion

The artificial example is a minimal form of quality control. It shows us that `triple()` and the various `nested()` versions converge (at least stop before reaching the 100,000 iteration limit), and all converge to the same solution (when started at the same initial point and when using the same stop criteria).

The example also shows us that convergence can be very slow and close to sublinear. Function `triple()` uses about 30,000 iterations, and `nested()` with a single inner iteration uses about 60,000 iterations. That means, in this case, 30,000 or 60,000 Eckart-Young decompositions. Ouch !

4.2 Correspondence Analysis

In correspondence analysis and multiple correspondence analysis the Kronecker weight are the diagonal matrices with the reciprocals of the marginals of the table.

```
library(anacor)
data(galton)
ymat <- as.matrix(galton) / sum(galton)
rmat <- diag(1 / rowSums(ymat))
cmat <- diag(1 / colSums(ymat))
wmat <- array(1, dim(ymat))
h <- triple(ymat, wmat, rmat, cmat, p = 2, verbose = FALSE)
#print(h$x)
#print(h$itel)
#print(h$loss)
#svd(h$x)
```

4.3 Canonical Analysis

4.4 Triple Weighting

5 Discussion

The complications in minimizing loss function (1) arise from the combination of the (R, C) and the W weights. If either one of these two sets of weights is removed (or set equal to the identity) minimization becomes much easier. One of the two majorizations in the nested process disappears. There is no need for the largest eigenvalue of the large matrix H from (3). The same thing is true if we replace loss function (1) by

$$\sigma(X) := \text{tr } R(W \circ (Y) - X)C(W \circ (Y) - X)'. \quad (15)$$

In that case we also only need the first majorization applied to $W \circ (Y)$.

The distinction between simple and nested majorization is relative. In simple majorization we may need to solve an eigenvalue problem in each iteration, and the eigenvalue computation itself defines an inner iteration. In that sense simple majorization is also nested, we just call it simple because eigenvalue computations are implemented as black boxes that had the iteration details. We could implement our projections as explicit iterative processes with update rules and stop criteria. In that case our nested majorization will have outer, inner, and innermost iterations and three sets of stop criteria.

In many previous majorization methods the stop criterion is defined using a decrease in the loss function from one iteration to the next of less than epsilon. In `triple()` and `nested()` we prefer to use the sup-norm of the change in the solution from one iteration to the next. This is a stricter criterion, because solutions generally converge slower than loss functions. It is also more satisfactory, because change in the solution measures basically how well the conditions for stationarity (vanishing of the gradient or the Karush-Kuhn-Tucker conditions in the constrained case) are satisfied. By computing the change in the projections we also circumvent identifiability problems. If the Eckart-Young projection, for example, is parametrized as $X = AB'$, then we should not look at the change in A and B from one iteration to the next, but in the change of AB' . In MDS we should not look at the change in the configuration, but in the change of the distances.

For large examples that would be impractical. It suggests that acceleration techniques are needed. As currently implemented all projections, such as Eckart-Young, have a cold start, i.e. do not use projections in previous iterations. This is clearly wasteful.

References

- De Leeuw, Jan. 1984. “Fixed-Rank Approximation with Singular Weight Matrices.” *Computational Statistics Quarterly* 1: 3–12. <https://jansweb.netlify.app/publication/deleeuw-a-84-b/deleeuw-a-84-b.pdf>.
- De Leeuw, Jan. 2022. *Block Relaxation Methods in Statistics*. Bookdown. <https://jansweb.netlify.app/publication/deleeuw-b-21-c/deleeuw-b-21-c.pdf>.
- De Leeuw, Jan, and Irina Kukuyeva. 2009. “Fixed-Rank Approximation with Constraints.” <https://jansweb.netlify.app/publication/deleeuw-kukuyeva-u-09/deleeuw-kukuyeva-u-09.pdf>.
- Gabriel, K. Ruben, and S. Zamir. 1979. “Lower Rank Approximation of Matrices by Least Squares with Any Choice of Weights.” *Technometrics* 21 (4): 489–98.
- Graffelman, Jan. 2026a. “On the Approximation of the Between-Set Correlation Matrix by Canonical Correlation Analysis.” Unpublished manuscript.
- Graffelman, Jan. 2026b. “Precision-Weighted Compositional Data Analysis by Weighted Alternating Least Squares and Majorization Algorithms.” Unpublished manuscript.
- Graffelman, Jan, and Jan de Leeuw. 2023. “Improved Approximation and Visualization of the Correlation Matrix.” *American Statistician* 77 (4): 432–42.
- Graffelman, Jan, and Jan de Leeuw. 2026. *Correlplot: A Collection of Functions for Graphing Correlation Matrices*. <https://CRAN.R-project.org/package=Correlplot>.
- Groenen, Patrick J. F., Patrizia Giaquinto, and Henk A. L. Kiers. 2003. *Weighted Majorization Algorithms for Weighted Least Squares Decomposition Models*. Econometric Institute Report EI 2003-09. Econometric Institute, Erasmus University Rotterdam. <https://repub.eur.nl/pub/1700>.
- Gupta, Arjan K., and D. K. Nagar. 2000. *Matrix Variate Distributions*. Chapman & Hall.
- Kiers, Henk A. L. 1997. “Weighted Least Squares Fitting Using Iterative Ordinary Least Squares Algorithms.” *Psychometrika* 62: 251–66.
- Lange, Kenneth. 2016. *MM Optimization Algorithms*. SIAM.