Overview of the functionalities of the package lavaSearch2

Brice Ozenne

March 18, 2020

Load `lavaSearch2` in the R session:

```r
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`:

```r
## 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 the corresponding latent variable model estimated by maximum likelihood:

```r
## 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 0.2205080 2.0855736 0.052401103
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 in the latent variable model and 0.052 in the linear regression. 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 t-distribution instead of a Gaussian distribution (informally using a t-value instead of z-value).
- use of an unbiased 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 exactly match the results of lm:

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

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Std. Error</th>
<th>t-value</th>
<th>P-value</th>
<th>df</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y~X1</td>
<td>0.1550938</td>
<td>0.2205078</td>
<td>0.7033483</td>
<td>0.49136012</td>
</tr>
<tr>
<td>Y~X2</td>
<td>0.4581556</td>
<td>0.2196783</td>
<td>2.0855754</td>
<td>0.05240092</td>
</tr>
<tr>
<td>Y~Y</td>
<td>0.6538707</td>
<td>0.2242758</td>
<td>2.9154759</td>
<td>NA</td>
</tr>
<tr>
<td>Y</td>
<td>0.7967775</td>
<td>0.2506765</td>
<td>3.1785096</td>
<td>0.00549580</td>
</tr>
</tbody>
</table>

### 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 and maximum likelihood estimation. While being asymptotically valid, they usually do 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. lavaSearch2 provides two improvements:

- using a Student’s t-distribution instead of a Gaussian 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.
- (partially) correcting for the first order bias in the ML estimates of the variance parameters. This correction also affects the standard error of the estimates.

### 1.3 Single univariate Wald test

We will illustrate the functionalities using a simulated dataset:

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

```r
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 the traditional approach using `summary`:

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

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>Std. Error</th>
<th>Z-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2</td>
<td>0.2335412</td>
<td>0.2448593</td>
<td>0.9537775</td>
<td>0.3401963</td>
</tr>
<tr>
<td>Y3</td>
<td>0.5114275</td>
<td>0.1785886</td>
<td>2.8637186</td>
<td>0.0041870</td>
</tr>
<tr>
<td>Y2~eta</td>
<td>0.9192847</td>
<td>0.2621248</td>
<td>3.5070497</td>
<td>0.0004531</td>
</tr>
<tr>
<td>Y3~eta</td>
<td>0.2626930</td>
<td>0.1558978</td>
<td>1.6850339</td>
<td>0.1419820</td>
</tr>
<tr>
<td>eta~X1</td>
<td>0.5150072</td>
<td>0.2513393</td>
<td>2.0490515</td>
<td>0.0405707</td>
</tr>
<tr>
<td>eta~X2</td>
<td>0.6212222</td>
<td>0.2118930</td>
<td>2.9317729</td>
<td>0.0033703</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>Std. Error</th>
<th>t-value</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2</td>
<td>0.2335412</td>
<td>0.2518218</td>
<td>0.9274067</td>
<td>0.3715161</td>
</tr>
<tr>
<td>Y3</td>
<td>0.5114275</td>
<td>0.1828716</td>
<td>2.7966475</td>
<td>0.0098488</td>
</tr>
<tr>
<td>Y2~eta</td>
<td>0.9192847</td>
<td>0.2653220</td>
<td>3.4647887</td>
<td>0.0315856</td>
</tr>
<tr>
<td>Y3~eta</td>
<td>0.2626930</td>
<td>0.1562776</td>
<td>1.6809386</td>
<td>0.1438266</td>
</tr>
<tr>
<td>eta~X1</td>
<td>0.5150072</td>
<td>0.2642257</td>
<td>1.9491180</td>
<td>0.0564161</td>
</tr>
<tr>
<td>eta~X2</td>
<td>0.6212222</td>
<td>0.2118930</td>
<td>2.9317729</td>
<td>0.0033703</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>Std. Error</th>
<th>t-value</th>
<th>P-value</th>
<th>df</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2</td>
<td>0.2335412</td>
<td>0.2518218</td>
<td>0.9274067</td>
<td>0.3715161</td>
<td>12.328385</td>
</tr>
<tr>
<td>Y3</td>
<td>0.5114275</td>
<td>0.1828716</td>
<td>2.7966475</td>
<td>0.0098488</td>
<td>24.707696</td>
</tr>
<tr>
<td>Y2~eta</td>
<td>0.9192847</td>
<td>0.2653220</td>
<td>3.4647887</td>
<td>0.0315856</td>
<td>3.515034</td>
</tr>
<tr>
<td>Y3~eta</td>
<td>0.2626930</td>
<td>0.1562776</td>
<td>1.6809386</td>
<td>0.1438266</td>
<td>5.993407</td>
</tr>
<tr>
<td>eta~X1</td>
<td>0.5150072</td>
<td>0.2642257</td>
<td>1.9491180</td>
<td>0.0564161</td>
<td>20.044312</td>
</tr>
<tr>
<td>eta~X2</td>
<td>0.6212222</td>
<td>0.2118930</td>
<td>2.9317729</td>
<td>0.0033703</td>
<td>27.718363</td>
</tr>
</tbody>
</table>

```

3
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.

To only use the bias correction, set the argument df to FALSE:

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

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Std. Error</th>
<th>t-value</th>
<th>P-value</th>
<th>df</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2</td>
<td>0.2335412</td>
<td>0.2518218</td>
<td>0.9274067</td>
<td>Inf</td>
</tr>
<tr>
<td>Y3</td>
<td>0.5114275</td>
<td>0.1828716</td>
<td>2.7966475</td>
<td>Inf</td>
</tr>
<tr>
<td>Y2~eta</td>
<td>0.9192847</td>
<td>0.2653220</td>
<td>3.4647887</td>
<td>Inf</td>
</tr>
<tr>
<td>Y3~eta</td>
<td>0.2626930</td>
<td>0.1562776</td>
<td>1.6809386</td>
<td>Inf</td>
</tr>
<tr>
<td>eta~X1</td>
<td>0.5150072</td>
<td>0.2642257</td>
<td>1.9491180</td>
<td>Inf</td>
</tr>
<tr>
<td>eta~X2</td>
<td>0.6212222</td>
<td>0.2221293</td>
<td>2.7966698</td>
<td>Inf</td>
</tr>
</tbody>
</table>

### 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.

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

user  system  elapsed
0.25   0.00   0.25

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):

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

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

```r
class(e2)
```

[1] "lvmfit2" "lvmfit"

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

```r
summary2(e2)$coef[c("Y2","Y3","Y2~eta","Y3~eta","eta~X1","eta~X2"), ]
```
<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>Std. Error</th>
<th>t-value</th>
<th>P-value</th>
<th>df</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2</td>
<td>0.2335412</td>
<td>0.2518218</td>
<td>0.9274067</td>
<td>0.371516094</td>
<td>12.328385</td>
</tr>
<tr>
<td>Y3</td>
<td>0.5114275</td>
<td>0.1828716</td>
<td>2.7966475</td>
<td>0.009848769</td>
<td>24.707696</td>
</tr>
<tr>
<td>Y2~eta</td>
<td>0.9192847</td>
<td>0.2653220</td>
<td>3.4647887</td>
<td>0.031585600</td>
<td>3.515034</td>
</tr>
<tr>
<td>Y3~eta</td>
<td>0.2626930</td>
<td>0.1562776</td>
<td>1.6809386</td>
<td>0.143826633</td>
<td>5.993407</td>
</tr>
<tr>
<td>eta~X1</td>
<td>0.5150072</td>
<td>0.2642257</td>
<td>1.9491180</td>
<td>0.065414617</td>
<td>20.044312</td>
</tr>
<tr>
<td>eta~X2</td>
<td>0.6212222</td>
<td>0.2212932</td>
<td>2.7966698</td>
<td>0.009275494</td>
<td>27.718363</td>
</tr>
</tbody>
</table>

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

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

```
user  system elapsed
0.19   0.00   0.19
```

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

```
user  system elapsed
0.15   0.00   0.16
```

### 1.5 Single multivariate Wald test

The function `compare` from the lava package 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`:

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

```r
$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
[Y2-eta] = 0  0  0  0  0  1  0  0  0  0  0  0  0
[eta-X1] = 0  0  0  0  0  0  0  1  0  0  0  0  0
[eta-eta] = 0  0  0  0  0  0  0  0  0  0  0  0  0

$null
```

5
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):

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

- Wald test -

Null Hypothesis:

- \( Y_2 \) = 0
- \( Y_2 - \eta \) = 0
- \( \eta - X_1 \) = 0

data:

chisq = 21.332, df = 3, p-value = 8.981e-05
sample estimates:

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Std.Err</th>
<th>2.5%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Y_2 )</td>
<td>0.2335412</td>
<td>-0.2463741</td>
<td>0.7134566</td>
</tr>
<tr>
<td>( Y_2 - \eta )</td>
<td>0.9192847</td>
<td>0.4055295</td>
<td>1.4330399</td>
</tr>
<tr>
<td>( \eta - X_1 )</td>
<td>0.5150072</td>
<td>0.0223912</td>
<td>1.0076231</td>
</tr>
</tbody>
</table>

`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`:

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

- Wald test -

Null Hypothesis:

- \( Y_2 \) = 0
- \( Y_2 - \eta \) = 0
- \( \eta - X_1 \) = 0

data:

F-statistic = 6.7118, df1 = 3, df2 = 11.1, p-value = 0.007596
sample estimates:

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Std.Err</th>
<th>df</th>
<th>2.5%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Y_2 )</td>
<td>0.2335412</td>
<td>12.328385</td>
<td>-0.3135148</td>
<td>0.7805973</td>
</tr>
<tr>
<td>( Y_2 - \eta )</td>
<td>0.9192847</td>
<td>3.515034</td>
<td>0.1407653</td>
<td>1.6978041</td>
</tr>
<tr>
<td>( \eta - X_1 )</td>
<td>0.5150072</td>
<td>20.044312</td>
<td>-0.0360800</td>
<td>1.0660943</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Estimate</th>
<th>Std.Err</th>
<th>df</th>
<th>2.5%</th>
<th>97.5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Y2] = 0</td>
<td>0.2335412</td>
<td>0.2448593</td>
<td>12.91188</td>
<td>-0.295812256</td>
</tr>
<tr>
<td>[Y2-eta] = 0</td>
<td>0.9192847</td>
<td>0.2621248</td>
<td>3.67464</td>
<td>0.165378080</td>
</tr>
<tr>
<td>[eta-X1] = 0</td>
<td>0.5150072</td>
<td>0.2513393</td>
<td>21.57121</td>
<td>-0.006840023</td>
</tr>
</tbody>
</table>

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"), ]
```
<table>
<thead>
<tr>
<th>Estimate</th>
<th>robust SE</th>
<th>t-value</th>
<th>P-value</th>
<th>df</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y2</td>
<td>0.2335412</td>
<td>0.2353245</td>
<td>0.9924222</td>
<td>0.3340117610</td>
</tr>
<tr>
<td>Y3</td>
<td>0.5114275</td>
<td>0.1897160</td>
<td>2.6957535</td>
<td>0.0099985389</td>
</tr>
<tr>
<td>Y2-eta</td>
<td>0.9192847</td>
<td>0.1791240</td>
<td>5.1321150</td>
<td>0.0002361186</td>
</tr>
<tr>
<td>Y3-eta</td>
<td>0.2626930</td>
<td>0.1365520</td>
<td>1.9237585</td>
<td>0.0653095551</td>
</tr>
<tr>
<td>eta-X1</td>
<td>0.5150072</td>
<td>0.2167580</td>
<td>2.3759546</td>
<td>0.0315112789</td>
</tr>
<tr>
<td>eta-X2</td>
<td>0.6212222</td>
<td>0.2036501</td>
<td>3.0504389</td>
<td>0.0035239307</td>
</tr>
</tbody>
</table>

or multivariate Wald test:

```r
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 = 23.97, p-value = 3.981e-05

sample estimates:

<table>
<thead>
<tr>
<th>Estimate</th>
<th>robust SE</th>
<th>t-value</th>
<th>P-value</th>
<th>df</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Y2] = 0</td>
<td>0.2335412</td>
<td>0.2353245</td>
<td>0.9924222</td>
<td>0.3340117610</td>
</tr>
<tr>
<td>[Y2-eta] = 0</td>
<td>0.9192847</td>
<td>0.1791240</td>
<td>5.1321150</td>
<td>0.0002361186</td>
</tr>
<tr>
<td>[eta-X1] = 0</td>
<td>0.5150072</td>
<td>0.2167580</td>
<td>2.3759546</td>
<td>0.0315112789</td>
</tr>
</tbody>
</table>

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:

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

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:

```r
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:

```r
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:

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

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

```r
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:

```r
mySimulation <- calibrateType1(e,
    param = "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:

```r
summary(mySimulation)
```

Estimated type 1 error rate [95% confidence interval]

> sample size: 50 | number of simulations: 50

<table>
<thead>
<tr>
<th>link statistic</th>
<th>correction</th>
<th>type1error</th>
<th>CI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y-Genell Wald</td>
<td>Gaus</td>
<td>0.12</td>
<td>[0.05492 ; 0.24242]</td>
</tr>
<tr>
<td></td>
<td>Satt</td>
<td>0.10</td>
<td>[0.04224 ; 0.21869]</td>
</tr>
<tr>
<td></td>
<td>SSC</td>
<td>0.10</td>
<td>[0.04224 ; 0.21869]</td>
</tr>
<tr>
<td></td>
<td>SSC + Satt</td>
<td>0.08</td>
<td>[0.03035 ; 0.19456]</td>
</tr>
</tbody>
</table>

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:

```r
## 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-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      0.3270 0.1589           2.058 0.002725 |
| Y-X2 == 0                     | 0      0.4025 0.1596           2.523 0.012054 |
| Y-X3 == 0                     | 0      0.5072 0.1383           3.669 0.000144 **
| Y-X4 == 0                     | 0      0.3161 0.1662           1.902 0.055682 |
| Y-X5 == 0                     | 0      0.3875 0.1498           2.586 0.007554 .
| Y-X6 == 0                     | 0      0.3758 0.1314           2.859 0.004582 *

---

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:

```r
  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 efficient:

```r
  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)

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:

```r
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)
```

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Age</th>
<th>Treatment</th>
<th>Y</th>
<th>Gender</th>
<th>Z1</th>
<th>Z2</th>
<th>Z3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>39.1229</td>
<td>39.10415</td>
<td>placebo</td>
<td>0.6088958</td>
<td>female</td>
<td>1.8714112</td>
<td>2.2960633</td>
<td>-0.09326935</td>
</tr>
<tr>
<td>2</td>
<td>39.56766</td>
<td>39.25191</td>
<td>SSRI</td>
<td>1.0001325</td>
<td>female</td>
<td>0.9709943</td>
<td>0.6296226</td>
<td>1.31035910</td>
</tr>
<tr>
<td>3</td>
<td>41.68751</td>
<td>43.05884</td>
<td>placebo</td>
<td>2.1551047</td>
<td>female</td>
<td>-1.1634011</td>
<td>-0.3332927</td>
<td>-1.30769267</td>
</tr>
<tr>
<td>4</td>
<td>44.68102</td>
<td>44.78019</td>
<td>SSRI</td>
<td>0.3852728</td>
<td>female</td>
<td>-1.0305476</td>
<td>0.6678775</td>
<td>0.99780139</td>
</tr>
<tr>
<td>5</td>
<td>41.42559</td>
<td>41.13105</td>
<td>placebo</td>
<td>-0.8666783</td>
<td>male</td>
<td>-1.6342816</td>
<td>-0.8285492</td>
<td>1.20450488</td>
</tr>
<tr>
<td>6</td>
<td>42.64811</td>
<td>41.75832</td>
<td>SSRI</td>
<td>-1.0710170</td>
<td>female</td>
<td>-1.2198019</td>
<td>-1.9602130</td>
<td>-1.85472132</td>
</tr>
</tbody>
</table>

We fit a model specific to each outcome:

```r
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:

```r
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

X: TreatmentSSRI 0 0 0 0 0
Y: Y-TreatmentSSRI 0 1 0 0 0
Z: eta-TreatmentSSRI 0 0 0 0 0

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:

| Estimater | 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:

| Estimater | 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:

```r
## 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 5.507 4.883 6.293 0.869 2.399 1
2-0.64  0.026 0.509-0.68  0.11 -2
3-2.58 2.617 0.715 0.173-0.82  3
4-2.53 2.518 0.901-0.16  0.29  4
5 1.64  0.37  0.79  0.13  0.13  5
6 0.49  1.75  1.50  1.69-1.06  6
```

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

```r
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:

<table>
<thead>
<tr>
<th>link statistic</th>
<th>statistic</th>
<th>p.value</th>
<th>adjusted.p.value</th>
<th>dp.Info</th>
<th>selected</th>
<th>nTests</th>
</tr>
</thead>
<tbody>
<tr>
<td>u~x2</td>
<td>6.036264</td>
<td>1.577228e-09</td>
<td>5.008615e-08</td>
<td>1</td>
<td>TRUE</td>
<td>10</td>
</tr>
<tr>
<td>y2~~y3</td>
<td>2.629176</td>
<td>8.559198e-03</td>
<td>6.055947e-02</td>
<td>1</td>
<td>TRUE</td>
<td>9</td>
</tr>
<tr>
<td>y3~x1</td>
<td>1.770997</td>
<td>7.656118e-02</td>
<td>2.814424e-01</td>
<td>1</td>
<td>FALSE</td>
<td>8</td>
</tr>
</tbody>
</table>

Confidence level: 0.9 (two sided, adjustment: fastmax)

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

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

<table>
<thead>
<tr>
<th>statistic</th>
<th>p.value</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.036264e+00</td>
<td>1.577228e-09</td>
</tr>
</tbody>
</table>

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

```r
data.frame(link = displayScore$table[,"link"],
one = displayScore$table[,"p.value"],
bonferroni = displayScore$table[,"p.value"]*displayScore$table[1,"nTests"],
max = displayScore$table[,"adjusted.p.value"])
```

<table>
<thead>
<tr>
<th>link</th>
<th>none</th>
<th>bonferroni</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>u~x2</td>
<td>1.577228e-09</td>
<td>1.577228e-08</td>
<td>5.008615e-08</td>
</tr>
<tr>
<td>y2~~y3</td>
<td>8.559198e-03</td>
<td>8.559198e-02</td>
<td>6.055947e-02</td>
</tr>
<tr>
<td>y3~x1</td>
<td>7.656118e-02</td>
<td>7.656118e-01</td>
<td>2.814424e-01</td>
</tr>
</tbody>
</table>

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 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`:

```r
resRed <- modelsearch2(e.lvm, link = c("y1~y2","y1~y3","y2~y3"), trace = FALSE)
print(resRed)
```

Sequential search for local dependence using the score statistic
The variable selection procedure did not retain any variable

<table>
<thead>
<tr>
<th>link statistic</th>
<th>p.value</th>
<th>adjusted.p.value</th>
<th>dp.Info</th>
<th>selected</th>
<th>nTests</th>
</tr>
</thead>
<tbody>
<tr>
<td>y1~y3</td>
<td>1.754102</td>
<td>0.07941299</td>
<td>0.1818963</td>
<td>FALSE</td>
<td>3</td>
</tr>
</tbody>
</table>

Confidence level: 0.95 (two sided, adjustment: fastmax)

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

```r
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:

```r
## 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:

```r
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:

```r
## 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_1.5.1 lava_1.6.4 ggplot2_3.1.0

loaded via a namespace (and not attached):
[1] Rcpp_1.0.0 pillar_1.3.1 compiler_3.5.1 plyr_1.8.4
[5] bindr_0.1.1 tools_3.5.1 tibble_2.0.1 gtable_0.2.0
[9] lattice_0.20-35 pkgconfig_2.0.2 rlang_0.3.1 Matrix_1.2-14
[13] parallel_3.5.1 mvtnorm_1.0-8 bindrcpp_0.2.2 withr_2.1.2
[17] dplyr_0.7.8 stringr_1.3.1 grid_3.5.1 tidyselect_0.2.5
[21] glue_1.3.0 R6_2.3.0 survival_2.42-6 multcomp_1.4-8
[25] TH.data_1.0-9 purrr_0.3.0 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 labeling_0.3
[37] sandwich_2.5-0 stringi_1.2.4 lazyeval_0.2.1 munsell_0.5.0
[41] crayon_1.3.4 zoo_1.8-4

20