Implementation of the Repeated Measures ANOVA

Statistical Programming Languages

Vorname, Name
Matrikelnummer:

January 31, 2017

Contents

1 Theorie and Motivation

1.1 Make Anova great again

The main goal of this project is to create an R-package, which bundles a number of functionality around the grand topic of repeated measures Anova. Some of the implemented functions are not yet avaiable in R, although the repeated measures anova itself is avaiable. But a functionality to curb with breach of sphericity and correction of confidence intervals are not avaiable.

1.2 The Ringelmann-Effect

The Ringelmann-effect describes the decreasing individual work performance when work is done in a group. It is therefore often cited in the context of social loafing. The effect was first examined and described by the french agricultural professor Maximillian Ringelmann in his paper from 1913 (**?**).

2 Simulating and Preparing the Data

Short theoretical introduction

2.1 Simulation

In order to demonstrate and evaluate the functions implemented in the package, we have developed a function to simulate data. It can then be used to test code or for educational purposes. First we will present the functionalities and the implementation of the simulation function.

```
# Run the data simulation
rma_data = sim_rma_data(n = 1000, k = 4, means = NULL, poly_order = 5,
    noice_sd = c(10, 20, 30, 20), between_subject_sd = 40, NAs = 0)
```

The data can be simulated by running the function shown above, which then returns a dataframe object. The function allows for a number of arguments to be passed. This makes it possible to simulate data with numerous features such as orthogonal polynomial contrasts and breach of sphericity.

The first arguments n is obligatory and defines the number of observation. The argument k sets the number of factors to be simulated. The output of the function will therefore be an matrix of the size n x (k + 1). The first column contains the subject ids to identify each simulated observation and the following columns contain the simulated realization for each factor.

```
    if (!is.null(means)) {
    con_means = means

    # Check if length of mean vector corresponds to k
```

```
if (length(means) != k) {
k = length(means)
print("Number of factors (k) was changed, because the length of means
    vector and argument k do not correspond.")
}
} else {

# Simulate conditional means
if (is.null(poly_order)) {
con_means = runif(k, min = 100, max = 300)
} else {

# Generate polinomial conditional means
factors = runif((poly_order + 1), min = 0, max = 1)
x = order(runif(k, min = 100, max = 300), decreasing = FALSE)
con_means = matrix(factors[1], nrow = k)

for (p in (2:(poly_order + 1))) {
con_means = con_means + factors[p] * x^p
}
}
}
```

The first step, when simulating the data is to simulate the means of each factors. Thereby each factor columns is filled with the mean for the corresponding factor. This results in all observations having the same value for each factor. Additionally by passing an integer not larger than k to the argument poly_order, the means will be simulated so that they create a polynomial contrast in the data. Instead of passing an integer k as, a vector with k means kann be passed to the argument means, this makes it possible to create data with a custom relationship between the factors.

The next step is to simulate the between subject variance and add the subject means, so variation is created between each entity. This is accomplished by adding a matrix of the size n x k with normal distributed random numbers to the factor realizations. The standard deviation of the normal distribution from which the entity deviations are drawn can be controlled with the argument between_subject_sd.

The last step of the simulation is to add noise to the simulated data. The size of the noise can be controlled by passing an integer to the argument noise. It is also possible to pass a vector of the size k, which makes it possible to simulate data which does not live up to the sphericity assumption. By simulating different standard deviations for each factor, the factor differences also have different variance.

2.2 Listwise Deletion

When computing a Repeated Measures Anova, the way to deal with missing values is listwise deletion. As we measure the change of our subjects over factor levels, a missing value in one factor level leads to a dropout of the whole observation from our analysis.

Since we use simulated data, we could easily avoid having missing values. However, for demonstration as well as for applicability to other data, we integrated a function for listwise deletion. In the above mentioned data simulation function rma_data, one specifies the number of NAs, which then randomly replace values of factor levels in the simulated data. Subsequently, the listwise deletion function checks for existing NAs and drops out the corresponding subject(s), while displaying a message informing about the id(s) of the subject(s) that has/have been deleted.

3 Estimating the Repeated Measures Anova

3.1 Estimation

4 SSE Reduction with the Repeated Measures Anova (currently in work...(Freddi))

4.1 Estimating Anova

4.2 Comparing the Error Terms

5 Effect Size

In addition to the inferential information derived from the omnibus F-test, it is often useful to provide a measurement of effect size as well. In an ANOVA design a widely used effect size measurement is eta squared (η^2) as well as partial eta squared (η_p^2). So this function computes eta squared as well as partial eta squared.

```
rma_eta(rma_data, id = 1, append = FALSE)
```

As input parameter it needs a repeated measures data file which meets the above described requirements. It also contains the $id = 1$ argument. If append = TRUE both effect size measures are added to the ANOVA-table.

Eta squared is basically the ANOVA equivalent to the determination coefficient [2] and therefore interpretable as the proportion of variance in the depended variable which can be explained by the factor and for a one way repeated measures ANOVA it is computed as

$$\frac{\eta^2 = SS_{Factor}}{SS_{Error} + SS_{Subject} + SS_{Factor}} \tag{1}$$

```
anova_table = rma(rma_data, id = id)[[1]]

SS_Factor = anova_table[2, 2] SS_Error = anova_table[4, 2] SS_K_Total =
    anova_table[6, 2]

# Compute standard eta^2
eta_sq = SS_Factor/SS_K_Total
```

In most cases where a repeated measures ANOVA is preferred to a ANOVA without repeated measures the effect of interest (factor effect) and therefore its sum of squares tends to be small in comparison to the between subject variance. If the aim is an effect size measure which only takes within subject variance into account, the partial eta square can be computed in the one way repeated measures ANOVA as

$$\frac{\eta_p^2 = SS_{Factor}}{SS_{Error} + SS_{Factor}} \tag{2}$$

```
# Compute partial eta^2
eta_partial = SS_Factor/(SS_Factor + SS_Error)
```

The results of the computational steps described above are summarized in a table, which is later returned by the function.

```
effect_size_table = data.frame(check.names = FALSE, Source = "Factor", `eta
    squared` = eta_sq, `partial eta squared` = eta_partial)
    rownames(effect_size_table) = NULL
```

However for a repeaded measures ANOVA the so called generalized eta squared is recommended by (Bakeman; 2005)(see also Olejnik and Algina (2003)), since it allows a comparison of effect sizes between different designs. Luckily in the one way ANOVA the generalized eta squared is equal to eta squared. So the function will eventually return the tables with both effect size measures. If the function argument append = TRUE the ANOVA table with both effect size measures attached will be provided as well.

```
if (append == TRUE) {
 return(anova_table) } else {
 return(effect_size_table)
}
```

For the analysis of the Ringelmann-data η^2 =???, which indicated that percent of the variation in the depended variable can be explained by the factor levels. If only the within subject variance is considered as explainable variation in the depended variable $eta_p^2 =$, which indicates that percent of the within subject variation in the depended variable can be explained by the factor levels. Necessarily $\eta^2 \leq \eta^2$ will always hold.

6 Confidence Intervals(CI)

In most cases it is desirable to present the results of a repeated measures ANOVA visually, by plotting the factor level means of the dependent variable. Additionally it is particular instructive (and therefore often required e.g. by many scientific journals) to present some depiction of the inferential accuracy. In ANOVA designs without repeated measures factors this is often accomplished by the inclusion of error bars which display the associated confidence intervals. These confidence intervals (CI) are computed by

$$\bar{y}_j \pm t_{n_j-1,1-\alpha/2} x SE(\bar{y}_j) \tag{3}$$

where

$$\bar{y}_j = \frac{1}{n_j} \sum_{i=1}^{n_j} y_j i \tag{4}$$

is the factor level mean and

$$SE(\bar{y}_j) = \frac{\frac{1}{n-1} \sum_{i=1}^{n_j} (y_{ji} - \bar{y}_j)^2}{\sqrt{n}} \tag{5}$$

is its standard error. This can be computed with R in the following way.

```
# Standard errors of the conditional means
SE = tapply(rma_data_long$value, rma_data_long$condition, sd)/sqrt(n)

# CI for ANOVA without repeated measures
CIdist_unadj = abs(qt((1 - C_level)/2, (n - 1))) * SE

# Compute upper and lower bound
up_unadj = MeFlm$Flm + CIdist_unadj low_unadj = MeFlm$Flm - CIdist_unadj
```

However in ANOVA designs which incorporate a repeated measures factor, this method is no longer applicable. ThatâĂŹs because this computational method includes the variance between entities into the standard error of the factor level mean, which the F-test of the ANOVA with a repeated measures factor does not.

O'Brien and Cousineau (2014) suggested a method for the computation of adjusted confidence

Intervals based on methods described by Loftus and Masson (1994), Cousineau (2005) and Morey (2008) which is conducted by the following function.

```
rma_ci(rma_data, C_level = 0.95, id = 1, print_plot = TRUE)
```

As input parameter it needs a repeated measures data file which meets the above described requirements. It also contains the $id = 1$ and $print\_plot = TRUE$ arguments. By the argument $C_level = 0.95$ it is also possible to select the desired alpha level of the confidence interval.

For the method of O'Brien und Cousineau (2014) the corresponding individual subject means are subtracted from every value of the depended variable and afterwards the grand mean is added.

$$y_{ji}^{'} = y_{yi} - \bar{y}_i + \bar{y} \tag{6}$$

Thereby all between subject variation is removed from the resulting values. Afterwards a correction factor has to be applied (see Morey (2008)) by

$$\frac{1}{k-1} x(y_{ji}^{\prime} - \bar{y}_j) + \bar{y}_j \tag{7}$$

```
# Correction factor etablished by Morey (2008)
cf = sqrt(k/(k - 1))

AdjVal = data.frame(Adj = (cf * ((rma_data_long$value - EEmlong$Em + Gm) -
    MeFlmlong$Flm)) + MeFlmlong$Flm)
rma_data_long_adj = cbind.data.frame(rma_data_long, AdjVal)
```

The adjusted values can be used to compute confidence Intervals in the same way as in the case of an ANOVA without repeated Measures.

```
# Standard errors of the conditional means adjusted with the method of
    O'Brien and Cousineau (2014)
SE_adj = (tapply(rma_data_long_adj$Adj, rma_data_long_adj$condition,
    sd)/sqrt(n))
CIdist_adj = abs(qt((1 - C_level)/2, (n - 1))) * SE_adj

# Compute upper and lower bound
up_adj = MeFlm$Flm + CIdist_adj
low_adj = MeFlm$Flm - CIdist_adj
```

The adjusted and unadjusted confidence intervals are summarized in a table, which is later returned by the function.

```
lu_adj_CI = cbind(low_adj, up_adj)
lu_unadj_CI = cbind(low_unadj, up_unadj)
colnames(lu_adj_CI) = colnames(lu_unadj_CI) = c("Lower bound", "Upper
    bound")
```

Error bar plots which show the adjusted and unadjusted confidence intervals are constructed using ggplot.

```
# create two vectors for lower ci values and upper ci values respectively
lower = c(low_adj, low_unadj)
upper = c(up_adj, up_unadj)

# create vector that is used for facetting i.e. for assigning the correct
    values to each plot
plot_nr = rep(c("Adjusted CI", "Unadjusted CI"), each = k)

# create data frame for ggplot: comparison of ci
ci_plot_data = data.frame(plot_nr, rbind(MeFlm, MeFlm), lower, upper)

# create plot with adjusted ci values
ci_plot = ggplot(data = ci_plot_data, aes(Me, Flm)) + geom_point(size = 2) +
geom_errorbar(aes(ymax = upper, ymin = lower), width = 0.1) +
    facet_grid(~plot_nr) +
labs(x = "Condition", y = "Value", title =
"Comparison of adjusted and unadjusted (standard) confidence intervals") +
    theme_bw() +
theme(panel.grid.major = element_blank(), panel.grid.minor =
    element_blank(), legend.key = element_rect(colour = "black"),
    plot.title = element_text(face="bold", hjust = .5))
```

The function will eventually create this plot and return the table with the adjusted and unadjusted confidence intervals.

```
if (print_plot == TRUE){ print(ci_plot)} return(list(confidence_intervals =
    data.frame(adjusted_CI = lu_adj_CI, unadjusted_CI = lu_unadj_CI),
    ci_plot = ci_plot))
```

As expected the application of this function on the Ringelmann-data revealed that the adjusted confidence regions become narrower then there unadjusted counterpart.

An important remark is that these confidence intervals should not be used to assess whether the sphericity assumption is violated as pointed out by Franz and Loftus (2012). The issues regarding sphericity will be discussed later on in greater detail.

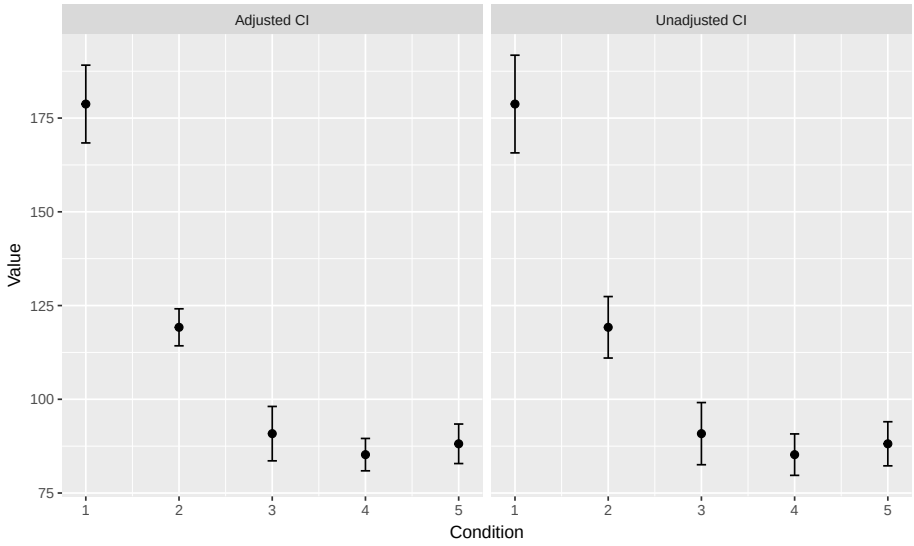

Abb. 1: Adjusted and Unadjusted Confidence Interval

7 Sphericity

One of the requirements for a repeated measures ANOVA is known to be sphericity. This assumption is met when the variances of the differences between related combinations of all factor levels are equal. Sphericity is related to homogeneity of variances in an ANOVA without repeated measures. The violation of sphericity causes the test to become too liberal. Therefore, determining whether sphericity has been violated is necessary if the omnibus F-test of a repeated measures ANOVA shall be interpreted. Though violation of sphericity is problematic, it is still possible to conduct and interpret the Omnibus F-test if some adjustments for the violation of sphericity are incorporated. This is achieved by estimating the degree to which sphericity has been violated. The corresponding estimator is commonly denoted by epsilon ($\epsilon \in [1, 0]$) where a value of 1 means the data is absolute spherical. This measure epsilon can then be used as a correction factor to the degrees of freedom of the F-distribution, which leads (in case of $\epsilon < 1$) to a decrease in the degrees of freedom and therefore to a more conservative F-test. Of course it will be only very rarely the case that sphericity is met exactly by the empirical data, so at first is it is necessary to tests, whether it can be assumed that sphericity is theoretically met or not. A popular Test to accomplish this is called Mauchly's sphericity test (Mauchly, 1940). It tests the ðİŘż0 that sphericity is met. Therefore a violation of sphericity shall be assumed whenever the Mauchly's sphericity test yields a significant results whereupon the ðİŘż0 is rejected. So if a repeated measures ANOVA is indicated it must be tested if the assumption of sphericity is met. If it is not, an appropriate adjustment of the omnibus F-Test has to be performed. This is implemented by the following function.

```
rma_spheri(rma_data, id = 1, append = FALSE)
```

As input parameter it needs a repeated measures data file which meets the above described re-

quirements. It also contains the $id = 1$ argument. If $append = TRUE$ the function chooses the appropriate correction (see Girden, 1992) for the omnibus F-test if sphericity is violated and appends the thereby adjusted p-value to the ANOVA-table. Obviously it is nonsensical to test sphericity, if there are only two factor levels ($k = 2$). So the function will abort in this case and provide a notification.

```
# Check whether a test for sphericity is needed if (k = 2) { stop("Note
    that there can't be a violation of sphericity since the factor of the
    one-way anova has only two factor levels")
 }
```

To conduct the Mauchly's sphericity test the first step is to construct a Helmert matrix. The construction of the first row is omitted since it is not required for further use in this procedure.

```
helmert = function(k, df) { H = matrix(0, k, k) diag(H) = (0:df) *
    (-((0:df) * ((0:df) + 1))^(-0.5)) for (i in 2:k) { H[i, 1:(i - 1)] =
    ((i - 1) * (i))^(-0.5) } # H[1,] = 1/sqrt(k) return(H) }
# The first row of the helmert matrix is not required for further use in
    this procedure C = helmert(k, df)[-1, ]
```

Thereby df denotes the effect degrees of freedom which equal $k-1$. Then the test statistic (Mauchly's W) for the Mauchly's sphericity test is computed by

$$w = \frac{|CVar(Y)C^T|}{(tr(CVar(Y)C^T)k-1)^{k-1}} \tag{8}$$

where ðİŘű is the Helmert matrix and ðİŚĽðİŚŐðİŚ§(ðİŚŇ) is the depended data covariance matrix.

```
# Empirical covariance matrix covariance_matix = cov(dependent_variable)
w = det(C %*% covariance_matix %*% t(C))/ ((sum(diag(C %*% covariance_matix
    %*% t(C)))/df)^(df))
```

The transformation

$$-(n-1)x(1-\frac{2x(k-1)^2+k+1}{6xk-1xn-1} \tag{9}$$

of Mauuchly's W in χ^2 distributed with

$$\frac{kx(k-1)}{2}-1 \tag{10}$$

degrrees of freedom. So a p-value for the test can be computed.

```
# Computing the degrees of freedom for the chi-square value df_w = ((k *
```

```
    df)/2) âĂŞ 1
# Computing the chi-square value f = 1 - ((2 * (df^2) + df + 2)/(6 * df *
    (n - 1))) chi_sq_w = -(n - 1) * f * log(w)
# Computing the corresponding p-value p_w = 1 - pchisq(chi_sq_w, df_w)
```

The results of the computational steps described above are summarized in a table, which is later returned by the function.

```
mauchly_table = data.frame(check.names = FALSE, Source = "Factor",
    ‘Mauchly’s W‘ = w, ‘Chi square‘ = chi_sq_w, df = df_w, p = p_w)
    rownames(mauchly_table) = NULL
```

The next step is to compute epsilon.

```
# Lower-Bound correction (Greenhouse & Geisser, 1959) epsilon_lb = 1/df
# Box correction (Geisser & Greenhouse, 1958) epsilon_gg = (sum(diag(C %*%
    covariance_matix %*% t(C)))^2)/ (df * sum(diag(t((C %*%
    covariance_matix %*% t(C))) %*% (C %*% covariance_matix %*% t(C)))))
# Huynh-Feldt correction (Huynh & Feldt, 1976) epsilon_hf = min((n * df *
    epsilon_gg - 2)/(df * (n - 1) - (df^2 * epsilon_gg)), 1)
```

There are three commonly used estimators for epsilon (see Rutherford (2001)). The Huynh-Feldt epsilon (Huynh and Feldt; 1976) on the one hand is a fairly liberal estimator, i.e. it tends to underestimate the degree of violation. The Box epsilon (Box et al.; 1954; Geisser et al.; 1958) on the other hand is a more conservative estimator and therefore tends to overestimates the degree of violation. The Greenhouse-Geisser or lower bound epsilon (Greenhouse and Geisser, 1959) lastly is the most conservative estimation possible under the given conditions, i.e. it assumes the worst-case scenario of violation, thus it is actually not sensitive for variation in the degree by which sphericity is violated. Each of these can be used as a correction factor to adjust the degrees of freedom and therefore the resulting p-value of the omnibus F-test (note that the empirical F value stays the same since the correction factors cancel out, but the theoretical F distribution of the tests statistic under H_0 changes).

```
anova_table = rma(rma_data)[[1]]
corrected_factor_df = anova_table[2, 3] * epsilon_lb corrected_error_df =
    anova_table[2, 3] * epsilon_lb p_factor_lb = 1 - pf(anova_table[2, 5],
    corrected_factor_df, corrected_error_df)
corrected_factor_df = anova_table[2, 3] * epsilon_gg corrected_error_df =
    anova_table[2, 3] * epsilon_gg p_factor_gg = 1 - pf(anova_table[2, 5],
    corrected_factor_df, corrected_error_df)
corrected_factor_df = anova_table[2, 3] * epsilon_hf corrected_error_df =
    anova_table[2, 3] * epsilon_hf p_factor_hf = 1 - pf(anova_table[2, 5],
    corrected_factor_df, corrected_error_df)
```

The results of the computational steps described above are summarized in a table, which is later returned by the function.

```
epsilon_table = data.frame(check.names = FALSE, Source = c("Epsilon",
    "Adjusted p-Value"), `Lower-Bound correction (Greenhouse & Geisser,
    1959)` = c(epsilon_lb, p_factor_lb), `Box correction (Geisser &
    Greenhouse, 1958)` = c(epsilon_gg, p_factor_gg), `Huynh-Feldt
    correction (Huynh & Feldt, 1976)` = c(epsilon_hf, p_factor_hf))
    rownames(epsilon_table) = NULL
```

Greenhouse and Geisser (1959) recommended in case of a violation of sphericity (indicated by a significant Mauchly's sphericity test) to use the lower bound correction first. If the results become insignificant, the Box epsilon is used to estimate the degree of violation. If the degree of violation is high (i.e. $\epsilon < 0.75$) the Box correction is preferable, if the degree of violation is low (i.e. $\epsilon < 0.75$) the Huynh-Feldt epsilon shall be used to correct the degrees of freedom (Girden, 1992).

```
if (p_w < 0.05) { if (p_factor_lb < 0.05) { anova_table[, "Recommended
    Lower-Bound corrected p-Value (Greenhouse & Geisser, 1959))"] = c(NA,
    p_factor_lb, NA, NA, NA, NA) } else { if (epsilon_gg < 0.75) {
    anova_table[, "Recommended Box corrected p-Value (Geisser & Greenhouse,
    1958)"] = c(NA, p_factor_gg, NA, NA, NA, NA) } else { anova_table[,
    "Recommended Huynh-Feldt corrected p-Value (Huynh & Feldt, 1976)"] =
    c(NA, p_factor_hf, NA, NA, NA, NA) } } }
```

The function will eventually return both tables with the results of the Mauchly's sphericity test as well as the correction factors with their associated adjusted p-values. If the function argument $append = TRUE$ the ANOVA-table with the recommended p-value is provided as well.

```
if (append == TRUE) { return(list(mauchly_table = mauchly_table,
    correction_factors_epsilon_table = epsilon_table, corrected_anova_table
    = anova_table)) } else { return(list(mauchly_test_table =
    mauchly_table, correction_factors_epsilon_table = epsilon_table)) }
```

The analysis of the Ringelmann-data reveals that sphericity is violated (Ł $=$,2 $=$, $<$). The recommended correction is adjusting the degreees of freedom with the Box-epsilon ($\epsilon =$). The thereby for the violation in sphericity corrected omnibus F-test is still significant ((,) $=$, $<$) so it can be assumed that there is indeed an effect of the group size on the individual work performace.

7.1 Test for Sphericity

Hier dann noch mal ein anderes Zitat (**?**).

7.2 Adjustment for Sphericity

8 Orthogonal polynomial contrasts

After the conduction of a repeated measures ANOVA we obtain an omnibus F-Value and thus a general idea, whether the factor indeed affects the depended variable. However, it remains unclear

what this dependency looks like in particular. To evaluate the impact of the different factor levels on the depended variable in further detail, one might conduct post hock t-tests between each pair of factor levels or even test specific a priori hypothesis by the implementation of complex contrast analyses (see e.g. Howell, 2010; Steavens, 2009). Since the factor levels of a repeated measures ANOVA most commonly possess an interval scale of measurement (or are at least believed to be approximately equidistant ordinal scaled), there is an additional method witch should be considered. This method is known as orthogonal polynomial contrasts or trend analysis. As the name implies, this method detects orthogonal polynomial trend components in the Effects of the factor. If there are k factor levels it decomposes the omnibus factor effect into a full set of $k-1$ orthogonal contrasts. These contrast all detect a specific polynomial trend of all orders from 1 to $k-1$ and test if these trends contribute significantly to the overall effect of the factor. This allows testing specific hypothesis regarding separate trends in the depended variable due to manipulations of the factor level.

```
rma_opc(rma_data, id = 1, maxpoly = NA, print_plot = TRUE)
```

The above function conducts this type of analysis. As input parameter it needs a repeated measures data file which meets the above described requirements. It also contains the $id = 1$ and $print_plot = TRUE$ Arguments. Since at is always possible to describe the data perfect with a polynom of the order $k-1$ and hence this is the maximal order of a polynomial trend used for this analysis we define the maxpoly variable. Certainly it is conceivable that not all $k-1$ orthogonal polynomial trends are of interest, so the function contains the additional argument $maxpoly = NA$. If a Value smaller than $k-1$ is inserted the function only computes and tests the contrasts up to the polynomial trend of the same order.

```
# maximal polynomial degree for orthogonal polynomials if((maxpoly > k - 1)
    | (is.na(maxpoly))){ maxpoly = k - 1}
```

At first it is necessary to calculate for each of the requested orthogonal trend components a so called linear contrast. A linear contrast is basically the sum of the weighted dependent variable values of each individual factor level. To obtain linear contrasts which represent all orthogonal components of $k-1$ possible polynomial trends, the weights are generated by using contr.poly().

```
# Defining Contrast weights for orthogonal polynomial contrasts
    contrast_weights = (t(contr.poly(k)))[1:maxpoly, ]
```

It should be noted, that it needs maxpoly times k contrast weights in total for this type of analysis. In the next step the weighting factors are applied and the linear contrasts for each specific trend are computed individually for each subject

$$L_S = sum_{i=1}^{k} c_i Y_{is}. \tag{11}$$

```
# Applying formula for linear contrasts weighted_dependend_variables =
    dependent_variable[rep(1:n, each = maxpoly), ] *
    (contrast_weights)[rep(1:maxpoly, n), ] linear_subject_contrasts =
    matrix(rowSums(weighted_dependend_variables), byrow = TRUE, ncol =
    maxpoly)
```

Following this step there are linear contrasts for each individual entity in the study, so it is necessary to construct an aggregated score to draw further conclusions. This is accomplished by taking the mean of al linear contrasts for one specific trend component to obtain the $contrast\_estimator$

$$\bar{L} = sum_{i=1}^{n} L_k. \tag{12}$$

```
# Computing contrast estimators for each orthogonal polynomial contrast as
    well as standard errors for thees estimators contrast_estimator =
    colMeans(linear_subject_contrasts) contrast_se =
    sqrt(apply(linear_subject_contrasts,2,var)) / sqrt(n)
```

The standard error of this contrast estimator $contrast\_se$ is computed as well for inference reasons. This allows to test whether this contrast score deviates from 0 significantly, which is accomplished via a t-test.

```
# Computing t-values for each contrast contrast_t_values =
    contrast_estimator / contrast_se #contrast_F_values = contrast_t_values^2

# Computing the corresponding p-values contrast_p_values = 1 -
    pt(abs(contrast_t_values), n-1)
```

If the individual trend components contribute significantly to the factor effect, it might be of interests how much a certain trend contributed to the effect of the factor. Therefore the sum of squares component which is explained by the factor is decomposed into maxpoly trend component sums of squares by computing the individual trend sum of squares

$$SS_L = \frac{nx\bar{L}^2}{\sum_{i=1}^{c_i^2}} \tag{13}$$

This is possible due to the fact, that the contrasts are orthogonal to each other, so they do not have redundant information. The proportion of a specific trend sum of square to the factor sum of square marks the percentage in contribution of this particular trend component to the whole factor effect

$$\sum_{j=1}^{k-1} SS_{L(orth.)_j} = SS_{Factor} \tag{14}$$

```
# Computing sums of squares for each contrast contrast_ss = n *
    contrast_estimator^2 / rowSums(contrast_weights^2)

# Computing amount of the variance in the dependent variable explained by
    the factor # which in turn can be explained by a cerain orthogonal
    polynomial trend (ss_trend / ss_factor) proportional_trend_contribution
    = contrast_ss / rep(sum(rep((Flm - Gm)^2, each = n)), maxpoly)
```

The results of the computational steps described above are summarized in a table, which is later returned by the function.

```
# define source variable source = rownames(contrast_weights) contrast_table
    = data.frame(check.names = FALSE, "Source" = source, "Sum of squares" =
    contrast_ss, "Proportional contribution to the factor effect" =
    proportional_trend_contribution, "Contrast estimator" =
    contrast_estimator, "Standard error" = contrast_se, "Degrees of
    freedom" = rep((n - 1), maxpoly), "t-value" = contrast_t_values,
    "p-value" = contrast_p_values) rownames(contrast_table) = NULL
```

To display the polynomial trends in the data, polynomial regressions were used. The results actually do not represent orthogonal trends but rather each trend line of a certain polynomial degree also includes all lower order polynomial trend contributions.

To store the orthogonal polynomial regression coefficients (OPRC) an empty matrix is set up with k rows and the number of columns equal to the highest order polynomial, previously specified as maxpoly.

```
poly_coef = data.frame(matrix(0, ncol = maxpoly, nrow = k))
```

The choice of these dimensions becomes clear when the following code is regarded:

```
The choice of these dimensions becomes clear when the following code is
    regarded:
for (i in 1:maxpoly) {
pfv = paste("poly_fit_", i, sep = "")
poly = assign(pfv, lm(rma_data_long$value ~ poly(rma_data_long$condition,
    degree = i, raw = TRUE)))
poly_coef[, i][1:(i + 1)] = poly$coef
}
```

In each cycle of the for loop, that runs from 1 to the maximal polynomial order(maxpoly), the following steps are executed: First, orthogonal polynomial regressions are estimated for the observed values (pulled weights) in long format suing the poly() function within lm(). The degree of the highest polynomial is specified by the index i. The model always contains an intercept as well as coefficients for the data to the i-th polynomial. The result, which is an object of type lm,

is stored in an object called poly which is overwritten in each cycle. In the second step, the coefficients are extracted from poly and stored in the previously prepared matrix (poly_coef) such that the coefficients are assigned to the i-th column. The subsetting is cruicial here to ensure that the dimensions in the assignment coincide. For example, in the second cycle, the second column of the data frame is chosen. This column is then filled with the intercept coefficient, the first order polynomial coefficient and the second order polynomial coefficient. Since in this cycle there is no higher order, the last element in the data frame remains equal to zero. The resulting data frame looks as follows:

19.668	4.713	-60.057
-4.651	10.305	113.320
0.000	-2.991	-49.255
0.000	0.000	6.169

Here, one can see that with each cycle of the loop the number of nonzero coefficients increases by one. So the first row stores the coefficients for the intercept, the second row the coefficients for the first order coefficient. The last row always contains the highest order coefficient (here: 3). The columns represent the resulting regression equations and can be seen as follows:

$$C1 : b0 + b1XC2 : b0 + b1X + b2X^2C3 : b0 + b1X + b2X^2 + b3X^3 \tag{15}$$

First, a vector x is defined as a sequence from one to k, the number of factor levels in steps of k/1000. Next, the outer() function is used to create a matrix that takes x to the power of 0 to maxpoly. The result is stored in the object poly_values.

```
x = seq(1, k, length.out = 1000)
poly_values = outer(x, 0:(k - 1), `^`)
```

The first six rows of the resulting matrix look as following:

table

One can see that the first column contains the values for the intercept, just like in a regular linear regression setup. These values are created for the polynomial regression curves that are about to be plotted. Next, a data frame is created containing the vector x and the matrix product of the data matrix poly_values and the regression coefficients.

```
poly_curve_data = data.frame(x, poly_values %*% poly_coef)
```

The first six rows of the resulting matrix look as following:
Table

The columns X1, X2 and X3 contain the estimated y-values from the regression equations stated in XXX for the corresponding x value in the column x. The final step is to transform the data to long format which makes plotting with ggplot2 more convenient.

```
poly_curve_data = gather(poly_curve_data, line, y, -x)
```

The resulting data frame is now in long format:

Table

The final plot is created using ggplot2 (Wickham, 2016).

```
poly_plot = ggplot(data = rma_data_long, aes(x = condition, y = value)) +
    geom_point() + labs(col = "Order of \npolynomial", x = "Condition", y =
    "Value", title = "Orthogonal polynomial contrasts") +
geom_path(data = poly_curve_data, aes(x, y, color = curve), lwd = 1.2) +
    scale_color_discrete(labels = as.character(1:(maxpoly))) +
theme_bw() + theme(panel.grid.major = element_blank(), panel.grid.minor =
    element_blank(), legend.key = element_rect(colour = "black"),
    plot.title = element_text(face="bold", hjust = .5))
```

A detailed resource for the use of ggplot2 is provided by Wickham (2016). Here, it is worth noticing that the code makes use of ggplot2's ability to plot data from multiple data frames in one plot. On the x-axis there are the factor levels, i.e. the groups from one to four. The y-axis displays the observed values, in this case the individually pulled weight. The data frame rma_data_long is used to plot the observed values of the subjects at the different factor levels. The regression curves are plotted using the geom_path() layer function. The curves are fitted smoothly through the 1000 x-values between 1 and 4 for each regression equation. Here, it is worth noticing that if the number of values for the regression curves is chosen too small, the curves might become wiggly. The color is mapped to the (categorical) curve variable such that each regression curve has a different color. Specifying the color argument withing the aes() function also initializes a legend. The other arguments are used to format the plot.

```
if (print_plot == TRUE){ print(poly_plot)} return(list(contrast_table =
    contrast_table, poly_plot = poly_plot))
```

The analysis of the Ringelmann-data reveals a strong and significant linear $(. = , (1) = , <)$ as well as a quadratic trend $(. = , (1) = , <)$. In combination with the assessment of the graphical representation of these trend components one shall deduct that the individual work performance is decreasing if the group size increases. In addition is the rate of decrease decreasing as well as the group size increases. This relation between group size and individual work performance was also already pointed out by Kravitz and Martin (1986).

9 R Packages

Packages in R are bundles of code, documentation, data and tests (Wickham; 2015). They make up a fundamental part of the R language since they allow to share code with other R users. In other programming languages these units are often referred to as libraries. The constantly growing amount of R packages is one characteristic of R as an open source software (Chambers; 2008). Packages can be downloaded from CRAN (https://cran.r-project.org/) which provides the official repository for R packages as well as from GitHub via the devtools package **?** Currently (January 26th, 2017), there are 9988 packages available on the CRAN website.

There are a few noticeable aspects why packages form such a fundamental basis of R. First, they allow to share code with others which enables a consistently growing range of functionalities in R. Moreover, when loading a package, all functions of that package are automatically loaded into the namespace. This is very time-saving especially when a package is frequently used as opposed to sourcing scripts with the required functions. Dependencies, other packages that the loaded package requires, are automatically loaded or even installed if necessary. And finally, the documentation of functions in a package allows the creator of the package as well as other users to understand the way a function works.

A personal motivation to create a package for repeated measures ANOVA was there were functionalities, like the adjusted confidence intervals, that we needed for a research project which had not been implemented in R, or at least not conveniently bundled in a single package. The MAGA package is an attempt to fill this void and provide the functionalities for other R users and researchers that use repeated measures ANOVA.

The Quantlet structure of the code made it easy to choose the functionalities that were turned into a function in the package. For creating the package, devtools (Wickham and Chang; 2015) and roxygen2 (Wickham et al.; 2013) were used. While devtools automates different tasks of package development, roxygen2 can be used to write package documentation directly into the R scripts of the functions which is then compiled to Rd format (Wickham, 2015). Since the MAGA package has not been published on CRAN yet (January, 26th 2017), devtools::install.github() may be used to install the package from its GitHub repository.

When writing the package, we carefully chose function names that indicate what the function does for a convenient use of the package. Another topic which is especially relevant when writing a package is error handling. This is crucial to ensure that the functions are robust against violation of the requirements of the function arguments. In each of the functions of the package, a sequence of if-statements was implemented to check if the requirements of the function input were met. When considering error handling in package creation it is important to differentiate between the functions stop() and warning(). Both return a message to the user but only stop() interrupts the code. So whenever we checked for requirements that were essential for the function to work we

chose stop(). Additionally, the function documentation specifies the data type of each argument (see section about function documentation). The following code displays the if-statements that were used to check whether the input data meets the requirements:

```
# id must either be an integer specifying the column position of the
    independent variable
if (id %in% 1:ncol(rma_data) == FALSE || length(id) != 1) {
stop("id must be an integer specifying the column position of the
    independent variable")
}
```

This part of the code ensures that the id variable is specified via a single integer which specifies the column position of the ID variable. This variable is an index variable which numerates the subjects. If the variable is not an integer between 1 and the number of columns of the input data an error message is returned.

```
# all variables must be numeric
if (all(sapply(rma_data, is.numeric)) == FALSE | any(sapply(rma_data,
    is.factor))) {
stop("All variables in rma_data must be numeric")
}
```

This if-statement controls for the right data type. All variables of the data must be numeric. Since R considers factor variables numeric, the code after the logical OR ensures that the function is also interrupted when one or more variables are factors. Again, an error message is returned in that case to inform the user about the requirements.ážđážđ

```
# n > k (i.e. more entities than factor levels)
if (n <= k) {
stop("Number of entities must exceed number of factor levels")
}
```

Moreover, for ANOVA methods to work, the number of subjects needs to exceed the number of factor levels. This means that the number of rows (automatically defined as n by the function) has to be strictly larger than k (the number of columns minus one for the index variable).

```
    # k >= 2 (i.e. at least two or more factor levels)
    if (k < 2) {
    stop("At least two factor factor levels required")
    }
```

Finally, the data needs to have at least two factor levels, since otherwise no ANOVA model can be estimated. This is controlled by the last of the if-statements. A particular case where the

warning() function was used is described in section XXX. Function documentation can be demonstrated on one particular function since the necessary steps apply to all other functions as well. For demonstration purposes, the function ow_rma_opc was chosen. The following code is compiled by roxygen2 to an Rd file which contains the information that are displayed when the help page in R is called, for example via the command ?ow_rma_opc.

```
Each line in the documentation needs to start with #'.
The first line is the heading of the help page and summarizes briefly what
    the function does.
#' Estimate and plot orthogonal polynomial trends
#'

The next line is displayed by roxygen2 via the heading Description and
    features short yet precise information about the function.
#' Compute orthogonal polynomial contrasts and plot orthogonal polynomial
    regression curves for a repeated measures ANOVA.
#'

The next section Usage is automatically created by roxygen2. It demonstrates
    how to call the function.

Via @param the function parameters are listed under the heading Arguments.
    Here, it is important to inform the user about the required data types,
    e.g. numeric vector or data.frame. Moreover, default values, if
    existing, are documented.
#' @param ow_rma_data An object of type data.frame. Each row should
    represent one subject and each column one variable.
#' @param id An integer specifying the column position of the subject ID.
    Default is 1. Set to "none" if the data does not contain an ID variable.
#' @param maxpoly Integer. Specifies the highest order polynomial for the
    contrast analysis. Truncated to number of factor levels (k) -1 if
    larger. Default is NA which will be set to k-1 within the function.
#'

In the next step, the object(s) that the function returns need to be
    specified. This is achieved via @return and summarized under the section
    label Value. Here, the function returns a list. In the next lines \item
    is used to describe the objects in particular. The list stores a data
    frame containing the results from the polynomial contrast analysis and
    the ggplot object that displays the orthogonal polynomial regression
    curves.
#' @return Returns an object of type list.
#' \item{contrast_table}{An object of type data.frame containing the
    contribution of the polynomial trends to the total factor effect and
    their respective significance levels}
```

```
#' \item{poly_plot}{A ggplot object. Displays the orthogonal polynomial
    regression curves.}

The last sections contain the authorsâĂŹ names, notes, literature and
    examples if provided.
#' @author Joachim Munch, Frederik Schreck, Quang Nguyen Duc, Constantin
    Meyer-Grant, Nikolas Hoeft
#' @note LITERATURE
#' @examples
#'

Finally @rdname specifies where the documentation is stored, here for the
    function ow_rma_opc, and @export is especially important to ensure that
    the function is added to the namespace file.
#' @rdname ow_rma_opc
#' @export
```

A final word needs to be said about the description file. A template for this is automatically created when building a new package in RStudio. Basically, this file contains some meta data about the package like a description, which does not require further elaboration. One important aspect though is the use of dependencies. These are other packages that are required to run the functions in the new package. In the section Imports the names of the required packages are specified. When the MAGA package is installed, R automatically ensures that the dependencies (here: dplyr, ggplot2 and tidyverse) are installed as well. Moreover, they are also loaded whenever MAGA is loaded via the library() command.

Package: MAGA Type: Package Title: A package to make ANOVA great again! Version: 0.1.0 Author: Joachim Munch, Frederik Schreck, Quang Nguyen Duc, Constantin Meyer-Grant, Nikolas Hoeft Maintainer: Nikolas Hoeft <nikolas.hoeft@hu-berlin.de> Description: A package for repeated measures ANOVA models. Allows to simulate data, compute ANOVA and RM ANOVA models, investigate error term reduction due to model selection, compute and display adjusted confidence intervals. Furthermore it enables to conduct Mauchly's test for sphericity and it provides various correction factors to adjust p-values in the presence of sphericity. The package also supplies a function to fit orthogonal polynomial contrasts and to plot orthogonal polynomial regression curves. Imports: dplyr, ggplot2, tidyverse License: Encoding: UTF-8 LazyData: true RoxygenNote: 5.0.1
Of course there are many other possible steps for package development. The ones described above however are the most important and basic ones and were therefore chosen for this report.

Appendix

Literaturverzeichnis

Bakeman, R. (2005). Recommended effect size statistics for repeated measures designs, *Behavior research methods* **37**(3): 379–384.

Box, G. E. P. et al. (1954). Some theorems on quadratic forms applied in the study of analysis of variance problems, i. effect of inequality of variance in the one-way classification, *The Annals of Mathematical Statistics* **25**(2): 290–302.

Chambers, J. (2008). *Software for data analysis: programming with R*, Springer Science & Business Media.

Cousineau, D. (2005). Confidence intervals in within-subject designs: A simpler solution to loftus and masson's method, *Tutorials in quantitative methods for psychology* **1**(1): 42–45.

Geisser, S., Greenhouse, S. W. et al. (1958). An extension of box's results on the use of the f distribution in multivariate analysis, *The Annals of Mathematical Statistics* **29**(3): 885–891.

Greenhouse, S. W. and Geisser, S. (1959). On methods in the analysis of profile data, *Psychometrika* **24**(2): 95–112.

Huynh, H. and Feldt, L. S. (1976). Estimation of the box correction for degrees of freedom from sample data in randomized block and split-plot designs, *Journal of educational statistics* **1**(1): 69–82.

Kravitz, D. A. and Martin, B. (1986). Ringelmann rediscovered: The original article.

Loftus, G. R. and Masson, M. E. J. (1994). Using confidence intervals in within-subject designs, *Psychonomic bulletin & review* **1**(4): 476–490.

Morey, R. D. (2008). Confidence intervals from normalized data: A correction to cousineau (2005), *reason* **4**(2): 61–64.

O'Brien, F. and Cousineau, D. (2014). Representing error bars in within-subject designs in typical software packages, *The Quantitative Methods for Psychology* **10**(1): 56–67.

Olejnik, S. and Algina, J. (2003). Generalized eta and omega squared statistics: measures of effect size for some common research designs, *Psychological methods* **8**(4): 434–447.

Rutherford, A. (2001). *Introducing ANOVA and ANCOVA: a GLM approach*, Sage.

Wickham, H. (2015). *R packages*, O'Reilly Media, Inc.

Wickham, H. (2016). *ggplot2: elegant graphics for data analysis*, Springer.

Wickham, H. and Chang, W. (2015). devtools: Tools to make developing r code easier, *R package version* **1**(0).

Wickham, H., Danenberg, P. and Eugster, M. (2013). roxygen2: In-source documentation for r, *R package version* **3**(0).