psych

MDPI

Article

Evaluating the Observed Log-Likelihood Function in Two-Level Structural Equation Modeling with Missing Data: From Formulas to R Code

Yves Rosseel

Department of Data Analysis, Ghent University, Henri Dunantlaan 1, B-9000 Ghent, Belgium; Yves.Rosseel@UGent.be

Abstract: This paper discusses maximum likelihood estimation for two-level structural equation models when data are missing at random at both levels. Building on existing literature, a computationally efficient expression is derived to evaluate the observed log-likelihood. Unlike previous work, the expression is valid for the special case where the model implied variance–covariance matrix at the between level is singular. Next, the log-likelihood function is translated to R code. A sequence of R scripts is presented, starting from a naive implementation and ending at the final implementation as found in the `lavaan` package. Along the way, various computational tips and tricks are given.

Keywords: structural equation modeling; multilevel; missing data; observed loglikelihood; R code

check for updates

Citation: Rosseel, Y. Evaluating the Observed Log-Likelihood Function in Two-Level Structural Equation Modeling with Missing Data: From Formulas to R Code. *Psych* **2021**, *3*, 197–232. https://doi.org/10.3390/psych3020017

Academic Editor: Alexander Robitzsch

Received: 30 April 2021
Accepted: 4 June 2021
Published: 7 June 2021

1. Introduction

Applied users are increasingly using multilevel structural equation modeling (SEM), partly due to its wide availability in several software packages. The open-source R package `lavaan` [1] currently only handles two-level continuous data using maximum likelihood (ML) estimation. Until now, data had to be complete: all cases with missing values were removed from the data before the analysis could start. However, in `lavaan` version 0.6–9, support for missing data is finally available. I was fortunate to have found two excellent sources in the literature [2,3] that discuss the (observed) log-likelihood function for two-level SEMs. Both sources describe how this log-likelihood function can be re-expressed in a computationally convenient form. From here on, I will refer to these two results as the McDonald solution and the du Toit and du Toit solution, respectively. Interestingly, the two solutions are different but equivalent, which I will clarify in this paper. Both solutions also suffer from the requirement that the model-implied variance–covariance matrix of the variables at the between level ($\boldsymbol{\Sigma}_b$) must be nonsingular. In practice, this often leads to numerical problems when $\boldsymbol{\Sigma}_b$ is singular, by design, or nearly singular for a particular model/data combination.

I have several goals for this paper. First, I wish to document the exact formulas that are used by `lavaan` 0.6–9 to compute the so-called observed log-likelihood used to obtain the ML estimates for two-level SEMs when data are missing. Second, I will suggest a small modification to the existing solutions that will allow $\boldsymbol{\Sigma}_b$ to be singular. Finally, I will show and discuss five R scripts that evaluate the log-likelihood function. An example model and dataset are also provided so that readers can run these scripts on their own machines. Because R can be very slow (compared to compiled programming languages), care is needed to obtain reasonably efficient code. The five scripts document the various stages of the development of "real" R code and hopefully offer some insight into the typical process that is used (at least by me) to translate formulas to production-ready R code.

The paper is organized as follows. In Section 2, I briefly review single-level SEM, with a focus on the log-likelihood function (or an equivalent discrepancy function), which is needed to obtain ML estimates. In Section 3, I discuss the log-likelihood function that is

used for two-level SEM. In both sections, I first discuss the complete data setting, followed by the missing data setting. In Section 4, I discuss the R code listed in the Appendices. The paper ends with some final thoughts housed in Section 5.

2. SEM for Single-Level Data

In this section, I briefly describe the statistical model used in traditional SEM if the data originate from a single population. The purpose of this section is mostly to introduce notation and to provide some building blocks for the two-level sections. For Equations (1)–(8), I have used the LISREL all-y notation [4] to describe the basic formulas, but I could as well have used the Bentler-Weeks [5] or the RAM [6] notation. The remaining formulas are unaffected by this choice.

2.1. Complete Data

Let $\boldsymbol{y}$ be a P-dimensional vector randomly drawn from a population with mean vector $\boldsymbol{\mu}$ and variance–covariance matrix $\boldsymbol{\Sigma}$. A common working assumption is that the population distribution is normal. In an SEM analysis, a model is postulated that implies a certain structure for $\boldsymbol{\mu} = \mathrm{E}(\boldsymbol{y})$ and $\boldsymbol{\Sigma} = \mathrm{Var}(\boldsymbol{y})$. The model typically consists of two parts: a measurement part and a structural part. The measurement part of the model is defined as

$$\boldsymbol{y} = \boldsymbol{\nu} + \boldsymbol{\Lambda}\,\boldsymbol{\eta} + \boldsymbol{\epsilon}, \tag{1}$$

where $\boldsymbol{y}$ is a $P \times 1$ vector of observed variables, $\boldsymbol{\nu}$ is a $P \times 1$ vector of intercepts, $\boldsymbol{\Lambda}$ is a $P \times M$ matrix of factor loadings relating the latent variables to the observed variables, and $\boldsymbol{\epsilon}$ is a $P \times 1$ vector of residual errors. The structural part of the model (also called the "latent variable model") is defined as

$$\boldsymbol{\eta} = \boldsymbol{\alpha} + \mathbf{B}\,\boldsymbol{\eta} + \boldsymbol{\zeta}, \tag{2}$$

where $\boldsymbol{\eta}$ is an $M \times 1$ vector of latent variables, $\boldsymbol{\alpha}$ is an $M \times 1$ vector of intercepts, $\mathbf{B}$ is an $M \times M$ matrix of coefficients for the regressions of the latent variables on each other, and $\boldsymbol{\zeta}$ is an $M \times 1$ vector of disturbance terms. The diagonal elements of $\mathbf{B}$ must be zero, and $(\mathbf{I} - \mathbf{B})$ should be invertible. To simplify the notation, I will use the convention that an observed variable involved in the structural part of the model is upgraded to a latent variable, with the observed variable as its only indicator with no measurement error.

To derive the model-implied moments, the structural part is written in its so-called reduced form, where the endogenous variables only appear on the left-hand side of the equation:

$$\begin{aligned} \boldsymbol{\eta} &= (\mathbf{I} - \mathbf{B})^{-1}(\boldsymbol{\alpha} + \boldsymbol{\zeta}) \\ &= (\mathbf{I} - \mathbf{B})^{-1}\boldsymbol{\alpha} + (\mathbf{I} - \mathbf{B})^{-1}\boldsymbol{\zeta}. \end{aligned} \tag{3}$$

Plugging in the reduced form of the structural model into the measurement model results in

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{\nu} + \boldsymbol{\Lambda}\left[(\mathbf{I} - \mathbf{B})^{-1}\boldsymbol{\alpha} + (\mathbf{I} - \mathbf{B})^{-1}\boldsymbol{\zeta}\right] + \boldsymbol{\epsilon} \\ &= \boldsymbol{\nu} + \boldsymbol{\Lambda}(\mathbf{I} - \mathbf{B})^{-1}\boldsymbol{\alpha} + \boldsymbol{\Lambda}(\mathbf{I} - \mathbf{B})^{-1}\boldsymbol{\zeta} + \boldsymbol{\epsilon}. \end{aligned} \tag{4}$$

Assuming $\mathrm{E}(\boldsymbol{\zeta}) = \mathbf{0}$ and writing $\mathrm{Var}(\boldsymbol{\zeta}) = \boldsymbol{\Psi}$, it follows from (3) that

$$\mathrm{E}(\boldsymbol{\eta}) = (\mathbf{I} - \mathbf{B})^{-1}\boldsymbol{\alpha} \tag{5}$$

$$\mathrm{Var}(\boldsymbol{\eta}) = (\mathbf{I} - \mathbf{B})^{-1}\boldsymbol{\Psi}\,(\mathbf{I} - \mathbf{B})^{-1\prime}. \tag{6}$$

Similarly, assuming $\mathrm{E}(\boldsymbol{\epsilon}) = \mathbf{0}$, and $\mathrm{Cov}(\boldsymbol{\zeta}, \boldsymbol{\epsilon}) = \mathbf{0}$ and using the notation $\mathrm{Var}(\boldsymbol{\zeta}) = \boldsymbol{\Psi}$ and $\mathrm{Var}(\boldsymbol{\epsilon}) = \boldsymbol{\Theta}$, it follows from (4) that

$$\boldsymbol{\mu} = \boldsymbol{\nu} + \boldsymbol{\Lambda}(\mathbf{I} - \mathbf{B})^{-1}\boldsymbol{\alpha} \tag{7}$$

$$\boldsymbol{\Sigma} = \boldsymbol{\Lambda}\,(\mathbf{I} - \mathbf{B})^{-1}\boldsymbol{\Psi}\,(\mathbf{I} - \mathbf{B})^{-1\prime}\boldsymbol{\Lambda}' + \boldsymbol{\Theta}. \tag{8}$$

The model matrices are then $\boldsymbol{\nu}$, $\boldsymbol{\alpha}$, $\boldsymbol{\Lambda}$, $\boldsymbol{\Theta}$, $\boldsymbol{\Psi}$, and $\mathbf{B}$. A structural model can be defined by setting some elements of these model matrices to a fixed constant (often zero or unity), while allowing other elements to be free. The T free elements are collected in a T-dimensional parameter vector $\boldsymbol{\theta}$. For a given choice of values of $\boldsymbol{\theta}$, the model-implied moments can be written as $\boldsymbol{\mu}(\boldsymbol{\theta})$ and $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ to stress that they are a function of these free parameters.

Given a set of i.i.d. data vectors $\mathcal{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N\}$, ML estimation seeks those values of $\boldsymbol{\theta}$ that maximize the multivariate normal log-likelihood given by

$$\mathrm{logl}(\boldsymbol{\theta}|\mathcal{Y}) = -\frac{NP}{2}\ln(2\pi) - \frac{N}{2}\ln|\boldsymbol{\Sigma}(\boldsymbol{\theta})| - \frac{1}{2}\sum_{i=1}^{N}[\boldsymbol{y}_i - \boldsymbol{\mu}(\boldsymbol{\theta})]'\boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}[\boldsymbol{y}_i - \boldsymbol{\mu}(\boldsymbol{\theta})]. \tag{9}$$

If the data are complete (i.e., no missing values), this log-likelihood can be rewritten in terms of the sample moments:

$$\begin{aligned}\mathrm{logl}(\boldsymbol{\theta}|\mathcal{Y}) = &-\frac{NP}{2}\ln(2\pi) - \frac{N}{2}\ln|\boldsymbol{\Sigma}(\boldsymbol{\theta})| \\ &- \frac{N}{2}\mathrm{tr}\left[\mathbf{S}\boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}\right] - \frac{N}{2}[\bar{\boldsymbol{y}} - \boldsymbol{\mu}(\boldsymbol{\theta})]'\boldsymbol{\Sigma}^{-1}[\bar{\boldsymbol{y}} - \boldsymbol{\mu}(\boldsymbol{\theta})],\end{aligned} \tag{10}$$

where $\bar{\boldsymbol{y}}$ is the sample mean vector, $\mathbf{S}$ is the (biased) sample variance–covariance matrix (divided by N), and $\mathrm{tr}[\cdot]$ is the trace operator that computes the sum of the diagonal elements of its matrix argument. The value of $\boldsymbol{\theta}$ that maximizes the log-likelihood in (10) is called the ML estimator of $\boldsymbol{\theta}$ and will be denoted by $\hat{\boldsymbol{\theta}}$. The resulting model-implied moments will be denoted by $\boldsymbol{\mu}(\hat{\boldsymbol{\theta}})$ and $\boldsymbol{\Sigma}(\hat{\boldsymbol{\theta}})$, or $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ for short.

The unrestricted (or saturated) model simply assumes that $\hat{\boldsymbol{\theta}} = (\bar{\boldsymbol{y}}', \mathrm{vech}[\mathbf{S}]')'$, where $\mathrm{vech}[\cdot]$ is an operator that stacks the elements of the lower triangular part of a symmetric matrix columnwise in a vector. As a result, for the unrestricted model, $\hat{\boldsymbol{\Sigma}} = \mathbf{S}$ and $\hat{\boldsymbol{\mu}} = \bar{\boldsymbol{y}}$. In this case, the log-likelihood equals

$$\mathrm{logl}(\bar{\boldsymbol{y}}, \mathbf{S}) = -\frac{NP}{2}\ln(2\pi) - \frac{N}{2}\ln|\mathbf{S}| - \frac{N}{2}P \tag{11}$$

because $\bar{\boldsymbol{y}} - \boldsymbol{\mu}(\hat{\boldsymbol{\theta}}) = \bar{\boldsymbol{y}} - \bar{\boldsymbol{y}} = \mathbf{0}$ and $\mathrm{tr}\left[\boldsymbol{\Sigma}(\hat{\boldsymbol{\theta}})^{-1}\mathbf{S}\right] = \mathrm{tr}\left[\mathbf{S}^{-1}\mathbf{S}\right] = \mathrm{tr}[\mathbf{I}_P] = P$, with $\mathbf{I}_P$ an identity matrix of dimension P. Comparing the restricted model to the saturated model can be accomplished by the standard likelihood ratio test (LRT) given by

$$\mathrm{LRT} = -2[\mathrm{logl}(\boldsymbol{\theta}) - \mathrm{logl}(\bar{\boldsymbol{y}}, \mathbf{S})] = 2[\mathrm{logl}(\bar{\boldsymbol{y}}, \mathbf{S}) - \mathrm{logl}(\boldsymbol{\theta})] \tag{12}$$

$$= N \cdot F_{ML}(\boldsymbol{\theta}), \tag{13}$$

where

$$F_{ML}(\boldsymbol{\theta}) = [\bar{\boldsymbol{y}} - \boldsymbol{\mu}(\boldsymbol{\theta})]'\boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}[\bar{\boldsymbol{y}} - \boldsymbol{\mu}(\boldsymbol{\theta})] + \mathrm{tr}\left[\mathbf{S}\boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}\right] - \ln|\mathbf{S}\boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}| - P \tag{14}$$

is commonly called the normal theory-based ML discrepancy function [7]. Because $\mathrm{logl}(\bar{\boldsymbol{y}}, \mathbf{S})$ does not depend on $\boldsymbol{\theta}$, minimizing (14) is equivalent to maximizing (10). In the common

case, when the mean structure $\boldsymbol{\mu}(\boldsymbol{\theta})$ is unrestricted (hence $\boldsymbol{\mu}(\boldsymbol{\theta}) = \bar{\boldsymbol{y}}$), the discrepancy function simplifies to

$$\begin{aligned} F_{ML} &= \text{tr}\left[\mathbf{S}\boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}\right] - \ln|\mathbf{S}\boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}| - P \\ &= \text{tr}\left[\mathbf{S}\boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}\right] + \ln|\boldsymbol{\Sigma}(\boldsymbol{\theta})| - \ln|\mathbf{S}| - P. \end{aligned} \tag{15}$$

To minimize the objective function in (14) or (15), an optimization method is needed. Popular choices include Newton–Raphson, Fisher scoring, and various quasi-Newton methods. The `lavaan` package uses by default a quasi-Newton method implemented in the `nlminb()` function, although a custom implementation of Fisher scoring is also available. In any case, it is necessary to evaluate the objective function many, many times. Therefore, it is crucial to be able to evaluate this function in a fast and efficient way. Fortunately, from a computational point of view, both (14) and (15) are easy objective functions to evaluate. The only complication is the inverse and the determinant of $\boldsymbol{\Sigma}(\boldsymbol{\theta})$, but this matrix has dimension $P \times P$, and P is usually small. The determinant of $\mathbf{S}$ only needs to be computed once. Note that the computation time is independent of the sample size because (14) and (15) only rely on summary statistics.

2.2. Missing Data

When some values in $\boldsymbol{y}$ are missing, I will assume in this paper that the missing mechanism is either missing completely at random (MCAR) or missing at random (MAR) [8]. In this case, ML can still be used to estimate $\boldsymbol{\theta}$ by using all available data. This is sometimes called full information maximum likelihood (FIML) estimation. Because it is no longer possible to summarize the incomplete data by its sample mean and variance–covariance matrix, I revert to the casewise log-likelihood function. For notational purposes, it is convenient to introduce a selection matrix $\mathbf{Q}_i$ for each observation. A selection matrix can convert a vector that includes both observed and missing components (say, $\boldsymbol{y}_i$) to a complete vector containing only the observed components (say, $\boldsymbol{y}_i^o$). To construct these selection matrices, I start from an identity matrix and remove the rows that correspond to the missing values. For example, suppose $P = 5$ and the second and fourth value are missing in $\boldsymbol{y}_i$; thus, $\mathbf{Q}_i$ is defined to be

$$\mathbf{Q}_i = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

so that $\boldsymbol{y}_i^o = \mathbf{Q}_i \boldsymbol{y}_i$. Let P_i be the number of rows of $\mathbf{Q}_i$. P_i may be different from observation to observation. For each observation, $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are defined as follows:

$$\boldsymbol{\mu}_i = \mathbf{Q}_i \boldsymbol{\mu} \tag{16}$$

and

$$\boldsymbol{\Sigma}_i = \mathbf{Q}_i \boldsymbol{\Sigma} \mathbf{Q}_i'. \tag{17}$$

With this notation in place, the log-likelihood for a single observation can be written as

$$\text{logl}(\boldsymbol{\theta}|\boldsymbol{y}_i) = -\frac{1}{2} P_i \ln(2\pi) - \frac{1}{2} \ln|\boldsymbol{\Sigma}_i(\boldsymbol{\theta})| - \frac{1}{2}[\boldsymbol{y}_i^o - \boldsymbol{\mu}_i(\boldsymbol{\theta})]' \boldsymbol{\Sigma}_i(\boldsymbol{\theta})^{-1} [\boldsymbol{y}_i^o - \boldsymbol{\mu}_i(\boldsymbol{\theta})]. \tag{18}$$

The total log-likelihood for the entire sample is obtained by summing over the individual log-likelihoods:

$$\text{logl}(\boldsymbol{\theta}|\mathcal{Y}) = \sum_{i=1}^{N} \text{logl}(\boldsymbol{\theta}|\boldsymbol{y}_i). \tag{19}$$

In practice, instead of maximizing (19), it is convenient to minimize an equivalent expression where the first term of (18) is dropped (because it does not depend on $\boldsymbol{\theta}$) and where

(18) is multiplied by a factor minus two. The resulting objective function for the entire sample then becomes

$$F_{FIML} = \sum_{i=1}^{N} \ln |\boldsymbol{\Sigma}_i(\boldsymbol{\theta})| + [\boldsymbol{y}_i^o - \boldsymbol{\mu}_i(\boldsymbol{\theta})]' \boldsymbol{\Sigma}_i(\boldsymbol{\theta})^{-1} [\boldsymbol{y}_i^o - \boldsymbol{\mu}_i(\boldsymbol{\theta})]. \tag{20}$$

Evaluating this objective function is much more costly than evaluating the objective function in (14) for the complete data setting. The determinant and the inverse of $\boldsymbol{\Sigma}_i(\boldsymbol{\theta})$ need to be computed for every observation, and the computation time will increase (linearly) with the sample size. To decrease the computational burden, two techniques are commonly used. The first technique is to combine all observations that have the same missing pattern. For all observations that share the same missing pattern, it is possible to use the same determinant and inverse of $\boldsymbol{\Sigma}(\boldsymbol{\theta})$, but only for this particular pattern. In addition, consider several observations that feature the same missing pattern; it is possible to summarize them by their summary statistics and to use (14) for this pattern. In many datasets, the number of missing patterns is far less than the number of observations. The second technique relates to the computation of the determinant and the inverse of $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ for each pattern. Instead, these quantities may be computed for the complete pattern and then "updated" for each pattern exploiting the well-known formulas for the inverse and determinant of a partitioned matrix (see Appendix B.1). This updating approach is much faster than computing the determinant and the inverse from scratch. By combining these two techniques, it becomes possible to dramatically lower the computation time that is needed to evaluate (20).

2.3. Implementation in `lavaan`

Readers can explore the source code of the `lavaan` package, which can be found on github using the link https://github.com/yrosseel/lavaan/tree/master/R. The file `lav_mvnorm.R` contains functions that are used when data are complete. The evaluation of the objective function for complete data in (14) is done in a function called `lav_mvnorm_loglik_samplestats()`. The file `lav_mvnorm_missing.R` contains various functions that handle missing data (assuming multivariate normality). The function used to evaluate the objective function in (20) in the presence of missing data is called `lav_mvnorm_missing_loglik_samplestats()`. The code for updating the determinant and the inverse of a symmetric matrix after removing rows and columns from that matrix can be found in the function `lav_matrix_symmetric_inverse_update()`, in the file `lav_matrix.R`.

3. SEM for Two-Level Data

In this section, I examine two-level data. Two-level data arise from a nested sampling scheme, where level-1 units (e.g., students) are nested within level-2 units (e.g., schools). Next, I briefly discuss the complete data setting before delving into the main topic of this paper: two-level data with missing values.

3.1. Complete Data

Given J clusters with variables measured at both levels, let $\boldsymbol{y}_{ji}$ be the P-dimensional vector of level-1 variables from unit i in cluster j, where $i = 1, 2, \ldots, n_j$ and $j = 1, 2, \ldots, J$. Let $\boldsymbol{z}_j$ be the K-dimensional vector of level-2 variables for cluster j. Combining $\boldsymbol{z}_j$ with the n_j observations of cluster j, a $P_j = K + (n_j \times P)$ dimensional vector $\boldsymbol{v}_j$ can be defined containing all observed variables for a single cluster j:

$$\boldsymbol{v}_j' = (\boldsymbol{z}_j', \boldsymbol{y}_{j1}', \boldsymbol{y}_{j2}', \ldots, \boldsymbol{y}_{jn_j}')'. \tag{21}$$

I will use the notation $\boldsymbol{\mu}_j = \mathrm{E}(\boldsymbol{v}_j)$ and $\mathbf{V}_j = \mathrm{Var}(\boldsymbol{v}_j)$. In this paper, I only consider two-level SEMs with random intercepts. For models with random slopes, see [9]. When only random intercepts are involved, the level-1 vector $\boldsymbol{y}_{ji}$ can be split into a within and a between part:

$$\boldsymbol{y}_{ji} = \boldsymbol{y}_{jb} + \boldsymbol{y}_{jwi}, \tag{22}$$

where it is assumed that $\mathrm{Cov}(\boldsymbol{y}_{jb}, \boldsymbol{y}_{jwi}) = \mathbf{0}$ and $\mathrm{Cov}(\boldsymbol{z}_j, \boldsymbol{y}_{jwi}) = \mathbf{0}$ for all i and j. I will use the notation $\boldsymbol{\mu}_y = \mathrm{E}(\boldsymbol{y}_{ji})$, $\boldsymbol{\Sigma}_b = \mathrm{Var}(\boldsymbol{y}_{jb})$, and $\boldsymbol{\Sigma}_w = \mathrm{Var}(\boldsymbol{y}_{jwi})$ for all i and j. For some level-1 variables in the model, it may be convenient to set the between part in (22) to zero. I refer to these variables as "within-only" variables. The corresponding rows and columns in $\boldsymbol{\Sigma}_b$ are then zero. In this paper, it is always assumed that $\boldsymbol{\Sigma}_w$ is positive definite, but $\boldsymbol{\Sigma}_b$ can be singular. Further, I will use the notation $\boldsymbol{\mu}_z = \mathrm{E}(\boldsymbol{z}_j)$, $\boldsymbol{\Sigma}_{zz} = \mathrm{Var}(\boldsymbol{z}_j)$, and $\boldsymbol{\Sigma}_{zy} = \mathrm{Cov}(\boldsymbol{z}_j, \boldsymbol{y}_{jb})$ for all j. The mean vector $\boldsymbol{\mu}_j$ can then be written as

$$\boldsymbol{\mu}_j' = (\boldsymbol{\mu}_z', \mathbf{1}_{n_j}' \otimes \boldsymbol{\mu}_y')', \tag{23}$$

where $\mathbf{1}_{n_j}$ is a unity vector of size n_j, and $\otimes$ is the Kronecker-product operator. The variance–covariance matrix $\mathbf{V}_j$ can be written as

$$\mathbf{V}_j = \begin{bmatrix} \boldsymbol{\Sigma}_{zz} & \mathbf{1}_{n_j}' \otimes \boldsymbol{\Sigma}_{zy} \\ \mathbf{1}_{n_j} \otimes \boldsymbol{\Sigma}_{yz} & \mathbf{1}_{n_j}\mathbf{1}_{n_j}' \otimes \boldsymbol{\Sigma}_b + \mathbf{I}_{n_j} \otimes \boldsymbol{\Sigma}_w \end{bmatrix}, \tag{24}$$

where I use $\boldsymbol{\Sigma}_{yz} = \boldsymbol{\Sigma}_{zy}'$. Note that $\mathbf{1}_{n_j}\mathbf{1}_{n_j}'$ is just a unity matrix of dimension $n_j \times n_j$.

Structural models may now be defined by restricting the elements of $\boldsymbol{\mu}_z$, $\boldsymbol{\mu}_y$, $\boldsymbol{\Sigma}_{zz}$, $\boldsymbol{\Sigma}_{zy}$, $\boldsymbol{\Sigma}_w$, and $\boldsymbol{\Sigma}_b$ to be a function of a parameter vector $\boldsymbol{\theta}$. This can be accomplished by setting up an SEM for each level, resulting in a set of model matrices for the within part and another set of model matrices for the between part—not unlike a two-group SEM analysis. For a given model and a given cluster, $\boldsymbol{\mu}_j$ and $\mathbf{V}_j$ can be written as a function of $\boldsymbol{\theta}$. Given data $\mathcal{Z} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_J\}$ for a random sample of J clusters, the total log-likelihood for the complete sample is given by

$$\mathrm{logl}(\boldsymbol{\theta}|\mathcal{Z}) = \sum_{j=1}^{J} \mathrm{logl}(\boldsymbol{\theta}|\boldsymbol{v}_j), \tag{25}$$

where

$$\mathrm{logl}(\boldsymbol{\theta}|\boldsymbol{v}_j) = -\frac{1}{2} P_j \ln(2\pi) - \frac{1}{2} \ln|\mathbf{V}_j(\boldsymbol{\theta})| - \frac{1}{2}[\boldsymbol{v}_j - \boldsymbol{\mu}_j(\boldsymbol{\theta})]' \mathbf{V}_j(\boldsymbol{\theta})^{-1} [\boldsymbol{v}_j - \boldsymbol{\mu}_j(\boldsymbol{\theta})]. \tag{26}$$

Again, maximizing the log-likelihood can be replaced by minimizing the objective function:

$$F_{ML2} = \sum_{j=1}^{J} \ln|\mathbf{V}_j(\boldsymbol{\theta})| + [\boldsymbol{v}_j - \boldsymbol{\mu}_j(\boldsymbol{\theta})]' \mathbf{V}_j(\boldsymbol{\theta})^{-1} [\boldsymbol{v}_j - \boldsymbol{\mu}_j(\boldsymbol{\theta})]. \tag{27}$$

Note the close similarity between Equations (18)–(20) respectively, but a crucial difference is that P_j (and therefore $\mathbf{V}_j$) can become very large within applications. For example, consider a fairly large dataset, with $P = 20$ and $K = 5$. For a given cluster, let $n_j = 100$, $P_j = 5 + 100 \times 20 = 2005$; this would necessitate finding the determinant and the inverse of a 2005×2005 dimensional matrix $\mathbf{V}_j$, only for this cluster. From a computational point of view, this is not practical. To overcome this problem, the special structure of $\mathbf{V}_j$ must be exploited. Ideally, (27) can be rewritten so that the determinant and the inverse of matrices that need to computed are of size P or K. Before showing a solution, I must first

introduce some notation. The matrix $\mathbf{V}_j$ in (24) consists of four blocks and can be written symbolically as

$$\mathbf{V}_j = \begin{bmatrix} \mathbf{V}_{j(zz)} & \mathbf{V}_{j(zy)} \\ \mathbf{V}_{j(yz)} & \mathbf{V}_{j(yy)} \end{bmatrix}. \tag{28}$$

For the corresponding blocks of the inverse of $\mathbf{V}_j$, I will use the following notation:

$$\mathbf{V}_j^{-1} = \begin{bmatrix} \mathbf{V}_{j(zz)} & \mathbf{V}_{j(zy)} \\ \mathbf{V}_{j(yz)} & \mathbf{V}_{j(yy)} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{V}_j^{zz} & \mathbf{V}_j^{zy} \\ \mathbf{V}_j^{yz} & \mathbf{V}_j^{yy} \end{bmatrix}. \tag{29}$$

Further, I define δ_j to be the difference between v_j and $\boldsymbol{\mu}_j$:

$$\delta_j' = (\boldsymbol{v}_j - \boldsymbol{\mu}_j)' = (\delta_{jz}', \delta_{jy}')', \tag{30}$$

where the last term δ_j is partitioned into a z part and a y part—just like $\mathbf{V}_j$. The quadratic form in (27) can be written as:

$$\begin{aligned} \delta_j' \mathbf{V}_j^{-1} \delta_j &= \delta_{jz}' \mathbf{V}_j^{zz} \delta_{jz} + \delta_{jz}' \mathbf{V}_j^{zy} \delta_{jy} + \delta_{jy}' \mathbf{V}_j^{yz} \delta_{jz} + \delta_{jy}' \mathbf{V}_j^{yy} \delta_{jy} && (31) \\ &= \delta_{jz}' \mathbf{V}_j^{zz} \delta_{jz} + 2(\delta_{jz}' \mathbf{V}_j^{zy} \delta_{jy}) + \delta_{jy}' \mathbf{V}_j^{yy} \delta_{jy} && (32) \\ &= q_j^{zz} + 2q_j^{zy} + q_j^{yy}. && (33) \end{aligned}$$

The objective function can then be written as follows:

$$F_{ML2} = \sum_{j=1}^{J} \ln |\mathbf{V}_j| + q_j^{zz} + 2q_j^{zy} + q_j^{yy}. \tag{34}$$

The main task now is to find a computationally convenient expression for the three quadratic terms q_j^{yy}, q_j^{zy}, and q_j^{zz} as well as for the determinant of $\mathbf{V}_j$. McDonald and Goldstein [10] accomplished this more than 30 years ago. They obtained the following expressions for the determinant and for the quadratic forms:

$$\begin{aligned} \ln |\mathbf{V}_j| &= \ln |\boldsymbol{\Sigma}_{zz}| + (n_j - 1) \ln |\boldsymbol{\Sigma}_w| + \ln |n_j \cdot \boldsymbol{\Sigma}_{b.z} + \boldsymbol{\Sigma}_w| && (35) \\ q_j^{yy} &= \mathrm{tr}[(\mathbf{Y}_j - \boldsymbol{\mu}_y)\boldsymbol{\Sigma}_w^{-1}(\mathbf{Y}_j - \boldsymbol{\mu}_y)'] - n_j \cdot \mathbf{t}_j' \boldsymbol{\Sigma}_w^{-1} \mathbf{t}_j + n_j \cdot \mathbf{t}_j' (n_j \boldsymbol{\Sigma}_{b.z} + \boldsymbol{\Sigma}_w)^{-1} \mathbf{t}_j && (36) \\ q_j^{zy} &= -n_j \cdot \mathbf{g}_j' (n_j \boldsymbol{\Sigma}_{b.z} + \boldsymbol{\Sigma}_w)^{-1} \mathbf{t}_j && (37) \\ q_j^{zz} &= \delta_{jz}' \boldsymbol{\Sigma}_{zz}^{-1} \delta_{jz} + n_j \cdot \mathbf{g}_j' (n_j \boldsymbol{\Sigma}_{b.z} + \boldsymbol{\Sigma}_w)^{-1} \mathbf{g}_j, && (38) \end{aligned}$$

where the $n_j \times P$ matrix $\mathbf{Y}_j$ contains all level-1 observations within cluster j, and where the following notation was used:

$$\begin{aligned} \mathbf{t}_j &= \bar{\mathbf{y}}_j - \boldsymbol{\mu}_y && (39) \\ \mathbf{g}_j &= \boldsymbol{\Sigma}_{yz} \boldsymbol{\Sigma}_{zz}^{-1} \delta_{jz} && (40) \\ \boldsymbol{\Sigma}_{b.z} &= (\boldsymbol{\Sigma}_b - \boldsymbol{\Sigma}_{yz} \boldsymbol{\Sigma}_{zz}^{-1} \boldsymbol{\Sigma}_{zy}). && (41) \end{aligned}$$

Although this may seem like a more complicated function, it can be evaluated much faster than the original function in (27). A little known fact (see [11,12]) is that instead of taking the sum over all clusters, it is possible to take the sum over all cluster sizes. The different number of cluster sizes (S) is often much smaller than the number of clusters (J). In the balanced case where all clusters have the same size, $S = 1$ and no summation is

even needed. Finally, further simplification can be achieved by making use of the pooled within-sample covariance matrix defined by

$$S_{pw} = \frac{1}{(N-J)} \sum_{j=1}^{J} \sum_{i=1}^{n_j} (\boldsymbol{y}_{ji} - \bar{\mathbf{y}}_j)(\boldsymbol{y}_{ji} - \bar{\mathbf{y}}_j)'. \tag{42}$$

This results in the objective function that is effectively used by `lavaan`:

$$F_{ML2} = (N-J)\Big(\ln|\boldsymbol{\Sigma}_w| + \operatorname{tr}\Big[\boldsymbol{\Sigma}_w^{-1} S_{pw}\Big]\Big) + \sum_{s=1}^{S} N_s \cdot \Big[d_s + q_s^{zz} + 2q_s^{zy} + q_s^{yy}\Big], \tag{43}$$

where

$$d_s = \ln|\boldsymbol{\Sigma}_{zz}| + \ln|n_s \cdot \boldsymbol{\Sigma}_{b.z} + \boldsymbol{\Sigma}_w| \tag{44}$$

$$q_s^{yy} = n_s \cdot \mathbf{t}_s'(n_s\boldsymbol{\Sigma}_{b.z} + \boldsymbol{\Sigma}_w)^{-1}\mathbf{t}_s \tag{45}$$

$$q_s^{zy} = -n_s \cdot \mathbf{g}_s'(n_s\boldsymbol{\Sigma}_{b.z} + \boldsymbol{\Sigma}_w)^{-1}\mathbf{t}_s \tag{46}$$

$$q_s^{zz} = \boldsymbol{\delta}_{sz}'\boldsymbol{\Sigma}_{zz}^{-1}\boldsymbol{\delta}_{sz} + n_s \cdot \mathbf{g}_s'(n_s\boldsymbol{\Sigma}_{b.z} + \boldsymbol{\Sigma}_w)^{-1}\mathbf{g}_s. \tag{47}$$

N_s is the number of observations from all the clusters that have the same cluster size n_s. Further, $\boldsymbol{\delta}_s$, $\mathbf{t}_s$, and $\mathbf{g}_s$ are now based on the observations of all clusters that share the same cluster size. The actual R code that is used by `lavaan` to evaluate this objective function can be found in the function `lav_mvnorm_cluster_loglik_samplestats_2l()` in the file `lav_mvnorm_cluster.R`.

3.2. Missing Data

To handle missing values at both levels, it will be convenient to introduce selection matrices $\mathbf{Q}_{ji}$ and $\mathbf{G}_j$ for the first and second levels, respectively. The matrix $\mathbf{Q}_j$ is defined to be the collection of all $\mathbf{Q}_{ji}$ matrices from the same cluster:

$$\mathbf{Q}_j = \begin{pmatrix} \mathbf{Q}_{j1} \\ \mathbf{Q}_{j2} \\ \vdots \\ \mathbf{Q}_{jn_j} \end{pmatrix}.$$

It is now possible to express $\boldsymbol{\mu}_j$ and $\mathbf{V}_j$ as follows:

$$\boldsymbol{\mu}_j' = [(\mathbf{G}_j\boldsymbol{\mu}_z)', (\mathbf{Q}_{j1}\boldsymbol{\mu}_y)', (\mathbf{Q}_{j2}\boldsymbol{\mu}_y)', \ldots, (\mathbf{Q}_{jn_j}\boldsymbol{\mu}_y)']' \tag{48}$$

and

$$\mathbf{V}_j = \begin{bmatrix} \mathbf{G}_j\boldsymbol{\Sigma}_{zz}\mathbf{G}_j' & \mathbf{G}_j\boldsymbol{\Sigma}_{zy}\mathbf{Q}_j' \\ \mathbf{Q}_j\boldsymbol{\Sigma}_{yz}\mathbf{G}_j' & \mathbf{V}_{j(yy)} \end{bmatrix}. \tag{49}$$

The $\mathbf{V}_{j(yy)}$ matrix is defined as

$$\mathbf{V}_{j(yy)} = \mathbf{Q}_j\boldsymbol{\Sigma}_b\mathbf{Q}_j' + \bigoplus_{i=1}^{n_j} \mathbf{Q}_{ji}\boldsymbol{\Sigma}_w\mathbf{Q}_{ji}', \tag{50}$$

where the direct sum $\oplus$ operator is used to create a block diagonal matrix, with each block being equal to $\mathbf{Q}_{ji}\boldsymbol{\Sigma}_w\mathbf{Q}_{ji}'$. If data are complete in cluster j, then $\mathbf{G}_j$ and $\mathbf{Q}_{ji}$ become identity matrices. In that case, $\mathbf{Q}_j\boldsymbol{\Sigma}_b\mathbf{Q}_j'$ equals $\mathbf{1}_{n_j}\mathbf{1}_{n_j}' \otimes \boldsymbol{\Sigma}_b$, $\bigoplus_{i=1}^{n_j} \mathbf{Q}_{ji}\boldsymbol{\Sigma}_w\mathbf{Q}_{ji}'$ equals $\mathbf{I}_{n_j} \otimes \boldsymbol{\Sigma}_w$, $\mathbf{G}_j\boldsymbol{\Sigma}_{zy}\mathbf{Q}_j'$ equals $\mathbf{1}_{n_j}' \otimes \boldsymbol{\Sigma}_{zy}$, and $\mathbf{G}_j\boldsymbol{\Sigma}_{zz}\mathbf{G}_j'$ equals $\boldsymbol{\Sigma}_{zz}$. Because an expression for $\boldsymbol{\mu}_j$ and $\mathbf{V}_j$

was found, it becomes possible to estimate the model parameters using ML by minimizing the following objective function:

$$F_{ML2} = \sum_{j=1}^{J} \ln |\mathbf{V}_j(\boldsymbol{\theta})| + [\boldsymbol{v}_j^o - \boldsymbol{\mu}_j(\boldsymbol{\theta})]' \mathbf{V}_j(\boldsymbol{\theta})^{-1} [\boldsymbol{v}_j^o - \boldsymbol{\mu}_j(\boldsymbol{\theta})], \tag{51}$$

where $\boldsymbol{v}_j^o$ only contains the observed values from $\boldsymbol{v}_j$:

$$\boldsymbol{v}_j^{o\prime} = [(\mathbf{G}_j \boldsymbol{z}_j)', (\mathbf{Q}_{j1} \boldsymbol{y}_{j1})', (\mathbf{Q}_{j2} \boldsymbol{y}_{j2})', \ldots, (\mathbf{Q}_{jn_j} \boldsymbol{y}_{jn_j})']'. \tag{52}$$

The objective function can be written again as in (34). Just like in the complete case, the challenge is to rewrite this objective function so as to avoid the computation of the determinant and the inverse of the formidable matrix $\mathbf{V}_j$ directly. Below, I briefly describe two solutions, put forth by McDonald [2] and du Toit and du Toit [3], respectively.

3.3. The McDonald (1993) Solution

In Appendix B.1, two identities are shown to invert a partitioned matrix. McDonald starts from the second identity, and he develops an expression for $\mathbf{V}_j^{yy}$. Using my notation, the expression becomes

$$\mathbf{V}_j^{yy} = \left[\mathbf{Q}_j \boldsymbol{\Sigma}_b \mathbf{Q}_j' + \bigoplus_{i=1}^{n_j} \mathbf{Q}_{ji} \boldsymbol{\Sigma}_w \mathbf{Q}_{ji}' - \mathbf{Q}_j \boldsymbol{\Sigma}_{yz} \mathbf{G}_j' \boldsymbol{\Sigma}_j^{zz} \mathbf{G}_j \boldsymbol{\Sigma}_{zy} \mathbf{Q}_j' \right]^{-1}, \tag{53}$$

where $\boldsymbol{\Sigma}_j^{zz}$ is defined by

$$\boldsymbol{\Sigma}_j^{zz} = (\mathbf{G}_j \boldsymbol{\Sigma}_{zz} \mathbf{G}_j')^{-1}. \tag{54}$$

By moving the $\bigoplus$ term to the front and rearranging the remaining terms, the expression can be rewritten as

$$\mathbf{V}_j^{yy} = \left[\bigoplus_{i=1}^{n_j} \mathbf{Q}_{ji} \boldsymbol{\Sigma}_w \mathbf{Q}_{ji}' + \mathbf{Q}_j \left(\boldsymbol{\Sigma}_b - \boldsymbol{\Sigma}_{yz} \mathbf{G}_j' \boldsymbol{\Sigma}_j^{zz} \mathbf{G}_j \boldsymbol{\Sigma}_{zy} \right) \mathbf{Q}_j' \right]^{-1}. \tag{55}$$

To simplify the notation, I will define

$$\boldsymbol{\Lambda}_j = \bigoplus_{i=1}^{n_j} \mathbf{Q}_{ji} \boldsymbol{\Sigma}_w \mathbf{Q}_{ji}' \tag{56}$$

and

$$\boldsymbol{\Sigma}_{j(b.z)} = \left(\boldsymbol{\Sigma}_b - \boldsymbol{\Sigma}_{yz} \mathbf{G}_j' \boldsymbol{\Sigma}_j^{zz} \mathbf{G}_j \boldsymbol{\Sigma}_{zy} \right). \tag{57}$$

This means Equation (55) can be written in a more compact form:

$$\mathbf{V}_j^{yy} = \left[\boldsymbol{\Lambda}_j + \mathbf{Q}_j \boldsymbol{\Sigma}_{j(b.z)} \mathbf{Q}_j' \right]^{-1}. \tag{58}$$

This is a typical example wherein the Woodbury identity (see Appendix B.2) can be exploited. Importantly, McDonald assumes that both $\boldsymbol{\Sigma}_b$ and $\boldsymbol{\Sigma}_{j(b.z)}$ are positive definite. He then uses the classic Woodbury identity in (A25) to obtain the following expression for $\mathbf{V}_j^{yy}$:

$$\mathbf{V}_j^{yy} = \boldsymbol{\Lambda}_j^{-1} - \boldsymbol{\Lambda}_j^{-1} \mathbf{Q}_j \left(\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{Q}_j' \boldsymbol{\Lambda}_j^{-1} \mathbf{Q}_j \right)^{-1} \mathbf{Q}_j' \boldsymbol{\Lambda}_j^{-1}. \tag{59}$$

Note that $\boldsymbol{\Lambda}_j$ is a block diagonal matrix. Therefore, the inverse $\boldsymbol{\Lambda}_j^{-1}$ can be easily computed by inverting the individual blocks (see Appendix B.4). From this, an expression for the

quadratic form q_j^{yy} may be produced:

$$\begin{aligned} q_j^{yy} &= \delta_{jy}' \mathbf{V}_j^{yy} \delta_{jy} \\ &= \delta_{jy}' \mathbf{\Lambda}_j^{-1} \delta_{jy} - \delta_{jy}' \mathbf{\Lambda}_j^{-1} \mathbf{Q}_j \left(\mathbf{\Sigma}_{j(b.z)}^{-1} + \mathbf{Q}_j' \mathbf{\Lambda}_j^{-1} \mathbf{Q}_j \right)^{-1} \mathbf{Q}_j' \mathbf{\Lambda}_j^{-1} \delta_{jy} \\ &= \delta_{jy}' \mathbf{\Lambda}_j^{-1} \delta_{jy} - \mathbf{p}_j' \left(\mathbf{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j \right)^{-1} \mathbf{p}_j, \end{aligned} \tag{60}$$

where I have used the following notation:

$$\mathbf{p}_j = \mathbf{Q}_j' \mathbf{\Lambda}_j^{-1} \delta_{jy} \quad \text{and} \quad \mathbf{A}_j = \mathbf{Q}_j' \mathbf{\Lambda}_j^{-1} \mathbf{Q}_j. \tag{61}$$

For the q_j^{zz} term, $V_{j(zz)}$ must first be inverted. Using the second identity of a partitioned matrix once more results in

$$\mathbf{V}_j^{zz} = \mathbf{\Sigma}_j^{zz} + \mathbf{\Sigma}_j^{zz} \mathbf{G}_j \mathbf{\Sigma}_{zy} \mathbf{Q}_j' \left[\mathbf{\Lambda}_j + \mathbf{Q}_j \mathbf{\Sigma}_{j(b.z)} \mathbf{Q}_j' \right]^{-1} \mathbf{Q}_j \mathbf{\Sigma}_{yz} \mathbf{G}_j' \mathbf{\Sigma}_j^{zz}. \tag{62}$$

Note that the Woodbury push-through (right) identity (see Appendix B.2) may be applied to the example $\left[\mathbf{\Lambda}_j + \mathbf{Q}_j \mathbf{\Sigma}_{j(b.z)} \mathbf{Q}_j' \right]^{-1} \mathbf{Q}_j$:

$$\left[\mathbf{\Lambda}_j + \mathbf{Q}_j \mathbf{\Sigma}_{j(b.z)} \mathbf{Q}_j' \right]^{-1} \mathbf{Q}_j = \mathbf{\Lambda}_j^{-1} \mathbf{Q}_j \left(\mathbf{\Sigma}_{j(b.z)}^{-1} + \mathbf{Q}_j' \mathbf{\Lambda}_j^{-1} \mathbf{Q}_j \right)^{-1} \mathbf{\Sigma}_{j(b.z)}^{-1}. \tag{63}$$

Plugging this into Equation (62) results in

$$\mathbf{V}_j^{zz} = \mathbf{\Sigma}_j^{zz} + \mathbf{\Sigma}_j^{zz} \mathbf{G}_j \mathbf{\Sigma}_{zy} \mathbf{Q}_j' \mathbf{\Lambda}_j^{-1} \mathbf{Q}_j \left(\mathbf{\Sigma}_{j(b.z)}^{-1} + \mathbf{Q}_j' \mathbf{\Lambda}_j^{-1} \mathbf{Q}_j \right)^{-1} \mathbf{\Sigma}_{j(b.z)}^{-1} \mathbf{\Sigma}_{yz} \mathbf{G}_j' \mathbf{\Sigma}_j^{zz} \tag{64}$$

$$= \mathbf{\Sigma}_j^{zz} + \mathbf{\Sigma}_j^{zz} \mathbf{G}_j \mathbf{\Sigma}_{zy} \mathbf{A}_j \left(\mathbf{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j \right)^{-1} \mathbf{\Sigma}_{j(b.z)}^{-1} \mathbf{\Sigma}_{yz} \mathbf{G}_j' \mathbf{\Sigma}_j^{zz}. \tag{65}$$

The quadratic form q_j^{zz} can then be written as

$$\begin{aligned} q_j^{zz} &= \delta_{jz}' \mathbf{V}_j^{zz} \delta_{jz} \\ &= \delta_{jz}' \mathbf{\Sigma}_j^{zz} \delta_{jz} + \delta_{jz}' \mathbf{\Sigma}_j^{zz} \mathbf{G}_j \mathbf{\Sigma}_{zy} \mathbf{A}_j \left(\mathbf{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j \right)^{-1} \mathbf{\Sigma}_{j(b.z)}^{-1} \mathbf{\Sigma}_{yz} \mathbf{G}_j' \mathbf{\Sigma}_j^{zz} \delta_{jz} \\ &= \delta_{jz}' \mathbf{\Sigma}_j^{zz} \delta_{jz} + \mathbf{g}_j' \mathbf{A}_j \left(\mathbf{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j \right)^{-1} \mathbf{\Sigma}_{j(b.z)}^{-1} \mathbf{g}_j, \end{aligned} \tag{66}$$

where the following notation has been used:

$$\mathbf{g}_j = \mathbf{\Sigma}_{yz} \mathbf{G}_j' \mathbf{\Sigma}_j^{zz} \delta_{jz}. \tag{67}$$

Finally, instead of q_j^{zy}, an expression for q_j^{yz} must be found; the inverse of $\mathbf{V}_{j(yz)}$ can be written as

$$\mathbf{V}_j^{yz} = -\left[\mathbf{\Lambda}_j + \mathbf{Q}_j \mathbf{\Sigma}_{j(b.z)} \mathbf{Q}_j' \right]^{-1} \mathbf{Q}_j \, \mathbf{\Sigma}_{yz} \mathbf{G}_j' \, \mathbf{\Sigma}_j^{zz}. \tag{68}$$

Again, using the identity from Equation (A27) results in

$$\begin{aligned} \mathbf{V}_j^{yz} &= -\mathbf{\Lambda}_j^{-1} \mathbf{Q}_j \left(\mathbf{\Sigma}_{j(b.z)}^{-1} + \mathbf{Q}_j' \mathbf{\Lambda}_j^{-1} \mathbf{Q}_j \right)^{-1} \mathbf{\Sigma}_{j(b.z)}^{-1} \mathbf{\Sigma}_{yz} \mathbf{G}_j' \mathbf{\Sigma}_j^{zz} \\ &= -\mathbf{\Lambda}_j^{-1} \mathbf{Q}_j \left(\mathbf{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j \right)^{-1} \mathbf{\Sigma}_{j(b.z)}^{-1} \mathbf{\Sigma}_{yz} \mathbf{G}_j' \, \mathbf{\Sigma}_j^{zz}. \end{aligned} \tag{69}$$

The quadratic form q_j^{zy} is therefore given by

$$\begin{aligned} q_j^{yz} &= \boldsymbol{\delta}_{jy}' \mathbf{V}_j^{yz} \boldsymbol{\delta}_{jz} \\ &= -\boldsymbol{\delta}_{jy}' \boldsymbol{\Lambda}_j^{-1} \mathbf{Q}_j \left(\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j \right)^{-1} \boldsymbol{\Sigma}_{j(b.z)}^{-1} \boldsymbol{\Sigma}_{yz} \mathbf{G}_j' \boldsymbol{\Sigma}_j^{zz} \boldsymbol{\delta}_{jz} \\ &= -\mathbf{p}_j' \left(\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j \right)^{-1} \boldsymbol{\Sigma}_{j(b.z)}^{-1} \mathbf{g}_j. \end{aligned} \tag{70}$$

For the determinant of $\mathbf{V}_j$, the formula for the determinant of a partitioned matrix in (A24) can be used to write $|\mathbf{V}_j|$ as

$$\begin{aligned} |\mathbf{V}_j| &= |\mathbf{V}_{j(zz)}| \cdot |\mathbf{V}_{j(yy)} - \mathbf{V}_{j(yz)} \mathbf{V}_j^{zz} \mathbf{V}_{j(zy)}| \\ &= |\mathbf{G}_j \boldsymbol{\Sigma}_{zz} \mathbf{G}_j'| \cdot |\mathbf{Q}_j \boldsymbol{\Sigma}_b \mathbf{Q}_j' + \bigoplus_{i=1}^{n_j} \mathbf{Q}_{ji} \boldsymbol{\Sigma}_w \mathbf{Q}_{ji}' - \mathbf{Q}_j \boldsymbol{\Sigma}_{yz} \mathbf{G}_j' \boldsymbol{\Sigma}_j^{zz} \mathbf{G}_j \boldsymbol{\Sigma}_{zy} \mathbf{Q}_j'| \\ &= |\mathbf{G}_j \boldsymbol{\Sigma}_{zz} \mathbf{G}_j'| \cdot |\boldsymbol{\Lambda}_j + \mathbf{Q}_j \boldsymbol{\Sigma}_{j(b.z)} \mathbf{Q}_j'|. \end{aligned} \tag{71}$$

The first determinant is simple because the largest dimension of $\mathbf{G}_j \boldsymbol{\Sigma}_{zz} \mathbf{G}_j'$ is $K \times K$ (or smaller, if missing values are present in $\boldsymbol{z}_j$). The argument of the second determinant, however, is still too large to be practical. McDonald [2] suggests a recursive formula, where the second determinant is computed in an incremental fashion. However, here I will give an explicit solution, making use of the determinant identity in (A31):

$$|\mathbf{V}_j| = |\mathbf{G}_j \boldsymbol{\Sigma}_{zz} \mathbf{G}_j'| \cdot |\boldsymbol{\Lambda}_j| \cdot |\boldsymbol{\Sigma}_{j(b.z)}| \cdot |\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{Q}_j' \boldsymbol{\Lambda}_j^{-1} \mathbf{Q}_j| \tag{72}$$

$$= |\mathbf{G}_j \boldsymbol{\Sigma}_{zz} \mathbf{G}_j'| \cdot |\boldsymbol{\Lambda}_j| \cdot |\boldsymbol{\Sigma}_{j(b.z)}| \cdot |\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j|. \tag{73}$$

To summarize and for future reference, the final expressions for the determinant and the three quadratic forms are as follows:

$$|\mathbf{V}_j| = |\mathbf{G}_j \boldsymbol{\Sigma}_{zz} \mathbf{G}_j'| \cdot |\boldsymbol{\Lambda}_j| \cdot |\boldsymbol{\Sigma}_{j(b.z)}| \cdot |\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j| \tag{74}$$

$$q_j^{yy} = \boldsymbol{\delta}_{jy}' \boldsymbol{\Lambda}_j^{-1} \boldsymbol{\delta}_{jy} - \mathbf{p}_j' \left(\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j \right)^{-1} \mathbf{p}_j \tag{75}$$

$$q_j^{yz} = -\mathbf{p}_j' \left(\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j \right)^{-1} \boldsymbol{\Sigma}_{j(b.z)}^{-1} \mathbf{g}_j \tag{76}$$

$$q_j^{zz} = \boldsymbol{\delta}_{jz}' \boldsymbol{\Sigma}_j^{zz} \boldsymbol{\delta}_{jz} + \mathbf{g}_j' \mathbf{A}_j \left(\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j \right)^{-1} \boldsymbol{\Sigma}_{j(b.z)}^{-1} \mathbf{g}_j. \tag{77}$$

To make the connection with the expressions for the complete data setting, note that when data are complete, $\boldsymbol{\Sigma}_{j(b.z)} = \boldsymbol{\Sigma}_{b.z}$, $\boldsymbol{\Sigma}_j^{zz} = \boldsymbol{\Sigma}_{zz}^{-1}$, $\mathbf{A}_j = n_j \boldsymbol{\Sigma}_w^{-1}$, $\mathbf{p}_j = n_j \boldsymbol{\Sigma}_w^{-1} (\bar{\mathbf{y}}_j - \boldsymbol{\mu}_y) = n_j \boldsymbol{\Sigma}_w^{-1} \mathbf{t}_j$, and $\mathbf{g}_j = \boldsymbol{\Sigma}_{yz} \boldsymbol{\Sigma}_{zz}^{-1} \boldsymbol{\delta}_{jz}$. Furthermore, when data are complete, both $\mathbf{p}_j$ and $\mathbf{A}_j$ contain $\boldsymbol{\Sigma}_w^{-1}$. Therefore, it can be verified (using the identity $(\mathbf{AB})^{-1} = \mathbf{B}^{-1} \mathbf{A}^{-1}$ for nonsingular $\mathbf{AB}$ and the classic Woodbury identity) that

$$\boldsymbol{\Sigma}_w^{-1} \left(\boldsymbol{\Sigma}_{(b.z)}^{-1} + n_j \boldsymbol{\Sigma}_w^{-1} \right)^{-1} \boldsymbol{\Sigma}_{(b.z)}^{-1} = \left(\boldsymbol{\Sigma}_w + n_j \boldsymbol{\Sigma}_{(b.z)} \right)^{-1}$$

and

$$n_j \boldsymbol{\Sigma}_w^{-1} \left(\boldsymbol{\Sigma}_{(b.z)}^{-1} + n_j \boldsymbol{\Sigma}_w^{-1} \right)^{-1} \boldsymbol{\Sigma}_w^{-1} = \boldsymbol{\Sigma}_w^{-1} - \left(\boldsymbol{\Sigma}_w + n_j \boldsymbol{\Sigma}_{(b.z)} \right)^{-1}.$$

3.4. The du Toit and du Toit (2008) Solution

To invert the partitioned matrix $\mathbf{V}_j$, McDonald [2] made use of the second identity in (A21) to develop the formulas. By contrast, du Toit and du Toit [3] start from the first identity in (A20). Because the rest of the paper will focus on the McDonald solution, the details are given in Appendix A. Below are the final expressions for the determinant

and the three quadratic forms:

$$|\mathbf{V}_j| = |\mathbf{G}_j\boldsymbol{\Sigma}_{j(z.y)}\mathbf{G}'| \cdot |\boldsymbol{\Lambda}_j| \cdot |\boldsymbol{\Sigma}_b| \cdot |\boldsymbol{\Sigma}_b^{-1} + \mathbf{A}_j| \tag{78}$$

$$q_j^{yy} = \boldsymbol{\delta}_{jy}'\boldsymbol{\Lambda}_j^{-1}\boldsymbol{\delta}_{jy} - \mathbf{p}_j'\left[\mathbf{B}_j - \mathbf{B}_j\boldsymbol{\Sigma}_b^{-1}\boldsymbol{\Sigma}_{yz}\mathbf{G}_j'\mathbf{V}_j^{zz}\mathbf{G}_j\boldsymbol{\Sigma}_{zy}\,\boldsymbol{\Sigma}_b^{-1}\mathbf{B}_j\right]\mathbf{p}_j \tag{79}$$

$$q_j^{yz} = -\mathbf{p}_j'\left(\boldsymbol{\Sigma}_b^{-1} + \mathbf{A}_j\right)^{-1}\boldsymbol{\Sigma}_b^{-1}\,\tilde{\mathbf{g}}_j \tag{80}$$

$$q_j^{zz} = \boldsymbol{\delta}_{jz}'\mathbf{V}_j^{zz}\boldsymbol{\delta}_{jz}, \tag{81}$$

where $\mathbf{B}_j = \left(\boldsymbol{\Sigma}_b^{-1} + \mathbf{A}_j\right)^{-1}$ and $\tilde{\mathbf{g}}_j = \boldsymbol{\Sigma}_{yz}\mathbf{G}_j'\mathbf{V}_j^{zz}$.

3.5. An Extension to Allow Σ_b to Be Singular

An important limitation of both approaches is that they both assume $\boldsymbol{\Sigma}_b$ and (in the case of McDonald) $\boldsymbol{\Sigma}_{j(b.z)}$ are nonsingular and therefore are invertible. However, sometimes rows and columns in $\boldsymbol{\Sigma}_b$ are all zero (e.g., if they correspond to within-only variables), or restrictions in the model have been imposed that result in a singular $\boldsymbol{\Sigma}_b$. For example, consider a one-factor model at the between level. If the residual variances are fixed to zero, then $\boldsymbol{\Sigma}_b$ will be singular. This is a common setting as zero-residual variances are sometimes used in cluster-invariant measurement models [13,14]. In other settings, $\boldsymbol{\Sigma}_b$ may be near singular, potentially leading to numerical instabilities. For all these reasons, it would be useful if the previous solutions could be adapted to handle the case where $\boldsymbol{\Sigma}_b$ is singular. The solution is simply to replace the classic Woodbury identity in (A25) by the more general version in (A26). I will illustrate how this changes the final expressions for the McDonald solution. The logic for the du Toit and du Toit approach is similar.

Consider again the expression for $\mathbf{V}_j^{yy}$ as it was presented in Equation (58):

$$\mathbf{V}_j^{yy} = \left[\boldsymbol{\Lambda}_j + \mathbf{Q}_j\boldsymbol{\Sigma}_{j(b.z)}\mathbf{Q}_j'\right]^{-1}. \tag{82}$$

Applying the classic Woodbury identity results in an expression (59) that contains the inverse of $\boldsymbol{\Sigma}_{j(b.z)}$. However, if the more general identity (A26) is used, then (59) may be rewritten as follows:

$$\begin{aligned}\mathbf{V}_j^{yy} &= \boldsymbol{\Lambda}_j^{-1} - \boldsymbol{\Lambda}_j^{-1}\mathbf{Q}_j\left(\mathbf{I}_P + \boldsymbol{\Sigma}_{j(b.z)}\mathbf{Q}_j'\boldsymbol{\Lambda}_j^{-1}\mathbf{Q}_j\right)^{-1}\boldsymbol{\Sigma}_{j(b.z)}\mathbf{Q}_j'\,\boldsymbol{\Lambda}_j^{-1} \\ &= \boldsymbol{\Lambda}_j^{-1} - \boldsymbol{\Lambda}_j^{-1}\mathbf{Q}_j\left(\mathbf{I}_P + \boldsymbol{\Sigma}_{j(b.z)}\mathbf{A}_j\right)^{-1}\boldsymbol{\Sigma}_{j(b.z)}\mathbf{Q}_j'\,\boldsymbol{\Lambda}_j^{-1}.\end{aligned} \tag{83}$$

In this expression, $\boldsymbol{\Sigma}_{j(b.z)}$ does not need to be inverted. On the other hand, if $\boldsymbol{\Sigma}_{j(b.z)}$ *is* invertible, then it follows that:

$$\begin{aligned}\left(\mathbf{I}_P + \boldsymbol{\Sigma}_{j(b.z)}\mathbf{A}_j\right)^{-1}\boldsymbol{\Sigma}_{j(b.z)} &= \left(\boldsymbol{\Sigma}_{j(b.z)}\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \boldsymbol{\Sigma}_{j(b.z)}\mathbf{A}_j\right)^{-1}\boldsymbol{\Sigma}_{j(b.z)} \\ &= \left(\boldsymbol{\Sigma}_{j(b.z)}(\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j)\right)^{-1}\boldsymbol{\Sigma}_{j(b.z)} \\ &= \left(\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j\right)^{-1}\boldsymbol{\Sigma}_{j(b.z)}^{-1}\boldsymbol{\Sigma}_{j(b.z)} \\ &= \left(\boldsymbol{\Sigma}_{j(b.z)}^{-1} + \mathbf{A}_j\right)^{-1}.\end{aligned} \tag{84}$$

The term on the right-hand side is used in the McDonald solution, but if it is replaced by the term on the left-hand side, the following alternative expressions for the quadratic forms emerge:

$$q_j^{yy} = \delta_{jy}' \mathbf{\Lambda}_j^{-1} \delta_{jy} - \mathbf{p}_j' \left(\mathbf{I}_P + \mathbf{\Sigma}_{j(b.z)} \mathbf{A}_j \right)^{-1} \mathbf{\Sigma}_{j(b.z)} \, \mathbf{p}_j \tag{85}$$

$$q_j^{yz} = -\mathbf{p}_j' \left(\mathbf{I}_P + \mathbf{\Sigma}_{j(b.z)} \mathbf{A}_j \right)^{-1} \mathbf{g}_j \tag{86}$$

$$q_j^{zz} = \delta_{jz}' \mathbf{\Sigma}_j^{zz} \delta_{jz} + \mathbf{g}_j' \, \mathbf{A}_j \left(\mathbf{I}_P + \mathbf{\Sigma}_{j(b.z)} \mathbf{A}_j \right)^{-1} \mathbf{g}_j. \tag{87}$$

None of these expressions require $\mathbf{\Sigma}_b$ or $\mathbf{\Sigma}_{j(b.z)}$ to be nonsingular, making them much more useful in practice. The last task is to find an expression for the determinant of $\mathbf{V}_j$ that can be used when $\mathbf{\Sigma}_b$ or $\mathbf{\Sigma}_{j(b.z)}$ is singular. Here, I will make use of the determinant identity in (A32), which results in the following:

$$\begin{aligned} |\mathbf{\Lambda}_j + \mathbf{Q}_j \mathbf{\Sigma}_{j(b.z)} \mathbf{Q}_j'| &= |\mathbf{\Lambda}_j| \cdot |\mathbf{I}_P + \mathbf{Q}_j' \mathbf{\Lambda}_j^{-1} \mathbf{Q}_j \mathbf{\Sigma}_{j(b.z)}| \\ &= |\mathbf{\Lambda}_j| \cdot |\mathbf{I}_P + \mathbf{A}_j \mathbf{\Sigma}_{j(b.z)}|. \end{aligned} \tag{88}$$

Thus, the determinant of $\mathbf{V}_j$ can be written as

$$|\mathbf{V}_j| = |\mathbf{G}_j \mathbf{\Sigma}_{zz} \mathbf{G}_j'| \cdot |\mathbf{\Lambda}_j| \cdot |\mathbf{I}_P + \mathbf{A}_j \mathbf{\Sigma}_{j(b.z)}|, \tag{89}$$

which again does not require the inversion of $\mathbf{\Sigma}_{j(b.z)}$.

4. Implementation in R Code

Formulas are useful to study, but for some readers, it may be more rewarding to study the implementation of these formulas in source code. Appendix C contains the source code of several short R scripts. The first script (`main.R`) contains the dataset and the example model and runs the code in the other scripts. Following this are five R scripts that each illustrate a different approach to evaluate the objective function in (51). Starting (`objective1.R`) with a naive implementation, the first step is to compute the determinant and the inverse of the full $\mathbf{V}_j$ matrix for each cluster. This script allows for the introduction of various ingredients in a simple way. In `objective2.R`, I will implement the formulas at the end of Section 3.5. This version is faster (see Table 1) than the naive version—as expected.

If the programmer feels that this implementation is still too slow, it may be tempting to rewrite the code in a compiled language, such as `C/C++` or `FORTRAN`. Without any effort (apart from the programming effort), this will result in a much faster code. I will attempt to improve the R code itself to allow the code to run even faster. In the `objective3.R` script, I will avoid the explicit construction of selection matrices and will include some minor improvements. This will further decrease the computation time. In the `objective4.R` script, I will make use of missing patterns within each cluster. Unfortunately, this hardly leads to any improvement because the cluster sizes are rather modest in my example dataset. In the `objective5.R` script, I will use missing patterns as found in the entire dataset. Although this requires much more housekeeping, the gain in computation time is significant. An overview of the computation time for the five objective functions is presented in Table 1. In the second column, the average computation time (over 100 replications) is shown in milliseconds. In the last column, I take the second objective function as the baseline and show the time differences in percentages.

Table 1. Timing results for the five objective functions. The second column contains the average computation time (over 100 replications) in milliseconds. The third column shows percentages, where the second objective function serves as the baseline.

Version	Absolute Time (ms)	Relative Time
Objective 1	3875.32	992.90
Objective 2	390.30	100.00
Objective 3	85.89	22.01
Objective 4	82.52	21.14
Objective 5	20.18	5.17

The R scripts are based on the McDonald solution, taking into account the extension for singular $\boldsymbol{\Sigma}_b$ matrices. Equivalent R scripts (not shown here) have been written for the du Toit and du Toit solution. However, the shortcut that is used in `objective5.R` does not seem to apply for the du Toit and du Toit solution. For this reason, I only report the scripts based on the McDonald solution. In what follows, I will briefly discuss each file, guiding readers through the source code.

4.1. The Main.R File

To explore the R code, readers should start with the file `main.R`, which is listed in Appendix C.1. The script starts with reading in the dataset (Lines 4–9). The (artificial) dataset contains 2500 observations of 20 variables, clustered in $J = 200$ clusters. The cluster sizes are 5, 10, 15, and 20, with 50 clusters of each size. The y1–y10 and x1–x3 variables are measured at the within level, whereas the z1–z4 and w1–w3 variables are measured at the between level. Roughly 10% of the data are missing at both levels. Then, I specify a two-level model using `lavaan` syntax (Lines 10–38). In this model, the y1–y6 variables are split into a within and a between part. The y7–y10 and x1–x3 variables are within-only variables. The z1–z4 and w1–w3 variables are between-only variables. Next, I call the `sem()` function (Lines 44–45), but with the `do.fit = FALSE` argument. This implies that only the model and the dataset are processed, without starting the optimization. In fact, I will simply keep the starting values and only use the resulting `dummy.fit` object to extract the model-implied statistics and information about the dataset. There are $T = 83$ free parameters in this model. The next lines in the script extract some useful information from the `dummy.fit` object. The `lavmodel` object (Line 52) contains the internal representation of the model, including all model matrices. `Lp` (Line 53) contains various information about the clusters: the number of clusters (Line 64), cluster size (Line 65), cluster index (Line 66), and an index of the between-level variables (Line 67). In Line 54, `Y1` contains the raw dataset as a 2500×20 matrix, and `Y2` (Line 55) contains the cluster aggregated data as a 200×20 matrix. The `lav_model_implied()` function (Line 57) is used to compute the model-implied statistics for this model—given the current set of parameters. This returns a list (`implied`), which contains a mean vector and a variance–covariance matrix per level. These implied statistics are then further broken down (Line 61) into smaller pieces: $\boldsymbol{\mu}_y$, $\boldsymbol{\mu}_z$, $\boldsymbol{\Sigma}_w$, $\boldsymbol{\Sigma}_b$, $\boldsymbol{\Sigma}_{zz}$, and $\boldsymbol{\Sigma}_{yz}$. These vectors and matrices will always be the input for the objective functions. For each of the five objective functions, I first read in the source code (e.g., Line 71), and then I run the function to compute the value of the objective function (e.g., Lines 72–74). Sometimes, I precompute a few quantities that are needed in the objective function. For example, for `objective4.R`, a list (MP) is needed that contains information about the missing patterns for every cluster (Lines 91–95). For `objective5.R`, information is needed about the missing patterns for the complete dataset at both levels (Lines 104–105). Finally, I use the `microbenchmark` package to benchmark the five different functions (Lines 112–129). The results are shown in Table 1.

4.2. The Objective1.R File

The first script for the objective function (`objective1.R`) is listed in Appendix C.2. This script is based on Section 3.2. Apart from the transpose of $\boldsymbol{\Sigma}_{yz}$ (Line 9), no preparations

are needed. Everything happens within the main `for-loop` (Lines 12–81), which runs over all clusters. For each cluster, the contribution to the (log)likelihood is computed and stored in a vector `loglik` (Line 11). If the function argument `loglik.` is `FALSE` (the default), then the constant is omitted and the log-likelihood contribution is multiplied by a factor -2 (minus two) (Lines 74–79). The final value of the objective function is the sum over all cluster contributions (Line 83). For a given cluster (j), the cluster size n_j must first be noted (Line 15). Then, any variables at the between level are ascertained (Line 17). In my example, this will always be the case. Then the data for this cluster are collected in an $n_j \times P$ dimensional matrix `Yw.j` for the within level (Line 18) and in a $K \times 1$ vector `z.j` for the between level (Line 19). For the data vector at the between level, the $\mathbf{G}_j$ selection matrix (Lines 21–26) is created to keep track of the missing values in `z.j`. If the data are complete, $\mathbf{G}_j$ will be an identity matrix. The missing values in `z.j` are removed (Line 27). Similarly, for the within level, a selection matrix $\mathbf{Q}_{ji}$ is created (Lines 34–42) for every observation in this cluster. A corresponding `W.ji` matrix (Line 43) is also created, which corresponds to $\mathbf{Q}_{ji}\boldsymbol{\Sigma}_w\mathbf{Q}_{ji}'$, because these matrices are needed to construct the block diagonal matrix $\bigoplus_{i=1}^{n_j}\mathbf{Q}_{ji}\boldsymbol{\Sigma}_w\mathbf{Q}_{ji}'$ (using the `lav_matrix_bdiag()` function, Line 54). All the $\mathbf{Q}_{ji}$ matrices for this cluster are then collected in a $P_j \times P$ dimensional matrix $\mathbf{Q}_j$ (Line 47). It is now possible to construct the variance–covariance matrix $\mathbf{V}_j$ for this cluster. First, the four parts (`V.zz`, `V.zy`, `V.yz`, and `V.yy` (Lines 51–54)) are constructed, and then the full `V.j` matrix (Lines 56–57) is assembled. The `est.j` vector contains the model-implied means for both the between and the within level (Lines 58–59). The `obs.j` vector contains the observed data for both levels (Lines 60–61). Both vectors contain P_j elements. The difference between these two vectors is stored in `delta.j` (Line 71). Perhaps the biggest task in this script (in terms of computation time) is the computation of the determinant and the inverse of the $P_j \times P_j$ dimensional matrix $\mathbf{V}_j$ (Lines 68–69). Once the inverse has been computed, it is possible to compute the quadratic form $\boldsymbol{\delta}_j'\mathbf{V}_j\boldsymbol{\delta}_j$, storing the (scalar) result in `q.j` (Line 72). Finally, the cluster contribution to the objective value is stored in `loglik`, and the next cluster can then be addressed. After repeating this $J = 200$ times, the final value for the objective function is stored in `out` and returned to the caller.

4.3. The Objective2.R File

The second script for the objective function (`objective2.R`), which is listed in Appendix C.3, is based on Section 3.3 and the extension in Section 3.5. The first part of the script (Lines 9–49) is identical to the `objective1.R` script. The only addition is the creation of the `A.ji` matrices, which are defined as $\mathbf{A}_{ji} = \mathbf{Q}_{ji}'(\mathbf{Q}_{ji}\boldsymbol{\Sigma}_w\mathbf{Q}_{ji}')^{-1}\mathbf{Q}_{ji}$. The sum of these n_j matrices is then stored in the matrix $\mathbf{A}_j$, which corresponds to $\mathbf{Q}_j'\boldsymbol{\Lambda}_j^{-1}\mathbf{Q}_j$—a matrix that plays a crucial role in this script. The name for $\boldsymbol{\Lambda}_j$ in this script is `bdiag_sigma.w.j`, indicating that this is a block diagonal matrix, where each block is based on a selection of rows and columns of the $\boldsymbol{\Sigma}_w$ matrix (Line 52). The inverse ($\boldsymbol{\Lambda}_j^{-1}$) is called `bdiag_sigma.w.j.inv` (Line 53). Next, we compute $\boldsymbol{\Sigma}_j^{zz}$ (Line 57), $\boldsymbol{\Sigma}_{j(b.z)}$ (Lines 58–60), and `IBZA.j`, which is defined as $\mathbf{I}_P + \boldsymbol{\Sigma}_{j(b.z)}\mathbf{A}_j$ (Line 64) and is perhaps the most important matrix of this script. The model-implied means (`est.j`) and observed data (`obs.j`) are defined just like in the first script (Lines 70–74). However, the difference is split between a z part (`delta.z`) and a y part (`delta.y`) (Lines 76–81). Finally, $\mathbf{p}_j$ (Line 82) and $\mathbf{g}_j$ (Line 83) are defined. All the ingredients are now in place to compute the quadratic forms `q.zz`, `q.zy`, `q.yy`, and eventually `q.j` (Lines 85–93). The `drop()` function is used to make sure the result is a scalar, not a 1×1 dimensional matrix. Some care is needed for clusters where all elements in `z.j` are missing. In that case, $\mathbf{G}_j$ is empty (zero rows), and only `q.y` is used to form `q.j` (Lines 96–101). To compute the (log) determinant, the (log) determinant of $\boldsymbol{\Lambda}_j$ is first computed by taking the sum of the n_j (log) determinants for each `W.ji` block (Line 106). Then the (log) determinant of the `IBZA.j` matrix (Line 107) is computed, followed by the (log) determinant of $\mathbf{G}_j\boldsymbol{\Sigma}_{zz}\mathbf{G}_j'$ (Line 108). The sum of these three log determinants provides the log determinant of the full $\mathbf{V}_j$ matrix (Line 109). The log-likelihood contribution for

this cluster is stored in `loglik` (Lines 116–121), and this completes the computations for this cluster. After $J = 200$ repetitions, the final value (`out`) is obtained and returned to the caller.

4.4. The Objective3.R File

The `objective2.R` script runs faster than the `objective1.R` script, but there is still much room for improvement. Perhaps the biggest problem in `objective2.R` is that `W.ji` must be computed and inverted (see Line 45 in `objective2.R`) for every observation. In addition, the construction and use of the selection matrices ($\mathbf{Q}_{ji}$, $\mathbf{G}_j$) are not practical and (as I shall demonstrate) unnecessary. In the third script for the objective function (listed in Appendix C.4), the global structure of the `objective2.R` script is kept intact, but I try to improve the code to make it more efficient.

Recall that `W.ji` is just $\mathbf{\Sigma}_w$, but the rows and columns that correspond to the missing values for this observation have been removed. In this script, I precompute the (log) determinant and the inverse of the full $\mathbf{\Sigma}_w$ matrix (Lines 12–13). If an observation has missing values, I can "update" the inverse (and determinant) (Lines 52–57) and use this information to incrementally construct `W.logdet.j` (Line 58), $\mathbf{A}_j$ (Lines 59–60), and $\mathbf{p}_j$ (Line 63) while the individual observations of this cluster are processed. In addition, I update q.yy.a (Line 64), which is the part of q.yy that only depends on within-level information. When an observation is complete, there is no need to "update" the inverse of $\mathbf{\Sigma}_w$, and I can compute `W.logdet.j` (Line 67), $\mathbf{A}_j$ (Line 68), $\mathbf{p}_j$ (Line 71), and q.yy.a (Line 72) immediately using the precomputed (log) determinant and the inverse of $\mathbf{\Sigma}_w$. In this script, I no longer explicitly use the selection matrices $\mathbf{Q}_{ji}$. Instead, I keep track of the missing-value indices (`na.idx`) and use these to remove the rows and columns of the matrices when needed. For the between part, I use a similar strategy. I precompute the (log) determinant and the inverse of $\mathbf{\Sigma}_{zz}$ (Lines 23–24) and $\mathbf{\Sigma}_{b.z}$ (Line 26) for the complete case. Then, for a specific cluster, I make a distinction between three scenarios: (1) all between values (of `obs.z`) are missing (Lines 87–88), (2) at least one value is missing (but not all) (Lines 90–101), or (3) no values are missing (Lines 103–108). In the latter case, the complete data statistics can be used immediately. When some values are missing, the (log) determinant and the inverse of $\mathbf{\Sigma}_{zz}$ (Lines 93–97) must be "updated." The selection matrix $\mathbf{G}_j$ is no longer used. The key matrix `IBZA.j` is now constructed in a slightly fancier way (Lines 113–114) in order to avoid the construction of a diagonal matrix (for every cluster). Other improvements are the construction of the quadratic forms (Lines 126–129), where matrix multiplications are avoided and maximally rely on scalar multiplications. The quadratic form for the y part is split into two parts (`q.yy.a` and `q.yy.b`), where the former was already computed when the individual observations of this cluster (Line 64 or 72) were processed. In a similar fashion, the z part is split into two parts (`q.zz.a` and `q.zz.b`), where the former is based on the raw data (`delta.z`), and the latter is based on the $\mathbf{g}_j$ vector for this cluster. Because I have already computed all the needed (log) determinants along the way, I can immediately compute `V.j.logdet` (Line 143). The remainder of the script is identical to the `objective2.R script`.

4.5. The Objective4.R File

In the final paragraph of the last Appendix (12.B.6) of their book chapter, du Toit and du Toit [3] write: "One could also compute the patterns of missingness within each level-2 unit ..." In the `objective4.R` script, listed in Appendix C.5, I followed this suggestion. First, the missing pattern information for each cluster will be precomputed. This is done in the `main.R` script (Lines 91–95), where we construct a list (`MP`) with missing pattern information for each cluster. Instead of running over all the observations within a cluster, the missing patterns within a cluster (Line 46) are run over. For each pattern, the (log) determinant and the inverse of $\mathbf{\Sigma}_w$ (Lines 56–60) are "updated," and the quantities `W.logdet.j`, **A.j**, $\mathbf{p}_j$, and `q.yy.z` are computed (Lines 61–68). Note that `freq` is used to account for the number of observations that belong to this pattern. Other small improvements in this

script are the precentering of `Y1w` (Lines 17–18) and `Z` (Lines 23–24), which facilitate the construction of the vector of observed data for each cluster (Line 65 or 75 and Line 95 or 108). The computation of q.zz.a now happens in the between section, although it is nothing more than a cosmetic change. Finally, the `NY` and `NZ` scalars keep track of the number of nonmissing values in the cluster (at the within and between level, respectively). They are used to compute P_j, which is only needed if `loglik. = TRUE` (Line 146). Based on Table 1, it would seem that this script is not faster than the previous script (`objective3.R`)—at least in my example. The reason is that the cluster sizes are rather small in my dataset, and the number of missing patterns is often not much smaller than the number of observations within a cluster. However, even for larger cluster sizes, this script is not much faster than the previous one.

4.6. The Objective5.R File

In order to further decrease the computation time, it is necessary to decrease the number of matrix inversions. In the `objective4.R` script, $\mathbf{\Sigma}_w$ must be inverted and the determinant must be computed for every missing pattern in each cluster. Across clusters, the same missing pattern often emerges. In this script, listed in Appendix C.6, I sought to invert $\mathbf{\Sigma}_w$ only once for each missing pattern in the entire level-1 dataset. In addition, missing patterns will be used for the between data. The price to pay is that more housekeeping becomes necessary. At the between level, a list `SIGMA.B.Z` (Line 39) is created, where the $\mathbf{\Sigma}_{j(b.z)}$ matrix is filled in for each missing pattern at the between level (Lines 62–63). At the same time, the $J \times 1$ vector `ZPAT2J` keeps track of the missing pattern that applies for each cluster. Later, when the `IBZA.j` matrix must be constructed for a given cluster, the correct $\mathbf{\Sigma}_{j(b.z)}$ matrix can then be "selected" from the list in `SIGMA.B.Z` (Line 145). Similarly, a $J \times P$ dimensional matrix `GJ` is created, where the $\mathbf{g}_j$ vectors are filled in for each cluster. The $\mathbf{g}_j$ vector is computed only once per missing pattern, whereupon this $\mathbf{g}_j$ fills in all rows of `GJ` that share the same missing pattern at the between level (Line 64). For this purpose, I make use of `j.idx` (Line 46), which keeps track of the cluster indices that share the same missing pattern. Some care is needed for the clusters where the observed data vector is completely empty (i.e., all values are missing). They are not part of the missing patterns in `Zp` and need to be handled separately (Lines 79–85). For the within level, I use a similar strategy, although the situation is more complex. The $J \times P$ dimensional matrix `PJ` and the list `ALIST` (of length J) are used to store the $\mathbf{p}_j$ vectors and the $\mathbf{A}_j$ matrices for all clusters, respectively. However, these containers should be filled in while processing the missing data patterns of the level-1 data. To better understand how this is accomplished, consider missing pattern number 2 in `Mp`. This missing pattern (with a missing value for the `y1` variable only) occurs 123 times in the entire dataset (so `freq = 123`). The cluster numbers where this pattern is observed are stored in `j.idx`. Most cluster numbers are unique, but sometimes this pattern occurs multiple times within the same cluster. These frequencies are stored in `npatj` (Line 100), and the unique cluster numbers are stored in `j1.idx`. Recall that $\mathbf{A}_j = \mathbf{Q}_{ji}'\mathbf{\Lambda}_j\mathbf{Q}_{ji}$ is just the sum (over all observations within a cluster) of the inverse of $\mathbf{\Sigma}_w$, where (for each observation) the rows and columns (corresponding to the missing values) have been removed before the inverse was taken. Then the result is plugged back into a matrix where these rows and columns are replaced by zeroes. Each time the (updated) inverse of $\mathbf{\Sigma}_w$ is computed, the contribution of this pattern can be added to all the $\mathbf{A}_j$ matrices where j is in `j1.idx` (Lines 118–122). After all missing patterns in `Mp` are processed, the completed $\mathbf{A}_j$ matrices will be available in `ALIST`, so they can be extracted per cluster when it becomes necessary to compute the `IBZA.j` matrix (Line 146). In a similar fashion, the `PJ` matrix is gradually updated when running over the missing patterns. After all patterns have been processed, each row in `PJ` will contain the $\mathbf{p}_j$ vector for this cluster—to be used in the computation of `q.yy.b` (Line 145 or 154). This scheme will bring down the number of times it is necessary to invert (a submatrix of) $\mathbf{\Sigma}_w$ to a minimum. The same is true for $\mathbf{\Sigma}_{zz}$ at the between level. Unfortunately, the `IBZA.j` matrix still needs to be inverted for every cluster and remains the bottleneck (in terms of computation time) of this script. A last

small improvement in this script is that I replaced every occurrence of `(solve(A) %*% B)` by `solve(A, B)`. Finally, some generic function calls (e.g., `solve()`) were replaced by more specific function calls (e.g., `solve.default()`) to avoid the (small) overhead that comes with method dispatching, which is part of R's S3 system.

Based on Table 1, this script runs significantly faster than the previous scripts. However, there is no doubt that, given more thought, even more improvements are possible. However, `objective5.R` represents the current state of affairs at the time of this writing, and this script is used (with some cosmetic changes) in `lavaan 0.6-9`.

4.7. Stochastic Versus Fixed Covariates

In our scripts, I have made no distinction between endogenous ("y") and exogenous ("x") observed variables. In `lavaan` terminology, this corresponds to `fixed.x = FALSE`, implying that the "x" variables are treated as stochastic (assuming multivariate normality). This was only done for ease of presentation. By default, `lavaan` uses `fixed.x = TRUE`, and the "x" variables are treated as fixed constants. The advantage of this is that distributional assumptions are no longer needed for the "x" variables. However, the price is that when `fixed.x = TRUE`, "x" variables must not have missing values. As a result, all observations with missing values in any of the level-1 "x" variables will be deleted from the dataset before the analysis. In addition, if missing values are present in any of the level-2 "x" variables, the entire cluster is removed from the dataset before the analysis.

5. Conclusions

In this paper, I focused on the evaluation of the (observed) log-likelihood function for a two-level SEM in the presence of missing data. This makes it possible to obtain ML estimates of the model parameters using all available data. I have discussed two approaches (the McDonald solution and the du Toit and du Toit solution), which have been published in the literature, and have added a small extension to handle the case where $\boldsymbol{\Sigma}_b$ is singular. Finally, I presented a sequence of R scripts to evaluate this observed log-likelihood, starting from a naive (and slow) version and ending with a more refined (and faster) version. This sequence was not written with this paper in mind, but it truly reflects how the R code has evolved over time (over a period of many months) in order to prepare the code for inclusion in `lavaan`. Of course, the objective function is just a small part of the optimization machinery. Another essential ingredient is the gradient of the objective function. This gradient can be approximated numerically, but this will be slow, especially when the number of free parameters is rather large. Therefore, I also derived an analytic expression for this gradient. These derivations are not shown in this paper, but the corresponding code can be found in the file `lav_mvnorm_cluster_missing.R` in the lavaan source code. Another limitation of this paper is that I only considered the random-intercept setting. No random slopes were allowed. I refer interested readers to [9] for a detailed description of how the log-likelihood function can be expressed to handle random slopes in the complete data setting. As noted by Rockwood (see the Appendix of [9]), the adaption to the missing data setting is straightforward. Translating this to efficient R code, however, is not so straightforward. In the near future, the plan is to incorporate the Rockwood approach into `lavaan`, so that both missing data and random slopes can be handled. It would also be interesting to compare the lavaan approach with Rampart, which is implemented in OpenMx [15]. This algorithm was designed to evaluate many-level multilevel SEMs in a computationally efficient way. Some ideas of Rampart (in particular the orthogonal rotation) could perhaps be integrated in our code. As a final outlook, the plan is also to construct an expectation–maximization (EM) algorithm for the two-level SEM setting with missing data and random slopes. This is clearly doable, as this has been available in a commercial SEM package for almost two decades. Unfortunately, to the best of my knowledge, there is no (published) technical literature on this algorithm. In addition, it is not clear whether this EM algorithm can be adapted to handle the case where $\boldsymbol{\Sigma}_b$ is singular.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

Appendix A. Derivation of the du Toit and du Toit Solution

The steps to derive the du Toit and du Toit solution are similar to the steps that were used to derive the McDonald solutioin. Except that the starting point is now the first identity (A20) in Appendix B.1. Before I start with the individual blocks of $\mathbf{V}_j$, I will first derive some intermediate results. The first is an expression for $\mathbf{V}_{j(yy)}^{-1}$, which is different from $\mathbf{V}_j^{yy}$. Using the notation $\mathbf{\Lambda}_j = \bigoplus_{i=1}^{n_j} \mathbf{Q}_{ji}\mathbf{\Sigma}_w\mathbf{Q}_{ji}'$, $\mathbf{V}_{j(yy)}$ can be written as

$$\mathbf{V}_{j(yy)} = \mathbf{\Lambda}_j + \mathbf{Q}_j\mathbf{\Sigma}_b\mathbf{Q}_j'. \tag{A1}$$

To take the inverse, the (classic) Woodbury identity in (A25) can be used to obtain

$$\begin{aligned}\mathbf{V}_{j(yy)}^{-1} &= \mathbf{\Lambda}_j^{-1} - \mathbf{\Lambda}_j^{-1}\mathbf{Q}_j\left(\mathbf{\Sigma}_b^{-1} + \mathbf{Q}_j'\mathbf{\Lambda}_j^{-1}\mathbf{Q}_j\right)^{-1}\mathbf{Q}_j'\,\mathbf{\Lambda}_j^{-1} \\ &= \mathbf{\Lambda}_j^{-1} - \mathbf{\Lambda}_j^{-1}\mathbf{Q}_j\left(\mathbf{\Sigma}_b^{-1} + \mathbf{A}_j\right)^{-1}\mathbf{Q}_j'\,\mathbf{\Lambda}_j^{-1} \\ &= \mathbf{\Lambda}_j^{-1} - \mathbf{\Lambda}_j^{-1}\mathbf{Q}_j\,\mathbf{B}_j\,\mathbf{Q}_j'\,\mathbf{\Lambda}_j^{-1},\end{aligned} \tag{A2}$$

where the notation $\mathbf{A}_j = \mathbf{Q}_j'\mathbf{\Lambda}_j^{-1}\mathbf{Q}_j$ and $\mathbf{B}_j = \left(\mathbf{\Sigma}_b^{-1} + \mathbf{A}_j\right)^{-1}$ has been used. The second result is for $\mathbf{V}_{j(yy)}^{-1}\mathbf{Q}_j$ as this calls for an application of the Woodbury push-through right identity in (A27):

$$\begin{aligned}\mathbf{V}_{j(yy)}^{-1}\mathbf{Q}_j &= \left(\mathbf{\Lambda}_j + \mathbf{Q}_j\mathbf{\Sigma}_b\mathbf{Q}_j'\right)^{-1}\mathbf{Q}_j \\ &= \mathbf{\Lambda}_j^{-1}\mathbf{Q}_j\left(\mathbf{\Sigma}_b^{-1} + \mathbf{A}_j\right)^{-1}\mathbf{\Sigma}_b^{-1} \\ &= \mathbf{\Lambda}_j^{-1}\mathbf{Q}_j\mathbf{B}_j\,\mathbf{\Sigma}_b^{-1}.\end{aligned} \tag{A3}$$

Similarly, a third result is for $\mathbf{Q}_j'\mathbf{V}_{j(yy)}^{-1}$, for which we need the Woodbury push-through left identity in (A29):

$$\begin{aligned}\mathbf{Q}_j'\mathbf{V}_{j(yy)}^{-1} &= \mathbf{Q}_j'\left(\mathbf{\Lambda}_j + \mathbf{Q}_j\mathbf{\Sigma}_b\mathbf{Q}_j'\right)^{-1} \\ &= \mathbf{\Sigma}_b^{-1}\left(\mathbf{\Sigma}_b^{-1} + \mathbf{A}_j\right)^{-1}\mathbf{Q}_j'\mathbf{\Lambda}_j^{-1} \\ &= \mathbf{\Sigma}_b^{-1}\mathbf{B}_j\,\mathbf{Q}_j'\,\mathbf{\Lambda}_j^{-1}.\end{aligned} \tag{A4}$$

It follows that $\mathbf{Q}_j'\mathbf{V}_{j(yy)}^{-1}\mathbf{Q}_j$ can be written as

$$\begin{aligned}\mathbf{Q}_j'\mathbf{V}_{j(yy)}^{-1}\mathbf{Q}_j &= \mathbf{Q}_j'\mathbf{\Lambda}_j^{-1}\mathbf{Q}_j\,\mathbf{B}_j\,\mathbf{\Sigma}_b^{-1} \\ &= \mathbf{A}_j\mathbf{B}_j\,\mathbf{\Sigma}_b^{-1}.\end{aligned} \tag{A5}$$

I can now proceed with finding an expression for $\mathbf{V}_j^{zz}$. By using (A20), $\mathbf{V}_j^{zz} = \mathbf{V}_{j(z.y)}^{-1}$, where $\mathbf{V}_{j(z.y)}$ can be written as:

$$\begin{aligned}\mathbf{V}_{j(z.y)} &= \mathbf{V}_{j(zz)} - \mathbf{V}_{j(zy)}\mathbf{V}_{j(yy)}^{-1}\mathbf{V}_{j(zy)} \\ &= \mathbf{G}_j\boldsymbol{\Sigma}_{zz}\mathbf{G}_j' - \mathbf{G}_j\boldsymbol{\Sigma}_{zy}\mathbf{Q}_j'\,\mathbf{V}_{j(yy)}^{-1}\mathbf{Q}_j\boldsymbol{\Sigma}_{yz}\mathbf{G}_j' \\ &= \mathbf{G}_j\boldsymbol{\Sigma}_{zz}\mathbf{G}_j' - \mathbf{G}_j\boldsymbol{\Sigma}_{zy}\mathbf{A}_j\mathbf{B}_j\boldsymbol{\Sigma}_b^{-1}\boldsymbol{\Sigma}_{yz}\mathbf{G}_j' \\ &= \mathbf{G}_j\Big(\boldsymbol{\Sigma}_{zz} - \boldsymbol{\Sigma}_{zy}\mathbf{A}_j\mathbf{B}_j\boldsymbol{\Sigma}_b^{-1}\boldsymbol{\Sigma}_{yz}\Big)\mathbf{G}_j', \end{aligned} \tag{A6}$$

where (A5) has been used to get from the second to the third line. Using the following notation:

$$\boldsymbol{\Sigma}_{j(z.y)} = \boldsymbol{\Sigma}_{zz} - \boldsymbol{\Sigma}_{zy}\mathbf{A}_j\mathbf{B}_j\boldsymbol{\Sigma}_b^{-1}\boldsymbol{\Sigma}_{yz} \tag{A7}$$

$\mathbf{V}_j^{zz}$ can be written as

$$\mathbf{V}_j^{zz} = \Big(\mathbf{G}_j\boldsymbol{\Sigma}_{j(z.y)}\mathbf{G}_j'\Big)^{-1}, \tag{A8}$$

and the quadratic form q_j^{zz} becomes

$$\begin{aligned} q_j^{zz} &= \boldsymbol{\delta}_{jz}'\mathbf{V}_j^{zz}\boldsymbol{\delta}_{jz} \\ &= \boldsymbol{\delta}_{jz}'\Big(\mathbf{G}_j\boldsymbol{\Sigma}_{j(z.y)}\mathbf{G}_j'\Big)^{-1}\boldsymbol{\delta}_{jz}. \end{aligned} \tag{A9}$$

Next, I will seek an expression for $\mathbf{V}_j^{yz}$. Consider (A20)

$$\begin{aligned}\mathbf{V}_j^{yz} &= -\Big(\boldsymbol{\Lambda}_j + \mathbf{Q}_j\boldsymbol{\Sigma}_b\mathbf{Q}_j'\Big)^{-1}\mathbf{Q}_j\,\boldsymbol{\Sigma}_{yz}\mathbf{G}_j'\,\mathbf{V}_j^{zz} \\ &= -\boldsymbol{\Lambda}_j^{-1}\mathbf{Q}_j\mathbf{B}_j\,\boldsymbol{\Sigma}_b^{-1}\boldsymbol{\Sigma}_{yz}\mathbf{G}_j'\,\mathbf{V}_j^{zz}, \end{aligned} \tag{A10}$$

wherein the result in (A3) was used to get from the first to the second line. The quadratic form q_j^{zy} is therefore derived from

$$\begin{aligned} q_j^{yz} &= \boldsymbol{\delta}_{jy}'\mathbf{V}_j^{yz}\boldsymbol{\delta}_{jz} \\ &= -\boldsymbol{\delta}_{jy}'\boldsymbol{\Lambda}_j^{-1}\mathbf{Q}_j\mathbf{B}_j\,\boldsymbol{\Sigma}_b^{-1}\boldsymbol{\Sigma}_{yz}\mathbf{G}_j'\mathbf{V}_j^{zz}\boldsymbol{\delta}_{jz} \\ &= -\mathbf{p}_j'\,\mathbf{B}_j\boldsymbol{\Sigma}_b^{-1}\,\tilde{\mathbf{g}}_j, \end{aligned} \tag{A11}$$

where I have used

$$\mathbf{p}_j = \mathbf{Q}_j\boldsymbol{\Lambda}_j^{-1}\boldsymbol{\delta}_{jy} \quad \text{and} \quad \tilde{\mathbf{g}}_j = \boldsymbol{\Sigma}_{yz}\mathbf{G}_j'\mathbf{V}_j^{zz}. \tag{A12}$$

The tilde on $\tilde{\mathbf{g}}_j$ should alert readers that this expression is similar, but not identical to $\mathbf{g}_j$, as defined in (67). Finally, for the $\mathbf{V}_j^{yy}$ term, I will use (A20) once more to find

$$\begin{aligned}\mathbf{V}_j^{yy} &= \Big(\boldsymbol{\Lambda}_j + \mathbf{Q}_j\boldsymbol{\Sigma}_b\mathbf{Q}_j'\Big)^{-1} \\ &\quad + \Bigg[\Big(\boldsymbol{\Lambda}_j + \mathbf{Q}_j\boldsymbol{\Sigma}_b\mathbf{Q}_j'\Big)^{-1}\mathbf{Q}_j\boldsymbol{\Sigma}_{yz}\mathbf{G}_j'\,\mathbf{V}_j^{zz} \\ &\quad\quad \mathbf{G}_j\boldsymbol{\Sigma}_{zy}\,\mathbf{Q}_j'\Big(\boldsymbol{\Lambda}_j + \mathbf{Q}_j\boldsymbol{\Sigma}_b\mathbf{Q}_j'\Big)^{-1}\Bigg]. \end{aligned} \tag{A13}$$

The first line is the inverse of (A1) and can be rewritten as in (A2). The second line equals (minus) the expression for $\mathbf{V}_j^{yz}$ in (A10). The last line corresponds to the result in (A4), premultiplied by $\mathbf{G}_j\boldsymbol{\Sigma}_{zy}$. Applying these results, (A13) may be rewritten as

$$\begin{aligned}\mathbf{V}_j^{yy} &= \boldsymbol{\Lambda}_j^{-1} - \boldsymbol{\Lambda}_j^{-1}\mathbf{Q}_j\mathbf{B}_j\mathbf{Q}_j'\boldsymbol{\Lambda}_j^{-1} + \boldsymbol{\Lambda}_j^{-1}\mathbf{Q}_j\mathbf{B}_j\boldsymbol{\Sigma}_b^{-1}\boldsymbol{\Sigma}_{yz}\mathbf{G}_j'\mathbf{V}_j^{zz}\mathbf{G}_j\boldsymbol{\Sigma}_{zy}\,\boldsymbol{\Sigma}_b^{-1}\mathbf{B}_j\mathbf{Q}_j'\boldsymbol{\Lambda}_j^{-1} \\ &= \boldsymbol{\Lambda}_j^{-1} - \boldsymbol{\Lambda}_j^{-1}\mathbf{Q}_j\Big[\mathbf{B}_j - \mathbf{B}_j\boldsymbol{\Sigma}_b^{-1}\boldsymbol{\Sigma}_{yz}\mathbf{G}_j'\mathbf{V}_j^{zz}\mathbf{G}_j\boldsymbol{\Sigma}_{zy}\,\boldsymbol{\Sigma}_b^{-1}\mathbf{B}_j\Big]\mathbf{Q}_j'\boldsymbol{\Lambda}_j^{-1}.\end{aligned} \tag{A14}$$

As a side note, du Toit and du Toit [3] further simplify the last expression by writing the term within the square brackets as $-\mathbf{H}_j$, so that $\mathbf{V}_j^{yy}$ can be written more compactly as $\boldsymbol{\Lambda}_j^{-1} + \boldsymbol{\Lambda}_j^{-1}\mathbf{Q}_j\mathbf{H}_j\mathbf{Q}_j'\boldsymbol{\Lambda}_j^{-1}$. The quadratic form q_j^{yy} can be written as

$$\begin{aligned}q_j^{yy} &= \boldsymbol{\delta}_{jy}'\mathbf{V}^{yy}\boldsymbol{\delta}_{jy} \\ &= \boldsymbol{\delta}_{jy}'\boldsymbol{\Lambda}_j^{-1}\boldsymbol{\delta}_{jy} - \mathbf{p}_j'\Big[\mathbf{B}_j - \mathbf{B}_j\boldsymbol{\Sigma}_b^{-1}\boldsymbol{\Sigma}_{yz}\mathbf{G}_j'\mathbf{V}_j^{zz}\mathbf{G}_j\boldsymbol{\Sigma}_{zy}\,\boldsymbol{\Sigma}_b^{-1}\mathbf{B}_j\Big]\mathbf{p}_j.\end{aligned} \tag{A15}$$

To derive an expression for the determinant of $\mathbf{V}_j$, this time using (A23), results in

$$|\mathbf{V}_j| = |\mathbf{V}_{j(yy)}| \cdot |\mathbf{V}_{j(z.y)}| \tag{A16}$$

$$= |\mathbf{V}_{j(yy)}| \cdot |\mathbf{G}_j\boldsymbol{\Sigma}_{j(z.y)}\mathbf{G}'|. \tag{A17}$$

Recall that $\mathbf{V}_{j(yy)} = \boldsymbol{\Lambda}_j + \mathbf{Q}_j\boldsymbol{\Sigma}_b\mathbf{Q}_j'$. Using the determinant identity in (A32), results in

$$|\mathbf{V}_j| = |\boldsymbol{\Sigma}_b^{-1} + \mathbf{Q}_j'\boldsymbol{\Lambda}_j^{-1}\mathbf{Q}_j| \cdot |\boldsymbol{\Sigma}_b| \cdot |\boldsymbol{\Lambda}_j| \cdot |\mathbf{G}_j\boldsymbol{\Sigma}_{j(z.y)}\mathbf{G}'| \tag{A18}$$

$$= |\boldsymbol{\Sigma}_b^{-1} + \mathbf{A}_j| \cdot |\boldsymbol{\Sigma}_b| \cdot |\boldsymbol{\Lambda}_j| \cdot |\mathbf{G}_j\boldsymbol{\Sigma}_{j(z.y)}\mathbf{G}'|. \tag{A19}$$

The only "large" matrix for which the determinant must be computed is $\boldsymbol{\Lambda}_j$, but that is a block diagonal matrix, so the determinant is easily obtained as in (A35). The final results are summarized in Section 3.4.

Appendix B. Some Useful Formulas of Matrix Algebra

Appendix B.1. Inverse and Determinant of a Partitioned Matrix

There are two different but equivalent identities for the inverse of a partitioned matrix. Identity 1 is given by

$$\begin{pmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{pmatrix}^{-1} = \begin{pmatrix}\mathbf{M}^{-1} & -\mathbf{M}^{-1}\mathbf{B}\mathbf{D}^{-1}\\ -\mathbf{D}^{-1}\mathbf{C}\mathbf{M}^{-1} & \mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{C}\mathbf{M}^{-1}\mathbf{B}\mathbf{D}^{-1}\end{pmatrix}, \tag{A20}$$

where

$$\mathbf{M} = \mathbf{A} - \mathbf{B}\mathbf{D}^{-1}\mathbf{C},$$

and Identity 2 is given by

$$\begin{pmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{C} & \mathbf{D}\end{pmatrix}^{-1} = \begin{pmatrix}\mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\mathbf{Q}^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{Q}^{-1}\\ -\mathbf{Q}^{-1}\mathbf{C}\mathbf{A}^{-1} & \mathbf{Q}^{-1}\end{pmatrix}, \tag{A21}$$

where

$$\mathbf{Q} = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}.$$

Similarly, the determinant of a partitioned matrix can be computed in two different ways:

$$|\mathbf{A}| = \begin{vmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{vmatrix} \tag{A22}$$

$$= |\mathbf{A}_{22}| \, |\mathbf{A}_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{A}_{21}| \tag{A23}$$

$$= |\mathbf{A}_{11}| \, |\mathbf{A}_{22} - \mathbf{A}_{21}\mathbf{A}_{11}^{-1}\mathbf{A}_{12}|. \tag{A24}$$

Appendix B.2. The Woodbury Identity

The Woodbury identity gives an expression for the following inverse:

$$(\mathbf{A} + \mathbf{UBV})^{-1},$$

where $\mathbf{A}$ ($p \times p$) is square and nonsingular; $\mathbf{U}$ ($p \times r$), $\mathbf{B}$ ($r \times s$), and $\mathbf{V}$ ($s \times p$) can be rectangular (or square). Several variants and special cases exist. An overview can be found in [16]. I only provide two identities. The first is the classic identity that is widely used in the literature. It assumes that $\mathbf{B}$ is square and nonsingular so that its inverse $\mathbf{B}^{-1}$ exists:

$$(\mathbf{A} + \mathbf{UBV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{B}^{-1} + \mathbf{VA}^{-1}\mathbf{U})^{-1}\mathbf{VA}^{-1}. \tag{A25}$$

The second identity is less known but is given in [16] for the general case and in [17] for the symmetric case where $\mathbf{V} = \mathbf{U}'$. This version is more general and allows $\mathbf{B}$ to be singular:

$$(\mathbf{A} + \mathbf{UBV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{I}_r + \mathbf{BVA}^{-1}\mathbf{U})^{-1}\mathbf{BVA}^{-1}. \tag{A26}$$

The following identities give expressions for the setting where the original term is either postmultiplied with $\mathbf{U}$ or premultiplied with $\mathbf{V}$. Expressions are known as the Woodbury push-through "right" identities when they are postmultiplied. Here are two versions: The first version requires $\mathbf{B}$ to be nonsingular, but the second version does not:

$$(\mathbf{A} + \mathbf{UBV})^{-1}\mathbf{U} = \mathbf{A}^{-1}\mathbf{U}(\mathbf{B}^{-1} + \mathbf{VA}^{-1}\mathbf{U})^{-1}\mathbf{B}^{-1} \tag{A27}$$

$$(\mathbf{A} + \mathbf{UBV})^{-1}\mathbf{U} = \mathbf{A}^{-1}\mathbf{U}(\mathbf{I}_r + \mathbf{BVA}^{-1}\mathbf{U})^{-1}. \tag{A28}$$

Expressions are known as Woodbury push-through "left" identities when they are premultiplied. Again, two versions are given, but only the first version requires $\mathbf{B}$ to be nonsingular:

$$\mathbf{V}(\mathbf{A} + \mathbf{UBV})^{-1} = \mathbf{B}^{-1}(\mathbf{B}^{-1} + \mathbf{VA}^{-1}\mathbf{U})^{-1}\mathbf{VA}^{-1} \tag{A29}$$

$$\mathbf{V}(\mathbf{A} + \mathbf{UBV})^{-1} = (\mathbf{I}_s + \mathbf{VA}^{-1}\mathbf{UB})^{-1}\mathbf{VA}^{-1}. \tag{A30}$$

Appendix B.3. Determinant Identities

A well-known determinant identity (e.g., [18], p. 420) is related to the Woodbury identity:

$$|\mathbf{A} + \mathbf{UBV}| = |\mathbf{B}^{-1} + \mathbf{VA}^{-1}\mathbf{U}| \cdot |\mathbf{B}| \cdot |\mathbf{A}|, \tag{A31}$$

where $\mathbf{A}$ ($p \times p$) and $\mathbf{B}$ ($r \times r$) are square and nonsingular, whereas $\mathbf{U}$ ($p \times r$) and $\mathbf{V}$ ($r \times p$) can be rectangular or square. Because $|\mathbf{AB}| = |\mathbf{A}| \cdot |\mathbf{B}|$, this identity can also be rewritten as follows:

$$\begin{aligned} |\mathbf{A} + \mathbf{UBV}| &= |\mathbf{B}^{-1} + \mathbf{VA}^{-1}\mathbf{U}| \cdot |\mathbf{B}| \cdot |\mathbf{A}| \\ &= |(\mathbf{B}^{-1} + \mathbf{VA}^{-1}\mathbf{U})\mathbf{B}| \cdot |\mathbf{A}| \\ &= |\mathbf{B}^{-1}\mathbf{B} + \mathbf{VA}^{-1}\mathbf{UB}| \cdot |\mathbf{A}| \\ &= |\mathbf{I}_r + \mathbf{VA}^{-1}\mathbf{UB}| \cdot |\mathbf{A}|. \end{aligned} \tag{A32}$$

The latter expression has the advantage that it can be applied in settings where $\mathbf{B}$ is singular.

Appendix B.4. Block Diagonal Matrices

A block diagonal matrix is similar to a diagonal (square) matrix, but every diagonal element is itself a (square) matrix. All other elements are zero. For example, a block diagonal matrix with K blocks has the following form:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_K \end{bmatrix}. \tag{A33}$$

The inverse of a block diagonal matrix is again a block diagonal matrix where the diagonal elements are the inverted blocks (assuming their inverses exist):

$$\mathbf{A}^{-1} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_K \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}_1^{-1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2^{-1} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_K^{-1} \end{bmatrix}. \tag{A34}$$

The determinant of a block diagonal matrix is simply the product of the determinants of the individual blocks:

$$|\mathbf{A}| = |\mathbf{A}_1| \times |\mathbf{A}_1| \times \dots |\mathbf{A}_K|. \tag{A35}$$

Appendix C. R Code

Appendix C.1. The main.R File

```
library(lavaan) # must be 0.6-9 (or higher)

# read in two-level data with missing data at both levels
Data <- read.table("https://www.da.ugent.be/datasets/demo2_missing.dat",
                   na.strings = "-999999")
names(Data) <- c("y1", "y2", "y3", "y4", "y5", "y6", "y7", "y8", "y9", "y10",
                 "x1", "x2", "x3", "z1", "z2", "z3", "z4", "w1", "w2", "w3",
                 "cluster")

model <- '
    level: 1
      # y1-y6 are splitted
      fw1 =~ y1 + y2 + y3
      fw2 =~ y4 + y5 + y6
      fw1 ~~ start(0.02)*fw2

      # within-only factor
      fa  =~ y7 + y8 + y9 + y10
      fa ~ start(0.3)*fw1 + start(0.4)*fw2

      # regression
      fw1 ~ start(0.1)*x1 + start(0.2)*x2 + start(0.3)*x3

  level: 2
      # y1-y6 are splitted
      fb1 =~ y1 + y2 + y3
      fb2 =~ y4 + y5 + y6
      fb1 ~~ start(0.01)*fb2

      # between-only factor
      fbz =~ z1 + z2 + z3 + z4

      # regression with latent predictors
      fbz ~ start(0.1)*fb1 + start(0.2)*fb2

      # regression with observed predictors
      fb1 ~ start(0.3)*w1 + start(0.4)*w2 + start(0.5)*w3
```

```
'

# we run lavaan, but without fitting (do.fit = FALSE)
# - we will use the starting values only
# - lavaan will compute the model implied statistics for these starting values
# - lavaan will also analyze the missing patterns
dummy.fit <- sem(model, data = Data, cluster = "cluster", do.fit = FALSE,
                 fixed.x = FALSE, missing = "ml", h1 = FALSE)

# extract parameter vector (npar = 83)
x <- coef(dummy.fit)

# extract some internal slots (warning: don't do this in real world
# R code that you upload to CRAN!)
lavmodel <- dummy.fit@Model  # internal model representation
Lp <- dummy.fit@Data@Lp[[1]] # information about the clustering
Y1 <- dummy.fit@Data@X[[1]]  # raw data
Y2 <- dummy.fit@SampleStats@YLp[[1]][[2]]$Y2 # raw data level 2

# model implied statistics (within and between)
implied <- lav_model_implied(lavmodel)

# model implied statistics (smaller components: sigma.w, sigma.b, sigma.yz...)
out <- lavaan:::lav_mvnorm_cluster_implied22l(Lp = Lp, implied = implied)

# extract some useful values from Lp
nclusters    <- Lp$nclusters[[2]]
cluster.size <- Lp$cluster.size[[2]]
cluster.idx  <- Lp$cluster.idx[[2]]
between.idx  <- Lp$between.idx[[2]]


# 1. evaluate using objective1() function
source("objective1.R")
print(.obj1(mu.y = out$mu.y, mu.z = out$mu.z, sigma.w = out$sigma.w,
                      sigma.b = out$sigma.b, sigma.zz = out$sigma.zz,
                      sigma.yz = out$sigma.yz), digits = 12)

# 2. evaluate using objective2() function
source("objective2.R")
print(.obj2(mu.y = out$mu.y, mu.z = out$mu.z, sigma.w = out$sigma.w,
                      sigma.b = out$sigma.b, sigma.zz = out$sigma.zz,
                      sigma.yz = out$sigma.yz), digits = 12)

# 3. evaluate using objective3() function
source("objective3.R")
print(.obj3(mu.y = out$mu.y, mu.z = out$mu.z, sigma.w = out$sigma.w,
                      sigma.b = out$sigma.b, sigma.zz = out$sigma.zz,
                      sigma.yz = out$sigma.yz), digits = 12)

# 4. evaluate using objective4() function
source("objective4.R")
# pre-compute information about missing patterns per cluster
MP <- vector("list", length = nclusters)
for(j in seq_len(nclusters)) {
    Yw.j <- Y1[cluster.idx == j, -between.idx, drop = FALSE]
    MP[[j]] <- lavaan:::lav_data_missing_patterns(Yw.j)
}

print(.obj4(mu.y = out$mu.y, mu.z = out$mu.z, sigma.w = out$sigma.w,
                      sigma.b = out$sigma.b, sigma.zz = out$sigma.zz,
                      sigma.yz = out$sigma.yz), digits = 12)

# 5. evaluate using objective5() function
source("objective5.R")
# pre-compute information about missing patterns for the full dataset
Mp <- dummy.fit@Data@Mp[[1]]
Zp <- Mp$Zp

```

```
print(.obj5(mu.y = out$mu.y, mu.z = out$mu.z, sigma.w = out$sigma.w,
                        sigma.b = out$sigma.b, sigma.zz = out$sigma.zz,
                        sigma.yz = out$sigma.yz), digits = 12)


library(microbenchmark)
benchmark <- microbenchmark(
.obj1(mu.y = out$mu.y, mu.z = out$mu.z, sigma.w = out$sigma.w,
                        sigma.b = out$sigma.b, sigma.zz = out$sigma.zz,
                        sigma.yz = out$sigma.yz),
.obj2(mu.y = out$mu.y, mu.z = out$mu.z, sigma.w = out$sigma.w,
                        sigma.b = out$sigma.b, sigma.zz = out$sigma.zz,
                        sigma.yz = out$sigma.yz),
.obj3(mu.y = out$mu.y, mu.z = out$mu.z, sigma.w = out$sigma.w,
                        sigma.b = out$sigma.b, sigma.zz = out$sigma.zz,
                        sigma.yz = out$sigma.yz),
.obj4(mu.y = out$mu.y, mu.z = out$mu.z, sigma.w = out$sigma.w,
                        sigma.b = out$sigma.b, sigma.zz = out$sigma.zz,
                        sigma.yz = out$sigma.yz),
.obj5(mu.y = out$mu.y, mu.z = out$mu.z, sigma.w = out$sigma.w,
                        sigma.b = out$sigma.b, sigma.zz = out$sigma.zz,
                        sigma.yz = out$sigma.yz), times = 100
)
```

Appendix C.2. The objective1.R File

```
# objective function 1
#
# this is the 'naive' function where we construct the 'full' V_j matrix
# and compute the determinant and inverse of V_j

.obj1 <- function(mu.y = NULL, mu.z = NULL, sigma.w = NULL, sigma.b = NULL,
                  sigma.zz = NULL, sigma.yz = NULL, loglik. = FALSE) {

    sigma.zy <- t(sigma.yz)

    loglik <- numeric(nclusters)
    for(j in seq_len(nclusters)) {

        # cluster size
        nj <- cluster.size[j]

        if(length(between.idx) > 0L) {
            Yw.j <- Y1[cluster.idx == j, -between.idx, drop = FALSE]
            z.j  <- as.numeric(Y2[j, between.idx, drop = FALSE])

            # create G.j
            G.j <- diag(length(between.idx))
            na.idx <- which(is.na(z.j))
            if(length(na.idx) > 0L) {
                # remove na rows
                G.j <- G.j[-na.idx,,drop = FALSE]
                z.j <- z.j[-na.idx]
            }
        } else {
            Yw.j <- Y1[cluster.idx == j, , drop = FALSE]
        }

        # create Q.ji for each unit
        Q.ji <- vector("list", length = nj)
        W.ji <- vector("list", length = nj)
        for(i in seq_len(nj)) {
            dtmp <- diag( ncol(Yw.j) )
            na.idx <- which(is.na(Yw.j[i,]))
            if(length(na.idx) > 0L) {
                dtmp <- dtmp[-na.idx,,drop = FALSE]
            }
            Q.ji[[i]] <- dtmp
```

```
            W.ji[[i]] <- Q.ji[[i]] %*% sigma.w %*% t(Q.ji[[i]])
        }

        # create Q.j
        Q.j <- do.call("rbind", Q.ji)

        # create V.j
        if(length(between.idx) > 0L) {
            V.zz <- G.j %*% sigma.zz %*% t(G.j)
            V.zy <- G.j %*% sigma.zy %*% t(Q.j)
            V.yz <- Q.j %*% sigma.yz %*% t(G.j)
            V.yy  <- Q.j %*% sigma.b  %*% t(Q.j) + lav_matrix_bdiag(W.ji)

            V.j <- rbind( cbind(V.zz, V.zy),
                          cbind(V.yz, V.yy))
            est.j <- c(as.numeric(G.j %*% mu.z),
                       unlist(sapply(Q.ji, function(x) x %*% mu.y)))
            obs.j <- c(as.numeric(z.j),
                         as.numeric(na.omit(lav_matrix_vecr(Yw.j))))
        } else {
           V.j <- Q.j %*% sigma.b  %*% t(Q.j) + lav_matrix_bdiag(W.ji)
           est.j <- as.numeric(unlist(sapply(Q.ji, function(x) x %*% mu.y)))
           obs.j <- as.numeric(na.omit(lav_matrix_vecr(Yw.j)))
        }

        V.j.inv <- lavaan:::lav_matrix_symmetric_inverse(V.j, logdet = TRUE)
        V.j.logdet <- attr(V.j.inv, "logdet")

        delta.j <- obs.j - est.j
        q.j <- sum(delta.j %*% V.j.inv * delta.j)

        if(loglik.) {
            P <- length(obs.j); LOG.2PI <- log(2 * pi)
            loglik[j] <- -(P * LOG.2PI + V.j.logdet + q.j)/2
        } else {
            loglik[j] <-  V.j.logdet + q.j
        }

    }

    out <- sum(loglik)

    out
}
```

Appendix C.3. The objective2.R File

```
# objective function 2
#
# This is a straighforward implementation of the 'McDonald' solution,
# adapted to handle the case where sigma.b can be singular

.obj2 <- function(mu.y = NULL, mu.z = NULL, sigma.w = NULL, sigma.b = NULL,
                  sigma.zz = NULL, sigma.yz = NULL, loglik. = FALSE) {

    sigma.zy <- t(sigma.yz)

    loglik <- numeric(nclusters)
    for(j in seq_len(nclusters)) {

        # cluster size
        nj <- cluster.size[j]

        if(length(between.idx) > 0L) {
            Yw.j <- Y1[cluster.idx == j, -between.idx, drop = FALSE]
            z.j  <- as.numeric(Y2[j, between.idx, drop = FALSE])

            # create G.j
```

```
            G.j <- diag(length(between.idx))
            na.idx <- which(is.na(z.j))
            if(length(na.idx) > 0L) {
                # remove na rows
                G.j <- G.j[-na.idx,,drop = FALSE]
                z.j <- z.j[-na.idx]
            }
        } else {
            Yw.j <- Y1[cluster.idx == j, , drop = FALSE]
        }

        # create Q.ji for each unit
        Q.ji <- vector("list", length = nj)
        W.ji <- vector("list", length = nj)
        A.ji <- vector("list", length = nj)
        for(i in seq_len(nj)) {
            dtmp <- diag( ncol(Yw.j) )
            na.idx <- which(is.na(Yw.j[i,]))
            if(length(na.idx) > 0L) {
                dtmp <- dtmp[-na.idx,,drop = FALSE]
            }
            Q.ji[[i]] <- dtmp
            W.ji[[i]] <- Q.ji[[i]] %*% sigma.w %*% t(Q.ji[[i]])
            A.ji[[i]] <- t(Q.ji[[i]]) %*% solve(W.ji[[i]]) %*% Q.ji[[i]]
        }

        # create Q.j
        Q.j <- do.call("rbind", Q.ji)

        A.j <- Reduce("+", A.ji)
        bdiag_sigma.w.j <- lav_matrix_bdiag(W.ji)
        bdiag_sigma.w.j.inv <- lav_matrix_bdiag( lapply(W.ji, solve) )


        if(length(between.idx) > 0L && nrow(G.j) > 0L) {
            sigma.j.zz.inv <- solve(G.j %*% sigma.zz %*% t(G.j))
            sigma.j.b.z <-
                ( sigma.b -
                  sigma.yz %*% t(G.j) %*% sigma.j.zz.inv %*% G.j %*% sigma.zy )
        } else {
            sigma.j.b.z <- sigma.b
        }
        IBZA.j <- diag(ncol(sigma.j.b.z)) + sigma.j.b.z %*% A.j
        IBZA.j.inv <- solve(IBZA.j)


        if(length(between.idx) > 0L && nrow(G.j) > 0L) {

            est.j <- c(as.numeric(G.j %*% mu.z),
                       unlist(sapply(Q.ji, function(x) x %*% mu.y)))

            obs.j <- c(as.numeric(z.j),
                       as.numeric(na.omit(lav_matrix_vecr(Yw.j))))

            delta.j <- obs.j - est.j
            z.idx <- seq_len(nrow(G.j))
            y.idx <- seq_len(length(est.j) - length(z.idx)) + length(z.idx)

            delta.z <- obs.j[ z.idx] - est.j[ z.idx]
            delta.y <- obs.j[ y.idx] - est.j[ y.idx]
            p.j <- t(Q.j) %*% bdiag_sigma.w.j.inv %*% delta.y
            g.j <- sigma.yz %*% t(G.j) %*% sigma.j.zz.inv %*% delta.z

            q.zz <- drop( t(delta.z) %*% sigma.j.zz.inv %*% delta.z +
              t(g.j) %*% A.j %*% IBZA.j.inv %*% g.j )

            q.yz <- -1 * drop( t(p.j) %*% IBZA.j.inv %*% g.j)

            q.yy <- drop( t(delta.y) %*% bdiag_sigma.w.j.inv %*% delta.y -
```

```
                            t(p.j) %*% IBZA.j.inv %*% sigma.j.b.z %*% p.j )

            q.j <- q.yy + 2*q.yz + q.zz

        } else {
            est.j <- as.numeric(unlist(sapply(Q.ji, function(x) x %*% mu.y)))
            obs.j <- as.numeric(na.omit(lav_matrix_vecr(Yw.j)))
            delta.y <- obs.j - est.j
            p.j <- t(Q.j) %*% bdiag_sigma.w.j.inv %*% delta.y
            q.j <- drop( t(delta.y) %*% bdiag_sigma.w.j.inv %*% delta.y -
                            t(p.j) %*% IBZA.j.inv %*% sigma.j.b.z %*% p.j )
        }

        # determinant
        if(length(between.idx) > 0L && nrow(G.j) > 0L) {
            Yw.j <- sum(sapply(W.ji, function(x) { log(det(x)) }))
            tmp2 <- log(det(IBZA.j))
            tmp3 <- log(det(G.j %*% sigma.zz %*% t(G.j)))
            V.j.logdet <- Yw.j + tmp2 + tmp3
        } else {
            Yw.j <- sum(sapply(W.ji, function(x) { log(det(x)) }))
            tmp2 <- log(det(IBZA.j))
            V.j.logdet <- Yw.j + tmp2
        }

        if(loglik.) {
            P <- length(obs.j); LOG.2PI <- log(2 * pi)
            loglik[j] <- -(P * LOG.2PI + V.j.logdet + q.j)/2
        } else {
            loglik[j] <-  V.j.logdet + q.j
        }
    }

    out <- sum(loglik)

    out
}
```

Appendix C.4. The objective3.R File

```
# objective function 3
#
# - avoid Q.ji, W.ji, A.ij, G.j, and other minor improvements
# - update inverse/determinant of sigma.w and sigma.zz
# - treat complete cases/clusters more efficiently

.obj3 <- function(mu.y = NULL, mu.z = NULL, sigma.w = NULL, sigma.b = NULL,
                  sigma.zz = NULL, sigma.yz = NULL, loglik. = FALSE) {

    LOG.2PI <- log(2 * pi)

    sigma.w.inv <- solve(sigma.w)
    sigma.w.logdet <- log(det(sigma.w))

    # y
    ny <- NCOL(sigma.w)
    ny.diag.idx <- lav_matrix_diag_idx(ny)

    # z
    nz <- length(between.idx)
    if(nz > 0L) {
        sigma.zy <- t(sigma.yz)
        sigma.zz.inv <- solve.default(sigma.zz)
        sigma.zz.logdet <- log(det(sigma.zz))
        sigma.zi.zy <- sigma.zz.inv %*% sigma.zy
        sigma.b.z <- sigma.b - sigma.yz %*% sigma.zi.zy
    }

```

```
    loglik <- numeric(nclusters)
    for(j in seq_len(nclusters)) {

        # cluster size
        nj <- cluster.size[j]

        # y
        Yw.j <- Y1[cluster.idx == j, -between.idx, drop = FALSE]
        obs.y <- lav_matrix_vecr(Yw.j)
        y.na.idx <- which(is.na(obs.y))
        if(length(y.na.idx) > 0L) {
            obs.y <- obs.y[-y.na.idx]
        }

        # compute within-level quantities
        W.logdet.j <- 0
        A.j <- matrix(0, ny, ny)
        p.j <- matrix(0, ny, 1L)
        q.yy.a <- 0
        for(i in seq_len(nj)) {
            na.idx <- which(is.na(Yw.j[i,]))
            if(length(na.idx) > 0L) {
                wij <- sigma.w[-na.idx, -na.idx, drop = FALSE]
                wij.inv <-
                    lavaan:::lav_matrix_symmetric_inverse_update(
                        S.inv = sigma.w.inv,
                        rm.idx = na.idx, logdet = TRUE,
                        S.logdet = sigma.w.logdet)
                wij.logdet <- attr(wij.inv, "logdet")
                W.logdet.j <- W.logdet.j + wij.logdet
                A.j[-na.idx, -na.idx] <-
                    A.j[-na.idx, -na.idx, drop = FALSE] + wij.inv
                delta.i <- (Yw.j[i,-na.idx] - mu.y[-na.idx])
                wi.delta.i <- wij.inv %*% delta.i
                p.j[-na.idx, 1L] <- p.j[-na.idx, 1L] + wi.delta.i
                q.yy.a <- q.yy.a + sum(wi.delta.i * delta.i)
            } else {
                # complete case
                W.logdet.j <- W.logdet.j + sigma.w.logdet
                A.j <- A.j + sigma.w.inv
                delta.i <- (Yw.j[i,] - mu.y)
                wi.delta.i <- sigma.w.inv %*% delta.i
                p.j <- p.j + wi.delta.i
                q.yy.a <- q.yy.a + sum(wi.delta.i * delta.i)
            }
        }

        # between
        if(nz > 0L) {
            obs.z <- as.numeric(Y2[j, between.idx, drop = FALSE])
            est.z <- mu.z
            z.na.idx <- which(is.na(obs.z))

            # three possibilities:
            # - all z are missing  -> sigma.j.b.z = sigma.b, no between
            # - some z are missing -> update sigma.j.zz.{inv/logdet}
            # - z is comlete       -> just use sigma.zz
            if(length(z.na.idx) == nz) {
                sigma.j.b.z <- sigma.b
                obs.z <- obs.z[-z.na.idx] # for P
            } else if(length(z.na.idx) > 0L) {
                obs.z <- obs.z[-z.na.idx]
                est.z <- est.z[-z.na.idx]
                delta.z <- obs.z - est.z
                sigma.j.zz.inv <- lavaan:::lav_matrix_symmetric_inverse_update(
                        S.inv = sigma.zz.inv,
                        rm.idx = z.na.idx, logdet = TRUE,
                        S.logdet = sigma.zz.logdet)
                sigma.j.zz.logdet <- attr(sigma.j.zz.inv, "logdet")
```

```
                sigma.j.zi.zy <-
                    sigma.j.zz.inv %*% sigma.zy[-z.na.idx, ,drop = FALSE]
                sigma.j.b.z <- ( sigma.b -
                    sigma.yz[, -z.na.idx, drop = FALSE] %*% sigma.j.zi.zy )
            } else { # no missings for z
                delta.z <- obs.z - est.z
                sigma.j.zz <- sigma.zz
                sigma.j.zz.inv <- sigma.zz.inv
                sigma.j.zz.logdet <- sigma.zz.logdet
                sigma.j.zi.zy <- sigma.zi.zy
                sigma.j.b.z <- sigma.b.z
            }
        }

        # IBZA.j
        IBZA.j <- sigma.j.b.z %*% A.j
        IBZA.j[ny.diag.idx] <- IBZA.j[ny.diag.idx] + 1
        IBZA.j.inv <- solve(IBZA.j) # NOT symmetric!
        IBZA.j.logdet <- log(det(IBZA.j))

        # quadratic form
        if(nz > 0L && length(z.na.idx) != nz) {

            A.IBZA.j.inv  <- A.j %*% IBZA.j.inv
            IBZA.j.inv.BZ <- IBZA.j.inv %*% sigma.j.b.z
            g.j <- drop(delta.z %*% sigma.j.zi.zy)
            p.j <- drop(p.j)

            q.zz.a <-       sum(colSums(delta.z * sigma.j.zz.inv) * delta.z)
            q.zz.b <-       sum(colSums(g.j     * A.IBZA.j.inv)   * g.j)
            q.zy   <- -1 * sum(colSums(p.j     * IBZA.j.inv)     * g.j)
            q.yy.b <-       sum(colSums(p.j     * IBZA.j.inv.BZ)  * p.j)

            q.j <- (q.yy.a - q.yy.b) + 2*q.zy + (q.zz.a + q.zz.b)

        } else {
           IBZA.j.inv.BZ <- IBZA.j.inv %*% sigma.j.b.z
           p.j <- drop(p.j)

           q.yy.b <- sum(colSums(p.j * IBZA.j.inv.BZ) * p.j)
           q.j <- (q.yy.a - q.yy.b)
        }

        # determinant
        if(nz > 0L && length(z.na.idx) != nz) {
            V.j.logdet <- W.logdet.j + IBZA.j.logdet + sigma.j.zz.logdet
        } else {
            V.j.logdet <- W.logdet.j + IBZA.j.logdet
        }

        if(loglik.) {
            P <- length(obs.y) + length(obs.z)
            loglik[j] <- -(P * LOG.2PI + V.j.logdet + q.j)/2
        } else {
            loglik[j] <-  V.j.logdet + q.j
        }
    }

    out <- sum(loglik)

    out
}
```

Appendix C.5. The objective4.R File

```
# objective function 4
#
# - use missing patterns per cluster
```

```
4
5  .obj4 <- function(mu.y = NULL, mu.z = NULL, sigma.w = NULL, sigma.b = NULL,
6                    sigma.zz = NULL, sigma.yz = NULL, loglik. = FALSE) {
7
8      LOG.2PI <- log(2 * pi)
9
10     sigma.w.inv <- solve(sigma.w)
11     sigma.w.logdet <- log(det(sigma.w))
12
13     # y
14     ny <- NCOL(sigma.w)
15     ny.diag.idx <- lav_matrix_diag_idx(ny)
16     Y1w <- Y1[, -between.idx, drop = FALSE]
17     Y1w.c <- t( t(Y1w) - mu.y )
18
19     # z
20     nz <- length(between.idx)
21     if(nz > 0L) {
22         Z <- Y2[, between.idx, drop = FALSE]
23         Z.c <- t( t(Z) - mu.z )
24         sigma.zy <- t(sigma.yz)
25         sigma.zz.inv <- solve(sigma.zz)
26         sigma.zz.logdet <- log(det(sigma.zz))
27         sigma.zi.zy <- sigma.zz.inv %*% sigma.zy
28         sigma.b.z <- sigma.b - sigma.yz %*% sigma.zi.zy
29     }
30
31     loglik <- numeric(nclusters)
32     for(j in seq_len(nclusters)) {
33
34         # cluster size
35         nj <- cluster.size[j]
36
37         # centered data Y1
38         Yw.j <- Y1w.c[cluster.idx == j, , drop = FALSE]
39
40         # missing patterns for this cluster
41         Mp.j <- MP[[j]]
42
43         # for each Y1 pattern, compute within-level quantities
44         NY <- 0L; A.j <- matrix(0, ny, ny); p.j <- numeric(ny)
45         W.logdet.j <- 0; q.yy.a <- 0
46         for(p in seq_len(Mp.j$npatterns)) {
47
48             freq <- Mp.j$freq[p]; na.idx <- which(!Mp.j$pat[p,])
49
50             # store number of observed values (only needed if loglik. = TRUE)
51             if(loglik.) {
52                 NY <- NY + (sum(Mp.j$pat[p,]) * freq)
53             }
54
55             if(length(na.idx) > 0L) {
56                 wp <- sigma.w[-na.idx, -na.idx, drop = FALSE]
57                 wp.inv <- lavaan:::lav_matrix_symmetric_inverse_update(
58                               S.inv = sigma.w.inv, rm.idx = na.idx,
59                               logdet = TRUE, S.logdet = sigma.w.logdet)
60                 wp.logdet <- attr(wp.inv, "logdet")
61                 W.logdet.j <- W.logdet.j + (wp.logdet * freq)
62                 A.j[-na.idx, -na.idx] <-
63                     A.j[-na.idx, -na.idx, drop = FALSE] + (wp.inv * freq)
64
65                 this.y <- Yw.j[Mp.j$case.idx[[p]], -na.idx, drop = FALSE]
66                 wi.delta.p <- this.y %*% wp.inv
67                 p.j[-na.idx] <- p.j[-na.idx] + colSums(wi.delta.p)
68                 q.yy.a <- q.yy.a + sum(wi.delta.p * this.y) # TR
69
70             } else {
71                 # complete case
72                 W.logdet.j <- W.logdet.j + (sigma.w.logdet * freq)
```

```
            A.j <- A.j + (sigma.w.inv * freq)

            this.y <- Yw.j[Mp.j$case.idx[[p]], , drop = FALSE]
            wi.delta.p <- this.y %*% sigma.w.inv
            p.j <- p.j + colSums(wi.delta.p)
            q.yy.a <- q.yy.a + sum(wi.delta.p * this.y)
        }
    }

    # between
    g.j <- numeric(ny); q.zz.a <- 0
    if(nz > 0L) {
        z.na.idx <- which(is.na(Z.c[j, ,drop = FALSE]))
        NZ <- nz - length(z.na.idx)

        # three possibilities:
        # - all z are missing  -> sigma.j.b.z = sigma.b, no between
        # - some z are missing -> update sigma.j.zz.{inv/logdet}
        # - z is complete      -> just use sigma.zz
        if(length(z.na.idx) == nz) {
            sigma.j.b.z <- sigma.b
        } else if(length(z.na.idx) > 0L) {
            delta.z <- Z.c[j, -z.na.idx, drop = TRUE]
            sigma.j.zz.inv <- lavaan:::lav_matrix_symmetric_inverse_update(
                    S.inv = sigma.zz.inv,
                    rm.idx = z.na.idx, logdet = TRUE,
                    S.logdet = sigma.zz.logdet)
            sigma.j.zz.logdet <- attr(sigma.j.zz.inv, "logdet")
            q.zz.a <- sum(colSums(delta.z * sigma.j.zz.inv) * delta.z)
            sigma.j.zi.zy <-
                sigma.j.zz.inv %*% sigma.zy[-z.na.idx, ,drop = FALSE]
            g.j <- drop(delta.z %*% sigma.j.zi.zy)
            sigma.j.b.z <- ( sigma.b -
                sigma.yz[, -z.na.idx, drop = FALSE] %*% sigma.j.zi.zy )
        } else { # no missings for z
            delta.z <- Z.c[j, , drop = TRUE]
            sigma.j.zz <- sigma.zz
            sigma.j.zz.inv <- sigma.zz.inv
            q.zz.a <- sum(colSums(delta.z * sigma.j.zz.inv) * delta.z)
            sigma.j.zz.logdet <- sigma.zz.logdet
            g.j <- drop(delta.z %*% sigma.zi.zy)
            sigma.j.b.z <- sigma.b.z
        }
    }

    # IBZA.j
    IBZA.j <- sigma.j.b.z %*% A.j
    IBZA.j[ny.diag.idx] <- IBZA.j[ny.diag.idx] + 1
    IBZA.j.inv <- solve(IBZA.j) # NOT symmetric!
    IBZA.j.logdet <- log(det(IBZA.j))
    IBZA.j.inv.BZ <- IBZA.j.inv %*% sigma.j.b.z

    # quadratic form
    if(nz > 0L && length(z.na.idx) != nz) {
        A.IBZA.j.inv <- A.j %*% IBZA.j.inv
        q.zz.b <-      sum(colSums(g.j      * A.IBZA.j.inv)  * g.j)
        q.zy   <- -1 * sum(colSums(p.j      * IBZA.j.inv)    * g.j)
        q.yy.b <-      sum(colSums(p.j      * IBZA.j.inv.BZ) * p.j)

        q.j <- ( q.yy.a - q.yy.b) + 2*q.zy + (q.zz.a + q.zz.b )
    } else {
       q.yy.b <- sum(colSums(p.j * IBZA.j.inv.BZ) * p.j)
       q.j <- (q.yy.a - q.yy.b)
    }

    # determinant
    if(nz > 0L && length(z.na.idx) != nz) {
        V.j.logdet <- W.logdet.j + IBZA.j.logdet + sigma.j.zz.logdet
    } else {
```

```
            V.j.logdet <- W.logdet.j + IBZA.j.logdet
        }

        if(loglik.) {
            P <- NY + NZ
            loglik[j] <- -(P * LOG.2PI + V.j.logdet + q.j)/2
        } else {
            loglik[j] <-  V.j.logdet + q.j
        }
    }

    out <- sum(loglik)

    out
}
```

Appendix C.6. The objective5.R File

```
# objective function 5
#
# use missing patterns for Y and Z (not per cluster, but for the complete data)

.obj5 <- function(mu.y = NULL, mu.z = NULL, sigma.w = NULL, sigma.b = NULL,
                  sigma.zz = NULL, sigma.yz = NULL, loglik. = FALSE) {

    LOG.2PI <- log(2 * pi)


    sigma.w.inv <- solve(sigma.w)
    sigma.w.logdet <- log(det(sigma.w))

    # y
    ny <- NCOL(sigma.w)
    ny.diag.idx <- lav_matrix_diag_idx(ny)
    Y1w <- Y1[, -between.idx, drop = FALSE]
    Y1w.c <- t( t(Y1w) - mu.y )

    # z
    nz <- length(between.idx)
    if(nz > 0L) {
        Z <- Y2[, between.idx, drop = FALSE]
        Z.c <- t( t(Z) - mu.z )
        sigma.zy <- t(sigma.yz)
        sigma.zz.inv <- solve(sigma.zz)
        sigma.zz.logdet <- log(det(sigma.zz))
        sigma.zi.zy <- sigma.zz.inv %*% sigma.zy
        sigma.b.z <- sigma.b - sigma.yz %*% sigma.zi.zy
    }

    # containers per cluster
    q.yy.b <- q.zy <- q.zz.b <- numeric(nclusters)
    IBZA.j.logdet <- numeric(nclusters)
    ALIST <- rep(list(matrix(0, ny, ny)), nclusters)

    # Z per missing pattern
    if(nz > 0L) {
        SIGMA.B.Z <- vector("list", length = Zp$npatterns + 1L) # +1 for empty
        ZPAT2J <- integer(nclusters) # which SIGMA.B.Z per cluster

        sigma.j.zz.logdet <- q.zz.a <- 0
        GJ <- matrix(0, nrow = nclusters, ncol = ny)
        for(p in seq_len(Zp$npatterns)) {
            freq <- Zp$freq[p]; na.idx <- which(!Zp$pat[p,])
            j.idx <- Zp$case.idx[[p]] # cluster indices with this pattern
            ZPAT2J[j.idx] <- p

            if(length(na.idx) > 0L) {
                zp <- sigma.zz[-na.idx, -na.idx, drop = FALSE]
```

```
51                  zp.inv <- lavaan:::lav_matrix_symmetric_inverse_update(
52                              S.inv = sigma.zz.inv, rm.idx = na.idx,
53                              logdet = TRUE, S.logdet = sigma.zz.logdet)
54                  zp.logdet <- attr(zp.inv, "logdet")
55                  sigma.j.zz.logdet <- sigma.j.zz.logdet + (zp.logdet*freq)
56
57                  this.z <- Z.c[Zp$case.idx[[p]], -na.idx, drop = FALSE]
58                  zi.delta.p <- this.z %*% zp.inv
59                  q.zz.a <- q.zz.a + sum(zi.delta.p * this.z)
60
61                  sigma.j.zi.zy <- zp.inv %*% sigma.zy[-na.idx, , drop = FALSE]
62                  SIGMA.B.Z[[p]] <- ( sigma.b -
63                      sigma.yz[, -na.idx, drop = FALSE] %*% sigma.j.zi.zy )
64                  GJ[j.idx, ] <- zi.delta.p %*% sigma.zy[-na.idx, , drop = FALSE]
65              } else {
66                  # complete case
67                  sigma.j.zz.logdet <-
68                      sigma.j.zz.logdet + (sigma.zz.logdet * freq)
69
70                  this.z <- Z.c[Zp$case.idx[[p]], , drop = FALSE]
71                  zi.delta.p <- this.z %*% sigma.zz.inv
72                  q.zz.a <- q.zz.a + sum(zi.delta.p * this.z)
73
74                  SIGMA.B.Z[[p]] <- sigma.b.z
75                  GJ[j.idx, ] <- this.z %*% sigma.zi.zy
76              }
77          } # p
78
79          # add empty patterns (if any)
80          if(length(Zp$empty.idx) > 0L) {
81              j.idx <- Zp$empty.idx
82              ZPAT2J[j.idx] <- p + 1L
83              SIGMA.B.Z[[p+1L]] <- sigma.b
84              GJ[j.idx, ] <- numeric(ny)
85          }
86
87      } else { # no between z
88          SIGMA.B.Z <- list(sigma.b)
89          ZPAT2J <- rep(1L, nclusters)
90      }
91
92
93      # Y per missing pattern
94      q.yy.a <- W.logdet <- 0
95      PJ <- matrix(0, nrow = nclusters, ncol = ny)
96      for(p in seq_len(Mp$npatterns)) {
97          # missing pattern info
98          freq <- Mp$freq[p]; na.idx <- which(!Mp$pat[p,])
99          j.idx  <- Mp$j.idx[[p]]; j1.idx <- Mp$j1.idx[[p]]
100         npatj <- integer(nclusters); npatj[j1.idx] <- Mp$j.freq[[p]]
101
102         # compute sigma.w.inv for this pattern
103         if(length(na.idx) > 0L) {
104             wp <- sigma.w[-na.idx, -na.idx, drop = FALSE]
105             wp.inv <- lavaan:::lav_matrix_symmetric_inverse_update(
106                         S.inv = sigma.w.inv, rm.idx = na.idx,
107                         logdet = TRUE, S.logdet = sigma.w.logdet)
108             wp.logdet <- attr(wp.inv, "logdet")
109             W.logdet <- W.logdet + (wp.logdet * freq)
110
111             this.y <- Y1w.c[Mp$case.idx[[p]], -na.idx, drop = FALSE]
112             wi.delta.p <- this.y %*% wp.inv
113             q.yy.a <- q.yy.a + sum(wi.delta.p * this.y)
114
115             PJ[j1.idx, -na.idx] <- ( PJ[j1.idx, -na.idx, drop = FALSE] +
116                         rowsum.default(wi.delta.p, j.idx, reorder = FALSE) )
117
118             A.j <- matrix(0, ny, ny)
119             A.j[-na.idx, -na.idx] <- wp.inv
```

```
            for(j in j1.idx) {
                ALIST[[j]] <- ALIST[[j]] + (A.j * npatj[j])
            }
        } else {
            # complete case
            W.logdet <- W.logdet + (sigma.w.logdet * freq)

            this.y <- Y1w.c[Mp$case.idx[[p]], , drop = FALSE]
            wi.delta.p <- this.y %*% sigma.w.inv
            q.yy.a <- q.yy.a + sum(wi.delta.p * this.y) # TR

            PJ[j1.idx, ] <- ( PJ[j1.idx, ,drop = FALSE] +
                     rowsum.default(wi.delta.p, j.idx, reorder = FALSE) )

            for(j in j1.idx) {
                ALIST[[j]] <- ALIST[[j]] + (sigma.w.inv * npatj[j])
            }
        }
    } # p

    # per cluster
    for(j in seq_len(nclusters)) {
        if(nz > 0L) {
            g.j <- GJ[j,]; p.j <- PJ[j,]

            sigma.j.b.z <- SIGMA.B.Z[[ ZPAT2J[j] ]]
            IBZA.j <- sigma.j.b.z %*% ALIST[[j]]
            IBZA.j[ny.diag.idx] <- IBZA.j[ny.diag.idx] + 1

            IBZA.j.inv.g  <- solve.default(IBZA.j, g.j)
            IBZA.j.inv.BZ <- solve.default(IBZA.j, sigma.j.b.z)
            tmp <- determinant.matrix(IBZA.j, logarithm = TRUE)
            IBZA.j.logdet[j] <- tmp$modulus * tmp$sign
            A.IBZA.j.inv.g  <- ALIST[[j]] %*% IBZA.j.inv.g

            q.zz.b[j] <-  sum(g.j * A.IBZA.j.inv.g)
            q.zy[j]   <- -sum(p.j * IBZA.j.inv.g)
            q.yy.b[j] <-  sum(p.j * (IBZA.j.inv.BZ %*% p.j))

        } else {
            p.j <- PJ[j,]
            IBZA.j <- sigma.b %*% ALIST[[j]]
            IBZA.j[ny.diag.idx] <- IBZA.j[ny.diag.idx] + 1
            IBZA.j.inv.B <- solve.default(IBZA.j, sigma.b)
            tmp <- determinant.matrix(IBZA.j, logarithm = TRUE)
            IBZA.j.logdet[j] <- tmp$modulus * tmp$sign

            q.yy.b[j] <- sum(p.j * (IBZA.j.inv.B %*% p.j))
        }
    }

    if(nz > 0L) {
        q.j <- (q.yy.a - sum(q.yy.b)) + 2*sum(q.zy) + (q.zz.a + sum(q.zz.b))
        LOGDET <- W.logdet + sum(IBZA.j.logdet) + sigma.j.zz.logdet
        if(loglik.) {
            P <- sum(!is.na(Y1w.c)) + sum(!is.na(Z.c))
            loglik <- -(P * LOG.2PI + LOGDET + q.j)/2
        } else {
            loglik <- q.j + LOGDET
        }
    } else {
        q.j <- (q.yy.a - sum(q.yy.b))
        LOGDET <- W.logdet + sum(IBZA.j.logdet)
        if(loglik.) {
            P <- sum(!is.na(Y1w.c))
            loglik <- -(P * LOG.2PI + LOGDET + q.j)/2
        } else {
            loglik <- q.j + LOGDET
        }
```

```
    }

    out <- loglik
    out
}
```

References

1. Rosseel, Y. lavaan: An R Package for Structural Equation Modeling. *J. Stat. Softw.* **2012**, *48*, 1–36. [CrossRef]
2. McDonald, R.P. A general model for two-level data with responses missing at random. *Psychometrika* **1993**, *58*, 575–585. [CrossRef]
3. Du Toit, S.H.; Du Toit, M. Multilevel structural equation modeling. In *Handbook of Multilevel Analysis*; de Leeuw, J., Meijer, E., Eds.; Springer: New York, NY, USA, 2008; Chapter 12, pp. 435–478.
4. Jöreskog, K.G.; Sörbom, D. *LISREL 8: User's Reference Guide*; Scientific Software International: Chicago, IL, USA, 1997.
5. Bentler, P.M.; Weeks, D.G. Linear Structural Equations with Latent-Variables. *Psychometrika* **1980**, *45*, 289–308. [CrossRef]
6. McArdle, J.J.; McDonald, R.P. Some algebraic properties of the Reticular Action Model for moment structures. *Br. J. Math. Stat. Psychol.* **1984**, *37*, 234–251. [CrossRef] [PubMed]
7. Browne, M.W.; Arminger, G. Specification and Estimation of Mean- and Covariance-Structure Models. In *Handbook of Statistical Modeling for the Social and Behavioral Sciences*; Arminger, G., Clogg, C.C., Sobel, M.E., Eds.; Plenum Press: New York, NY, USA, 1995; pp. 311–359.
8. Rubin, D.B. Inference and missing data. *Biometrika* **1976**, *63*, 581–592. [CrossRef]
9. Rockwood, N.J. Maximum Likelihood Estimation of Multilevel Structural Equation Models with Random Slopes for Latent Covariates. *Psychometrika* **2020**, *85*, 275–300. [CrossRef] [PubMed]
10. McDonald, R.P.; Goldstein, H. Balanced versus unbalanced designs for linear structural relations in two-level data. *Br. J. Math. Stat. Psychol.* **1989**, *42*, 215–232. [CrossRef]
11. Lee, S.Y. Multilevel analysis of structural equation models. *Biometrika* **1990**, *77*, 763–772. [CrossRef]
12. Muthén, B.O. Mean and covariance structure analysis of hierarchical data, 1990. In Proceedings of the Psychometric Society Meeting, Princeton, NJ, USA, 10 June 1990.
13. Jak, S.; Oort, F.J.; Dolan, C.V. A test for cluster bias: Detecting violations of measurement invariance across clusters in multilevel data. *Struct. Equ. Model. A Multidiscip. J.* **2013**, *20*, 265–282. [CrossRef]
14. Jak, S.; Jorgensen, T.D. Relating measurement invariance, cross-level invariance, and multilevel reliability. *Front. Psychol.* **2017**, *8*, 1640. [CrossRef] [PubMed]
15. Neale, M.C.; Hunter, M.D.; Pritikin, J.N.; Zahery, M.; Brick, T.R.; Kirkpatrick, R.M.; Estabrook, R.; Bates, T.C.; Maes, H.H.; Boker, S.M. OpenMx 2.0: Extended structural equation and statistical modeling. *Psychometrika* **2016**, *81*, 535–549. [CrossRef] [PubMed]
16. Henderson, H.V.; Searle, S.R. On deriving the inverse of a sum of matrices. *Siam Rev.* **1981**, *23*, 53–60. [CrossRef]
17. Harville, D.A. Maximum likelihood approaches to variance component estimation and to related problems. *J. Am. Stat. Assoc.* **1977**, *72*, 320–338. [CrossRef]
18. Harville, D.A. *Matrix Algebra from a Statistician's Perspective*; Springer: New York, NY, USA, 2008.