IFRS9 FLI - OLS, number of observations, number of independent variables and violation of the normality assumption

How bad can it gets?

Andrija Djurovic*

*www.linkedin.com/in/andrija-djurovic

The predominant statistical approach in IFRS9 FLI modeling involves using OLS regression, with the p-value as one of the criteria for selecting the final model. Even though one of the assumptions underlying OLS regression is the normality of residuals, this assumption is often overlooked in practice. The primary reason for this neglect stems from the need for more consensus among practitioners regarding the significance of this assumption in influencing regression estimates and inference.

Even in literature, opinions vary regarding the importance of the normality assumption. This spectrum of views ranges from suggestions to completely omit testing to the more commonly held belief that violating this assumption impacts the inference drawn from OLS regression, including test statistics and p-values.

In the context of IFRS9 FLI modeling, this assumption, coupled with a relatively low ratio between the number of observations and the number of independent variables, can pose specific challenges. Consequently, this document aims to present the framework for exploring the correlation between the number of observations per independent variable, simulated deviations from normality, and type I error. It is important to emphasize that the provided example should not be viewed as a broad generalization of deviations from normality. Instead, it can serve as a basis for tailoring the process to meet specific requirements and address unique use cases.

Let's begin by defining the assumptions and implementing the simulation process in R.

```
#simulation parameters
sim.rep <- 100                #simulation replicates
iv.n <- c(1, 2, 3, 5, 9, 15) #number of independent variables
n <- 45                       #number of observations
iv.l <- length(iv.n)          #num. of options for indep. vars
alpha <- 0.05                 #significance level
B <- 100                      #number of mc simulations
```

Based on the inputs provided above, we can determine the average number of observations for each set of independent variables.

```
#average number of observations per independent variables
n / iv.n
```

```
## [1] 45.0 22.5 15.0  9.0  5.0  3.0
```

By incorporating assumptions concerning the distribution of variables (precisely, a t-distribution with 2 degrees of freedom) and the anticipated zero value for the estimates of independent variables in relation to the target, we can model the impact on type I error, as demonstrated in the subsequent code snippet.

```r
#-------------------------run the simulations-------------------------#
#empty list to store the results
res <- vector("list", sim.rep)

for   (z in 1:sim.rep) {

       res.z <- vector("list", iv.l)

       for   (i in 1:iv.l) {
              set.seed(i + z)
              iv.i <- iv.n[i]

              #simulate the independent variables (t-distribution, df = 2)
              iv <- replicate(n = iv.i,
                              expr = rt(n = n, df = 2)
                              )
              iv <- data.frame(iv)
              names(iv) <- paste0("x", 1:ncol(iv))

              #store the xs into data frame
              db <- cbind.data.frame(iv)

              #regression formula
              frm <- paste0("y ~ ", paste(names(iv), collapse = " + "))

              #-------mc simulations-------#
              #empty data frame to store the results
              res.i <- data.frame(matrix(data = NA,
                                         ncol = 2 * iv.i + 2,
                                         nrow = B)
                                  )
              names(res.i) <- c("simulation",
                                names(db),
                                paste0(names(db), ".pval"),
                                "f.test"
                                )

              for   (j in 1:B) {
                     set.seed(i + j*10 + z)
```

```
                    #simulate the target (t-distribution, df = 2)
                    db$y <-  rt(n = n, df = 2)
                    #run the regression
                    lr.j  <- lm(formula = frm,
                                data = db)
                    #regression summary
                    lr.j.s <- summary(lr.j)
                    #f-test
                    fs.j <- lr.j.s$fstatistic
                    fs.p <- pf(q = fs.j[1],
                               df1 = fs.j[2],
                               df2 = fs.j[3],
                               lower.tail = FALSE)
                    #summarize the results
                    res.j <- c(j,                              #simulation id
                               lr.j.s$coefficients[-1, 1], #estimates
                               lr.j.s$coefficients[-1, 4], #p-value of estimates
                               fs.p                            #f-test p-value
                               )
                    res.i[j, ] <- res.j
                    }

            #add the number of indep. var
            res.i <- cbind.data.frame(iv.n = iv.i,
                                      res.i)
            #store the results
            res.z[[i]] <- res.i
            }

      #type I error of the regression
      res[[z]] <- sapply(X = res.z,
                         FUN = function(x) mean(x[, "f.test"] < alpha))
      }
```

In conclusion, considering the abovementioned assumptions, we can summarize the outcomes and examine the pattern of type I error across various sets of independent variables in the model.

```
#summarise the results
res <- do.call("rbind", res)
colnames(res) <- iv.n
```

```r
#calculate the average type I error
colMeans(res)
```

```
##      1      2      3      5      9     15
## 0.0540 0.0633 0.0729 0.0809 0.0883 0.0957
```

As observed in the results, a decrease in the number of observations per independent variable corresponds to an increase in type I error. Moreover, within the simulated distribution, there is a noticeable rise in type I error beyond the expected significance level. This indicates a potential issue with the inference from regression in this specific configuration.

Let's now repeat the same exercises in `Python`.

```python
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

# Simulation parameters
sim_rep = 100                 #simulation replicates
iv_n = [1, 2, 3, 5, 9, 15]    #number of independent variables
n = 45                        #number of observations
iv_l = len(iv_n)              #num. of options for indep. vars
alpha = 0.05                  #significance level
B = 100                       #number of mc simulations

#average number of observations per independent variables
np.array([n / iv for iv in iv_n])
```

```
## array([45. , 22.5, 15. ,  9. ,  5. ,  3. ])
```

```python
#------------------------run the simulations-----------------------#
#empty list to store the results
res = [None]*sim_rep

for z in range(sim_rep):

    res_z = [None]*iv_l

    for i in range(iv_l):
        np.random.seed((i + 1) + (z + 1))
        iv_i = iv_n[i]
```

```
        #simulate independent variables (t-distribution, df = 2)
        iv = np.random.standard_t(df = 2, size = (n, iv_i))

        #store the xs into data frame
        db = pd.DataFrame(iv, columns = [f"x{i}" for i in range(0, iv_i)])

        #regression formula
        formula = f'y ~ {" + ".join(db.columns)}'

        #-------mc simulations-------#
        #empty data frame to store the results
        res_i = pd.DataFrame(np.nan, index = range(0, B),
                             columns = ["simulation"] +
                                       list(db.columns) +
                                       [f"{col}.pval" for col in db.columns] +
                                       ["f.test"])

        for j in range(0, B):
            np.random.seed((i + 1) + (j + 1) * 10 + (z + 1))
            #simulate the target (t-distribution, df = 2)
            db["y"] = np.random.standard_t(df = 2, size = n)
            #run the regression
            lr_j = smf.ols(formula = formula, data = db).fit()
            #f-test
            fs_j = lr_j.fvalue
            fs_p = lr_j.f_pvalue
            #summarize the results
            res_j = [j] + list(lr_j.params[1:]) + list(lr_j.pvalues[1:]) + [fs_p]
            res_i.loc[j] = res_j

        #add number of indep. var
        res_i.insert(0, "iv_n", iv_i)
        #store the results
        res_z[i] = res_i

    #type I error of the regression
    res_z_mean = [np.mean(x["f.test"] < alpha) for x in res_z]
    res[z] = res_z_mean
```

```python
#summarise the results
res = pd.DataFrame(res, columns = iv_n)

#calculate the average type I error
res.mean()
```

```
## 1     0.0558
## 2     0.0652
## 3     0.0734
## 5     0.0858
## 9     0.0908
## 15    0.0922
## dtype: float64
```

In addition to the provided setup, readers are encouraged to explore variations such as different numbers of observations, simulations, and diverse distributions for independent variables, target, and residuals. They are also encouraged to examine individual estimates' confidence interval coverage and customize the examples for power calculation.