Overview of the functionalities of the package lavaSearch2

Brice Ozenne

March 18, 2019

Load **lavaSearch2** in the R session:

```
library(lavaSearch2)
```

1 Inference

1.1 Introductory example

You may have noticed that for simple linear regression, the p-values of the Wald tests from `lm`:

```
## simulate data
mSim <- lvm(Y[1:1]~0.3*X1+0.2*X2)
set.seed(10)
df.data <- sim(mSim, 2e1)

## fit linear model
summary(lm(Y~X1+X2, data = df.data))$coef
```

```
             Estimate Std. Error   t value    Pr(>|t|)
(Intercept) 0.7967775  0.2506767 3.1785069 0.005495832
X1          0.1550938  0.2205080 0.7033477 0.491360483
X2          0.4581556  0.2196785 2.0855736 0.052401103
```

differ from those obtained with lava:

```
## fit latent variable model
m <- lvm(Y~X1+X2)
e <- estimate(m, data = df.data)

## extract Wald tests
summary(e)$coef
```

```
      Estimate Std. Error   Z-value      P-value
Y~X1 0.1550938  0.2032984 0.7628877 0.4455303456
Y~X2 0.4581556  0.2025335 2.2621221 0.0236898575
Y~~Y 0.5557910  0.1757566 3.1622777           NA
Y    0.7967775  0.2311125 3.4475747 0.0005656439
```

For instance, the p-value for the effect of X2 is 0.024 according to lava and 0.052 according to `lm`. The discrepancy is due to 2 corrections that `lm` applies in order to improve the control of the type 1 error of the Wald tests:

- use of a student distribution instead of a normal distribution (informally t-value instead of z-value).
- use of a bias-corrected estimator of the residuals variance instead of the ML-estimator.

lavaSearch2 attempts to generalize these corrections to models with correlated and heteroschedastic measurements. In the case of a simple linear regression, Wald tests obtained with **lavaSearch2** closely approximate the results of `lm`:

```
summary2(e)$coef
```

```
     Estimate Std. Error   t-value    P-value    df
Y~X1 0.1550938  0.2205078 0.7033483 0.49136012 17.00
Y~X2 0.4581556  0.2196783 2.0855754 0.05240092 17.00
Y~~Y 0.6538707  0.2242758 2.9154759         NA  4.25
Y    0.7967775  0.2506765 3.1785096 0.00549580 17.00
```

1.2 How it works in a nutshell

When using **lava**, the p.values that are obtained from the summary (Wald tests) rely on a Gaussian approximation. While being asymptotically valid, this approximation may not provide a very accurate control of the type 1 error rate in small samples. Simulations have shown that the type 1 error rate tends to be too large, i.e. the p.values are have a downward bias. gg **lavaSearch2** improves the Gaussian approximation in two way:

- using a Student's t distribution instead of a normal distribution to account for the uncertainty on the variance of the coefficients. The degrees of freedom are estimated using Satterwaite approximation, i.e. identifying the chi-squared distribution that best fit the observed moments of the variance of the coefficients.
- correct for the first order bias in the residual variance due to ML estimation. This bias also affects the standard error of the estimates and the control of the type 1 error. The correction does not change the estimates (i.e. the column "Estimate" in the summary remain unchanged), but it changes the corresponding standard error and degree of freedoms (i.e. columns "Std. Error" and "df" in the summary are modified).

1.3 Single univariate Wald test

We will illustrate the functionalities using a simulated dataset:

```
## simulate data
mSim <- lvm(Y1~eta,Y2~eta,Y3~0.4+0.4*eta,Y4~0.6+0.6*eta,eta~0.5*X1+0.7*X2)
latent(mSim) <- ~eta
set.seed(12)
```

```
df.data <- sim(mSim, n = 3e1, latent = FALSE)

## display
head(df.data)
```

```
         Y1         Y2          Y3         Y4         X1         X2
1 -1.7606233  0.1264910  0.66442611  0.2579355  0.2523400 -1.5431527
2  3.0459417  2.4631929  0.00283511  2.1714802  0.6423143 -1.3206009
3 -2.1443162 -0.3318033  0.82253070  0.3008415 -0.3469361 -0.6758215
4 -2.5050328 -1.3878987 -0.10474850 -1.7814956 -0.5152632 -0.3670054
5 -2.5307249  0.3012422  1.22046986 -1.0195188  0.3981689 -0.5138722
6 -0.9521366  0.1669496 -0.21422548  1.5954456  0.9535572 -0.9592540
```

We first fit the latent variable model using, as usual, the `estimate` function:

```
m <- lvm(c(Y1,Y2,Y3,Y4)~eta, eta~X1+X2)
e <- estimate(m, data = df.data)
```

We can extract the Wald tests based on a normal approximation using `summary`:

```
summary(e)$coef[c("Y2","Y3","Y2~eta","Y3~eta","eta~X1","eta~X2"), ]
```

```
        Estimate Std. Error   Z-value      P-value
Y2     0.2335412  0.2448593 0.9537775 0.3401962906
Y3     0.5114275  0.1785886 2.8637186 0.0041869974
Y2~eta 0.9192847  0.2621248 3.5070497 0.0004531045
Y3~eta 0.2626930  0.1558978 1.6850339 0.0919820326
eta~X1 0.5150072  0.2513393 2.0490515 0.0404570768
eta~X2 0.6212222  0.2118930 2.9317729 0.0033703310
```

As explain at the begining of this section, **lavaSearch2** implements two corrections that can be directly applied by calling the `summary2` method:

```
summary2(e)$coef[c("Y2","Y3","Y2~eta","Y3~eta","eta~X1","eta~X2"), ]
```

```
        Estimate Std. Error   t-value     P-value        df
Y2     0.2335412  0.2518218 0.9274067 0.371516094 12.328385
Y3     0.5114275  0.1828716 2.7966475 0.009848769 24.707696
Y2~eta 0.9192847  0.2653220 3.4647887 0.031585600  3.515034
Y3~eta 0.2626930  0.1562776 1.6809386 0.143826633  5.993407
eta~X1 0.5150072  0.2642257 1.9491180 0.065414617 20.044312
eta~X2 0.6212222  0.2221293 2.7966698 0.009275494 27.718363
```

To use the Satterthwaite correction alone, set the argument `bias.correct` to `FALSE`:

```
summary2(e, bias.correct = FALSE)$coef[c("Y2","Y3","Y2~eta","Y3~eta","eta~X1","eta~
    X2"), ]
```

```
        Estimate Std. Error   t-value     P-value        df
Y2     0.2335412  0.2448593 0.9537775 0.357711941 12.911877
Y3     0.5114275  0.1785886 2.8637186 0.008210968 25.780552
Y2~eta 0.9192847  0.2621248 3.5070497 0.028396459  3.674640
Y3~eta 0.2626930  0.1558978 1.6850339 0.141185621  6.222912
eta~X1 0.5150072  0.2513393 2.0490515 0.052814794 21.571210
eta~X2 0.6212222  0.2118930 2.9317729 0.006351686 30.370334
```

When using the Satterthwaite correction alone, the standard error are left unchanged compared to the original lava output. The only change is how the p-values are computed, i.e. based on the quantiles of a Student's t distribution instead of a Gaussian distribution.

1.4 Saving computation time with `sCorrect`

For each call to `summary2` the small sample size correction(s) will be recalculated. However the calculation of the sample correction(s) can be time consuming.

```
system.time(
    res <- summary2(e, bias.correct = FALSE)
)
```

```
user  system elapsed
0.24    0.00    0.24
```

In such a case one can pre-compute the main terms of the correction (e.g. the derivative of the variance-covariance matrix) once for all using the `sCorrect` method (`sCorrect` stands for Satterthwaite correction). When calling `sCorrect`, the right hand side indicates whether the bias correction should be used (equivalent to `bias.correct` argument described previously):

```
e2 <- e
sCorrect(e2) <- TRUE
```

`sCorrect` automatically store the pre-computed terms in the `sCorrect` slot of the object. It also adds the class `lvmfit2` to the object:

```
class(e2)
```

```
[1] "lvmfit2" "lvmfit"
```

Then p-values computed using the small sample correction can be obtained calling the `summary` method, as usual:

```
summary2(e2)$coef[c("Y2","Y3","Y2~eta","Y3~eta","eta~X1","eta~X2"), ]
```

```
        Estimate Std. Error   t-value     P-value        df
Y2     0.2335412  0.2518218 0.9274067 0.371516094 12.328385
Y3     0.5114275  0.1828716 2.7966475 0.009848769 24.707696
Y2~eta 0.9192847  0.2653220 3.4647887 0.031585600  3.515034
Y3~eta 0.2626930  0.1562776 1.6809386 0.143826633  5.993407
eta~X1 0.5150072  0.2642257 1.9491180 0.065414617 20.044312
eta~X2 0.6212222  0.2221293 2.7966698 0.009275494 27.718363
```

The summary2 methods take approximately the same time as the usual summary method:

```
system.time(
    summary2(e2)
)
```

```
user  system elapsed
0.10    0.00    0.09
```

```
system.time(
    summary(e2)
)
```

```
user  system elapsed
0.08    0.00    0.08
```

1.5 Single multivariate Wald test

The function compare can be use to perform multivariate Wald tests, i.e. to test simultaneously several linear combinations of the coefficients. compare uses a contrast matrix to encode in lines which linear combination of coefficients should be tested. For instance if we want to simultaneously test whether all the mean coefficients are 0, we can create a contrast matrix using createContrast:

```
resC <- createContrast(e2, par = c("Y2=0","Y2~eta=0","eta~X1=0"))
resC
```

```
$contrast
             Y2 Y3 Y4 eta Y2~eta Y3~eta Y4~eta eta~X1 eta~X2 Y1~~Y1 Y2~~Y2 Y3~~Y3 Y4~~Y4
[Y2] = 0      1  0  0   0      0      0      0      0      0      0      0      0      0
[Y2~eta] = 0  0  0  0   0      1      0      0      0      0      0      0      0      0
[eta~X1] = 0  0  0  0   0      0      0      0      1      0      0      0      0      0
             eta~~eta
[Y2] = 0            0
[Y2~eta] = 0        0
[eta~X1] = 0        0

$null
    [Y2] = 0 [Y2~eta] = 0 [eta~X1] = 0
           0            0            0

$Q
[1] 3
```

We can then test the linear hypothesis by specifying in compare the left hand side of the hypothesis (argument contrast) and the right hand side (argument null):

```
resTest0 <- lava::compare(e2, contrast = resC$contrast, null = resC$null)
resTest0
```

```
        - Wald test -

        Null Hypothesis:
        [Y2] = 0
        [Y2~eta] = 0
        [eta~X1] = 0

data:
chisq = 21.332, df = 3, p-value = 8.981e-05
sample estimates:
          Estimate   Std.Err       2.5%     97.5%
[Y2]     0.2335412 0.2448593 -0.2463741 0.7134566
[Y2~eta] 0.9192847 0.2621248  0.4055295 1.4330399
[eta~X1] 0.5150072 0.2513393  0.0223912 1.0076231
```

`compare` uses a chi-squared distribution to compute the p-values. Similarly to the Gaussian approximation, while being valid asymptotically this procedure may not provide a very accurate control of the type 1 error rate in small samples. Fortunately, the correction proposed for the univariate Wald statistic can be adapted to the multivariate Wald statistic. This is achieved by `compare2`:

```
resTest1 <- compare2(e2, contrast = resC$contrast, null = resC$null)
resTest1
```

```
        - Wald test -

        Null Hypothesis:
        [Y2] = 0
        [Y2~eta] = 0
        [eta~X1] = 0

data:
F-statistic = 6.7118, df1 = 3, df2 = 11.1, p-value = 0.007596
sample estimates:
              Estimate   Std.Err        df       2.5%     97.5%
[Y2] = 0     0.2335412 0.2518218 12.328385 -0.3135148 0.7805973
[Y2~eta] = 0 0.9192847 0.2653220  3.515034  0.1407653 1.6978041
[eta~X1] = 0 0.5150072 0.2642257 20.044312 -0.0360800 1.0660943
```

The same result could have been obtained using the par argument to define the linear hypothesis:

```
resTest2 <- compare2(e2, par = c("Y2","Y2~eta","eta~X1"))
identical(resTest1,resTest2)
```

```
[1] TRUE
```

Now a F distribution is used to compute the p-values. As before on can set the argument `bias.correct` to `FALSE` to use the Satterthwaite approximation alone:

```
resTest3 <- compare2(e, bias.correct = FALSE,
                     contrast = resC$contrast, null = resC$null)
resTest3
```

```
        - Wald test -

        Null Hypothesis:
        [Y2] = 0
        [Y2~eta] = 0
        [eta~X1] = 0

data:
F-statistic = 7.1107, df1 = 3, df2 = 11.13, p-value = 0.006182
sample estimates:
              Estimate   Std.Err       df         2.5%     97.5%
[Y2] = 0     0.2335412 0.2448593 12.91188 -0.295812256 0.7628948
[Y2~eta] = 0 0.9192847 0.2621248  3.67464  0.165378080 1.6731913
[eta~X1] = 0 0.5150072 0.2513393 21.57121 -0.006840023 1.0368543
```

In this case the F-statistic of `compare2` is the same as the chi-squared statistic of `compare` divided by the rank of the contrast matrix:

```
resTest0$statistic/qr(resC$contrast)$rank
```

```
   chisq
7.110689
```

1.6 Robust Wald tests

When one does not want to assume normality distributed residuals, robust standard error can be used instead of the model based standard errors. They can be obtain by setting the argument `robust` to `TRUE` when computing univariate Wald tests:

```
summary2(e, robust = TRUE)$coef[c("Y2","Y3","Y2~eta","Y3~eta","eta~X1","eta~X2"), ]
```

```
        Estimate robust SE   t-value     P-value        df
Y2     0.2335412 0.2353245 0.9924222 0.340071187 12.328385
Y3     0.5114275 0.1897160 2.6957535 0.012449993 24.707696
Y2~eta 0.9192847 0.1791240 5.1321150 0.009609880  3.515034
Y3~eta 0.2626930 0.1365520 1.9237585 0.102782655  5.993407
eta~X1 0.5150072 0.2167580 2.3759546 0.027585480 20.044312
eta~X2 0.6212222 0.2036501 3.0504389 0.004989038 27.718363
```

or multivariate Wald test:

```
compare2(e2, robust = TRUE, par = c("Y2","Y2~eta","eta~X1"))
```

```
      - Wald test -

      Null Hypothesis:
      [Y2] = 0
      [Y2~eta] = 0
      [eta~X1] = 0

data:
F-statistic = 12.526, df1 = 3, df2 = 11.1, p-value = 0.0006959
sample estimates:
              Estimate robust SE        df        2.5%     97.5%
[Y2] = 0     0.2335412 0.2353245 12.328385 -0.27767612 0.7447586
[Y2~eta] = 0 0.9192847 0.1791240  3.515034  0.39369139 1.4448780
[eta~X1] = 0 0.5150072 0.2167580 20.044312  0.06292197 0.9670923
```

Only the standard error is affected by the argument `robust`, the degrees of freedom are the one of the model-based standard errors. It may be surprising that the (corrected) robust standard errors are (in this example) smaller than the (corrected) model-based one. This is also the case for the uncorrected one:

```
rbind(robust = diag(crossprod(iid(e2))),
      model = diag(vcov(e2)))
```

```
               Y2         Y3         Y4        eta     Y2~eta     Y3~eta     Y4~eta
robust 0.04777252 0.03325435 0.03886706 0.06011727 0.08590732 0.02179453 0.02981895
model  0.05995606 0.03189389 0.04644303 0.06132384 0.06870941 0.02430412 0.03715633
          eta~X1     eta~X2    Y1~~Y1    Y2~~Y2     Y3~~Y3     Y4~~Y4  eta~~eta
robust 0.05166005 0.05709393 0.2795272 0.1078948 0.03769614 0.06923165 0.3198022
model  0.06317144 0.04489865 0.1754744 0.1600112 0.05112998 0.10152642 0.2320190
```

This may be explained by the fact the robust standard error tends to be liberal in small samples (e.g. see Kauermann 2001, A Note on the Efficiency of Sandwich Covariance Matrix Estimation).

1.7 Assessing the type 1 error of the testing procedure

The function `calibrateType1` can be used to assess the type 1 error of a Wald statistic on a specific example. This however assumes that the estimated model is correctly specified. Let's make an example. For this we simulate some data:

```
set.seed(10)
m.generative <- lvm(Y ~ X1 + X2 + Gene)
categorical(m.generative, labels = c("ss","ll")) <- ~Gene
d <- lava::sim(m.generative, n = 50, latent = FALSE)
```

Let's now imagine that we want to analyze the relationship between Y and Gene using the following dataset:

```
head(d)
```

```
           Y         X1         X2 Gene
1 -1.14369572 -0.4006375 -0.7618043   ss
2 -0.09943370 -0.3345566  0.4193754   ss
3 -0.04331996  1.3679540 -1.0399434   ll
4  2.25017335  2.1377671  0.7115740   ss
5  0.16715138  0.5058193 -0.6332130   ss
6  1.73931135  0.7863424  0.5631747   ss
```

For this we fit define a LVM:

```
myModel <- lvm(Y ~ X1 + X2 + Gene)
```

and estimate the coefficients of the model using `estimate`:

```
e <- estimate(myModel, data = d)
e
```

```
                  Estimate Std. Error  Z-value  P-value
Regressions:
   Y~X1            1.02349    0.12017  8.51728   <1e-12
   Y~X2            0.91519    0.12380  7.39244   <1e-12
   Y~Genell        0.48035    0.23991  2.00224  0.04526
Intercepts:
   Y              -0.11221    0.15773 -0.71141   0.4768
Residual Variances:
   Y               0.67073    0.13415  5.00000
```

We can now use `calibrateType1` to perform a simulation study. We just need to define the null hypotheses (i.e. which coefficients should be set to 0 when generating the data) and the number of simulations:

```
mySimulation <- calibrateType1(e,
                               null = "Y~Genell",
                               n.rep = 50,
                               trace = FALSE, seed = 10)
```

To save time we only make 50 simulations but much more are necessary to really assess the type 1 error rate. Then we can use the `summary` method to display the results:

```
summary(mySimulation)
```

```
Estimated type 1 error rate [95% confidence interval]
  > sample size: 50 | number of simulations: 50
     link statistic correction type1error                 CI
 Y~Genell      Wald       Gaus       0.12 [0.05492 ; 0.24242]
```

```
                      Satt       0.10 [0.04224 ; 0.21869]
                       SSC       0.10 [0.04224 ; 0.21869]
                  SSC + Satt       0.08 [0.03035 ; 0.19456]

Corrections: Gaus = Gaussian approximation
             SSC  = small sample correction
             Satt = Satterthwaite approximation
```

2 Adjustment for multiple comparisons

2.1 Univariate Wald test, single model

When performing multiple testing, adjustment for multiple comparisons is necessary in order to control the type 1 error rate, i.e. to provide interpretable p.values. The **multcomp** package enables to do such adjustment when all tests comes from the same `lvmfit` object:

```
## simulate data
mSim <- lvm(Y ~ 0.25 * X1 + 0.3 * X2 + 0.35 * X3 + 0.4 * X4 + 0.45 * X5 + 0.5 * X6)
set.seed(10)
df.data <- sim(mSim, n = 4e1)

## fit lvm
e.lvm <- estimate(lvm(Y ~ X1 + X2 + X3 + X4 + X5 + X6), data = df.data)
name.coef <- names(coef(e.lvm))
n.coef <- length(name.coef)

## Create contrast matrix
resC <- createContrast(e.lvm, par = paste0("Y~X",1:6), rowname.rhs = FALSE)
resC$contrast
```

```
     Y Y~X1 Y~X2 Y~X3 Y~X4 Y~X5 Y~X6 Y~~Y
Y~X1 0    1    0    0    0    0    0    0
Y~X2 0    0    1    0    0    0    0    0
Y~X3 0    0    0    1    0    0    0    0
Y~X4 0    0    0    0    1    0    0    0
Y~X5 0    0    0    0    0    1    0    0
Y~X6 0    0    0    0    0    0    1    0
```

```
e.glht <- multcomp::glht(e.lvm, linfct = resC$contrast, rhs = resC$null)
summary(e.glht)
```

```
         Simultaneous Tests for General Linear Hypotheses

Fit: estimate.lvm(x = lvm(Y ~ X1 + X2 + X3 + X4 + X5 + X6), data = df.data)

Linear Hypotheses:
          Estimate Std. Error z value Pr(>|z|)
Y~X1 == 0   0.3270     0.1589   2.058  0.20725
Y~X2 == 0   0.4025     0.1596   2.523  0.06611 .
Y~X3 == 0   0.5072     0.1383   3.669  0.00144 **
Y~X4 == 0   0.3161     0.1662   1.902  0.28582
Y~X5 == 0   0.3875     0.1498   2.586  0.05554 .
Y~X6 == 0   0.3758     0.1314   2.859  0.02482 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

Note that this correction relies on the Gaussian approximation. To use small sample corrections implemented in **lavaSearch2**, just call `glht2` instead of `glht`:

```
e.glht2 <- glht2(e.lvm, linfct = resC$contrast, rhs = resC$null)
summary(e.glht2)
```

```
         Simultaneous Tests for General Linear Hypotheses

Fit: estimate.lvm(x = lvm(Y ~ X1 + X2 + X3 + X4 + X5 + X6), data = df.data)

Linear Hypotheses:
          Estimate Std. Error t value Pr(>|t|)
Y~X1 == 0   0.3270     0.1750   1.869   0.3290
Y~X2 == 0   0.4025     0.1757   2.291   0.1482
Y~X3 == 0   0.5072     0.1522   3.333   0.0123 *
Y~X4 == 0   0.3161     0.1830   1.727   0.4128
Y~X5 == 0   0.3875     0.1650   2.349   0.1315
Y~X6 == 0   0.3758     0.1447   2.597   0.0762 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
```

The single step method is the appropriate correction when one wants to report the most significant p-value relative to a set of hypotheses. If the second most significant p-value is also to be reported then the method "free" is more appropriate:

```
summary(e.glht2, test = multcomp::adjusted("free"))
```

```
         Simultaneous Tests for General Linear Hypotheses

Fit: estimate.lvm(x = lvm(Y ~ X1 + X2 + X3 + X4 + X5 + X6), data = df.data)

Linear Hypotheses:
          Estimate Std. Error t value Pr(>|t|)
Y~X1 == 0   0.3270     0.1750   1.869   0.1291
Y~X2 == 0   0.4025     0.1757   2.291   0.0913 .
Y~X3 == 0   0.5072     0.1522   3.333   0.0123 *
Y~X4 == 0   0.3161     0.1830   1.727   0.1291
Y~X5 == 0   0.3875     0.1650   2.349   0.0913 .
Y~X6 == 0   0.3758     0.1447   2.597   0.0645 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- free method)
```

Indeed, here there is no relations between the hypotheses. See the book: "Multiple Comparisons Using R" by Frank Bretz, Torsten Hothorn, and Peter Westfall (2011, CRC Press) for details about the theory underlying the **multcomp** package.

2.2 Univariate Wald test, multiple models

Pipper et al. in "A Versatile Method for Confirmatory Evaluation of the Effects of a Covariate in Multiple Models" (2012, Journal of the Royal Statistical Society, Series C) developed a method to assess the effect of an exposure on several outcomes when a different model is fitted for each outcome. This method has been implemented in the `mmm` function from the **multcomp** package for glm and Cox models. **lavaSearch2** extends it to `lvm`.

Let's consider an example where we wish to assess the treatment effect on three outcomes X, Y, and Z. We have at hand three measurements relative to outcome Z for each individual:

```
mSim <- lvm(X ~ Age + 0.5*Treatment,
            Y ~ Gender + 0.25*Treatment,
            c(Z1,Z2,Z3) ~ eta, eta ~ 0.75*treatment,
            Age[40:5]~1)
latent(mSim) <- ~eta
categorical(mSim, labels = c("placebo","SSRI")) <- ~Treatment
categorical(mSim, labels = c("male","female")) <- ~Gender

n <- 5e1
set.seed(10)
df.data <- sim(mSim, n = n, latent = FALSE)
head(df.data)
```

```
        X      Age Treatment          Y Gender         Z1         Z2          Z3
1 39.12289 39.10415   placebo  0.6088958 female  1.8714112  2.2960633 -0.09326935
2 39.56766 39.25191      SSRI  1.0001325 female  0.9709943  0.6296226  1.31035910
3 41.68751 43.05884   placebo  2.1551047 female -1.1634011 -0.3332927 -1.30769267
4 44.68102 44.78019      SSRI  0.3852728 female -1.0305476  0.6678775  0.99780139
5 41.42559 41.13105   placebo -0.8666783   male -1.6342816 -0.8285492  1.20450488
6 42.64811 41.75832      SSRI -1.0710170 female -1.2198019 -1.9602130 -1.85472132
   treatment
1  1.1639675
2 -1.5233846
3 -2.5183351
4 -0.7075292
5 -0.2874329
6 -0.4353083
```

We fit a model specific to each outcome:

```
lmX <- lm(X ~ Age + Treatment, data = df.data)
lvmY <- estimate(lvm(Y ~ Gender + Treatment), data = df.data)
lvmZ <- estimate(lvm(c(Z1,Z2,Z3) ~ 1*eta, eta ~ -1 + Treatment),
                 data = df.data)
```

and combine them into a list of `lvmfit` objects:

```
mmm.lvm <- multcomp::mmm(X = lmX, Y = lvmY, Z = lvmZ)
```

We can then generate a contrast matrix to test each coefficient related to the treatment:

```
resC <- createContrast(mmm.lvm, var.test = "Treatment", add.variance = TRUE)
resC$contrast
```

```
                     X: (Intercept) X: Age X: TreatmentSSRI X: sigma2 Y: Y
X: TreatmentSSRI                  0      0                1         0    0
Y: Y~TreatmentSSRI                0      0                0         0    0
Z: eta~TreatmentSSRI              0      0                0         0    0
                     Y: Y~Genderfemale Y: Y~TreatmentSSRI Y: Y~~Y Z: Z1 Z: Z2 Z: Z3
X: TreatmentSSRI                     0                  0       0     0     0     0
Y: Y~TreatmentSSRI                   0                  1       0     0     0     0
Z: eta~TreatmentSSRI                 0                  0       0     0     0     0
                     Z: eta~TreatmentSSRI Z: Z1~~Z1 Z: Z2~~Z2 Z: Z3~~Z3 Z: eta~~eta
X: TreatmentSSRI                        0         0         0         0           0
Y: Y~TreatmentSSRI                      0         0         0         0           0
Z: eta~TreatmentSSRI                    1         0         0         0           0
```

```
lvm.glht2 <- glht2(mmm.lvm, linfct = resC$contrast, rhs = resC$null)
summary(lvm.glht2)
```

```
         Simultaneous Tests for General Linear Hypotheses

Linear Hypotheses:
                           Estimate Std. Error t value Pr(>|t|)
X: TreatmentSSRI == 0       0.4661     0.2533   1.840    0.187
Y: Y~TreatmentSSRI == 0    -0.5421     0.2613  -2.074    0.117
Z: eta~TreatmentSSRI == 0  -0.6198     0.4404  -1.407    0.393
(Adjusted p values reported -- single-step method)
```

This can be compared to the unadjusted p.values:

```
summary(lvm.glht2, test = multcomp::univariate())
```

```
         Simultaneous Tests for General Linear Hypotheses

Linear Hypotheses:
                           Estimate Std. Error t value Pr(>|t|)
X: TreatmentSSRI == 0       0.4661     0.2533   1.840   0.0720 .
Y: Y~TreatmentSSRI == 0    -0.5421     0.2613  -2.074   0.0435 *
Z: eta~TreatmentSSRI == 0  -0.6198     0.4404  -1.407   0.1659
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Univariate p values reported)
```

3 Model diagnostic

3.1 Detection of local dependencies

The `modelsearch` function of **lava** is a diagnostic tool for latent variable models. It enables to search for local dependencies (i.e. model misspecification) and add them to the model. Obviously it is a data-driven procedure and its usefulness can be discussed, especially in small samples:

- the procedure is instable, i.e. is likely to lead to two different models when applied on two different dataset sampled from the same generative model.
- it is hard to define a meaningful significance threshold since p-values should be adjusted for multiple comparisons and sequential testing. However traditional methods like Bonferroni-Holm tend to over corrected and therefore reduce the power of the procedure since they assume that the test are independent.

The function `modelsearch2` in **lavaSearch2** partially solves the second issue by adjusting the p-values for multiple testing. Let's see an example:

```
## simulate data
mSim <- lvm(c(y1,y2,y3)~u, u~x1+x2)
latent(mSim) <- ~u
covariance(mSim) <- y2~y3
transform(mSim, Id~u) <- function(x){1:NROW(x)}
set.seed(10)
df.data <- lava::sim(mSim, n = 125, latent = FALSE)
head(df.data)
```

```
         y1           y2         y3         x1         x2 Id
1  5.5071523  4.883752014  6.2928016  0.8694750  2.3991549  1
2 -0.6398644  0.025832617  0.5088030 -0.6800096 -0.0898721  2
3 -2.5835495 -2.616715027 -2.8982645  0.1732145 -0.8216484  3
4 -2.5312637 -2.518185427 -2.9015033 -0.1594380 -0.2869618  4
5  1.6346220 -0.001877577  0.3705181  0.7934994  0.1312789  5
6  0.4939972  1.759884014  1.5010499  1.6943505 -1.0620840  6
```

```
## fit model
m <- lvm(c(y1,y2,y3)~u, u~x1)
latent(m) <- ~u
addvar(m) <- ~x2
e.lvm <- estimate(m, data = df.data)
```

`modelsearch2` can be used to sequentially apply the `modelsearch` function with a given correction for the p.values:

```
resScore <- modelsearch2(e.lvm, alpha = 0.1, trace = FALSE)
displayScore <- summary(resScore)
```

```
Sequential search for local dependence using the score statistic
The variable selection procedure retained 2 variables:
    link statistic      p.value adjusted.p.value dp.Info selected nTests
1   u~x2  6.036264 1.577228e-09     5.008615e-08       1     TRUE     10
2 y2~~y3  2.629176 8.559198e-03     6.055947e-02       1     TRUE      9
3  y3~x1  1.770997 7.656118e-02     2.814424e-01       1    FALSE      8
Confidence level: 0.9 (two sided, adjustement: fastmax)
```

This indeed matches the highest score statistic found by `modelsearch`:

```
resScore0 <- modelsearch(e.lvm, silent = TRUE)
c(statistic = sqrt(max(resScore0$test[,"Test Statistic"])),
  p.value = min(resScore0$test[,"P-value"]))
```

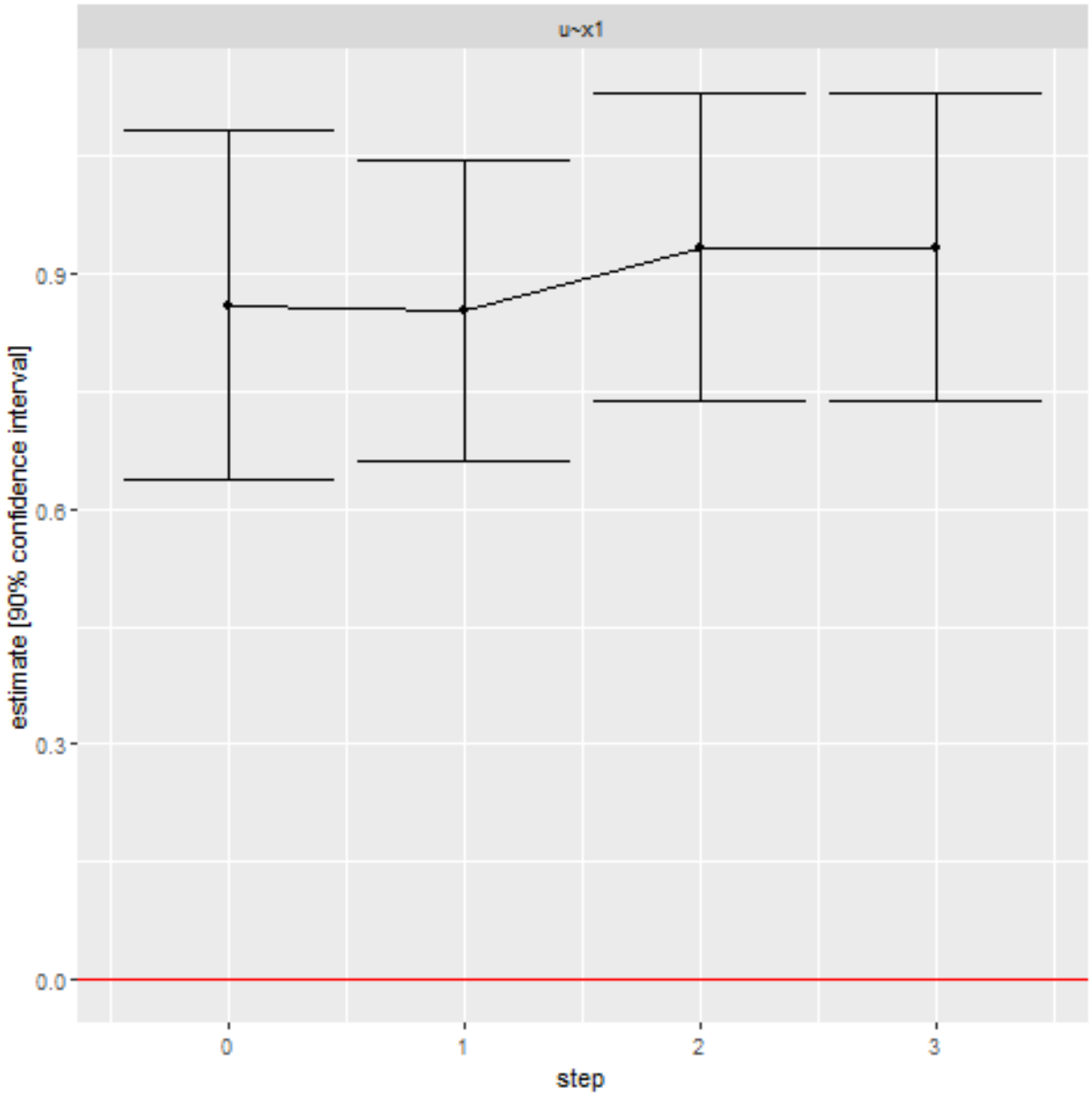
```
   statistic      p.value
6.036264e+00 1.577228e-09
```

We can compare the adjustment using the max distribution to bonferroni:

```
data.frame(link = displayScore$table[,"link"],
           none = displayScore$table[,"p.value"],
           bonferroni = displayScore$table[,"p.value"]*displayScore$table[1,"nTests"],
           max = displayScore$table[,"adjusted.p.value"])
```

```
    link         none   bonferroni          max
1   u~x2 1.577228e-09 1.577228e-08 5.008615e-08
2 y2~~y3 8.559198e-03 8.559198e-02 6.055947e-02
3  y3~x1 7.656118e-02 7.656118e-01 2.814424e-01
```

In theory, the correction based on the max statistic should give a p value that is smaller or equal than the p value adjusted using Bonferroni. However for for very small p-values, the max-correction can be numerically inaccurate and result in p-values that are slightly larger. The evolution of the estimation of a given coefficient across the sequential search can be displayed using `autoplot`:



In many cases, all links are not plausible so the user should indicates which links should be investigated by `modelsearch2`. This can be done via the argument `link`:

```
modelsearch2(e.lvm, link = c("y1~~y2","y1~~y3","y2~~y3"), trace = FALSE)
```

```
Sequential search for local dependence using the score statistic
The variable selection procedure did not retain any variable
    link statistic    p.value adjusted.p.value dp.Info selected nTests
1 y1~~y3  1.754102 0.07941299        0.1819309       1    FALSE      3
Confidence level: 0.95 (two sided, adjustement: fastmax)
```

The function `findNewLink` can help the user to identify the set of relevant links:

```
findNewLink(e.lvm$model, type = "covariance")$link
```

```
[1] "y1~~y2" "y1~~y3" "y2~~y3"
```

3.2 Checking that the names of the variables in the model match those of the data

When estimating latent variable models using **lava**, it sometimes happens that the model does not converge:

```
## simulate data
set.seed(10)
df.data <- sim(lvm(Y~X1+X2), 1e2)

## fit model
mWrong <- lvm(Y ~ X + X2)
eWrong <- estimate(mWrong, data = df.data)
```

```
Warning messages:
1: In estimate.lvm(mWrong, data = df.data) :
  Lack of convergence. Increase number of iteration or change starting values.
2: In sqrt(diag(asVar)) : NaNs produced
```

This can have several reasons:

- the model is not identifiable.
- the optimization routine did not managed to find a local optimum. This may happen for complex latent variable model where the objective function is not convex or locally convex.
- the user has made a mistake when defining the model or has not given the appropriate dataset.

The `checkData` function enables to check the last point. It compares the observed variables defined in the model and the one given by the dataset. In case of mismatch it returns a message:

```
checkData(mWrong, df.data)
```

```
Missing variable in data: X
```

In presence of latent variables, the user needs to explicitely define them in the model, otherwise `checkData` will identify them as an issue:

```
## simulate data
set.seed(10)
mSim <- lvm(c(Y1,Y2,Y3)~eta)
latent(mSim) <- ~eta
df.data <- sim(mSim, n = 1e2, latent = FALSE)

## fit model
m <- lvm(c(Y1,Y2,Y3)~eta)
checkData(m, data = df.data)
```

```
Missing variable in data: eta
```

```
latent(m) <- ~eta
checkData(m, data = df.data)
```

```
No issue detected
```

4 Information about the R session used for this document

```
sessionInfo()
```

```
R version 3.5.1 (2018-07-02)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 7 x64 (build 7601) Service Pack 1

Matrix products: default

locale:
[1] LC_COLLATE=Danish_Denmark.1252  LC_CTYPE=Danish_Denmark.1252
[3] LC_MONETARY=Danish_Denmark.1252 LC_NUMERIC=C
[5] LC_TIME=Danish_Denmark.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] lavaSearch2_1.4 ggplot2_3.0.0   lava_1.6.3

loaded via a namespace (and not attached):
 [1] Rcpp_0.12.19      pillar_1.3.0      compiler_3.5.1    plyr_1.8.4
 [5] bindr_0.1.1       tools_3.5.1       tibble_1.4.2      gtable_0.2.0
 [9] lattice_0.20-35   pkgconfig_2.0.2   rlang_0.2.2       Matrix_1.2-14
[13] parallel_3.5.1    mvtnorm_1.0-8     bindrcpp_0.2.2    withr_2.1.2
[17] dplyr_0.7.6       stringr_1.3.1     grid_3.5.1        tidyselect_0.2.4
[21] glue_1.3.0        R6_2.2.2          survival_2.42-6   multcomp_1.4-8
[25] TH.data_1.0-9     purrr_0.2.5       reshape2_1.4.3    magrittr_1.5
[29] scales_1.0.0      codetools_0.2-15  MASS_7.3-50       splines_3.5.1
[33] assertthat_0.2.0  colorspace_1.3-2  numDeriv_2016.8-1 sandwich_2.5-0
[37] stringi_1.2.4     lazyeval_0.2.1    munsell_0.5.0     crayon_1.3.4
[41] zoo_1.8-4
```