Miscellaneous Problems

Dealing with missing data

We will analyse the Holzinger Swineford data included in the **lavaan** package.

```
library("lavaan")
summary(HolzingerSwineford1939)
```

```
##        id             sex            ageyr        agemo
##  Min.   :  1.0   Min.   :1.000   Min.   :11   Min.   : 0.000
##  1st Qu.: 82.0   1st Qu.:1.000   1st Qu.:12   1st Qu.: 2.000
##  Median :163.0   Median :2.000   Median :13   Median : 5.000
##  Mean   :176.6   Mean   :1.515   Mean   :13   Mean   : 5.375
##  3rd Qu.:272.0   3rd Qu.:2.000   3rd Qu.:14   3rd Qu.: 8.000
##  Max.   :351.0   Max.   :2.000   Max.   :16   Max.   :11.000
##
##          school        grade             x1               x2
##  Grant-White:145   Min.   :7.000   Min.   :0.6667   Min.   :2.250
##  Pasteur    :156   1st Qu.:7.000   1st Qu.:4.1667   1st Qu.:5.250
##                    Median :7.000   Median :5.0000   Median :6.000
##                    Mean   :7.477   Mean   :4.9358   Mean   :6.088
##                    3rd Qu.:8.000   3rd Qu.:5.6667   3rd Qu.:6.750
##                    Max.   :8.000   Max.   :8.5000   Max.   :9.250
##                    NA's   :1
##        x3              x4              x5              x6
##  Min.   :0.250   Min.   :0.000   Min.   :1.000   Min.   :0.1429
##  1st Qu.:1.375   1st Qu.:2.333   1st Qu.:3.500   1st Qu.:1.4286
##  Median :2.125   Median :3.000   Median :4.500   Median :2.0000
##  Mean   :2.250   Mean   :3.061   Mean   :4.341   Mean   :2.1856
##  3rd Qu.:3.125   3rd Qu.:3.667   3rd Qu.:5.250   3rd Qu.:2.7143
##  Max.   :4.500   Max.   :6.333   Max.   :7.000   Max.   :6.1429
##
##        x7              x8               x9
##  Min.   :1.304   Min.   : 3.050   Min.   :2.778
##  1st Qu.:3.478   1st Qu.: 4.850   1st Qu.:4.750
##  Median :4.087   Median : 5.500   Median :5.417
##  Mean   :4.186   Mean   : 5.527   Mean   :5.374
##  3rd Qu.:4.913   3rd Qu.: 6.100   3rd Qu.:6.083
##  Max.   :7.435   Max.   :10.000   Max.   :9.250
##
```

We will fit a three-factor CFA model to the x variables in the dataset:

```r
HS.model <- '
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  speed   =~ x7 + x8 + x9
'
```

```r
CD_fit <- cfa(HS.model, data = HolzingerSwineford1939, meanstructure = TRUE)
#summary(CD_fit, standardized = TRUE)
fit.inds <- c("chisq", "df", "pvalue", "cfi", "rmsea", "srmr", "aic", "bic")
fitmeasures(CD_fit, fit.inds)
```

Benchmark: Complete data

```
##    chisq       df   pvalue      cfi    rmsea     srmr      aic      bic
##   85.306   24.000    0.000    0.931    0.092    0.060 7535.490 7646.703
```

Generate missingness We introduce some missing data. The values will be missing completely at random, with a probability of .2 for any value being missing:

```r
HSMiss <- HolzingerSwineford1939[,paste("x", 1:9, sep="")]
set.seed(42)
randomMiss <- rbinom(prod(dim(HSMiss)), 1, 0.20)
randomMiss <- matrix(as.logical(randomMiss), nrow=nrow(HSMiss))
HSMiss[randomMiss] <- NA
head(HSMiss)
```

```
##         x1   x2    x3       x4   x5        x6       x7   x8       x9
## 1       NA 7.75 0.375 2.333333   NA 1.2857143 3.391304   NA       NA
## 2       NA 5.25 2.125 1.666667   NA 1.2857143 3.782609 6.25 7.916667
## 3 4.500000 5.25 1.875       NA 1.75 0.4285714       NA 3.90       NA
## 4       NA   NA 3.000 2.666667 4.50 2.4285714 3.000000 5.30 4.861111
## 5 4.833333   NA 0.875 2.666667 4.00 2.5714286 3.695652   NA 5.916667
## 6 5.333333 5.00 2.250 1.000000 3.00 0.8571429 4.347826 6.65 7.500000
```

```r
LD_fit <- cfa(HS.model, data = HSMiss, meanstructure = TRUE)
```

Listwise deletion approach

```
## Warning: lavaan->lav_object_post_check():
##    covariance matrix of latent variables is not positive definite ; use
##    lavInspect(fit, "cov.lv") to investigate.
```

```r
lavInspect(LD_fit, "cov.lv")
```

```
##         visual textul speed
## visual   0.503
## textual  0.774  1.365
## speed    0.152  0.061 0.118
```

```
#summary(LD_fit, standardized = TRUE)
fitmeasures(LD_fit, fit.inds)
```

```
##    chisq       df   pvalue      cfi    rmsea     srmr      aic      bic
##   22.454   24.000    0.552    1.000    0.000    0.076 1298.143 1355.503
```

Multiple imputation approach We now impute the data using package **`mice`**. We use generate five imputed datasets and use the predictive mean matching method, which is (a.f.a.i.k.) the current state of the art in missing data imputation:

```
library("mice")
m <- 5
set.seed(42)
imp_data <- mice(HSMiss, m = m, method = "pmm")
```

```
##
##  iter imp variable
##   1   1  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   1   2  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   1   3  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   1   4  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   1   5  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   2   1  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   2   2  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   2   3  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   2   4  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   2   5  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   3   1  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   3   2  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   3   3  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   3   4  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   3   5  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   4   1  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   4   2  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   4   3  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   4   4  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   4   5  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   5   1  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   5   2  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   5   3  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   5   4  x1  x2  x3  x4  x5  x6  x7  x8  x9
##   5   5  x1  x2  x3  x4  x5  x6  x7  x8  x9
```

We extract the imputed datasets using function `complete()` and save them in a list:

```r
data_list <- list()
for (i in 1:m) data_list[[i]] <- complete(imp_data, action = i)
lapply(data_list, head)
```

```
## [[1]]
##          x1   x2    x3       x4   x5        x6       x7   x8       x9
## 1 3.666667 7.75 0.375 2.333333 3.00 1.2857143 3.391304 4.70 3.777778
## 2 5.666667 5.25 2.125 1.666667 3.00 1.2857143 3.782609 6.25 7.916667
## 3 4.500000 5.25 1.875 1.000000 1.75 0.4285714 3.130435 3.90 3.361111
## 4 5.833333 7.25 3.000 2.666667 4.50 2.4285714 3.000000 5.30 4.861111
## 5 4.833333 5.00 0.875 2.666667 4.00 2.5714286 3.695652 6.30 5.916667
## 6 5.333333 5.00 2.250 1.000000 3.00 0.8571429 4.347826 6.65 7.500000
## 
## [[2]]
##          x1   x2    x3       x4   x5        x6       x7   x8       x9
## 1 4.166667 7.75 0.375 2.333333 5.25 1.2857143 3.391304 6.40 7.916667
## 2 5.333333 5.25 2.125 1.666667 4.25 1.2857143 3.782609 6.25 7.916667
## 3 4.500000 5.25 1.875 1.666667 1.75 0.4285714 3.695652 3.90 5.500000
## 4 4.500000 5.75 3.000 2.666667 4.50 2.4285714 3.000000 5.30 4.861111
## 5 4.833333 8.00 0.875 2.666667 4.00 2.5714286 3.695652 4.30 5.916667
## 6 5.333333 5.00 2.250 1.000000 3.00 0.8571429 4.347826 6.65 7.500000
## 
## [[3]]
##          x1   x2    x3       x4   x5        x6       x7   x8       x9
## 1 3.833333 7.75 0.375 2.333333 3.00 1.2857143 3.391304 4.50 3.333333
## 2 6.000000 5.25 2.125 1.666667 2.75 1.2857143 3.782609 6.25 7.916667
## 3 4.500000 5.25 1.875 2.666667 1.75 0.4285714 3.956522 3.90 4.833333
## 4 4.500000 5.75 3.000 2.666667 4.50 2.4285714 3.000000 5.30 4.861111
## 5 4.833333 5.25 0.875 2.666667 4.00 2.5714286 3.695652 9.10 5.916667
## 6 5.333333 5.00 2.250 1.000000 3.00 0.8571429 4.347826 6.65 7.500000
## 
## [[4]]
##          x1   x2    x3       x4   x5        x6       x7   x8       x9
## 1 2.666667 7.75 0.375 2.333333 4.25 1.2857143 3.391304 5.10 3.361111
## 2 4.166667 5.25 2.125 1.666667 4.50 1.2857143 3.782609 6.25 7.916667
## 3 4.500000 5.25 1.875 2.000000 1.75 0.4285714 2.043478 3.90 5.083333
## 4 3.166667 8.00 3.000 2.666667 4.50 2.4285714 3.000000 5.30 4.861111
## 5 4.833333 5.25 0.875 2.666667 4.00 2.5714286 3.695652 6.95 5.916667
## 6 5.333333 5.00 2.250 1.000000 3.00 0.8571429 4.347826 6.65 7.500000
## 
## [[5]]
##          x1   x2    x3       x4   x5        x6       x7   x8       x9
## 1 4.833333 7.75 0.375 2.333333 3.00 1.2857143 3.391304 4.00 4.416667
## 2 3.166667 5.25 2.125 1.666667 3.25 1.2857143 3.782609 6.25 7.916667
## 3 4.500000 5.25 1.875 1.666667 1.75 0.4285714 2.000000 3.90 5.083333
## 4 4.833333 8.75 3.000 2.666667 4.50 2.4285714 3.000000 5.30 4.861111
## 5 4.833333 6.50 0.875 2.666667 4.00 2.5714286 3.695652 5.35 5.916667
## 6 5.333333 5.00 2.250 1.000000 3.00 0.8571429 4.347826 6.65 7.500000
```

We see that the missing values have been imputed with different values in every dataset.

Now we use the `cfa.mi()` function to fit a CFA model on the imputed data:

```
library("semTools")
```

```
MI_fit <- cfa.mi(HS.model, data_list, meanstructure = TRUE)
```

```
## Warning:
## The runMI() function and lavaan.mi-class have been deprecated and will cease to be included in futur
##
## Support is still provided for analyzing lavaan.mi-class objects (e.g., compRelSEM() can estimate reli
##
## The deprecated runMI() function now creates an object of class OLDlavaan.mi, which can be analyzed u
##
## Find more details help('semTools-deprecated)
```

```
summ_MI_fit <- summary(MI_fit)
```

```
## OLDlavaan.mi object based on 5 imputed data sets.
## See class?OLDlavaan.mi help page for available methods.
##
## Convergence information:
## The model converged on 5 imputed data sets
##
## Rubin's (1987) rules were used to pool point and SE estimates across 5 imputed data sets, and to cal
```

We see that fitting a SEM model on imputed data is quite straightforward: we use function `cfa.mi()` instead of `cfa()`. Using function `summary()`, we obtain the pooled result as a single model. The output is very similar to what were used to with a single dataset. The only difference is that with imputed data, we get t instead of z statistics for every parameter estimate:

```
summ_MI_fit
```

```
##
## Parameter Estimates:
##
##   Standard errors                             Standard
##   Information                                 Expected
##   Information saturated (h1) model          Structured
##
## Latent Variables:
##                    Estimate  Std.Err  t-value       df  P(>|t|)
##   visual =~
##     x1                1.000
##     x2                0.429    0.104    4.110   20.098    0.001
##     x3                0.591    0.107    5.532   20.155    0.000
##   textual =~
##     x4                1.000
##     x5                1.132    0.081   13.900   75.696    0.000
##     x6                0.936    0.068   13.703   34.335    0.000
##   speed =~
##     x7                1.000
##     x8                1.126    0.214    5.250  647.659    0.000
##     x9                1.444    0.274    5.270   22.627    0.000
```

```
##
## Covariances:
##                     Estimate  Std.Err  t-value       df  P(>|t|)
##   visual ~~
##     textual            0.495    0.090    5.474 2829.821    0.000
##     speed              0.292    0.070    4.182  163.670    0.000
##   textual ~~
##     speed              0.169    0.054    3.122  297.158    0.002
##
## Intercepts:
##                     Estimate  Std.Err  t-value       df  P(>|t|)
##    .x1                 4.952    0.080   61.554 1231.955    0.000
##    .x2                 6.123    0.081   76.022      Inf    0.000
##    .x3                 2.200    0.075   29.527   54.113    0.000
##    .x4                 3.009    0.078   38.614  117.055    0.000
##    .x5                 4.324    0.088   48.866  317.076    0.000
##    .x6                 2.162    0.074   29.104   92.839    0.000
##    .x7                 4.184    0.074   56.562   80.495    0.000
##    .x8                 5.512    0.068   80.810  157.431    0.000
##    .x9                 5.357    0.073   73.302  110.788    0.000
##     visual             0.000
##     textual            0.000
##     speed              0.000
##
## Variances:
##                     Estimate  Std.Err  t-value       df  P(>|t|)
##    .x1                 0.446    0.144    3.106   43.694    0.003
##    .x2                 1.197    0.123    9.754   41.235    0.000
##    .x3                 0.850    0.097    8.730   52.643    0.000
##    .x4                 0.358    0.055    6.486   87.401    0.000
##    .x5                 0.469    0.071    6.591  161.355    0.000
##    .x6                 0.355    0.051    6.936   22.414    0.000
##    .x7                 0.861    0.099    8.738   78.258    0.000
##    .x8                 0.609    0.084    7.268   13.055    0.000
##    .x9                 0.520    0.105    4.973   10.066    0.001
##     visual             0.920    0.186    4.953   81.836    0.000
##     textual            0.924    0.127    7.261  201.440    0.000
##     speed              0.293    0.091    3.220  164.978    0.002
```

```
tmp <- fitmeasures(MI_fit)
```

```
## Warning:
## The runMI() and related lavaan.mi functions have been deprecated and will cease to be included in fu
##
## Support is still provided for analyzing lavaan.mi-class objects (e.g., compRelSEM() can estimate rel
##
## The deprecated runMI() function now creates an object of class OLDlavaan.mi, which can be analyzed u
##
## Find more details help('semTools-deprecated)
```

```
## Warning:
## The runMI() function and lavaan.mi-class have been deprecated and will cease to be included in futur
##
```

```
## Support is still provided for analyzing lavaan.mi-class objects (e.g., compRelSEM() can estimate reli
##
## The deprecated runMI() function now creates an object of class OLDlavaan.mi, which can be analyzed us
##
## Find more details help('semTools-deprecated)
```

```
## Warning: lavaan->modindices():
##    list with extra parameters is empty; to release equality constraints, use
##    lavTestScore()

## Warning: lavaan->modindices():
##    list with extra parameters is empty; to release equality constraints, use
##    lavTestScore()

## Warning: lavaan->modindices():
##    list with extra parameters is empty; to release equality constraints, use
##    lavTestScore()

## Warning: lavaan->modindices():
##    list with extra parameters is empty; to release equality constraints, use
##    lavTestScore()

## Warning: lavaan->modindices():
##    list with extra parameters is empty; to release equality constraints, use
##    lavTestScore()
```

```
## Warning:
## The runMI() and related lavaan.mi functions have been deprecated and will cease to be included in fut
##
## Support is still provided for analyzing lavaan.mi-class objects (e.g., compRelSEM() can estimate reli
##
## The deprecated runMI() function now creates an object of class OLDlavaan.mi, which can be analyzed us
##
## Find more details help('semTools-deprecated)
```

```r
round(tmp[fit.inds], digits = 3L)
```

```
##    chisq       df   pvalue      cfi    rmsea     srmr      aic      bic
##   43.658   24.000    0.008    0.959    0.052    0.050 7526.811 7638.024
```

```r
FIML_fit <- cfa(HS.model, data = HSMiss, missing = "ml")
#summary(FIML_fit)
fitmeasures(LD_fit, fit.inds)
```

Full information Maximum Likelihood (FIML)

```
##    chisq       df   pvalue      cfi    rmsea     srmr      aic      bic
##   22.454   24.000    0.552    1.000    0.000    0.076 1298.143 1355.503
```

Comparison of methods

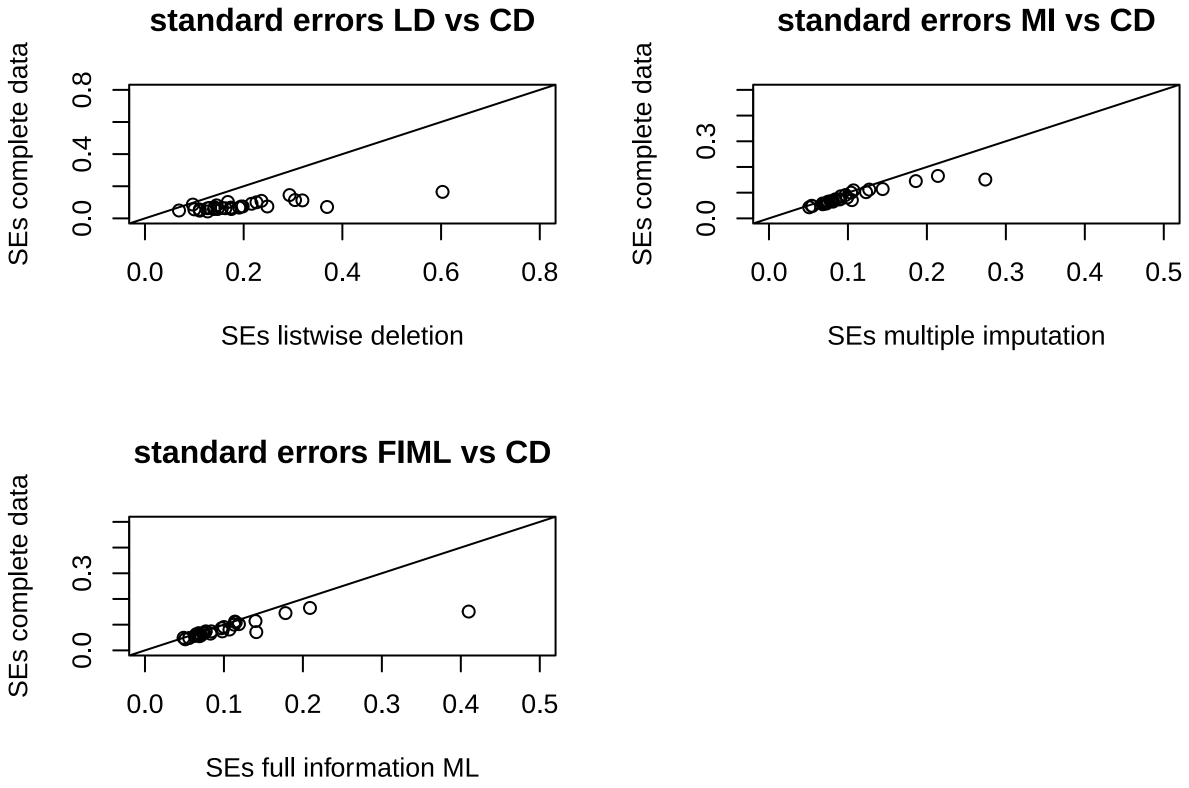
We compare parameter estimates and standard errors between the complete dataset, listwise deletion, multiple imputation and FIML:

```
comp_data <-
  cbind(parameterestimates(LD_fit, standardized = TRUE)[ , 1:3],
        LD = round(parameterestimates(LD_fit, standardized = TRUE)[ , 4:5],
                   digits = 3L),
        MI = round(data.frame(summ_MI_fit)[ , 5:6], digits = 3L),
        FIML = round(parameterestimates(FIML_fit, standardized = TRUE)[ , 4:5],
                     digits = 3L),
        CD = round(parameterestimates(CD_fit, standardized = TRUE)[ , 4:5],
                   digits = 3L))
comp_data <- comp_data[comp_data$LD.se > 0, ]
comp_data
```

```
##         lhs op     rhs LD.est LD.se MI.est MI.se FIML.est FIML.se CD.est CD.se
## 2   visual =~      x2  0.660 0.226  0.429 0.104    0.482   0.113  0.554 0.100
## 3   visual =~      x3  0.558 0.236  0.591 0.107    0.633   0.115  0.729 0.109
## 5  textual =~      x5  0.965 0.131  1.132 0.081    1.154   0.083  1.113 0.065
## 6  textual =~      x6  0.764 0.111  0.936 0.068    0.951   0.069  0.926 0.055
## 8    speed =~      x8  1.289 0.603  1.126 0.214    1.122   0.209  1.180 0.165
## 9    speed =~      x9  2.796 1.465  1.444 0.274    1.586   0.410  1.082 0.151
## 10      x1 ~~      x1  1.317 0.304  0.446 0.144    0.477   0.140  0.549 0.114
## 11      x2 ~~      x2  0.771 0.168  1.197 0.123    1.179   0.119  1.134 0.102
## 12      x3 ~~      x3  1.059 0.216  0.850 0.097    0.852   0.100  0.844 0.091
## 13      x4 ~~      x4  0.144 0.111  0.358 0.055    0.346   0.056  0.371 0.048
## 14      x5 ~~      x5  0.696 0.175  0.469 0.071    0.447   0.071  0.446 0.058
## 15      x6 ~~      x6  0.537 0.127  0.355 0.051    0.341   0.051  0.356 0.043
## 16      x7 ~~      x7  0.683 0.145  0.861 0.099    0.855   0.107  0.799 0.081
## 17      x8 ~~      x8  0.891 0.195  0.609 0.084    0.658   0.098  0.488 0.074
## 18      x9 ~~      x9  0.055 0.369  0.520 0.105    0.429   0.141  0.566 0.071
## 19  visual ~~  visual  0.503 0.293  0.920 0.186    0.898   0.178  0.809 0.145
## 20 textual ~~ textual  1.365 0.319  0.924 0.127    0.902   0.114  0.979 0.112
## 21   speed ~~   speed  0.118 0.097  0.293 0.091    0.262   0.097  0.384 0.086
## 22  visual ~~ textual  0.774 0.248  0.495 0.090    0.467   0.084  0.408 0.074
## 23  visual ~~   speed  0.152 0.100  0.292 0.070    0.274   0.063  0.262 0.056
## 24 textual ~~   speed  0.061 0.069  0.169 0.054    0.157   0.049  0.173 0.049
## 25      x1 ~1          4.877 0.191  4.952 0.080    4.949   0.073  4.936 0.067
## 26      x2 ~1          5.895 0.141  6.123 0.081    6.136   0.075  6.088 0.068
## 27      x3 ~1          2.038 0.156  2.200 0.075    2.212   0.069  2.250 0.065
## 28      x4 ~1          2.747 0.174  3.009 0.078    3.014   0.068  3.061 0.067
## 29      x5 ~1          4.165 0.198  4.324 0.088    4.319   0.077  4.341 0.074
## 30      x6 ~1          2.186 0.163  2.162 0.074    2.167   0.065  2.186 0.063
## 31      x7 ~1          4.383 0.127  4.184 0.074    4.176   0.068  4.186 0.063
## 32      x8 ~1          5.707 0.147  5.512 0.068    5.501   0.064  5.527 0.058
## 33      x9 ~1          5.424 0.140  5.357 0.073    5.361   0.066  5.374 0.058
```

Those are a lot of numbers to compare, let's create some plots:

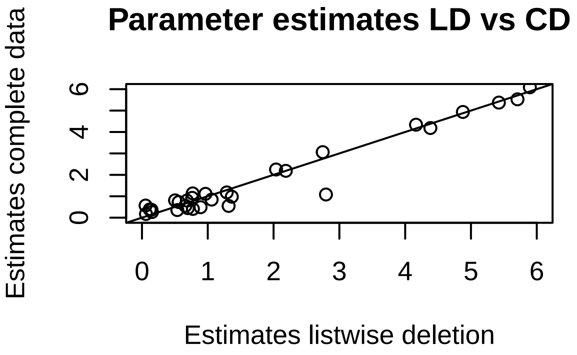
```
par(mfrow = c(2, 2))
plot(comp_data$LD.se, comp_data$CD.se, xlim = c(0, 0.8), ylim = c(0, 0.8),
     main = "standard errors LD vs CD",
     xlab = "SEs listwise deletion",
     ylab = "SEs complete data")
abline(0, 1)
plot(comp_data$MI.se, comp_data$CD.se, xlim = c(0, 0.5), ylim = c(0, 0.5),
     main = "standard errors MI vs CD",
     ylab = "SEs complete data",
     xlab = "SEs multiple imputation")
abline(0, 1)
plot(comp_data$FIML.se, comp_data$CD.se, xlim = c(0, 0.5), ylim = c(0, 0.5),
     main = "standard errors FIML vs CD",
     ylab = "SEs complete data",
     xlab = "SEs full information ML")
abline(0, 1)
```

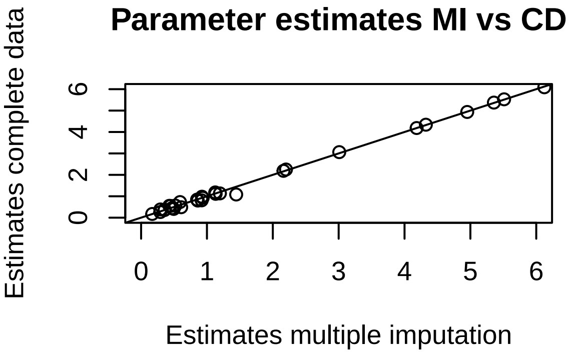


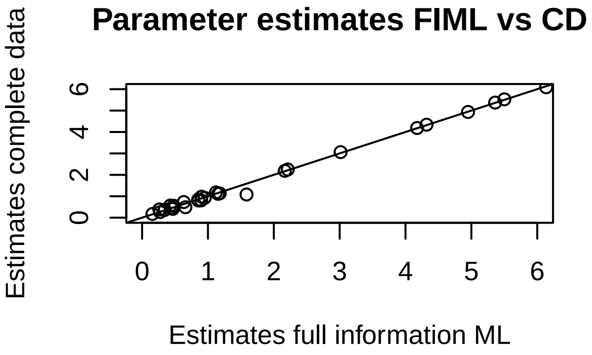
Listwise deletion yields much larger standard errors than we would obtain if we had the complete data. The standard errors obtained with multiply imputed data are much closer to those obtained with the complete data. The MI standard errors tend to be somewhat higher, but this is what should happen, as we did not use the full dataset with MI. The bottom-left plot indicates a similar pattern for (full information) ML: standard errors are only somewhat larger than when analysing complete data.

```
par(mfrow = c(2, 2))
plot(comp_data$LD.est, comp_data$CD.est, xlim = c(0, 6), ylim = c(0, 6),
     main = "Parameter estimates LD vs CD",
     xlab = "Estimates listwise deletion",
     ylab = "Estimates complete data")
```

```
abline(0, 1)
plot(comp_data$MI.est, comp_data$CD.est, xlim = c(0, 6), ylim = c(0, 6),
     main = "Parameter estimates MI vs CD",
     ylab = "Estimates complete data",
     xlab = "Estimates multiple imputation")
abline(0, 1)
plot(comp_data$FIML.est, comp_data$CD.est, xlim = c(0, 6), ylim = c(0, 6),
     main = "Parameter estimates FIML vs CD",
     ylab = "Estimates complete data",
     xlab = "Estimates full information ML")
abline(0, 1)
```







The parameter estimates with listwise deletion vary much more from the parameter estimates than would have been obtained with the complete data. The parameter estimates with MI and FIML resemble those obtained with the complete data much more closer.

Parameters relating to exogenous variables

In many SEM analyses, parameters relating to exogenous variables will often not be provided. Often, exogenous variables will be considered fixed. As a result, their (co)variances are fixed to their sample (co)variances, instead of being estimated as parameters in the model. For the model fit (χ^2 and *df*), this does not make a difference. But sometimes you may want to inspect the variation or associations between the exogonous variables.

```
HS_data <- HolzingerSwineford1939
HS_data$age <- with(HS_data, ageyr + agemo/12)
HS_data$sex <- HS_data$sex - 1 # to make it 0-1 coded
HS.model2 <- '
  visual  =~ x1 + x2 + x3
  textual =~ x4 + x5 + x6
  visual + textual ~ sex + age
'
HS_mod1 <- cfa(HS.model2, data = HS_data, estimator = "MLR")
summary(HS_mod1, standardized = TRUE)
```

```
## lavaan 0.6-18 ended normally after 30 iterations
## 
##   Estimator                                         ML
##   Optimization method                           NLMINB
##   Number of model parameters                        17
## 
##   Number of observations                           301
## 
## Model Test User Model:
##                                               Standard      Scaled
##   Test Statistic                                35.619      35.485
##   Degrees of freedom                                16          16
##   P-value (Chi-square)                           0.003       0.003
##   Scaling correction factor                                  1.004
##     Yuan-Bentler correction (Mplus variant)
## 
## Parameter Estimates:
## 
##   Standard errors                             Sandwich
##   Information bread                           Observed
##   Observed information based on                Hessian
## 
## Latent Variables:
##                    Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##   visual =~
##     x1                1.000                               0.850    0.729
##     x2                0.635    0.163    3.890    0.000    0.540    0.459
##     x3                0.804    0.174    4.610    0.000    0.683    0.605
##   textual =~
##     x4                1.000                               0.993    0.855
##     x5                1.110    0.067   16.632    0.000    1.102    0.856
##     x6                0.919    0.061   14.952    0.000    0.912    0.834
## 
## Regressions:
```

```
##                     Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##   visual ~
##     sex              -0.329    0.123   -2.676    0.007   -0.387   -0.194
##     age              -0.038    0.064   -0.593    0.553   -0.045   -0.045
##   textual ~
##     sex               0.076    0.122    0.624    0.533    0.077    0.038
##     age              -0.236    0.057   -4.129    0.000   -0.237   -0.241
##
## Covariances:
##                     Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##  .visual ~~
##    .textual           0.384    0.105    3.652    0.000    0.479    0.479
##
## Variances:
##                     Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##    .x1                0.636    0.171    3.714    0.000    0.636    0.468
##    .x2                1.091    0.110    9.957    0.000    1.091    0.789
##    .x3                0.808    0.111    7.294    0.000    0.808    0.634
##    .x4                0.364    0.050    7.257    0.000    0.364    0.270
##    .x5                0.445    0.058    7.606    0.000    0.445    0.268
##    .x6                0.364    0.048    7.559    0.000    0.364    0.304
##    .visual            0.695    0.192    3.613    0.000    0.963    0.963
##    .textual           0.925    0.112    8.235    0.000    0.937    0.937
```

We see that the (co)variances of the exogenous variables (`sex` and `age`) are not estimated in the model. As a results, we cannot inspect their association. To include them in the model as model parameters, we have to additionally specify `fixed.x = FALSE` in the call to `cfa()`:

```r
HS_mod2 <- cfa(HS.model2, data = HS_data, estimator = "MLR", fixed.x = FALSE)
summary(HS_mod2, standardized = TRUE)
```

```
## lavaan 0.6-18 ended normally after 32 iterations
##
##   Estimator                                         ML
##   Optimization method                           NLMINB
##   Number of model parameters                        20
##
##   Number of observations                           301
##
## Model Test User Model:
##                                               Standard      Scaled
##   Test Statistic                                35.619      35.485
##   Degrees of freedom                                16          16
##   P-value (Chi-square)                           0.003       0.003
##   Scaling correction factor                                  1.004
##     Yuan-Bentler correction (Mplus variant)
##
## Parameter Estimates:
##
##   Standard errors                             Sandwich
##   Information bread                           Observed
##   Observed information based on                Hessian
##
```

```
## Latent Variables:
##                    Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##   visual =~
##     x1                1.000                               0.850    0.729
##     x2                0.635    0.163    3.890    0.000    0.540    0.459
##     x3                0.804    0.174    4.610    0.000    0.683    0.605
##   textual =~
##     x4                1.000                               0.993    0.855
##     x5                1.110    0.067   16.632    0.000    1.102    0.856
##     x6                0.919    0.061   14.952    0.000    0.912    0.834
##
## Regressions:
##                    Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##   visual ~
##     sex              -0.329    0.123   -2.676    0.007   -0.387   -0.194
##     age              -0.038    0.064   -0.593    0.553   -0.045   -0.045
##   textual ~
##     sex               0.076    0.122    0.624    0.533    0.077    0.038
##     age              -0.236    0.057   -4.129    0.000   -0.237   -0.241
##
## Covariances:
##                    Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##  .visual ~~
##    .textual           0.384    0.105    3.652    0.000    0.479    0.479
##   sex ~~
##     age              -0.081    0.029   -2.791    0.005   -0.081   -0.160
##
## Variances:
##                    Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##    .x1                0.636    0.171    3.714    0.000    0.636    0.468
##    .x2                1.091    0.110    9.957    0.000    1.091    0.789
##    .x3                0.808    0.111    7.294    0.000    0.808    0.634
##    .x4                0.364    0.050    7.257    0.000    0.364    0.270
##    .x5                0.445    0.058    7.606    0.000    0.445    0.268
##    .x6                0.364    0.048    7.559    0.000    0.364    0.304
##    .visual            0.695    0.192    3.613    0.000    0.963    0.963
##    .textual           0.925    0.112    8.235    0.000    0.937    0.937
##     sex               0.250    0.001  289.990    0.000    0.250    1.000
##     age               1.035    0.087   11.907    0.000    1.035    1.000
```