

A tutorial on estimating ‘true-score’ multitrait-multimethod models with lavaan in R

DL Oberski, Utrecht University

September 27, 2017

Abstract

This report gives a short description of the “true score multitrait-multimethod” (TS-MTMM) model, and demonstrates how this model can be estimated using R and the package lavaan. Using European Social Survey data from round 3, an example analysis is discussed and calculations show how estimates from the TS-MTMM model can be used to obtain estimates from “basic MTMM” models and vice versa.

Introduction

Download this Rmd and data used here:

- http://daob.nl/files/mtmm-lavaan/TS_mtmm_ESS3.Rmd
- http://daob.nl/files/mtmm-lavaan/ESS3_merged.rdata

Please cite this document if you find it useful!

Oberski, D.L. *A tutorial on estimating ‘true-score’ multitrait-multimethod models with lavaan in R*. DOI: <https://doi.org/10.5281/zenodo.999565> .

Thanks are due to Ting Yan for asking the question that led to this report.

Theory

First the theory.

The TS-MTMM model was formulated by Saris & Andrews (1991). It is mathematically equivalent to the regular MTMM model. However, in contrast with the regular MTMM, it directly yields estimates of the “true score reliability” and “true score validity”. These are interesting because they, in turn connect with the theory of Lord & Novick (1969). In this theory, every survey answer has some expectation, which is called the “true score” (TS). So the “true score” is not *literally* a true score (“Platonic true score” in Lord & Novick’s terms), but just the expectation of the answer:

$$\tau := E(y)$$

and

$$y := \tau + \epsilon$$

Note that this is just a definition (hence the $:=$) and you can’t really argue with it. The existence of the true score is not, repeat **not, an assumption**.

What the expectation might be over is a point of some contention. Most commonly, people ask you to imagine that you ask a question, wipe the person’s memory, and “immediately” ask it again. That way, the second time the answer will be different purely due to “measurement error” (things that aren’t interesting). So whatever you’d like to define as “measurement error” is what the expectation is over, and I’ll leave it there. There is more in the Lord & Novick book and Denny Borsboom wrote an interesting dissertation on these concepts.

So anyway,

Measurement error (ϵ) is just defined as whatever the current answer's deviation is from the average answer I'd give.

The TS might be biased, in the sense that each person will, for the same true stimulus, give a different answer: each person has a “response function” that determines how you respond *on average* to a question compared with other people, *for the same underlying feeling*. For example, mine might be $\tau_{\text{Daniel}} = \eta + 1$ and yours $\tau_{\text{Ting}} = \eta - 1$. That means I'm always biased upwards (by +1 relative to our average) and you downwards (by -1). We'll give different answers even though we had the same true underlying value, η . (The existence of an η is an assumption.) We'll call this personal bias ξ .

Now, this personal bias might happen on more than one question. For example, Krosnick showed that if you ask people completely unrelated agree-disagree questions, some people tend to “agree” with all of them. Even if they're contradictory or have no content at all. So that shows that the *same* ξ operates on different τ 's:

$$y_1 = \tau_1 + \epsilon_1 \tau_1 = \eta_1 + \xi$$

and

$$y_2 = \tau_2 + \epsilon_2 \tau_2 = \eta_2 + \xi$$

Where the ξ is the same (for example because they're both agree-disagree questions) but the η 's and τ 's differ (because they're on different topics).

Now back to “measurement error”. We'd like to know η but all we got was y . There are two reasons for that:

- Random measurement error ϵ and;
- Systematic (correlated) measurement error ξ .

Actually the name “systematic” is extremely confusing here because most people use that term to mean “bias in the average”. Here it does not mean that but rather “person-specific bias that's the same across questions”. Another term for ξ is “method factor” and η is called the “trait factor”.

So now we're ready to see what is meant by “true score reliability” and “true score validity” (again a super-confusing term but bear with me):

- True score reliability is the (squared) correlation between the true score and the observed answer, $\text{cor}(\tau, y)$;
- True score validity is the (squared) correlation between trait and the true score, $\text{cor}(\eta, \tau)$.

To separate these, the most straightforward method is just to formulate the model above directly as a structural equation model, e.g. in `lavaan`. Note that τ is a “phantom” latent variable here: it is just defined as trait PLUS method, without any further residual (unique variance). If we copy that in the syntax, we'll get the right correlations back as standardized loadings. This is shown below.

Data

I'm using the “enjoying life” items from ESS 3. The exact questions are in the main and supplementary questionnaire. You need to download both the main and supplementary data files and merge them. Anthony Damico wrote an R package that automatizes (part of) this: <http://asdfree.com/>.

We'll use `tidyverse` to munge data and `lavaan` to fit models.

```
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr
```

```
## Conflicts with tidy packages -----
```

```
## filter(): dplyr, stats
```

```
## lag(): dplyr, stats
```

```
library(lavaan)
```

```
## This is lavaan 0.5-22
```

```
## lavaan is BETA software! Please report any bugs.
```

First select the nine variables that form a part of the experiment. The experiment is also described in Revilla, Saris & Krosnick (2014, experiment 3).

```
load("ESS3_merged.rdata")
```

```
mtmm_sub <- ess3.mgd %>%
```

```
  select(idno, cntry, lrnnew, accdng, plprftr, testb7, testb8, testb9, testb19, testb20, testb21) %>%
```

```
  filter(cntry == "NL")
```

```
mtmm_sub <- mtmm_sub %>% mutate(idno = as.factor(idno)) %>% purrr::map_if(is.numeric, ~ .x - mean(.x, na.rm = TRUE))
```

Because I'm not allowed to distribute the full ESS dataset, I only provide the result of the above calls here:

```
mtmm_sub <- read.csv("mtmm_sub.csv")
```

```
head(mtmm_sub)
```

```
##      idno cntry      lrnnew      accdng      plprftr      testb7      testb8
## 1 3000031   NL -0.03563348 -0.3647192 -0.7845023         NA         NA
## 2 3000061   NL  0.96436652  0.6352808  0.2154977         NA         NA
## 3 3000101   NL -0.03563348 -0.3647192 -0.7845023  0.02568493  0.5910653
## 4 3000121   NL -1.03563348 -0.3647192  1.2154977 -0.97431507 -0.4089347
## 5 3000131   NL -0.03563348 -0.3647192  0.2154977 -0.97431507 -0.4089347
## 6 3000141   NL -0.03563348  0.6352808 -0.7845023  0.02568493 -0.4089347
##      testb9 testb19 testb20 testb21
## 1         NA      NA      NA      NA
## 2         NA      NA      NA      NA
## 3 -0.7247863      NA      NA      NA
## 4  0.2752137      NA      NA      NA
## 5  0.2752137      NA      NA      NA
## 6 -0.7247863      NA      NA      NA
```

I'm using only Dutch data here because the full dataset has a waiting time that's too long for my limited patience. But you're welcome to change this.

As an exploratory move, show the correlations:

```
mtmm_sub %>% select(-(1:2)) %>% cor(use = "pair") %>% round(2)
```

```
##      lrnnew accdng plprftr testb7 testb8 testb9 testb19 testb20 testb21
## lrnnew    1.00  0.25  0.20  0.63  0.20  0.19  -0.58  -0.30  -0.23
## accdng    0.25  1.00  0.21  0.24  0.45  0.17  -0.20  -0.42  -0.16
## plprftr   0.20  0.21  1.00  0.21  0.16  0.70  -0.11  -0.14  -0.61
## testb7    0.63  0.24  0.21  1.00  0.29  0.20      NA      NA      NA
## testb8    0.20  0.45  0.16  0.29  1.00  0.22      NA      NA      NA
## testb9    0.19  0.17  0.70  0.20  0.22  1.00      NA      NA      NA
## testb19  -0.58 -0.20 -0.11      NA      NA      NA      1.00  0.48  0.35
## testb20  -0.30 -0.42 -0.14      NA      NA      NA      0.48  1.00  0.34
## testb21  -0.23 -0.16 -0.61      NA      NA      NA      0.35  0.34  1.00
```

Note that some are NA, missing, because this is a split-ballot questionnaire: people only got the main questionnaire (version 1 of the questions) plus version 2 OR plus version 3. Nobody got both versions 2 and 3.

Models

Just traits model

A basic sanity check is to forget about the MTMM for a moment and check that a reasonable factor model results from just letting each of the different versions of the same question load on a single factor.

```
trait_basic <- "
  T1 =~ 1*lrnnew + testb7 + testb19
  T2 =~ 1*accdng + testb8 + testb20
  T3 =~ 1*plprftr+ testb9 + testb21
"
fit_trait_basic <- cfa(trait_basic, data = mtmm_sub, missing = "ml")

## Warning in lav_data_full(data = data, group = group, group.label =
## group.label, : lavaan WARNING: due to missing values, some pairwise
## combinations have less than 10% coverage

summary(fit_trait_basic, standardized = TRUE)
```

```
## lavaan (0.5-22) converged normally after 72 iterations
##
##   Number of observations                  1771
##
##   Number of missing patterns              12
##
##   Estimator                             ML
##   Minimum Function Test Statistic        108.185
##   Degrees of freedom                     24
##   P-value (Chi-square)                   0.000
##
## Parameter Estimates:
##
##   Information                        Observed
##   Standard Errors                    Standard
##
## Latent Variables:
##           Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##   T1 =~
##     lrnnew           1.000           0.552    0.699
##     testb7           1.304    0.094   13.913    0.000    0.720    0.892
##     testb19          -3.657    0.272  -13.441    0.000   -2.019   -0.893
##   T2 =~
##     accdng           1.000           0.424    0.548
##     testb8           1.365    0.139    9.830    0.000    0.579    0.747
##     testb20          -4.036    0.386  -10.443    0.000   -1.711   -0.875
##   T3 =~
##     plprftr           1.000           0.671    0.685
##     testb9           1.374    0.117   11.743    0.000    0.922    0.980
##     testb21          -3.623    0.334  -10.850    0.000   -2.432   -0.967
##
## Covariances:
```

```
##              Estimate Std.Err z-value P(>|z|) Std.lv Std.all
## T1 ~~
## T2          0.129    0.014   9.473   0.000   0.553   0.553
## T3          0.129    0.016   8.142   0.000   0.349   0.349
## T2 ~~
## T3          0.108    0.015   7.418   0.000   0.381   0.381
##
## Intercepts:
##              Estimate Std.Err z-value P(>|z|) Std.lv Std.all
## .lrnnew      0.001    0.019   0.053   0.958   0.001   0.001
## .testb7      0.015    0.028   0.545   0.586   0.015   0.019
## .testb19     0.037    0.080   0.462   0.644   0.037   0.016
## .accdng      0.002    0.018   0.099   0.921   0.002   0.002
## .testb8      0.003    0.030   0.116   0.908   0.003   0.004
## .testb20     0.032    0.073   0.434   0.665   0.032   0.016
## .plprftr     0.001    0.023   0.032   0.975   0.001   0.001
## .testb9      0.012    0.032   0.388   0.698   0.012   0.013
## .testb21     0.058    0.087   0.664   0.506   0.058   0.023
## T1           0.000             0.000   0.000
## T2           0.000             0.000   0.000
## T3           0.000             0.000   0.000
##
## Variances:
##              Estimate Std.Err z-value P(>|z|) Std.lv Std.all
## .lrnnew      0.319    0.021  14.995   0.000   0.319   0.511
## .testb7      0.133    0.037   3.607   0.000   0.133   0.204
## .testb19     1.034    0.301   3.431   0.001   1.034   0.202
## .accdng      0.419    0.021  20.381   0.000   0.419   0.700
## .testb8      0.265    0.041   6.412   0.000   0.265   0.442
## .testb20     0.892    0.280   3.183   0.001   0.892   0.234
## .plprftr     0.509    0.039  12.955   0.000   0.509   0.531
## .testb9      0.035    0.069   0.510   0.610   0.035   0.040
## .testb21     0.415    0.543   0.763   0.445   0.415   0.066
## T1           0.305    0.026  11.839   0.000   1.000   1.000
## T2           0.180    0.021   8.608   0.000   1.000   1.000
## T3           0.450    0.045  10.067   0.000   1.000   1.000
```

Looks fine to me. Note the third method gives negative loadings because the question answer options are reversed. You can see this in the correlation matrix too. This is just SEM taking care of reverse coding automatically. We can generally ignore signs.

(The 10% coverage warning is due to the missing correlations. It can also be safely ignored in this case.)

Run-of-the mill MTMM-1

Below I show how to fit an MTMM that **not** a true-score MTMM. This is the standard thing most people do when they MTMM. I've left out the second method (so it's MTMM-1, see Eid 2000).

```
mtmm_basic <- "
  T1 =~ lrnnew + testb7 + testb19
  T2 =~ accdng + testb8 + testb20
  T3 =~ plprftr+ testb9 + testb21

  M1 =~ 1*lrnnew + 1*accdng + 1*plprftr
  M3 =~ 1*testb19 + 1*testb20 + 1*testb21
```

```

T1~~1*T1
T2~~1*T2
T3~~1*T3

T1~~T2+T3
T2~~T3
"

fit_mtm_basic <- lavaan(mtm_basic, data = mtmm_sub, missing = "ml",
                        auto.fix.first = FALSE, auto.var = TRUE)

```

```

## Warning in lav_data_full(data = data, group = group, group.label =
## group.label, : lavaan WARNING: due to missing values, some pairwise
## combinations have less than 10% coverage

```

```
summary(fit_mtm_basic, standardized = TRUE)
```

```

## lavaan (0.5-22) converged normally after 66 iterations
##
##   Number of observations              1771
##
##   Number of missing patterns          12
##
##   Estimator                          ML
##   Minimum Function Test Statistic    25.967
##   Degrees of freedom                 31
##   P-value (Chi-square)               0.723
##
## Parameter Estimates:
##
##   Information                        Observed
##   Standard Errors                   Standard
##
## Latent Variables:
##
##           Estimate  Std.Err  z-value  P(>|z|)  Std.lv  Std.all
##   T1 =~
##     lrnnew          0.585    0.029   20.292   0.000    0.585    0.739
##     testb7          0.708    0.041   17.421   0.000    0.708    0.878
##     testb19        -1.704    0.115  -14.876   0.000   -1.704   -0.773
##   T2 =~
##     accdng          0.470    0.032   14.912   0.000    0.470    0.608
##     testb8          0.585    0.047   12.550   0.000    0.585    0.758
##     testb20        -1.382    0.119  -11.665   0.000   -1.382   -0.714
##   T3 =~
##     plprftr         0.732    0.043   17.115   0.000    0.732    0.748
##     testb9          0.871    0.053   16.358   0.000    0.871    0.927
##     testb21        -2.119    0.147  -14.449   0.000   -2.119   -0.845
##   M1 =~
##     lrnnew          1.000                0.168    0.212
##     accdng          1.000                0.168    0.217
##     plprftr         1.000                0.168    0.171
##   M3 =~
##     testb19          1.000                0.891    0.404
##     testb20          1.000                0.891    0.460

```

```

##      testb21          1.000          0.891    0.355
##
## Covariances:
##      Estimate Std.Err z-value P(>|z|) Std.lv Std.all
##      T1 ~~
##      T2          0.475    0.034   13.931   0.000    0.475    0.475
##      T3          0.302    0.032    9.601   0.000    0.302    0.302
##      T2 ~~
##      T3          0.327    0.035    9.223   0.000    0.327    0.327
##
## Intercepts:
##      Estimate Std.Err z-value P(>|z|) Std.lv Std.all
##      .lrnnew      0.000          0.000    0.000
##      .testb7      0.000          0.000    0.000
##      .testb19     0.000          0.000    0.000
##      .accdng      0.000          0.000    0.000
##      .testb8      0.000          0.000    0.000
##      .testb20     0.000          0.000    0.000
##      .plprftr     0.000          0.000    0.000
##      .testb9      0.000          0.000    0.000
##      .testb21     0.000          0.000    0.000
##      T1          0.000          0.000    0.000
##      T2          0.000          0.000    0.000
##      T3          0.000          0.000    0.000
##      M1          0.000          0.000    0.000
##      M3          0.000          0.000    0.000
##
## Variances:
##      Estimate Std.Err z-value P(>|z|) Std.lv Std.all
##      T1          1.000          1.000    1.000
##      T2          1.000          1.000    1.000
##      T3          1.000          1.000    1.000
##      .lrnnew     0.256    0.027    9.571   0.000    0.256    0.409
##      .testb7     0.150    0.043    3.504   0.000    0.150    0.230
##      .testb19     1.160    0.257    4.509   0.000    1.160    0.239
##      .accdng     0.349    0.025   13.933   0.000    0.349    0.583
##      .testb8     0.254    0.044    5.726   0.000    0.254    0.426
##      .testb20     1.040    0.231    4.509   0.000    1.040    0.278
##      .plprftr     0.393    0.053    7.413   0.000    0.393    0.411
##      .testb9     0.125    0.077    1.609   0.108    0.125    0.141
##      .testb21     1.006    0.468    2.150   0.032    1.006    0.160
##      M1          0.028    0.010    2.718   0.007    1.000    1.000
##      M3          0.793    0.137    5.771   0.000    1.000    1.000

```

The reason for leaving out the second method is that when including this method, its variance is estimated to be close to zero. Revilla & Saris (2013) wrote a paper on this and other issues with split-ballot MTMMs.

True score MTMM-1

Here is the true-score MTMM model. The first part defines the observed variables as equal to true score (e.g. `lrnnew_TS =~ 1*lrnnew`), except for random error variance (e.g. `lrnnew~~lrnnew`). The second part defines each true score to exactly equal trait PLUS method. E.g. `T1 =~ lrnnew_TS` and `M1 =~ 1*lrnnew_TS` and there's no further variance (`auto.var = FALSE` in the call below). The methods don't correlate but the traits do.

```

mtmm_ts <- "
  lrnnew_TS =~ 1*lrnnew
  accdng_TS =~ 1*accdng
  plprftr_TS =~ 1*plprftr
  testb7_TS =~ 1*testb7
  testb8_TS =~ 1*testb8
  testb9_TS =~ 1*testb9
  testb19_TS =~ 1*testb19
  testb20_TS =~ 1*testb20
  testb21_TS =~ 1*testb21

  T1 =~ lrnnew_TS + testb7_TS + testb19_TS
  T2 =~ accdng_TS + testb8_TS + testb20_TS
  T3 =~ plprftr_TS + testb9_TS + testb21_TS

  M1 =~ 1*lrnnew_TS + 1*accdng_TS + 1*plprftr_TS
  M3 =~ 1*testb19_TS + 1*testb20_TS + 1*testb21_TS

  lrnnew~~lrnnew
  accdng~~accdng
  plprftr~~plprftr
  testb7 ~~testb7
  testb8 ~~testb8
  testb9 ~~testb9
  testb19~~testb19
  testb20~~testb20
  testb21~~testb21

  M1~~M1
  M3~~M3

  T1~~1*T1
  T2~~1*T2
  T3~~1*T3

  T1~~T2+T3
  T2~~T3
"

fit_mtmm_ts <- lavaan(mtmm_ts, data = mtmm_sub, missing = "ml",
  auto.fix.first = FALSE, auto.var = FALSE)

```

```

## Warning in lav_data_full(data = data, group = group, group.label =
## group.label, : lavaan WARNING: due to missing values, some pairwise
## combinations have less than 10% coverage

```

The model estimates are shown below. Note they're different than for the run-of-the-mill MTMM. But the model fit and df are the same, demonstrating mathematical equivalence:

```

fit_mtmm_ts

## lavaan (0.5-22) converged normally after 66 iterations
##
##   Number of observations              1771
##

```



```
## Number of missing patterns      12
##
## Estimator                       ML
## Minimum Function Test Statistic 25.967
## Degrees of freedom              31
## P-value (Chi-square)            0.723
```

```
fit_mtm-basic
```

```
## lavaan (0.5-22) converged normally after 66 iterations
##
## Number of observations          1771
##
## Number of missing patterns      12
##
## Estimator                       ML
## Minimum Function Test Statistic 25.967
## Degrees of freedom              31
## P-value (Chi-square)            0.723
```

As per usual, the “true score validity” (don’t get me started on this term) estimates are pretty high, e.g. 0.961 for `lrnnew` and 1 by definition for the second method, but the reliabilities are around 0.8, e.g. standardized `lrnnew_TS` \sim `lrnnew` is 0.769. This is typical in ESS.

```
summary(fit_mtm-ts, standardized = TRUE)
```

```
## lavaan (0.5-22) converged normally after 66 iterations
##
## Number of observations          1771
##
## Number of missing patterns      12
##
## Estimator                       ML
## Minimum Function Test Statistic 25.967
## Degrees of freedom              31
## P-value (Chi-square)            0.723
##
## Parameter Estimates:
##
## Information                      Observed
## Standard Errors                  Standard
##
## Latent Variables:
##      Estimate Std.Err z-value P(>|z|) Std.lv Std.all
## lrnnew_TS =~
##   lrnnew      1.000              0.608  0.769
## accdng_TS =~
##   accdng      1.000              0.499  0.646
## plprftr_TS =~
##   plprftr     1.000              0.751  0.768
## testb7_TS =~
##   testb7      1.000              0.708  0.878
## testb8_TS =~
##   testb8      1.000              0.585  0.758
## testb9_TS =~
##   testb9      1.000              0.871  0.927
```

```

## testb19_TS =~
## testb19          1.000          1.923    0.873
## testb20_TS =~
## testb20          1.000          1.644    0.850
## testb21_TS =~
## testb21          1.000          2.299    0.917
## T1 =~
## lrnnew_TS        0.585    0.029   20.292    0.000    0.961    0.961
## testb7_TS         0.708    0.041   17.421    0.000    1.000    1.000
## testb19_TS       -1.704    0.115  -14.876    0.000   -0.886   -0.886
## T2 =~
## accdng_TS         0.470    0.032   14.912    0.000    0.942    0.942
## testb8_TS         0.585    0.047   12.550    0.000    1.000    1.000
## testb20_TS       -1.382    0.119  -11.665    0.000   -0.841   -0.841
## T3 =~
## plprftr_TS        0.732    0.043   17.115    0.000    0.975    0.975
## testb9_TS         0.871    0.053   16.358    0.000    1.000    1.000
## testb21_TS       -2.119    0.147  -14.449    0.000   -0.922   -0.922
## M1 =~
## lrnnew_TS         1.000          0.276    0.276
## accdng_TS         1.000          0.336    0.336
## plprftr_TS        1.000          0.223    0.223
## M3 =~
## testb19_TS        1.000          0.463    0.463
## testb20_TS        1.000          0.542    0.542
## testb21_TS        1.000          0.387    0.387
##
## Covariances:
##          Estimate Std.Err z-value P(>|z|) Std.lv Std.all
## T1 ~~
## T2          0.475    0.034   13.931    0.000    0.475    0.475
## T3          0.302    0.032    9.601    0.000    0.302    0.302
## T2 ~~
## T3          0.327    0.035    9.223    0.000    0.327    0.327
##
## Intercepts:
##          Estimate Std.Err z-value P(>|z|) Std.lv Std.all
## .lrnnew        0.000          0.000    0.000
## .accdng         0.000          0.000    0.000
## .plprftr        0.000          0.000    0.000
## .testb7         0.000          0.000    0.000
## .testb8         0.000          0.000    0.000
## .testb9         0.000          0.000    0.000
## .testb19        0.000          0.000    0.000
## .testb20        0.000          0.000    0.000
## .testb21        0.000          0.000    0.000
## lrnnew_TS       0.000          0.000    0.000
## accdng_TS       0.000          0.000    0.000
## plprftr_TS      0.000          0.000    0.000
## testb7_TS       0.000          0.000    0.000
## testb8_TS       0.000          0.000    0.000
## testb9_TS       0.000          0.000    0.000
## testb19_TS      0.000          0.000    0.000
## testb20_TS      0.000          0.000    0.000

```

```
##      testb21_TS      0.000      0.000      0.000
##      T1              0.000      0.000      0.000
##      T2              0.000      0.000      0.000
##      T3              0.000      0.000      0.000
##      M1              0.000      0.000      0.000
##      M3              0.000      0.000      0.000
##
## Variances:
##      Estimate Std.Err z-value P(>|z|) Std.lv Std.all
##      .lrnnew   0.256   0.027   9.571   0.000   0.256   0.409
##      .accdng   0.349   0.025  13.933   0.000   0.349   0.583
##      .plprftr  0.393   0.053   7.413   0.000   0.393   0.411
##      .testb7   0.150   0.043   3.504   0.000   0.150   0.230
##      .testb8   0.254   0.044   5.726   0.000   0.254   0.426
##      .testb9   0.125   0.077   1.609   0.108   0.125   0.141
##      .testb19  1.160   0.257   4.509   0.000   1.160   0.239
##      .testb20  1.040   0.231   4.509   0.000   1.040   0.278
##      .testb21  1.006   0.468   2.150   0.032   1.006   0.160
##      M1        0.028   0.010   2.718   0.007   1.000   1.000
##      M3        0.793   0.137   5.771   0.000   1.000   1.000
##      T1        1.000      0.000      0.000   1.000   1.000
##      T2        1.000      0.000      0.000   1.000   1.000
##      T3        1.000      0.000      0.000   1.000   1.000
##      lrnnew_TS  0.000      0.000      0.000   0.000   0.000
##      accdng_TS  0.000      0.000      0.000   0.000   0.000
##      plprftr_TS 0.000      0.000      0.000   0.000   0.000
##      testb7_TS  0.000      0.000      0.000   0.000   0.000
##      testb8_TS  0.000      0.000      0.000   0.000   0.000
##      testb9_TS  0.000      0.000      0.000   0.000   0.000
##      testb19_TS 0.000      0.000      0.000   0.000   0.000
##      testb20_TS 0.000      0.000      0.000   0.000   0.000
##      testb21_TS 0.000      0.000      0.000   0.000   0.000
```

The reliability coefficients, validity coefficients, and method effects are above and also here:

```
std_ts <- standardizedsolution(fit_mtmms) %>% filter(op == "~") %>% select(1:4)
std_ts
```

```
##      lhs op      rhs      est.std
## 1  lrnnew_TS =~ lrnnew 0.7689815
## 2  accdng_TS =~ accdng 0.6458489
## 3  plprftr_TS =~ plprftr 0.7675850
## 4  testb7_TS =~ testb7 0.8777321
## 5  testb8_TS =~ testb8 0.7576548
## 6  testb9_TS =~ testb9 0.9268046
## 7  testb19_TS =~ testb19 0.8725297
## 8  testb20_TS =~ testb20 0.8498356
## 9  testb21_TS =~ testb21 0.9165255
## 10 T1 =~ lrnnew_TS 0.9612032
## 11 T1 =~ testb7_TS 1.0000000
## 12 T1 =~ testb19_TS -0.8862848
## 13 T2 =~ accdng_TS 0.9418774
## 14 T2 =~ testb8_TS 1.0000000
## 15 T2 =~ testb20_TS -0.8406231
## 16 T3 =~ plprftr_TS 0.9747377
```

```
## 17      T3 =~ testb9_TS  1.0000000
## 18      T3 =~ testb21_TS -0.9218924
## 19      M1 =~ lrnnew_TS  0.2758413
## 20      M1 =~ accdng_TS  0.3359567
## 21      M1 =~ plprftr_TS 0.2233529
## 22      M3 =~ testb19_TS 0.4631407
## 23      M3 =~ testb20_TS 0.5416206
## 24      M3 =~ testb21_TS 0.3874459

std_basic <- standardizedsolution(fit_mtm-basic) %>%
  filter(op == "~") %>% select(1:4)
std_basic
```

```
##    lhs op    rhs    est.std
## 1   T1 =~ lrnnew 0.7391475
## 2   T1 =~ testb7 0.8777321
## 3   T1 =~ testb19 -0.7733098
## 4   T2 =~ accdng 0.6083105
## 5   T2 =~ testb8 0.7576548
## 6   T2 =~ testb20 -0.7143914
## 7   T3 =~ plprftr 0.7481940
## 8   T3 =~ testb9 0.9268046
## 9   T3 =~ testb21 -0.8449379
## 10  M1 =~ lrnnew 0.2121169
## 11  M1 =~ accdng 0.2169773
## 12  M1 =~ plprftr 0.1714423
## 13  M3 =~ testb19 0.4041040
## 14  M3 =~ testb20 0.4602884
## 15  M3 =~ testb21 0.3551041
```

Calculating standardized coefficients in one model from the other

The standardized coefficients of one model can be calculated from the other. For example, starting from the TS solution, we get the “basic” trait and method loadings by multiplying reliability with validity and method effect, respectively. Here’s an example for the `lrnnew` variable (T1 and M1):

```
reliability_coefficient <- std_ts$est.std[std_ts$rhs == "lrnnew"]

val_and_met_coefficients <- std_ts$est.std[std_ts$rhs == "lrnnew_TS"]

reliability_coefficient * val_and_met_coefficients

## [1] 0.7391475 0.2121169
```

This can be verified by looking at the standardized solution of the “basic” model directly:

```
std_basic %>% filter(rhs == "lrnnew")

##    lhs op    rhs    est.std
## 1   T1 =~ lrnnew 0.7391475
## 2   M1 =~ lrnnew 0.2121169
```

Note these are identical even though obtained from different models.

Of course, we can also reverse the calculations: starting from the “basic” solution, calculate the standardized coefficients of the TS model without actually estimating that model. Calling the standardized trait and method factor loading from the basic model λ and γ respectively, we have $\lambda = r \cdot v$, $\gamma = r \cdot m$, and since τ

doesn't contain any unique variance, $m^2 + v^2 = 1$, implying that

$$r = \sqrt{\lambda^2 + \gamma^2}, v = \lambda/r, \text{ and } m = \gamma/r.$$

```
basic_coefs <- std_basic %>% filter(rhs == "lrnnew") %>% .$est.std
lambda <- basic_coefs[1]
gamma <- basic_coefs[2]

r <- sqrt(lambda^2 + gamma^2)
v <- lambda / r
m <- gamma / r

c(r=r, v=v, m=m)
```

```
##           r           v           m
## 0.7689815 0.9612032 0.2758413
```

which are indeed identical to `reliability_coefficient` and `val_and_met_coefficients` obtained from the basic model:

```
reliability_coefficient
```

```
## [1] 0.7689815
```

```
val_and_met_coefficients
```

```
## [1] 0.9612032 0.2758413
```