STATISTICKÁ ANALYSA DAT V R

Vojtěch Spiwok
Ústav biochemie a mikrobiologie
VŠCHT Praha
http://web.vscht.cz/spiwok/statistika

Srpen 2015

Předmluva

Tento text má sloužit pro účastnice a účastníky kurzu Statistické analýzy dat. Kurz se sestává z přednášek a ze cvičení v programu R. I když tento text obsahuje řadu pokusů o vysvětlení, rozhodně nemá ambici nahrazovat přednášky. Místo toho má být návodem pro cvičení v programu R, který obsahuje všechny důležité statistické funkce a navíc řadu funkcí pro zpracování čistě biologických dat.

Funkce programu R a jiné parametry jsou v textu zobrazeny neproporcionálním písmem, např. `plot`. Čtenář si může tyto příkazy zkopírovat a vložit je do prostředí R a tak si může jednotlivé funkce vyzkoušet. Je nutné mít na paměti, že řada funkcí R využívá náhodná čísla. Výsledky vámi provedených funkcí se tak mohou kvantitativně lišit od výsledků v textu; výsledné sdělení by ale mělo být stejné.

Některá funkce v programu R mohou být trochu jako „černé skříňky“. Například směrodatnou odchylku souboru `x` je možné vypočítat „ručně“ jako součet druhých mocnin odchylek, který následně vydělíme počtem hodnot mínus jedna a nakonec odmocníme. V programu R na to můžeme použít příkaz `sd(x)`. Z didaktických důvodů jsem se u vybraných funkcí pokusil nabídnout kromě funkce v R i „ruční“ postup. Týká se to například t-testu, metody ANOVA nebo analýzy hlavních komponent.

Text je doplněn o řadu obrázků, které byly vytvořeny v programu R pomocí funkce `png` s defaultním nastavením rozlišení. V důsledku toho je rozlišení obrázků poněkud malé. Domnívám se, že to není na škodu ze dvou důvodů: za prvé není pdf soubor tohoto učebního textu příliš velký a za druhé má čtenář možnost si většinu obrázků sám vygenerovat.

Další informace o programu R může čtenář nalézt na stránce *www.r-project.org*. Dalším cenným zdrojem je kniha *An Introduction to R* (autoři Venables, Smith a R Core Team), která je k dispozici jak v tištěné verzi, tak zdarma online (*https://cran.r-project.org/*

manuals.html). Existuje řada dalších knih věnovaných programu R nebo jeho speciálním aplikací a mnohé z nich je možné získat v elektronické verzi nebo půjčit v knihovně NTK a VŠCHT. V českém jazyce je programu R věnována dvojice knih *Moderní analýza biologických dat. 1. Zobecněné lineární modely v prostředí R* a *Moderní analýza biologických dat. 2. Lineární modely s korelacemi v prostředí R* autorů Pekára a Brabce, která vyšla v nakladatelství Scientia.

Obsah

Kapitola 1

Základy R

Program R vznikl kolem roku 1996 jako projekt odštěpený od programu S. Na rozdíl od svého předchůdce je „R-ko“ dostupné zdarma na adrese http://www.r-project.org v rámci licence General Public License (GPL). Tento fakt má dva pozitivní důsledky. Zaprvé to jsou nulové pořizovací náklady. Zadruhé, díky jeho otevřenosti je možné do programu přidávat různé balíčky. Proto se stal oblíbený v komunitě zaměřené na bioinformatiku, hlavně na zpracování dat z microarray experimentů, proteomiky a dalších -omik. Kromě toho se uplatňuje v dalších oborech a samozřejmě ve statistice a matematice. V základní verzi obsahuje nástroje pro statistické testy, lineární a nelineární regresi, klastrovou analýzu nebo analysu hlavních komponent. Silnou stránkou programu je tvorba grafů. Ty je možné uložit v různých vektorových a bitmapových grafických formátech s volitelným rozlišením. R rovněž obsahuje základní programovacích prvků (cykly, příkazy `for`, `while`, `if` atd.) Pro speciální použití je možné si celkem jednoduše a zdarma stáhnout různé balíčky, jako je například Bioconductor (http://www.bioconductor.org) pro zpracování microarray a podobných experimentů. Balíčky jsou dostupné na serveru CRAN (The Comprehensive R Archive Network, http://cran.r-project.org). Určitou nevýhodou programu je fakt, že je program ovládán pomocí příkazů a nikoliv klikáním v menu. Není tedy uživatelsky přívětivý jako různé „kilací“ programy. To ale může být i výhodou, neboť to nutí uživatele přemýšlet o tom co dělá, zatímco v klikacím statistickém programu se člověk může bezhlavě „doklikat“ k naprosto špatnému výsledku.

Uživatelé operačního systému MS Windows si mohou program stáhnou na výše zmíněných stránkách (http://www.r-project.org) a nainstalovat obvyklým způsobem. Program

spouštíme pomocí ikony v nabídce „Start“ nebo na ploše. Tím se spustí jednoduché uživatelské prostředí s příkazovou řádkou.

Uživatelé Linuxu mají tento program často nainstalován s vlastním operačním systémem. Pokud tomu tak není, mohou si jej stáhnout a nainstalovat dle instrukcí na stránkách. V Linuxu spouštíme program příkazem `R`. Na rozdíl od Windows se pouze změní podoba příkazové řádky. Případně je možné i pro Linux nalézt různá grafická prostředí pro R. Pokud někdo nemá zkušenosti s Linuxem a nerozumí předchozím dvěma větám, nechť je laskavě ignoruje.

Pokud jste nastartovali R, pak můžeme vyzkoušet první funkci. Do prostoru pro příkazy (za zobáček >) napište:

```
> q()
```

Pokud zmáčknete Enter, pak se vás program zeptá, jestli chcete uložit pracovní profil („Save workspace image?“). K významu tohoto dotazu se vrátíme později, teď zvolte odpověď „Ne“. Pokud tak učiníte, program se vypne a grafické prostředí zavře. Funkce `q()` totiž vypíná program R. Jejím synonymem je `quit()`. Závorka se uvádí proto, že se jedná o funkci a R používá zápis, který známe z matematicky, např. $f(x)$, $sin(x)$ a podobně. Protože na vypínání programu není nic sofistikovaného, je možné nechat závorku prázdnou, tedy bez argumentů. Pokud bychom si chtěli ukázat použití funkce s argumentem, můžeme si uvést příklad:

```
> q("n")
```

nebo

```
> q(save="n")
```

Program se vypne úplně stejně jako v případě pouhého `q()`, akorát s tím rozdílem, že se neptá na ukládání profilu. Volba `"n"` (jako No) znamená že nechceme uložit pracovní profil. K příkazům, které jsme již použili, se v prostředí R můžeme dostat pomocí šipek nahoru a dolu.

Většina funkcí v R má více argumentů, z nichž některé jsou hlavní a některé vedlejší. Například funkce `plot` nakreslí graf složený z bodů v prostoru os x a y. Hlavními argumenty jsou tedy série hodnot, jedna pro x a druhá pro y. Nejprve si vytvoříme hodnoty x a y (bude vysvětleno později):

```
> a <- 1:10
> b <- sin(a)
```

Pokud napíšeme:

```
> plot(a, b)
```

pak bude program podle pořadí vědět, že hodnoty *x* jsou v sérii a `a` hodnoty *y* jsou v sérii `b`. Příkladem vedlejšího argumentu může být argument `main`, který grafu přidá hlavní titulek. Například:

```
> plot(a, b, main="Graf")
```

nakreslí stejný graf s titulkem „Graf“. Příkaz:

```
> plot(a, b, "Graf")
```

nefunguje, neboť `main="Graf"` je vedlejší argument a program neví, že zrovna argument `main` má hodnotu „Graf“. Naopak příkaz:

```
> plot(x=a, y=b, main="Graf")
```

funguje normálně.

Výklad funkcí začneme nápovědou. Tu získáme příkazem `help()`, kde parametrem je název funkce. Samotné `help()` (totožné s `help(help)`) ukáže jak používat funkci `help`. Pokud používáte Windows, tak získáte nápovědu v internetovém prohlížeči. Pokud používáte Linux, pak se nápověda zobrazí přímo v okně. Stejný výsledek dostanete, když napíšeme otazník a za ním bez mezery název funkce, tedy `?plot` je totožné s `help(plot)`. Pokud nevíte jak se funkce nazývá, je možné použít:

```
> apropos("svd")
```

nebo

```
> help.search("svd")
```

která nám najde funkce, jenž obsahují v názvu nebo textu nápovědy řetězec „svd“.

Důležitou funkcí je funkce `example`, která ukáže příklady použití dané funkce. Podobnou funkcí je `demo`, která je ale dostupná jen pro vybrané kategorie. Zkuste:

```
> example(image)
> demo(graphics)
```

Jednotlivé obrázky je možné procházet pomocí klávesy Enter.

Po nápovědě se můžeme podívat na aritmetické operace. Pokud napíšeme:

```
> 1+1
[1] 2
```

program vypočte, že $1+1=2$. Význam závorky `[1]` souvisí s prací s vektory. Pokud by se jednalo o dlouhý vektor, který se nevejde na jeden řádek, pak může uživatel pomocí čísílek v závorce snadno zjistit z kolika prvků se vektor skládá. Pro odečítání se používá mínus (pomlčka), pro násobení hvězdička a pro dělení lomítko:

```
> 2-1
[1] 1
> 3*3
[1] 9
> 6/3
[1] 2
```

Pokud napíšeme:

```
> 5/2
[1] 2.5
```

tak získáme hodnotu 2,5, což asi není velké překvapení. Tento příklad uvádím proto, že některé programy a programovací jazyky vyžadují více nebo méně striktní odlišování celých a reálných čísel. Pokud provedeme něco podobného v programovacím jazyce Python, dostaneme výsledek:

```
>>> 5/2
2
```

Pro správný výsledek musíme napsat:

```
>>> 5.0/2.0
2.5
```

Program R rozpoznává typ čísel se všemi s tímto spojenými výhodami a nevýhodami. R-ko samozřejmě používá desetinou tečku, nikoliv čárku, která je českou záležitostí.

Kromě sčítání, odčítaní, násobení a dělení si můžeme uvést například mocniny `^`, modulo `%%` (zbytek po dělení) a dělení beze zbytku `%/%`.

```
> 3^3
[1] 27
> 5%%2
[1] 1
> 5%/%2
[1] 2
```

Pokud zmáčkneme Enter předčasně, program to rozpozná a čeká na dokončení příkazu:

```
> 1+
+
+ 1
[1] 2
```

Místo zobáčku se na příkazové řádce objeví znamínko plus. To můžeme využít pokud je nějaká funkce se všemi argumenty moc dlouhá a je vhodné pro přehlednost ji rozdělit na více řádků. Mezery navíc jsou ignorovány:

```
> 1+1
[1] 2
> 1 + 1
[1] 2
> 1 +       1
[1] 2
```

R dále obsahuje konstantu `pi` a základní matematické funkce:

```
> pi
[1] 3.141593
> cos(pi)
[1] -1
> sin(pi)
[1] 1.224606e-16
> exp(1)
[1] 2.718282
> abs(-4)
[1] 4
```

Přirozený logaritmus získáme funkcí `log()`, dekadický získáme funkcí `log10()` a dvojkový funkcí `log2()`:

```
> log(exp(2))
[1] 2
> log10(1000)
[1] 3
> log2(8)
[1] 3
```

Program umožňuje práci i s komplexními čísly, pokud to někoho v biologických vědách zajímá:

```
> 2i
[1] 0+2i
> 2i*2i
[1] -4+0i
```

Proměnou můžeme vytvořit a číslo k ní přiřadit následujícími způsoby:

```
> x <- 20
> x
[1] 20
> y <- 10
> y
```

```
[1] 10
> x+y
[1] 30
```

Kromě číselných hodnot mohou hodnotu funkcí nabývat logické hodnoty `TRUE` a `FALSE`, nebo hodnoty řetězce znaků:

```
> x<-FALSE
> x
[1] FALSE
> y<-"nazev"
> y
[1] "nazev"
```

Tvar <− představuje jakousi šipku. Podobně funguje obyčejné rovnítko:

```
> x = 20
> y = 10
> x+y
[1] 30
```

ale my jej nebudeme raději používat pro přiřazování hodnot, pouze v argumentech funkcí. Programátoři znají výrazy typu „x = x + 1“:

```
> x <- 10
> x <- x + 1
> x
[1] 11
```

tedy že se hodnota `x` se tímto výrazem zvýší o jednu.

Ohledně názvů proměnných je potřeba mít na paměti, že program rozlišuje velká a malá písmena:

```
> a<-1
> A<-2
> a
[1] 1
> A
[1] 2
> a+A
[1] 3
```

Proměnné není vhodné pojmenovávat „data“, neboť R obsahuje vzorová data (zkuste příkazy `help(data)` nebo `data()`). Osobně pokud chci, aby název proměnné obsahoval „data“, pak volím názvy jako „indata“, „mydata“, „mojedata“ a podobně. Dále není vhodné používat v názvech proměnných podtržítko.

Silnou stránkou R jsou operace s vektory. Zde se pojmem vektor myslí série několika čísel s daným pořadím. Počet nemusí být 2 nebo 3 pro dvou- nebo trojrozměrný prostor, ale mohou být vyšší, například odpovídat počtu měření. Vektor můžeme vytvořit příkazem:

```
> x <- c(1, 3, 2)
> x
[1] 1 3 2
```

často můžeme potřebovat vektor tvořený aritmetickou řadou, který získáme:

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
```

Výraz `10:1` vytvoří vektor s opačným pořadím prvků. K jednotlivým pozicím můžeme přistupovat pomocí hranatých závorek:

```
> x <- c(1,5,2,3,4,7)
> x
[1] 1 5 2 3 4 7
> x[1]
[1] 1
> x[2]
[1] 5
> x[3:6]
[1] 2 3 4 7
> x[c(1,3)]
[1] 1 2
```

Mezi další možnosti jak vytvořit různé uspořádané vektory patří následující funkce:

```
> seq(from=6, to=21, by=2)
[1]  6  8 10 12 14 16 18 20
> rep((1:4), times=2)
[1] 1 2 3 4 1 2 3 4
> rep((1:4), each=2)
[1] 1 1 2 2 3 3 4 4
```

S vektory je možné provádět různé operace jako násobení, dělení, přičítání a odečítání čísel atd:

```
> x<-1:5
> x
[1] 1 2 3 4 5
```

```
> x*2.5
[1]  2.5  5.0  7.5 10.0 12.5
> x/2.5
[1] 0.4 0.8 1.2 1.6 2.0
> x+2.5
[1] 3.5 4.5 5.5 6.5 7.5
> x-2.5
[1] -1.5 -0.5  0.5  1.5  2.5
```

Vektory se stejným počtem prvků můžeme samozřejmě sčítat a odečítat obvyklým způsobem:

```
> x<-c(1,3,2)
> y<-4:6
> x+y
[1] 5 8 8
```

Pokud vynásobíme dva vektory pomocí klasické hvězdičky, program vynásobí první číslo prvního vektoru prvním číslem druhého vektoru, druhé číslo prvního vektoru druhým číslem druhého vektoru a tak dále:

```
> x<-1:4
> y<-c(7,2,3,1)
> x*y
[1] 7 4 9 4
```

Pokud byste chtěli udělat skalární součin, pak je na to příkaz:

```
> x%*%y
     [,1]
[1,]   24
```

Výsledek je skalární součin v podobě matice s jedním sloupcem a řádkem (proto ta dekorace `[,1]` a `[1,]`). Podobně jako násobení funguje i dělení:

```
> 1:4/1:4
[1] 1 1 1 1
```

nebo funkce:

```
> x<-1:4
> exp(x)
[1]  2.718282  7.389056 20.085537 54.598150
```

Program umí rovněž pracovat s maticemi. Matici můžeme získat tímto příkazem:

```
> x<-matrix(1:12, ncol=3, byrow=TRUE)
> x
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> x<-matrix(1:12, ncol=3, byrow=FALSE)
> x
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

nebo jako skalární součin vektoru a transponovaného vektoru:

```
> 1:4
[1] 1 2 3 4
> t(1:4)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
> 1:4%*%t(1:4)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    2    4    6    8
[3,]    3    6    9   12
[4,]    4    8   12   16
```

kde `t()` je transpozice.

Vektory je možné spojovat do jakýchsi matic buď horizontálně nebo vertikálně pomocí příkazů `cbind` a `rbind` (pro collumn a row):

```
> x<-1:4
> y<-c(3,2,6,5)
> rbind(x, y)
  [,1] [,2] [,3] [,4]
x    1    2    3    4
y    3    2    6    5
> cbind(x, y)
     x y
[1,] 1 3
[2,] 2 2
```

```
[3,] 3 6
[4,] 4 5
```

K jednotlivým políčkům, sloupcům a řádkům je možné přistupovat následujícím způsobem:

```
> x<-1:4
> y<-c(3,2,6,5)
> xy <- cbind(x, y)
> xy
     x y
[1,] 1 3
[2,] 2 2
[3,] 3 6
[4,] 4 5
> xy[1,]
x y
1 3
> xy[1,1]
[1] 1
> xy[1,2]
[1] 3
> xy[,1]
[1] 1 2 3 4
> xy[,2]
[1] 3 2 6 5
```

Zvláštním a v případě statistického zpracování dat používaným objektem je `data.frame`:

```
> x<-c("a","a","b","b")
> y<-c(3,2,6,5)
> mydata <- data.frame(x,y)
> mydata
  x y
1 a 3
2 a 2
3 b 6
4 b 5
```

který umožňuje vytvořit jakousi matici z různých typů dat, například čísel a řetězců. K položkám se dostaneme tímto způsobem:

```
> mydata[1]
```

```
  x
1 a
2 a
3 b
4 b
> mydata[2]
  y
1 3
2 2
3 6
4 5
> mydata$x
[1] a a b b
Levels: a b
> mydata[1,1]
[1] a
Levels: a b
> mydata[2,1]
[1] a
Levels: a b
> mydata[2,]
  x y
2 a 2
```

Tvar `mydata$x` je specifický pro objekt `data.frame`. Dále je nutné zmínit, že program dokáže měnit mezi typy `vector`, `matrix`, `data.frame` pomocí funkcí `as.vector`, `as.matrix` respektive `as.data.frame`.

Program R umí pracovat s logickými funkcemi:

```
> x<-TRUE
> !x
[1] FALSE
> y<-FALSE
> x|x
[1] TRUE
> x|y
[1] TRUE
> y|y
[1] FALSE
> y&y
[1] FALSE
```

```
> x&y
[1] FALSE
> x&x
[1] TRUE
```

kde `&` je konjunkce, `|` disjunkce a `!` negace. Pro porovnávání můžeme použít:

```
> 1<2
[1] TRUE
> 1>2
[1] FALSE
> 1==1
[1] TRUE
> 1==2
[1] FALSE
```

kde == je logické porovnávání.

Jednoduchým příkladem cyklu v R může být:

```
> for(i in 1:3) {
+ print(i)
+ }
[1] 1
[1] 2
[1] 3
```

Kromě `for` umí R další programátorské konstrukce jako `if`, `while`, `repeat`, `break`, `next`, `ifelse` a `switch`. Funkce `print()` je dobrá do skriptů a programů, neboť při běžném používání programu R se k hodnotě *i* dostaneme jednoduše tak, že napíšeme `i`.

V programu R je možné definovat vlastní funkce pomocí funkce `function` a `return`. Zde je příklad:

```
> sinpluscos <- function(x) {
+ y<-sin(x)+cos(x)
+ return(y)
+ }
> sin(1)+cos(1)
[1] 1.381773
> sinpluscos(1)
[1] 1.381773
```

V programu je možné použít znak # pro komentář. Vše co následuje za tímto znakem je ignorováno:

```
> # ahoj
> 1 + 1 # + 1
[1] 2
```

Balíčky je možné spravovat pomocí příkazů:

```
> installed.packages()
           Package      LibPath               Version   Priority      Bundle
base       "base"       "/usr/lib/R/library" "2.5.1"   "base"        NA
boot       "boot"       "/usr/lib/R/library" "1.2-28"  "recommended" NA
...
waveslim   NA                                  "2.5.0"
wavethresh NA                                  "2.5.0"
```

Nový balíček nainstalujeme pomocí:

```
> install.packages("igraph")
```

Program vám dá vybrat k jakému úložišti balíčku R se chcete připojit. Fungování této funkce závisí na připojení k Internetu a na vašich uživatelských právech. Pokud si balíček nainstalujete a pak jej chcete aktivovat, musíte napsat:

```
> library(igraph)
```

Hned v první ukázce, konkrétně v případě vypínací funkce `q()`, jsme zamlčeli význam otázky „Save workspace image?“. Pokud během práce v R vytvoříme nějakou proměnnou, pak si bude program pamatovat její hodnotu, alespoň pokud ji nezměníme do konce naší R-kové seance. Pokud při vypínání programu R odpovíte na otázku „Save workspace image?“ kladně, pak si ji program bude pamatovat i v další seanci. To může mít výhody i nevýhody. Mezi nevýhody patří fakt, že si můžeme zaplevelovat proměnné. Naštěstí máme nástroje pro „úklid“ pracovního profilu programu. Například pomocí funkce `ls` se dostaneme k seznamu proměnných v profilu:

```
> ls()
[1] "X"
```

Určitou proměnnou je možné smazat pomocí funkce `rm`. Program rovněž obsahuje spoustu vzorových dat, od farmakokinetiky indomethacinu po počty přeživších na Titaniku podle kategorie cestujících (posádka/třetí/druhá/první třída, muži/ženy). Pro seznam zkuste funkci `data()`.

Kapitola 2

Vstup a výstup do souborů

Program R samozřejmě dokáže číst data ze souborů a do souborů data zapisovat. Pokud máme data uložené v textovém souboru `mojedata.txt` (ve Windows vytvořeném například programem Notepad), který má tento obsah:

```
1 2 3
2 3 4
```

pak jej můžeme načíst funkcí `read.table`:

```
> mojedata<-read.table(file="mojedata.txt", header=FALSE)
> mojedata
  V1 V2 V3
1  1  2  3
2  2  3  4
> mojedata[1,]
  V1 V2 V3
1  1  2  3
> mojedata$V1
[1] 1 2
> mojedata$V2
[1] 2 3
```

argument `header=FALSE` znamená, že první řádek neobsahuje názvy sloupců, ale rovnou data. Pokud bychom chtěli vyzkoušet volbu `header=TRUE`, pak by soubor `mojedata2.txt` musel vypadat asi takto:

```
a1 a2 a3
1  2  3
2  3  4
```

a jeho načtení by mohlo vypadat takto:

```
> mojedata<-read.table(file="mojedata2.txt", header=TRUE)
> mojedata
  a1 a2 a3
1  1  2  3
2  2  3  4
> mojedata$a2
[1] 2 3
```

V obou případech platí, že jsou jednotlivé položky oddělené mezerou nebo libovolným počtem mezer. Oddělovač je možné změnit argumentem `sep`. Pokud chceme načíst data oddělená tabulátorem, pak pro tento znak použijeme volbu `sep="\t"`. Kromě `header` a `sep` jsou dalšími užitečnými argumenty `dec`, kterým umožňuje načíst „česká“ data oddělená desetinou čárkou. Pokud se popisek sloupečku skládá z více slov, pak je nutné jej dát do uvozovek, aby si je program při načítání správně spojil. Kromě základní funkce `read.table` existují odvozené funkce `read.csv`, `read.csv2`, `read.delim`, `read.delim2`, `read.fwf` a `read.ftable`.

Alternativně je možné načíst data přímo ze schránky, z Excelového souboru, různých databasí (MySQL, SQLite, Oracle, Microsoft SQL Server a jiné) nebo formátu XML. R-ko obsahuje i balíčky pro načítání obrázků a jejich analysu.

Pro výstup do souboru je možné využít funkci `write.table`. Tu si můžeme ukázat tak, když si jeden z předchozích souborů načteme a pak jej uložíme do souboru `mojedata3.txt`:

```
> mojedata<-read.table(file="mojedata2.txt", header=TRUE)
> mojedata
  a1 a2 a3
1  1  2  3
2  2  3  4
> write.table(mojedata, file="mojedata3.txt")
```

Soubor `mojedata3.txt` vypadá takto:

```
"a1" "a2" "a3"
"1" 1 2 3
"2" 2 3 4
```

Funkci `write.table` můžeme ovládat pomocí podobných argumentů jako funkci `read.table`.

Pokud se vám nedaří funkce čtení a zápisu zprovoznit, pak může být problém ve špatném adresáři kde program hledá soubory. V prostředí Linuxu funkce `read.table`

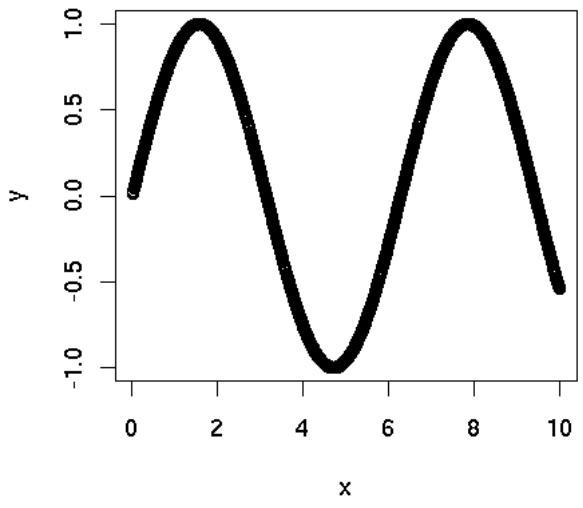
hledá a funkce `write.table` zapisuje soubory do adresáře z nějž byl program spuštěn. Samozřejmě je možné položit argument file rovný relativní nebo absolutní cestě k souboru. V instalacích programu R v počítačových učebnách VŠCHT ukládá a hledá soubory v adresáři „Documents“. I zde je možné zadat absolutní i relativní cestu, případně je možné zjistit adresář pro čtení a zápis pomocí funkce `getwd` a změnit pomocí `setwd`.

Kapitola 3

Grafy

Základní funkcí pro tvorbu grafů v R je `plot`. Pokud vytvoříme dva vektory o stejném počtu prvků, pak můžeme zobrazit závislost veličin uložených v obou vektorech:

```
> x<-1:1000/100
> y<-sin(x)
> plot(x,y)
```



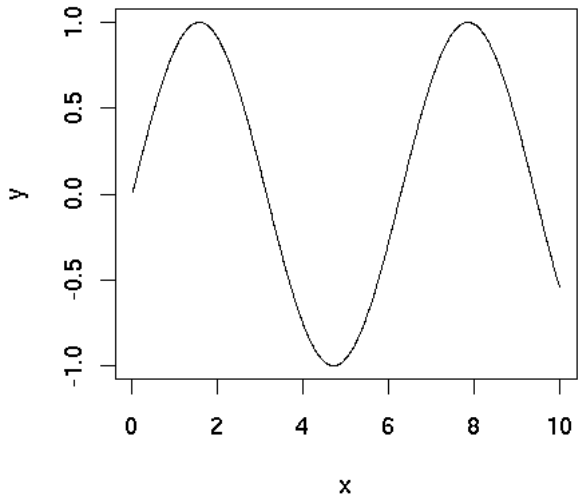
Obr. 3.1 Funkce `plot`

Místo dvou vektorů je možné použít jako argument jen jeden (`plot(y)`). Pak bude na horizontální ose pořadí ve vektoru *y*.

Pokud chceme místo puntíků zobrazit spojnice, použijeme argument `type="l"`:

```
> plot( x, y, type="l")
```

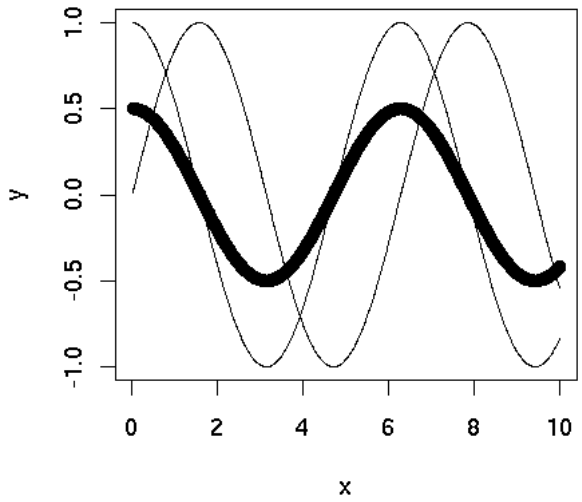
Celkem existuje několik typů grafů typu `plot`: `"p"` pro body (points), `"l"` pro linie, `"b"`, `"c"` a `"o"` pro různé kombinaci obou, `"h"` pro histogramový styl a `"s"` spolu



Obr. 3.2 Funkce `plot` s volbou `type="l"`

s `"S"` pro různé „schody“. Dále existuje volba `type="n"`, kdy jsou vykresleny pouze osy bez vlastního grafu. Pozorného čtenáře hned napadne otázka k čemu je to dobré. Ve většině případů chceme v grafech zobrazit ne jednu, ale několik veličin v závislosti na jedné nezávislé proměnné. Potom použijeme funkce `points` nebo `lines`. Nejdříve vytvoříme pomocí funkce `plot` graf s první závislostí. Okno s grafem nezavíráme. Pak do tohoto grafu můžeme přidat pomocí `lines` nebo `points` další závislosti:

```
> plot(x,y, type="l")
> lines(x,cos(x))
> points(x,0.5*cos(x))
```

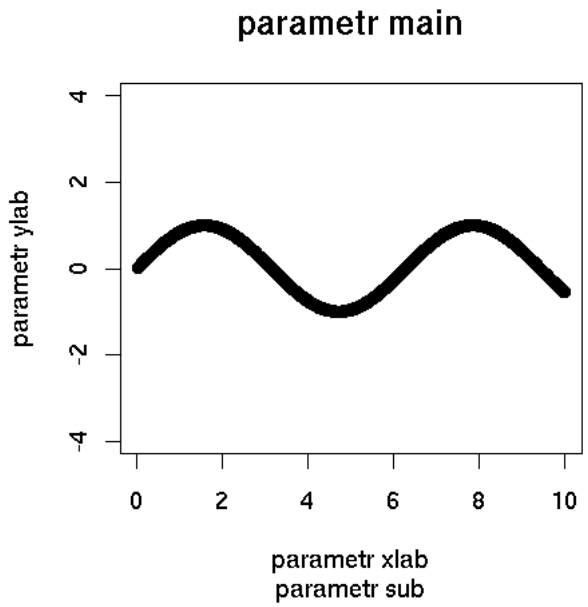


Obr. 3.3 Funkce `plot`, `points` a `lines`

Vycházet můžeme i z prázdného grafu vytvořeného právě s volbou `type="n"`.

Funkce plot nabízí argumenty `main`, `sub`, `xlab`, `ylab` a `asp`. První čtyři funkce představují horní a dolní titulek a názvy os *x* a *y*. Argument `asp` řídí poměr stran grafu:

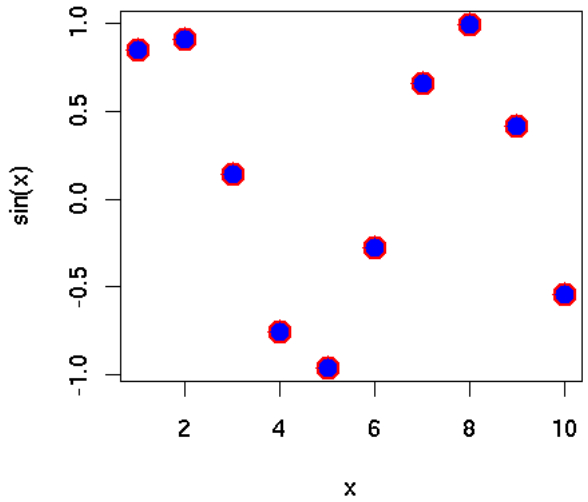
```
> plot( x, y, main="parametr main", sub="parametr sub",
+           xlab="parametr xlab", ylab="parametr ylab", asp=1)
```



Obr. 3.4 Funkce `plot` s různými názvy

U bodů (`points`) můžeme použít argument `pch`, který mění tvar bodu (vyzkoušejte různé hodnoty od 0 do 25). Argumenty `col` a `bg` mění barvu bodů. V případě vyplněného kolečka (`pch=21`) je barva každé kružnice daná argumentem `col` a výplň argumentem `bg`. Argument `cex` mění velikost bodů a `lwd` mění šířku čáry kružnice:

```
> x<-1:10
> plot(x, sin(x), pch=21, col="red", bg="blue", cex=2, lwd=2)
```



Obr. 3.5 Funkce `plot` s nastavením typů bodů a barev

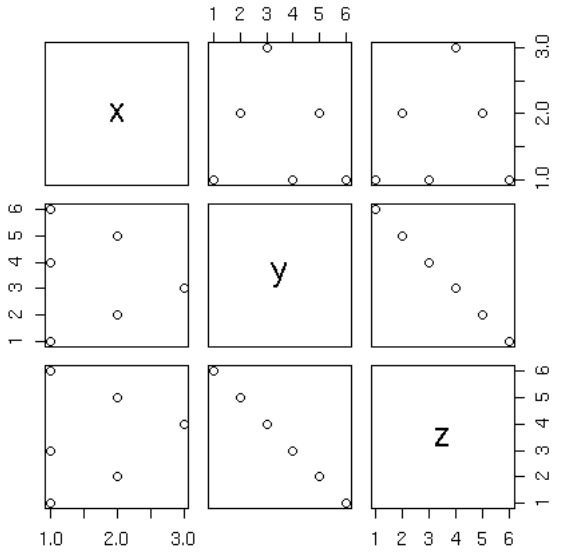
Parametr `col` a `lwd` se používají i u funkce `lines`.

Rozsah os můžeme ovlivnit parametry `xlim` a `xlim` (např.

`plot(1:10, xlim=c(0,100), ylim=c(-20,20))`).

Funkce `plot` může mít speciální význam společně s různými objekty, jako jsou histogramy, výsledky klastrové analysy, analysy hlavních komponent a další. Na začátek si můžeme ukázat použití funkce `plot` pro objekt `data.frame`:

```
> x<-c(1,2,3,1,2,1)
> y<-1:6
> z<-6:1
> xyz<-data.frame(x,y,z)
> plot(xyz)
```



Obr. 3.6 Funkce `plot` spolu s objektem `data.frame`

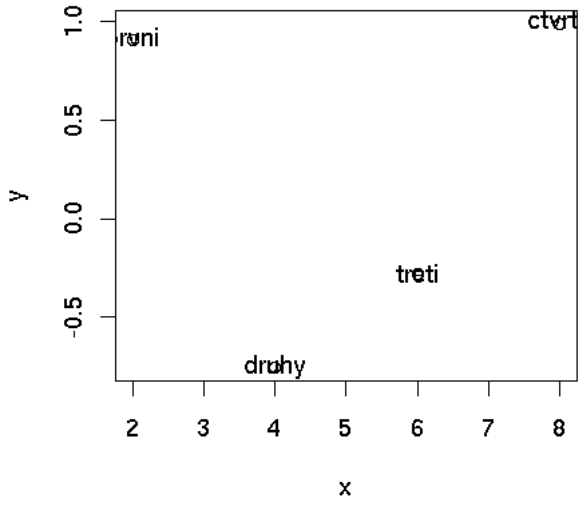
která zobrazí závislosti jednotlivých veličin na sobě. Jiná použití si ukážeme v dalších kapitolách.

Zajímavá je i funkce `text`, která umístí text do různých bodů v prostoru:

```
> x<-1:4*2
> y<-sin(x)
> pointnames<-c("prvni", "druhy", "treti", "ctvrty")
> plot(x,y)
> text(x, y, labels=pointnames)
```

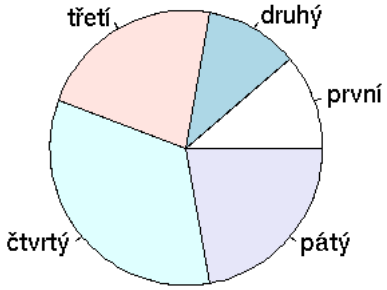
Pomocí argumentů této funkce můžeme ovlivnit font, barvu a jak bude bod posunut vůči vlastním polohám bodů (nad, pod, vpravo, vlevo).

R umí tvořit koláčové grafy:



Obr. 3.7 Funkce `text`

```
> x<-c(1,1,2,3,2)
> nam<-c("první","druhý","třetí","čtvrtý","pátý")
> pie(x, labels=nam)
```



Obr. 3.8 Funkce `pie`

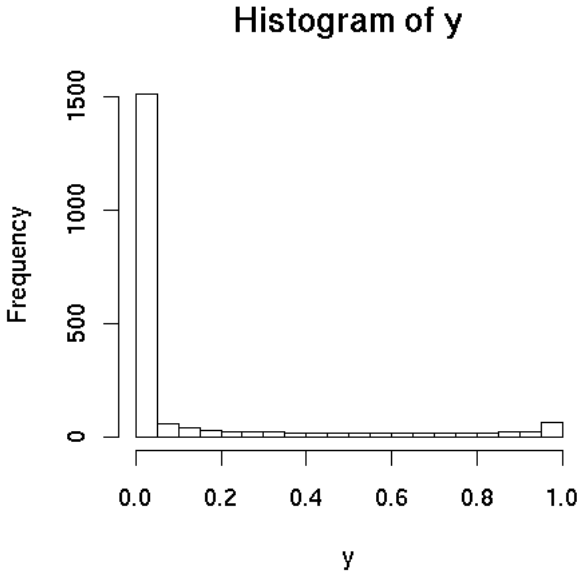
Sloupcový graf vytvoříme pomocí funkce `barplot`. Pokud chceme vytvářet sloupcový graf s chybovými úsečkami, pak je nutné použít například funkci `bargraph.CI` z balíčku `sciplot`. Stejný balíček obsahuje i funkci `lineplot.CI` pro liniový graf s chybovými úsečkami. Další balíček, který umožňuje zobrazit chybové úsečky, a kromě toho řadu hezkých grafických vizualizací, je `ggplot2`.

Dále v R-ku můžeme tvořit histogramy:

```
> x<- (-1000:1000)/10
> y<-exp(-x*x/200)
> hist(y,br=20)
```

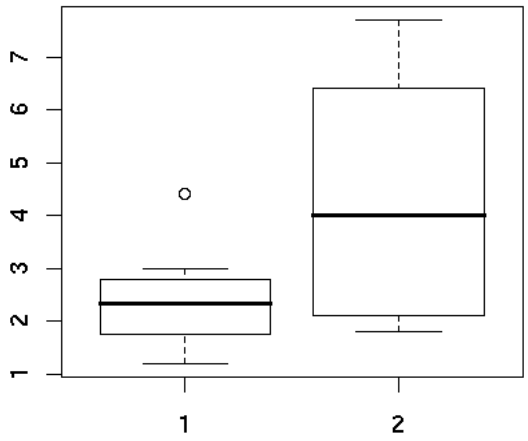
Speciálním typem grafu je krabicový graf (boxplot):

```
> x<-c(1.2,2.2,1.3,4.4,3.0,2.2,2.5,2.6)
```



Obr. 3.9 Funkce `hist`

```
> y<-c(3.3,2.3,1.8,5.5,7.7,7.3,1.9,4.7)
> boxplot(x, y)
```



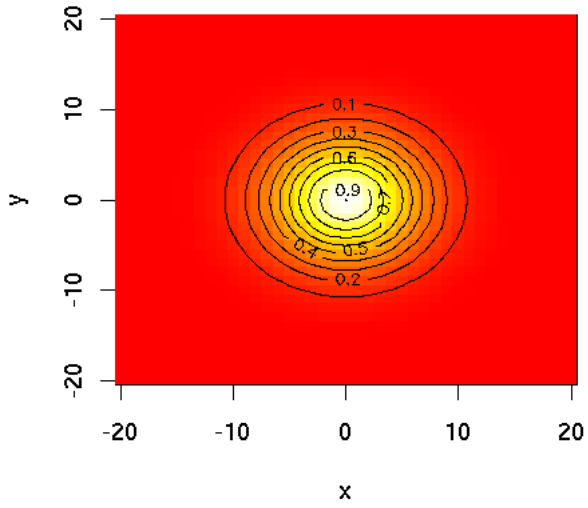
Obr. 3.10 Funkce `boxplot`

Význam jednotlivých části tohoto grafu si vysvětlíme v kapitole věnované popisné statistice.

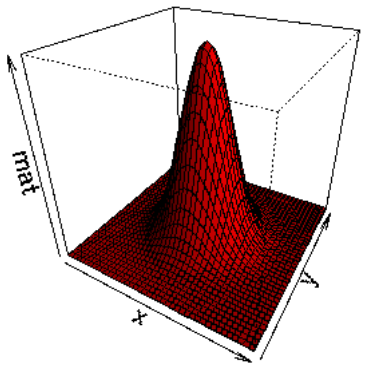
Bez dalšího vysvětlování si ukážeme funkce `image`, `contour` a `persp`:

```
> x<- -20:20
> y<- -20:20
> mat<-matrix(0,ncol=41,nrow=41)
> for (i in 0:40) {
+   for (j in 0:40) {
+     mat[i,j]=exp(-(x[i]*x[i]+y[j]*y[j])/50)
+   }
```

```
+ }
> image(x, y, mat, col=heat.colors(100))
> contour(x, y, mat, levels=seq(0, 1, by=0.1), add=TRUE)
> persp(x, y, mat, col="red",
+  theta=30, phi=30, shade=0.75, ltheta=100)
```



Obr. 3.11 Funkce `image` a `contour`



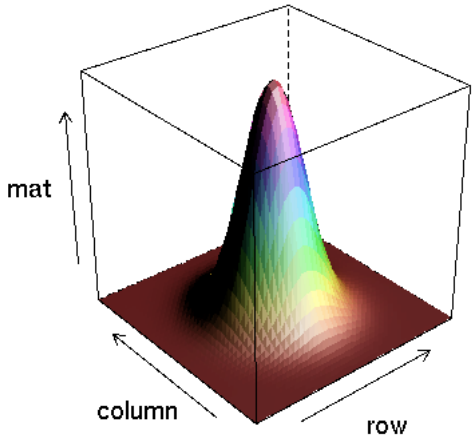
Obr. 3.12 Funkce `persp`

Ještě hezčí obrázky získáte s použitím balíčku `lattice`:

```
> library(lattice)
> wireframe(mat, shade=TRUE,light.source = c(10,0,10))
```

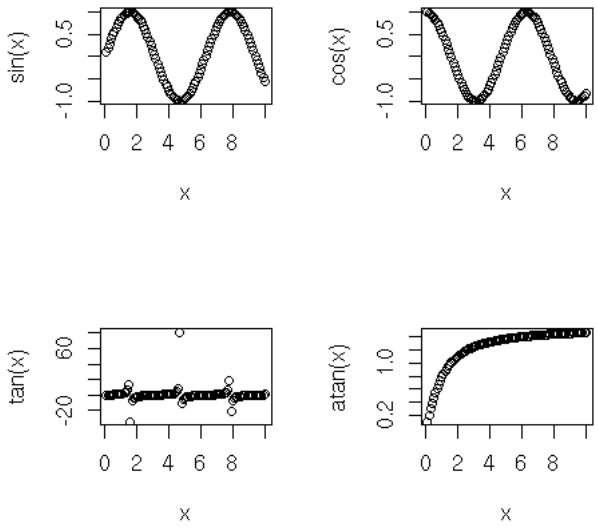
U grafů je možné měnit nastavení os pomocí funkce `axis`. Spoustu parametrů grafů je možné měnit pomocí funkce `par`. Pokud pomocí funkce `par` nastavíme nějaký parametr, tak všechny grafy, které od té doby vytvoříme budou mít takto pozměněný parametr. Jako nejužitečnější příklad si uveďme nastavení více grafů na jednom listu:

```
> par(mfrow=c(2,2))
> x<-1:100/10
```



Obr. 3.13 Funkce `wireframe` knihovny `lattice`

```
> plot(x, sin(x))
> plot(x, cos(x))
> plot(x, tan(x))
> plot(x, atan(x))
```



Obr. 3.14 Funkce `par`

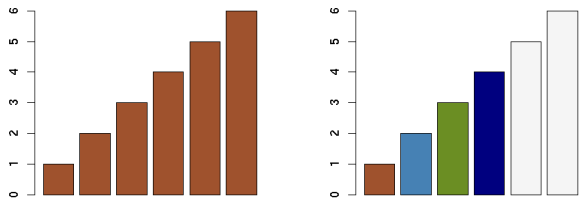
V popisu funkce plot bylo zmíněno, že je možné využívat různé barvy. R má velmi rozsáhlou paletu barev, které získáte funkcí `colors`]index`colors`. V mé instalaci se jednalo o 655 barev. Funkci `barplot` můžeme použít jak s jednou barvou sloupců:

```
> barplot(1:6, col="sienna")
```

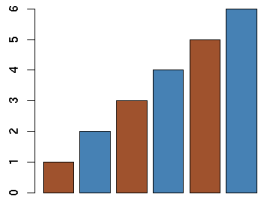
tak i s barvami specifikovanými pro jednotlivé sloupce:

```
> barplot(1:6, col=c("sienna", "steelblue", "olivedrab",
+                    "navy", "whitesmoke", "whitesmoke"))
```

Pokud je vektor barev menší než počet sloupků, pak se barvy opakují:



Obr. 3.15 Použití barev



Obr. 3.16 Použití barev

```
> barplot(1:6, col=c("sienna", "steelblue"))
```

Různé odstíny šedi je možné generovat pomocí funkce `gray` s argumentem v rozmezí 0-1:

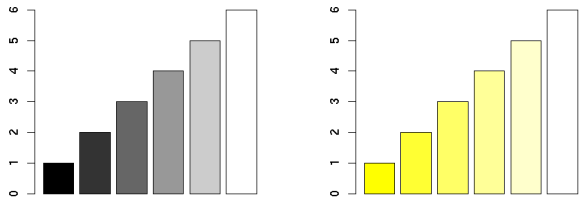
```
> x<-0:5/5
> gray(x)
[1] "#000000" "#333333" "#666666" "#999999" "#CCCCCC" "#FFFFFF"
> barplot(1:6, col=gray(x))
```

kde je výsledkem hexadecimální kód pro složky červené, zelené a modré. Například označení `#B2B2B2` značí, že intenzita červené je B2, tedy 11x16+2=178 z 256, tedy necelých 70 %. Stejná intenzita je i pro zelenou a modrou. Barvy v prostoru RGB je možné možné definovat i pomocí funkce `rgb`:

```
> rgb(1,1,x)
[1] "#FFFF00" "#FFFF33" "#FFFF66" "#FFFF99" "#FFFFCC" "#FFFFFF"
> barplot(1:6, col=rgb(1,1,x))
```

Atraktivní jsou rovněž palety barev `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors` a `cm.colors`.

Pokud vytvoříme nějaký graf na obrazovce počítače a jsme s ním spokojeni, pak je dobrý nápad si jej uložit do počítače ve vhodném grafickém formátu. Program R



Obr. 3.17 Použití barev `gray` a `rgb`

umožňuje ukládat obrázky v bitmapových formátech *png* a *jpeg* a ve vektorových formátech *pdf*, *svg* a *ps*. Pokud chceme obrázek uložit například ve formátu *png*, pak použijeme funkci `png` s argumentem, kterým je název souboru. Poté zopakujeme příkaz pro tvorbu grafu. V tomto případě se nám nezobrazí žádný obrázek, neboť se místo na obrazovku zapisuje do souboru. Nakonec vypneme zapisování do souboru pomocí funkce `dev.off()`:

```
> png("plot.png")
> barplot(1:6)
> dev.off()
null device
          1
```

Velikost obrázků a rozlišení je možné ovlivnit pomocí argumentů `width`, `height`, `res` a `pointsize`. Místo na disku kam se soubor zapíše se řídí podobnými pravidly jako vstup a výstup do souborů. Můžeme produkovat více obrázků zasobou. Pokud zadáme jako název souboru například `plot%03d.png` a funkci plot nebo podobnou použijeme několikrát, pak se v každém kroku uloží obrázek plot001.png, plot002.png a tak dále. To se může hodit pokud chcete udělat sérii obrázků, tu rozpohybovat jako animaci a dát třeba Youtube.

Kromě výše zmíněných typů grafů umí R-ko i různá exotická zobrazení. Zájemce najde ukázky například na http://gallery.r-enthusiasts.com/. Ke každé ukázce je zpravidla dostupný i kód. Program umí zobrazovat například trojúhelníkové diagramy. Data je možné zobrazit na různých geografických mapách pomocí balíčků `maps` a `maptools`. V R-ku je možné vytvářet i populární visualizace Word Cloud. Genomická data je možné získat balíčkem Genome Graphs nebo `ggbio`. Pro analysu sítí a grafů (ve významu matematické teorie grafů) je možné použít balíčky `SNA` a `igraph`.

Cílem této kapitoly bylo ukázat jak všestranný a silný nástroj pro přípravu grafů je program R. Letmý pohled do prestižních vědeckých časopisů ukazuje, že se popularita tohoto programu rozšiřuje a že výrazně ovlivňuje visuální a estetickou stránku presentace vědeckých dat.

Kapitola 4

Základy práce s daty

V této kapitole si ukážeme jak využít program R pro základní analýzu dat. Jako datový soubor si zvolíme žebříček 500 nejbohatších lidí v roce 2012 seřazených podle jména a příjmení. Ten najdete na stránce http://web.vscht.cz/spiwokv/statistika/forbes.txt. Tento soubor si můžete nahrát na svůj počítač a otevřít příkazem:

```
> forbes <- read.table("forbes2012.txt", header=T, sep=";")
```

Soubor bude nahrán jako objekt typu `data.frame`. Celý datový soubor, tedy všech 500 lidí s jejich příjmy, zeměmi původu a druhu podnikání, si můžete vytisknout tím, že napíšete jeho název:

```
> forbes
                              Name Age Billions          Source
1 Abdul Aziz Al Ghurair and family  58      2.9         banking
2                  Abigail Johnson  50     10.3        Fidelity
...
```

To není příliš praktické pro velké soubory, neboť se pak už nekouknete na před tím použité příkazy (s výjimkou šipky nahoru). Většinou užitečnější jsou příkazy `head` a `tail`, které zobrazí začátek respektive konec souboru, konkrétně prvních respektive posledních deset řádků:

```
> head(forbes)
> tail(forbes)
```

Další příkaz, který se často hodí, je `dim`, který obrazí počet řádků a sloupců souboru:

```
> dim(forbes)
[1] 500   6
```

Podobně, samotný počet řádků a sloupců získáme pomocí:

```
> nrow(forbes)
[1] 500
> ncol(forbes)
[1] 6
```

Funkce `length` je primárně určena pro vektory, ale funguje také pro objekt typu `data.frame` a vypíše počet sloupců:

```
> length(forbes)
[1] 6
```

Pokud chceme vypsat pouze určité řádky, sloupce nebo buňky, je možné je definovat v hranaté závorce za jménem objektu `data.frame`. Například když napíšeme:

```
> forbes[1]
                               Name
1 Abdul Aziz Al Ghurair and family
2                   Abigail Johnson
...
```

tak program vypíše celý první sloupec. Tento zápis raději pro `data.frame` nepoužívejte. Místo toho použijte:

```
> forbes[,1]
```

Naopak první řádek je možné vypsat pomocí:

```
> forbes[1,]
                               Name Age Billions  Source              Country
1 Abdul Aziz Al Ghurair and family  58      2.9 banking United Arab Emirates
  Industry
1  Finance
...
```

Buňku na prvním řádku a v prvním sloupci získáme:

```
> forbes[1,1]
[1] Abdul Aziz Al Ghurair and family
500 Levels: Abdul Aziz Al Ghurair and family ... Zong Qinghou
```

Pokud chceme vytisknout první tři řádky, je možné napsat:

```
> forbes[1:3,]
                              Name Age Billions   Source              Country
1 Abdul Aziz Al Ghurair and family  58      2.9  banking United Arab Emirates
2                  Abigail Johnson  50     10.3 Fidelity        United States
3          Abilio dos Santos Diniz  75      3.6   retail               Brazil
            Industry
1            Finance
2           Business
3 Fashion and Retail
```

Řádky typu `data.frame` mají své sloupce pojmenované. Pokud se chcete dostat k hodnotám nějakého sloupce, můžete k tomu využít buď číslo sloupce jak bylo právě ukázáno, nebo jejich jména. Jména sloupců si může uživatel vypsat příkazem `names`:

```
> names(forbes)
[1] "Name"      "Age"       "Billions" "Source"    "Country"   "Industry"
```

Jeden ze sloupců je první sloupec s názvem "Name", který označuje jméno boháče. Místo `forbes[,1]` můžeme použít příkaz:

```
> forbes[,"Name"]
[1] Abdul Aziz Al Ghurair and family Abigail Johnson
[3] Abilio dos Santos Diniz          Akira Mori and family
...
500 Levels: Abdul Aziz Al Ghurair and family ... Zong Qinghou
```

V případě objektu `data.frame` je možné místo hranatých závorek použít znak dolaru:

```
> forbes$Name
```

Výsledkem je vektor, takže si můžeme vypsat jeho první tři hodnoty:

```
> forbes$Name[1:3]
[1] Abdul Aziz Al Ghurair and family Abigail Johnson
[3] Abilio dos Santos Diniz
500 Levels: Abdul Aziz Al Ghurair and family ... Zong Qinghou
```

Pokud by nás zajímalo, v jakých zemích boháči sídlí, tak si můžeme samozřejmě vypsat odpovídající sloupec. To ale není příliš praktické, protože některé země budou vypsány mnohokrát. Pokud chceme vypsat seznam zemí původu boháčů tak, aby tam byla každá zvlášť, pak je možné použít příkaz `levels`:

```
> levels(forbes$Country)
 [1] "Argentina"            "Australia"            "Austria"
...
[52] "United States"        "Venezuela"
```

Pokud chceme zjistit počet zemí, můžeme použít funkci `nlevels`:

```
> nlevels(forbes$Country)
[1] 53
```

Seznam zemí s jejich zastoupením je možné si vypsat funkcí `table`:

```
> table(forbes$Country)

           Argentina            Australia              Austria
                   1                    7                    4
...
       United States            Venezuela
                 168                    2
```

Pokud nás zajímá jaký je rozsah majetku, tedy jaký je nejmenší a největší majetek v souboru, můžeme se podívat pomocí funkce `range`:

```
> range(forbes$Billions)
[1]  2.5 69.0
```

Pokud bychom chtěli vypsat boháče, můžeme to udělat takto:

```
> forbes[forbes[,"Country"]=="Czech Republic",]
            Name Age Billions             Source        Country Industry
365 Petr Kellner  48      8.2 banking, insurance Czech Republic  Finance
```

Všimněte si, že zde máme objekt `data.frame` s názvem forbes a za ním hranatou závorku. V ní máme napsáno před čárkou `forbes[,"Country"]=="Czech Republic"]`. Pokud napíšete samotný tento výraz, pak vám program vypíše vektor s logickými hodnotami `TRUE` a `FALSE`. U českého boháče byste našli `TRUE`, u ostatních `FALSE`. Tento výraz je v hranaté závorce před čárkou a za čárkou není nic, tedy program vypíše celé řádky, pro které má vnitřní výraz hodnotu `TRUE`. Podobně si člověk může vypsat všechny boháče, jejichž majetek je větší či menší než vybraná částka:

```
> forbes[forbes[,"Billions"]>40,]
                          Name Age Billions             Source       Country
51              Bernard Arnault  63       41               LVMH        France
56                   Bill Gates  56       61          Microsoft United States
66  Carlos Slim Helu and family  72       69            telecom        Mexico
480              Warren Buffett  82       44 Berkshire Hathaway United States
              Industry
51  Fashion and Retail
56          Technology
66             Telecom
480        Investments
```

Majetky si můžeme seřadit pomocí funkce `sort` od nejvyššího po nejnižší:

```
> sort(forbes$Billions)
  [1]  2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.5  2.6  2.6  2.6
...
[496] 37.5 41.0 44.0 61.0 69.0
```

nebo od nejnižšího po nejvyšší:

```
> sort(forbes$Billions, decreasing=T)
  [1] 69.0 61.0 44.0 41.0 37.5 36.0 30.0 26.0 25.5 25.4 25.3 25.0 25.0
...
[496]  2.5  2.5  2.5  2.5  2.5
```

Tím získáme setříděný seznam majetku, ale ztratíme informace o tom komu patří. Pokud chceme celý seznam setříděný podle majetku, můžeme použít funkci `order`, která nám (s volbou `decreasing=T`) poskytne pořadí na jakém místě se daný boháč nachází. Když bude první boháč v seznamu například 105 nejbohatším člověkem, pak první prvek výsledného vektoru bude 105. Pak je možné řádky objektu `data.frame` přeházet tak, abychom získali boháče od nejchudšího po nejbohatšího:

```
> forbes[order(forbes$Billions),]
                    Name Age Billions                              Source
45          Bahaa Hariri  46      2.5 real estate, investments, logistics
46  Barbara Carlson Gage  70      2.5                 hotels, restaurants
...
```

a naopak od nejbohatšího po nejchudšího:

```
> forbes[order(forbes$Billions, decreasing=T),]
                           Name Age Billions             Source       Country
66  Carlos Slim Helu and family  72     69.0            telecom        Mexico
56                  Bill Gates  56     61.0          Microsoft United States
...
```

Poslední na co se koukneme je zacházení s chybějícími daty. V souboru boháčů u některých chyběl jejich věk a místo věku byla uvedena pomlčka. Pokud použijete například funkci `boxplot` na sloupeček *Age*, pak program nahlásí chybu. Program R používá jako defaultní hodnotu pro chybějící údaj symbol `NA` jako *not available*. Pokud chceme, aby se pomlčky načetly jako chybějí data, pak musíme použít při načítání dat funkci `read.table` s volbou `na.strings="-"`:

```
> ifile <- read.table("forbes2012.txt", header=T, sep=";", na.strings="-")
> head(ifile)
                             Name Age Billions          Source
1 Abdul Aziz Al Ghurair and family  58      2.9         banking
2                  Abigail Johnson  50     10.3        Fidelity
3          Abilio dos Santos Diniz  75      3.6          retail
4            Akira Mori and family  76      3.5     real estate
5      Alain and Gerard Wertheimer  NA      7.5          Chanel
6         Alain Merieux and family  74      3.7 pharmaceuticals
              Country           Industry
1 United Arab Emirates            Finance
2        United States           Business
3               Brazil Fashion and Retail
4                Japan        Real Estate
5               France Fashion and Retail
6               France        Health care
> boxplot(ifile$Age)
```

Pak bude funkce `boxplot` fungovat na dostupných datech a chybějící data budou ignorována.

Kapitola 5

Náhodná čísla v R a jejich rozdělení

V programu R je k dispozici celá řada funkcí pro generování náhodných čísel s různým rozdělením. My můžeme tyto funkce použít pro generování modelových výsledků měření a na nich si ukazovat jak fungují statistické metody. Ve statistice nás bude nejvíce zajímat normální rozdělení. Sérii náhodných čísel s normálním rozdělením si může vygenerovat pomocí funkce `rnorm`:

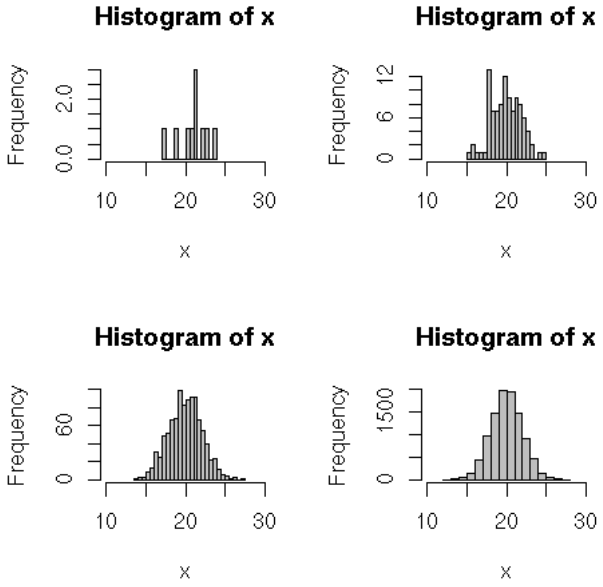
```
> rnorm(10, mean=20, sd=2)
 [1] 20.44410 21.05293 23.13803 23.63433 20.19606 22.21550 18.78641 19.04648
 [9] 22.31397 21.86754
```

kde 10 je počet vygenerovaných čísel, `mean` je střední hodnota a `sd` je směrodatná odchylka. Pokud si tuto funkci vyzkoušíte sami, pak pochopitelně dostanete jiná čísla. Nyní si vyzkoušíme vytvořit grafy a histogramy pro různé počty vygenerovaných čísel:

```
> x<-rnorm(10, mean=20, sd=2)
> hist(x, br=20, xlim=c(10,30), col="gray")
> x<-rnorm(100, mean=20, sd=2)
> hist(x, br=20, xlim=c(10,30), col="gray")
> x<-rnorm(1000, mean=20, sd=2)
> hist(x, br=20, xlim=c(10,30), col="gray")
> x<-rnorm(10000, mean=20, sd=2)
> hist(x, br=20, xlim=c(10,30), col="gray")
```

Je vidět, že s přibývajícím počtem bodů se průběh funkce přibližuje k ideální Gaussově křivce.

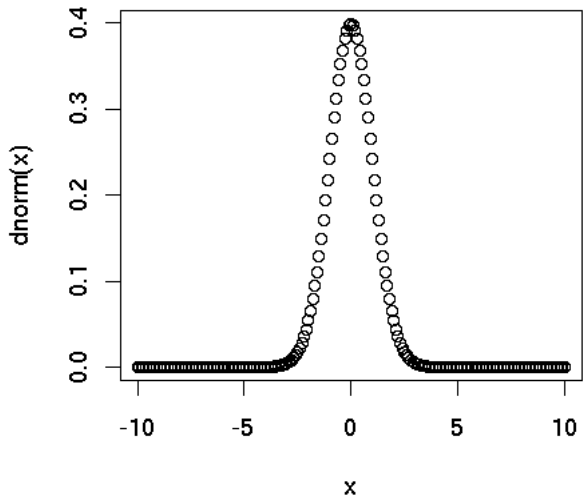
Normálního rozdělení se ještě týkají funkce `dnorm`, `pnorm` a `qnorm`. První z nich vrací hustotu rozdělení (density). Pokud napíšeme například `dnorm(0.7)`, pak nám funkce vrátí hodnotu 0,3122539. To znamená, že pokud bychom provedli měření veličiny x, která má střední hodnotu rovnou nule a směrodatnou odchylku rovnou jedné (defaultní



Obr. 5.1 Normální rozdělení 10, 100, 1 000 a 10 000 čísel

nastavení, jinak nutné použít argumenty `mean` a `sd`), pak pravděpodobnost, že naměříme hodnotu mezi 0,7 a 0,7 + δx, je rovná 0,3122539 $\times \delta x$. Profil si můžeme vykreslit:

```
> x<--100:100/10
> plot(x, dnorm(x))
```



Obr. 5.2 Normální rozdělení – funkce `dnorm`

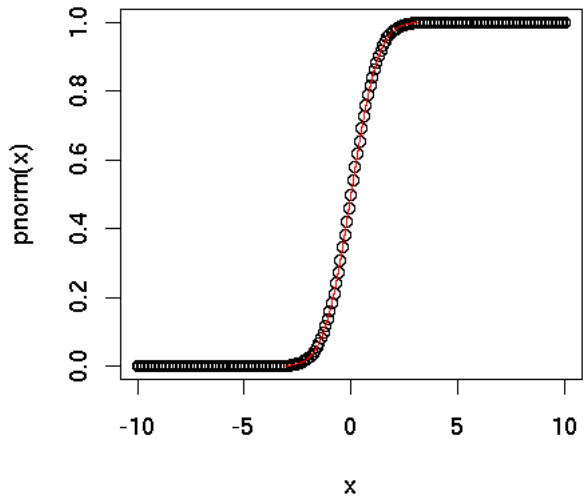
Funkce `pnorm` zobrazuje distribuční funkci, která je integrálem hustoty rozdělení. To si můžeme ukázat jednoduchou numerickou integraci lichoběžníkovou metodou pro hodnotu:

```
> x<- -1000:70/100
> 0.01*sum(dnorm(x))
[1] 0.7595958
```

```
> pnorm(0.7)
[1] 0.7580363
```

Odchylka je způsobena nepřesností numerické metody. Tato funkce představuje kumulativní pravděpodobnost. Hodnota `pnorm(0.7)` tedy představuje pravděpodobnost, že pro naši veličinu naměříme hodnotu od mínus nekonečna do 0,7. Funkce `qnorm` – kvantil normálního rozdělení – je inverzní funkcí k `pnorm`. Tato funkce nám naopak vrátí hodnotu měření pro danou kumulativní pravděpodobnost. To, že se jedná o inverzní funkci, můžeme ukázat například takto:

```
> x<--100:100/10
> probs<-1:999/1000
> plot(x, pnorm(x))
> lines(qnorm(probs), probs, col="red")
```



Obr. 5.3 Normální rozdělení – funkce `pnorm` a `qnorm`

Všimněte si, že ve funkci `lines` je nejprve `qnorm(probs)` a pak `probs`, díky čemuž získáme graf inverzní funkce. Pro další statistická rozdělení má program R funkce `dchisq`, `pchisq`, `qchisq` a `rchisq` pro rozdělení chi-kvadrát, `dt`, `pt`, `qt` a `rt` pro Studentovo t-rozdělení a `df`, `pf`, `qf` a `rf` pro F-rozdělení.

Kapitola 6

Popisná statistika

Nyní vyzkoušíme funkce popisné statistiky. Popisná (nebo také deskriptivní) statistika se snaží pomocí několika veličin popsat vlastnosti souboru, například výsledků měření. Základní parametry popisné statistiky získáme pomocí funkce `summary`:

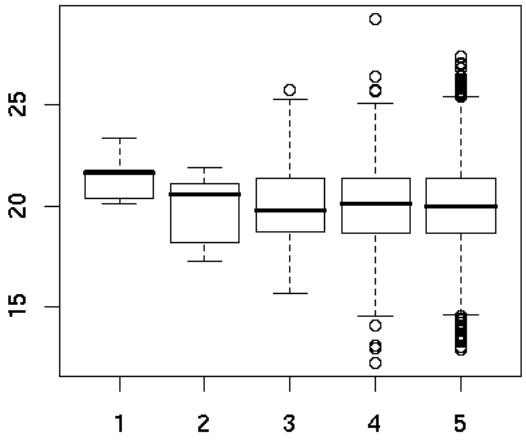
```
> x<-rnorm( 10, mean=20, sd=2)
> x
 [1] 19.70748 22.87544 21.35853 18.97514 20.85349 17.98534 21.08760 17.84988
 [9] 21.34702 18.76020
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  17.85   18.81   20.28   20.08   21.28   22.88
```

Konkrétně získáme minimum, první kvartil, medián (druhý kvartil), průměr, třetí kvartil a maximum. K jednotlivým položkám se dostaneme buď takto:

```
> xs<-summary(x)
> xs
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  17.85   18.81   20.28   20.08   21.28   22.88
> xs[1]
 Min.
17.85
> xs[2]
1st Qu.
  18.81
> xs[6]
 Max.
22.88
```

nebo pomocí speciálních funkcí se snadno odhadnutelnými názvy:

```
> min(x)
[1] 17.84988
> max(x)
[1] 22.87544
> median(x)
[1] 20.28048
> mean(x)
[1] 20.08001
```



Obr. 6.1 Příklad grafu boxplot

V minulé kapitole jsme si ukázali bez bližšího výkladu krabicový graf, neboli boxplot, vynalezený americkým statistikem Tukeyem. Modelovy graf si můžeme ukázat na tomto příkladě:

```
> boxplot(rnorm(     5, mean=20, sd=2), rnorm(   10, mean=20, sd=2),
+         rnorm(   100, mean=20, sd=2), rnorm( 1000, mean=20, sd=2),
+         rnorm( 10000, mean=20, sd=2))
```

Každý sloupec v tomto typu grafu představuje jednu sérii dat, v našem případě sérii náhodných čísel s normálním rozdělením s různým počtem hodnot. Tlustá horizontála uvnitř krabice představuje medián. Spodek a vršek krabice představují první a třetí kvartil. Ze spodku nebo vršku krabice vycházejí „vousy“. Jejich délka může dosáhnout maximálně 1,5-násobku výšky krabice. Pokud se všechny body v tomto rozsahu nachází, pak jsou vousy vedeny pouze k minimální respektive maximální hodnotě. Pokud vzdálenost nějakých dat přesahuje 1,5-násobek výšky krabice, pak jsou vousy vedeny k minimální respektive maximální hodnotě, která se ještě v tomto rozsahu nachází, zatímco body, které se v rozsahu nenachází, jsou zobrazeny jako kolečka. Boxplot tedy umožňuje visuálně posoudit střední hodnotu, odchylky, symetrii rozdělení a přítomnost odlehlých bodů.

Kapitola 7

Základní statistiky souboru

V této kapitole si ukážeme jak v programu R vypočítat základní statistiky souboru, jimiž jsou odhad střední hodnotu a směrodatné odchylky a střední chyba průměru. Odhad střední hodnoty náhodného výběru můžeme vypočítat jako průměr hodnot, buď „ručně“ jako podíl součtu (`sum`) a počtu (`length`) prvků, nebo pomocí funkce `mean`:

```
> x<-rnorm(10, mean=20, sd=2)
> x
 [1] 21.39152 20.65200 20.86989 20.89594 20.06385 19.21771 18.18409 18.42394
 [9] 22.41639 19.77035
> sum(x)/length(x)
[1] 20.18857
> mean(x)
[1] 20.18857
```

Odhad směrodatné odchylky (standard deviation) můžeme získat opět ručně nebo pomocí funkce `sd`:

```
> sqrt(sum((x-mean(x))^2)/(length(x)-1))
[1] 1.327257
> sd(x)
[1] 1.327257
```

Odhad rozptylu, neboli druhou mocninu odhadu směrodatné odchylky, získáme:

```
> sum((x-mean(x))^2)/(length(x)-1)
[1] 1.761612
> var(x)
[1] 1.761612
```

Pro střední chybu průměru (standard error of the mean), alespoň pokud je mi známo,

Tabulka 7.1 Základní statistické veličiny

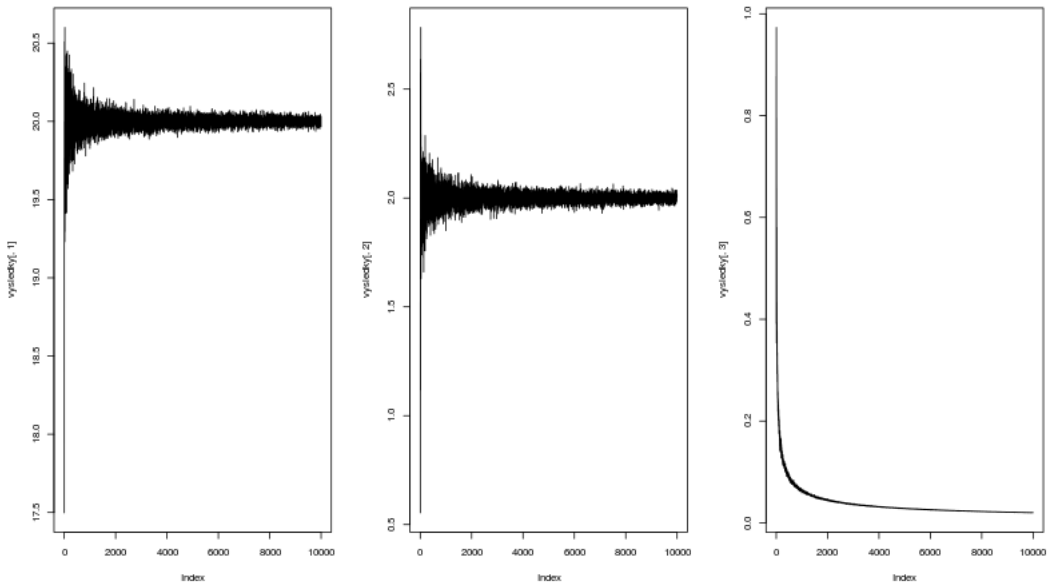
česky	anglicky	R	vzoreček
odhad střední hodnoty	mean	`mean()`	$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$
odhad směrodatné odchylky	standard deviation	`sd()`	$s = \sqrt{\frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N-1}}$
rozptyl	variance	`var()`	$s^2 = \frac{\sum_{i=1}^{N}(x_i-\mu)^2}{N-1}$
střední chyba průměru	standard error of the mean	–	$SEM = \frac{s}{\sqrt{N}}$

není v R žádná speciální funkce. Získáme jí jako podíl odhadu směrodatné odchylky a odmocniny z počtu hodnot:

```
> sd(x)/sqrt(length(x))
[1] 0.4197157
```

V případě opakovaného měření nějaké hodnoty vyjadřuje odhad směrodatné odchylky přesnost každého jednotlivého měření. Naproti tomu, střední chyba průměru vyjadřuje přesnosti celé série měření jako celku. Pokud budeme přidávat další a další měření, pak se hodnota směrodatné odchylky bude přibližovat skutečné směrodatné odchylce, která je například dána přesností měřícího přístroje. Pro nekonečně mnoho měření bychom měli získat přesnou hodnotu směrodatné odchylky. Naproti tomu, střední chyba průměru má tendenci s počtem měření klesat. S nekonečným počtem měřením se dostaneme na přesnou hodnotu průměru a střední chyba průměru bude nulová. Ukázat si to můžeme na jednoduchém prográmku:

```
> mojestatistika<-function(n) {
+   x<-rnorm(n, mean=20, sd=2)
+   xmean <- mean(x)
+   xsd <- sd(x)
+   xsem <- sd(x)/sqrt(length(x))
+   return(c(xmean, xsd, xsem))
+ }
> mojestatistika(1)
[1] 18.08028       NA       NA
> mojestatistika(2)
[1] 17.937201  2.639565  1.866454
> vysledky<-c()
> for(i in 2:10000) {
+   vysledky<-rbind(vysledky, mojestatistika(i))
```



Obr. 7.1 Odhad střední hodnoty, odhad směrodatné odchylky a střední chyba průměru pro různě velké výběry

```
+ }
> plot(vysledky[,1], type="l")
> plot(vysledky[,2], type="l")
> plot(vysledky[,3], type="l")
```

Grafy zobrazují závislost odhadu střední hodnoty, odhadu směrodatné odchylky a střední chyby průměru na velikosti souboru. Zatímco odhad střední hodnoty se blíží skutečné střední hodnotě (20) a odhad směrodatné odchylky se blíží skutečné směrodatné odchylce (2), střední chyba průměru se s rostoucí velikostí souboru blíží nule.

Kapitola 8

Interval spolehlivosti

Pokud provedeme sérii měření nějaké veličiny, pak na základě nich můžeme odhadnout interval spolehlivosti. Vypočteme jej jako střední chybu průměru vynásobenou koeficientem Studentova t-rozdělení. Pro tento účel má program R k dispozici funkci `qt`. Pro naše data získáme interval spolehlivosti na hladině pravděpodobnosti 95 % takto:

```
> x<-rnorm(10, mean=20, sd=2)
> x
 [1] 20.19800 20.86360 21.90173 21.50015 21.13737 21.15444 19.42366 21.63679
 [9] 19.60339 16.91308
> sem<-sd(x)/sqrt(length(x))
> mean(x)+sem*c(qt(p=0.025, df=(length(x)-1)),qt(p=0.975, df=(length(x)-1)))
[1] 19.36419 21.50225
```

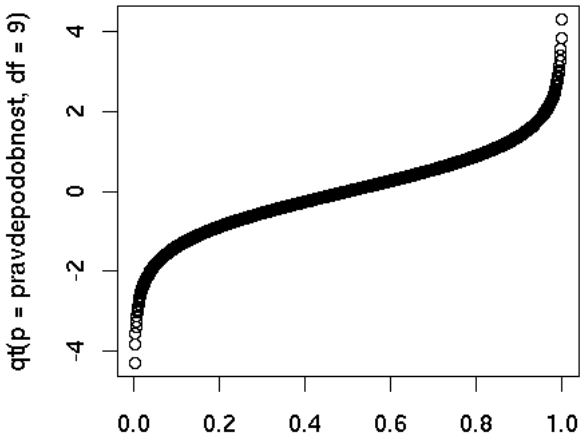
Tedy že střední hodnota (která je 20, což bychom ale v případě reálného měření nevěděli) leží s 95% pravděpodobností v intervalu od 19,36419 do 21,50225. A ono tomu tak ve skutečnosti je. Podívejme se na funkci `qt`, která poskytuje kvantil Studentova t-rozdělení. Můžeme si nakreslit graf:

```
> pravdepodobnost <- 1:999/1000
> plot(pravdepodobnost, qt(p=pravdepodobnost, df=9))
```

Argument `df` určuje počet stupňů volnosti, který je rovný počtu měření mínus jedna. Argument `p` určuje hladinu pravděpodobnosti. Hodnota `qt(p=0, df=9)` má hodnotu mínus nekonečno; hodnota `qt(p=1, df=9)` plus nekonečno:

```
> qt(p=0, df=9)
[1] -Inf
> qt(p=1, df=9)
[1] Inf
```

To znamená, že abychom získali interval pro stoprocentní spolehlivost, pak bychom museli střední chybu průměru násobit mínus a plus nekonečnem, tedy že střední hodnota



Obr. 8.1 Kvantil Studentova t-rozdělení pro různé hladiny pravděpodobnosti

s jistotou leží v intervalu mínus nekonečno – plus nekonečno. Hodnotu `qt` pro pravděpodobnost 0,025 (tedy 2,5 %) získáme:

```
> qt(p=0.025, df=9)
[1] -2.262157
```

To znamená, že se střední hodnota s 2,5% pravděpodobností nachází v intervalu od mínus nekonečna (průměr + střední chyba průměru násobená mínus nekonečnem) do hodnoty průměr + střední chyba průměru násobená hodnotou -2,262157. S 97,5% pravděpodobností se pak nachází v intervalu od hodnoty průměr + střední chyba průměru násobená hodnotou -2,262157 do plus nekonečna. Pro pravděpodobnost 0,975 je hodnota stejná `qt`, akorát s opačným znaménkem:

```
> qt(p=0.975, df=9)
[1] 2.262157
```

Tedy s 97,5% pravděpodobností se střední hodnota nachází v intervalu od mínus nekonečna do hodnoty průměr + střední chyba průměru násobená hodnotou +2,262157. Když dáme tyto informace dáme dohromady, pak nám vyjde, že s 95% pravděpodobností se střední hodnota nachází v intervalu od hodnoty průměr + střední chyba průměru násobená hodnotou -2,262157 do průměr + střední chyba průměru násobená hodnotou +2,262157.

Kapitola 9

p-Hodnota

Řekněme že v rámci svého výzkumného projektu studujeme efekt sloučeniny na růst buněčné kultury. Provedeme čtyři pokusy, ve kterých měříme růst buněk s přídavkem sloučeniny, a čtyři pokusy bez přídavku. Poté použijeme t-test, abychom statisticky otestovali vliv sloučeniny. Klasický „tabulkový“ postup při testování statistické hypotesy je následující. Nulovou hypotesou je, že střední hodnota pro neošetřené a ošetřené buňky je stejná. Alternativní hypotesou je, že se střední hodnoty liší. Nejprve vezmeme naměřené hodnoty a podle určitého postupu, který je daný typem testu, vypočteme určité kritérium. V statistických tabulkách si poté na zvolené hladině spolehlivosti nalezneme hodnotu koeficientu daného rozdělení, v našem případě Studentova t-rozdělení. Nakonec srovnáváme hodnotu kritéria a hodnotu koeficientu a podle toho, která z nich je nižší, buď zamítáme nebo nezamítáme nulovou hypotesu.

Hypoteticky bychom mohli, pokud bychom měli dostatečně velké statistické tabulky a dost času, hledat na různých hladinách pravděpodobnosti tak dlouho, až by se koeficient daného rozdělení přesně rovnal kritériu. Tuto hladinu pravděpodobnosti bychom mohli označit jako p-hodnotu (p-value). Počítač, tedy alespoň program R, udělá tuto práci za nás. p-Hodnoty nalezneme nejen v R a v klasické statistice, ale taktéž ve výsledcích různých bioinformatických nástrojů, například při prohledávání sekvenčních databasí nebo při identifikaci mikroorganismu podle hmotnostních spekter.

Co tedy p-hodnota znamená a co neznamená? Přesná definice je, že se jedná o pravděpodobnost výsledku statistického testu, který by byl tak extrémní jako výsledek, který nám vyšel, za předpokladu, že je nulová hypotesa pravdivá. Jak této definici rozumět?

Představte si, že nám při zpracování výsledků testu působení sloučeniny na buněčnou kulturu vyšly relativně velké rozdíly mezi mezi ošetřenými a neošetřenými buňkami s p-hodnotou rovnou 0,0002959. Pokud bychom si vybrali hladinu pravděpodobnosti 10 %, 5 % nebo 1 %, pak bychom ve všech příkladech mohli zamítnout nulovou hypotesu.

Nyní vezmeme generátor náhodných čísel a vygenerujeme stejný počet hodnot měření tak, aby platily následující podmínky: čísla mají normální rozdělení, mají stejné směrodatné odchylky jako naše data z reálného měření a platí nulová hypotesa, tedy že jsou jejich střední hodnoty stejné. Vzhledem k poslední podmínce, tedy stejným středním hodnotám, je velmi pravděpodobné, že se hodnoty nebudou příliš lišit. Naopak pravděpodobnost, že bychom dostali tak velké rozdíly mezi ošetřenými a neošetřenými buňkami jako v případě reálného měření, je velmi nízká. Touto hodnotou pravděpodobnosti je právě 0,0002959.

Podobná situace je i mimo klasickou statistiku. V bioinformatice se velmi často používá program BLAST pro prohledávání sekvenčních databasí. Do tohoto programu je možné zadat sekvenci proteinu a nechat program aby prohledal database a našel podobné proteiny. U každého proteinu je možné nalézt p-hodnotu. Význam p-hodnoty je analogický t-testu. Jedná se o pravděpodobnost, že bychom našli stejně podobný protein ve stejně veliké databasi náhodných sekvencí. Podobně při identifikaci bakterie pomocí hmotnostních spekter se jedná o pravděpodobnost, že bychom nalezli stejně podobná spektra v databasy náhodných spekter.

Co p-hodnota není? NEJEDNÁ se o pravděpodobnost nulové hypotesy. Celá koncepce statistického testování je založená na předpokladu, že pozorovaný výsledek je dílem náhody. Testujeme tedy, že tento předpoklad je špatný, nikoliv že opačná hypotesa je pravdivá. Rovněž p-hodnota NENÍ pravděpodobnost, že falešně zamítneme nulovou hypotesu. Zároveň p-hodnota NENÍ ani pravděpodobnost, že další série pokusů povede k jiným závěrům.

Kapitola 10

t-Test

Ekvivalentem intervalů spolehlivosti je jednovýběrový t-test, který si ukážeme na vygenerovaných datech. Nulovou hypotesou bude, že je střední chyba průměru rovná hodnotě 20. Alternativní hypotesou je, že střední hodnota není rovna 20. Ručně tento test můžeme provést takto:

```
> x <- rnorm(10, mean=20)
 [1] 20.19800 20.86360 21.90173 21.50015 21.13737 21.15444 19.42366 21.63679
 [9] 19.60339 16.91308
> mean(x)
[1] 20.43322
> sem<-sd(x)/sqrt(length(x))
> R<-(mean(x)-20.0)*sqrt(length(x))/sd(x)
> R
[1] 0.9167299
> qt(p=0.975, df=(length(x)-1))
[1] 2.262157
```

Nejdříve vypočteme střední hodnotu a odhad směrodatné odchylky. Pak vypočteme kritérium *R*. Jeho hodnotu srovnáme s hodnotou koeficientu Studentova t-rozdělení na hladině pravděpodobnosti 0.95 (95 %). Důvod proč uvádíme `p=0.975` a nikoliv `p=0.95` byl vysvětlen v minulé kapitole. Vzhledem k tomu, že absolutní hodnota kritéria *R* (0,9167) je menší než koeficient Studentova t-rozdělení (2,2622), nezamítáme nulovou hypotesu. Pokud by byla situace opačná, pak bychom mohli nulovou hypotesu zamítnout. Nulovou hypotesu nepřijímáme, pouze ji můžeme zamítnout.

Představte si, že máme například místnost, jejíž délka má být 20 m, a my chceme tento předpoklad ověřit. Můžeme pomocí vhodného měřidla čtyřikrát změřili její délku

a pomocí t-testu otestovat nulovou hypotesu, že její délka je skutečně 20 m. Pokud nám vyjde, že nemáme zamítnou nulovou hypotesu, pak můžeme předpokládat, že délka je opravdu 20 m. Pokud nám vyjde, že můžeme zamítnout nulovou hypotesu, pak s rizikem odpovídajícím dané hladině pravděpodobnosti můžeme předpokládat, že místnost 20 m mená. Nulovou hypotesu ale nepřijímáme. To by znamenalo, že předpokládáme, že místnost má 20,000 m s nekonečnem nul, což zcela jistě nemá. Nepřijímáme ani alternativní hypotesu, potože by to znamenalo, že tvrdíme že místnost nemá 20,000 m s nekonečnem nul, což je zcela jistě pravda.

V programu R můžeme t-test provést nejen ručně, ale také pomocí speciální funkce `t.test`:

```
> t.test(x, mu=20, conf.level=0.95)

        One Sample t-test

data:  x
t = 0.9167, df = 9, p-value = 0.3832
alternative hypothesis: true mean is not equal to 20
95 percent confidence interval:
 19.36419 21.50225
sample estimates:
mean of x
 20.43322
```

Jako hladinu pravděpodobnosti uvádíme `conf.level=0.95`. Tato funkce nám vypočte střední hodnotu a interval spolehlivosti. Důležitá hodnota je `p-value` (0,3832). Hodnota p-value, tedy pravděpodobnost, že za podmínek platnosti nulové hypotesy získáme stejný rozdíl mezi průměrem náhodně generovaných dat a hodnotou 20, je 38,32 %, tedy více než 5 %. Proto nezamítáme nulovou hypotesu.

Pokud vám ještě uniká půvab t-testu, možná vás přesvědčí následující cvičení. Vytvoříme si funkci `jedentest`. Tato funkce bude mít parametry `xn`, `xmean`, `xsd` a `xprob`. Funkce si vytvoří vektor náhodných čísel na základě těchto hodnot. Pak na hladině pravděpodobnosti `xprob` otestuje, jestli se průměr těchto hodnot rovná nastavené hodnotě `xmean`. K tomu využijeme interval spolehlivosti `ttest$conf.int[1]` a `ttest$conf.int[2]`. Pokud odhad střední hodnoty leží v intervalu spolehlivosti, pak funkce vrátí hodnotu jedna, v opačném případě vrátí nulu. Když tuto funkci použijeme řekněme 10 000x a hladinu pravděpodobnosti dáme rovnou 0,5. Pokud posčítáme nuly a jedničky, pak bychom měli dostat hodnotu přibližně odpovídající násobku počtu pokusů (10 000) a hladiny pravděpodobnosti (0,5), tedy přibližně 5 000. Račte si to zkusit

s různými hodnotami `xn`, `xmean`, `xsd` a `xprob`. Upozorňujeme, že výpočet bude chvíli trvat:

```
> jedentest<-function(xn, xmean, xsd, xprob) {
+   x<-rnorm(xn, mean=xmean, sd=xsd)
+   ttest<-t.test(x, mu=xmean, conf.level=xprob)
+   odpoved <- 0
+   if ((ttest$conf.int[1]<xmean)&(ttest$conf.int[2]>xmean)) odpoved <- 1
+   return(odpoved)
+ }
> result<-0
> for(i in 1:10000) {
+   result<-result+jedentest(xn=10, xmean=0, xsd=5, xprob=0.5)
+ }
> result
[1] 4938
```

Naším výsledkem je 4 938, tedy přibližně 5 000. S hodnotou `xprob=0.95` bychom měli dostat přibližně 9 500, dostali jsme 9 448.

Kromě oboustranného t-testu je možné v programu provést i jednostranný t-test. Nulová hypotesa v následující ukázce je, že střední hodnota je vyšší nebo rovná 20. Alternativní hypotesa je, že je střední hodnota nižší než 20. Pro jednostranný t-test použijeme argument `alternative`:

```
> t.test(x, mu=20, alternative="less")

        One Sample t-test

data:  x
t = 0.9167, df = 9, p-value = 0.8084
alternative hypothesis: true mean is less than 20
95 percent confidence interval:
    -Inf 21.2995
sample estimates:
mean of x
 20.43322
```

Výsledkem je, že s 95 % pravděpodobností leží střední hodnota v intervalu od mínus nekonečna do 21,2995. Na základě hodnoty p-value nezamítáme nulovou hypotesu.

V biologických vědách nejčastěji použijeme dvouvýběrový t-test. Místo porovnávání jedné nepřesné veličiny s jednou přesnou porovnáváme dvě nepřesné veličiny. Pokud

chceme zjistit, jestli má nějaká sloučenina vliv na růst rostliny, pak můžeme provést porovnání výšky rostlin ošetřených a neošetřených sloučeninou. Počet opakování bude roven řekněme deseti. Před tím, než začneme s t-testem, bychom měli správně otestovat, jestli jsou směrodatné odchylky pro ošetřené a neošetřené rostliny různé a podle toho použít tu správnou variantu testu. Pro jednoduchost budeme uvažovat stejné směrodatné odchylky. Zde je ukázkový t-test pro data vygenerovaná funkcí `rnorm`. Nulovou hypotesou je, že jsou střední hodnoty obou výběrů stejné. Alternativní hypotesou je, že se liší. V našem případě vychází:

```
> neosetrene<-rnorm(10, mean=12.3, sd=3.3)
> osetrene<-rnorm(10, mean=8.5, sd=3.3)
> neosetrene
 [1] 10.038366  9.094181 11.289843 15.878454 15.250237  8.415832  6.604380
 [8] 11.411414 11.793384 14.677340
> osetrene
 [1] 14.138496  8.304396  6.384113 17.792928 10.135895  8.015353 12.868893
 [8] 10.341616  7.910172  9.081289
> t.test(neosetrene, osetrene)

        Welch Two Sample t-test

data:  neosetrene and osetrene
t = 0.6462, df = 17.728, p-value = 0.5264
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.137451  4.033507
sample estimates:
mean of x mean of y
 11.44534  10.49732
```

Hodnota `p-value` je 0,5264, tedy nezamítáme nulovou hypotesu, jinými slovy nemůžeme na dané hladině pravděpodobnosti prokázat, že má sloučenina vliv na výšku rostliny.

Jak bylo řečeno, dříve než začneme s t-testem, bychom měli nejprve otestovat, jestli mají porovnávané skupiny stejné rozptyly, a podle toho použít odpovídající variantu t-testu. V předchozím příkladě byl použit t-test zrealizovaný pomocí funkce `t.test` bez dalších parametrů. Tato funkce má argument `var.equal`, která má defaultní hodnotu `FALSE`. Pokud předpokládáme, že oba výběry mají stejné rozptyly, pak musíme použít t-test s volbou `var.equal=TRUE`. Výsledek je velmi podobný:

```
> t.test(x,y,var.equal=TRUE)

        Two Sample t-test
```

```
data:  x and y
t = 0.6462, df = 18, p-value = 0.5263
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.134054  4.030110
sample estimates:
mean of x mean of y
 11.44534  10.49732
```

Výsledná p-hodnota se liší až na čtvrtém desetinném místě, to znamená, že náš prohřešek spočívající v neotestování shod rozptylů neměl fatální důsledky. Pokud chceme otestovat, jestli jsou rozptyly stejné nebo různé, můžeme použít `var.test`.

```
> var.test(x,y)

        F test to compare two variances

data:  x and y
F = 0.7794, num df = 9, denom df = 9, p-value = 0.7165
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.1936038 3.1380526
sample estimates:
ratio of variances
         0.7794479
```

Nulovou hypotesou je, že poměr rozptylů je rovný nule, tedy že oba rozptyly jsou stejné. Na základě p-hodnoty 0,7165 nezamítáme nulovou hypotesu, tedy použijeme t-test s nastavením `var.equal=TRUE`.

Poslední variantou t-testu, kterou si představíme, je párový t-test. Představte si, že chceme statisticky otestovat hypotesu, že teplota v Praze je jiná než v Peci pod Sněžkou. K dispozici máme záznam teplot během roku 2009, konkrétně průměrnou teplotu v lednu, únoru a tak dále, vždy v Praze a v Peci. Každý duševně zdravý člověk vám řekne, že v Peci bude větší zima. Pokud ale použijete běžný t-test, je možné, že výsledek bude nejednoznačný.

```
> pec  <-c(-6,-3, 1, 7, 9,12,14,14,12, 3, 3,-3)
> praha<-c(-3, 0, 4,13,14,15,18,19,16, 8, 7,-1)
> t.test(pec,praha)
```

```
        Welch Two Sample t-test

data:  pec and praha
t = -1.2915, df = 21.823, p-value = 0.2100
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -10.208747   2.375414
sample estimates:
mean of x mean of y
  5.250000  9.166667
```

Důvodem nejednoznačnosti je fakt, že se teploty v Praze budou pohybovat mezi -3 a $+19°$C a na Sněžce bude -6 až $+14°$C, takže rozdíl mezi průměry je malý a směrodatné odchylky velké. Nápad zprůměrovat teploty a ty potom porovnávat je nesprávný a daleko lepší je porovnávat rozdíly teplot v lednu, únoru a tak dále. K tomu slouží párový t-test, který použijeme pokud zvolíme argument `paired=TRUE`:

```
> t.test(pec,praha, paired=TRUE)

        Paired t-test

data:  pec and praha
t = -11.6511, df = 11, p-value = 1.574e-07
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.656555 -3.176779
sample estimates:
mean of the differences
              -3.916667
```

Nejistota je rázem ta tam a zdravý rozum zvítězil nad nesprávným použitím statistické metody.

Poslední záležitost, kterou si probereme v souvislosti s t-testem, je jeho použití na objekt typu `data.frame`. Dosud jsme t-test používali pouze pro modelová data uložená ve dvou proměnných (např. *a* a *b*) se zápisem `t.test(a,b)`. Místo toho si vytvoříme objekt `data.frame`, například si ho nahrát ze souboru, a pak na něj použít jiný zápis funkce `t.test`. Pro nás bude tento zápis šikovnější pro další použití společně s analysou rozptylu a dalšími metodami. Pokud si nahrajeme `data.frame` obsahující výšky ošetřených a neošetřených rostlin v tomto tvaru:

```
> df
  f       val
1 o  9.790633
2 o 11.643531
3 o  8.297789
4 o 11.880794
5 c  8.411770
6 c 10.736672
7 c  8.489036
8 c  6.450199
```

kde *f* je faktor, který má hodnotu „o“ pro ošetřené a „c“ pro kontrolní rostliny, a *val* je výška rostliny, pak můžeme provést t-test jako:

```
> t.test(val~f, data=df)

    Welch Two Sample t-test

data:  val by f
t = -1.5473, df = 5.991, p-value = 0.1728
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.857456  1.094920
sample estimates:
mean in group c mean in group o
       8.521919       10.403187
```

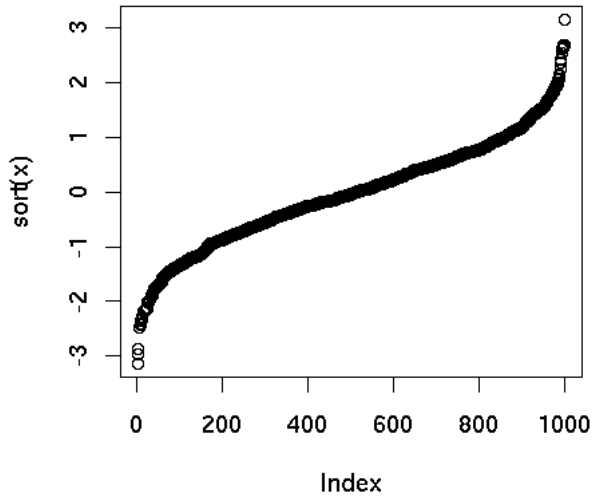
Podobný zápis můžeme použít i pro funkci `plot`, tedy `plot(val f, data=df)`. Račte vyzkoušet sami.

Kapitola 11

Neparametrické testy

Dosud jsme předpokládali, že naše data mají normální rozdělení. To ale nemusí platit. Pokud nemají data normální rozdělení, pak je nutné použít jiné metody než t-test, takzvané neparametrické testy. První co bychom tedy měli otestovat je zdali ozdělení je normální. V programu R je k dispozici zajímavý test pro grafické ověření normálního rozdělení. Pro toto rozdělení je typická velmi nízká hustota bodů daleko od střední hodnoty a vysoká v její blízkosti. Můžeme vzít hodnoty náhodného výběru, seřadit je od nejmenšího po největší (funkcí `sort`) a takto je zobrazit:

```
> x<-rnorm(1000)
> plot(sort(x))
```

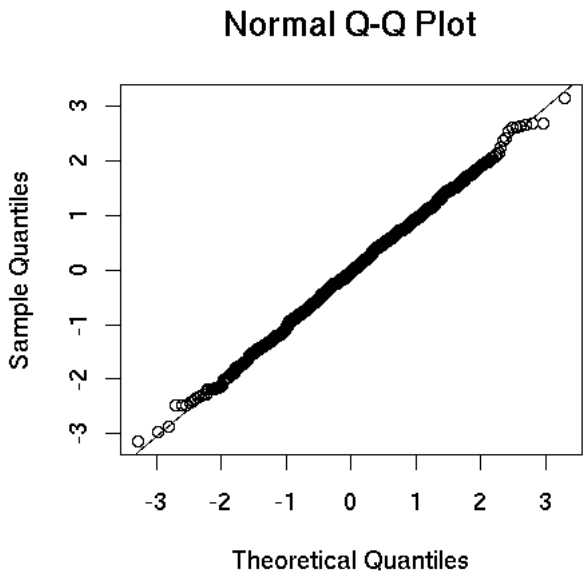


Obr. 11.1 Náhodný výběr s normálním rozdělením seřazený podle hodnot

Když víme, jak má tento profil pro daný průměr a směrodatnou odchylku teoreticky vypadat, pak můžeme funkci „narovnat“ a porovnat teoretická a skutečný profil. K tomu má program funkce `qqnorm` a `qqline`:

```
> qqnorm(x)
```

```
> qqline(x)
```



Obr. 11.2 QQ-výnos pro stejná data

které zobrazí takzvaný QQ-výnos (kvantil-kvantil). Odchylky od normálního rozdělení se projeví jako odchylka od lineárního průběhu. Zatímco odchylky uprostřed grafu (kolem střední hodnoty) značí odchylky od normálního rozdělení, odchylky na okrajích naznačují odlehlé hodnoty.

QQ-výnos představuje vizuální nástroj jak posoudit, zdali analysovaná data mají normální rozdělení. Kvantitativně je možné toto testovat pomocí testu podle Shapira a Wilka. Tento test je možné v R provést funkcí `shapiro.test`. Ten si můžeme ukázat nejprve na datech s normálním rozdělením a poté na datech, která normální rozdělení nemají (jedná se o normální rozdělení se dvěma středy):

```
> x<-rnorm(20)
> shapiro.test(x)

        Shapiro-Wilk normality test

data:  x
W = 0.96, p-value = 0.5429

> x<-c(rnorm(10), rnorm(10, mean=4))
> shapiro.test(x)

        Shapiro-Wilk normality test
```

```
data:  x
W = 0.8849, p-value = 0.02168
```

Přípomínám, že nulovou hypotesou je, že data mají normální rozdělení.

Co ale s daty, která nemají normální rozdělení a tedy nemůžeme použít t-test? Alternativou t-test, za předpokladu, že nemůžeme předpokládat normální rozdělení, je Wilcoxonův dvouvýběrový test (rovněž Mannův-Whitneyův test). V R tento test realizujeme funkcí `wilcox.test`. Jeho použití (a v případě normálního rozdělení i výsledky) jsou podobné, jako v případě t-testu:

```
> x<-rnorm(10)
> y<-rnorm(10, mean=2)
> wilcox.test(x,y)

        Wilcoxon rank sum test

data:  x and y
W = 12, p-value = 0.002879
alternative hypothesis: true location shift is not equal to 0

> t.test(x,y)

        Welch Two Sample t-test

data:  x and y
t = -3.4554, df = 17.593, p-value = 0.002900
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.8695355 -0.6972672
sample estimates:
mean of x mean of y
0.1979817 1.9813831
```

Kapitola 12

Mnohonásobné porovnání

Představte si, že chcete zjistit, jestli má hraní určité hudby vliv na růst rostlin. Můžete provést pokus, kdy budete pěstovat například pět rostlin v tichu a dalším pěti budete z reproduktorů přehrávat určitou hudbu. Po zvolené době změříte výšku všech rostlin a výsledky vyhodnotíte pomocí dvou-výběrového t-testu. Nulovou hypotesou je, že hudba nemá vliv na výšku rostlin. Pokud zvolíte hladinu pravděpodobnosti 95 % (P = 0,05), pak máte 5% pravděpodobnost, že nám vyjde, že hudba má vliv, i když vliv nemá.

Řekněme, že nechcete testovat vliv pouze jednoho druhu hudby, ale co nejvíce žánrů. Pak je možné pět rostlin pěstovat v tichu (kontrolní skupina) a dalších 500 rostlin rozdělit do skupin po pěti a každé skupině pustit jiný hudební žánr. Nejprve si ukážeme nesprávné zpracování a pak si vysvětlíme proč není správné. Jako nejjednoduší možné zpracování výsledků nás napadne provést sto t-testů a v každém porovnat kontrolní skupinu s každým jednotlivým žánrem. Pokud by nám vyšlo $P < 0{,}05$, interpretovali bychom to tak, že daný žánr ovlivňuje růst rostliny. Pak bychom mohli vydat tiskovou zprávu, že „... vědci z Dejvic zjistili, že normalizační pop, thrash metal, balkánská dechovka, středověký kancionál a švýcarský folklór ovlivňují růst rostlin“. Důvod proč toto zpracování a interpretace nejsou správné je následující: Předpokládejme nejprve platnost nulové hypotesy, tedy že hudba, bez ohledu na žánr, nemá vliv na růst rostlin. Když jsme testovali vliv jednoho žánru na růst rostliny, tak jsme měli 5% pravděpodobnost, že nesprávně zamítneme nulovou hypotesu, a tedy 95% pravděpodobnost, že ji správně nezamítneme. Pokud bychom měli 100 různých žánrů, pak pravděpodobnost, že vždy správně nezamítneme nulovou hypotesu, je $0{,}95^{100}$, tedy 0,0059. Pravděpodobnost, že alespoň jednou nesprávně zamítneme

nulovou hypotesu, je 1 - 0,0059 = 0,9941, tedy 99,4%. Máme tedy 99,4% pravděpodobnost, že najdeme alespoň jeden žánr, který ovlivňuje růst rostliny, i když žádná hudba růst rostliny neovlivňuje.

Na stejný problém, tedy problém vícečetného porovnávání, narážíme při zpracovávání experimentů v biochemii a molekulární biologii velmi často. Například při hledání nějakého nového léčiva je potřeba otestovat velké série různých sloučenin. Jen nepatrný zlomek z nich je skutečně aktivní. Pokud bychom tento problém ignorovali, vyšlo by nám, že každá dvacátá sloučenina (pro P=0,05) je biologicky aktivní, i když ve skutečnosti je jich aktivních podstatně méně. Podobně, když bychom pomocí DNA čipů porovnávali koncentrace mRNA v buňkách ovlivněných a neovlivněných nějakou sloučeninou, pak by nám vyšlo, že každý dvacátý, tedy u člověka 30000x0,05=1500 genů, i když by sloučenina ovlivnila expresi jen několika desítek genů. Dalším příkladem, kdy ignorování problému mnohonásobného porovnávání může způsobit škody, jsou studie, kdy jsou testovány různé vlivy (faktory). Například když někdo měří krevní tlak velké skupině pacientů a ví o nich, zda jsou muži/ženy, mladí/staří, kuřáci/nekuřáci, svobodní/sezdaní atd. Opět zde, pokud by byl ignorován problém vícečetného porovnání, by s rostoucím počtem faktorů rostla pravděpodobnost, že najdete faktor, který má vliv na krevní tlak, i kdyby žádný faktor vliv neměl.

Představme si ale následující situaci: Jako medicinální chemik připravíte pět různých sloučenin s možnou protinádorovou aktivitou. U těchto sloučenin změříte vliv na růst nádorových buněk. U sloučeniny 1, 2, 4 a 5 nezjistíte pomocí t-testu řádnou signifikantní změnu proti kontrole. U sloučeniny 3 zjistíte, že je změna signifikantní. Když ale pomocí některá výše uvedené metody provedete korekci problému mnohonásobného porovnávání, vyjde vám, že ani ta sloučenina číslo 3 není signifikantně aktivní. Co s tím? Je nutné kvůli problému mnohonásobného porovnávání zahodit roční práci, i když t-test ukázal signifikantní aktivitu? Jedna možnost je presentovat výsledky testů v publikaci, absolventské práci a podobně a čestně přiznat, že t-test ukázal signifikantní aktivitu, ale korekce na vícečetné porovnání tuto aktivitu zpochybnila. Je to daleko lepší řešení, než vyhodit celoroční práci. Ještě lepší, pokud ta možnost existuje, je zapomenout na předchozí výsledky a provést nové kultivace kontrolních buněk a buněk ovlivněných sloučeninou 3 a výsledky porovnat t-testem.

Kapitola 13

Analysa rozptylu

Jedna z možností jak vyzrát na problém mnohonásobného porovnávání je provést test, jehož nulovou hypotesou je, že průměr všech souborů jsou stejné, tedy například že hudba, bez ohledu na žánr, neovlivňuje růst rostlin. Alternativní hypotesou je, že se průměry liší, tedy že hudba obecně nebo nějaký žánr růst ovlivňuje. Neprovádíme tedy sérii testů jednotlivých hudebních žánrů, ale vliv hudby jako takový. Přesně to dělá analýza rozptylu, neboli ANOVA (Analysis of variance).

Analysu rozptylu si předvedeme na statistickém hodnocení vlivu nějakého potenciálního léku na lidský organismus v klinickém testu. První co člověka napadne je rozdělit skupiny dobrovolníků na dvě poloviny, jedné podávat lék, druhou použít jako kontrolní a po vybrané době porovnat biologickou aktivitu, například t-testem. Tento postup ale není správný. Důvodem je placebo efekt. Pro opravdu kvalifikovanou analysu bychom měli porovnat kontrolní skupinu dobrovolníků, skupinu, které byla podávána testovaná látka a skupinu, které bylo podáváno placebo. V principu je možné provést trojici t-testů, kontrola-placebo, kontrola-testovaná látka a placebo-testovaná látka. Tento postup je ale z důvodu mnohonásobného porovnávání nesprávný. Naopak, správným postupem je provést analysu rozptylu. Nejprve si ukážeme základní verzi této metody „ručně“. Vytvoříme si tři série vzorků, jeden pro kontrolu, jeden pro testovanou sloučeninu a jeden pro placebo. Nulová hypotesa je, že střední hodnoty všech tří kategorií jsou stejné. Alternativní hypotesou je, že alespoň jedna střední hodnota je odlišná. Začneme vytvořením dat:

```
> kontrola<-rnorm(10, mean=100, sd=25)
> sloucenina<-rnorm(10, mean=70, sd=30)
> placebo<-rnorm(10, mean=90, sd=25)
> kontrola
 [1] 151.01585 107.57115 130.19239  65.95538 143.52040  86.14916  93.46906
 [8]  83.37128  68.60852  82.36360
```

```
> sloucenina
 [1]  80.52774  74.89851  82.40174  23.49004  46.68248  41.89712 107.00530
 [8]  81.99111  63.29744  98.52454
> placebo
 [1]  38.66621 104.48646 129.65401 121.42684  87.66300 105.00737 111.59478
 [8]  89.36779 121.69991  85.42165
```

Nyní vypočteme součet čtverců odchylek od průměru v každé skupině:

```
> skontrola<-sum((kontrola-mean(kontrola))^2)
> ssloucenina<-sum((sloucenina-mean(sloucenina))^2)
> splacebo<-sum((placebo-mean(placebo))^2)
```

Tyto hodnoty sečteme a součet si označíme `SSW`, jako *sum of squares within groups*:

```
> SSW<-skontrola+ssloucenina+splacebo
> SSW
[1] 20800.22
```

Nyní si pospojujeme všechny skupiny do jedné:

```
> vsechno<-c(kontrola, sloucenina, placebo)
> vsechno
 [1] 151.01585 107.57115 130.19239  65.95538 143.52040  86.14916  93.46906
 [8]  83.37128  68.60852  82.36360  80.52774  74.89851  82.40174  23.49004
[15]  46.68248  41.89712 107.00530  81.99111  63.29744  98.52454  38.66621
[22] 104.48646 129.65401 121.42684  87.66300 105.00737 111.59478  89.36779
[29] 121.69991  85.42165
```

Pro tuto veleskupinu spočítáme součet čtverců odchylek od jejího průměru, který označíme `SST` (*sum of squares total*):

```
> SST<-sum((vsechno-mean(vsechno))^2)
> SST
[1] 26931.07
```

Tato hodnota je větší nebo rovna `SSW`. V případě, že si jsou `SSW` a `SST` téměř rovné, pak platí buď to, že jsou si jejich průměry blízké, nebo že rozptyly jsou vysoké ve srovnání s rozdíly průměrů. Nyní vypočteme rozdíl veličin a označíme si jej `SSB` (*sum of squares between groups*):

```
> SSB<-SST-SSW
> SSB
[1] 6130.852
```

Pak vypočteme kritérium `FE` které bude mít tvar:

```
> FE<-(SSB*27)/(SSW*2)
> FE
[1] 3.979117
```

Hodnota 27 je první počet stupňů volnosti, vypočtený jako celkový počet vzorků (30) mínus počet kategorií (3 pro kontrolu, sloučeninu a placebo). Hodnota 2 je druhý počet stupňů volnosti, vypočtený jako počet kategorií mínus jedna. Tuto hodnotu porovnáme s kritériem F-rozdělení, které vyžaduje zadání obou stupňů volnosti:

```
> qf(p=0.95, df1=2, df2=27)
[1] 3.354131
```

Hodnota je nižší než kritérium, proto zamítáme nulovou hypotesu. Existuje tedy rozdíl mezi tím, jestli pacient dostává léčivo, placebo nebo nedostává nic.

V programu R můžeme použít funkci `aov`. Nejprve vytvoříme faktory:

```
> labels<-gl(3,10)
> labels
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
Levels: 1 2 3
```

Význam slova faktor zatím ponecháme bez vysvětlování. Nyní použijeme funkci `aov`:

```
> mujmodel<-aov(vsechno~labels)
> mujmodel
Call:
   aov(formula = vsechno ~ labels)

Terms:
                   labels Residuals
Sum of Squares   6130.852 20800.218
Deg. of Freedom         2        27

Residual standard error: 27.75569
Estimated effects may be unbalanced
```

Význam vlnovky $\sim$ si objasníme v kapitole věnované regresi. K výsledkům se dostaneme pomocí funkce `summary`:

```
> summary(mujmodel)
            Df  Sum Sq Mean Sq F value  Pr(>F)
labels       2  6130.9  3065.4  3.9791 0.03058 *
Residuals   27 20800.2   770.4
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Výsledkem je p-hodnota rovná 0,03058, tedy menší než 5 %. Nulovou hypotesu můžeme na hladině pravděpodobnosti 95 % zamítnout. Tato tabulka je takzvaná ANOVA tabulka prvního typu a zobrazuje název komponenty modelu, počet stupňů volnosti (`df` – *degrees of freedom*), součet čtverců odchylek (`Sum Sq` – *sum of squares*), průměrné součty čtverců (`Mean Sq`, podíl předchozích dvou sloupků), hodnotu F-testového kritéria a p-hodnotu. Ekvivalentem kombinace `aov` a `summary` je funkce `anova` spolu s funkcí `lm`, kterou si ukážeme v kapitole věnované regresi:

```
> anova(lm(vsechno~lables))
Analysis of Variance Table

Response: vsechno
          Df  Sum Sq Mean Sq F value  Pr(>F)
lables     2  6130.9  3065.4  3.9791 0.03058 *
Residuals 27 20800.2   770.4
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Vztah mezi regresí a metodou ANOVA si rovněž vysvětlíme.

Ještě je nutné zmínit, že pokud bychom použili metodu ANOVA pouze pro dva výběry, pak je to to samé, jako t-test s volbou stejných rozptylů `t.test(var.equal=TRUE)`. Nechám na čtenářích jestli si to sami vyzkouší.

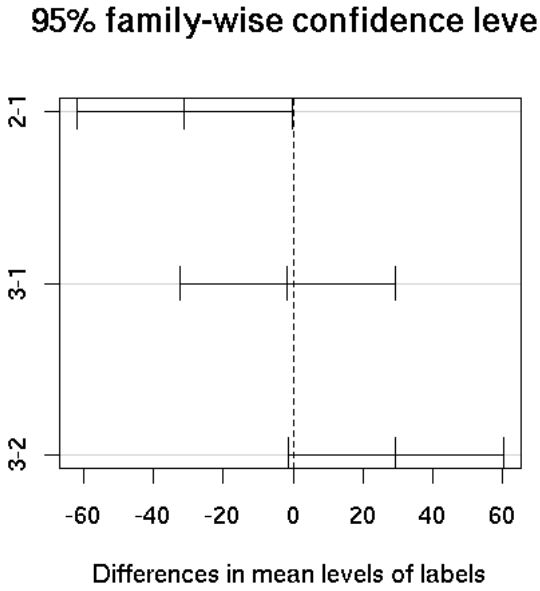
Pokud víme, že střední hodnoty nejsou stejné, jak zjistit jestli jestli léčivo funguje, nebo jestli funguje stejně jako placebo? Pořád musíme mít na paměti fakt, že použití t-testů způsobem každý s každým není správné. Je možné použít například Tukeyův test HSD (*Honest Significant Difference*), který si ukážeme bez bližšího vysvětlování:

```
> TukeyHSD(aov(vsechno~labels))
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = vsechno ~ labels)

$labels
          diff        lwr        upr     p adj
2-1 -31.150077 -61.926401 -0.3737527 0.0468546
3-1  -1.722877 -32.499201 29.0534473 0.9894393
3-2  29.427200  -1.349124 60.2035243 0.0629721
```

Metoda srovná každý výběr s každým. Pokud je `p adj` menší než zvolená pravděpodobnost (pro 95% pravděpodobnost to bude 0,05), pak je možné považovat tyto výběry

za rozdílné. V našem případě můžeme říci, že je rozdílná kontrola vůči sloučenině (2-1). Výsledek si můžeme vykreslit:

```
> plot(TukeyHSD(aov(vsechno~labels)))
```



Obr. 13.1 Výnos Tukeyova testu HSD

Zatím jsme se nezabývali významem faktorů vytvořeným funkcí `gl`. Místo nich je možné se stejným výsledkem použít písmena `a`, `b` a `c`:

```
> jinefaktory<-as.factor(c(rep("a", times=10),
+                          rep("b", times=10),
+                          rep("c", times=10)))
> jinefaktory
 [1] a a a a a a a a a a b b b b b b b b b b c c c c c c c c c c
Levels: a b c
> anova(lm(vsechno~jinefaktory))
Analysis of Variance Table

Response: vsechno
            Df  Sum Sq Mean Sq F value  Pr(>F)
jinefaktory  2  6130.9  3065.4  3.9791 0.03058 *
Residuals   27 20800.2   770.4
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
> TukeyHSD(aov(vsechno~jinefaktory))
  Tukey multiple comparisons of means
    95% family-wise confidence level
```

```
Fit: aov(formula = vsechno ~ jinefaktory)

$jinefaktory
          diff        lwr        upr     p adj
b-a -31.150077 -61.926401 -0.3737527 0.0468546
c-a  -1.722877 -32.499201 29.0534473 0.9894393
c-b  29.427200  -1.349124 60.2035243 0.0629721
```

Analysa rozptylu v podstatě porovnává rozptyl dat za předpokladu jedné hypotezy (například že záleží na tom, jestli pacient dostává lék, placebo nebo nedostává nic) s jinou hypotezou (že na podávání léčiva ani placeba nezáleží). Kromě jediného faktoru je možné testovat vliv více faktorů a jejich kombinací. Představme si, že chceme testovat vliv dvou různých sloučenin na růst tkáňových buněk. Tyto buňky vyžadují nějaký metabolit, který může být synthetisován dvěma různými metabolickými drahami. Pokud k buňkám přidáme inhibitor jedné nebo druhé metabolické dráhy, pak je možné předpokládat malý nebo žádný vliv na růst buněk, neboť inhibice jedné metabolické dráhy bude kompenzována druhou drahou. Podstatného utlumení růstu je možné dosáhnout pouze současným působením inhibitorů obou drah. Design pokusu může vypadat například takto: k první kultuře nebude přidáván žádný inhibitor, k druhé bude přidán inhibitor A, ke třetí inhibitor B a ke čtvrté budou přidány oba inhibitory současně. Pro každý ze čtyř vzorků budou provedena tři biologická opakování. Vygenerujme si modelová data:

```
> none <- rnorm(3, mean=10)
> justA <- rnorm(3, mean=10)
> justB <- rnorm(3, mean=10)
> AandB <- rnorm(3, mean=4)
> vsechno <- c(none, justA, justB, AandB)
> boxplot(none, justA, justB, AandB)
```

Přidáme faktory a vše uložíme do struktury indata typu data.frame:

```
> addedA <- as.factor(c("n","n","n","y","y","y","n","n","n","y","y","y"))
> addedA
 [1] n n n y y y n n n y y y
Levels: n y
> addedB <- as.factor(c("n","n","n","n","n","n","y","y","y","y","y","y"))
> addedB
 [1] n n n n n n y y y y y y
Levels: n y
> indata <- data.frame(addedA, addedB, vsechno)
> indata
```

```
   addedA addedB   vsechno
1       n      n  9.025124
2       n      n 10.572969
3       n      n  8.871044
4       y      n 11.239133
5       y      n  9.738088
6       y      n 10.707252
7       n      y 11.012759
8       n      y  8.819868
9       n      y 10.548955
10      y      y  4.405583
11      y      y  5.804360
12      y      y  4.070786
```

Nakonec provedeme analysu rozptylu:

```
> m1 <- lm(vsechno~addedA+addedB, data=indata)
> m1

Call:
lm(formula = vsechno ~ addedA + addedB, data = indata)

Coefficients:
(Intercept)      addedAy      addedBy
     10.804       -2.929       -2.705
> anova(m1)
Analysis of Variance Table

Response: vsechno
          Df Sum Sq Mean Sq F value  Pr(>F)
addedA     1 25.736  25.736  5.5824 0.04241 *
addedB     1 21.954  21.954  4.7620 0.05697 .
Residuals  9 41.491   4.610
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Toto provedení analysy rozptylu odpovídá jednofaktorové analyse, neboť testujeme jestli růst buněk závisí na přídavku A a na přídavku B, ale nikoliv na jejich kombinaci. Jinými slovy prokládáme data rovnicí:
$vsechno = a \cdot addedA + b \cdot addedB + c$
kde $addedA$ a $addedB$ zaujímá hodnoty 0 nebo 1, podle toho jestli byla sloučenina A

respektive B přidána. Jak ale tušíme, nezávisí pouze na tom, jestli daná sloučenina byla přidána, ale také na tom, jestli byly sloučeniny přidány současně. V řeči modelu by to vypadalo takto:
$vsechno = a \cdot addedA + b \cdot addedB + c \cdot addedA \cdot addedB + d$
kde součin addedA s addedB nabývá hodnotu 1 pokud jsou přidány obě sloučeniny. V jazyce R je tento model vyjádřen zápisem `vsechno~addedA*addedB`. Tento zápis je ekvivalentní zápisu `vsechno~addedA+addedB+addedA:addedB` (více o zápisu modelů bude v tabulce 13.1). Analysu rozptylu tedy provedeme takto:

```
> m2 <- lm(vsechno~addedA*addedB, data=indata)
> m2

Call:
lm(formula = vsechno ~ addedA * addedB, data = indata)

Coefficients:
    (Intercept)          addedAy          addedBy  addedAy:addedBy
         9.1312           0.4169           0.6407          -6.6918

> anova(m2)
Analysis of Variance Table

Response: vsechno
              Df Sum Sq Mean Sq F value    Pr(>F)
addedA         1 25.736  25.736  26.040 0.0009265 ***
addedB         1 21.954  21.954  22.213 0.0015157 **
addedA:addedB  1 33.585  33.585  33.981 0.0003919 ***
Residuals      8  7.907   0.988
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Je vidět, že přídavek prvku `addedA:addedB`, tedy takzvané interakce posílil výsledný model. O tom se můžeme přesvědčit tak, že oba modely porovnáme pomocí funkce `anova`:

```
> anova(m1,m2)
Analysis of Variance Table

Model 1: vsechno ~ addedA + addedB
Model 2: vsechno ~ addedA * addedB
  Res.Df    RSS Df Sum of Sq      F    Pr(>F)
```

```
1      9 41.491
2      8  7.907  1    33.585 33.981 0.0003919 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Z výsledku (nízká p-hodnota) vyplývá, že druhý model je signifikantně lepší než první. To je logické, neboť je růst buněk inhibován přídavkem obou sloučenin najednou, což první model ignoruje. Obecně platí, že pokud vytváříme složitější a složitější modely pro popis experimentálních dat, pak nám tyto modely budou data lépe a lépe prokládat. Zároveň ale zvyšujeme počet stupňů volnosti a s nějakým obrovským bychom mohli proložit cokoliv. Právě analysa rozptylu může sloužit k tomu, abychom mohli z modelu vypustit všechny nepotřebné prvky, které příliš nezlepšují jeho kvalitu, ale přidávají stupně volnosti navíc. Je tedy možné navrhnou ještě jednoduší model $vsechno = a \cdot addedA \cdot addedB + b$. Pro takovýto model si musíme vytvořit faktor `addedboth`:

```
> addedboth <- as.factor(c(rep("n", times=9), rep("y", times=3)))
> addedboth
 [1] n n n n n n n n n y y y
Levels: n y
```

který zaujímá hodnotu `n` pokud není přidán žádný nebo jen jeden inhibitor a `y` pokud jsou přidány oba (model nejde zapsat jako `vsechno~addedA:addedB`, protože ten je ekvivalentní `vsechno~addedA*addedB`, zkoušel jsem to). Analysu rozptylu provedeme obvyklým způsobem:

```
> m3<-lm(vsechno~addedboth)
> m3

Call:
lm(formula = vsechno ~ addedboth)

Coefficients:
(Intercept)   addedbothy
      9.484       -5.987

> anova(m3)
Analysis of Variance Table

Response: vsechno
           Df Sum Sq Mean Sq F value    Pr(>F)
addedboth   1 80.640  80.640  94.414 2.067e-06 ***
Residuals 10  8.541   0.854
```

```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Oba modely poté můžeme porovnat funkcí `anova`:

```
> anova(m3,m2)
Analysis of Variance Table

Model 1: vsechno ~ addedboth
Model 2: vsechno ~ addedA * addedB
  Res.Df    RSS Df Sum of Sq     F Pr(>F)
1     10 8.5411
2      8 7.9067  2    0.6344 0.321 0.7344
```

Nízká p-hodnota značí, že snížení rozptylu použitím složitějšího modelu není signifikantní. Jinými slovy nemáme dostatek důkazů pro to, abychom předpokládali, že složitější model `vsechno~addedA*addedB` vystihuje data lépe než model `vsechno~bothadded`. Ještě poznámka, pokud byste nechtěli vytvářet speciální faktor `bothadded`, pak je možnost využít zápisu `vsechno~I(addedA*addedB)`, který ale funguje pouze pokud `addedA` a `addedB` budou nabývat hodnoty 1 a 0, nikoliv `y` a `n` (viz kapitola věnovaná regresi).

Použití analysy rozptylu při hledání nejlepšího modelu si ještě ukážeme na příkladech lineární a nelineární regrese. Dalším nástrojem, který je možné použít při zjednodušování složitých modelů, je Akaikeho informační kritérium. V R pro něj existuje funkce `AIC`. Pokud někoho vědecká kariéra zavede do oblasti, kde bude muset hodnotit a zjednodušovat model, pak mu doporučuji zaměřit svoji pozornost i na tuto funkci.

Při použití metody ANOVA bychom měli mít na paměti, že výběry mají mít normální rozdělení. Neparametrickým zobecněním analýzy rozptylu je Kruskalův-Wallisův test, který je možné v programu R provést funkcí `kruskal.test`.

Kapitola 14

Korekce p-hodnot

V předchozí kapitole jsme jako řešení problému mnohonásobného porovnávání vyzkoušeli analýzu rozptylu. Pomocí funkce aov nebo anova jsme testovali zda jsou všechny soubory stejné nebo zda mezi nimi existuje rozdíl. Pokud nám vyjde, že mezi výběry není rozdíl, pak nemá cenu se jimi dále zabývat. Pokud vyjde, že rozdíl mezi výběry je, pak je možné použili funkci TukeyHSD abychom porovnali každý výběr s každým. Existují ale i další postupy. Postupy, které si ukážeme v této kapitole, jsou založené na provedení mnoha dvouvýběrových t-testů a následné korekci p-hodnot.

Program R obsahuje funkci pairwise.t.test. Ta umožňuje porovnat několik souborů každý s každým. Ukážeme si ji na souboru z klinického testu:

```
labels<-gl(3,10)
vsechno<-c(kontrola, sloucenina, placebo)
> pairwise.t.test(vsechno, labels, p.adjust.method="none", pool.sd=F)

    Pairwise comparisons using t tests with non-pooled SD

data:  vsechno and labels

  1     2
2 0.025 -
3 0.894 0.022

P value adjustment method: none
```

Tato funkce provede t-test každého souboru s každým a vyhodí všechny p-hodnoty v matici. Pokud si vyzkoušíte všechny možné t-testy, pak by měl být výsledek stejný. Pokud vynecháte volbu pool.sd=F, pak program předpokládá, že všechny soubory mají stejnou

směrodatnou odchylku, vypočte její odhad a použije ji pro t-testy:

```
> pairwise.t.test(vsechno, labels, p.adjust.method="none")

    Pairwise comparisons using t tests with pooled SD

data:  vsechno and labels

  1     2
2 0.018 -
3 0.891 0.025

P value adjustment method: none
```

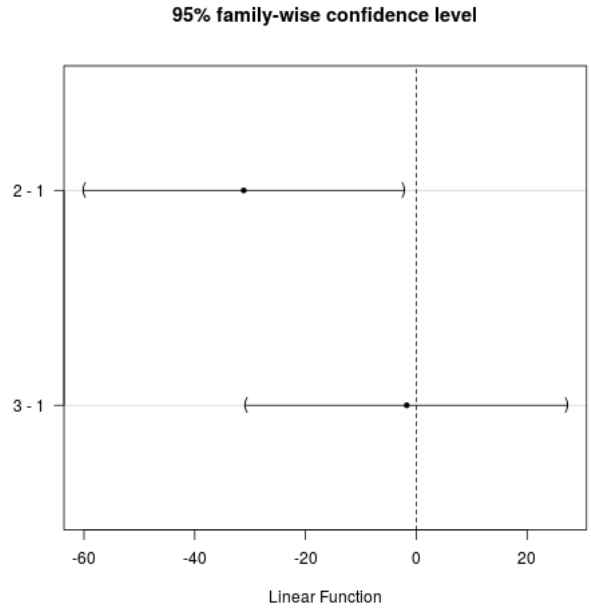
Takto získané p-hodnoty jsou nesprávné kvůli problému mnohonásobného porovnávání. Funkce pairwise.t.test obsahuje několik korekcí p-hodnot, které si můžeme vypsat pomocí help(p.adjust.methods). Nejstarší metoda je Bonferroniho korekce (”bonferroni”), která je uvedena spíše z historických důvodů a byla překonána. Tato metoda spočívá v tom, že p-hodnoty jsou vynásobeny počtem porovnávání (pokud přesáhne násobek hodnotu jedna, pak je automaticky jedna). Pro náš soubor vyjde:

```
> pairwise.t.test(vsechno, labels, p.adjust.method="bonferroni")

    Pairwise comparisons using t tests with pooled SD

data:  vsechno and labels

  1     2
2 0.055 -
3 1.000 0.075

P value adjustment method: bonferroni
```

Novější metoda je podle Holma a Bonferroniho (”holm”). Spočívá v tom, že se nejnižší p-hodnota násobí počtem porovnávání, druhá nejnižší se násobí počtem porovnávání mínus jedna atd:

```
> pairwise.t.test(vsechno, labels, p.adjust.method="holm")

    Pairwise comparisons using t tests with pooled SD

data:  vsechno and labels
```

```
  1     2
2 0.055 -
3 0.891 0.055

P value adjustment method: holm
```



Obr. 14.1 Grafická representace výsledků Dunnettova testu

Asi nejpoužívanější korekční metodou v biologických vědách je dnes metoda podle Benjaminiho a Hochberga. Pokud například otestujeme 10 000 protinádorových sloučenin a touto metodou na hladině pravděpodobnosti 95 % identifikujeme 100 aktivních molekul a nakonec těchto 100 molekul znovu otestujeme, pak by nám tento test měl potvrdit aktivitu u přibližně 95 z nich a přibližně 5 by mělo být falešně positivních. Metoda podle Benjaminiho a Hochberga se použije pomocí volby ”BH”nebo ”fdr”(jako false discovery rate):

```
> pairwise.t.test(vsechno, labels, p.adjust.method="BH")

    Pairwise comparisons using t tests with pooled SD

data:  vsechno and labels

  1     2
2 0.038 -
3 0.891 0.038

P value adjustment method: BH
```

Zatím všechny metody porovnávali každý soubor s každým. V biologických vědách se často setkáme s porovnáváním velké série souborů, které například odpovídají různým testovaným sloučeninám, s kontrolním experimentem. Pro tento účel existuje neprávem opomíjený Dunnettův test. Pro jeho použití potřebujeme balíček multcomp. Ukážeme si jej na datech z klinického testu sloučeniny. Nejprve si aktivujeme balíček multcomp a vytvoříme si data.frame:

```
> require(multcomp)
> mydata <- data.frame(labels, vsechno)
```

Pak musíme programu říct co je kontrola:

```
> mydata$labels <- relevel(mydata$labels, ref=1)
```

Nakonec provedeme analýzu rozptylu, vypočteme p-hodnoty, intervaly spolehlivosti a nakreslíme graf:

```
> mydata.aov <- aov(vsechno ~ labels, data=mydata)
> mydata.dunnett <- glht(mydata.aov, linfct = mcp(labels="Dunnett"))
> summary(mydata.dunnett)

     Simultaneous Tests for General Linear Hypotheses

Multiple Comparisons of Means: Dunnett Contrasts


Fit: aov(formula = vsechno ~ labels, data = mydata)

Linear Hypotheses:
           Estimate Std. Error t value Pr(>|t|)
2 - 1 == 0  -31.150     12.413  -2.510    0.034 *
3 - 1 == 0   -1.723     12.413  -0.139    0.986
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Adjusted p values reported -- single-step method)
> confint(mydata.dunnett)

     Simultaneous Confidence Intervals

Multiple Comparisons of Means: Dunnett Contrasts


Fit: aov(formula = vsechno ~ labels, data = mydata)
```

```
Quantile = 2.3334
95% family-wise confidence level


Linear Hypotheses:
           Estimate lwr      upr
2 - 1 == 0 -31.1501 -60.1141  -2.1860
3 - 1 == 0  -1.7229 -30.6869  27.2412

> plot(mydata.dunnett)
```

Kromě výše uvedených korekcí existují další, například založené na Bayesovské statistice (podmíněné pravděpodobnosti), které se využívají při zpracování dat z microarray a podobných experimentů.

Kapitola 15

Grafická representace statistických testů v biologických vědách

V biologických vědách, konkrétně v biochemii, molekulární a buněčné biologii, je používán velmi oblíbený, ale do určité míry specifický způsob jak graficky prezentovat výsledky. Představte si, že chcete v odborném článku presentovat vliv různých sloučenin na růst buněk. Provedete kultivaci buněk bez jakékoliv přidané sloučeniny a s přídavkem jednotlivých sloučenin. U každého pokusu provedete čtyři opakování. Pak vyhodnotíte růst a z výsledků vypočtete průměr, směrodatnou odchylku, případně střední chybu průměru. Výsledky je potom možné vynést ve formě sloupcového grafu, kde každý sloupec bude odpovídat jednotlivým sloučeninám spolu s jedním sloupcem pro kontrolu. Pokud bude výška sloupce pro nějakou sloučeninu porovnatelná s kontrolou, pak to znamená, že sloučenina nemá žádný vliv na růst. Pokud bude sloupec miniaturní, pak se jedná o silný inhibitor růstu a možná například i potenciální protinádorové léčivo.

Je ale záhodno nějak graficky vyjádřit přesnost dat. Obvykle se k tomuto účelu používají chybové úsečky. Chybové úsečky mohou představovat buď směrodatné odchylky nebo střední chyby průměru. První veličinu použijeme v případě, že chceme vyjádřit variabilitu dat. Střední chybu průměru bychom použili v případě, že chceme vyjádřit přesnost dat (což by byl asi i případ našich buněk). Je možné se setkat i s jinými veličinami vynesenými jako chybové úsečky. V každém případě člověk nic nezkazí tím, že do popisku grafu uvede jaké veličiny byla vyneseny jako chybové úsečky.

Samotné chybové úsečky nemohou nahradit testování hypotes pomocí t-testu, ana-

lysy rozptylu a dalších metod. Tím se dostáváme ke zvláštnosti grafů používaných v biologických vědách. Velmi často se setkáme s tím, že nad jednotlivými sloupci najdeme jednu, dvě nebo tři hvězdičky, případně zkratku „N.S“. To znamená, že byl proveden test statistické hypotesy (zase je vhodné uvést jaký) a jeho výsledky jsou vyjádřeny těmito symboly. Tři hvězdičky obvykle značí P-hodnotu 0 až 0,001, dvě hvězdičky značí 0,001 až 0,01 a jedna hvězdička 0,01 až 0,05. Zkratka N.S. značí not significant, tedy více než 0,05. Někdy je přímo uvedena P-hodnota, například „P = 0.021“. Význam hvězdiček ale může být i jiný a není na škodu jej vysvětlit v popisku grafu. Pokud se hvězdičky vyskytují nad jednotlivými sloupci ve sloupcovém grafu, pak to znamená, že byl proveden test, který porovnal data odpovídající jednotlivým sloupcům s vhodným referenčním pokusem (v našem případě s neošetřenými buňkami). Jindy jsou pomocí statistických testů porovnávány data odpovídající jednotlivým sloupcům. Pak je v grafu přidána vodorovná přímka nebo jakýsi můstek, který spojuje dva sloupce, a hvězdičky (nebo „N.S.“) jsou uvedeny nad ním. Mimo sloupcových grafů se s hvězdičkami setkáme i u box-plotů a dalších grafu.

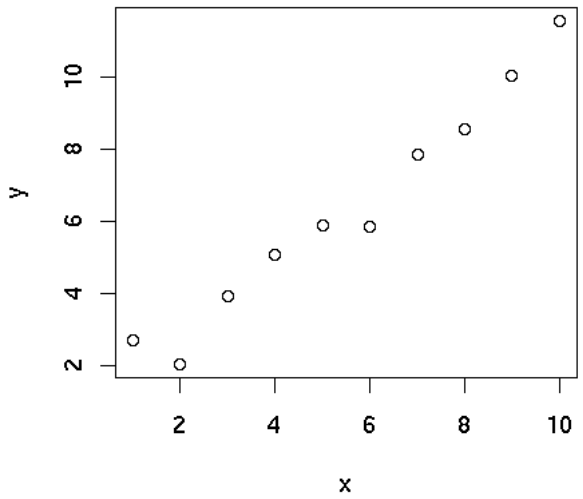
Jak bylo uvedeno v úvodu této kapitoly, tento způsob zobrazování výsledků je specifický pro molekulární biologii a nepoužívají jej pravověrní chemici, fyzici, matematici, statistici, dokonce ani bioinformatici. Program R vychází z komunity statistiků a do biologických věd jej zavlekli bioinformatici. Vzhledem k tomu, že ani jedna z těchto skupin nemá vřelý vztah k hvězdičkám v grafech, není tato možnost v R podporována. Proto jsem se pokusil tuto možnost, alespoň provizorně do R přidat. Prozatímní výsledek této snahy předkládám na stránkách `http://web.vscht.cz/spiwokv/rasterisk.html`. Velmi ocením jakékoliv náměty a připomínky, které mohou vést k tomu, že v budoucnosti bude tato snaha přetransformována do formy balíčku v R.

Kapitola 16

Popisná vícerozměrná statistika

Dříve než se vrhneme na výklad o lineární a nelineární regresi, tak si představíme dvě základní veličiny popisné statistiky vícerozměrných dat, a to korelací a kovariancí. Tyto dvě veličiny bývají tím prvním na co se člověk podívá, když hledá vztahy mezi veličinami. Nejprve si vytvoříme modelová data:

```
> x<-1:10
> y<-2:11+rnorm(10, sd=0.5)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> y
 [1]  2.709754  2.048211  3.947423  5.087165  5.889646  5.869065  7.855641
 [8]  8.561714 10.018594 11.542838
> plot(x,y)
```



Obr. 16.1 Modelová data

Kovarianční koeficient vypočteme „ručně“ takto:

Tabulka 16.1 Korelace a kovariance

veličina	R	vzoreček
kovariance	`cov()`	$cov(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{N}(x-\mu(\mathbf{x}))(y-\mu(\mathbf{y}))}{N-1}$
korelace	`cor()`	$cor(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{N}(x-\mu(\mathbf{x}))(y-\mu(\mathbf{y}))}{\sqrt{\sum_{i=1}^{N}(x-\mu(\mathbf{x}))^2\sum_{i=1}^{N}(y-\mu(\mathbf{y}))^2}}$

```
> sum((x-mean(x))*(y-mean(y)))/(length(x)-1)
[1] 9.258152
```

Korelační koeficient (také Pearsonův korelační koeficient) vypočteme takto:

```
> sum((x-mean(x))*(y-mean(y)))/sqrt(sum((x-mean(x))^2)*sum((y-mean(y))^2))
[1] 0.9826675
```

Samozřejmě program R má pro obě veličiny své funkce:

```
> cov(x,y)
[1] 9.258152
> cor(x,y)
[1] 0.9826675
```

Rozdíl mezi korelací a kovariancí je ten, že kovariance je veličinou absolutní, kdežto korelace je relativní. Korelační koeficient je možné vypočítat také vydělením kovariance směrodatnými odchylkami obou veličin:

```
> cov(x,y)/(sd(x)*sd(y))
[1] 0.9826675
```

Funkce `cov` a `cor` je možné použít i ve spojení s objekty `data.frame` a `matrix`. V tom případě vrátí program kovarianční, respektive korelační matici, tedy vypočte kovarianci/korelaci každého sloupce s každým.

Kapitola 17

Lineární regrese

Pro vlastní lineární regresi má program R funkci `lm`, čili linear model. Ta umožňuje prokládat data lineární regresí a to jak funkcí jedné, tak i dvou a více proměnných. Umožňuje i použít polynomiální regresi a podobné regrese, kde je možné funkci lineárně zkombinovat z více funkcí. Jak bylo vidět na příkladu analysy rozptylu, funkce `lm` má daleko širší použití. Por lineární regresi modelových dat z předchozí kapitoly je možné použít tento postup:

```
> linfit <- lm(y~x)
> linfit

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
     0.7981       1.0100

> summary(linfit)

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-0.9889 -0.2403  0.0805  0.2195  0.9017

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
```

Tabulka 17.1 Příklady lineárních modelů v R

vzoreček	R
$f(x) = \alpha$	`y~1`
$f(x) = \alpha + \beta x$	`y~x`
$f(x) = \beta x$	`y~-1 + x`
$f(x) = \alpha + \beta x + \gamma x^2$	`y~x+I(x^2)`
$f(x) = \alpha + \beta_1 x_1 + \beta_2 x_2$	`y~x1+x2`
$f(x) = \alpha + \beta_1 x_1 + \beta_2 x_2 + \gamma x_1 x_2$	`y~x1*x2`

```
(Intercept)  0.79811    0.41797   1.909   0.0926 .
x            1.00998    0.06736  14.993 3.87e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6118 on 8 degrees of freedom
Multiple R-Squared: 0.9656,     Adjusted R-squared: 0.9613
F-statistic: 224.8 on 1 and 8 DF,  p-value: 3.867e-07
```

Tím proložíme data modelem $f(x) = \alpha + \beta x$. Koeficient β je ve výsledku označen `x` a má hodnotu 1,00998. Koeficient α je označen jako `Intercept` (zbytek na ose y) a má hodnotu 0,79811. K oběma veličinám je možné nalézt střední chyby (`Std. Error`). Někomu může připadat poněkud zvláštní zápis `y~x`. Program R má pro definování modelů tento zvláštní jazyk. V Tabulce uvádím příklady některých modelů a jejich zápisu v R:

Pokud se chceme dostat k hodnotám koeficientů, můžeme učinit například toto:

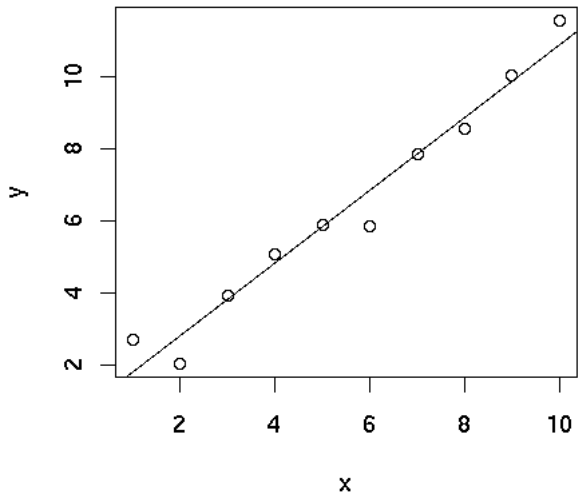
```
> linfit$coefficients[1]
(Intercept)
   0.7981139
> linfit$coefficients[2]
        x
   1.009980
```

nebo použít funkci `coef`:

```
> coef(linfit)[1]
(Intercept)
   0.7981139
> coef(linfit)[2]
       x
1.009980
```

Tyto veličiny můžeme použít k nakreslení přímky, která prokládá data, nebo jednodušeji můžeme použít funkci `abline`:

```
> plot(x,y)
> abline(linfit)
```



Obr. 17.1 Proložení dat funkcemi `lm` a `abline`

Dosud jsme používali analysu rozptylu pro nespojité nezávisle proměnné, tedy faktory. Například při hledání rozdílů mezi pacienty, jimž bylo podáváno léčivo, placebo nebo nic, jsme měli nezávisle proměnnou – faktor, který může nabývat tří nespojitých hodnot pro léčivo, placebo nebo nic. Proč ale nevyužít analysu rozptylu pro spojitá data? Funkce `lm` slouží k vytváření lineárních modelů a „je jí jedno“, jestli nezávisle proměnná veličina je nebo není spojitá. Analysa rozptylu pro testování vlivu léčiva prokládá data funkcí:
účinek = $a \cdot$ *léčivo* + $b \cdot$ *placebo* + c
kde proměnné *léčivo* a *placebo* nabývají hodnot 0 nebo 1. Stejně tak je možné využít analysu rozptylu pro spojité nezávisle proměnné.

Tato vlastnost se hodí pokud chceme zjistit, zdali zesložiťování nějakého modelu má nebo nemá opodstatnění. Pokud například proložíme nějaká naměřená data lineráním modelem ($y = a \cdot x + b$), pak to zkusíme polynomem druhého stupně ($y = a \cdot x^2 + b \cdot x + c$),

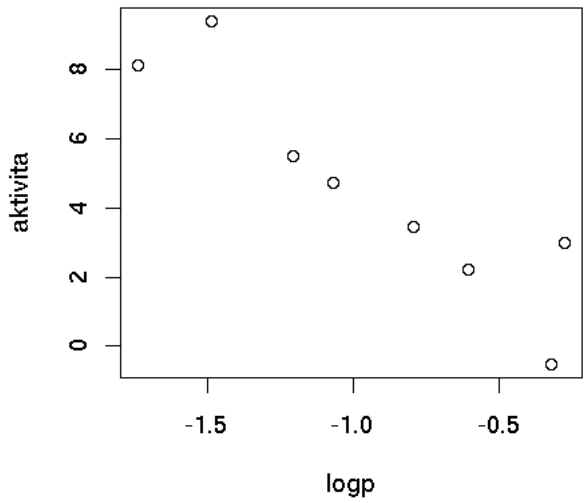
třetího stupně a tak dále, bude nám vycházet, že čím je polynom vyšší tím je proložení dat lepší. Podobně když budeme nějaký regresní model doplňovat jinými funkcemi než jsou polynomy, tak také můžeme pozorovat zlepšování proložení, čili pokles součtu čtverců odchylek. Je ale jasné, že nemá význam zesložiťovat model donekonečna. Místo toho je vhodné nalézt nějaký způsob jak odhalit, zdali nějaký prvek v modelu přináší nebo nepřináší signifikantně lepší proložení. Přesně v tomto duchu funguje analýza rozptylu. V úloze věnované porovnání kontroly, léčiva a placeba jsme porovnali dvě hypotesy, buď že je jedno co pacienti dostávají, nebo na tom záleží. Pro obě tyto hypotesy jsme vypočetli rozptyly a ty jsme porovnali. Podobnou operaci můžeme provést se dvěma regresními modely, například pro model $y = a \cdot x$ a model $y = a \cdot x + b$. Data proložíme pomocí obou modelů, spočítáme rozptyly a porovnáme je. Tak zjistíme, jestli přídavek konstatny b do modelu vedl k signifikantnímu zlepšení modelu, nebo jestli to bylo jen zbytečné zesložitění modelu.

V modelové úloze, na které si ukážeme analysu rozptylu v kombinaci s regresí, nás bude zajímat, jestli účinnost potenciálního léčiva závisí na jeho polárnosti lineárně nebo jestli je lepší použít polynom druhého stupně. Pokus by vypadal tak, že by bylo nejprve nutné připravit sérii derivátů nějaké biologicky aktivní látky, například u nějakého léčiva vyměnit acetylovou skupinu za propionyl, butyryl atd. U každé jednotlivé sloučeniny by pak bylo nutné změřit nebo vypočítat polárnost (nejčastěji *logP*, tedy logaritmus rozdělovacího koeficientu mezi oktanol a vodu) a také otestovat biologickou aktivitu. Připravíme si modelová data, která budou vycházet z lineárního vztahu:

```
> logp <- -0.2*1:8+0.1*rnorm(8)
> aktivita<-1:8+rnorm(8)
> plot(logp, aktivita)
```

Použitím funkcí `lm` a `anova` s lineárním modelem se dozvíme, že na polárnosti molekul záleží:

```
> mod1 <- lm(aktivita~logp)
> mod1

Call:
lm(formula = aktivita ~ logp)

Coefficients:
(Intercept)         logp
    -0.6795      -5.5187
```



Obr. 17.2 Modelová data pro kombinaci regrese a analysy rozptylu

```
> anova(mod1)
Analysis of Variance Table

Response: aktivita
          Df Sum Sq Mean Sq F value   Pr(>F)
logp       1 60.084  60.084  29.709 0.001587 **
Residuals  6 12.135   2.022
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Kromě lineárního modelu chceme otestovat ještě polynom druhého řádu. Pro něj můžeme použít funkci `lm`, protože se jedná o takzvaný obecný lineární model, tedy že závisle proměnnou můžeme vyjádřit jako lineární kombinaci x^2, x^1 a x^0. Model bude vypadat takto:

```
> mod2 <- lm(aktivita~poly(logp,2))
> mod2

Call:
lm(formula = aktivita ~ poly(logp, 2))

Coefficients:
   (Intercept)  poly(logp, 2)1  poly(logp, 2)2
        4.4876         -7.7514          0.5006

> anova(mod2)
Analysis of Variance Table
```

```
Response: aktivita
              Df Sum Sq Mean Sq F value  Pr(>F)
poly(logp, 2)  2 60.334  30.167  12.692 0.01098 *
Residuals      5 11.884   2.377
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Pokud chcete mít v ANOVA tabulce jak prvek pro x tak i pro x^2, zkuste zapsat model jako:

```
> mod2 <- lm(aktivita~logp+I(logp^2))
```

Modely `mod1` a `mod2` můžeme porovnat pomocí funkce `anova`:

```
> anova(mod2, mod1)
Analysis of Variance Table

Model 1: aktivita ~ poly(logp, 2)
Model 2: aktivita ~ logp
  Res.Df     RSS Df Sum of Sq      F Pr(>F)
1      5 11.8839
2      6 12.1346 -1   -0.2506 0.1054 0.7585
```

čímž zjistíme, že zlepšení modelu přídavkem polynomu druhého řádu není signifikantní. Jinými slovy nemáme dostatek důkazů pro to, abychom předpokládali, že binomický model vystihuje experimentální data lépe než lineární model.

Kapitola 18

Nelineární regrese

Klasickým uplatněním nelineární regrese v biologických vědách je prokládání naměřených hodnot vztahem:
$y = \frac{ax}{b+x}$
který funguje na enzymovou kinetiku (model podle Michaelise a Menten), vazbu ligandu na protein a jiné procesy, kdy je nějaké vazebné místo saturováno. Je sice možné tento vztah převést do tvaru: $1/y = (b/a)1/x + 1/a$ a pak lineárně prokládat hodnoty $1/y$ jako funkci $1/x$ (a také se to hodně dělá), ale tento postup zkresluje chyby a může vést ke špatným řešením. Daleko elegantnější je použití nelineární regrese. V programu R se nelineární regrese provádí pomocí funkce `nls` (jako *non-linear least squares* – nelineární metoda nejmenších čtverců). Její použití si ukážeme na modelových datech uložených v souboru. Soubor má následující tvar:

```
1.0     0.56    0.58    0.37    0.39
2.0     0.95    0.94    0.50    0.48
3.0     1.21    1.19    0.57    0.55
4.0     1.38    1.38    0.60    0.60
5.0     1.54    1.51    0.62    0.61
```

První sloupec značí koncentraci substrátu, která odpovídá veličině x. Druhý a třetí sloupeček jsou dvě opakování měření rychlosti reakce, která v rovnici figuruje jako y. Další sloupečky jsou měření v přítomnosti inhibitoru a zatím je budeme ignorovat. Soubor si načteme do R:

```
> indata <- read.table("kinetika.txt")
> indata
  V1   V2   V3   V4   V5
1  1 0.56 0.58 0.37 0.39
2  2 0.95 0.94 0.50 0.48
3  3 1.21 1.19 0.57 0.55
4  4 1.38 1.38 0.60 0.60
```

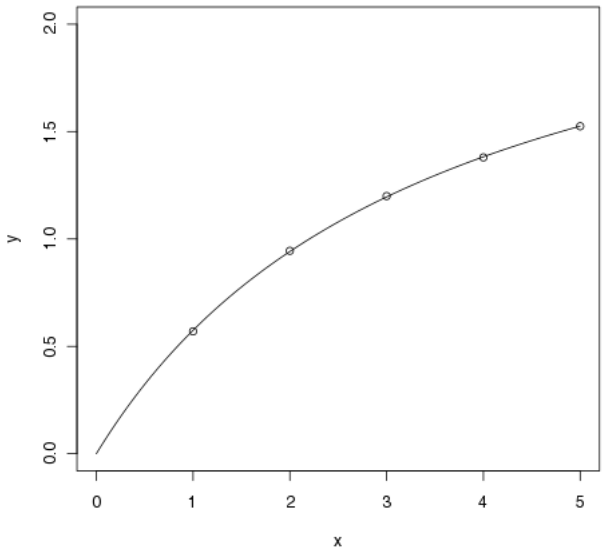
```
5  5 1.54 1.51 0.62 0.61
```

Pak si hodnoty koncentrací nahrajeme do vektoru *x* a průměr rychlostí do *y*:

```
> x <- indata[,1]
> x
[1] 1 2 3 4 5
> y <- (indata[,2]+indata[,3])/2
> y
[1] 0.570 0.945 1.200 1.380 1.525
```

Vzhledem k tomu, že nelineární regrese probíhá na rozdíl od lineární numericky, je nutné na začátku zadat odhady hodnot *a* a *b*. Hodnota *a* (limitní rychlost) by měla být lehce nad hodnotami *y*, takže zvolíme 2. Hodnota *b* (Michaelisova konstanta) by se měla pohybovat někde mezi hodnotami *x*, takže zvolíme také 2. Vlastní regrese probíhá takto:

```
> nlsfit <- nls(y~a*x/(b+x), start=list(b=2, a=2))
> nlsfit
Nonlinear regression model
  model:  y ~ a * x/(b + x)
   data:  parent.frame()
    b     a
3.522 2.600
 residual sum-of-squares: 5.897e-05

Number of iterations to convergence: 4
Achieved convergence tolerance: 5.443e-07
```



Obr. 18.1 Proložení dat neinhibované reakce funkcí `nls`

K přesnostem hodnot a ke středním chybám se dostaneme funkcí `summary`:

```
> summary(nlsfit)

Formula: y ~ a * x/(b + x)

Parameters:
  Estimate Std. Error t value Pr(>|t|)
b  3.52188    0.06159   57.18 1.18e-05 ***
a  2.60025    0.02322  112.01 1.57e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.004433 on 3 degrees of freedom

Number of iterations to convergence: 4
Achieved convergence tolerance: 5.443e-07
```

A vše si můžeme nakreslit takto:

```
> plot(x,y)
> t <- 0:500/100
> lines(t, coef(nlsfit)["a"]*t/(coef(nlsfit)["b"]+t))
```

Podobný výpočet je možné provést i pro inhibovanou reakci (další dva sloupce v souboru).

Při studiu inhibice enzymů nás často zajímá, jestli je testovaný inhibitor kompetitivní, nekompetitivní nebo akompetitivní. To se dá poznat z toho, jak se přídavkem inhibitoru snižují nebo zvyšují hodnoty v_{lim} a K_M (u nás a a b). Podívejme se tedy na naše výsledky:

```
> nlsfit
Nonlinear regression model
  model:  y ~ a * x/(b + x)
   data:  parent.frame()
    b     a
3.522 2.600
 residual sum-of-squares: 5.897e-05

Number of iterations to convergence: 4
Achieved convergence tolerance: 5.443e-07
> yi <- (indata[,4]+indata[,5])/2
> nlsfiti <- nls(yi~ai*x/(bi+x), start=list(bi=2, ai=2))
> nlsfiti
Nonlinear regression model
  model:  yi ~ ai * x/(bi + x)
   data:  parent.frame()
```

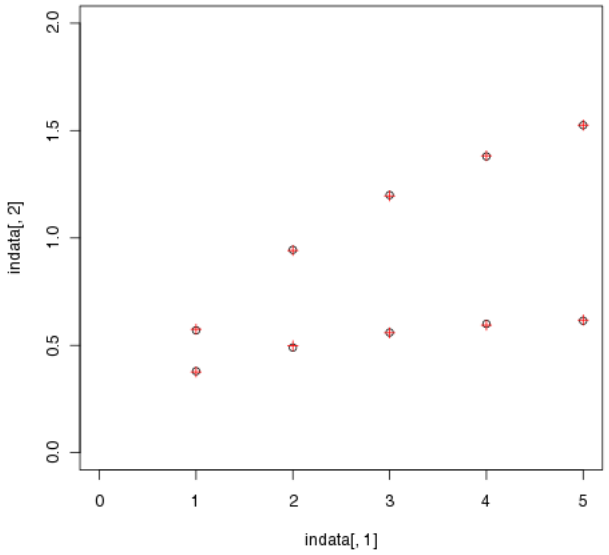
```
    bi     ai
0.9566 0.7362
 residual sum-of-squares: 0.0001247

Number of iterations to convergence: 6
Achieved convergence tolerance: 1.468e-07
```

Obě hodnoty se přídavkem inhibitoru snížili, a to přibližně ve stejném poměru. To odpovídá akompetitivní inhibici. Kvantitativně je možné toto otestovat tak, že porovnáme hodnoty každé veličiny pro inhibovanou a neihibovanou reakci, a to například t-testem. Zde si ale ukážeme ještě rigoróznější postup, a to porovnání nelineárních modelů metodou ANOVA. ANOVA porovnává součet čtverců odchylek, který vyjde za předpokladu, že testovaný faktor bereme a nebereme v úvahu. V tomto případě by testovaným faktorem mohlo být přítomnost inhibitoru. Upravme si testovaná data do tabulky, kde budou hodnoty rychlostí reakcí v jednou sloupečku a navíc přibude sloupeček vyjadřující přítomnost inhibitoru.

```
> x <- c(indata[,1], indata[,1])
> ys<-c(y,yi)
> isinh<-c(rep(0, times=5), rep(1, times=5))
> indata <- data.frame(x,ys,isinh)
> indata
   x    ys isinh
1  1 0.570     0
2  2 0.945     0
3  3 1.200     0
4  4 1.380     0
5  5 1.525     0
6  1 0.380     1
7  2 0.490     1
8  3 0.560     1
9  4 0.600     1
10 5 0.615     1
```

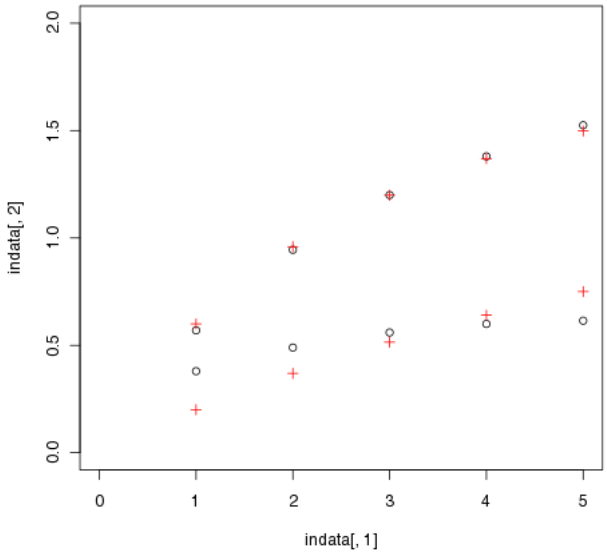
Pak se na celou rovnici můžeme dívat jako na rovnici dvou proměnných: koncentrace substrátu `x` a přítomnosti inhibitoru `isinh`. Pro regresi použijeme model, který bude zahrnovat oba faktory:

```
> nlsfit <- nls(ys~(a+deltaa*isinh)*x/((b+deltab*isinh)+x),
+               data=indata, start=list(b=2, a=2, deltaa=1, deltab=1))

> nlsfit
```



Obr. 18.2 Proložení dat neinhibované a inhibované reakce za předpokladu, že přídavek inhibitoru ovlivňuje jak hodnotu limitní rychlosti, tak i Michaelisovy konstanty (akompetitvní inhibice). Data představují kolečka, model křížky.

```
Nonlinear regression model
  model:  ys ~ (a + deltaa * isinh) * x/((b + deltab * isinh) + x)
   data:  indata
     b      a deltaa deltab
 3.522  2.600 -1.864 -2.565
 residual sum-of-squares: 0.0001836

Number of iterations to convergence: 8
Achieved convergence tolerance: 1.071e-07
> summary(nlsfit)

Formula: ys ~ (a + deltaa * isinh) * x/((b + deltab * isinh) + x)

Parameters:
       Estimate Std. Error t value Pr(>|t|)
b       3.52188    0.07685   45.83 7.23e-09 ***
a       2.60025    0.02897   89.76 1.29e-10 ***
deltaa -1.86400    0.03041  -61.29 1.27e-09 ***
deltab -2.56524    0.08923  -28.75 1.17e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.005532 on 6 degrees of freedom

Number of iterations to convergence: 8
```



Obr. 18.3 Proložení dat neinhibované a inhibované reakce za předpokladu, že přídavek inhibitoru ovlivňuje pouze hodnotu Michaelisovy konstanty (kompetitivní inhibice). Data představují kolečka, model křížky.

```
Achieved convergence tolerance: 1.071e-07
```

Tři hvězdičky u `deltaa` a `deltab` ukazují, že se hodnota `a` a `b` mění s přídavkem inhibitoru. To odpovídá akompetitivní inhibici, kdy se mění jak `a`, tak i `b`. Výsledek si můžeme nakreslit:

```
> plot(indata[,1], indata[,2], xlim=c(0,5), ylim=c(0,2))
> points(indata[,1],
+ (coef(nlsfit)["a"]+coef(nlsfit)["deltaa"]*isinh)*indata[,1]/
+ ((coef(nlsfit)["b"]+coef(nlsfit)["deltab"]*isinh)+indata[,1]),
+ pch=3, col="red")
```

Podobný model můžeme provést například pro kompetitivní inhibici, kdy se mění pouze *b*:

```
> nlsfitkomp <- nls(ys~a*x/((b+deltab*isinh)+x), data=indata,
+  start=list(b=2, a=2, deltab=1))
> plot(indata[,1], indata[,2], xlim=c(0,5), ylim=c(0,2))
> points(indata[,1],
+  coef(nlsfitkomp)["a"]*indata[,1]/
+  ((coef(nlsfitkomp)["b"]+coef(nlsfitkomp)["deltab"]*isinh)+indata[,1]),
+  pch=3, col="red")
```

Jak je vidět, výsledný model prokládá experimentální data mnohem hůře. Kvantitativně to můžeme otestovat tak, že oba modely porovnáme funkcí ANOVA:

```
> anova(nlsfit, nlsfitkomp)
Analysis of Variance Table
```

```
Model 1: ys ~ (a + deltaa * isinh) * x/((b + deltab * isinh) + x)
Model 2: ys ~ a * x/((b + deltab * isinh) + x)
  Res.Df Res.Sum Sq Df    Sum Sq F value    Pr(>F)
1      6   0.000184
2      7   0.070714 -1 -0.070531  2304.5 5.478e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

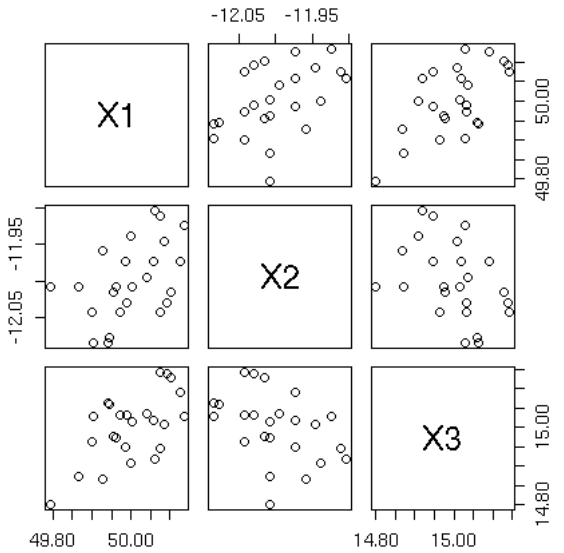
Podobně můžeme vytvořit model `nlsfitnekomp` pro nekompetitivní inhibici (mění se a, nemění b), srovnat jej s modelem `nlsfit` a tak prokázat, zda se jedná o inhibici akompetitivní.

Kapitola 19

Analysa hlavních komponent

S analysou hlavních komponent (principal component analysis, PCA) se setkáme snad ve všech oblastech vědy. Princip této metody si vysvětlíme na poněkud speciálním příkladu. Představte si, že jste na výletě v Českém Ráji a chcete jít na pěší tůru. Protože tento kraj neznáte, bylo by dobré mít k dispozici mapu. Váš kamarád mapu má, ale nechce vám ji půjčit, protože sám tento kraj nezná a chystá se na tůru ve stejný den jako vy. Proto se s ním domluvíte, že vezme svou mapu, pomocí pravítka změří souřadnice různých orientačních bodů, jako jsou vesnice, kopce, pamětihodnosti a rozcestí, dá vám jejich seznam spolu se jejich souřadnicemi a vy si pak budete moci na milimetrovém papíře vytvořit vlastní mapu. Jenže váš kamarád je zvrhlík. Místo toho, aby položil mapu na rovný stůl a pro každý bod změřil souřadnice x a y, zavěsí mapu náhodně do prostoru os x, y a z a pro každý orientační bod změří tři Kartézské souřadnice.

Pokud bychom si vybrali jenom dvojice os x-y, y-z nebo x-z, pak by byla výsledná mapa na milimetrovém papíře výrazně deformovaná. Jak tedy překreslit mapu tak, aby alespoň trochu připomínala původní pomůcku turisty? Je nutné nalézt rovinu, na níž se všechny body nachází, a vyjádřit jejich polohu na této rovině. K tomuto účelu je možné použít analysu hlavních komponent. Intuitivně by bylo možné použít lineární regresi a proložit například hodnoty souřadnic z pomocí vztahu $z = ax + by + c$. Nevýhodou ale je, že v této regresi máme závisle a nezávisle proměnné a klidně se můžeme dostat ke koeficientům a nebo b blízkým nekonečnu. Analysa hlavních komponent představuje jakýsi zobecněný postup. V první kroku je vypočten geometrický střed zvrhlíkovy mapy Českého Ráje. Souřadnice x, y a z tohoto středu vypočteme jako průměry x-ových, y-ových



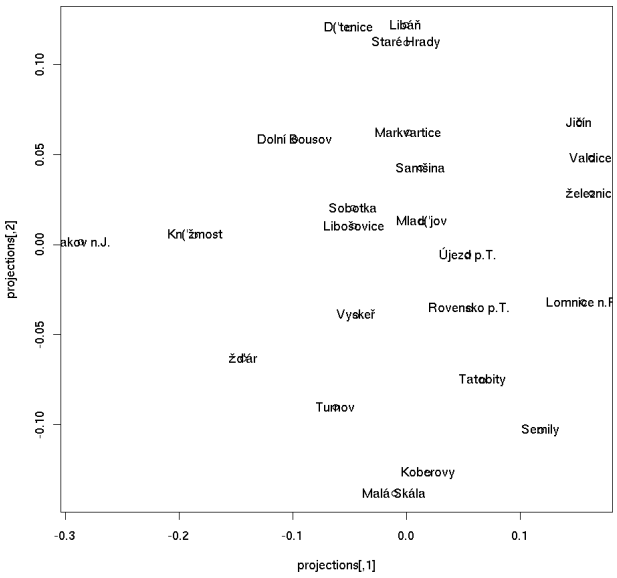
Obr. 19.1 Zvrhlíkova mapa

a z-ových souřadnic přes všechny orientační body. Poté od souřadnic jednotlivých orientačních bodů odečteme souřadnice geometrického středu, tedy posuneme mapu Českého Ráje tak, aby její střed byl v bodě $(0,0,0)$. Díky tomu je problém jak dostat souřadnice do té správné roviny redukován z problému natočení a posunutí na pouhé natočení. Program na základě těchto vycentrovaných souřadnic nalezne tři navzájem kolmé vektory. První z nich vyjadřuje směr, ve kterém jsou body nejvíce rozprostřené. Naopak třetí vektor vyjadřuje směr, ve kterém jsou body nejméně rozprostřené. Pro každý bod můžeme získat nové souřadnice jako vzdálenosti od geometrického středu ve směru prvního, druhého a třetího vektoru. Pokud na milimetrový papír nebo do grafu vyneseme první dvě z těchto nových souřadnic, získáme kýženou plochou mapu.

Mapa může být vůči originálu natočena tak, aby na ose x byla delší než na ose y. Pokud bychom místo Českého Ráje analyzovali například americký stát Tennessee, pak by jsme získali mapu jak má být, protože je tento stát natažený od východu k západu a na nové mapě by byl natažený podél osy x. Pokud bychom analyzovali Itálii, pak by byla natočená o přibližně 90 stupňů, protože je natažená od severu k jihu. Nová mapa Itálie by vypadala tak, že bychom měli vpravo jih a vlevo sever nebo obráceně (k tomu se ještě dostaneme).

Nyní si ukážeme tuto analysu v programu R. Načteme si modelovou zvrhlíkovu mapu ze souboru a uděláme si obrázek:

```
> zvrhlikovamapa<-read.table("zvrhlikovamapa.txt", header=TRUE)
> zvrhlikovamapa
      body.místo       X1        X2       X3
1   Dolní Bousov 49.90128 -12.04325 14.96228
```



Obr. 19.2 Projekce zvrhlíkovy mapy do dvourozměrného prostoru pomocí analysy hlavních komponent (omluvte fakt, že si R neporadí se znakem ě)

```
2       Kněžmost 49.86431 -12.00842 14.87273
3     Bakov n.J. 49.79464 -12.00842 14.80099
...
23     Železnice 50.10301 -12.01539 15.12880
24       Valdice 50.09272 -12.02932 15.13880
> plot(zvrhlikovamapa[,2:4])
```

Jak vidíte, pro popis bodů potřebujeme znát všechny tři souřadnice, pouhé dvojice souřadnic dávají deformovanou mapu. Nyní si ukážeme analysu hlavních komponent „ručně“. Nejprve posuneme souřadnice do geometrického středu pomocí příkazu `scale`, který umožňuje data buď centrovat (odečíst průměry, náš případ) nebo škálovat (odečíst průměry a výsledné souřadnice vydělit odhady směrodatných odchylek, to my nechceme). Pro vycentrování použijeme tento příkaz:

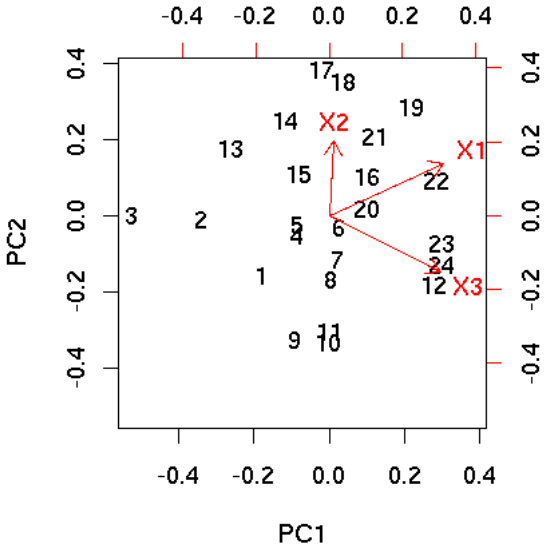
```
> coordinates<-cbind(scale(zvrhlikovamapa[,2],center=TRUE,scale=FALSE),
+                    scale(zvrhlikovamapa[,3],center=TRUE,scale=FALSE),
+                    scale(zvrhlikovamapa[,4],center=TRUE,scale=FALSE))
```

Poté vypočteme kovarianční matici:

```
> covariance<-cov(coordinates)
```

ze které vypočteme funkcí `eigen` vlastní čísla (*eigenvalues*) a vlastní vektory (*eigenvectors*). Pokud nevíte co jsou vlastní čísla a vlastní vektory, tak se podívejte do nějaké učebnice (lineární) algebry. Vlastní vektory jsou právě těmi vektory které hledáme a pomocí nichž získáme požadovanou dvourozměrnou projekci.

```
> evalvecs<-eigen(covariance)
```



Obr. 19.3 Projekce zvrhlíkovy mapy do dvourozměrného prostoru pomocí analysy hlavních komponent (funkcemi `prcomp` a `biplot`)

```
> evalvecs
$values
[1] 1.239185e-02 5.537674e-03 1.734723e-18

$vectors
           [,1]       [,2]       [,3]
[1,] 0.71582688 -0.4876526  0.4997868
[2,] 0.02657276 -0.6961998 -0.7173561
[3,] 0.69777200  0.5267835 -0.4854002
```

Novou mapu získáme tak, že provedeme projekci původní trojrozměrné mapy do dvojrozměrného prostoru prvních dvou hlavních komponent. Prakticky to znamená pro každý bod vypočítat vzdálenost od středu ve směru prvního a druhého vlastního vektoru. Vzhledem k tomu, že vlastní vektory jsou z principu jednotkové (jejich délka je rovná jedné), pak je možné vypočíst projekci tak, že vezmeme pozici bodu a vlastní vektor a vypočteme jejich skalární součin.

```
> projections<-cbind(coordinates%*%evalvecs$vectors[,1],
+                    coordinates%*%evalvecs$vectors[,2])
> plot(projections)
> text(projections, labels=zvrhlikovamapa$body.místo)
```

Pokud se podíváme na obrázek, došlo k „narovnání“ zvrhlíkovy mapy jak bylo předpokládáno. Osa x odpovídá přibližně ose x v originální mapě, neboť Český Ráj, alespoň vybrané obce, je více roztažen ze západu na východ. Osa y je v porovnání s originálem

otočená, tedy máme jih nahoře a sever dole. To se může stát. Můžeme si to představit tak, že metoda analysy hlavních komponent „neví“ jak se má na mapu v trojrozměrném prostoru „dívat“ a náhodou to vyjde tak, ze se na ni „podívá“ z její spodní strany. Proto je obraz zrcadlově otočený podle horizontální osy.

Zatím jsme se věnovali vlastním vektorům, ale ne vlastním číslům. Vlastní čísla jsou kladné a jsou seřazené od nejvyššího po nejnižší. Velikost čísla vyjadřuje míru variability ve směru vlastních vektorů. V případě Českého Ráje bude druhé vlastní číslo o něco málo nižší než první. Třetí vlastní číslo bude naproti tomu téměř nulové. Znamená to, že je Český Ráj je v jednom směru o něco více natažen než ve druhém smeru. Když vše srovnáme se skutečnou mapou tak zjistíme, že je Český Ráj nejvíce natažen od západu k východu, o něco méně od severu k jihu. Téměř nulové třetí vlastní číslo značí, že mapa téměř není variabilní kolmo k zemskému povrchu. Pokud bychom měli plastickou mapu a pokud by se jednalo o více hornatý kraj, nebo pokud bychom uvažovali zakřivení země, pak by třetí vlastní číslo vyšlo o něco větší. Variabilitu ve směru vlastních vektorů je možné vyjadřovat i jinými veličinami.

Ještě si uvedeme jak udělat analysu hlavních komponent nikoliv ručně, ale pomocí speciální funkce: :

```
> zvrhlikovamapa<-read.table("zvrhlikovamapa.txt", header=TRUE)
> pcaresults<-prcomp(zvrhlikovamapa[2:4])
> pcaresults
Standard deviations:
[1] 1.113187e-01 7.441555e-02 2.577062e-14

Rotation:
          PC1        PC2        PC3
X1 0.71582688  0.4876526 -0.4997868
X2 0.02657276  0.6961998  0.7173561
X3 0.69777200 -0.5267835  0.4854002
> biplot(pcaresults)
```

Analysa hlavních komponent neslouží pouze výletníkům s divnými kamarády. Podívejme se na význam této analysy. Zvrhlíkovu mapu tvoří body v trojrozměrném prostoru. Ve skutečnosti tyto data leží v rovině, tudíž jsou dvourozměrná (v případě, že by zvrhlík použil plastickou mapu, pak by byl třetí rozměr nenulový, ale nízký). Analysa hlavních komponent umožňuje projekci trojrozměrných dat do jedno- nebo dvourozměrném prostoru tak, aby tím byly v maximální míře zobrazeny vzdálenosti mezi body a struktura

dat. Místo zvrhlíkovy mapy můžeme použít analysu hlavních komponent na výsledky microarray experimentů. Různým nemocným nebo zdravím jedincům odebereme vzorek určité tkáně, isolujeme mRNA a změříme koncentrace mRNA jednotlivých genů. Každý vzorek (nemocných nebo zdravý jedinec) bude v podobné roli jako je vesnice, kopec nebo rozcestí na zvrhlíkově mapě. Místo tří souřadnic na zvrhlíkově mapě je pro každý vzorek změřena koncentrace několika desítek tisíc genů. Místo trojrozměrného prostoru zvrhlíkovy mapy tedy máme několik-tisíc-rozměrný prostor. Je ale možné předpokládat, že koncentrace jednotlivých mRNA budou spolu nějak souviset, nejlépe že budou korelované. Koncetrace některých mRNA budou klesat respektive růst s fysiologickým stavem buněk. Produkty některých mRNA mohou fungovat jako transkripční faktory nebo jiné proteiny, které přímo nebo nepřímo ovlivňují synthesu jiných mRNA. Proto růst koncentrace takovéto mRNA vede k růstu nebo poklesu koncentrací řady dalších mRNA a jejich koncentrace se v rámci série vzorků stávají korelované, podobně jako jsou korelované souřadnice vesnic, kopců a rozcestí na zvrhlíkově mapě.

Pokud se podíváme na dvou- nebo trojrozměrnou projekci dat z microarray experimentů, mělo by dojít k separaci nemocných a zdravých jedinců, případně i jedinců s různými nemocemi. Můžeme pak provést microarray experimenty s novými pacienty, provést projekci do dvou- nebo trojrozměrného prostoru a podle výsledků zjistit jejich diagnosu. Vlastní vektory nám také mohou naznačit, které geny jsou více exprimovány u nemocných a které u zdravých jedinců. Na základě toho můžeme zjistit vztahy mezi geny, například které geny naleží do společných regulačních kaskád.

V případě zvrhlíkovy mapy nebo microarray experimentů jsme analysovali veličiny stejného charakteru, ať už to byly souřadnice v kilometrech nebo odezva microarray detektoru. Analysa hlavních komponent ale umožňuje analysovat veličiny s různým charakterem. V předchozích ukázkách byly data před vlastní analysou vycentrována. Kromě vycentrování (tedy odečtení průměru) je možné data ještě navíc škálovat, tedy vydělit je směrodatnou odchylkou. To umožní analysovat „jablka“ s „hruškami“. Pro takovouto analysu je možné použít funkci `prcomp` s volbou `scale=TRUE`.

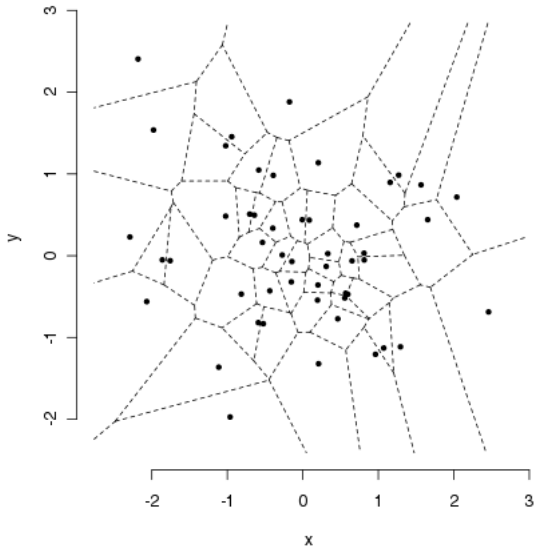
Kapitola 20

Shluková analysa

Shluková (nebo chcete-li klastrová či clusterová) analysa složí ke klasifikaci objektů do shluků (klastrů) a to tak, aby si objekty v jednotlivých shlucích byly více podobné než objeky mezi různými shluky. Shlukovou analysu je možné použít například pokud analysujeme expresi vybraných genů (koncentraci mRNA) pomocí mikročipů či real-time PCR. Tuto analysu provedeme pro několik vzorků tkání pocházejících od zdravých a pro několik vzorků od nemocných jedinců. Pokud bychom porovnávali množství mRNA pouze jednoho genu, pak je malá pravděpodobnost, že by se nám podařilo rozlišit zdravé jedince od nemocných. Když těchto genů proměříme větší počet (klidně i všechny), pak sice máme lepší možnost správně identifikovat zdravé a nemocné, ale dostáváme se do problému s vysokým počtem (neboli s vysokou dimenzionalitou) analysovaných dat. Tento problém může vyřešit analysa hlavních komponent představená v minulé kapitole, nebo shluková analysa.

Objekty je možné shlukovat buď hierarchicky nebo nehierarchicky. Nejdříve si vysvětlíme nehierarchické shlukování, konkrétně metodu K-středů (K-means clustering). Symbol K značí počet shluků. Toto číslo si musíme zvolit před vlastní analysou. S metodou K-středů souvisí takzvaná Voronoiova teselace. Princip tohoto výpočtu je představen na obrázku 16.1. Nejprve náhodně „rozsypeme“ body do dvourozměrného prostoru. Pak kolem bodů uděláme „chlívečky“ tak, aby hranice mezi chlívečky byla přesně mezi nejbližšími sousedními body. V metodě K-středů se snažíme analysovat sérii objektů. Jednotlivými objekty mohou být vzorky pocházející od zdravých jedinců a od nemocných (celkem například patnáct vzorků). Ke každému vzorku máme k dispozici koncentrace

mRNA několika desítek (například třiceti) genů. Každý vzorek je tedy bodem ve třicetirozměrném prostoru. Jak bylo řečeno, nejprve si musíme zvolit počet shluků, tedy hodnotu K. Pokud bychom chtěli například odlišit zdravé jedince od pacientů s mírným a s vážným průběhem nemoci, pak by počet shluků byl zvolen jako tři pro tyto tři skupiny. Jak metoda K-středů funguje? Nejprve rozdělí data náhodně do K, tedy tří, skupin. Pro každou skupiny vypočte střed dat, tedy pro první až třicátý gen vypočte jeho průměrnou koncentraci v první, druhé a třetí skupině. V dalším kroku program provede Voronoiovu teselaci, kterou rozdělí třicetirozměrný prostor na tři části. Dále jsou objekty přeuspořádány do tří nových skupin podle toho v které části prostoru se nacházejí. Pak následuje další vypočtení středů, další Voronoiova teselace a tak dále dokud se složení skupin nemění. Výsledné skupiny jsou kýženými shluky.



Obr. 20.1 Ukázka Voronoiovy teselace ve dvojrozměrném prostoru

Pro ukázku metody K-středů si vygenerujeme data v trojrozměrném prostoru. Deseti bodům schválně dáme takové hodnoty, aby tvořily klastry tvořené třemi, třemi a čtyřmi body:

```
> x1<-rnorm(3, mean=3)
> x2<-rnorm(3, mean=7)
> x3<-rnorm(4, mean=1)
> x<-c(x1,x2,x3)
> y1<-rnorm(3, mean=1)
> y2<-rnorm(3, mean=5)
> y3<-rnorm(4, mean=3)
> y<-c(y1,y2,y3)
```

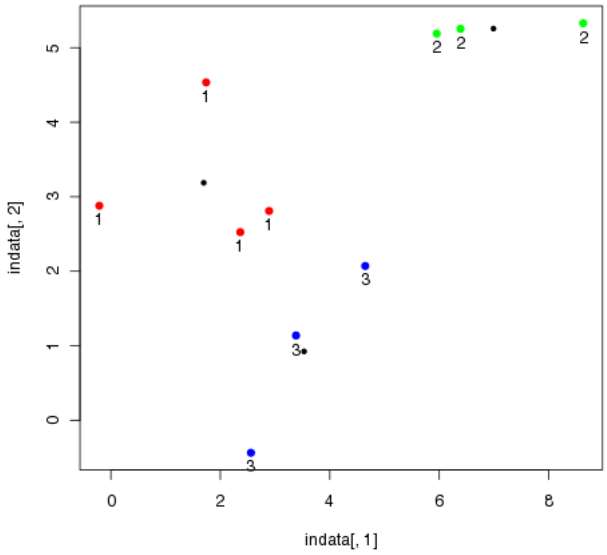
```
> z<-rnorm(10, mean=5)
> indata <- data.frame(x, y, z)
> indata
            x          y        z
1  4.05654058 0.02105170 4.630515
2  2.53129596 1.11930907 6.296396
3  4.18042477 1.90297515 5.087641
4  7.91514528 4.74960380 3.495780
5  7.06424552 4.56567774 3.982281
6  6.89718860 5.62821497 4.685218
7  0.09775463 2.20631035 5.789628
8  1.60428406 3.08544665 5.605750
9  0.54458010 2.15094587 4.534332
10 1.28003417 2.20125972 5.667473
```

Vlastní analysu provedeme funkcí `kmeans` s parametrem `centers=3`:

```
> clusters <- kmeans(indata, centers=3)
> clusters
K-means clustering with 3 clusters of sizes 4, 3, 3

Cluster means:
         x         y        z
1 1.696957 3.1875574 4.872737
2 6.990491 5.2573832 6.137364
3 3.530307 0.9246889 6.356074

Clustering vector:
 [1] 3 3 3 2 2 2 1 1 1 1

Within cluster sum of squares by cluster:
[1] 8.945195 6.204464 8.117839

Available components:
[1] "cluster"  "centers"  "withinss" "size"
>
```

Ve vektoru `clusters$cluster` najdeme přiřazení jednotlivým klastrům. Program správně identifikoval klastry tvořené třemi, třemi a čtyřmi body. Pro vaše data může vyjít jiné pořadí, ale rozdělení do shluků by měly být stejné.

```
> clusters$cluster
 [1] 3 3 3 2 2 2 1 1 1 1
```



Obr. 20.2 Shluková analysa metodou K-středů

Centra klastrů najdeme ve vektoru `clusters$centers`:

```
          x        y        z
1 1.696957 3.1875574 4.872737
2 6.990491 5.2573832 6.137364
3 3.530307 0.9246889 6.356074
```

Nyní si můžeme vykreslit výsledky analysy:

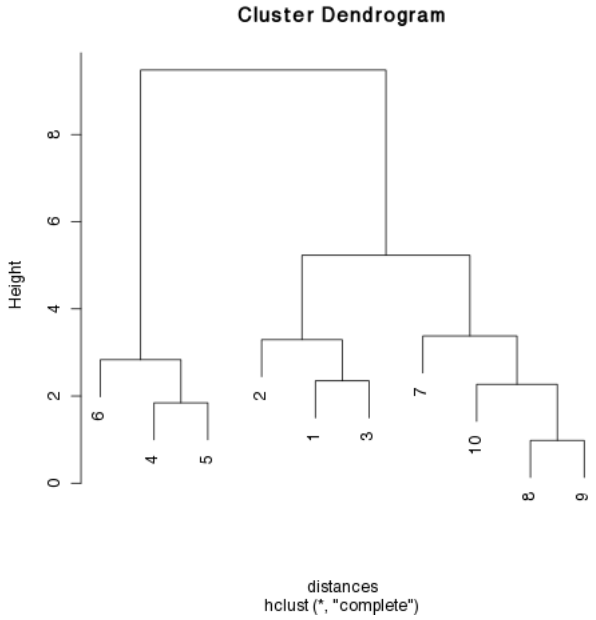
```
> plot(indata[,1], indata[,2], col=rainbow(3)[clusters$cluster], pch=19)
> text(indata[,1], indata[,2], labels=clusters$cluster, pos=1)
> points(clusters$centers[,1], clusters$centers[,2], pch=20)
```

Shlukování metodou K-středů může probíhat některou ze čtyř metod: `Hartigan-Wong`, `Lloyd`, `Forgy` a `MacQueen`, přičemž algoritmus naznačený v úvodu kapitoly odpovídá Lloydově metodě. Ještě bych rád upozornil na knihovnu `cluster`, která umí krásně zobrazit výsledky metody K-středů, k čemuž navíc požívá analysu hlavních komponent.

Místo nehierarchického klastrování metodou K-středů je možné použít některou z metod hierarchického klastrování. Biologové toto velmi dobře znají z fylogenetických analys organismů. Na základě podobnosti sekvencí nukleových kyselin, proteinů nebo na základě jiných parametrů je možné vytvořit „strom života“, na němž jsou si větve odpovídající podobným (a evolučně blízkým) organismům blízké. Myslím, že je tento koncept natolik intuitivní a v biologických vědách vžitý, že jej není nutné dále představovat. Pro hierarchické klastrování má R funkci `hclust`. Navíc budeme potřebovat funkci `dist` pro výpočet vzdálenosti objektů. Tyto funkci si můžeme ukázat na stejných datech:

```
> distances <- dist(indata)
> distances
           1         2         3         4         5         6         7
2  3.2934950
3  2.3482786 2.8270119
4  3.7533895 6.8763172 5.7708785
5  3.4808336 6.6934610 4.8842209 1.8455322
6  5.2017794 8.4593021 6.7849976 2.8333024 2.6795152
7  5.1881382 4.4782313 5.2290252 7.0447943 7.0277189 9.4816452
8  1.9273249 3.2660774 2.6817265 4.3182734 4.0374040 6.3556776 3.3790072
9  2.5625249 3.0312195 3.3016481 4.8622171 4.8572826 7.1056007 2.6594831
10 3.9992626 5.0951760 4.7951260 4.7087353 4.7275869 7.2114199 2.5934062
           8         9
2
3
4
5
6
7
8
9  0.9823218
10 2.2692366 2.1095246
> hierarch1 <- hclust(distances)
> hierarch1

Call:
hclust(d = distances)

Cluster method   : complete
Distance         : euclidean
Number of objects: 10

> plot(hierarch1)
```

Jak je vidět z obrázku, výsledný strom obsahuje tři větve odpovídající správným klastrům. Hierarchické shlukování má na rozdíl od nehierarchického podstatně větší volnost co se týká parametrů metod. První co musíme nastavit jsou parametry funkce `dist`, která počítá vzdálenosti bodů ve vícerozměrném prostoru. Defaultním nastavením je Euklidovská vzdálenost, tedy vzdálenost vypočtená pomocí Pythagorovy věty. Kromě této volby (`method="euclidean"`) je možné použít metody `maximum`, `manhattan`,



Obr. 20.3 Hierarchická shluková analysa

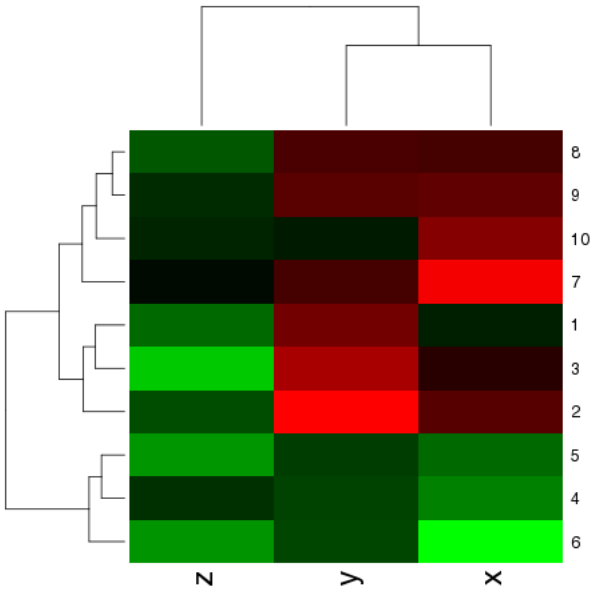
`canberra`, `binary` nebo `minkowski`. Například metoda `manhattan` vypočte vzdálenost mezi body jako součet absolutních hodnot rozdílů souřadnic x, y, z atd., podobně jako by bylo možné vypočítat pěší vzdálenost mezi body na Manhattanu, kde se člověk může pohybovat pouze po pravoúhle uspořádaných ulicích. Dalším nastavením, která může výrazně ovlivnit výsledek, je volba metody shlukování. Funkce `hclust` nabízí možnosti: `ward`, `single`, `complete`, `average`, `mcquitty`, `median` nebo `centroid`. Nechám na čtenářích, aby si vyzkoušeli jednotlivé metody, případně pronikli do jejich tajů.

Jak bylo ukázáno, při hierarchickém shlukování je možné volit různé parametry, hlavně metodu pro výpočet vzdáleností a vlastní shlukovací metodu. Jak ale vybrat tu nejlepší? Určitým vodítkem může být použití kofenetického korelačního koeficientu. Výsledný obrázek hierarchického shlukování „se snaží“ co nejlépe popsat vzdálenosti mezi body. Pokud byste vzali pravítko a měřili délky větviček, pak by mělo být možné se (alespoň přibližně) dopočítat ke vzdálenostem v původním prostoru. Čím lépe bylo shlukování provedeno, tím lepší by měla být shoda mezi vzdálenostmi. Korelaci těchto vzdáleností nazýváme konfenickým korelačním koeficientem a v R ho můžeme vypočítat takto:

```
> hc1<-hclust(dist(indata),method="ward")
> hc2<-hclust(dist(indata),method="single")
> hc3<-hclust(dist(indata),method="complete")
> hc4<-hclust(dist(indata),method="average")
> hc5<-hclust(dist(indata),method="mcquitty")
```

```
> hc6<-hclust(dist(indata),method="median")
> hc7<-hclust(dist(indata),method="centroid")
> cor(dist(indata),cophenetic(hc1))
[1] 0.765884
> cor(dist(indata),cophenetic(hc2))
[1] 0.7425763
> cor(dist(indata),cophenetic(hc3))
[1] 0.7724275
> cor(dist(indata),cophenetic(hc4))
[1] 0.7752757
> cor(dist(indata),cophenetic(hc5))
[1] 0.773544
> cor(dist(indata),cophenetic(hc6))
[1] 0.7620257
> cor(dist(indata),cophenetic(hc7))
[1] 0.7665261
```

Nejlépe tedy pro daná data dopadla metoda `average`, neboť vykazuje nejvyšší hodnotu koeficientu. Nejhůře dopadla metoda `single`.



Obr. 20.4 Heatmap

Podobně jako u analysy hlavních komponent je možné i v případě shlukové analysy sčítat „jablka“ s „hruškami“. Pro vybrané bakterie například zjistíme rychlost růstu v exponenciální fázi na médiu obsahujícím glycerol, kulatost buňky pod mikroskopem, maximální koncentraci antibiotika při které bakterie roste a další z fyzikálního hlediska zcela

různorodé veličiny. Shlukovou analysou těchto veličin chceme vytvořit jakýsi nástroj pro klasifikaci studovaných bakterií. Problém různého charakteru veličin můžeme vyřešit podobně jako v případě analysy hlavních komponent, to znamená pro každou veličinu vypočítat průměr a odhad směrodatné odchylky, pak od každé hodnoty průměr odečíst a výsledek vydělit odhadem směrodatné odchylky. Tak získáme data, která je už možné zpracovat funkcem `kmeans`, `hclust` atd. V R-ku k tomu můžeme použít funkci `scale`. Někdy je možné ještě před tím vybrané veličiny transformovat například logaritmicky, pokud to dovoluje charakter veličiny.

Naprostou lahůdkou na závěr je zobrazení zvané `heatmap`. Toto zobrazení je v současnosti populární při zpracování microarray, proteomických a dalších -omických experimentů. Toto zobrazení vychází z funkce `image`, tedy dvourozměrné různobarevné mřížky. Její sloupce odpovídají jednotlivým veličinám (například mRNA jednotlivých genů). Řádky odpovídají jednotlivým vzorkům (např. pacientům). Měřené hodnoty (tedy v uvedeném případě koncentrace mRNA) jsou vyjádřeny barvou políčka. Nejvíce „frčí“ barevná škála zelená – černá – červená, asi podle barev používaných při fluorescenčním značení biomolekul. Jednotlivé vzorky, stejně tak i jednotlivé veličiny, jsou hierarchicky shluknuty a odpovídající dendrogram je uveden nad a vedle mřížky. Tento graf elegantně ukazuje, které geny a které pacienty je možné seskupit. Malá ukázka pro již vygenerovaná data je zde:

```
> red<-c(100:0/100, rep(0,100))
> green<-c(rep(0,100),0:100/100)
> blue<-rep(0,201)
> heatmap(as.matrix(indata), scale="none", col=rgb(red,green,blue))
```

Kapitola 21

Vybrané funkce v R

fuknce	popis
`AIC`	Akaikeho informační kritérium
`SNA`	balíček pro analýzu sociálních sítí (Social Network Analysis)
`TukeyHSD`	Tukeyův HSD test
`abline`	nakreslení regresní přímky do grafu
`anova`	analýza rozptylu
`aov`	analýza rozptylu
`as.data.frame`	převod na typ data.frame
`as.matrix`	převod na typ matrix
`as.vector`	převod na typ vektor
`axis`	nakreslení os do grafu
`bargraph.CI`	sloupcový graf s chybovými úsečkami
`barplot`	sloupcový graf
`biplot`	graf výsledků PCA
`boxplot`	krabicový graf
`break`	přerušení cyklu

fuknce	popis
`cbind`	připojení sloupců
`cm.colors`	paleta barev cyan-magenta
`contour`	kontury v grafu
`cor`	korelační koeficient
`cov`	kovariance
`data.frame`	vytvoření objektu data.frame
`data`	vypsání modelových sad dat
`dchisq`	hustota Chi-Square rozdělení
`demo`	demo skripty
`dev.off`	přerušení vykreslování grafů do souboru
`df`	hustota Fisherova rozdělení
`dim`	rozměr matice, vektoru atd.
`dist`	vzdálenost mezi vektory, řádky matice a podobně
`dnorm`	hustota normálního rozdělení
`dt`	hustota Studentova rozdělení
`eigen`	výpočet vlastních čísel a vektorů matice
`example`	příklady použití
`for`	cyklus for
`function`	vytvoření funkce
`getwd`	vypsání pracovního adresáře
`ggbio`	balíček pro analýzu genomových dat
`ggplot2`	balíček pokročilých grafů
`gl`	vytvoření vektoru faktorů
`gray`	paleta odstínů šedé
`hclust`	hierarchické klastrování
`head`	vypsání začátku matice, objektu data.frame a podobně
`heat.colors`	paleta barev od chladných po teplé
`heatmap`	graf typu heatmap

fuknce	popis
help	nápověda
hist	histogramy
ifelse	podmínka ifelse
if	podmínka if
igraph	balíček pro využití metod teorie grafů
image	graf matice pixelů
jpeg	uložení obrázku ve formátu jpeg
kmeans	klastrování metodou K-středů
kruskal.test	Kruskalův-Wallisův test
lattice	balíček pokročilých grafů
length	počet prvků vektoru
levels	vypsání počtu hodnot
lineplot.CI	liniový graf s chybovými úsečkami
lines	liniový graf
lm	lineární model
log10	dekadický logaritmus
log2	dvojkový logaritmus
log	přirozený logaritmus
ls	výčet proměnných
maps	balíček pro zobrazování v zeměpisných mapách
maptools	balíček pro zobrazování v zeměpisných mapách
matrix	vytvoření matice
mean	průměr
names	jména sloupců objektu data.frame
next	podmínka next
nlevels	počet hodnot vektoru
nls	nelineární model
order	vypíše indexy podle pořadí hodnot ve vektoru
par	změna parametrů (např. Grafu)
pchisq	pravděpodobnost Chi-Square rozdělení

fuknce	popis
`pdf`	uložení obrázku ve formátu pdf
`persp`	3D graf
`pf`	pravděpodobnost Fisherova rozdělení
`pie`	koláčový graf
`pi`	hodnota pi
`plot`	graf
`png`	uložení obrázku ve formátu png
`pnorm`	pravděpodobnost normálního rozdělení
`points`	přidá body do grafu
`prcomp`	analýza hlavních komponent
`print`	vypíše hodnotu
`ps`	uložení obrázku ve formátu Postscript
`pt`	pravděpodobnost Studentova rozdělení
`qchisq`	kvantily Chi-Square rozdělení
`qf`	kvantily Fisherova rozdělení
`qnorm`	kvantily normálního rozdělení
`qqline`	teoretický průběh normalizovaného QQ výnosu
`qqnorm`	normalizovaný QQ výnos
`qt`	kvantily Studentova rozdělení
`quit`	opuštění prostředí R
`q`	opuštění prostředí R
`rainbow`	paleta duhových barev
`range`	rozsah hodnot
`rbind`	spojení řádků matice nebo objektu data.frame
`rchisq`	náhodná čísla s Chi-Square rozdělením
`read.csv2`	načtení dat CSV
`read.csv`	načtení dat CSV
`read.delim2`	načtení dat s oddělovačem
`read.delim`	načtení dat s oddělovačem
`read.ftable`	načtení dat v prostorově uspořádaném formátu

fuknce	popis
`read.fwf`	načtení dat v prostorově uspořádaném formátu
`read.table`	načtení dat
`repeat`	cyklus repeat
`return`	vrácení hodnoty funkce
`rf`	náhodná čísla s Fisherovým rozdělením
`rgb`	vytvoření barvy z červené, zelené a modré
`rm`	smazání proměnné
`rnorm`	náhodná čísla s normálním rozdělením
`rt`	náhodná čísla se Studentovým rozdělením
`scale`	centrování a/nebo škálování dat
`sciplot`	balíček pokročilých grafů
`sd`	odhad směrodatné odchylky
`setwd`	nastavení pracovního adresáře
`shapiro.test`	Shapirův test normálního rozdělení
`sort`	setřídění hodnot vektoru
`summary`	víceúčelová funkce
`sum`	součet
`svg`	uložení obrázku ve formátu svg
`switch`	přerušení cyklu
`t.test`	Studentův t-test
`table`	tabulka s hodnotami a jejich četností
`tail`	vypsání konce matice, objektu data.frame a podobně
`terrain.colors`	paleta barev jako na mapě
`text`	přidání textu do grafu
`topo.colors`	paleta barev jako na mapě
`t`	transpozice
`var.test`	test shodnosti rozptylů
`while`	cyklus while
`wilcox.test`	Wilcoxonův test

fuknce	popis
`wireframe`	3D graf
`write.table`	zápis do souboru

Rejstřík funkcí