lavaan note: equality constraints

Yves Rosseel

April 4, 2015

1 Introduction

- in version 0.5-17 (and lower), there were two types of equality constraints in lavaan:
  1. simple equality constraints (usually specified in the model syntax by using the same label for multiple parameters, or in the multiple group setting, by using the `group.equal` argument)
  2. general (linear or nonlinear) equality constraints, explicitly specified by using the '`==`' operator
- the two types of constraints were handled in a very different way. We will provide the details below. In version 0.5-18, we make another distinction:
  1. linear equality constraints
  2. non-linear equality constraints
- again, linear equality constraints are handled in a very different manner than non-linear equality constraints. But for the linear constraints, we no longer make the distinction between simple equality constraints and more general (but linear) equality constraints
- this note documents the implementational details of both the old (0.5-17) and the new (0.5-18) approach

2 Equality constraints in 0.5-17

2.1 Approach 1: implicit constraints using labels

- we will use the Holzinger & Swineford 3-factor CFA example to illustrate the details; consider the following model:

```
HS.model <- '
    visual  =~ x1 + a*x2 + a*x3
    textual =~ x4 + b*x5 + b*x6
    speed   =~ x7 + c*x8 + c*x9
'
fit <- cfa(HS.model, data=HolzingerSwineford1939
```

- this model took about 0.185 seconds to run
- have a look at the parameter table:

```
> parTable(fit)
      id     lhs op     rhs user group free ustart exo label eq.id unco
   1   1  visual =~      x1    1     1    0      1   0           0    0
   2   2  visual =~      x2    1     1    1     NA   0     a     2    1
   3   3  visual =~      x3    1     1    1     NA   0     a     2    2
   4   4 textual =~      x4    1     1    0      1   0           0    0
   5   5 textual =~      x5    1     1    2     NA   0     b     5    3
   6   6 textual =~      x6    1     1    2     NA   0     b     5    4
   7   7   speed =~      x7    1     1    0      1   0           0    0
   8   8   speed =~      x8    1     1    3     NA   0     c     8    5
   9   9   speed =~      x9    1     1    3     NA   0     c     8    6
   10 10      x1 ~~      x1    0     1    4     NA   0           0    7
   11 11      x2 ~~      x2    0     1    5     NA   0           0    8
   12 12      x3 ~~      x3    0     1    6     NA   0           0    9
   13 13      x4 ~~      x4    0     1    7     NA   0           0   10
   14 14      x5 ~~      x5    0     1    8     NA   0           0   11
   15 15      x6 ~~      x6    0     1    9     NA   0           0   12
   16 16      x7 ~~      x7    0     1   10     NA   0           0   13
   17 17      x8 ~~      x8    0     1   11     NA   0           0   14
   18 18      x9 ~~      x9    0     1   12     NA   0           0   15
   19 19  visual ~~  visual    0     1   13     NA   0           0   16
   20 20 textual ~~ textual    0     1   14     NA   0           0   17
```

```
   21 21   speed ~~   speed    0     1   15     NA   0           0   18
   22 22  visual ~~ textual    0     1   16     NA   0           0   19
   23 23  visual ~~   speed    0     1   17     NA   0           0   20
   24 24 textual ~~   speed    0     1   18     NA   0           0   21
```

- the nonzero elements in the 'free' column are the free model parameters (18 in this example); some numbers (1, 2 and 3) are duplicated, reflecting the fact that these parameters are one and the same; the nonzero elements in the 'unco' column are the free model parameters if we ignore the equality constraints; the 'eq.id' column contains the parameter 'id' numbers (1st column) of those parameters that were constrained to be equal to each other

- as far as lavaan is concerned, there are only 18 free parameters in this model, as reflected in the output of `coef()` and `vcov()`:

```
> coef(fit)
              a               b               c           x1~~x1
          0.652           1.004           1.156           0.552
         x2~~x2          x3~~x3          x4~~x4          x5~~x5
          1.113           0.875           0.380           0.514
         x6~~x6          x7~~x7          x8~~x8          x9~~x9
          0.321           0.816           0.520           0.534
  visual~~visual textual~~textual   speed~~speed visual~~textual
          0.806           0.971           0.368           0.420
   visual~~speed   textual~~speed
          0.265           0.175
> dim( vcov(fit) )
[1] 18 18
```

- this implies that we can use unconstrained optimization (with 18 free parameters), and there is no need to use more sophisticated algorithms for constrained optimization

- internally, we often need to convert between the unconstrained 21-parameter vector, and the constrained 18-parameter vector; this is done using a (manually constructed) 21×18 indicator matrix stored in `fit@Model@eq.constraints.K`:

```
> fit@Model@eq.constraints.K
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
 [1,]    1    0    0    0    0    0    0    0    0     0     0     0     0
 [2,]    1    0    0    0    0    0    0    0    0     0     0     0     0
 [3,]    0    1    0    0    0    0    0    0    0     0     0     0     0
 [4,]    0    1    0    0    0    0    0    0    0     0     0     0     0
 [5,]    0    0    1    0    0    0    0    0    0     0     0     0     0
 [6,]    0    0    1    0    0    0    0    0    0     0     0     0     0
 [7,]    0    0    0    1    0    0    0    0    0     0     0     0     0
 [8,]    0    0    0    0    1    0    0    0    0     0     0     0     0
 [9,]    0    0    0    0    0    1    0    0    0     0     0     0     0
[10,]    0    0    0    0    0    0    1    0    0     0     0     0     0
[11,]    0    0    0    0    0    0    0    1    0     0     0     0     0
[12,]    0    0    0    0    0    0    0    0    1     0     0     0     0
[13,]    0    0    0    0    0    0    0    0    0     1     0     0     0
[14,]    0    0    0    0    0    0    0    0    0     0     1     0     0
[15,]    0    0    0    0    0    0    0    0    0     0     0     1     0
[16,]    0    0    0    0    0    0    0    0    0     0     0     0     1
[17,]    0    0    0    0    0    0    0    0    0     0     0     0     0
[18,]    0    0    0    0    0    0    0    0    0     0     0     0     0
[19,]    0    0    0    0    0    0    0    0    0     0     0     0     0
[20,]    0    0    0    0    0    0    0    0    0     0     0     0     0
[21,]    0    0    0    0    0    0    0    0    0     0     0     0     0
      [,14] [,15] [,16] [,17] [,18]
 [1,]     0     0     0     0     0
 [2,]     0     0     0     0     0
 [3,]     0     0     0     0     0
 [4,]     0     0     0     0     0
 [5,]     0     0     0     0     0
 [6,]     0     0     0     0     0
 [7,]     0     0     0     0     0
 [8,]     0     0     0     0     0
 [9,]     0     0     0     0     0
[10,]     0     0     0     0     0
[11,]     0     0     0     0     0
[12,]     0     0     0     0     0
[13,]     0     0     0     0     0
[14,]     0     0     0     0     0
[15,]     0     0     0     0     0
[16,]     0     0     0     0     0
[17,]     1     0     0     0     0
[18,]     0     1     0     0     0
[19,]     0     0     1     0     0
[20,]     0     0     0     1     0
[21,]     0     0     0     0     1
```

- there are only a few places in the lavaan code where this matrix is used:

1. to convert the unconstrained (21-parameter) gradient vector to a constrained (18-parameter) gradient vector:

```
# lav_model_gradient.R: line 138
dx <- as.numeric( t(lavmodel@eq.constraints.K) %*% dx )
```

2. to convert/reduce the columns of the 'Delta' matrix (the Jacobian matrix $\partial s(\theta)/\partial\theta'$, where $s(\cdot)$ is a function returning the model-implied sample statistics, and θ is the parameter vector)

```
# lav_model_gradient.R: line 519
Delta.group <- Delta.group %*% lavmodel@eq.constraints.K
```

3. to handle the equality constraints when computing standardized parameter values

```
# lav_object_methods.R: line 727
JAC <- JAC %*% object@Model@eq.constraints.K
```

- the first two are related to the computation of the (analytical) gradient, and as a result, the various information matrices (expected, observed or first.order) always have the dimension 18×18; for example

```
> E <- lavaan:::computeExpectedInformation(lavmodel = fit@Model, lavsamplestats = fit@SampleStats)
> dim(E)
[1] 18 18
```

2.2 Approach 2: explicit constraints using the '==' operator

- consider the following syntax, where we use an alternative way to impose the equality constraints:

```
HS.model <- '
    visual  =~ x1 + a1*x2 + a2*x3
    textual =~ x4 + b1*x5 + b2*x6
    speed   =~ x7 + c1*x8 + c2*x9

    a1 == a2
    b1 == b2
    c1 == c2
'
fit <- cfa(HS.model, data=HolzingerSwineford1939)
```

- this model took 0.259 seconds to run; however, the fit, and all estimated parameter estimates are identical to the estimates obtained in the model fit from the 1st approach

- this is the parameter table:

```
> parTable(fit)
   id     lhs op     rhs user group free ustart exo label eq.id unco
1   1  visual =~      x1    1     1    0      1   0            0    0
2   2  visual =~      x2    1     1    1     NA   0    a1      0    1
3   3  visual =~      x3    1     1    2     NA   0    a2      0    2
4   4 textual =~      x4    1     1    0      1   0            0    0
5   5 textual =~      x5    1     1    3     NA   0    b1      0    3
6   6 textual =~      x6    1     1    4     NA   0    b2      0    4
7   7   speed =~      x7    1     1    0      1   0            0    0
8   8   speed =~      x8    1     1    5     NA   0    c1      0    5
9   9   speed =~      x9    1     1    6     NA   0    c2      0    6
10 10      x1 ~~      x1    0     1    7     NA   0            0    7
11 11      x2 ~~      x2    0     1    8     NA   0            0    8
12 12      x3 ~~      x3    0     1    9     NA   0            0    9
13 13      x4 ~~      x4    0     1   10     NA   0            0   10
14 14      x5 ~~      x5    0     1   11     NA   0            0   11
15 15      x6 ~~      x6    0     1   12     NA   0            0   12
16 16      x7 ~~      x7    0     1   13     NA   0            0   13
17 17      x8 ~~      x8    0     1   14     NA   0            0   14
18 18      x9 ~~      x9    0     1   15     NA   0            0   15
19 19  visual ~~  visual    0     1   16     NA   0            0   16
20 20 textual ~~ textual    0     1   17     NA   0            0   17
21 21   speed ~~   speed    0     1   18     NA   0            0   18
22 22  visual ~~ textual    0     1   19     NA   0            0   19
23 23  visual ~~   speed    0     1   20     NA   0            0   20
24 24 textual ~~   speed    0     1   21     NA   0            0   21
25 25      a1 ==      a2    1     0    0     NA   0            0    0
26 26      b1 ==      b2    1     0    0     NA   0            0    0
27 27      c1 ==      c2    1     0    0     NA   0            0    0
```

- here, we have 21 'free' parameters (and the free and unco columns are identical); the 'eq.id' column is empty, but we have three additional rows in the parameter table reflecting the user-specified equality constraints

- unlike the previous setting, lavaan will now use constrained optimization, which can take (much) longer

- the output of `coef()` and `vcov()` now reflects the fact that we have 21 (instead of 18) parameters, although they do take the constraints into account

```
> coef(fit)
              a1               a2               b1               b2
           0.652            0.652            1.004            1.004
              c1               c2            x1~~x1           x2~~x2
           1.156            1.156            0.552            1.113
           x3~~x3           x4~~x4           x5~~x5           x6~~x6
           0.875            0.380            0.514            0.321
           x7~~x7           x8~~x8           x9~~x9   visual~~visual
           0.816            0.520            0.534            0.806
textual~~textual     speed~~speed  visual~~textual    visual~~speed
           0.971            0.368            0.420            0.265
  textual~~speed
           0.175
> dim( vcov(fit) )
[1] 21 21
```

- there is no matrix `fit@Model@eq.constraints.K`, but lavaan has contructed a 'constraint function' (say, $a(\theta)$ where θ is the unconstrained parameter vector) to evaluate the (three) equality constraints:

```
> fit@Model@ceq.function
function (x, ...)
{
    out <- rep(NA, 3)
    a1 = x[1]
    a2 = x[2]
    out[1] = a1 - (a2)
    b1 = x[3]
    b2 = x[4]
    out[2] = b1 - (b2)
    c1 = x[5]
    c2 = x[6]
    out[3] = c1 - (c2)
    out[is.na(out)] <- Inf
    return(out)
}
```

- the input of this constraint function is the full (21-parameter) vector of free model parameters; for each constraint, the output should be (close to) zero if the constraint is satisfied; the Jacobian of this constraint function $A = \partial a(\theta)/\partial \theta'$ gives the constraint matrix (A):

```
> A <- fit@Model@con.jac[,]
> A
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]    1   -1    0    0    0    0    0    0    0     0     0     0     0     0
[2,]    0    0    1   -1    0    0    0    0    0     0     0     0     0     0
[3,]    0    0    0    0    1   -1    0    0    0     0     0     0     0     0
     [,15] [,16] [,17] [,18] [,19] [,20] [,21]
[1,]     0     0     0     0     0     0     0
[2,]     0     0     0     0     0     0     0
[3,]     0     0     0     0     0     0     0
```

- only if the constraints are linear, this matrix will contain constants; if the constraints are nonlinear, the elements in this matrix will be a function of the (current) parameter values (and they will change from iteration to iteration)

- the information matrices (observed, first.order or expected) all have dimension 21×21; but they do NOT reflect the equality constraints yet; for example

```
> E <- lavaan:::computeExpectedInformation(fit@Model, lavsamplestats=fit@SampleStats)
> dim(E)
[1] 21 21
> qr(E)$rank
[1] 21
```

- here, the rank is 21 because the constraints are not needed to identify the model; if the constraints are needed to identify the model, the rank will be lower than 21

- when we need to invert the information matrix, we first 'augment' the rows and the columns of this matrix with the A matrix, resulting in a so-called 'bordered information matrix' (of dimension 24×24); then we take the (generalized) inverse of this bordered information matrix; and finally, we again remove the extra borders, and only the retain the first 21 rows and columns to get our 'inverted information matrix'; to illustrate:

```
> E.augment <- rbind( cbind(E, t(A)),
                      cbind(A, matrix(0,nrow(A),nrow(A))) )
> E.augment.inv <- solve(E.augment)
> E.inv <- E.augment.inv[1:21, 1:21]
```

- the real code is slightly more complicated (also taking into account possible inequality constraints), and can be found for example in the file `lav_vcov.R` around the lines 22–42

2.3 The problem

- the problem with lavaan 0.5-17 (and lower) is that it uses two (very different) approaches to handle equality constraints
- when the two approaches are combined, this leads to problems, for example (in 0.5-17):

```
> HS.model <- '
      visual  =~ x1 + a*x2 + a*x3
      textual =~ x4 + b*x5 + b*x6
      speed   =~ x7 + c1*x8 + c2*x9

      c1 == c2
  '
> fit <- cfa(HS.model, data=HolzingerSwineford1939)
> modindices(fit)
Error in if (nrow(H) > 0L) { : argument is of length zero
```

- the first approach is much, much more efficient, but only works for simple equality constraints; we only see the 'reduced' vectors/matrices, and the information matrix is automatically adjusted to take the constraints into account
- the second approach relies on constrained optimization, and is slower, yet more general; it has the advantage that many vectors/matrices retain their original size (reflecting the size of the unconstrained parameter vector), but of course, care is needed when we need to compute for example the inverse of the information matrix, where we need to explicitly take the constraints into account
- what if we could combine the speed/efficiency of the 1st approach, with the generality of the second approach? this is what motivated the change in 0.5-18

3 Equality constraints in 0.5-18

- in 0.5-18, we make a distinction between
  1. linear equality constraints (including simple equality constraints)
  2. non-linear equality constraints
- as an example of 'general' linear constraints, consider the following model:

```
HS.model <- '
    visual  =~ NA*x1 + a1*x1 + a2*x2 + a3*x3
    textual =~ NA*x4 + b1*x4 + b2*x5 + b3*x6
    speed   =~ NA*x7 + c1*x7 + c2*x8 + c3*x9

    a1 + a2 + a3 == 3
    b1 + b2 + b3 == 3
    c1 + c2 + c3 == 3
'
fit <- cfa(HS.model, data=HolzingerSwineford1939)
```

- both simple equality constraints (as in the examples above), and these more general 'linear' equality constraints are now handled in a uniform way
- for all equality constraints, we set up a constraint function (again in `fit@Model@ceq.function`), and we construct the Jacobian matrix A, which is now stored in `fit@Model@ceq.JAC`:

```
> fit@Model@ceq.JAC
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
[1,]    1    1    1    0    0    0    0    0    0     0     0     0     0     0
[2,]    0    0    0    1    1    1    0    0    0     0     0     0     0     0
[3,]    0    0    0    0    0    0    1    1    1     0     0     0     0     0
     [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
[1,]     0     0     0     0     0     0     0     0     0     0
[2,]     0     0     0     0     0     0     0     0     0     0
[3,]     0     0     0     0     0     0     0     0     0     0
```

- the `fit@Model@con.jac` matrix is still available, but may also include inequality constraints
- the constants $a_0 = (3, 3, 3)$ that appear on the 'righ-hand side' of the constraints are stored in `fit@Model@ceq.rhs`, while `fit@Model@ceq.linear.idx` and `fit@Model@ceq.nonlinear.idx` contain the indices of the rows of A that correspond to linear and nonlinear constraints respectively
- consider again the setting with simple equality constraints:

```
HS.model <- '
    visual  =~ x1 + a*x2 + a*x3
    textual =~ x4 + b*x5 + b*x6
    speed   =~ x7 + c*x8 + c*x9
'
fit <- cfa(HS.model, data=HolzingerSwineford1939)
```

- in 0.5-18, the parameter table still contains the (unused) 'unco' and 'eq.id' columns, but this is only temporarily (so that external packages can adapt); from 0.5-19 onwards, the parameter table will look as follows:

```
> parTable(fit)
   id     lhs op     rhs user group free ustart exo label plabel start
1   1  visual =~      x1    1     1    0      1   0         .p1. 1.000
2   2  visual =~      x2    1     1    1     NA   0     a   .p2. 0.778
3   3  visual =~      x3    1     1    2     NA   0     a   .p3. 1.107
4   4 textual =~      x4    1     1    0      1   0         .p4. 1.000
5   5 textual =~      x5    1     1    3     NA   0     b   .p5. 1.133
6   6 textual =~      x6    1     1    4     NA   0     b   .p6. 0.924
7   7   speed =~      x7    1     1    0      1   0         .p7. 1.000
8   8   speed =~      x8    1     1    5     NA   0     c   .p8. 1.225
9   9   speed =~      x9    1     1    6     NA   0     c   .p9. 0.854
10 10      x1 ~~      x1    0     1    7     NA   0        .p10. 0.679
11 11      x2 ~~      x2    0     1    8     NA   0        .p11. 0.691
12 12      x3 ~~      x3    0     1    9     NA   0        .p12. 0.637
13 13      x4 ~~      x4    0     1   10     NA   0        .p13. 0.675
14 14      x5 ~~      x5    0     1   11     NA   0        .p14. 0.830
15 15      x6 ~~      x6    0     1   12     NA   0        .p15. 0.598
16 16      x7 ~~      x7    0     1   13     NA   0        .p16. 0.592
17 17      x8 ~~      x8    0     1   14     NA   0        .p17. 0.511
18 18      x9 ~~      x9    0     1   15     NA   0        .p18. 0.508
19 19  visual ~~  visual    0     1   16     NA   0        .p19. 0.050
20 20 textual ~~ textual    0     1   17     NA   0        .p20. 0.050
21 21   speed ~~   speed    0     1   18     NA   0        .p21. 0.050
22 22  visual ~~ textual    0     1   19     NA   0        .p22. 0.000
23 23  visual ~~   speed    0     1   20     NA   0        .p23. 0.000
24 24 textual ~~   speed    0     1   21     NA   0        .p24. 0.000
25 25    .p2. ==    .p3.    2     0    0     NA   0              0.000
26 26    .p5. ==    .p6.    2     0    0     NA   0              0.000
27 27    .p8. ==    .p9.    2     0    0     NA   0              0.000
```

- note that the non-zero elements in the 'free' column correspond to the unconstrained model parameters; there is no longer a need for the 'eq.id' column nor an 'unco' column; but all equality constraints are explicitly added to the parameter table as extra rows using the '==' operator

- unlike lavaan 0.5-17, lavaan 0.5-18 'thinks' there are 21 free parameters, and this is reflected in the output of `coef()` and `vcov`:

```
> coef(fit)
                a                 a                 b                 b
            0.652             0.652             1.004             1.004
                c                 c             x1~~x1            x2~~x2
            1.156             1.156             0.552             1.113
           x3~~x3            x4~~x4            x5~~x5            x6~~x6
            0.875             0.380             0.514             0.321
           x7~~x7            x8~~x8            x9~~x9    visual~~visual
            0.816             0.520             0.534             0.806
textual~~textual     speed~~speed  visual~~textual    visual~~speed
            0.971             0.368             0.420             0.265
  textual~~speed
            0.175
> dim( vcov(fit) )
[1] 21 21
```

- a somewhat unfortunate byproduct of this approach is that the `inspect()` function (currently) does not show which parameters are constrained or not:

```
> inspect(fit)$lambda
   visual textul speed
x1      0      0     0
x2      1      0     0
x3      2      0     0
x4      0      0     0
x5      0      3     0
x6      0      4     0
x7      0      0     0
x8      0      0     5
x9      0      0     6
```

- in general, it should be understood that all parameter vectors/matrices are now based on the unconstrained parameter vector; the constraints must be handled separately

3.1 Estimation with linear equality constraints

- if some constraints are nonlinear, we revert back to constrained optimization
- if all constraints are linear, the Jacobian matrix A stays constant, and we can exploit this feature to project the unconstrained parameter vector (say, 'x') to a reduced parameter vector (say, 'x.red') (see e.g., Nocedal & Wright (2006) section 15.3 for a good description); this can be done as follows:

```
x.red <- (x - fit@Model@eq.constraints.k0) %*% lavmodel@eq.constraints.K
```

  where `fit@Model@eq.constraints.k0` and `lavmodel@eq.constraints.K` are computed as follows (see the file `lav_constraints.R`):

```
QR <- qr(t(ceq.JAC))
ranK <- QR$rank
Q <- qr.Q(QR, complete = TRUE)
ceq.JAC.NULL <- Q[,-seq_len(ranK), drop = FALSE]
ceq.rhs.NULL <-  ceq.rhs %*% qr.coef(QR, diag(ncol(A)))
eq.constraints.K <- ceq.JAC.NULL
eq.constraints.k0 <- ceq.rhs.NULL
```

- each time the objective function needs to be evaluated, we must project the reduced parameter vector back to the unconstrained vector, which is done as follows:

```
x <- lavmodel@eq.constraints.K %*% x.red + lavmodel@eq.constraints.k0
```

- this avoids using a constrained optimization algorithm, and seems to be the most effective method to handle (linear) equality constraints

3.2 The information matrix with linear equality constraints

- if we need to invert the information matrix, all constraints (linear, nonlinear, equality, inequality) are now handled uniformly by using the 'bordered' (or augmented) information matrix approach (see Approach 2, in 0.5-17)
- a (new) convenience function `lavTech(fit, "information")` returns the unconstrained information matrix (which can be useful on its own); an (internal) function takes care of the augmentation (and optional inversion):

```
E <- lavTech(fit, "information")
E.inv <-
   lavaan:::lav_model_information_augment_invert(lavmodel = fit@Model,
                                                 information = E,
                                                 inverted = TRUE)
```

- for convenience, in lavaan 0.5.18, the following options are available through the lavTech/lavInspect/inspect functions:

```
E <- lavTech(fit, "information")
E <- lavTech(fit, "information.expected")
E <- lavTech(fit, "information.observed")
E <- lavTech(fit, "information.first.order")
E.aug <- lavTech(fit, "augmented.information")
E.aug <- lavTech(fit, "augmented.information.expected")
E.aug <- lavTech(fit, "augmented.information.observed")
E.aug <- lavTech(fit, "augmented.information.first.order")
E.inv <- lavTech(fit, "inverted.information")
E.inv <- lavTech(fit, "inverted.information.expected")
E.inv <- lavTech(fit, "inverted.information.observed")
E.inv <- lavTech(fit, "inverted.information.first.order")
```

- a 'reduced' version of many vectors/matrices can still be obtained by using the `eq.constraints.K` matrix, but note that the rows of this matrix are now normalized, and this may have some (unwanted) side-effects

4 Conclusion and outlook

- for applications that heavily rely on linear equality constraints, the changes in 0.5-18 have resulted in huge improvements both in terms of speed and stability
- at the time of writing, all the side-effects of this change have not yet been fully addressed (for example, in 0.5-18, `the lav_partable_df()` function does not take the equality constraints into account)

- in addition, several add-on packages still rely on the 'old' (0.5-17) approach
- in the future, we plan to handle linear inequality constraints in a similar way (using slack variables)
- even for nonlinear constraints, the 'reduction' approach may be used to some extent; we will explore this later