0. SUPPLEMENTAL TABLES and FIGURES

0.1 Fig. S1

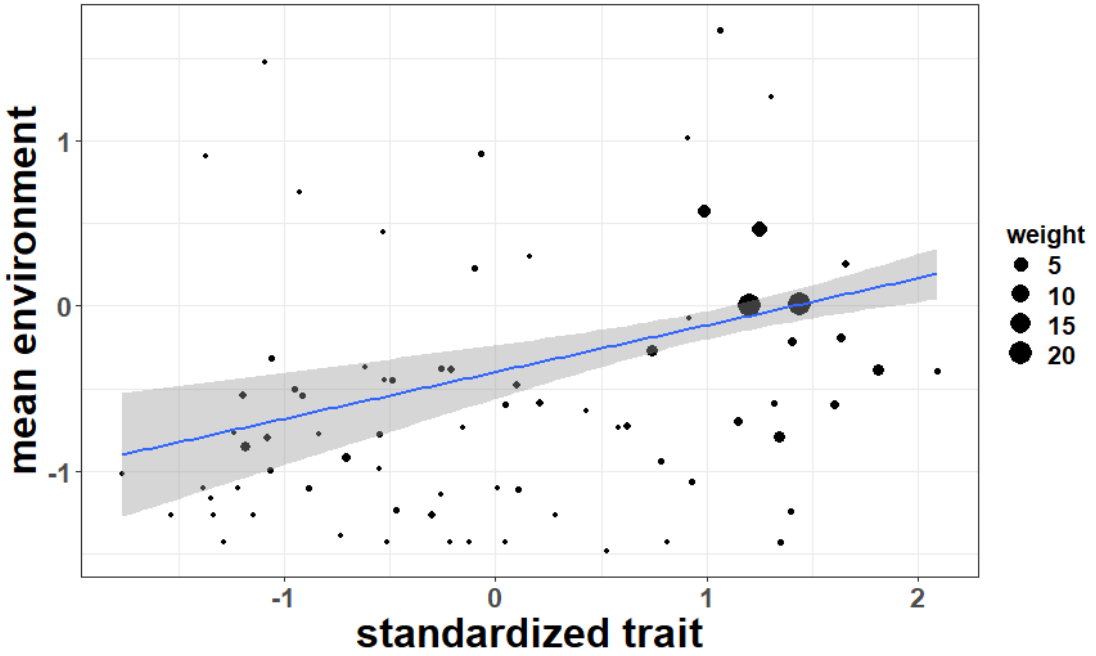

Fig. S1. Weighted regression of the mean environment of a species on to a particular trait for 75 species with their total abundance as weight. The environmental variable is a moisture gradient and the trait the leaf carbon-to-nitrogen ratio (ter Braak 2019). The fitted line with confidence band is based on parametric weighted regression.

0.2 Fig. S2

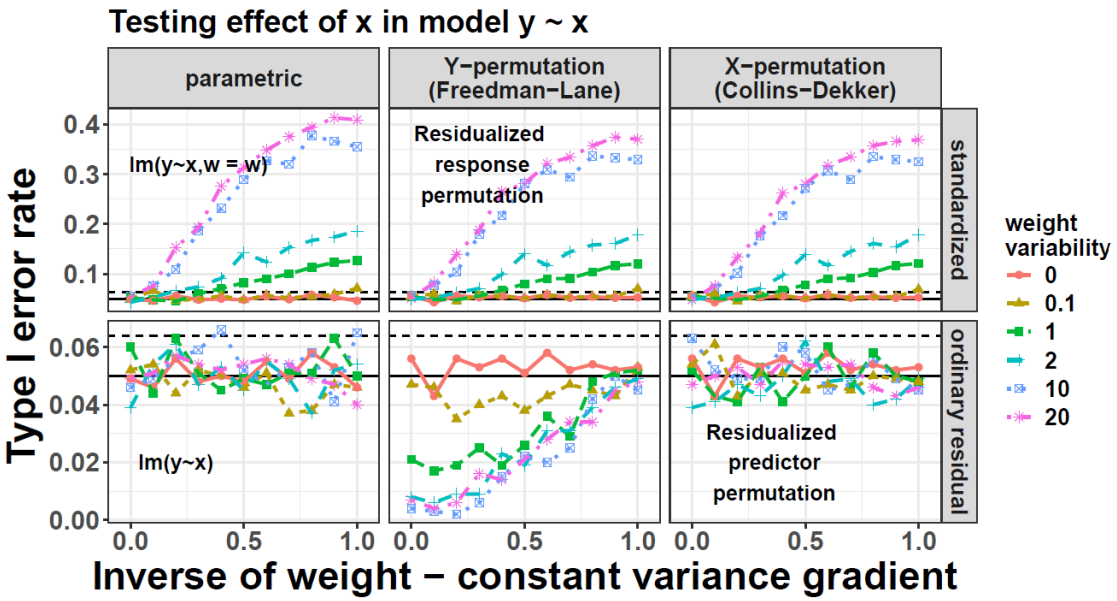

Fig. S2. Type I error rates of testing the effect of x in the linear model y~1+x with error variances that runs from purely inversely of weight (scale value 0) to constant variance (scale value 1). All panels use WLS, except for the lower-left panel which uses unweighted regression (OLS). Weights were drawn from a negative binomial with mean 10, increased by 1 to avoid zero weights; weight variability is its overdispersion, except that the value 0 indicates 'no variability', i.e. equal weights. The six levels of increasing weight variability yield coefficients of variation of the weights of 0, 0.41, 0.95, 1.3, 2.9 and 4.1. The horizontal solid line is at the nominal significance threshold; rates (from 1000 simulations) above the dashed line (at 0.064) are significantly greater than 0.05.

0.3 Fig. S3

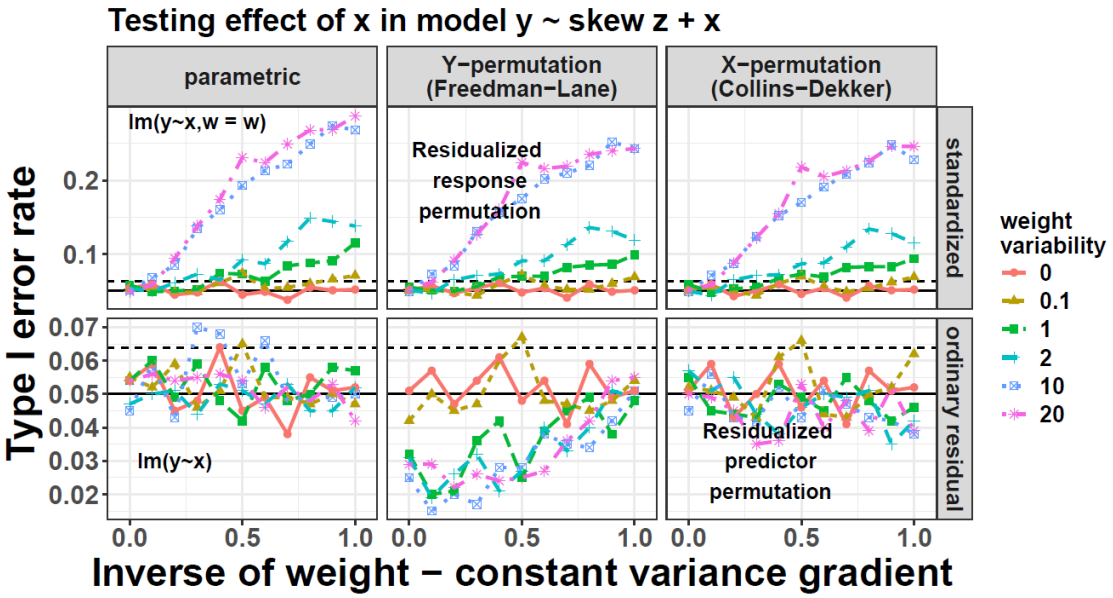

Fig. S3. As Fig. 1, but with covariates and predictors made skew by the transformation $(x, z \leftarrow \exp(x, z)$.

0.4 Fig. S4

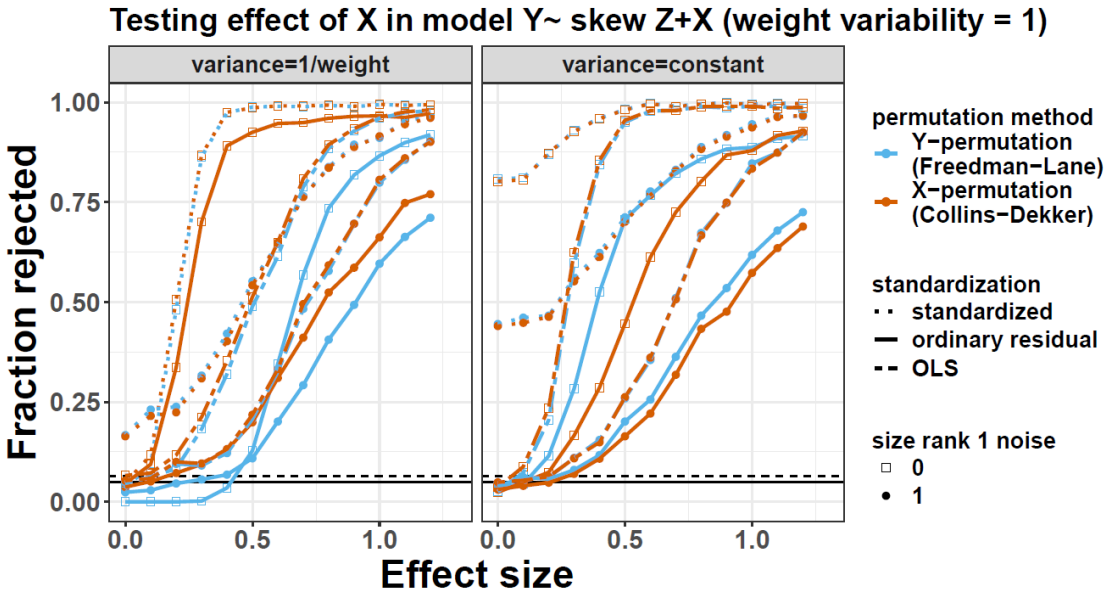

Fig. S4. As Fig. 4, but with covariates and predictors made skew by the transformation ((**X**,**Z**← exp (**X**, **Z**)).

Appendices to:

ter Braak, C.J.F. (2021) Predictor versus response permutation for significance testing in weighted regression and redundancy analysis. *Journal of Statistical Computation and Simulation.*
http://dx.doi.org/10.1080/00949655.2021.2019256

The R code is also available in a zip file on
http://dx.doi.org/10.6084/m9.figshare.14207795

1. Appendix S1: Reduced rank model simulation

The data for the simulations in Sections 4.2 "Testing effects by weighted RDA" and 4.3 "Testing the axes of a weighted RDA" was generated by the linear model

$$y_{ik} = a_{0k} + a^*_{1k} z^*_i + b^*_{1k} x^*_{1i} + b^*_{2k} x^*_{2i} + e_{ik} \text{ with } i = 1, \ldots n; k = 1, \ldots, m \quad (1)$$

in which $\mathbf{z}^* = (z^*_1, .., z^*_n)'$, $\mathbf{x}^*_1 = (x^*_{11}, .., x^*_{1n})'$ and $\mathbf{x}^*_2 = (x^*_{21}, .., x^*_{2n})'$ are three latent variables generated as

$$\mathbf{z}^* = \mathbf{Z}\tilde{\mathbf{a}},\ \mathbf{x}^*_1 = \tilde{\mathbf{X}}_1\tilde{\mathbf{b}}_1 \text{ and } \mathbf{x}^*_2 = \tilde{\mathbf{X}}_2\tilde{\mathbf{b}}_2 \quad (2)$$

with $\mathbf{Z}$, $\tilde{\mathbf{X}}_1$ and $\tilde{\mathbf{X}}_2$ with 2, 4 and 4 columns, respectively, with each row an independent multivariate normal with mean zero and an AR(1) covariance matrix, so that each column has variance 1 and the correlation between the j^{th} and j'^{th} column of each matrix is $\rho^{j-j'}$; the elements of vectors $\tilde{\mathbf{a}}$, $\tilde{\mathbf{b}}_1$ and $\tilde{\mathbf{b}}_2$ are all equal and set such that $\mathbf{z}^*$, $\mathbf{x}^*_1$ and $\mathbf{x}^*_2$ have unit variance. The coefficients a^*_{1k}, b^*_{1k} and b^*_{2k} are independent standard normal and multiplied by $a_1 = 1$, b_1 and b_2 so as to set the importance of the three latent dimensions (axes) of the model. The error is a combination of rank-1 and random noise

$$e_{ik} = d^*_k e^*_i + \tilde{e}_{ik} \text{ with } i = 1, \ldots n; k = 1, \ldots, m \quad (3)$$

with d^*_k, e^*_i and $\tilde{e}_{ik}$ all independent standard normal. The rank-1 error, when present, is of the same size as that of $\mathbf{z}^*$. The random noise was independent normal with zero mean and variance that was either constant (1) or inversely related to the simulated weights ($var(e_{ik}) \propto 1/w_i$), k=1,...,m).

The columns in $\mathbf{Z}$, $\tilde{\mathbf{X}}_1$ and $\tilde{\mathbf{X}}_2$ are independent. The data for analysis are made more challenging by adding a fourth set of predictor variables without any effect on the response and by generating AR(1) correlation between the four sets of variables. In particular, the data used for analysis are $\mathbf{Z}$, $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$, with $\mathbf{X}_1$ is a convex combination of $\tilde{\mathbf{X}}_1$ and $\mathbf{z}^*$, and $\mathbf{X}_2$ a convex combination of $\tilde{\mathbf{X}}_2$ and $\mathbf{x}^*_1$ so that the correlation between the columns of $\mathbf{X}_1$and $\mathbf{X}_2$ with $\mathbf{z}^*$ and $\mathbf{x}^*_1$, respectively, is 0.7. Finally, $\mathbf{X}_3$ has 4 columns, generated as $\mathbf{X}_2$, except that these columns have a correlation of 0.7 with $\mathbf{x}^*_2$ instead of with $\mathbf{x}^*_1$. The variables in $\mathbf{X}_3$ have no influence on the response $\mathbf{Y}$.

Eqs (1) and (2) define a reduced-rank model of the form

$$\mathbf{Y}_{(n\times m)} = \mathbf{1}_{(n)}\boldsymbol{\mu}'_{(m)} + \mathbf{Z}_{(n\times 2)}\tilde{\mathbf{a}}_{(2)}\mathbf{a}^{*\prime}_{(m)} + \tilde{\mathbf{X}}_{1(n\times 4)}\tilde{\mathbf{b}}_{1(4)}\mathbf{b}^{*\prime}_{1(m)} + \tilde{\mathbf{X}}_{2(n\times 4)}\tilde{\mathbf{b}}_{2(4)}\mathbf{b}^{*\prime}_{2(m)} + \mathbf{E}, \quad (4)$$

where subscripts between parentheses indicate the dimension of the vectors and matrices with $\mathbf{z}^* = \mathbf{Z}_{(n\times 2)}\tilde{\mathbf{a}}_{(2)}$, $\mathbf{x}_1^* = \widetilde{\mathbf{X}}_{1(n\times 4)}\tilde{\mathbf{b}}_{1(4)}$ and $\mathbf{x}_1^* = \widetilde{\mathbf{X}}_{2(n\times 4)}\tilde{\mathbf{b}}_{2(4)}$ defining the three axes (dimensions) of the model. In terms of the multivariate regression model of Eq. **Error! Reference source not found.**, $\mathbf{Z} = (\mathbf{1}_n : \mathbf{Z}_{n\times 2})$, $\mathbf{X} = \left(\widetilde{\mathbf{X}}_{1(n\times 4)} : \widetilde{\mathbf{X}}_{2(n\times 4)}\right)$, $\mathbf{A} = \left(\boldsymbol{\mu}'_{(m)} : \tilde{\mathbf{a}}_{(2)}\mathbf{a}^{*\prime}_{(m)}\right)$ and $\mathbf{B} = (\mathbf{B}_1 : \mathbf{B}_2)$, with $\mathbf{B}_1 = \tilde{\mathbf{b}}_{1(4)}\mathbf{b}^{*\prime}_{1(m)}$ and $\mathbf{B}_2 = \tilde{\mathbf{b}}_{2(4)}\mathbf{b}^{*\prime}_{2(m)}$. For the analysis, $\mathbf{X} = (\mathbf{X}_1 : \mathbf{X}_2 : \mathbf{X}_3)$, where $\mathbf{X}_1$, $\mathbf{X}_2$ and $\mathbf{X}_3$ are a convex combination of $\widetilde{\mathbf{X}}_1$, $\widetilde{\mathbf{X}}_2$ and $\widetilde{\mathbf{X}}_3$ and $\mathbf{z}^*$, $\mathbf{x}_1^*$ and $\mathbf{x}_2^*$, respectively.

2. Appendix S2: Illustration of definitions of X- and Y-permutation methods and computational equivalences in R

With univariate data, generated as in Section 4.1, this R-script define the four version of permutation using the `lm` and `anova` function and its associates (`fitted`, `residuals`) and considers computational short-cuts. The Monte Algorithm of Section 2.2 is given and is applied to each permutation method, using both `lm` with `anova` and using the `randperm_RDA` function of Appendix S3.

```
rm(list=ls(all=TRUE))  # remove all existing items from the workspac
e
{
# if you have the required file residualXY_permutation.r in the  fol
der /Rfunctions
do_compare_with_function_randperm_RDA <- TRUE
# otherwise outcomment
# do_compare_with_function_randperm_RDA <- FALSE
if (do_compare_with_function_randperm_RDA) {
  source("Rfunctions/residualXY_permutation.r")
  mysvd <- dummysvd
}

part_constantVariance <- 0.5 # should be between 0 and 1
weight_overdispersion <-  20 # 0 gives equal weights, otherwise the
variance(w)= 10 + 100*weight_overdispersion
CV_weights <- sqrt(10+100*weight_overdispersion)/ (10+1)
print(c(CV_weights=CV_weights )) # 3.5 for weight_overdispersion = 1
5

n <- 30
effectX <- 0 # 0 to simulate the null hypothesis, unequal 0 for the
alternative hypothesis

# Generate univariate data (y,x,z); (x,z) with correlation 0.7 -----
-------
#
set.seed(123)
eps <- 1.0e-12
number_of_errors <- 0 # count for number of  untrue numerical tests
if (weight_overdispersion > 0)  w <- rnbinom(n, size = 1/weight_over
dispersion, mu = 10)+1 else w <- rep(1,n)
Sigma <- matrix(c(1,0.7,0.7,1), nrow = 2, ncol=2)
```

```
XZ <- MASS::mvrnorm(n,mu=rep(0,2),Sigma)
z <- XZ[,1]
x <- XZ[,2]
sdw <- 1/sqrt(w)
sdw <-  n * sdw/sum(sdw)
Noise1 <- rnorm(n)
Noise2 <- sdw*rnorm(n)
noise <- part_constantVariance * Noise1 + sqrt(1-part_constantVarian
ce^2) * Noise2
# generative linear model for regression data
y <- 1 + z + effectX * x   + noise # equation (1)


# Section 2.1 & 2.3: weighted linear model analysis ---------------
---------------------------------

H0 <- lm(y~z, weights = w)
H1 <- lm(y~z+x, weights = w)
summary(H1)
y0 <- fitted(H0)
ey0 <- y - y0
anova(H0, H1) #
F0 <- anova(H0, H1)$F[2]  # equation (3)
F0_alt0 <- anova(H1)$F[2]
(if (abs(F0 - F0_alt0)> eps) number_of_errors <- number_of_errors +
1) # NULL is OK

# Section 2.3: unweighted Y-permutation (ordinary Freedman-Lane)  --
-----------------------------------------------------------------
-

permutation <- sample(n)
pi_ey0 <- ey0[permutation]
y_pi <- y0 + pi_ey0 # equation (6)
H0 <- lm(y_pi ~ z, weights = w)
H1 <- lm(y_pi ~ z + x, weights = w)
anova(H0, H1) #
Fk_uwY <- anova(H0, H1)$F[2]
Fk_uwY_alt0 <- anova(H1)$F[2]
(if (abs(Fk_uwY - Fk_uwY_alt0)> eps) number_of_errors <- number_of_e
rrors + 1) # NULL is OK


# Section 2.1: weighted linear model analysis via transformation to
ordinary least-squares (OLS) -------------------------------------
------------
# transformation: multiplication by the square-root of w;
# do not forget the intercept!
sw   <-  sqrt(w)
yw   <-  sw * y
intw <-  sw * rep(1, length(y))
zw   <-  sw * z
xw   <-  sw * x
#remove(y,x,z,w)
```

```r
# Section 2.1: ordinary least-squares (OLS) on weighted data -------
---------------

H0 <- lm(yw~0 + intw + zw)
H1 <- lm(yw~0 + intw + zw+xw)
summary(H1)
yw0 <- fitted(H0)
eyw0 <- yw - yw0  # equation (4)
anova(H0, H1) #
F0w <- anova(H0, H1)$F[2]  # equation (3)
(if (abs(F0 - F0w)> eps) number_of_errors <- number_of_errors + 1) #
NULL is OK

# Section 2.3: weighted Y-permutation (standardized Freedman-Lane) v
ia transformation to unweighted ----------------------------------
----------------------------------

pi_eyw0 <- eyw0[permutation]
yw_pi <- yw0 + pi_eyw0 # equation (7)
H0w <- lm(yw_pi ~0 + intw + zw)
H1w <- lm(yw_pi ~0 + intw + zw + xw)
anova(H0w, H1w) #
Fk_wY <- anova(H0w, H1w)$F[2]

# Section 2.4: unweighted X-permutation (ordinary Collins-Dekker)  -
---------------------------------------------------------------------
--

# calculated x-residuals  (same as above)
ex  <- residuals(lm(x~1+z, weights = w)) # below equation (5)
# we use the inverse of the permutation used above to get
# results that are as similar as possible to Y-permutation
inverse_permutation <- order(permutation)
pi_ex <- ex[inverse_permutation]

H0 <- lm(y ~ z,         weights = w)
H1 <- lm(y ~ z + pi_ex, weights = w)
anova(H0, H1) #
Fk_unwX <- anova(H0, H1)$F[2]

# Intermezzo: # Note of section paragraph of section 2.4 X-permutati
on
# note: y~z+x and y~z+ex give same F-ratio
#         (i.e pi_ex for the identity permutation, i.e no permutatio
n )
#       as can be seen from the next two anovas
H1e <- lm(y ~ z + ex, weights = w)
H1x <- lm(y ~ z +  x, weights = w)
anova(H1e) #
anova(H1x)
F0_alt1 <- anova(H0, H1e)$F[2]
(if (abs(F0 - F0_alt1)> eps) number_of_errors <- number_of_errors +
1) # NULL is OK
```

```r
# Intermezzo end



# Section 2.4: weighted X-permutation (standardized Collins-Dekker)
via transformation to OLS ----------------------------------------
---------------------------

# we need eyw0 and ex
H0 <- lm(yw~0 + intw + zw)
eyw0 <- residuals(H0)
exw  <- residuals(lm(xw~ 0 + intw + zw))
H1 <- lm(yw ~ 0 + intw + zw + xw)
F0X <- anova(H1)$`F value`[3]
F0-F0X

pi_exw <- exw[inverse_permutation]
pi_exw_orth_to_zw <- residuals(lm(pi_exw ~ 0 + intw + zw))
H1 <- lm(yw ~ 0 + intw + zw + pi_exw_orth_to_zw )
Fk_wX  <-  anova(H1)$F[3]


# Section 2.4: weighted X permutation via WLS ----------------------
-----------------------------------------------

pi_exw2 <- exw[inverse_permutation]/sw # equation (9)

H1 <- lm(y ~ z + pi_exw2, weights = w)
Fk_wX_alt <- anova(H1)$F[2]  # below equation (9)
(if (abs(Fk_wX - Fk_wX_alt)> eps) number_of_errors <- number_of_erro
rs + 1) # NULL is OK




# Section 2.5 unweighted Y-permutation (ordinary Freedman-Lane), alt
ernative computational methods -----------------------------------
---------------------------------

# a: use permuted residual pi_ey0 instead of  y_pi
H0_ey0  <- lm(pi_ey0 ~ z, weights = w)
H1_ey0  <- lm(pi_ey0 ~ z + x, weights = w)
anova(H0_ey0 , H1_ey0) #
Fk_uwY_alt1 <- anova(H0_ey0 , H1_ey0 )$F[2]
(if (abs(Fk_uwY - Fk_uwY_alt1)> eps) number_of_errors <- number_of_e
rrors + 1) # NULL is OK

# alternative computation: section 2.5
# b: compute above in two steps using Z and X w-orthonal to Z
# residual sum of square of pi_ey0 ~ z is:
ss_ez <- anova(H0_ey0)$`Sum Sq`[2]
# calculate sum of squares of due to x of model pi_ey0 ~ ex is:

# ex is w-orthogonal to 1 and z
```

```r
ex  <- residuals(lm(x~1+z, weights = w))
H1_ey0_ex <- lm(pi_ey0 ~ ex, weights = w)
# fitted sum of squares of pi_ey0 ~ ex is:
ss_x  <-anova(H1_ey0_ex)$`Sum Sq`[1]
Fk_uwY_alt2 <- ss_x/((ss_ez - ss_x)/(n-3))
(if (abs(Fk_uwY - Fk_uwY_alt2)> eps) number_of_errors <- number_of_e
rrors + 1) # NULL is OK

# alternative computation: section 2.5
# The above b: uses extensive R-code and how anova works.
# Make things more explicit using simple code
# The difficult is that pi_ey0 has a non-zero weighted mean
# you can see by inspecting fit_int or ss_int below.
# We need the pure regression sum of squares due to z and x|z
# So:
# c: compute above in two steps using Z and X w-orthonal to Z
# make z w_orthogonal to the intercept
z_orth  <- residuals(lm(z~1,              weights = w))
fit_int <-    fitted(lm(pi_ey0~ 1,        weights = w)) # this is a
term overlooked in Canoco 3-5.12, corrected in Canoco 5.15
fit_z   <-    fitted(lm(pi_ey0~ 0+z_orth, weights = w))
fit_x   <-    fitted(lm(pi_ey0~ 0+ex,     weights = w))

ss_tot <- sum(w*pi_ey0^2)
ss_int <- sum(w*fit_int^2)
ss_z   <- sum(w*fit_z^2)
ss_x   <- sum(w*fit_x^2)
rss <- ss_tot - ss_int - ss_z - ss_x
Fk_uwY_alt3 <- ss_x / (rss/(n-3) )
(if (abs(Fk_uwY - Fk_uwY_alt3)> eps) number_of_errors <- number_of_e
rrors + 1) # NULL is OK
number_of_errors



# Section 2.5: unweighted X-permutation (ordinary Collins-Dekker), a
lternative computational methods ----------------------------------
-----------------------------------

# a: we can use residual y under null model instead of y itself
H0 <- lm(ey0 ~ z,         weights = w)
# note that the second partialling of Double Semi-Partialling (DSP)o
f Dekker et al. (2007)
# is implicit in the next line as z is included in the model.
H1 <- lm(ey0 ~ z + pi_ex, weights = w)
anova(H0, H1) #
Fk_unwX_alt1 <- anova(H0, H1)$F[2]
(if (abs(Fk_unwX - Fk_unwX_alt1)> eps) number_of_errors <- number_of
_errors + 1) # NULL is OK

# b: compute the same in two steps
# step 1: compute the residual of regression of pi_ex on Z
#           to ensure that pi_ex_orth_to_z is w-orthogonal to 1 and
z.
```

```r
# this is the explicit second partialling of Double semi-partialling
of Dekker et al. (2007)
pi_ex_orth_to_z <- residuals(lm(pi_ex~1+z, weights=w))
# step 2: fit ey0 to pi_ex_orth_to_z
fit_x <- fitted(lm(ey0~ 0+pi_ex_orth_to_z, weights = w))
ss_x <- sum(w*fit_x^2)
ss_tot <- sum(w*ey0^2)
rss <- ss_tot - ss_x
Fk_unwX_alt2 <- ss_x / ((ss_tot - ss_x)/(n-3) )
(if (abs(Fk_unwX - Fk_unwX_alt2)> eps) number_of_errors <- number_of
_errors + 1) # NULL is OK
# Conclusion the F-ratio is monotonic with ss_x as a/(a-b) is monoto
nic with a for fixed b.
number_of_errors

# Section 2.5: weighted Y-permutation (standardized Freedman-Lane),
alternative computation, via transformation to OLS ----------------
----------------------------------------------------

# a: use permuted residual pi_eyw0 instead of  yw_pi
H0_eyw0  <- lm(pi_eyw0 ~ 0 + intw +  zw)
H1_eyw0  <- lm(pi_eyw0 ~ 0 + intw +  zw + xw)
anova(H0_eyw0 , H1_eyw0) #
Fk_wY_alt1 <- anova(H0_eyw0 , H1_eyw0 )$F[2]
(if (abs(Fk_wY - Fk_wY_alt1)> eps) number_of_errors <- number_of_err
ors + 1) # NULL is OK

#b: use an x that is made w-orthogonal to int w and zw
# i.e. make xw w-orthogonal to intw and zw: new variable exw
exw  <- residuals(lm(xw ~ 0 + intw + zw))

# The difficulty is that pi_eyw0 has a non-zero weighted mean.
# You can see by inspecting fit_int or ss_int below.
# We need the pure regression sum of squares due to zw and xw|zw
# So:
# c:
# make zw w_orthogonal to the intercept
zw_orth  <-    residuals(lm(zw   ~ 0 + intw))
fit_int <-     fitted(lm(pi_eyw0 ~ 0 + intw)) # this is a term overl
ooked in Canoco 3-5.12, corrected in Canoco 5.15
fit_zw   <-    fitted(lm(pi_eyw0 ~ 0 + zw_orth))
fit_xw   <-    fitted(lm(pi_eyw0 ~ 0 + exw))

ss_tot <- sum(pi_eyw0^2)
ss_int <- sum(fit_int^2)
ss_zw   <- sum(fit_zw^2)
ss_xw   <- sum(fit_xw^2)
rss <- ss_tot - ss_int - ss_zw - ss_xw
Fk_wY_alt3 <- ss_xw / (rss/(n-3) )
(if (abs(Fk_wY - Fk_wY_alt3)> eps) number_of_errors <- number_of_err
ors + 1) # NULL is OK
number_of_errors
```

```r
# Section 2.5: UNweighted X-permutation (ordinary Collins-Dekker) vi
a transformation to OLS-------------------------------------------
-------------------------

# calculated x-residuals as exw residual divided by sw
ex2  <- residuals(lm(xw~ 0 + intw + zw)) /sw
range(ex-ex2)
(if (sum(abs(range(ex-ex2)))> eps) number_of_errors <- number_of_err
ors + 1) # NULL is OK
# permute the x-residual ex2 and multiply by sw to use in unweigthed
analysis.
pi_exw <- ex2[inverse_permutation]*sw

H0 <- lm(yw ~ 0 + intw +  zw)
H1 <- lm(yw ~ 0 + intw +  zw + pi_exw)
anova(H0, H1) #
Fk_unwX_alt3 <- anova(H0, H1)$F[2]
(if (abs(Fk_unwX - Fk_unwX_alt3)> eps) number_of_errors <- number_of
_errors + 1) # NULL is OK

# alternative computation: section 2.5
# a: we can use residual yw under null model instead of yw itself
H0 <- lm(eyw0 ~ 0 + intw +  zw)
H1 <- lm(eyw0 ~ 0 + intw +  zw + pi_exw)
anova(H0, H1) #
Fk_unwX_alt4 <- anova(H0, H1)$F[2]
(if (abs(Fk_unwX - Fk_unwX_alt4)> eps) number_of_errors <- number_of
_errors + 1) # NULL is OK

# alternative computation: section 2.5
# b: compute the same in two steps
# step 1: compute the residual of regression of pi_exw on intw + zw
#           to ensure that pi_exw_orth_to_zw is orthogonal to 1 and
zw.
pi_exw_orth_to_zw <- residuals(lm(pi_exw~ 0 + intw + zw))
# step 2: fit eyw0 to intw + pi_exw_orth_to_zw
fit_xw <- fitted(lm(eyw0~ 0 + intw + pi_exw_orth_to_zw))
ss_xw <- sum(fit_xw^2)
ss_tot <- sum(eyw0^2)
rss <- ss_tot - ss_xw
Fk_unwX_alt5 <- ss_xw / ((ss_tot - ss_xw)/(n-3) )
(if (abs(Fk_unwX - Fk_unwX_alt5)> eps) number_of_errors <- number_of
_errors + 1) # NULL is OK
# Conclusion the F-ratio is monotonic with ss_xw as a/(a-b) is monot
onic with a for fixwed b.
number_of_errors


#  Section 2.6: raw X- and Y-permutation (Z =1) -------------------

w1 <- rep(1, n)
#w1 <- w # so see numerically that raw Y-data permutation is equal t
o unweigthed Y-permutation,
#          but otherwise
```

```r
#          to give a counterexample of potential equivalences for un
equal weights even with a trivial covariate
# a trivial covariate  (a constant, here 0)
z0 <- rep(0,n)
#z0 <- z # to give a counterexample of potential equivalences for a
non-trivial covariate, even with equal weights

#  Section 2.6: overall-null Y-permutation  ----------------------
-------------------
y0 <- fitted(lm(y~z0, weights = w1))
ey0 <- y - y0
pi_ey0 <- ey0[permutation]
y_pi <- y0 + pi_ey0 # equation (6)
H0 <- lm(y_pi ~ z0, weights = w1)
H1 <- lm(y_pi ~ z0 + x, weights = w1)
anova(H0, H1) #
Fk_Ye <- anova(H0, H1)$F[2]

# Section 2.6: overall-null X-permutation  ----------------------
-----------------------------------------

# calculated x-residuals  (same as above)
ex  <- residuals(lm(x~1+z0, weights = w1))
# we use the inverse of the permutation used above to get
# results that are as similar as possible to Y-permutation
pi_ex <- ex[inverse_permutation]
pi_ex_orth_to_z <- residuals(lm(pi_ex ~ 1 + z0 , weights = w1))

H0 <- lm(y ~ z0,         weights = w1)
H1 <- lm(y ~ z0 + pi_ex_orth_to_z, weights = w1)
anova(H0, H1) #
Fk_Xe <- anova(H0, H1)$F[2]
Fk_Ye - Fk_Xe
(if (abs(Fk_Ye - Fk_Xe)> eps) number_of_errors <- number_of_errors +
1) # NULL is OK
# X- and Y-permutation equivalent for equal weight and trivial covar
iate.

# Section 2.6: raw Y-data permutation ----------------------------
-------------------


pi_y <- y[permutation]
diff(range(pi_y - y_pi)) # numerically zero # This also hold true fo
r unequal weights of unweighted Y-permutation
H0r <- lm(pi_y ~ z0, weights = w1)
H1rawy <- lm(pi_y ~ z0 + x, weights = w1)
anova(H0r, H1rawy) #
Fk_raw_y <- anova(H0r, H1rawy)$F[2]
(if (abs(Fk_Xe - Fk_raw_y)> eps) number_of_errors <- number_of_error
s + 1) # NULL is OK
# raw y and equivalent with X-permutation  for equal weights and tri
vial covariate
(if (abs(Fk_Ye - Fk_raw_y)> eps) number_of_errors <- number_of_error
```

```
s + 1) # NULL is OK
# raw y-data permutation equivalent with Y-permutation for equal wei
ghts and trivial covariate
abs(Fk_Ye - Fk_raw_y) #

# Section 2.6: raw X-data permutation ----------------------------
-------------------

pi_x <- x[inverse_permutation]
H0r <- lm(y ~ z0, weights = w1)
H1rawx <- lm(y ~ z0 + pi_x, weights = w1)
anova(H0r, H1rawx) #
Fk_raw_x <- anova(H0r, H1rawx)$F[2]
(if (abs(Fk_raw_y - Fk_raw_x)> eps) number_of_errors <- number_of_er
rors + 1) # NULL is OK
# raw y-data permutation and raw x-data permutation equivalent with
equal weights and trivial covariate
(if (abs(Fk_Xe - Fk_raw_x)> eps) number_of_errors <- number_of_error
s + 1) # NULL is OK
# raw x-data permutation and unweighted X-permutation equivalent wit
h equal weights and trivial covariate


# run this code
#by redefining w1 <- w and see that things are not equivalent
# or
#by redefining z0 <- z

# with unequal weights and trivial covariate  :
# raw Y-permutation is equivalent to unweighted Y-permutation. diff(
range(pi_y - y_pi)) < 1.e-10
cat("number of errors ", number_of_errors, "\n")


# Section 2.2: Monte Carlo algorithm and tests of function randperm_
RDA (four methods of permutation) ------------------------------
----

K <- 199 # number of Monte Carlo data sets; number of repetitions
N <- length(y)
perm.mat <- matrix(0,nrow = K, ncol = N)
for (k in 1:K){
  perm.mat[k,] <- sample(N)
}


# Section 2.2: Step 1 F0 of data

H1 <- lm(y~z+x, weights = w)
F0 <- anova(H1)$F[2]

#  Monte Carlo test by Weighted Y- permutation via transformation to
LS  ------------------------------------------------------------
-------
```

```r
# Residualized response permutation (RRP)

sw   <-  sqrt(w)
yw   <-  sw * y
intw <-  sw * rep(1, length(y))
zw   <-  sw * z
xw   <-  sw * x

H1 <- lm(yw ~ 0 +intw + zw + xw)
# the test statistic for the data
F0yw <- anova(H1)$`F value`[3]
(if (abs(F0 - F0yw)> eps) number_of_errors <- number_of_errors + 1)
# NULL is OK

# the fitted values and (standardized) residuals under the null hypo
thesis
H0w <- lm(yw~0 + intw + zw)
y0w <- fitted(H0w)
eyw0 <- residuals(H0w)

Fyw <- numeric(K)

for (k in 1:K){
  # Step 2: Generate new data set by unweighted Y-permutation
  permutation_k <- perm.mat[k,] #sample(n)
  pi_ey0 <- eyw0[permutation_k]
  y_pi_w <- y0w + pi_ey0     # equation (6)
  # Step 3: Calculate test statistic Fk
  H1k <- lm(y_pi_w ~ 0 + intw + zw + xw)
  Fyw[k] <- anova(H1k)$F[3]
}
# Step 4: Calculate the Monte Carlo significance level
pval_Y <- sum(c(F0,Fyw) >= F0)  / (K  + 1)
# simpler alternative, used throughout in the sequel:
pval_Y_alt <- (sum(Fyw >= F0) + 1)  / (K  + 1)
(if (abs(pval_Y - pval_Y_alt) > eps) number_of_errors <- number_of_e
rrors + 1) # NULL is OK


if (do_compare_with_function_randperm_RDA){
  # permutation.type = "Y" or "Y2" gives weighted Y-permutation, i.e
. residualized response permutation
  out_RDA <- randperm_RDA(Y= matrix(y),X= matrix(x), Z = matrix(z),
w = w, permutation.type = "Y", nrepet = K, perm.mat= perm.mat)
  pval_Y_RDA <- out_RDA$pval[2]
  (if (abs(pval_Y - pval_Y_RDA) > eps) number_of_errors <- number_of
_errors + 1) # NULL is OK
  Fy_alt <- out_RDA$Fvals[,"Fval"]
  (if (abs(diff(range(Fyw-Fy_alt ))) > eps) number_of_errors <- numb
er_of_errors + 1) # NULL is OK
}


#  Monte Carlo test by Unweighted X- permutation (ordinary Collins-D
```

```
ekker) via WLS ------------------------------------------------------
-----------------
# Residualized predictor permutation (RPP)

N <- length(y)
# the inverse permutations (not needed, but used here for possible c
orrespondence of X and Y permutation, e.g. in unweighted overall nul
l)
perm.mat.inv <- t(apply(perm.mat,1,order))

ex  <- residuals(lm(x~1+z, weights = w))
H1 <- lm(y~z+ex, weights = w)
F0 <- anova(H1)$F[2] # same value as above for Y-permutation

Fx <- numeric(K)

for (k in 1:K){
  # Step 2: Generate new data set by unweighted X-permutation
  permutation_k <- perm.mat.inv[k,] #sample(n)
  pi_ex <- ex[permutation_k]
  # Step 3: Calculate test statistic Fk
  H1 <- lm(y ~ z + pi_ex, weights = w)
  Fx[k] <- anova(H1)$F[2]
}
# Step 4: Calculate the Monte Carlo significance level
pval_X <- (sum(Fx >= F0) + 1)  / (K  + 1)

if (do_compare_with_function_randperm_RDA){
  # permutation.type = "X" or "X1" gives unweighted X-permutation, i
.e. residualized predictor permutation
  out_RDA <- randperm_RDA(Y= matrix(y),X= matrix(x), Z = matrix(z),
w = w, permutation.type = "X", nrepet = K, perm.mat= perm.mat.inv)
  pval_X_RDA <- out_RDA$pval[2]
  (if (abs(pval_X - pval_X_RDA) > eps) number_of_errors <- number_of
_errors + 1) # NULL is OK
  Fx_alt1 <- out_RDA$Fvals[,"Fval"]
  (if (abs(diff(range(Fx-Fx_alt1 ))) > eps) number_of_errors <- numb
er_of_errors + 1) # NULL is OK
}



#  Monte Carlo test by Unweighted X- permutation (ordinary Collins-D
ekker) via transformation to OLS; not necessarily quicker in R------
-----------------------------------------------------------------
# Residualized predictor permutation (RPP)

# we need eyw0 and ex
H0 <- lm(yw~0 + intw + zw)
eyw0 <- residuals(H0)
exw  <- residuals(lm(xw~ 0 + intw + zw))
ex <- exw/sw
H1 <- lm(eyw0 ~ 0 + exw)
df_correction <- (n-3)/(n-1)# it looks like a model with a single pr
```

```
edictor, but there are 2+intercept involved
F0X <- anova(H1)$`F value`[1]*df_correction
F0-F0X
Fx_alt2 <- numeric(K)

for (k in 1:K){
  # Step 2: Generate new data set by unweighted X-permutation
  permutation_k <- perm.mat.inv[k,] #sample(n)
  pi_exw <- ex[permutation_k]*sw
  pi_exw_orth_to_zw <- residuals(lm(pi_exw ~ 0 + intw + zw))
  # Step 3: Calculate test statistic Fk
  H1 <- lm(eyw0 ~ 0 + pi_exw_orth_to_zw )
  Fx_alt2[k] <- anova(H1)$F[1]*df_correction
}
# Step 4: Calculate the Monte Carlo significance level
pval_X2 <- (sum(Fx_alt2 >= F0X) + 1)  / (K  + 1)
(if (abs(pval_X-pval_X2) > eps) number_of_errors <- number_of_errors
+ 1) # NULL is OK
(if (diff(range(Fx_alt2-Fx)) > eps) number_of_errors <- number_of_er
rors + 1) # NULL is OK

print(c(p_value_RRP_w_Y = pval_Y, p_value_RPP_unw_X = pval_X))

cat("number of errors ", number_of_errors, "\n")


#  Monte Carlo test by Unweighted Y- permutation (ordinary Freedman-
Lane) via WLS  ----------------------------------------------------
------------------
# Residualized response permutation (RRP)
H0 <- lm(y~z, weights = w)
y0 <- fitted(H0)
ey0 <- y - y0

H1 <- lm(y ~ z + x, weights = w)
F0 <- anova(H1)$F[2]

Fy_unw <- numeric(K)

for (k in 1:K){
  # Step 2: Generate new data set by unweighted Y-permutation (ordin
ary Freedman-Lane)
  permutation_k <- perm.mat[k,] #sample(n)
  pi_ey0 <- ey0[permutation_k]#
  y_pi <- y0 + pi_ey0     # equation (6)
  # Step 3: Calculate test statistic Fk
  H1 <- lm(y_pi ~ z + x, weights = w)
  Fy_unw[k] <- anova(H1)$F[2]
}
# Step 4: Calculate the Monte Carlo significance level

pval_Y_unw <- (sum(Fy_unw >= F0) + 1)  / (K  + 1)

if (do_compare_with_function_randperm_RDA){
```

```r
  # permutation.type = "Y1"  gives unweighted Y-permutation (ordinar
y Freedman-Lane)
  out_RDA <- randperm_RDA(Y= matrix(y),X= matrix(x), Z = matrix(z),
w = w, permutation.type = "Y1", nrepet = K, perm.mat= perm.mat)
  pval_Y_RDA <- out_RDA$pval[2]
  (if (abs(pval_Y_RDA - pval_Y_unw) > eps) number_of_errors <- numbe
r_of_errors + 1) # NULL is OK
  Fy_unw_alt <- out_RDA$Fvals[,"Fval"]
  (if (abs(diff(range(Fy_unw-Fy_unw_alt ))) > eps) number_of_errors
<- number_of_errors + 1) # NULL is OK
}

summary(out_RDA$parts[,"sstot_eY_perm"]) # note that sstot_eY_perm i
s not constant in unweighted Y-permutation when solved via transform
ation to LS

# Monte Carlo test by Weighted X permutation (standardized Collins-D
ekker) via transformation to OLS ----------------------------------
----------------------------------

# we need eyw0 and ex
H0 <- lm(yw~0 + intw + zw)
eyw0 <- residuals(H0)
exw  <- residuals(lm(xw~ 0 + intw + zw))
H1 <- lm(yw ~ 0 + intw + zw + xw)
F0X <- anova(H1)$`F value`[3]
F0-F0X
Fx_weighted <- numeric(K)

for (k in 1:K){
  # Step 2: Generate new data set
  permutation_k <- perm.mat.inv[k,] #sample(n)
  pi_exw <- exw[permutation_k]
  pi_exw_orth_to_zw <- residuals(lm(pi_exw ~ 0 + intw + zw))
  # Step 3: Calculate test statistic Fk
  H1 <- lm(yw ~ 0 + intw + zw + pi_exw_orth_to_zw )
  Fx_weighted[k] <- anova(H1)$F[3]
}
# Step 4: Calculate the Monte Carlo significance level
pval_X_weigthed <- (sum(Fx_weighted >= F0X) + 1)  / (K  + 1)

if (do_compare_with_function_randperm_RDA){
  # permutation.type = "X2" gives weighted X-permutation (standardiz
ed Collins-Dekker), i.e. residualized response permutation
  out_RDA <- randperm_RDA(Y= matrix(y),X= matrix(x), Z = matrix(z),
w = w, permutation.type = "X2", nrepet = K, perm.mat= perm.mat.inv)
  pval_X_RDA <- out_RDA$pval[2]
  (if (abs(pval_X_weigthed - pval_X_RDA) > eps) number_of_errors <-
number_of_errors + 1) # NULL is OK
  Fx_weighted_alt <- out_RDA$Fvals[,"Fval"]
  (if (abs(diff(range(Fx_weighted-Fx_weighted_alt ))) > eps) number_
of_errors <- number_of_errors + 1) # NULL is OK
}
```

```
cat("number of errors ", number_of_errors, "\n")
}
```

```
## CV_weights
##   4.075729
## number of errors  0
##   p_value_RRP_w_Y p_value_RPP_unw_X
##             0.155             0.395
## number of errors  0
## number of errors  0
```

3. Appendix S3: R-functions for X- and Y-permutation using Freedman-Lane and Collins-Dekker

3.1 residualXY_permutation.r

This files contains, in this order, the functions `randperm_RDA`, `test_axes_RDA`, four functions called by `randperm_RDA` and four functions for efficiently obtaining fitted values and residuals in unweighted least-squares (LS) problems using matrix operations instead of using `fitted` and `residuals` based on the `lm` function.

The function `randperm_RDA` performs the four methods of permutation (weighted and unweigthed X- and Y-permutation for matrices **Y**, **X** and **Z** (which can be NULL) and a vector of weights **w**, with notation as in the main text. The `permutation.type` argument determines which permutation method is carried out. The value is a list which include *P*-values, *F*-statistics of the data and permutations and all parts of the fit due to the intercept, **Z** and **X** for each simulated data set. The function first transforms the weighted least-squares problem to an OLS one so that the permutation can be performed with a minimum usage of the weights. After transformation to OLS, it is the permutation type which determines which function is called to carry out a particular version of the four available (`randperm_eX0sqrtw` for ordinary (or unweighted) X-permutation "X" or "X1", `randperm_eY0_sqrtw` for ordinary (unweighted) Y-permutation "Y1", `randperm_eX0` for standardized (or weighted) X-permutation "X2" and `randperm_eY0` for standardized (or weighted) Y-permutation "Y" or "Y2"). Note that the standardized versions of the methods do not need the weights as an argument, whereas the ordinary versions do require the weight as the residual after transformation must be backtransformed to unweighted before the original residual can be permuted. This reflects the ratios of weights that appear in the main text.

The function `test_axes_RDA` tests the significance of four consecutive axes of RDA. It calls `randperm_RDA`. The `permutation.type` argument determines which permutation method is carried out.

The four functions for efficiently obtaining fitted values and residuals in unweighted least-squares (LS) problems using matrix operations are

`unweighted_lm_pq_fit(Y,X)` – fitted values of a multivariate multiple regression of Y on X using qr-decomposition (`qr.fitted`); equivalent of `fitted(lm(Y~0+X))`

`unweighted_lm_Orthnorm_fit(Y,X)` – fitted values of a multivariate multiple regression of Y on X for orthonormal X only; equivalent of `fitted(lm(Y~0+X))` for orthonormal X.

`unweighted_lm_Orthnorm(Y,X)` – residuals of a multivariate multiple regression of Y on X for orthonormal X only; equivalent of `residuals(lm(Y~0+X))` for orthonormal X.

`SVD(X)` - returns orthonormal X by using a singular value decomposition (`svd`)

```
# User functions for testing by weighted RDA ----------------------
-----------------------------------

randperm_RDA <- function(Y,X, Z = NULL, w = rep(1,nrow(Y)), permutat
ion.type = "X", nrepet = 999, perm.mat= NULL, V1t = NULL){
  # Y = response, X = predictors of interest,
  # Z = nuisance variable aka covariables, covariates or conditionin
g variable
```

```
  # (standardized and ordinary) residual Y and X permutation (for RDA)
  #         for permutation.type = "X1"=="X","X2" "Y1", "Y2"="Y"
  #
  # X/Y 1: ordinary;        permutation of the (unstandardized) residual.
  # X/Y 2: standardized;    permutation of the standardized residual;

  # Y == Y2; standardized Y-permutation,   aka residualized response permutation (Freedman-Lane for w = 1)
  # X == X1; ordinary X-permutation, aka residualized predictor permutation (Collins-Dekker for w = 1)
  # residualization with respect to the nuisance variable (covariates/covariables) Z
  #
  # Permutation in all methods is carried out
  #   after transformation to least-squares (LS)
  #   by multiplication by sqrt of w
  #
  # this version of the function gives
  #   both the correct p value and the pval of Canoco 3-5.12
  # For ordinary analysis, there is no difference between the two.
  # see function randperm_eY0
  #           for the difference in definition of the two p-values
  #
  Wn <- w / sum(w)
  sWn <- sqrt(Wn)
  Yw <- Y*sWn
  Xw <- X*sWn
  # add the intercept for the analysis
  #   which is NOT one or constant for unequal w
  Zw <- cbind(matrix(sWn, nrow = nrow(Y), ncol=1),Z*sWn)
  # Y =Yw; X = Xw; Z = Zw
  if (permutation.type %in% c("Y","Y2")) {
    # standardized Y-permutation, aka
    # residualized response permutation:
    #   standardized WLS Y-residuals wrt Z (i.e. the residuals of Y after transformation to OLS)
    #      are permuted (and regressed again transformed X-residuals with Z)
    #    Freedman & Lane with permutation of reweighted WLS-residuals of null model ==
    #    Freedman & Lane with permutation of null-model residuals after transformation to ordinary least-squares (OLS),i.e.
    # step 1: transform to an OLS problem
    # step 2: carry out Freedman & Lane in the transformed LS problem i.e. using OLS
    # This is equal to
    # step 1W:  get residuals of the null model Y~Z in the WLS problem using weights w
    # step 2W:  permute the residuals of step 1W and reweight them (1/sqrt(w))
    # step 3W   carry out the Freedman & Lane using WLS using weight
```

```
s w
    # the first version is how the method is implemented here for co
mputational speed
    # the second version gives the method its name "standardized Fre
edman-Lane"; Y-residuals are standardized in the WLS problem
    result <- randperm_eY0(Yw,Xw, Zw, nrepet = nrepet, perm.mat= per
m.mat)
  } else if (permutation.type == "YsqrtLev") {
    result <- randperm_eY1Z_leverage(Yw,Xw, Zw, nrepet = nrepet, per
m.mat= perm.mat)
  } else if (permutation.type == "Y1") {
    # ordinary Y-permutation
    #                         i.e. unstandardized WLS Y-residuals are
permuted
    # step 1: transform to an OLS problem (premultiply Y,X,Z with sq
rt(w))
    # step 2: get residuals of the null model Y~Z in the OLS problem
    # step 3: permute the residuals of step 2 and reweight them (1/s
qrt(w))
    # step 2: carry out Freedman & Lane on the reweighted residuals
    # This is equal to
    # step 1W:  get residuals of the null model Y~Z in the WLS probl
em using weights w
    # step 2W:  permute the residuals of step 1W (without any weight
ing)
    # step 3W   carry out the Freedman & Lane using WLS using weight
s w
    # the first version is how the method is implemented here for co
mputational speed
    # the second version gives the method its name "ordinary Freedma
n-Lane"; ; Y-residuals are unweighted in the WLS problem
    result <- randperm_eY0sqrtw(Yw,Xw, Zw, sWn = sWn, nrepet = nrepe
t, perm.mat= perm.mat)
    result$ssX <-  c(result$ssX,  result$ssX/(result$sstot-result$ss
Int),result$ssX/(result$sstot-result$ssInt-result$ssZ))
  } else if (permutation.type == "X2") {
    # standardized X-permutation
    #                         i.e. standardized X-residuals are permu
ted
    # step 1: transform to an OLS problem
    # step 2: carry out Collins-Dekker in the transformed LS problem
    # This is equal to
    # step 1W:  get residuals of the model X ~ Z in the WLS problem
    # step 2W:  permute the residuals of step 1W and reweight them (
1/sqrt(w))
    # step 3W   carry out the Collins-Dekker using WLS
    # the first version is how the method is implemented here for co
mputational speed
    # the second version gives the method its name "standardized Col
lins-Dekker";  X-residuals are standardized in the WLS problem

    result <- randperm_eX0(Yw,Xw, Zw, nrepet = nrepet, perm.mat= per
m.mat)
  } else if (permutation.type %in% c("X", "X1")){
```

```
    # ordinary X-permutation
    # residualized predictor permutation:
    #                         i.e. unstandardized X-residuals are per
muted
    # ordinary Y-permutation
    #                         i.e. unstandardized Y-residuals are per
muted
    # step 1: transform to an OLS problem
    # step 2: get residuals of the model X ~ Z (after the transforma
tion in step 1) in the OLS problem
    # step 3: permute the LS residuals of step 2 and reweight them
    # step 2: carry out Collins-Dekker using the reweighted residual
s of step 3
    # This is equal to
    # step 1W:  get residuals of the model X ~ Z in the WLS problem
    # step 2W:  permute the residuals of step 1W (without any weight
ing)
    # step 3W   carry out the Freedman & Lane using WLS
    # the first version is how the method is implemented here for co
mputational speed
    # the second version gives the method its name "ordinary Collins
-Dekker";  X-residuals are unweighted in the WLS problem

    result <- randperm_eX0sqrtw(Yw,Xw, Zw, sWn = sWn, nrepet = nrepe
t, perm.mat= perm.mat)
    result$ssX <-  c(result$ssX,  result$ssX/result$ssGz)
  } else {errorCondition("permutation.type misspecified")}
  result$method <- "RDA"
  result$rowweights <- Wn
  result$RDA_ax1 <- result$EigVector1/sWn

  return(result)
}

test_axes_RDA <- function(Y,X, Z = NULL, w = rep(1,nrow(Y)), permuta
tion.type = "X", nrepet = 999, perm.mat= NULL, silent = FALSE){
  mysvd <- svd
  # perm.mat is a list for 4 perm.mat matrices
  if (is.null(perm.mat)){perm.mat <- list(a=NULL, b= NULL, c=NULL, d
= NULL)}
  if (!is.list(perm.mat)) errorCondition("perm.mat in test_axes_RDA
must be a list of matrices")
  out_tes2 <- list(ssX= NULL, ssGz =NULL, F0=NULL,pval= NULL)
  out_tes3 <- list(ssX= NULL, ssGz =NULL, F0=NULL,pval= NULL)
  out_tes4 <- list(ssX= NULL, ssGz =NULL, F0=NULL,pval= NULL)
  out_tes1 <- randperm_RDA(Y,X, Z, w=w, nrepet = nrow(perm.mat[[1]])
, permutation.type = permutation.type, perm.mat= perm.mat[[1]])
  if (out_tes1$rank >= 2) out_tes2 <- randperm_RDA(Y,X, Z = cbind(Z,
out_tes1$RDA_ax1), w=w, nrepet = nrow(perm.mat[[2]]), permutation.ty
pe = permutation.type, perm.mat= perm.mat[[2]])
  if (out_tes1$rank >= 3) out_tes3 <- randperm_RDA(Y,X, Z = cbind(Z,
out_tes1$RDA_ax1,out_tes2$RDA_ax1), w=w, nrepet = nrow(perm.mat[[3]]
), permutation.type = permutation.type, perm.mat= perm.mat[[3]])
  if (out_tes1$rank >= 4) out_tes4 <- randperm_RDA(Y,X, Z = cbind(Z,
```

```
out_tes1$RDA_ax1,out_tes2$RDA_ax1,out_tes3$RDA_ax1), w=w, nrepet = n
row(perm.mat[[4]]), permutation.type = permutation.type, perm.mat= p
erm.mat[[4]])
  ExplConstax <- c(out_tes1$ssX[1]/out_tes1$ssGz,out_tes2$ssX[1]/out
_tes1$ssGz,
                   out_tes3$ssX[1]/out_tes1$ssGz,out_tes4$ssX[1]/out
_tes1$ssGz)
  F_axes <- c(out_tes1$F0[1],out_tes2$F0[1],out_tes3$F0[1],out_tes4$
F0[1])
  p_val_axes1 <- cummax(c(out_tes1$pval[1],out_tes2$pval[1],out_tes3
$pval[1],out_tes4$pval[1]))
  p_val_axes2 <- cummax(c(out_tes1$pval[2],out_tes2$pval[2],out_tes3
$pval[2],out_tes4$pval[2]))
  eig <- out_tes1$eig
  axsig_RDA <- rbind(ExplConstax,F_axes,p_val_axes1,p_val_axes2)
  rownames(axsig_RDA) <- c("Explained by constrained axis","Pseudo-F
value", "P value eig1", "P value trace")

  colnames(axsig_RDA) <- seq_along(F_axes)
  attr(axsig_RDA, "method") <- out_tes1$method
  attr(axsig_RDA, "permutation.type") <- permutation.type
  if (!silent) print(round (axsig_RDA,3))
  result <- list(summary = axsig_RDA, ax1 =out_tes1, ax2= out_tes2,
ax3= out_tes3, ax4= out_tes4, eig = eig)
}

# functions called by randperm_RDA ------------------------------

#permutation.type = Y = Y2
randperm_eY0 <- function(Y,X, Z = matrix(1, nrow = nrow(Y),ncol =1),
nrepet = 999, perm.mat= NULL){
  # for getting standardized Y-permutation, i.e. standardize Freedma
n-Lane (residualized response permutation),
  #    permuting the standardized Y-residuals of the original weight
ed analysis,
  #    in the calling function
  # Y X Z should have been transformed to LS in the calling function
by
  #  by multiplication by sWn = sqrt(w/sum(w)) inc. intercept!
  #  so that this function does not need weights.
  #
  # Note that the first column of Z (Z_intercept) is
  #         1 in an ordinary analysis and
  #   sqrt(w) in a    standardized analysis.
  #
  # This version gives both the correct p value and
  #                                  the p value of Canoco 3-5.12 (pv
al_Can3)
  # The difference between the two lies in the calculation of the re
sidual sum of squares of the permuted residuals
  # pval (Correct)     :: rss_perm <-   sstot_eY                - ss
Z_perm - ssX_perm
  # pval_Can3          :: rss_perm <-   sstot_eY - ss_int_perm  - ss
Z_perm - ssX_perm
```

```
  # By consequence, pval_Can3 is equal to or lower than that of stan
dardized Y-permutation,
  #                   yielding somewhat higher rejection rates in si
mulations.
  # For ordinary analysis, ss_int_perm = 0,
  #                           so that there is no difference between
the two p-values.
  #
  # preparations and data value

  N <- nrow(Y)
  if (is.null(perm.mat)){
    perm.mat <- matrix(0,nrow = nrepet, ncol = N)
    for (i in 1:nrepet){
      perm.mat[i,] <- sample(N)
    }
  }
  sstot <- sum(Y^2)
  # separate first column from the remainder of Z.
  #
  Z_intercept<- Z[,1,drop = FALSE]
  # make the remainder of Z orthogonal to the first column and ortho
gonalize
  if (ncol(Z)>1) {
    Z_orth <- SVD(unweighted_lm_Orthnorm(Z[,-1,drop = FALSE], Z_inte
rcept))
  } else {Z_orth <- matrix(0,nrow = N, ncol=1)}
  Yfit00 <- unweighted_lm_Orthnorm_fit(Y,Z_intercept)
  Yfit_Z <- unweighted_lm_Orthnorm_fit(Y,Z_orth)
  ssInt <- sum(Yfit00^2)
  Yfit_IntwZ <- Yfit_Z + Yfit00
  ssZ <- sum(Yfit_Z^2)
  # residuals or errors of Y from Y~Z under null model
  eY <- Y - Yfit_IntwZ
  # residuals or errors of X from X~Z
  eX <- unweighted_lm_Orthnorm(X, cbind(Z_intercept, Z_orth))
  # step 2: ss(X)
  eX <- SVD(eX)
  Yfit <- unweighted_lm_Orthnorm_fit(eY,eX)
  ssX <- sum(Yfit^2)
  svd_Yfit  <- mysvd(Yfit)
  ssX_eig1 <- svd_Yfit$d[1]^2
  rank <- sum(svd_Yfit$d/sum(svd_Yfit$d) > 1.e-3)
  EigVector1 <- svd_Yfit$u[,1,drop = FALSE]
  df_cor <- (N - ncol(Z)- ncol(eX))/ncol(eX)  #(N-nz-nx-1)/nx
  # residual sum of squares under null model ==
  #                           total sum of squares in permutation of
eY
  sstot_eY <- sstot - ssInt - ssZ
  F0 <- ssX / (sstot_eY - ssX) * df_cor
  F0_eig1 <- ssX_eig1 / (sstot_eY -  ssX_eig1) * df_cor
  # end preparations and data value
  #
  Fval <- numeric(nrepet)
```

```
  ss_int_perm <- numeric(nrepet)
  ssZ_perm <- numeric(nrepet)
  ssX_perm <- numeric(nrepet)
  ssX_eig1_perm <- numeric(nrepet)
  #
  for (i in seq_len(nrepet)){
    # permute residual Y
    i_perm  <- perm.mat[i,]
    eY_perm <-    eY[i_perm,, drop = FALSE]
    eYfit_permInt <- unweighted_lm_Orthnorm_fit(eY_perm,Z_intercept)
    eYfit_permZ <- unweighted_lm_Orthnorm_fit(eY_perm,Z_orth)
    # note that eX and Z stay orthogonal
    eYfit_permX <- unweighted_lm_Orthnorm_fit(eY_perm,eX)
    ss_int_perm[i] <- sum(eYfit_permInt^2)
    ssZ_perm[i] <- sum(eYfit_permZ^2)
    ssX_perm[i] <- sum(eYfit_permX^2)
    ssX_eig1_perm[i] <- mysvd(eYfit_permX)$d[1]^2
  }
  rss_perm <-    sstot_eY - ss_int_perm  - ssZ_perm - ssX_perm
  rss_eig1_perm <-   sstot_eY - ss_int_perm  - ssZ_perm - ssX_eig1_p
erm
  # Canoco 3.0 - 5.12 version:
  rss_perm3 <-   sstot_eY               - ssZ_perm - ssX_perm
  Fval <- ssX_perm/rss_perm * df_cor
  Fval3 <- ssX_perm/rss_perm3 * df_cor
  Fval_eig1 <- ssX_eig1_perm/rss_eig1_perm * df_cor
  isna.r <-  sum(is.na(Fval))
  isna3.r <-  sum(is.na(Fval3))
  pval <- (sum(Fval >= F0, na.rm = TRUE) + 1)  / (nrepet- isna.r  +
1)
  pval_eig1 <- (sum(Fval_eig1 >= F0_eig1, na.rm = TRUE) + 1)  / (nre
pet- isna.r  + 1)
  pval3 <- (sum(Fval3 >= F0, na.rm = TRUE) + 1)  / (nrepet- isna3.r
+ 1)
  parts <- cbind(Fval, sstot_perm=sstot_eY, ss_int_perm, rss_perm,
                 ssX_perm, ssZ_perm, ssX_eig1_perm, rss_eig1_perm,Fv
al_eig1)
  return(list(pval = c(pval_eig1= pval_eig1, pval = pval,pval_Can3=p
val3),
              Fvals = cbind(Fval_eig1,Fval, Fval3), F0 = c(F0_eig1,F
0),
              eX= eX, ssX = c(eig1 = ssX_eig1,trace = ssX),ssGz = ss
tot_eY,
              ssZ = ssZ, ssInt=ssInt, sstot = sstot, rank = rank, ei
g = svd_Yfit$d^2,
              parts = parts,EigVector1 = EigVector1))
}

#for permutation.type = X = X1
randperm_eX0sqrtw <- function(Y,X, Z = matrix(1, nrow = nrow(Y),ncol
=1), sWn = rep(1,nrow(Y)), nrepet = 999, perm.mat= NULL){

  # for getting ordinary X-permutation, i.e. ordinary Collins-Dekker
(residualized predictor permutation),
```

```
  #    permuting the X-residuals of the original weighted analysis,
  #    in the calling function
  # Y X Z should have been transformed to LS in the calling function
by
  #  sWn = sqrt(w/sum(w)) of the calling function
  #  so that this function does not need weights.
  #  except for calculating the original residual X given Z
  #
  # preparations and data value
  N <- nrow(Y)
  if (is.null(perm.mat)){
    perm.mat <- matrix(0,nrow = nrepet, ncol = N)
    for (i in 1:nrepet){
      perm.mat[i,] <- sample(N)
    }
  }
  Z_orth <- SVD(Z)
  # Y-residuals from Y~Z under null model
  eY <- unweighted_lm_Orthnorm(Y, Z_orth)
  # orthogonalize X with respect to Z giving eX
  # eX = X_orth = residuals of X from X~Z
  eX <- unweighted_lm_Orthnorm(X, Z_orth)
  eXw <- eX/sWn # the X-residuals of the original weighted analysis
  # step 2: ss(X)
  Yfit_X <- unweighted_lm_pq_fit(eY,eX)
  ssX <- sum(Yfit_X^2)
  svd_Yfit_X <- mysvd(Yfit_X)
  ssX_eig1 <- svd_Yfit_X$d[1]^2
  EigVector1 <- svd_Yfit_X$u[, 1, drop= FALSE]
  rank <- sum(svd_Yfit_X$d/sum(svd_Yfit_X$d) > 1.e-3)
  sstot <- sum(eY^2)
  p_eX <- qr(eX)$rank
  df_cor <- (nrow(eY)- ncol(Z_orth)- p_eX)/p_eX  #(N-nz-nx-1)/nx
  F0 <- ssX / (sstot - ssX) * df_cor
  F0_eig1 <- ssX_eig1 / (sstot -  ssX_eig1) * df_cor
  # end preparations and data value
  ssX_perm <- numeric(nrepet)
  ssX_eig1_perm <- numeric(nrepet)
  #i=1
  #
  for (i in seq_len(nrepet)){
    # permute residual eX
    i_perm  <- perm.mat[i,]
    eX_perm <- eXw[i_perm, , drop = FALSE]*sWn
    # orthogonalize permuted orthogonaled-X
    #  (i.e. eX which is permuted to give eX_perm) with respect to Z
    #  giving eX_perm_orth_to_Z
    # residuals of X_orth_perm on ~Z :
    eX_perm_orth_to_Z <- unweighted_lm_Orthnorm(eX_perm, Z_orth)
    # regress the Y-residuals w.r.t. Z (i.e. eY) on eX_perm_orth_to_
Z
    # to determine the sum of squares due to eX_perm in the regressi
on lm(Y~ Z + eX_perm), i.e.
    # for one response variable:
```

```
    # my_lm <- lm(Y~ Z + eX_perm, weights = Wn); anova(my_lm)['eX_pe
rm',2]
    Yfit_permX <- unweighted_lm_pq_fit(eY,eX_perm_orth_to_Z) #
    ssX_perm[i] <- sum(Yfit_permX^2)
    ssX_eig1_perm[i] <- mysvd(Yfit_permX)$d[1]^2
  }
  rss_perm <- sstot-ssX_perm
  rss_eig1_perm <-   sstot - ssX_eig1_perm
  Fval <- ssX_perm /rss_perm * df_cor
  Fval_eig1 <- ssX_eig1_perm/rss_eig1_perm * df_cor
  isna.r <-  sum(is.na(Fval))
  pval <- (sum(Fval >= F0, na.rm = TRUE) + 1)  / (nrepet- isna.r  +
1)
  pval_eig1 <- (sum(Fval_eig1 >= F0_eig1, na.rm = TRUE) + 1)  / (nre
pet- isna.r  + 1)
  parts <- cbind(Fval, rss_perm, ssX_perm, ssX_eig1_perm, rss_eig1_p
erm,
                 Fval_eig1)
  return(list(pval = c(pval_eig1= pval_eig1, pval = pval),
              Fvals = cbind(Fval_eig1,Fval), F0 = c(F0_eig1,F0),
              ssX = c(eig1 = ssX_eig1,trace = ssX),RSS = c(sstot-ssX
,sstot-ssX_eig1),
              ssGz = sstot, rank = rank, eig = svd_Yfit_X$d^2,
              parts = parts,EigVector1 = EigVector1))

}


# for permutation.type = X2
randperm_eX0 <- function(Y,X, Z = matrix(1, nrow = nrow(Y),ncol =1),
nrepet = 999, perm.mat= NULL){
  # ordinary X-permutation to result in standardized  X-permutation,
i.e. standardized Collins-Dekker
  #   in the calling function
  #
  # preparations and data value
  N <- nrow(Y)
  if (is.null(perm.mat)){
    perm.mat <- matrix(0,nrow = nrepet, ncol = N)
    for (i in 1:nrepet){
      perm.mat[i,] <- sample(N)
    }
  }
  Z_orth <- SVD(Z)
  # residuals or errors of Y from Y~Z under null model
  eY <- unweighted_lm_Orthnorm(Y, Z_orth)
  # orthogonalize X with respect to Z giving eX
  # eX = X_orth = residuals of X from X~Z
  eX <- unweighted_lm_Orthnorm(X, Z_orth)
  # step 2: ss(X)
  Yfit_X <- unweighted_lm_pq_fit(eY,eX)
  ssX <- sum(Yfit_X^2)
  #ssX_eig1 <- mysvd(Yfit_X)$d[1]^2
  svd_Yfit_X <- mysvd(Yfit_X)
```

```
  ssX_eig1 <- svd_Yfit_X$d[1]^2
  EigVector1 <- svd_Yfit_X$u[, 1, drop= FALSE]
  rank <- sum(svd_Yfit_X$d/sum(svd_Yfit_X$d) > 1.e-3)
  sstot <- sum(eY^2)
  df_cor <- (nrow(eY)- ncol(Z_orth)- ncol(eX))/ncol(eX)  #(N-nz-nx-1
)/nx
  F0 <- ssX / (sstot - ssX) * df_cor
  F0_eig1 <- ssX_eig1 / (sstot -  ssX_eig1) * df_cor
  # end preparations and data value
  ssX_perm <- numeric(nrepet)
  ssX_eig1_perm <- numeric(nrepet)
  #i=1
  #
  for (i in seq_len(nrepet)){
    # permute residual eX
    i_perm  <- perm.mat[i,]
    eX_perm <- eX[i_perm, , drop = FALSE]
    # orthogonalize permuted orthogonaled-X
    #   (i.e. eX which is permuted to give eX_perm) with respect to
Z
    # giving eX_perm_orth_to_Z, i.e.
    # residuals of X_orth_perm on ~Z:
    eX_perm_orth_to_Z <- unweighted_lm_Orthnorm(eX_perm, Z_orth)
    # regress the Y-residuals w.r.t. Z (i.e. eY) on eX_perm_orth_to_
Z
    # to determine the sum of squares due to eX_perm
    # in the regression lm(Y~ Z + eX_perm), i.e.
    # for one response variable:
    # my_lm <- lm(Y~ Z + eX_perm, weights = Wn); anova(my_lm)['eX_pe
rm',2]
    Yfit_permX <- unweighted_lm_pq_fit(eY,eX_perm_orth_to_Z) #
    ssX_perm[i] <- sum(Yfit_permX^2)
    ssX_eig1_perm[i] <- mysvd(Yfit_permX)$d[1]^2
  }
  rss_perm <- sstot-ssX_perm
  rss_eig1_perm <-   sstot - ssX_eig1_perm
  Fval <- ssX_perm /rss_perm * df_cor
  Fval_eig1 <- ssX_eig1_perm/rss_eig1_perm * df_cor
  isna.r <-  sum(is.na(Fval))
  pval <- (sum(Fval >= F0, na.rm = TRUE) + 1)  / (nrepet- isna.r  +
1)
  pval_eig1 <- (sum(Fval_eig1 >= F0_eig1, na.rm = TRUE) + 1)  / (nre
pet- isna.r  + 1)
  parts <- cbind(Fval, ssX_perm, rss_perm, ssX_eig1_perm, rss_eig1_p
erm)
  return(list(pval = c(pval_eig1= pval_eig1, pval = pval),
              Fvals = cbind(Fval_eig1,Fval), F0 = c(F0_eig1,F0),
              ssX = c(eig1 =ssX_eig1,trace = ssX),RSS = c(sstot-ssX,
sstot-ssX_eig1), rank = rank, eig = svd_Yfit_X$d^2,
              ssGz = sstot,
              parts = parts,EigVector1 = EigVector1))
}

# for permutation.type = Y1
```

```
randperm_eY0sqrtw <- function(Y,X, Z = matrix(1, nrow = nrow(Y),ncol
=1), sWn = rep(1,nrow(Y)), nrepet = 999, perm.mat= NULL){
  # for getting ordinary Y-permutation,
  #      permuting with unstandardized residuals,
  #      in the calling function. To get ordinary Freedman-Lane in t
he calling function.
  # Y X Z should have been transformed in the calling function, sWn
= sqrt(w/sum(w)) of the calling function
  # called by randperm_eY for RDA where the weights are set
  # this version gives both the correct p value and the pval of Cano
co 3-5.12 (pval_Can3)
  # The difference between the two lies in the calculation of the re
sidual sum of squares of the permuted residuals
  # pval (Correct)     :: rss_perm <-   sstot_eY                - ss
Z_perm - ssX_perm
  # pval_Can3          :: rss_perm <-   sstot_eY - ss_int_perm  - ss
Z_perm - ssX_perm
  # For ordinary analysis, there is no difference between the two.
  # Note that the first column of Z (Z_intercept) is       1 in an u
nweighted analysis and
  #                                                  sqrt(w) in a
weighted analysis.
  #
  # preparations and data value
  N <- nrow(Y)
  if (is.null(perm.mat)){
    perm.mat <- matrix(0,nrow = nrepet, ncol = N)
    for (i in 1:nrepet){
      perm.mat[i,] <- sample(N)
    }
  }
  sstot <- sum(Y^2)
  # separate first column from the remainder of Z.
  #
  Z_intercept<- Z[,1,drop = FALSE]
  # make the remainder of Z orthogonal to the first colum and orthog
onalize
  if (ncol(Z)>1) {
    Z_orth <- SVD(unweighted_lm_Orthnorm(Z[,-1,drop = FALSE], Z_inte
rcept))
  } else {Z_orth <- matrix(0,nrow = N, ncol=1)}
  Yfit00 <- unweighted_lm_Orthnorm_fit(Y,Z_intercept)
  Yfit_Z <- unweighted_lm_Orthnorm_fit(Y,Z_orth)
  ssInt <- sum(Yfit00^2)
  Yfit_IntwZ <- Yfit_Z + Yfit00
  ssZ <- sum(Yfit_Z^2)
  # residuals or errors of Y from Y~Z under null model
  eY <- Y - Yfit_IntwZ
  eYw <- eY/sWn
  # residuals or errors of X from X~Z (X-orth)
  eX <- unweighted_lm_Orthnorm(X, cbind(Z_intercept, Z_orth))
  # step 2: ss(X)
  eX <- SVD(eX)
  Yfit <- unweighted_lm_Orthnorm_fit(eY,eX)
```

```
  svd_Yfit_X <- mysvd(Yfit)
  ssX_eig1 <- svd_Yfit_X$d[1]^2
  EigVector1 <- svd_Yfit_X$u[, 1, drop= FALSE]
  rank <- sum(svd_Yfit_X$d/sum(svd_Yfit_X$d) > 1.e-3)
  ssX <- sum(Yfit^2)
  df_cor <- (N - ncol(Z)- ncol(eX))/ncol(eX)  #(N-nz-nx-1)/nx
  # residual sum of squares under null model
  #       == total sum of squares in permuation of eY
  sstot_eY <- sstot - ssInt - ssZ
  F0 <- ssX / (sstot_eY - ssX) * df_cor
  F0_eig1 <- ssX_eig1 / (sstot_eY -  ssX_eig1) * df_cor
  # end preparations and data value
  Fval <- numeric(nrepet)
  #i=719
  sstot_eY_perm <- numeric(nrepet)
  ss_int_perm <- numeric(nrepet)
  ssZ_perm <- numeric(nrepet)
  ssX_perm <- numeric(nrepet)
  ssX_eig1_perm <- numeric(nrepet)
  #
  for (i in seq_len(nrepet)){
    # permute residual Y
    i_perm  <- perm.mat[i,]
    eY_perm <-    eYw[i_perm,, drop = FALSE] * sWn
    eYfit_permInt <- unweighted_lm_Orthnorm_fit(eY_perm,Z_intercept)
    eYfit_permZ <- unweighted_lm_Orthnorm_fit(eY_perm,Z_orth)
    # eX and Z stay orthogonal
    eYfit_permX <- unweighted_lm_Orthnorm_fit(eY_perm,eX)
    sstot_eY_perm[i] <- sum(eY_perm^2)
    ss_int_perm[i] <- sum(eYfit_permInt^2)
    ssZ_perm[i] <- sum(eYfit_permZ^2)
    ssX_perm[i] <- sum(eYfit_permX^2)
    ssX_eig1_perm[i] <- mysvd(eYfit_permX)$d[1]^2
  }
  rss_perm <-   sstot_eY_perm - ss_int_perm  - ssZ_perm - ssX_perm
  rss_eig1_perm <-   sstot_eY_perm - ss_int_perm  - ssZ_perm - ssX_e
ig1_perm
  rss_perm3 <-   sstot_eY_perm               - ssZ_perm - ssX_perm
  Fval <- ssX_perm/rss_perm * df_cor
  Fval3 <- ssX_perm/rss_perm3 * df_cor
  Fval_eig1 <- ssX_eig1_perm/rss_eig1_perm * df_cor
  isna.r <-  sum(is.na(Fval))
  isna3.r <-  sum(is.na(Fval3))
  pval <- (sum(Fval >= F0, na.rm = TRUE) + 1)  / (nrepet- isna.r  +
1)
  pval_eig1 <- (sum(Fval_eig1 >= F0_eig1, na.rm = TRUE) + 1)  / (nre
pet- isna.r  + 1)
  pval3 <- (sum(Fval3 >= F0, na.rm = TRUE) + 1)  / (nrepet- isna3.r
+ 1)
  parts <- cbind(Fval, sstot_eY_perm, ss_int_perm, ssX_perm, ssZ_per
m, rss_perm)
  return(list(
              pval = c(pval_eig1= pval_eig1, pval = pval,pval_Can3=p
val3),
```

```
              Fvals = cbind(Fval_eig1,Fval), F0 = c(F0_eig1,F0), F0
= F0, eX= eX, ssZ = ssZ,
              ssInt=ssInt, sstot = sstot, rank = rank, eig = svd_Yfi
t_X$d^2, ssX = c(eig1 = ssX_eig1,trace = ssX),ssGz = sstot_eY,
              parts = parts,EigVector1 = EigVector1))
}


# function dummysvd is to avoid calculation of the eigen vector/
#                                                 eigen value test
statistic
#           set: mysvd <- dummysvd # with eigenvalue test statistic
# otherwise set  mysvd <- svd      # witout eigenvalue test statisti
c
dummysvd <- function(Y){list(d = 1,
                             U = matrix(1,nrow=nrow(Y), ncol=1),
                             V = matrix(1,nrow=ncol(Y), ncol=1) )}


# unweighted least-squares (OLS) functions ------------------------
------------------------

unweighted_lm_pq_fit <- function(Y, X){
  # multivariate multiple regression of Y on X
  # using qr decomposition
  # value Y_fit
  Y_fit <- qr.fitted( qr(X),Y)
  return(Y_fit)
}

unweighted_lm_Orthnorm_fit <- function(Y, X=numeric(0)){
  # multivariate multiple regression of Y on orthonormal X
  # value Y_residual
  beta <-  t(X) %*% Y
  Yfit <- X %*% beta
  return(Yfit)
}

unweighted_lm_Orthnorm <- function(Y, X=numeric(0)){
  # multivariate multiple regression of Y on orthonormal X
  # value Y_residual
  beta <-  t(X) %*% Y
  Y <- Y - X %*% beta
  return(Y)
}



SVD <- function(Y){
  svdY <- svd(Y)
  Ustar <- svdY$u
  id <- which(svdY$d > 1.e-6)
  return(Ustar[,id, drop = FALSE])
}
```

```
# end OLS versions ------------------------------------------------
```

3.2 Test_4XYmethods_functions

This file contains the functions `testXgivenZ4_with_OLS` and `tests_NaXis`, which are the core functions used in Section 4.2 and 4.3 of the main text and Appendix S6 and Appendix S7, respectively.

The function `testXgivenZ4_with_OLS` calls `randperm_RDA` for each of the four methods of permutation via the `permutation.type` argument and results in the *P*-value given by each method. It generates *K* (`nrepet`) random permutations that are then used in both versions of Y-permutation; the inverse of each permutation is used in both versions of X-permutation, so as to give a maximum similarity between X- and Y-permutation, also for small values of *K*. See Appendix S6 for an example of its use.

The function `tests_NaXis` calls `test_axes_RDA` for unweighted X- and weighted Y-permutation via the `permutation.type` argument of `test_axes_RDA`. See Appendix S7 for an example of its use.

```
testXgivenZ4_with_OLS <- function(Y,X, Z=NULL, w= 1:nrow(Y), nrepet
= 999){
  # testing the effects of X on Y given Z by 6 different methods:
  # Y & X2 (weighted methods) and Y1 & X (unweigthed) permutation me
thods, corresponding to
  # standardized Freedman-Lane and Collins-Dekker  and ordinary Free
dman-Lane and Collins-Dekker
  # OLS-analyses using Freedman-Lane and Collins-Dekker are included
. This makes the difference
  # with the next function testXgivenZ4 which does WLS only.
  mysvd <- dummysvd # no eig1 tests are needed (avoid svd)
  N <- nrow(Y)
  perm.mat <- matrix(0,nrow = nrepet, ncol = N)
  for (i in 1:nrepet){
    perm.mat[i,] <- sample(N)
  }
  w1 <- rep(1, N)
  pval_Y_RDA <- randperm_RDA(Y,X, Z = Z, w = w, permutation.type = "
Y", nrepet = nrepet, perm.mat= perm.mat)$pval[2:3]
  pval_Y1_RDA <- randperm_RDA(Y,X, Z = Z, w = w, permutation.type =
"Y1", nrepet = nrepet, perm.mat= perm.mat)$pval[2]
  pval_Y_RDA_OLS <- randperm_RDA(Y,X, Z = Z, w = w1, permutation.typ
e = "Y", nrepet = nrepet, perm.mat= perm.mat)$pval[2]

  perm.mat <- t(apply(perm.mat,1,order))
  pval_X_RDA <- randperm_RDA(Y,X, Z = Z, w = w, permutation.type = "
X", nrepet = nrepet, perm.mat= perm.mat)$pval[2]
  pval_X2_RDA <- randperm_RDA(Y,X, Z = Z, w = w, permutation.type =
"X2", nrepet = nrepet, perm.mat= perm.mat)$pval[2]
  pval_X_RDA_OLS <- randperm_RDA(Y,X, Z = Z, w = w1, permutation.typ
e = "X2", nrepet = nrepet, perm.mat= perm.mat)$pval[2]
```

```
  if (ncol(Y)==1){
    if (is.null(Z)) Z <- matrix(1,nrow = nrow(Y),ncol=1)
    pval_par <- anova(lm(Y~Z), lm(Y~Z+X))$'Pr(>F)'[2]
    nam <- "parametric"
    pval_parw <- anova(lm(Y~Z, weights = w),lm(Y~Z + X, weights = w)
)$'Pr(>F)'[2]
    namw <- "parametric weighted"
  } else {pval_par<- NULL; pval_parw<- NULL; nam <- NULL; namw <- NU
LL}

  pval <- c(pval_Y_RDA,pval_X2_RDA, pval_parw, pval_Y1_RDA,pval_X_RD
A,pval_par,pval_Y_RDA_OLS,pval_X_RDA_OLS)


  names(pval) <- c("RDA_Y","RDA_Ycan3","RDA_X2", namw,"RDA_Y1","RDA_
X", nam, "RDA_Y_OLS", "RDA_X_OLS")
  mysvd<- svd # eig1 tests can be done
  return(pval)
}



testXgivenZ4 <- function(Y,X, Z=NULL, w= 1:nrow(Y), nrepet = 999){
  # testing the effects of X on Y given Z by 4 different methods:
  # Y & X2 (weighted methods) and Y1 & X (unweigthed) permutation me
thods
  mysvd <- dummysvd # no eig1 tests are needed (avoid svd)
  N <- nrow(Y)
  perm.mat <- matrix(0,nrow = nrepet, ncol = N)
  for (i in 1:nrepet){
    perm.mat[i,] <- sample(N)
  }

  pval_Y_RDA <- randperm_RDA(Y,X, Z = Z, w = w, permutation.type = "
Y", nrepet = nrepet, perm.mat= perm.mat)$pval[2:3]
  pval_Y1_RDA <- randperm_RDA(Y,X, Z = Z, w = w, permutation.type =
"Y1", nrepet = nrepet, perm.mat= perm.mat)$pval[2]

  perm.mat <- t(apply(perm.mat,1,order))
  pval_X_RDA <- randperm_RDA(Y,X, Z = Z, w = w, permutation.type = "
X", nrepet = nrepet, perm.mat= perm.mat)$pval[2]
  pval_X2_RDA <- randperm_RDA(Y,X, Z = Z, w = w, permutation.type =
"X2", nrepet = nrepet, perm.mat= perm.mat)$pval[2]

  if (ncol(Y)==1){
    if (is.null(Z)) Z <- matrix(1,nrow = nrow(Y),ncol=1)
    pval_par <- anova(lm(Y~Z), lm(Y~Z+X))$'Pr(>F)'[2]
    nam <- "parametric"
    pval_parw <- anova(lm(Y~Z, weights = w),lm(Y~Z + X, weights = w)
)$'Pr(>F)'[2]
    namw <- "parametric weighted"
  } else {pval_par<- NULL; pval_parw<- NULL; nam <- NULL; namw <- NU
LL}
```

```
  pval <- c(pval_Y_RDA,pval_X2_RDA, pval_parw, pval_Y1_RDA,pval_X_RD
A,pval_par)


  names(pval) <- c("RDA_Y","RDA_Ycan3","RDA_X2", namw,"RDA_Y1","RDA_
X", nam)
  mysvd<- svd # eig1 tests can be done
  return(pval)
}



tests_NaXis <- function(Y,X, Z = NULL, w= 1:nrow(Y), nrepet = 999){
  # Y and X permutation
  N <- nrow(Y)
  perm.mat.list <- list()
  for (i in 1:4) {
    perm.mat <- matrix(0,nrow = nrepet, ncol = N)
    for (i in 1:nrepet){
      perm.mat[i,] <- sample(N)
    }
    perm.mat.list <- c(perm.mat.list, list(perm.mat))
  }

  pval_Y_RDA <- test_axes_RDA(Y,X, Z = Z, w=w, permutation.type = "Y
", nrepet = nrepet, perm.mat= perm.mat.list, silent = TRUE)
  eigen_values <- pval_Y_RDA$eig
  pval_Y_RDA <- pval_Y_RDA$summary[c(3,4),,drop = FALSE]

  for (i in 1:4){
    perm.mat.list[[i]] <- t(apply( perm.mat.list[[i]], 1,order))
  }

  pval_X_RDA <- test_axes_RDA(Y,X, Z = Z, w=w, permutation.type = "X
", nrepet = nrepet, perm.mat= perm.mat.list, silent = TRUE)$summary[
c(3,4),,drop = FALSE]
  pval <- rbind(pval_Y_RDA, pval_X_RDA)
  txt <- c("eig1","trace")
  md <- c("RDA_Y","RDA_X")

  rownames(pval) <-  kronecker(md,txt, FUN = paste)
  return(list(pval=pval, eigen_values =eigen_values))
}



# Hill number of order 2: N2
fN2 <- function(x){x <- x/sum(x); 1/sum(x*x)}
whi <-function(x){which(x == FALSE)[1]} # for which dimension is sig
nificant
which2 <- function(x)as.numeric(names(x)[which.max(x)]) # for which
dimension is modal
sdw_names <- function(x){fsdw( as.numeric(names(x)), w = x) } # for
which dimension is modal
```

4. Appendix S4: Weighted univariate regression in R

This R-script was used in Section 4.1, in particular to obtain the data for Fig. 2. A small modification (a const_series replacing the overalleffect_series with effect = 0) of the script was used for Fig. 1.

```
rm(list=ls(all=TRUE))  # remove all existing items from the workspac
e
# permutation tests in weighted univariate simple and multiple regre
ssion.
library(mvtnorm)
source("Rfunctions/residualXY_permutation.r")
source("rfunctions/Test_4XYmethods_functions.r")
mysvd<- dummysvd # if no eig1 tests are needed (avoid svd)
Scenario.list <-   list()

Datasave <- FALSE # if true all Data generated is saved in Data.list
TEST <- TRUE # few simulations and permutations/shuffles
text_expX <- "_power_none"
#load("Rdata/SeriesUnivariate.Rdata") # OR next line
Data.list <- list() # if no data are loaded

n_series <- 30 # c(5:10,30, 100)
n_simul <- 1000
nrepet <- 199
nominal.level  <- 0.05

set.seed(3151)

overalleffect_series <-  rev(c(seq(from =0, to = 1, by = 0.1), seq(f
rom=1.25, to =1.5, by = 0.25)))
correlat_series <- 0.7
CvsVIW_series <- 0.5  #rev(seq(from =0, to = 1, by = 0.1))
#CvsVIW_series <-  1 # gradient from 0 (v=1/w) to 1 (variance = cons
tant)
weight_overdispersion_series <-  rev(c(0, 0.1, 1, 2, 10, 20))
effectZ_series = 1 # effectZ =  0 for simple regression (in data gen
eration)
includeZ_series = TRUE # FALSE for not including Z (in analysis)
XZtrans_series <- "none" # c("none","expX", "binX")
corwX_series <- 0 # rev(seq(from=0, to= 1, by = 0.25))
distr_series <- "student"# c("student", "normal")

df_series <- c(5)

if (TEST){
  n_simul <- 250
  nrepet <- 49

  overalleffect_series <-  c(0,0.5,1) #rev(c(seq(from =0, to = 1, by
= 0.1), seq(from=1.25, to =1.5, by = 0.25)))
  correlat_series <- 0.7
  CvsVIW_series <- c(0.5)  #rev(seq(from =0, to = 1, by = 0.1))
```

```
  #CvsVIW_series <-  1 # gradient from 0 (v=1/w) to 1 (variance = co
nstant)
  weight_overdispersion_series <- c(0,20) #rev(c(0, 0.1, 1, 2, 10, 2
0))
  effectZ_series = 1 # effectZ =  0 for simple regression (in data g
eneration)
  includeZ_series = TRUE # FALSE for not including Z (in analysis)
  XZtrans_series <- "none" # c("none","expX", "binX")
  corwX_series <- 0 # rev(seq(from=0, to= 1, by = 0.25))
  distr_series <- "student"# c("student", "normal")
}

scenarios <- expand.grid(n = n_series, effect= overalleffect_series,
weight_overdispersion = weight_overdispersion_series,
CvsVIW = CvsVIW_series, correlat = correlat_series, effectZ = effect
Z_series, includeZ = includeZ_series,
XZtrans = XZtrans_series, corwX = corwX_series, df  = df_series, dis
tr = distr_series)

dim(scenarios)

## [1]  6 11

klist1 <- 1:nrow(scenarios)
if (length(Data.list)>0) Datasave <- FALSE; # do not save again if D
ata.list already exists with data

threshold = (nominal.level + 2*sqrt(nominal.level*(1-nominal.level)/
n_simul))
print(threshold)

## [1] 0.0775681

scenarios

##    n effect weight_overdispersion CvsVIW correlat effectZ include
Z XZtrans
## 1 30    0.0                    0    0.5      0.7       1     TRU
E    none
## 2 30    0.5                    0    0.5      0.7       1     TRU
E    none
## 3 30    1.0                    0    0.5      0.7       1     TRU
E    none
## 4 30    0.0                   20    0.5      0.7       1     TRU
E    none
## 5 30    0.5                   20    0.5      0.7       1     TRU
E    none
## 6 30    1.0                   20    0.5      0.7       1     TRU
E    none
##   corwX df   distr
## 1     0  5 student
## 2     0  5 student
## 3     0  5 student
## 4     0  5 student
```

```
## 5     0  5 student
## 6     0  5 student

k=1
i=1
####


for (k in klist1){

  if (length(Data.list)< k) Data.list[[k]]<- list()
  effect <- scenarios[k,"effect"]
  weight_overdispersion <- scenarios[k,"weight_overdispersion"]
  CvsVIW <- scenarios[k, "CvsVIW"]
  correlat <- scenarios[k, "correlat"]
  Sigma <- matrix(c(1,correlat,correlat,1), nrow = 2, ncol=2)
  effectZ <- scenarios[k, "effectZ"]
  includeZ <-  scenarios[k, "includeZ"]
  corwX <-  scenarios[k, "corwX"]
  n <- scenarios[k,"n"]
  df <- scenarios[k,"df"]
  distr <- scenarios[k,"distr"]
  # regression model  data----------------------------------
  for (i in 1:n_simul){

    if (length(Data.list[[k]])<i){

     # generate data
      if (weight_overdispersion > 0) {
      w0 <- rnbinom(n, size = 1/weight_overdispersion, mu = 10)+1
      } else w0 <- rep(1,n)
      if (distr =="normal") {
        XZ <- MASS::mvrnorm(n,mu=rep(0,2),Sigma)
      } else {
        XZ <-  mvtnorm::rmvt(n, sigma = Sigma * (df - 2) / df, df =
df)
      }
      X <- XZ[,1,drop = FALSE]
      Z <- XZ[,2,drop = FALSE]

      if (weight_overdispersion >0 & diff(range(w0))) {
        logw0 <- log(w0)
        logw <-  mean(logw0) + sd(logw0) * (corwX * X[,1] + sqrt(1 -
corwX^2) * scale(logw0))
        w <- exp(logw[,1])
      } else w<- w0

      sWn <- sqrt(w)
      sdw <- 1/sWn
      sdw <-  n * sdw/sum(sdw)
      Noise1 <- matrix(rnorm(n),nrow = n, ncol =1)
      Noise2 <- matrix(sdw*rnorm(n),nrow = n, ncol =1)
      #var(Noise1)
      #var(Noise2)
```

```
      Y <- 1 + effect * X +  effectZ * Z  +
      CvsVIW * Noise1 + sqrt(1-CvsVIW^2) * Noise2
      obj <- list(Y =Y, X=X, Z = Z, w=w)
      if (Datasave) Data.list[[k]][[i]] <-list(obj= obj, k=k,scenari
o = scenarios[k,])
    } else {
      obj <- Data.list[[k]][[i]]$obj
      Y <- obj$Y; X <- obj$X; Z <- obj$Z; w <- obj$w
    }
    if (!includeZ)  Z <- NULL
    # test effect of X on Y given Z (if !NULL) using weights w
    if (scenarios[k,"XZtrans"]=="expX")  {
      X <- exp(X)
    } else if (scenarios[k,"XZtrans"]=="bin") {
      X <- ifelse(X>0,0,1)
    }
    pvals <- testXgivenZ4(Y,X, Z,  w = w, nrepet = nrepet)
    if (i == 1){
    Res <- matrix(NA, nrow = n_simul, ncol = length(pvals))
    colnames(Res) <- names(pvals)
    N2 <- rep(NA, n_simul)
    }
    Res[i,]<- pvals
    N2[i] <- fN2(w)
  } # n_simul
  signi <- Res <= nominal.level
  scenk <- scenarios[k,]
  names(scenk) <- names(scenarios)
  print(data.frame(scenk ,k=k, outof = nrow(scenarios)))
  Rejection_rate <-   colMeans(signi, na.rm=TRUE)
  cat("Rejection_rates\n")
  print(Rejection_rate)

  Scenario.list[[length(Scenario.list)+1]] <-  list(  Rejection_rate
=   Rejection_rate, pvals = Res, N2 = N2,
  scenario =scenk,
  n_simul= n_simul)
  save( Scenario.list, nrepet, n_simul, nominal.level ,  threshold,
     file = paste("Rdata/Series_uni_T",text_expX,".Rdata",sep = ""))
}

##    n effect weight_overdispersion CvsVIW correlat effectZ include
Z XZtrans
## 1 30      0                    0    0.5      0.7       1     TRU
E    none
##   corwX df   distr k outof
## 1     0  5 student 1     6
## Rejection_rates
##               RDA_Y          RDA_Ycan3              RDA_X2
##               0.048              0.048               0.040
## parametric weighted             RDA_Y1               RDA_X
##               0.052              0.048               0.040
##          parametric
##               0.052
```

```
[…]
##    n effect weight_overdispersion CvsVIW correlat effectZ include
Z XZtrans
## 4 30      0                    20    0.5      0.7       1     TRU
E    none
##   corwX df   distr k outof
## 4     0  5 student 4     6
## Rejection_rates
##               RDA_Y           RDA_Ycan3              RDA_X2
##               0.232               0.248               0.236
## parametric weighted              RDA_Y1               RDA_X
##               0.280               0.008               0.040
##          parametric
##               0.044
##    n effect weight_overdispersion CvsVIW correlat effectZ include
Z XZtrans
## 5 30    0.5                    20    0.5      0.7       1     TRU
E    none
##   corwX df   distr k outof
## 5     0  5 student 5     6
## Rejection_rates
##               RDA_Y           RDA_Ycan3              RDA_X2
##               0.528               0.532               0.524
## parametric weighted              RDA_Y1               RDA_X
##               0.552               0.168               0.276
##          parametric
##               0.384
[…]

if (Datasave) save(Data.list, scenarios,includeZ, n_simul, nominal.l
evel,threshold,
                   file = "Rdata/SeriesUnivariate.Rdata")
```

5. Appendix S5: Generation of low-rank data in R

The function `generate_LR_data`, with the function `setoptions4LR` to set the `model.coefs` argument, implements the reduced rank model of Appendix S1 and is used in Appendices S6 and S7.

```
generate_XZ_normal <- function(n, model.coefs){
  # generates predictor(X) and covariate (Z) data that are multivari
ate normal distributed

  # n is the numbers of cases (rows)
  # The number of explanatory variables is determined by model.coefs
$pE :
  # model.coefs$pE[1:4] for number of environmental variables for ax
1, ax2, noise and covariables (covariates Edat4)
  #           $ correlat[1:5] AR1 autocorrelation of these variables
with the fifth for the AR1 correlation in the noise
  #           $ effect[1:4] effect sizes of Edat1, Edat2 and Edat4 (
as Edat3 is noise variables) effect[4] for a latent variable of unme
asured Variables
```

```
  #                          See E4 below
  #           $ corZX[1:3] correlation Edat1 with Edat4$E, Edat2 wit
h Edat1$E, Edat3 with Edat1$E


  pE <- model.coefs$pE
  correlat <- model.coefs$correlat
  corXZ <- model.coefs$corXZ
  # normal distribution
  Edat1 <- generate_Edata(n,p = pE[1], correlation =correlat[1])# fo
r ax1
  Edat2 <- generate_Edata(n,p = pE[2], correlation =correlat[2])# fo
r ax2
  if (pE[3] >0) Edat3 <- generate_Edata(n,p = pE[3], correlation =co
rrelat[3])# environmental noise variables
  Edat4 <- generate_Edata(n,p = pE[4], correlation =correlat[4])# fo
r covariate data (ax0)
  Edat5 <- generate_Edata(n,p = 1, correlation =0) # latent variable
s without measurements
  Edat1$Edat <- sqrt(1 - corXZ[1]^2) * Edat1$Edat + matrix(corXZ[1]
* Edat4$E, nrow = n, ncol = pE[1]) #
  Edat2$Edat <- sqrt(1 - corXZ[2]^2) * Edat2$Edat + matrix(corXZ[2]
* Edat1$E, nrow = n, ncol = pE[2]) #
  if (pE[3] >0) Edat3$Edat <- sqrt(1 - corXZ[3]^2) * Edat3$Edat + ma
trix(corXZ[3] * Edat2$E, nrow = n, ncol = pE[3]) #
  else Edat3 <- list(Edat = NULL, E = NULL)#
  Edat <- cbind(Edat1$Edat,Edat2$Edat,Edat3$Edat) # p1 + p2 + p3 pre
dictor variables
  E4 <- cbind(Edat1$E, Edat2$E, Edat4$E, Edat5$E) # NB no Edat3 as i
t is intended as noise variables
  colnames(E4) <- c("constrained ax1","constrained ax2", "covariable
ax0", "unconstained ax1")
  Zdat <- Edat4$Edat; Z <- Edat4$E
  if (model.coefs$pE[4]==0) Zdat <- NULL # no Z#
  obj <- list( Edat = Edat, Zdat= Zdat, E4 = E4, Z=Z, Sigma0 = NULL)
  # E4: 4 latent variables: belonging to (Edat1,Edat2) , covariable
Z (from Zdat) and
  # (fourth) a latent variable giving an extra axis (dimension)
  #  which would give a  first axis in PCA when effect=c(0,0,0,1)
  return(obj)
}
generate_XZ_student <- function(n, model.coefs){
  # generates predictor(X) and covariate (Z) data that are multivari
ate student distributed with df = model.coefs$df
  # n is the numbers of cases (rows)  to be generated.
  # The function calls the function generate_XZ_normal to set the co
variance matrix

  pE <- model.coefs$pE
  # normal distribution
  Sigma0 <- model.coefs$Sigma0
  # sigma ------------------------------------------------------------
---------
  if (is.null(Sigma0)){
```

```
    Sigma0 <- matrix(0,nrow = sum(pE),ncol=sum(pE))
    nrp <- 1000
    for (ii in 1:nrp){
      obj <- generate_XZ_normal(n, model.coefs)
      XZ <- cbind(obj$Edat, obj$Zdat)
      Sigma0 <- Sigma0 + cov2cor(cov(XZ))
    }
    Sigma0 <- cov2cor(Sigma0)
    # guarantee that uncorrelated variables are really uncorrelated
    Sigma0 <- ifelse(abs(Sigma0)<0.1, 0, Sigma0)
    #View(round(Sigma0,2))
  } else Sigma0 <- model.coefs$Sigma0
  df <- model.coefs$df
  XZ<-mvtnorm::rmvt(n,sigma = Sigma0 * (df - 2) / df, df = df)
  #XZ<-mvtnorm::rmvnorm(n,sigma = Sigma0)
  # end sigma ------------------------------------------------------
---------
  p12 <- sum(pE[1:2]);p123 <- sum(pE[1:3]);p1234 <- sum(pE[1:4])
  Edat1 <- XZ[,1:pE[1]]
  Edat2 <- XZ[,pE[1]+(1:pE[2])]
  Edat3 <- XZ[,p12+(1:pE[3])]
  Edat4 <- XZ[,p123+(1:pE[4])]
  lincomb <- function(Edat, Sigma, b= rep(1, ncol(Edat))){
    sdE <-  sqrt(b %*% Sigma %*%b)
    coefs <- b/c(sdE)
    E <- Edat %*%coefs
  }

  # make residuals of Edat1 orthogonal to Edat4
  Edat1.4 <- scale(qr.resid(qr(Edat4),Edat1))
  # make residuals of Edat1 orthogonal to Edat4 and Edat1
  Edat2.14 <- scale(qr.resid(qr(cbind(Edat1,Edat4)),Edat2))
  Ax1 <- lincomb(Edat1.4, Sigma = cov2cor(cov(Edat1.4)))
  Ax2 <- lincomb(Edat2.14, Sigma = cov2cor(cov(Edat2.14)))
  Ax0 <- lincomb(Edat4, Sigma = cov2cor(cov(Edat4) ))
  Edat1 <- list(Edat= Edat1, E = Ax1)
  Edat2 <- list(Edat= Edat2, E = Ax2)
  Edat3 <- list(Edat= Edat3, E = NULL)
  Edat4 <- list(Edat= Edat4, E = Ax0)
  Edat5 <- generate_Edata(n,p = 1, correlation =0) #  variance of no
ise NOT dependent on Z or X
  # note that the variance of the noise would be dependent on Z and
X if Edat5 is generated from a joint multivariate t
  Edat <- cbind(Edat1$Edat,Edat2$Edat,Edat3$Edat) # p1 + p2 + p3 pre
dictor variables
  E4 <- cbind(Edat1$E, Edat2$E, Edat4$E, Edat5$E) # NB no Edat3 as i
t is intended as noise variables
  colnames(E4) <- c("constrained ax1","constrained ax2", "covariable
ax0", "unconstained ax1")
  Zdat <- Edat4$Edat; Z <- Edat4$E
  if (model.coefs$pE[4]==0) Zdat <- NULL # no Z#
  obj <- list( Edat = Edat, Zdat= Zdat, E4 = E4, Z=Z, Sigma0 = Sigma
0)
  # E4: 4 latent variables: belonging to (Edat1,Edat2) , covariable
```

```
Z (from Zdat) and
  # (fourth) a latent variable giving an extra axis (dimension)
  #  which would give a  first axis in PCA when effect=c(0,0,0,1)
  return(obj)
}


generate_LR_data <- function(n,m, model.coefs){
  # LR low rank  data
  # n and m are the numbers of cases (rows) and response variables t
o be generated.

  # The number of explanatory variables is determined by model.coefs
$pE :
  # model.coefs$pE[1:4] for number of environmental variables for ax
1, ax2, noise and covariables (covariates Edat4)
  #           $ correlat[1:5] AR1 autocorrelation of these variables
with the fifth for the AR1 correlation in the noise
  #           $ effect[1:4] effect sizes of Edat1, Edat2 and Edat4 (
as Edat3 is noise variables) effect[4] for a latent variable of unme
asured Variables
  #                         See E4 and mu.interaction below
  #           $ corZX[1:3] correlation Edat1 with Edat4$E, Edat2 wit
h Edat1$E, Edat3 with Edat1$E
  #           $ weight_overdispersion is the overdispersion paramete
r of a negative binomial (=1/size), with 0 giving equal weights
  #           $ corwX correlation between log(w) and predictor x1 =
Edat[,1]

  pE <- model.coefs$pE

  overalleffect <- model.coefs$overalleffect
  effect <- model.coefs$effect
  if (m ==1) model.coefs$double_center <- FALSE # cannot have compos
itional data with m= 1
  if (pE[4]==0) {effect[3]=0; pE[4]<-1} # note:model.coefs$pE[4] rem
ains unchanged!
  effect[c(1,2)] <- overalleffect * effect[c(1,2)] # scale the effec
t defining axis 1 and 2 (Edat1, Edat2), same effects of  covars and
noise

  correlat <- model.coefs$correlat
  corXZ <- model.coefs$corXZ
  sigma_epsilon_ij  = model.coefs$sigma_epsilon_ij;

  if (model.coefs$distr == "normal"){
    obj <- generate_XZ_normal(n,model.coefs)
  } else {
    obj <- generate_XZ_student(n,model.coefs)
  }
  Edat <- obj$Edat; Zdat <- obj$Zdat; E4 <- obj$E4; Z <- obj$Z; Sigm
a0 <- obj$Sigma0
  if (length(effect)!= ncol(E4)) errorCondition(" length of effect u
nequal ncol of  E4 in generate_LR_community")
```

```
  Bdat <- generate_Edata(m, p = length(effect), correlation = 0)$Eda
t

  # E4: 2 independent axes E1 and E2 + 1 covariable (from Edat4)  +
1 "unconstrained" PCA axis# noise
  mu.interaction <-  E4 %*% diag(effect) %*% t(Bdat)

  Noise <- sigma_epsilon_ij[1] * generate_Edata(n,p = m, correlation
=correlat[5])$Edat +
           sigma_epsilon_ij[2] * matrix(rnorm(n*m), nrow =n, ncol =
m )

  weight_overdispersion <- model.coefs$weight_overdispersion
  if (  abs(weight_overdispersion)<1.0e-10 ) {w0 <- rep(1,n)} else {
                                                    w0 <- rnbin
om(n, size = 1/weight_overdispersion , mu = 10)+1
  }
  corwX <- model.coefs$corwX
  E41 <- scale(Edat[,1])

  if (weight_overdispersion >0 & diff(range(w0))) {
    logw0 <- log(w0)
    logw <-  mean(logw0) + sd(logw0) * (corwX * E41 + sqrt(1 - corwX
^2) * scale(logw0))
    w <- exp(logw[,1])
  } else w <- w0

    if (model.coefs$VIW) {
      sWn <- sqrt(w)
      sdw <- 1/sWn
      sdw <-  n * sdw/sum(sdw)
    } else sdw = 1

  if (model.coefs$VIW) Noise <- diag(sdw) %*% Noise
  Y <- model.coefs$intercept + mu.interaction + Noise
  rownames(Y)<- paste("site", seq_len(nrow(Y)), sep="")
  colnames(Y)<- paste("spec",seq_len(ncol(Y)),sep="")
  obj <- list(Y=Y, Edat = Edat, Tdat= Bdat, Zdat= Zdat, Z= Z, E4=E4,
              effect = effect, mu.interaction =mu.interaction, w=w,
Sigma0 = Sigma0,
              Noise = Noise, weight_overdispersion = weight_overdisp
ersion)
  # E4: 4 latent variables: belonging to (Edat1,Edat2) , covariable
Z (from Zdat) and
  # (fourth) a latent variable giving an extra axis (dimension)
  #  which would give a  first axis in PCA when effect=c(0,0,0,1)
  return(obj) # Y = Y in the paper
}



setoptions4LR <- function( intercept = 30,  pE = rep(1,4), correlat
= rep(0,5), corXZ = c(0,0,0),
                           overalleffect = 1, effect = c(1,0.5,0,0),
```

```
double_center = FALSE, distr = "normal",XZtrans = "none",
                            sigma_epsilon_ij = c(0,1), VIW = FALSE, w
eight_overdispersion =0, corwX= 0, df = 5, Sigma0 = NULL
){
  # VIW logical whether variance should be inversely proportional to
the weight
  names(pE) <- c("nE_ax1", "nE_ax2","nE_noise", "nZ_covar")
  names(corXZ) <- c("r_Edat1_Z","r_Edat2_ax1","r_Edat3_ax2")
  names(effect) <- c("b_ax1","b_ax2", "b_Z","b_PCA1")
  names(correlat) <- c("r_Eax1","r_Eax2", "rE3","r_Z","r_noise")
  names(sigma_epsilon_ij) <- c("sigma_AR1noise","sigma_white_noise")
  names(corwX) <- c("cor(log(w),x1)")
  return(
    list( pE = pE, overalleffect = overalleffect, correlat = correla
t,
          effect = effect, corXZ = corXZ,corwX= corwX, distr = distr
, XZtrans = XZtrans,
          sigma_epsilon_ij  = sigma_epsilon_ij, VIW = VIW, intercept
=intercept, weight_overdispersion = weight_overdispersion,
          df = df, Sigma0 = Sigma0)
  )
}


# function to generate normally distributed data and its linear comb
ination  ------------------------------------
generate_Edata <-function(n,p, correlation = 0, b=rep(1,p), mixfacto
r = 0){
  # generate p correlated normally distibuted explanatory variables
in Edat and
  # a linear combination of these in E
  # rescale b and E so that E~N(0,1),
  # value: list with Edat, E and coefficients (rescaled b)
  if (p >1){
    ii <- 1:p
    Sigma <-  correlation^abs(outer(ii,ii,  '-'))
    sdE <-  sqrt(b %*% Sigma %*%b)
    coefs <- b/c(sdE)
    Edat<-MASS::mvrnorm(n,mu=rep(0,p),Sigma) + mixfactor * matrix(2*
((runif(n*p) > 0.5)-0.5),nrow = n,ncol =p)
    E <- Edat %*%coefs

  } else {
    E <- Edat <- matrix(rnorm(n), nrow =n, ncol=1)
    coefs <- 1
  }
  rownames(E)<- rownames(Edat)<- paste("site", seq_len(nrow(Edat)),
sep="")
  return(list(E = E, Edat= Edat, coefficients = coefs))
}
```

6. Appendix S6: Testing effects by weighted RDA in R

R-script used in Section 4.2 of the main text. The four versions of noise in Fig. 4 are obtained by setting `VIW` in `model.coefs` to `TRUE` or `FALSE` for variance proportional to the weights and constant variance, respectively, and the fourth entry of `effect` in `model.coefs` to either 1 or 0 (to include or not to include rank 1 error).

```
rm(list=ls(all=TRUE))  # remove all existing items from the workspac
e

source("Rfunctions/generate_lowrank_data.r")
source("Rfunctions/residualXY_permutation.r")
source("Rfunctions/Test_4XYmethods_functions.r")

#mysvd <- svd if eig1 test is needed
mysvd<- dummysvd # if no eig1 tests are needed (avoids svd)

Scenario.list <-   list()
Data.list <- list()

n_series <- c(30)
m <- 50

p1 <- 4
p2 <- 4
p3 <- 4
p4 <- 2

TEST <- TRUE
Datasave <- FALSE # if true all Data generarated is saved in Data.li
st
if (Datasave) memory.limit(size = 50000)
text_expX <- "_power_WO"
n_simul <-   1000
nominal.level  <- 0.05


set.seed(31517)

nrepet <- 199

overalleffect_series <- c(seq(from =0, to = 1.2, by = 0.1))
#weight_overdispersion_series <- c(0,0.01,0.1, 0.25, 0.5, 1)
weight_overdispersion_series <- 1#

correlat_series <- 0.7 #rev(c(seq(from =0, to = 0.9, by = 0.1),seq(f
rom =0.91, to = 0.99, by = 0.01)))
VIW_series <-  c(FALSE,TRUE) # TRUE if var(Noise) is inverse proport
ional to the weight
rank1_series <-  c(0,1)
effectZ_series = 1 # effectZ =  0 for simple regression (in data gen
eration)
includeZ_series = TRUE # FALSE for not including Z (in analysis)
XZtrans_series <-"none"# c("none","expX", "binX")
```

```r
distr_series <- c("student")#, "normal")
corwX_series <- 0 #(seq(from=0, to= 1, by = 0.25))

if (TEST){
  n_simul <- 250
  nrepet <- 49
  overalleffect_series <- c(seq(from =0, to = 1.2, by = 0.4))
  weight_overdispersion_series <- 1#

  correlat_series <- 0.7 #rev(c(seq(from =0, to = 0.9, by = 0.1),seq
(from =0.91, to = 0.99, by = 0.01)))
  VIW_series <-  c(FALSE,TRUE) # TRUE if var(Noise) is inverse propo
rtional to the weight
  rank1_series <-  1#c(0,1)
  effectZ_series = 1 # effectZ =  0 for simple regression (in data g
eneration)
  includeZ_series = TRUE # FALSE for not including Z (in analysis)
  XZtrans_series <-"none"# c("none","expX", "binX")
  distr_series <- c("student")#, "normal")
  corwX_series <- 0# seq(from=0, to= 1, by = 0.5)
}


threshold = (nominal.level + 2*sqrt(nominal.level*(1-nominal.level)/
n_simul))
print(threshold)
```

```
## [1] 0.0775681
```

```r
scenarios <- expand.grid(n = n_series, effect= overalleffect_series,
weight_overdispersion = weight_overdispersion_series,
                         VIW = VIW_series, correlat = correlat_serie
s, effectZ = effectZ_series, includeZ = includeZ_series,
                         rank1_noise = rank1_series, XZtrans = XZtra
ns_series, corwX= corwX_series, distr = distr_series)
dim(scenarios)
```

```
## [1]  8 11
```

```r
klist1 <- 1:nrow(scenarios)

for (k in klist1){

    if (length(Data.list)< k) Data.list[[k]]<- list()


    n <- scenarios[k,"n"]
    effect <- scenarios[k,"effect"]
    weight_overdispersion <- scenarios[k,"weight_overdispersion"]
    VIW <- scenarios[k, "VIW"]
    correlat <- scenarios[k, "correlat"]
    effectZ <- scenarios[k, "effectZ"]
    includeZ <-  scenarios[k, "includeZ"]
    rank1_noise <- scenarios[k, "rank1_noise"]
    corwX <- scenarios[k,"corwX"]
```

```
    correlat5 <- rep(correlat,5)
    corXZ3 <-  rep(correlat,3)
    XZtrans <- scenarios[k,"XZtrans"]
    distr <- scenarios[k,"distr"]

    model.coefs <- setoptions4LR( intercept = 1,  pE = c(p1,p2,p3,p4
), correlat = correlat5, corXZ = corXZ3,
                                  overalleffect = effect, effect = c
(1,0.5,effectZ, rank1_noise),
                                  sigma_epsilon_ij = c(0,1), VIW = V
IW, weight_overdispersion = weight_overdispersion,
                                  corwX = corwX, XZtrans = XZtrans,
distr = distr)




    # regression model  data----------------------------------
    for (i in 1:n_simul){

# sec ------------------------------------------------------------
-------
        if (length(Data.list[[k]])<i){
         obj <- generate_LR_data(n,m, model.coefs)
         model.coefs$Sigma0 <- obj$Sigma0
         if (Datasave) Data.list[[k]][[i]] <-list(obj= obj, k=k,scen
ario = scenarios[k,], model.coefs=model.coefs)

        } else {
            obj <- Data.list[[k]][[i]]
        }
      Y <- obj$Y
      X <- obj$Edat
      Z <- obj$Zdat
      w <- obj$w
      if (!includeZ || model.coefs$pE[4]==0) Z <- NULL

      if (scenarios[k,"XZtrans"]=="expX")  {
        X <- exp(X)
      } else if (scenarios[k,"XZtrans"]=="bin") {
        X <- ifelse(X>0,0,1)
      }
      pvals <- testXgivenZ4_with_OLS(Y, X, Z = Z, w = w, nrepet = nr
epet)


  # end sec ------------------------------------------------------
---------


        if (i == 1){
            Res <- matrix(NA, nrow = n_simul, ncol = length(pvals))#
number of tests, 5 (obs site,obs spec, row/col/max), number of simul
```

```
ations
          colnames(Res) <- names(pvals)
          N2 <-  rep(NA)
        }
        Res[i,]<- pvals
        N2[i] <- fN2(w)
   } # n_simul
    signi <- Res <= nominal.level
    scenk <- scenarios[k,]
    names(scenk) <- names(scenarios)
    print(data.frame(scenk ,k=k, outof = nrow(scenarios)))
    Rejection_rate <-   colMeans(signi, na.rm=TRUE)
    print("rejection rates")
    print(Rejection_rate)

    Scenario.list[[length(Scenario.list)+1]] <-  list(  Rejection_ra
te =   Rejection_rate, pvals = Res, N2 = N2,
                                  scenario = scenk,
                                  n_simul= n_simul,model.coefs = mode
l.coefs)

    save( Scenario.list, nrepet, n_simul, nominal.level ,  threshold
, scenarios,
          file = paste("Rdata/Series3_results_T5",text_expX,".Rdata"
,sep = ""))
  }

##    n effect weight_overdispersion   VIW correlat effectZ includeZ
## 1 30      0                     1 FALSE      0.7       1     TRUE
##   rank1_noise XZtrans corwX   distr k outof
## 1           1    none     0 student 1     8
## [1] "rejection rates"
##     RDA_Y RDA_Ycan3    RDA_X2    RDA_Y1     RDA_X RDA_Y_OLS RDA_X
_OLS
##     0.576     0.600     0.564     0.020     0.040     0.024     0
.036

[..]

##    n effect weight_overdispersion  VIW correlat effectZ includeZ
## 8 30    1.2                     1 TRUE      0.7       1     TRUE
##   rank1_noise XZtrans corwX   distr k outof
## 8           1    none     0 student 8     8
## [1] "rejection rates"
##     RDA_Y RDA_Ycan3    RDA_X2    RDA_Y1     RDA_X RDA_Y_OLS RDA_X
_OLS
##     0.996     0.996     0.996     0.940     0.968     1.000     1
.000

 if (Datasave) save(Data.list, scenarios,includeZ, n_simul, nominal.
level,threshold, correlat, corXZ3,effectZ,
     p1,p2,p3,p4, n,m, overalleffect_series, powerval_series, klist1
,
     file = "Rdata/Series3_corwX.Rdata")
```

7. Appendix S7: Testing of the axes of weighted RDA in R

R-script used in Section 4.3 of the main text. As this script takes some time to complete, a `NextRun` variable is provided to restart the computation when required.

```
rm(list=ls(all=TRUE))  # remove all existing items from the workspac
e


source("Rfunctions/generate_lowrank_data.r")
source("Rfunctions/residualXY_permutation.r")
source("rfunctions/Test_4XYmethods_functions.r")
mysvd <- svd
NextRun <- FALSE # FALSE == first run/ no RDATA with scenario.list
TEST <- TRUE
Datasave <- FALSE
# if true all Data generarated is saved in Data.list
if (Datasave) memory.limit(size = 50000)

filen <- "Rdata/Series3_results_for_Fig5_None_Student_disp1.Rdata"
# file with Scenario.list results so far


if (NextRun){
  load(filen)
  idnext <- length(Scenario.list)+1
  id_series <- idnext : nrow(scenarios)
  text_expX <- "" # addition to results file
} else {

  set.seed(1537)
  Scenario.list <-   list()
  Data.list <- list() #
  #load("Rdata/Series2_fig3.Rdata") # analyse existing data file ins
tead of generating new.

  if (length(Data.list)>0 && !NextRun) Datasave <- FALSE;

  effectDim2_series <- seq(from=0, to =2, by = 0.25)
  weight_overdispersion_series <- 1
  VIW_series <-  c(TRUE, FALSE)
  rank1_noise_series <- c(1, 0)
  XZtrans_series <-"none" #c("none","expX", "binX")
  distr_series <- "student" #c("student", "normal")
  text_expX <- "none_student_distr1" # addition to results file

  n <- 30
  m <- 50
  p1 <- 4
  p2 <- 4
  p3 <- 4
  p4 <- 2
```

```
  effectZ = 1 # effect of Z
  includeZ = TRUE # FALSE for not including Z (in analysis)


  correlat =0.7
  correlat5 <- rep(correlat,5)
  corXZ3 <- rep(correlat,3) #  AR1 correlation between dimensions


  n_simul <-  1000
  nrepet <-  199
  nominal.level  <- 0.05

  if (TEST){
    n_simul <- 250
    nrepet <- 49
    threshold = (nominal.level + 2*sqrt(nominal.level*(1-nominal.lev
el)/n_simul))
    effectDim2_series <- seq(from=0, to =2, by = 0.1)
    rank1_noise_series <- 1 #c(1, 0)
  }

  threshold = (nominal.level + 2*sqrt(nominal.level*(1-nominal.level
)/n_simul))
  print(threshold)



  scenarios <- expand.grid(effectDim2= effectDim2_series,
                           weight_overdispersion = weight_overdisper
sion_series,
                           VIW = VIW_series, rank1_noise = rank1_noi
se_series,XZtrans = XZtrans_series, distr = distr_series)


  klist1 <- 1:nrow(scenarios)


}
```

```
## [1] 0.0775681
```

```
  for (k in klist1){

    if (length(Data.list)< k) Data.list[[k]]<- list()
    effectAx1A2ZR1noise = c(2, scenarios[k, "effectDim2"]  ,effectZ,
scenarios[k,"rank1_noise"])
    model.coefs <- setoptions4LR( intercept = 0,  pE = c(p1,p2,p3,p4
), correlat = correlat5, corXZ = corXZ3,
                                  overalleffect =1, effect = effectA
x1A2ZR1noise,
                                  weight_overdispersion = scenarios[
k, "weight_overdispersion"],
```

```
                                    sigma_epsilon_ij = c(0,1), VIW = s
cenarios[k, "VIW"],
                                    XZtrans = scenarios[k,"XZtrans"],d
istr= scenarios[k, "distr"] )



    # regression model  data----------------------------------
    for (i in 1:n_simul){

        if (length(Data.list[[k]])<i){

         obj <- generate_LR_data(n,m, model.coefs )
         if (Datasave) Data.list[[k]][[i]] <-list(obj= obj, scenario
s=scenarios[k,], model.coefs=model.coefs)

        } else {
            obj <- Data.list[[k]][[i]]$obj
        }
      Y <- obj$Y
      X <- obj$Edat
      Z <- obj$Zdat
      w <- obj$w
      if (!includeZ || model.coefs$pE[4]==0) Z <- NULL

      if (scenarios[k,"XZtrans"]=="expX")  {
        X <- exp(X)
      } else if (scenarios[k,"XZtrans"]=="bin") {
        X <- ifelse(X>0,0,1)
      }
      pvalseig <- tests_NaXis(Y, X, Z, w = w, nrepet = nrepet)
      pvals <- pvalseig$pval
      while (ncol(pvals)< 4) pvals <- cbind(pvals,1) # in case an ei
genvalue was too small...
      if (i == 1){
        Res <- array(NA, dim= c(dim(pvals),n_simul))
        dimnames(Res) <-  c(dimnames(pvals),list(simul=1:n_simul))
        Eigenvalues <- matrix(NA, nrow = n_simul, ncol = length(pval
seig$eigen_values))
        N2 <- rep(NA, n_simul)
      }
      Res[,,i]<- pvals
      Eigenvalues[i,] <- pvalseig$eigen_values
      N2[i] <- fN2(w)
   } # n_simul
    signi <- Res <= nominal.level
    scenk <- scenarios[k,]
    names(scenk) <- names(scenarios)
    print(data.frame(scenk ,k=k, outof = nrow(scenarios)))

    Rejection_rate <- apply(signi,c(1,2), mean, na.rm = TRUE)
    print("rejection rates")
    print(Rejection_rate)
```

```
    Scenario.list[[length(Scenario.list)+1]] <-  list(  Rejection_ra
te =   Rejection_rate, pvals = Res,
                                  scenario = scenk, N2 = N2,
                                  n_simul= n_simul,model.coefs = mode
l.coefs)

    save( Scenario.list, nrepet, n_simul, nominal.level ,  threshold
, scenarios,
          file = paste("Rdata/Series3_results_for_Fig5",text_expX,".
Rdata", sep = ""))
}

##   effectDim2 weight_overdispersion  VIW rank1_noise XZtrans   dis
tr k outof
## 1          0                     1 TRUE           1    none stude
nt 1    42
## [1] "rejection rates"
##
##               1     2 3 4
##   RDA_Y eig1  1 0.168 0 0
##   RDA_Y trace 1 0.144 0 0
##   RDA_X eig1  1 0.020 0 0
##   RDA_X trace 1 0.012 0 0
[…]
##    effectDim2 weight_overdispersion   VIW rank1_noise XZtrans   d
istr  k
## 42          2                     1 FALSE           1    none stu
dent 42
##    outof
## 42    42
## [1] "rejection rates"
##
##               1    2     3    4
##   RDA_Y eig1  1 1.00 0.184 0.00
##   RDA_Y trace 1 1.00 0.380 0.02
##   RDA_X eig1  1 1.00 0.048 0.00
##   RDA_X trace 1 0.98 0.040 0.00

 if (Datasave) save(Data.list, scenarios, n_simul, nominal.level,thr
eshold, correlat, corXZ3,includeZ,effectZ,
     p1,p2,p3,p4, n,m, klist1,
     file = "Rdata/Series3_fig5.Rdata")
```