Krzysztof Trajkowski

Przeglad pakietów do analizy zmiennych zależnych

8 stycznia 2013

1. Wprowadzenie

W tym poradniku zostanie dokonany przegląd kilku funkcji do analizy zmiennych zależnych z zastosowaniem pakietu R. Będzie to próba zaprezentowania kilku ciekawych alternatyw dla tradycyjnych procedur np. test t-Studenta vs. test Yuena czy test Friedmana vs. test Agresti–Pendergrast. Sądze że warto zwrócić na nie uwagę ponieważ w niektórych sytuacjach przewyższają one swoimi właściwościami klasyczne metody. Niektóre z nich są mało popularne ponieważ wymagają wykonania bardziej złożonych obliczeń. Jednak w dobie komputeryzacji stosowanie bardziej skomplikowanych algorytmów nie stanowi już żadnego problemu.

2. Dwie próby zależne

2.1. Test t-Studenta

Jednym z najbardziej popularnych testów do porównań dwóch zmiennych zależnych jest z pewnością test t-Studenta.

$$t = \frac{\bar{x}_{x_1-x_2}}{s_{x_1-x_2}}\sqrt{n} \tag{2.1}$$

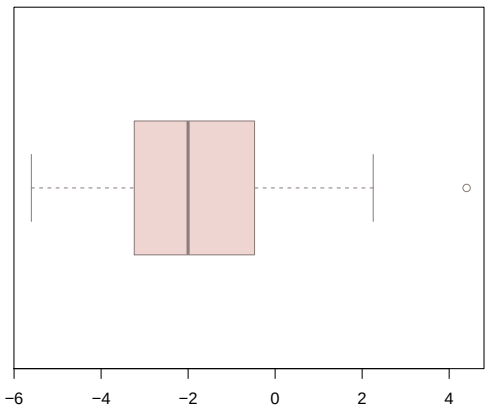
gdzie:

$\bar{x}_{x_1-x_2}$ to średnia dla różnic obserwacji,

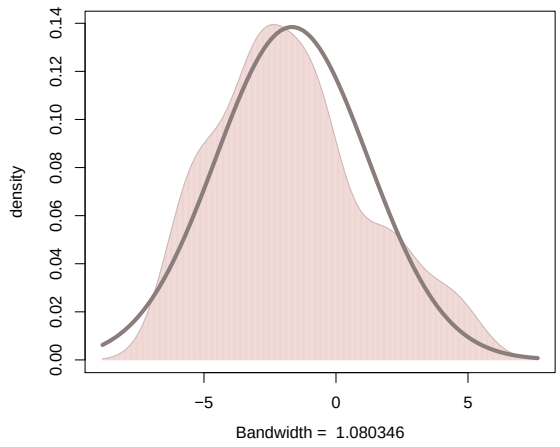
$s_{x_1-x_2}$ to odchylenie standardowe dla różnic obserwacji.

Jest to test parametryczny, a więc przed jego zastosowaniem należy zawsze sprawdzić pewne założenia np. normalność rozkładu.

```
# dane:
> set.seed(8974)
> a=rnorm(15,0,1);b=rnorm(15,0,4)
```



Rys. 2.1. Wykres pudełkowy dla różnic.



Rys. 2.2. Wykres gęstości dla różnic.

```
> shapiro.test(a-b)$p.value      # test Shapiro-Wilka
[1] 0.5597838
> library(nortest)
> sf.test(a-b)$p.value           # test Shapiro-Francia
[1] 0.5660159
```

```
# test t-Studenta:
> t.test(a,b,paired=T)

        Paired t-test

data:  a and b
t = -2.2432, df = 14, p-value = 0.04158
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.2644202 -0.0732495
sample estimates:
mean of the differences
              -1.668835
```

Korzystanie z pakietów statystycznych (np. R) jest dzisiaj normą, a więc bardziej zaawansowane obliczeniowo metody (np. test Yuena) powinny być częściej stosowane. Test Yuena jest przykładem ciekawej alternatywy dla testu t-Studenta. Procedura ta sprawdza się tak samo dobrze gdy założenia są spełnione oraz w przypadku gdy są naruszone np. występują obserwacje odstające.

Test Yuena:

$$t_Y = \frac{\bar{x}_{1t} - \bar{x}_{2t}}{\sqrt{\frac{n-1}{h(h-1)}\left(s_{w1}^2 + s_{w2}^2 - 2\rho_{w12}\right)}} \tag{2.2}$$

gdzie:

s_w^2 to wariancja winsorowska,

ρ_{w12} to kowariancja winsorowska,

Wartość h czyli wielkość próby po obcięciu jest obliczana na podstawie wzoru:

$$h = n - 2 \cdot \lfloor t \cdot n \rfloor \tag{2.3}$$

gdzie t oznacza frakcję jaka zastała przyjęta do obcięcia próby. Z kolei statystyka określona wzorem (2.2) ma rozkład t-Studenta z $df = h - 1$ stopniem swobody.

```
> library(WRS)
# test t-Studenta:
> yuend(a,b,tr=.0)$p.value
[1] 0.04157966
# test t-Studenta - bootstrap:
> ydbt(a,b,tr=.0,alpha=.05,nboot=10000,side=T)
[1] "Taking␣bootstrap␣samples.␣Please␣wait."
$ci
[1] -3.29332992 -0.04433973

$dif
[1] -1.668835

$p.value
[1] 0.0447
```

```
# test Yuena:
> yuend(a,b,tr=.2)$p.value
[1] 0.01625524
# test Yuena - bootstrap:
> ydbt(a,b,tr=.2,alpha=.05,nboot=10000,side=T)
```

```
[1] "Taking␣bootstrap␣samples.␣Please␣wait."
$ci
[1] -3.5802350 -0.5726989

$dif
[1] -2.076467

$p.value
[1] 0.0161
```

```
# moc testu t-Studenta:
> pT= replicate(10000,yuend(sample(a,15,T),sample(b,15,T),tr=.0)$p.↩
    value)
> mean(pT < 0.05)
[1] 0.5992
# moc testu Yuena:
> pY= replicate(10000,yuend(sample(a,15,T),sample(b,15,T),tr=.2)$p.↩
    value)
> mean(pY < 0.05)
[1] 0.609
```

2.2. Test Wilcoxona

Wśród testów nieparametrycznych wykorzystywanych do analiz zmiennych zależnych bardzo popularną metodą jest test rangowanych znaków Wilcoxona:

$$V_W^+ = \sum_{d_i>0} \text{rank}|d_i| \tag{2.4}$$

gdzie: d_i to różnice między obserwacjami po pominięciu różnic równych zero.

```
# dane:
> set.seed(254)
> A= sample(1:5,15,T); B= sample(1:5,15,T)
# statystyka Vw:
> d= A-B
> d0= d[d!=0]
> R= rank(abs(d0))
> Vw= sum(R[d0>0]); Vw
[1] 58
```

Jeśli liczebność próbki jest mniejsza niż 50 funkcja `wilcox.test(stats)` obliczy dokładną wartość $p-value$ ponieważ `exact=T` jest opcją domyślną. W przeciwnym wypadku (opcja `exact=F`) będzie wyznaczone asymptotyczne zbliżenie z korektą na ciągłość – `correct=T` to opcja domyślna.

Asymptotyczny test rangowanych znaków Wilcoxona:

$$z_{V_W} = \frac{\left[V_W^+ - \frac{1}{4}n(n+1)\right] \pm K}{\sqrt{\frac{1}{24}n(n+1)(2n+1) - \frac{1}{48}\sum_{i=1}^{c}(t_i^3 - t_i)}} \tag{2.5}$$

gdzie: n liczba par, V_W^+ to suma rang dodatnich, K to korekta na ciągłość która jest równa $\frac{1}{2}$, c to liczba grup pomiarów wiązanych, t_i to liczba pomiarów wiązanych w i-tej grupie pomiarów wiązanych.

Należy również podkreślić, że funkcja `wilcox.test(stats)` nie wyznaczy dokładnej wartości $p-value$ gdy w danych występują rangi wiązane lub wartości zerowe dla różnic.

```
> y=c(A,B); g=factor(rep(1:2,each=15)); b=factor(rep(1:15,2))
# asymptotyczny test Wilcoxona z korektą:
> wilcox.test(y~factor(g), paired=T)

        Wilcoxon signed rank test with continuity correction

data:  y by factor(g)
V = 58, p-value = 0.1373
alternative hypothesis: true location shift is not equal to 0

Komunikaty ostrzegawcze:
1: In wilcox.test.default(x = c(2L, 1L, 5L, 3L, 3L, 5L, 2L, 4L, 5L,:
  nie można obliczyć dokładnej wartości prawdopodobieństwa z więzami
2: In wilcox.test.default(x = c(2L, 1L, 5L, 3L, 3L, 5L, 2L, 4L, 5L,:
  nie można obliczyć dokładnej wartości prawdopodobieństwa z zerami
```

W takiej sytuacji lepszym rozwiązaniem jest skorzystanie z funkcji `wilcox.test` z pakietu `exactRankTests` lub funkcji `wilcoxsign_test` która jest dostępna w bibliotece `coin`:

```
# exact test Wilcoxona:
> library(exactRankTests)
> wilcox.exact(y~factor(g), paired=T)

        Exact Wilcoxon signed rank test

data:  y by factor(g)
V = 58, p-value = 0.1323
alternative hypothesis: true mu is not equal to 0
```

```
# exact test Wilcoxona:
>library(coin)
> wilcoxsign_test(y~g|b,distribution ="e",zero.method= "Wilcoxon")

       Exact Wilcoxon-Signed-Rank Test (zeros handled a la Wilcoxon)

data:  y by x (neg, pos)
         stratified by block
Z = -1.5261, p-value = 0.1323
alternative hypothesis: true mu is not equal to 0
```

Warto w tym miejscu podkreślić, że do wyznaczenia statystyki testu Wilcoxona funkcja `wilcoxsign_test` używa domyślnej opcji `zero.method="Pratt"`. Jest to metoda zaproponowana przez Pratta — rangi są ustalane na podstawie bezwzględnych różnic włącznie z zerami.

$$V_P^- = \sum_{d_i<0} \text{rank}|d_i| \tag{2.6}$$

gdzie: d_i to różnice między obserwacjami bez pomijania różnic równych zero.

```
# statystyka Vp:
> R= rank(abs(d))
> Vp= sum(R[d<0]); Vp
[1] 35
```

```
# exact test Wilcoxona - metoda Pratta:
> wilcoxsign_test(y~g|b,distribution ="e")

        Exact Wilcoxon-Signed-Rank Test (zeros handled a la Pratt)

data:  y by x (neg, pos)
         stratified by block
Z = -1.2723, p-value = 0.2334
alternative hypothesis: true mu is not equal to 0

Komunikat ostrzegawczy:
In wilcoxsign_test.IndependenceProblem(object = <S4 object of class ↩
   "IndependenceProblem">,  :
  Handling of zeros defaults to ‘Pratt’ in newer versions of coin
```

Asymptotyczny test rangowanych znaków Wilcoxona:

$$z_{V_P} = \frac{V_P^- - \frac{1}{4}\Big[n(n+1) - t_0(t_0+1)\Big]}{\sqrt{\frac{1}{24}\Big[n(n+1)(2n+1) - t_0(t_0+1)(2t_0+1)\Big] - \frac{1}{48}\sum_{i=1}^{c}(t_i^3 - t_i)}} \tag{2.7}$$

gdzie: n – liczba par, V_P^- – suma rang ujemnych, t_0 to ilość zerowych różnic, c to liczba grup pomiarów wiązanych, t_i to liczba pomiarów wiązanych w i-tej grupie pomiarów wiązanych.

```
# asymptotyczny test Wilcoxona - metoda Pratta:
> wilcoxsign_test(y~g|b)

  Asymptotic Wilcoxon-Signed-Rank Test (zeros handled a la Pratt)

data:  y by x (neg, pos)
         stratified by block
Z = -1.2723, p-value = 0.2033
alternative hypothesis: true mu is not equal to 0
```

Poniżej są wyniki asymptotycznego testu z zastosowaniem wzoru (2.5) bez korekty.

```
# asymptotyczny test Wilcoxona:
> wilcoxsign_test(y~g|b,zero.method = "Wilcoxon")

  Asymptotic Wilcoxon-Signed-Rank Test (zeros handled a la Wilcoxon)

data:  y by x (neg, pos)
         stratified by block
Z = -1.5261, p-value = 0.127
alternative hypothesis: true mu is not equal to 0
```

```
# asymptotyczny test Wilcoxona:
> wilcox.test(y~g,paired=T,correct=F)

        Wilcoxon signed rank test

data:  y by g
V = 58, p-value = 0.127
alternative hypothesis: true location shift is not equal to 0
```

Do porównań dwóch zmiennych zależnych ciekawą funkcję oferuje pakiet `afex`. Dzięki funkcji `compare.2.vectors` z opcją `paired=T` możemy otrzymać wynik kilku różnych testów: parametrycznego oraz nieparametrycznych.

```
# porównanie dwóch zmiennych zależnych:
> library(afex)
> compare.2.vectors(A,B,paired=T,coin=c("permutation","median"))
$parametric
  test test.statistic test.value test.df         p
1    t              t   1.581139      14 0.1361688

$nonparametric
             test test.statistic test.value test.df         p
1 stats::Wilcoxon              V  58.000000      NA 0.1372908
2     permutation              Z   1.507557      NA 0.1828100
3          median              Z  -1.133893      NA 0.4521300
```

Wartości $p - value$ dla dwóch ostatnich testów nieparametrycznych zostały wyznaczone za pomocą symulacji monte-carlo. Ilość symulacji jest określona za pomocą opcji `perm.distribution=approximate(100000)`. Jeśli chcemy wyznaczyć dokładne $p-value$ należy użyć polecenia `perm.distribution="exact"` z kolei dla testów asymptotycznych `perm.distribution="asymptotic"`. Metody te pochodzą z pakietu `coin`.

```
# symulacja monte-carlo - test permutacyjny:
> oneway_test(y~g|b, distribution= approximate(B= 100000))

        Approximative 2-Sample Permutation Test

data:  y by g (1, 2)
         stratified by b
Z = 1.5076, p-value = 0.1843
alternative hypothesis: true mu is not equal to 0
```

```
# symulacja monte-carlo - test mediany:
> median_test(y~g|b, distribution= approximate(B= 100000))

        Approximative Median Test

data:  y by g (1, 2)
         stratified by b
Z = -1.1339, p-value = 0.4536
alternative hypothesis: true mu is not equal to 0
```

```
# symulacja monte-carlo - test Wilcoxona (metoda Pratta):
> wilcoxsign_test(y~g|b, distribution= approximate(B= 100000))

  Approximative Wilcoxon-Signed-Rank Test (zeros handled a la Pratt)

data:  y by x (neg, pos)
         stratified by block
Z = -1.2723, p-value = 0.2341
alternative hypothesis: true mu is not equal to 0
```

3. Kilka zmiennych zależnych

3.1. Test ANOVA

Podobnie jak w innych testach parametrycznych także w teście ANOVA z pomiarami powtarzanymi muszą być spełnione pewne założenia np. normalność, kulistość. Jeśli można założyć, że warunek normalności został spełniony, trzeba upewnić się, co do sferyczności. Inaczej mówiąc trzeba zweryfikować hipotezę o równości wariancji różnic wśród wszystkich możliwych par eksperymentalnych. Zatem dla czterech powtarzanych pomiarów hipotezy badawcze będą miały postać:

$$\begin{aligned} H_0: &\quad \sigma^2_{1-2} = \sigma^2_{1-3} = \sigma^2_{1-4} = \sigma^2_{2-3} = \sigma^2_{2-4} = \sigma^2_{3-4} \\ H_1: &\quad \sigma^2_{1-2} \neq \sigma^2_{1-3} \neq \sigma^2_{1-4} \neq \sigma^2_{2-3} \neq \sigma^2_{2-4} \neq \sigma^2_{3-4} \end{aligned} \tag{3.1}$$

```
> set.seed(655)
> y=c(rnorm(10,2),rnorm(10,0),rnorm(10,2),rnorm(10,0))
> g=factor(rep(1:4,each=10))
> b=factor(rep(1:10,4))
> S=matrix(y,10,4)
# test sferyczności:
> mauchly.test(lm(S~1), X=~1)

        Mauchly's␣test␣of␣sphericity
␣␣␣␣␣␣␣␣Contrasts␣orthogonal␣to
␣␣␣␣␣␣␣␣~1

data:␣␣SSD␣matrix␣from␣lm(formula␣=␣S␣~␣1)
W␣=␣0.5085,␣p-value␣=␣0.3927
```

Ponieważ założenie dotyczące kulistości zostało spełnione (dla $\alpha = 0,05$) zastosujemy parametryczny test ANOVA z pomiarami powtarzanymi.

```
# test ANOVA:
> summary(aov(y~g+Error(b/g)))

Error: b
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  9  7.803   0.867

Error: b:g
          Df Sum Sq Mean Sq F value  Pr(>F)
g          3  20.64   6.881   6.379 0.00208 **
Residuals 27  29.13   1.079
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Po odrzuceniu hipotezy zerowej ($p-value = 0,00208$) o równości różnic średnich można zastosować test porównań wielokrotnych. Do porównania wszystkich różnic średnich zostanie wykorzystany test t-Studenta z poprawką Hochberga.

```
# porównaie wielokrotne parami:
> pairwise.t.test(y,g, paired= TRUE, p.adj= "hochberg")

        Pairwise comparisons using paired t tests

data:  y and g
```

```
  1      2      3
2 0.1378 -      -
3 0.3550 0.0041 -
4 0.1273 0.5941 0.0620

P value adjustment method: hochberg
```

Jeśli analizowane zmienne nie spełniają założenia sferyczności, to należy dostosować wyniki ANOVA za pomocą jednej z korekt: Greenhouse-Geisser [1958] lub Huynh and Feldt [1976]. Generalnie współczynnik korekcyjny HF jest używany częściej, ponieważ współczynnik GG jest zbyt konserwatywny tzn. nie zawsze udaje się wykryć prawdziwą różnicę między grupami.

```
# ANOVA, Greenhouse-Geisser, Huynh-Feldt:
> anova(lm(S~ 1), X= ~ 1, test= "Spherical" )
Analysis of Variance Table

Contrasts orthogonal to
~1

Greenhouse-Geisser epsilon: 0.6936
Huynh-Feldt epsilon:        0.9060

            Df      F num Df den Df    Pr(>F)    G-G Pr    H-F Pr
(Intercept)  1 6.3785      3     27 0.0020751 0.0072095 0.0030322
Residuals    9
```

Oprócz wartości $p - value$ (obliczonych na podstawie skorygowanych stopni swobody) podawane są także współczynniki ε–epsilon. Określają one odstępstwo od symetrii złożonej dla każdej z dwóch procedur: GG i HF. Im mniejsza wartość ε tym większe jest odstępstwo od warunku sferyczności.

```
# korekta Greenhouse-Geisser:
> eGG= 0.6936
> pf(6.3785,eGG*3,eGG*3*9,lower.tail=F)
[1] 0.007208134
# korekta Huynh-Feldt:
> eHF= 0.9060
> pf(6.3785,eHF*3,eHF*3*9,lower.tail=F)
[1] 0.003032141
```

Inne rozwiązanie to podejście wielowymiarowe z domyślną opcją `test="Pillai"`. Pozostałe warianty które możemy zastosować to: `"Wilks"`, `"Hotelling-Lawley"` oraz `"Roy"`. Jeśli założenie kulistość jest łamane, to w większości przypadków lepiej zastosować wielowymiarowe podejście.

```
# MANOVA:
> anova(lm(S~1), X=~1, test="Wilks")
Analysis of Variance Table

Contrasts orthogonal to
~1
            Df   Wilks approx F num Df den Df Pr(>F)
(Intercept)  1 0.22998   7.8124      3      7 0.0123 *
Residuals    9
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
# test ANOVA:
> library(car)
> G=factor(rep(1:4))
> a1=Anova(lm(S~1),idata=data.frame(G),idesign= ~G,type="III",test="↩
   Wilks")
> summary(a1, multivariate=FALSE)

Univariate Type III Repeated-Measures ANOVA Assuming Sphericity

                SS num Df Error SS den Df       F    Pr(>F)
(Intercept) 56.167      1   7.8031      9 64.7821 2.109e-05 ***
G           20.642      3  29.1256     27  6.3785  0.002075 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Mauchly Tests for Sphericity

  Test statistic p-value
G         0.5085 0.39269

Greenhouse-Geisser and Huynh-Feldt Corrections
 for Departure from Sphericity

   GG eps Pr(>F[GG])
G 0.69355   0.007209 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

   HF eps Pr(>F[HF])
G 0.90599   0.003032 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
# test MANOVA:
> a1

Type III Repeated Measures MANOVA Tests: Wilks test statistic
            Df test stat approx F num Df den Df    Pr(>F)
(Intercept)  1   0.12198   64.782      1      9 2.109e-05 ***
G            1   0.22998    7.812      3      7    0.0123 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Możemy posłużyć się też dużo wygodniejszym zapisem

```
# test ANOVA:
> library(afex)
> t= data.frame(y,g,b)
> univ(aov.car(y~g+Error(b/g),data=t ))
$anova
                 SS num Df  Error SS den Df         F       Pr(>F)
(Intercept) 56.16708      1  7.803137      9 64.782110 2.108576e-05
g           20.64205      3 29.125581     27  6.378532 2.075081e-03

$mauchly
  Test statistic   p-value
g      0.5084987 0.3926919

$sphericity.correction
```

```
     GG eps  Pr(>F[GG])    HF eps  Pr(>F[HF])
g 0.6935508 0.007209452 0.9059873 0.003032222
# test MANOVA:
> aov.car(y~g+Error(b/g),data=t )

Type III Repeated Measures MANOVA Tests: Pillai test statistic
            Df test stat approx F num Df den Df    Pr(>F)
(Intercept)  1   0.87802   64.782      1      9 2.109e-05 ***
g            1   0.77002    7.812      3      7    0.0123 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

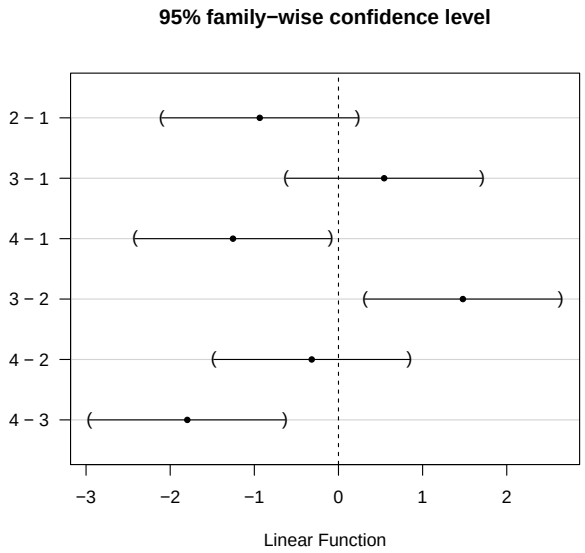
Często zdarza się, że do analizy danych z pomiarami powtarzanymi są wykorzystywane modele mieszane.

```
# model mieszany:
> library(nlme)
> m=lme(y ~ g, random = ~1|b/g,t)
> anova(m)
            numDF denDF  F-value p-value
(Intercept)     1    27 54.75454  <.0001
g               3    27  6.70764  0.0016
```

```
# kontrast Tukeya z poprawką Hochberga:
> library(multcomp)
> mg=glht(m, linfct = mcp(g="Tukey"))
> summary(mg,test= adjusted("hochberg"))

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Tukey Contrasts


Fit: lme.formula(fixed = y ~ g, data = t, random = ~1 | b/g)

Linear Hypotheses:
           Estimate Std. Error z value Pr(>|z|)
2 - 1 == 0  -0.9356     0.4529  -2.066 0.116594
3 - 1 == 0   0.5429     0.4529   1.199 0.461369
4 - 1 == 0  -1.2537     0.4529  -2.768 0.022569 *
3 - 2 == 0   1.4785     0.4529   3.264 0.005488 **
4 - 2 == 0  -0.3181     0.4529  -0.702 0.482513
4 - 3 == 0  -1.7966     0.4529  -3.966 0.000438 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Adjusted p values reported -- hochberg method)
```

```
# przedziały ufności:
> confint(mg,calpha= adjusted_calpha("hochberg"))

         Simultaneous Confidence Intervals

Multiple Comparisons of Means: Tukey Contrasts


Fit: lme.formula(fixed = y ~ g, data = t, random = ~1 | b/g)
```

```
Quantile = 2.5678
95% family-wise confidence level


Linear Hypotheses:
           Estimate lwr      upr
2 - 1 == 0 -0.93561 -2.09867  0.22745
3 - 1 == 0  0.54290 -0.62016  1.70596
4 - 1 == 0 -1.25370 -2.41676 -0.09064
3 - 2 == 0  1.47851  0.31545  2.64157
4 - 2 == 0 -0.31809 -1.48115  0.84497
4 - 3 == 0 -1.79660 -2.95966 -0.63354
# wykres:
plot(confint(mg,calpha= adjusted_calpha("hochberg")))
```

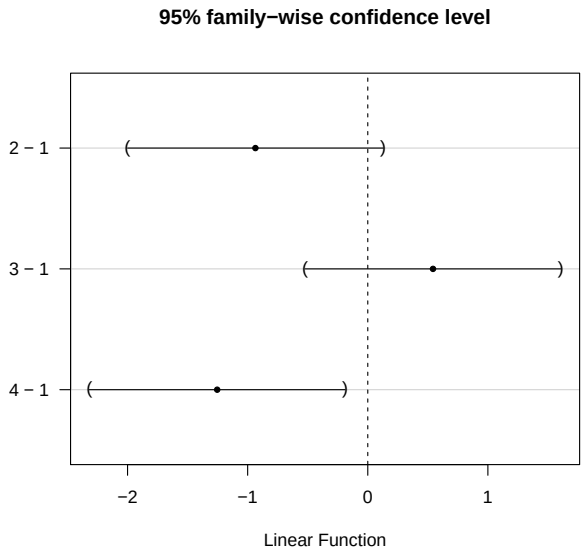


Rys. 3.1. Przedziały ufności dla kontrastu Tukeya z poprawką Hochberga.

Poniżej jest przedstawiony test Dunnetta (grupa pierwsza jest grupą referencyjną) z poprawką Westfalla:

```
# kontrast Dunnetta z poprawką Westfalla:
> MG=glht(m, linfct = mcp(g="Dunnett"))
> summary(MG,test= adjusted("Westfall"))

         Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts

Fit: lme.formula(fixed = y ~ g, data = t, random = ~1 | b/g)

Linear Hypotheses:
           Estimate Std. Error z value Pr(>|z|)
2 - 1 == 0  -0.9356     0.4529  -2.066   0.0712 .
3 - 1 == 0   0.5429     0.4529   1.199   0.2307
4 - 1 == 0  -1.2537     0.4529  -2.768   0.0159 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Adjusted p values reported -- Westfall method)
```

```
# przedział ufności:
> confint(MG,calpha= adjusted_calpha("Westfall"))

         Simultaneous Confidence Intervals

Multiple Comparisons of Means: Dunnett Contrasts

Fit: lme.formula(fixed = y ~ g, data = t, random = ~1 | b/g)

Quantile = 2.3492
95% family-wise confidence level

Linear Hypotheses:
           Estimate lwr     upr
2 - 1 == 0 -0.9356  -1.9997  0.1285
3 - 1 == 0  0.5429  -0.5212  1.6070
4 - 1 == 0 -1.2537  -2.3178 -0.1896
# wykres:
> plot(confint(MG,calpha= adjusted_calpha("Westfall")))
```



Rys. 3.2. Przedziały ufności dla kontrastu Dunnetta z poprawką Westfalla.

Do budowy modeli mieszanych można wykorzystać również funkcję `lmer(lme4)`.

```
# model mieszany:
> library(lme4)
> M=lmer(y~g+(1|b))
```

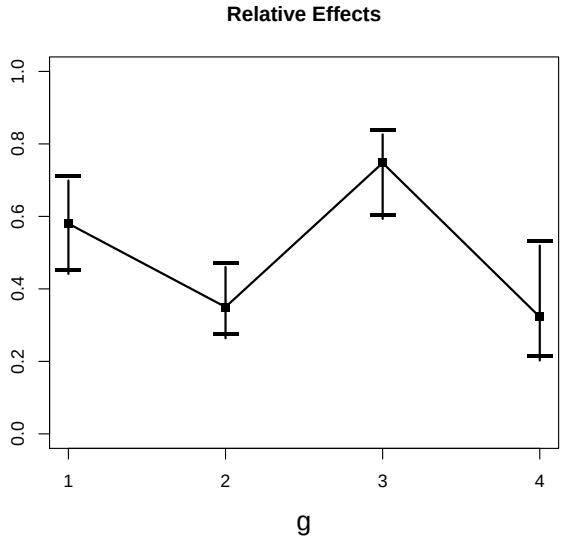
Gdy skorzystamy z funkcji `lme4` będziemy mieli możliwość skorzystania z pakietu `influence.ME` w celu wykrycia obserwacji wpływowych. Pakiet ten zapewnia kilka metod oceny wpływowych obserwacji np. DFBETAS, odległość Cooka. Należy w tym miejsce podkreślić, że funkcje z pakietu `influence.ME` działają tylko z funkcją `lmer(lme4)`. Więcej informacji na ten temat można znaleźć w dokumencie *influence.ME: Tools for Detecting Influential Data in Mixed Effects Models. The R Journal, 4(2):38-47, December 2012* autorstwa Rense Nieuwenhuis, Manfred te Grotenhuis, and Ben Pelzer.

Nieparametrycznym odpowiednikiem analizy wariancji z pomiarami powtarzanymi jest ANOVA-Type Statistics – ATS oraz Wald-Type Statistics – WTS. Test ATS

sprawdza hipotezę zerową, że względne efekty porównywanych grup (p_k) są takie same. Dla czterech grup ($k = 4$) badane hipotezy będą miały następującą postać:

$$\begin{aligned} H_0 &: \ p_1 = p_2 = p_3 = p_4 = 0,5 \\ H_1 &: \ p_1 \neq p_2 \neq p_3 \neq p_4 \neq 0,5 \end{aligned} \tag{3.2}$$

```
> library(nparLD)
> n=nparLD(y~g,data=t, subject=b, description=FALSE, plot.CI=T)
> summary(n)
Model:
LD F1 Model

Call:
y ~ g

Relative Treatment Effect (RTE):
   RankMeans Nobs    RTE
g1      23.7   10 0.5800
g2      14.5   10 0.3500
g3      30.4   10 0.7475
g4      13.4   10 0.3225

Wald-Type Statistc (WTS):
  Statistic df      p-value
g  34.48501  3 1.565011e-07

ANOVA-Type Statistc (ATS):
  Statistic       df      p-value
g  6.332924 2.353086 0.0009147741
```



Rys. 3.3. Przedziały ufności dla relatywnego efektu.

```
# przedziały ufności:
> nn=ld.ci(y=y, time=g, subject=b ,description=FALSE, alpha=0.05)
> nn
   Group Time Nobs RankMeans    RTE    Bias Variance  Lower  Upper
1 Group1    1   10      23.7 0.5800  0.0050   0.0467 0.4414 0.6989
```

```
2 Group1     2    10       14.5 0.3500 -0.0028    0.0261 0.2637 0.4605
3 Group1     3    10       30.4 0.7475  0.0003    0.0339 0.5932 0.8261
4 Group1     4    10       13.4 0.3225 -0.0025    0.0707 0.2025 0.5194
```

Warto także zwrócić uwagę na pakiet `WRS` za pomocą którego możemy skorzystać z kilku funkcji do analizy zmiennych zależnych. Poniżej przykład wykorzystania funkcji `rmanova(WRS)` dla średnich uciętych. Procedura ta jest wykorzystywana w sytuacji gdy jest naruszone założenie kulistości.

```
> library(WRS)
> rmanova(w, tr= 0.2)
[1] "The␣number␣of␣groups␣to␣be␣compared␣is"
[1] 4
$test
[1] 5.26926

$df
[1]  3 15

$siglevel
[1] 0.01107106

$tmeans
[1] 1.6780273 0.7447052 2.2475145 0.3055744

$ehat
[1] 0.8788141

$etil
[1] 1
```

```
> rmanova(w, tr=0.0)
[1] "The␣number␣of␣groups␣to␣be␣compared␣is"
[1] 4
$test
[1] 6.378532

$df
[1]  2.717962 24.461657

$siglevel
[1] 0.003032222

$tmeans
[1] 1.5965831 0.6609711 2.1394822 0.3428825

$ehat
[1] 0.6935508

$etil
[1] 0.9059873
```

Dokładny opis wszystkich funkcji z pakietu `WRS` można znaleźć w książce: *Introduction to Robust Estimation Hypothesis Testing* której autorem Rand R. Wilcox.

3.2. Test Friedmana

W środowisku R jest dostępnych wiele różnych pakietów w których są zaimplementowane nieparametryczne testy analizy wariancji. W pierwszej kolejności zostanie przedstawiony bardzo popularny test Friedmana.

$$\chi_F^2 = \frac{12}{nk(k+1)} \sum_{j=1}^{k} R_j^2 - 3n(k+1) \tag{3.3}$$

gdzie: n – to liczba bloków, k – to liczba grup, $\sum_{j=1}^{k} R_j^2$ – to suma kwadratów rang j-tej grupy.

W przypadku gdy występują rangi wiązane stosowany jest wzór:

$$\chi_{Ft}^2 = \frac{\chi_F^2}{1 - \frac{\sum_{i=1}^{c}(t_i^3 - t_i)}{nk(k^2-1)}} \tag{3.4}$$

gdzie: c – to liczba grup pomiarów wiązanych, t_i – to liczba pomiarów wiązanych w i-tej grupie pomiarów wiązanych.

Warto w tym miejscu zaznaczyć, że omawiany test jest często stosowany jako nieparametryczny odpowiednik jednoczynnikowej analizy wariancji dla pomiarów powtarzanych. Nie jest to jednak najlepsze rozwiązanie ponieważ metoda rang Friedmana jest rozszerzeniem testu znaków który ma zazwyczaj małą moc w stosunku do np. testu rangowanych znaków Wilcoxona.

Załóżmy, że dysponujemy danymi z pewnego eksperymentu w którym wzięło udział dziesięć osób - `b`. Interesuje nas odpowiedź na pytanie: czy średnie z każdej grupy - `g` różnią się na poziomie istotności $\alpha = 0,05$.

```
# dane:
> set.seed(197704)
> y= c(rexp(10,1),rnorm(10,0,1),rnorm(20,0,3))
> m= matrix(y,10,4)
> g= factor(rep(1:4,each=10)); b= factor(rep(1:10,4))
```

Na podstawie powyższych danych zostanie przeprowadzony test Friedmana jako nieparametryczny odpowiednik testu ANOVA z pomiarami powtarzanymi. W podstawowym pakiecie `stats` jest dostępna asymptotyczna wersja tego testu.

```
# asymptotyczny test Friedmana:
> friedman.test(y~g|b)

        Friedman rank sum test

data:  y and g and b
Friedman chi-squared = 8.76, df = 3, p-value = 0.03266
```

Jeśli odrzucimy hipotezę zerową (np. dla $\alpha = 0.05$) to możemy być zainteresowani które grupy różnią się między sobą dla danego poziomu α. Dla tak postawionego zadania wykorzystamy test rangowanych znaków Wilcoxona z poprawką Hochberga.

```
# porównania wielokrotne:
> pairwise.wilcox.test(y,g, p.adj="hochberg",paired=T)

        Pairwise comparisons using Wilcoxon signed rank test
```

```
data:  y and g

  1     2     3
2 0.168 -     -
3 0.109 0.168 -
4 0.193 0.068 0.035

P value adjustment method: hochberg
```

Warto również zwrócić uwagę na pakiet `coin` w którym wartość $p-value$ dla testu Friedmana można obliczyć za pomocą symulacji monte carlo.

```
> library(coin)
# asymptotyczny test Friedmana:
> friedman_test(y~g|b)

        Asymptotic Friedman Test

data:  y by g (1, 2, 3, 4)
         stratified by b
chi-squared = 8.76, df = 3, p-value = 0.03266
# symulacja Monte-Carlo:
> friedman_test(y~g|b, distribution= approximate(B= 100000))

        Approximative Friedman Test

data:  y by g (1, 2, 3, 4)
         stratified by b
chi-squared = 8.76, p-value = 0.03044
```

Ciekawym rozwiązaniem jest także wykorzystanie testu permutacyjnego.

```
# asymptotyczny test permutacyjny:
> oneway_test(y~g|b)

        Asymptotic K-Sample Permutation Test

data:  y by g (1, 2, 3, 4)
         stratified by b
maxT = 3.4943, p-value = 0.001876
# symulacja Monte-Carlo:
> oneway_test(y~g|b, distribution= approximate(B= 100000))

        Approximative K-Sample Permutation Test

data:  y by g (1, 2, 3, 4)
         stratified by b
maxT = 3.4943, p-value = 0.00094
```

Po wykonaniu komendy `?friedman_test` zostanie otworzony plik pomocy w którym jest dostępny kod do wykonania testu Wilcoxona-Nemenyi-McDonalda-Thompsona.

```
# kontrast Tukeya: Wilcoxon-Nemenyi-McDonald-Thompson test
> if(require("multcomp")) {
+   rtt <- symmetry_test(y ~ g | b, data = t,
+   teststat = "max",
+   xtrafo = function(data)
+   trafo(data, factor_trafo = function(x)
+   model.matrix(~ x - 1) %*% t(contrMat(table(x), "Tukey"))),
```

```
+   ytrafo = function(data)
+   trafo(data, numeric_trafo = rank, block = t$b))
+         print(pvalue(rtt))
+         print(pvalue(rtt, method = "single-step"))
+ }
[1] 0.02878098
99 percent confidence interval:
 0.02788892 0.02967304

2 - 1 0.72634901
3 - 1 0.22573820
4 - 1 0.82240323
3 - 2 0.82240406
4 - 2 0.22573058
4 - 3 0.02864417
```

W tej części opracowania zostanie zaprezentowany test Quade który jest kolejną alternatywą dla testu Friedmana. Jest on wykorzystywany w sytuacji gdy efekty blokowe znacznie różnią się wielkością (stosuje ważone rankingi) oraz założenie normalności jest naruszone. Warto też zaznaczyć, że test Quade jest rozszerzeniem testu rangowanych znaków Wilcoxona, a więc w wielu przypadkach ma większą moc niż metoda Friedmana.

Test Quade:

$$F_Q = \frac{(n-1)(\frac{1}{n})\sum_{j=1}^{k} S_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{k} S_{ij}^2 - (\frac{1}{n})\sum_{j=1}^{k} S_{ij}} \tag{3.5}$$

gdzie:

n – liczba bloków,
k – liczba grup,
R_{ij} – rangi obliczane dla każdego bloku,
Q_i – rangi dla różnic $x_{max} - x_{min}$ obliczanych dla każdego bloku,
S_{ij} – macierz o postaci:

$$S_{ij} = Q_i\left(R_{ij} - \frac{k+1}{2}\right) \tag{3.6}$$

oraz $df_1 = k - 1$ i $df_2 = (n-1)(k-1)$ to stopnie swobody.

```
# test Quade:
> quade.test(y~g|b)

        Quade test

data:  y and g and b
Quade F = 6.9574, num df = 3, denom df = 27, p-value = 0.001287
```

Po odrzuceniu hipotezy zerowej możemy wykonać test porównań wielokrotnych na podstawie poniższego wzoru:

$$|S_j - S_i| > t_{1-\alpha/2}\sqrt{\frac{2n\left[\sum_{i=1}^{n}\sum_{j=1}^{k} S_{ij}^2 - (\frac{1}{n})\sum_{j=1}^{k} S_{ij}\right]}{(n-1)(k-1)}} \tag{3.7}$$

Również pakiet `WRS` oferuje kilka ciekawych alternatyw dla testu Friedmana. Jedną z nich jest test Agresti–Pendergrast.

$$F_{AP} = \frac{n}{k-1}\mathbf{C}\bar{\mathbf{R}}^T\left[\mathbf{C}\mathbf{S}\mathbf{C}^T\right]^{-1}\mathbf{C}\bar{\mathbf{R}} \tag{3.8}$$

gdzie:

n – liczba bloków,

k – liczba grup,

$\bar{\mathbf{R}}^T$ – wektor średnich rang w grupach:

$$\bar{\mathbf{R}}^T = [\bar{R}_1 \dots \bar{R}_k] \tag{3.9}$$

$\mathbf{S}$ – macierz o postaci:

$$\mathbf{S} = \frac{(n-1)\cdot cov(\mathbf{R})}{n-k+1} \tag{3.10}$$

$\mathbf{C}$ – macierz o wymiarach $(k-1)\times k$:

$$\mathbf{C}_{(k-1)\times k} = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 & 0 \\ 0 & 1 & -1 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & -1 & 0 \\ 0 & 0 & 0 & \dots & 1 & -1 \end{bmatrix} \tag{3.11}$$

```
> apanova(w)
[1] "With␣n<=20,␣suggest␣using␣bprm"
$FTEST
         [,1]
[1,] 3.772023

$df1
[1] 3

$df2
[1] 27

$siglevel
           [,1]
[1,] 0.02210711
```

Ponieważ wielkość próbki jest mniejsza niż 20, pojawił się komunikat z propozycją aby wykonać test `bprm`.

```
> bprm(w)
$test.stat
[1] 4.204921

$nu1
[1] 2.412729

$p.value
[1] 0.009900732
```

Warto zaznaczyć, że test Agresti–Pendergrast jest dostępny także w bibliotece `asbio`.

```
> librasy(asbio)
> AP.test(m)
  df1 df2       F*       P(>F)
X   3  27 3.772023 0.02210711
```

Na zakończenie zostanie obliczona moc kilku testów.

```
> statystyka = NULL
> s=10000
> for (i in 1:s) {
+ m1=sample(m[,1],10,T)
+          m2=sample(m[,2],10,T)
+          m3=sample(m[,3],10,T)
+          m4=sample(m[,4],10,T)
+ statystyka[i] = friedman.test(cbind(m1,m2,m3,m4))$p.value
+ }
> mean(statystyka < 0.05)
[1] 0.5914               # moc testu Friedmana

> for (i in 1:s) {
+ m1=sample(m[,1],10,T)
+          m2=sample(m[,2],10,T)
+          m3=sample(m[,3],10,T)
+          m4=sample(m[,4],10,T)
+ statystyka[i] = quade.test(cbind(m1,m2,m3,m4))$p.value
+ }
> mean(statystyka < 0.05)
[1] 0.8842               # moc testu Quade

> for (i in 1:s) {
+ m1=sample(m[,1],10,T)
+          m2=sample(m[,2],10,T)
+          m3=sample(m[,3],10,T)
+          m4=sample(m[,4],10,T)
+ statystyka[i] = AP.test(cbind(m1,m2,m3,m4))[,4]
+ }
> mean(statystyka < 0.05)
[1] 0.7041               # moc testu Agresti-Pendergrast

> for (i in 1:s) {
+ m1=sample(m[,1],10,T)
+          m2=sample(m[,2],10,T)
+          m3=sample(m[,3],10,T)
+          m4=sample(m[,4],10,T)
+ statystyka[i] = bprm(cbind(m1,m2,m3,m4))$p.value
+ }
> mean(statystyka < 0.05)
[1] 0.691                # moc testu bprm
```