Multicollinearity and Heteroscedasticity

Jan Rovny

Multicollinearity and Heteroscedasticity and potential problems that prevent correct estimation of standard errors, and can consequently lead to erroneous hypohtesis tests about the significance of predicted coefficients.

Collinearity

Collinearity occurrs when two or more predictors are highly correlated. Let's create a dataset with four variables x, z, r and y, which contains some collinearity:

```
set.seed(243)
#creat collinear data
x<-rnorm(50)
z<-0.3*x+rnorm(50,0,0.5) #z is slightly correlated with x
r<-9*x+rnorm(50,0,0.2) #r is strongly correlated with x
y<-0+1*x+2*z+3*r+rnorm(50,0,3) #y is a function of x, z, and r
```

Notice that y is a function of x, r and z. Notice also that z is slightly correlated with x, while r is strongly correlated with x.

Let's now run a model predicting y with x, r and z. We know that this is the correct specification of the model.

```
#collinear model
model<-lm(y~x+z+r)
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x + z + r)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.7596 -1.9971 -0.5831  2.2013  5.5728
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.00159    0.42634   0.004   0.9970
## x            3.94076   17.76039   0.222   0.8254
## z            2.19920    0.96580   2.277   0.0275 *
## r            2.62481    1.98149   1.325   0.1918
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.957 on 46 degrees of freedom
## Multiple R-squared:  0.9918, Adjusted R-squared:  0.9912
## F-statistic:  1849 on 3 and 46 DF,  p-value: < 2.2e-16
```

First thing to notice is that we have a large R^2, but limited significance of individual predictors. Despite the fact that y is a function of all three predictors: x, z, and r, only z reaches conventional levels of statistical significance, while x and r are insignificant.

If we did not know the correct model specification we might wonder whether x and r have any effect on y. But in this case, we know they do. We just cannot see the effect because it is collinear. Said differently, we cannot assess the independent impact of x on y while holding r constant, because when we hold r constant, x does not change much. There is little independent variance of x.

How could we assess this in situations where we did not know the correct model specification? The first thing to always look at is the correlation matrix between predictors.

```
#check correlations
corr<-cbind(x,z,r) #here column bind the three predictors into an object `corr'
cor(corr) # run the correlations
```

```
##           x         z         r
## x 1.0000000 0.6318832 0.9997523
## z 0.6318832 1.0000000 0.6366779
## r 0.9997523 0.6366779 1.0000000
```

The correlation matrix clearly demonstrates the high level of collinearity in our data. X and z are correlated at 0.63, while x and r are correlated at 0.99 – which means that they are almost identical.

Most real-world data will not have such strong correlations, and so the correlation matrix may not be a sufficient way to assess collinearity. The reason for this is that collinearity may be caused by lack of one predictor's independent variance, variance which overlaps with not just one, but a number of the other predictors. A good way to assess the extent to which a predictor can vary independently of the other preditors in a model, is to measure its *tolerance*. Tolerance measures the share of a predictor that is independent of all other predictors. It is calculated as a function of so-called *variance inflation factor* (VIF). It can be done in the following way:

```
#assess collinearity
library(car)
```

```
## Loading required package: carData
```

```
vif(model)
```

```
##           x           z           r
## 2177.783015    1.814114 2200.058770
```

```
#above are variance inflation factors
tol<-1/vif(model) #calculating tolerance
list(tol)
```

```
## [[1]]
##            x            z            r
## 0.0004591826 0.5512333694 0.0004545333
```

```
#above are the tolerance values
```

We may assess multicollinearity by looking at the variance inflation factors (it is usually said that they should be below the value of 10). However, a much better measure is the measure of tolerance which $= \frac{1}{VIF}$. We can see from the results above that over 55% of z can vary independently of x and r, while only a tiny portion of the variance of x and r are independent. Clearly, we cannot assess the effect of x when controlling for r and vice versa.

How can we deal with this problem? One simple observation we made above was that x and r were almost identical. If two predictors are highly correlated, we can reasonably assume that they somehow capture the same phenomenon. Consequently, we could combine them into one common factor and use this instead.

One way of going about this is to use a method called *principle component analysis* (PCA) that finds the commonality of variables:

```r
#Principle Component analysis -- to create one variable out of x and r
xr<-cbind(x,r) #combine x and r in one matrix
pca<-prcomp(xr) #this runs the PCA
print(pca)        #this tells us the loadings of each variable
```

```
## Standard deviations (1, .., p=2):
## [1] 10.06173003  0.02455493
##
## Rotation (n x k) = (2 x 2):
##         PC1        PC2
## x 0.1102977 -0.9938986
## r 0.9938986  0.1102977
```

```r
summary(pca)      #this gives us the summary of importance of the components
```

```
## Importance of components:
##                          PC1     PC2
## Standard deviation     10.06 0.02455
## Proportion of Variance  1.00 0.00001
## Cumulative Proportion   1.00 1.00000
```

```r
new<-predict(pca)[,1] #this creates a new variable based on the first component
```

We are now in a position to rerun our model using the new variable created above:

```r
model2<-lm(y~new+z)
summary(model2)
```

```
##
## Call:
## lm(formula = y ~ new + z)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.6715 -1.9942 -0.5142  2.2277  5.7313
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.01987    0.41442  12.113 4.64e-16 ***
## new          3.04542    0.05389  56.515  < 2e-16 ***
## z            2.14619    0.92029   2.332    0.024 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.927 on 47 degrees of freedom
## Multiple R-squared:  0.9918, Adjusted R-squared:  0.9914
## F-statistic:  2831 on 2 and 47 DF,  p-value: < 2.2e-16
```

The new variable is highly significant. Clearly, this is a different model, it is modelling the effect of a common dimension of x and r on y, but it seems to be capturing the relationship.

Heteroscedasticity

Heteroscedasticity occurrs when we do not have a constant variance of errors. It causes problems with the estimation of standard errors and hypothesis testing.

Let's create a dataset containing heteroscedastic errors:

```r
set.seed(234)
A1=rnorm(1000,0,1)
A2=rnorm(1000,0,2)
A3=rnorm(1000,3,0.5)
E=rnorm(1000,0,1)
A4=4+0.4*A1+0.7*A2-1.4*A3+E*0.5*A3 #note that the error term is a function of A3
```

Note that this model predicts A4 as a function of A1, A2 and A3. Also note that the error term of the model is a function of A3. As A3 increases, the error increases. This means that the error variance is not constant, but increases with A3. Our model consequently suffers from heteroscedasticity! Let's predict A4:

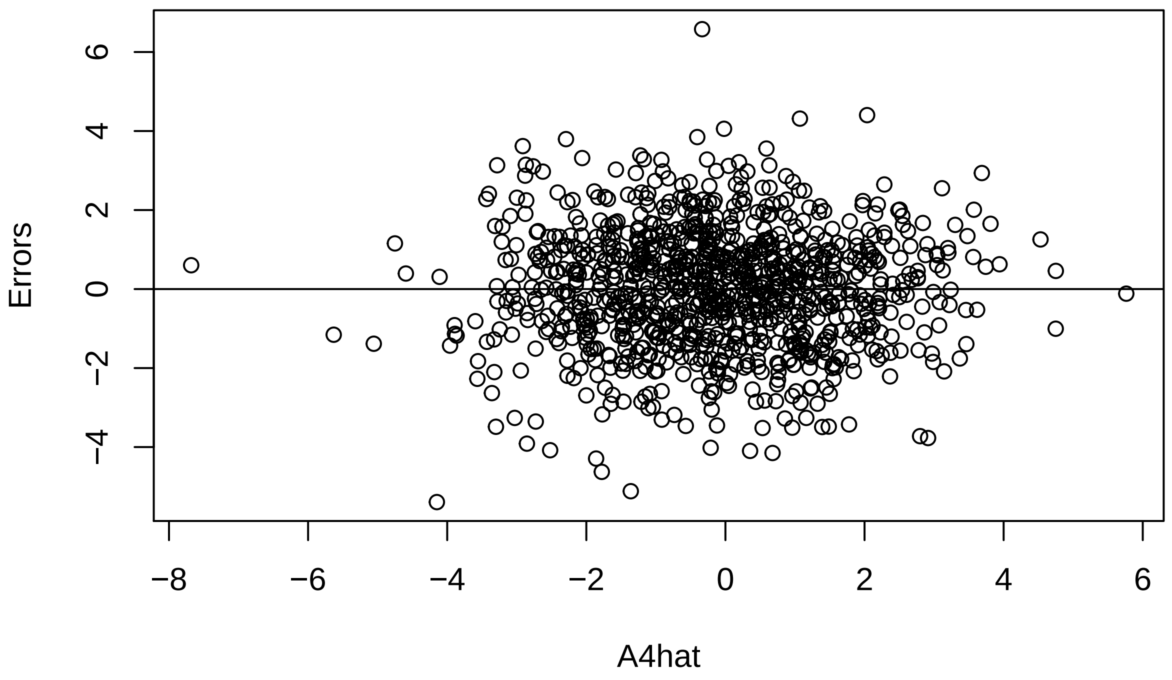
```r
het<-lm(A4~A1+A2+A3, x=T, y=T)
summary(het)
```

```
##
## Call:
## lm(formula = A4 ~ A1 + A2 + A3, x = T, y = T)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.3920 -1.0108  0.0359  0.9829  6.5782
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.52279    0.30156  14.998   <2e-16 ***
## A1           0.47747    0.04847   9.851   <2e-16 ***
## A2           0.71528    0.02397  29.841   <2e-16 ***
## A3          -1.56445    0.09957 -15.711   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.508 on 996 degrees of freedom
## Multiple R-squared:  0.5433, Adjusted R-squared:  0.542
## F-statistic:   395 on 3 and 996 DF,  p-value: < 2.2e-16
```

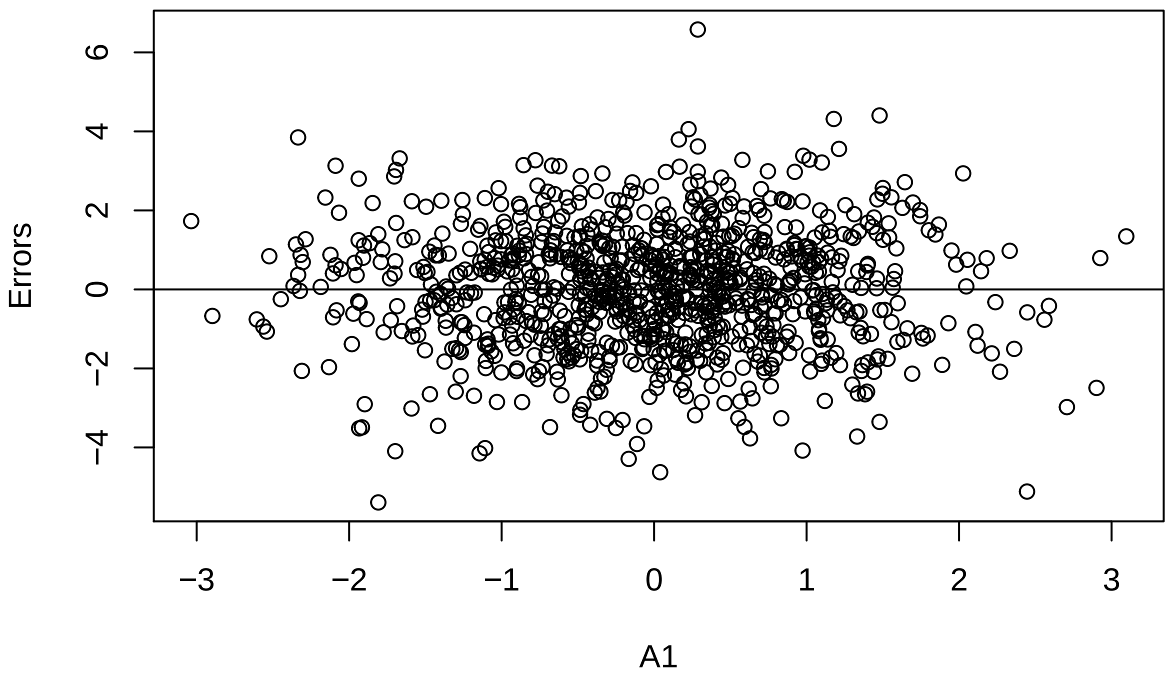
Although all our results are significant, the heteroscedastic error effected proper estimation of the standard errors. How can we test to see whether heteroscedastic error is present?

The first thing to consider is a plot of the model residuals (or errors):

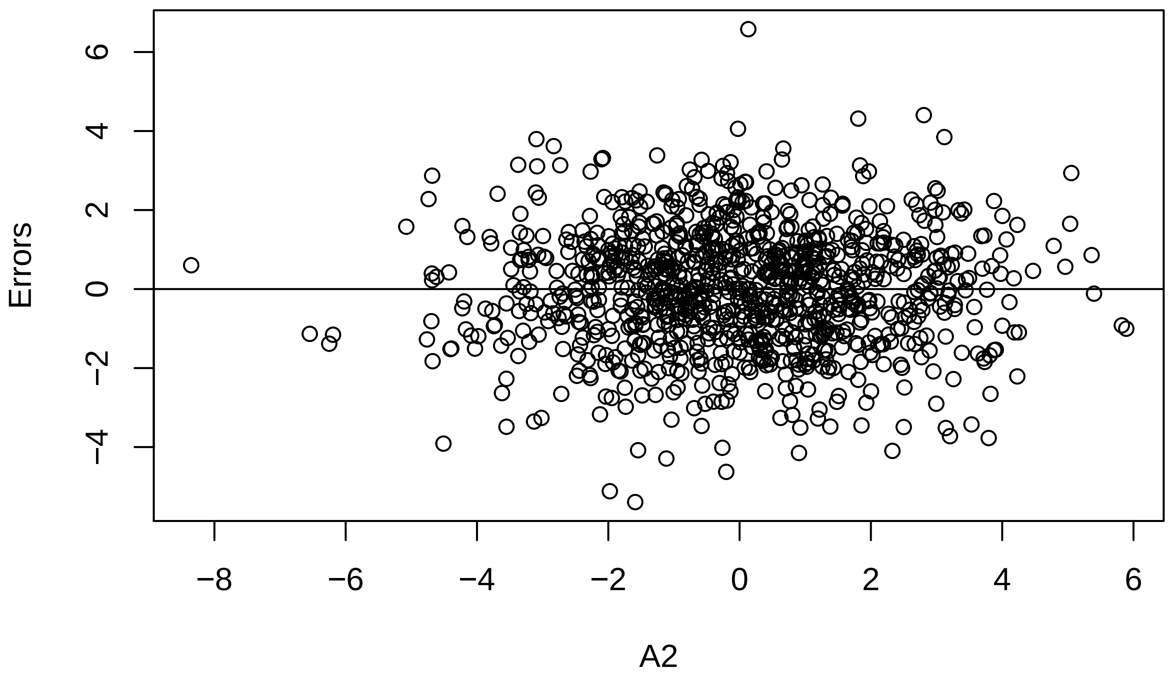
```r
#test for heteroscedasticity
#1) plotting
Errors <- residuals(het) #this creates an object containing the errors of the model
A4hat <- fitted(het) #this creates an object with the fitted values

plot(A4hat,Errors) #this produces the plot of fitted versus residuals
abline(h=0)
```



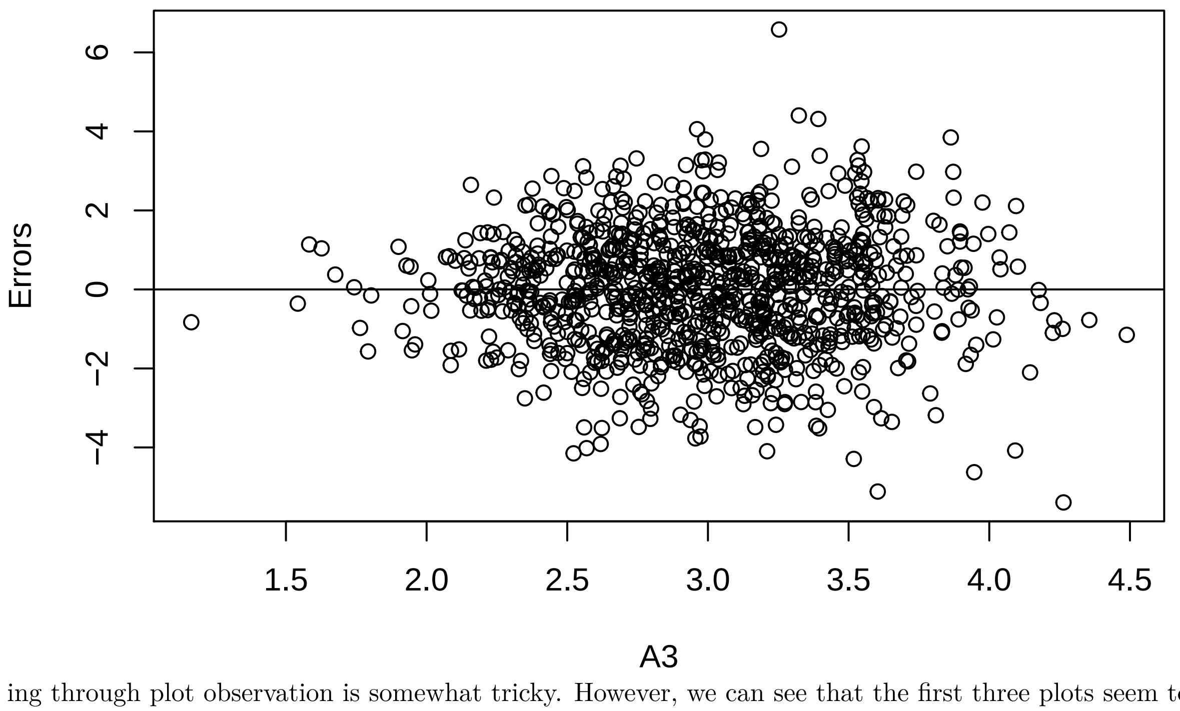
```r
plot(A1,Errors) #this produces the plot of A1 versus residuals
abline(h=0)
```



```r
plot(A2,Errors) #this produces the plot of A2 versus residuals
abline(h=0)
```



```r
plot(A3,Errors) #this produces the plot of A3 versus residuals
abline(h=0)
```



Testing through plot observation is somewhat tricky. However, we can see that the first three plots seem to show relatively random, and even band of errors around the line. However, the final plot shows some 'fanning' of the errors. The errors at the higher values of A3 have much greater variance than errors at the lower values of A3.

However, this can be hard to observe, and thus is an unreliable way to test heteroscedasticity. Let us turn to a statistical test instead:

```r
library(car)
ncvTest(het)
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 5.346474    Df = 1     p = 0.02076422
```

The H_0 in this test is 'constant variance', and consequently, we want to fail to reject H_0. However, in this case we see that the p-value is relatively small (smaller than 0.05), and thus we reject H_0, and conclude that our model is heteroscedastic (has non-constant variance).

How do we remedy the problem of non-constant variance? The best soultion is to re-estimate our standard errors using so-called *robust standard errors*. This estimation is a little tricky in R. We first need to import a user-written function for calculating robust standard errors:

```
# load necessary packages for importing the function
library(RCurl)
```

```
## Loading required package: bitops
```

```
# import the function from repository
url_r<-"https://raw.githubusercontent.com/IsidoreBeautrelet/economictheoryblog/master/robust_summary.R"
eval(parse(text = getURL(url_r, ssl.verifypeer = FALSE)),
     envir=.GlobalEnv)
```

Now we can re-estimate our model with robust standard errors:

```
summary(het, robust=T) #calculate robust se
```

```
##
## Call:
## lm(formula = A4 ~ A1 + A2 + A3, x = T, y = T)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.3920 -1.0108  0.0359  0.9829  6.5782
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.52279    0.28585  15.822   <2e-16 ***
## A1           0.47747    0.05181   9.216   <2e-16 ***
## A2           0.71528    0.02402  29.779   <2e-16 ***
## A3          -1.56445    0.09761 -16.028   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.508 on 996 degrees of freedom
## Multiple R-squared:  0.5433, Adjusted R-squared:  0.542
## F-statistic: 391.7 on 3 and 996 DF,  p-value: < 2.2e-16
```

```
summary(het, robust=F) #original model without robust se
```

```
##
## Call:
## lm(formula = A4 ~ A1 + A2 + A3, x = T, y = T)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.3920 -1.0108  0.0359  0.9829  6.5782
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.52279    0.30156  14.998   <2e-16 ***
## A1           0.47747    0.04847   9.851   <2e-16 ***
## A2           0.71528    0.02397  29.841   <2e-16 ***
## A3          -1.56445    0.09957 -15.711   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.508 on 996 degrees of freedom
## Multiple R-squared:  0.5433, Adjusted R-squared:  0.542
## F-statistic:   395 on 3 and 996 DF,  p-value: < 2.2e-16
```

Alternatively, and perhaps more easily, we can do the following:

```
#alternative way to carry out robust se estimation in R (see http://data.princeton.edu/wws509/r/robust.
library(lmtest)
```

```
## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

##
## Attaching package: 'lmtest'

## The following object is masked from 'package:RCurl':
##
##     reset
```

```
library(sandwich)
coeftest(het, vcov = vcovHC(het, type="HC1"))
```

```
##
## t test of coefficients:
##
##              Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)  4.522789   0.285854  15.8220 < 2.2e-16 ***
## A1           0.477468   0.051806   9.2165 < 2.2e-16 ***
## A2           0.715284   0.024020  29.7788 < 2.2e-16 ***
## A3          -1.564447   0.097607 -16.0281 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice that when we estimate a model with robust standard erros our estimates are unchanged, but our standard errors are different, and therefore our t-tests provide different hypotehsis test results. These results are robust to the non-constant error variance in the model.