Statistika a programování v R

Laboratorní cvičení

Aleš Kozubík
Žilinská univerzita v Žilině

Project: Innovative Open Source Courses for Computer Science

31. 5. 2021

Innovative Open Source Courses for Computer Science

This teaching material was written as one of the outputs of the project
"Innovative Open Source Courses for Computer Science",
funded by the Erasmus+ grant no. 2019-1-PL01-KA203-065564.
The project is coordinated by West Pomeranian University of Technology in Szczecin (Poland)
and is implemented in partnership with Mendel University in Brno (Czech Republic)
and University of Žilina (Slovak Republic).
The project implementation timeline is September 2019 to December 2022.

Innovative Open Source Courses for Computer Science

Project was implemented under the Erasmus+.
Project name: "Innovative Open Source courses for Computer Science curriculum"
Project no.: 2019-1-PL01-KA203-065564
Key Action: KA2 – Cooperation for innovation and the exchange of good practices
Action Type: KA203 – Strategic Partnerships for higher education

Consortium: Zachodniopomorski uniwersytet technologiczny w Szczecinie
Mendelova univerzita v Brně
Žilinská univerzita v Žiline

Erasmus+ Disclaimer: This project has been funded with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Statistika a programování v R

I. Úvod do prostředí R

Instalace R

Je volně dostupný ze zdroje Comprehensive R Archive Network (skratka CRAN).

Na internetu je umístěn na adrese https://cran.r-project.org.

K dispozici jsou předkompilované binární soubory pro běžné platformy Linux, Mac OS a Windows.

Ke stažení instalačního balíčku si můžeme zvolit vhodné zrcadlo.

Instalace balíčků R

Ku jádru R existuje bohatý soubor balíčků rozšiřujících jeho funkce.

Balíčky zvyšují výkon R.

Na instalaci balíčků používáme funkci `install.packages()`

První spuštění R

jestliže jsme nainstalovali R, můžeme ověřit jeho funkčnost.

Prostředí R spustíme jednoduše z příkazového řádku zadáním:

```
username@host:~$ R
```

Zobrazí se krátká úvodní poznámka, následovaná znakem

```
>
```

Tento symbol je znakem příkazového řádku prostředí R.

Opuštění prostředí R

Prostředí R je nyní připraveno k práci.

Pro ukončení práce v prostředí R jednoduše zadáme

```
> q()
```

R reaguje otázkou:

```
Save workspace image? [y/n/c]:
```

Zvolíme-li `y`, záznam celé historie provedených příkazů se uloží do souboru `.Rhistory`, jenž se zapisuje do pracovního adresáře.

Pracovní plocha a navigace

Všechny příkazy zadáváme interaktivně na příkazovém řádku.

V historii příkazů se pohybujeme pomocí kurzorových kláves, šipek směrem nahoru a dolů.

Tak se lze vrátit ke starším příkazům bez nutnosti jejich opětovného zápisu. Jen si vybereme požadovaný příkaz a opětovně ho odešleme klávesou Enter.

Uložíme-li si při opuštění prostředí historii, lze se vrátit i k příkazům z předchozí relace.

Komunikace s OS

Předvolený pracovní adresář je adresář, v němž byl program R spuštěn. V tomto aktuálním pracovním adresáři R čte a ukládá soubory a výsledky. Aktuální pracovní adresář ověříme pomocí funkce `getwd()`.

Aktuální pracovní adresář lze měnit použitím funkce `setwd()`.

Ke spuštění příkazů operačního systému užijeme funkce `system()`.

Nový adresář vytvoříme příkazem

```
> system("mkdir new")
```

Získání nápovědy

Funkce pro získání nápovědy má obecně jednoduchý tvar `help()` nebo zkráceně pomocí operátoru `?`.

Chceme-li získat informace o rozšiřujících balíčcích, použijeme

```
> help(package="jméno balíčku")
```

Některé balíčky obsahují rovněž ukázky kódu, které spustíme pomocí funkce `demo()`, například

```
> demo(package="stats")
```

R jako kalkulátor

Konzola příkazového řádku umožňuje interaktívní výpočet výsledků operací a funkcí

```
> 5+3
[1] 8
```

Nevidíme-li úvodní znak příkazového řádku, může to být způsobeno tím, že jsme nezadali úplný příkaz

```
> 5-
+
```

Musíme doplnit zbytek příkazu a poté stlačit Enter nebo zrušit příkaz stlačením klávesy Esc.

Objekty

R je objektově orientovaný jazyk

V R je všechno objektem a představuje nějaké údaje, které byly uloženy v paměti

Objekty mohou mít libovolné názvy, je však nutno dodržovat tato pravidla:

- název se skládá pouze z malých nebo velkých písmen, číslic, podtržítek a teček,
- název začíná velkým nebo malým písmenem,
- R rozlišuje velikost písmen (to znamená, že A a a jsou dva různé objekty),
- názvem nesmí být žádné z rezervovaných slov R (jejich seznam lze zobrazit zadáním `help(reserved)`),

Vytváření objektů

Nový objekt vytvoříme snadno pomocí operátoru přiřazení.

Operátor přiřazení má dvě možné podoby: <- nebo =.

Doporučuje se používat <-, protože = může někdy způsobovat chyby:

```
> log(x=25,base=5)
[1] 2
> x
Error: object 'x' not found
> log(x<-25,base=5)
[1] 2
> x
[1] 25
```

Seznam a odstraňování objektů

Seznam všech existujících objektů získáme jako výstup funkce `ls()`.

Objekty, jež nebudeme v budoucnu používat lze odstranit z paměti pomocí funkce `rm()`.

Statistika a programování v R

II. Datové struktury v R

Data tyou `nunmeric`

Data typu `numeric` představují reálná desetinná čísla .

Toto je implicitní typ každého nového objektu.

Vznikne, pokud do libovolné nové proměnné přiřadíme reálné číslo.

Typ libovolného objektu si ověříme pomocí funkce `class()`.

Data `numeric` – příklad

Podívejme se na příklad.

```
1 > x<-12.35
2 > class(x)
3 [1] numeric
```

Poznámka

Číslo je reprezentováno jako vektor délky 1. Znak `[1]` na začátku řádku označuje první pozici v tomto vektoru.

Datový typ `numeric`

Vložením celého čísla do proměnné se nezmění její typ, ale zůstane `numeric`.

Viz příklad

```
1 > z<-100
2 > class(z)
3 [1] "numeric"
```

Na typ se rovněž můžeme dotázat pomocí funkce `is.integer()`.

```
1 > is.integer(z)
2 [1] FALSE
```

Data typu integer

Pro vytvoření objektu typu `integer` použijeme funkci `as.integer()`

Příklad

```
> a <- as.integer(12)
> a
[1] 12
> class(a)
[1] "integer"
> is.integer(a)
[1] TRUE
```

Data typu integer

Alternativně lze proměnné typu `integer` zadávat jako celá čísla zakončená písmenem `L`

Příklad

```
1 > n<-as.integer(10)
2 > class(n)
3 [1] "integer"
4 > n<-10L
5 > class(n)
6 [1] "integer"
```

Typ dat integer

Co se stane, když vložíme hodnotu, která není celé číslo?

```
1 > as.integer(2.718)
2 [1] 2
3 > as.integer(TRUE)
4 [1] 1
```

Hodnota se zaokrouhlí nebo převede na celé číslo.

Typ dat integer

Výjimkou jsou však znaky nebo řetězce

Tyto nejsou transformovány

```
> as.integer("frcka")
[1] NA
Warning message:
NAs introduced by coercion
```

Typ dat complex

Prostředí R umožňuje pracovat i s komplexními čísly.

Komplexní hodnota je v R definována imaginární jednotkou `i`

Příklad

```
> z<-1+2i
> class(z)
[1] "complex"
```

Typ dat complex

Všimněme si, že hodnota -1 není typu complex, a tudíž

```
1 > sqrt(-1)
2 [1] NaN
3 Warning message:
4 In sqrt(-1) : NaNs produced
```

Musíme zadat

```
1 > sqrt(-1+0i)
2 [1] 0+1i
```

Typ dat `complex`

Znáte alternativní řešení?

Typ dat `complex`

Znáte alternativní řešení?

Použíjeme funkci `as.complex()`

Typ dat complex

```
1 > sqrt(as.complex(-1))
2 [1] 0+1i
```

Funkce `sqrt()` a `as.complex()` musí být zadány v daném pořadí.

```
1 > as.complex(sqrt(-1))
2 [1] NaN+0i
3 Warning message:
4 In sqrt(-1) : NaNs produced
```

Typ dat complex

Při zadávání komplexního čísla s jednotkovou imaginární částí je třeba zadat také koeficient.

V opačném případě je imaginární jednotka považována za objekt.

Podívejme se na příklad

```
1 > a<-1+i
2 Error: object 'i' not found
3 > a<-1+1i
4 > a
5 [1] 1+1i
```

Typ dat `logical`

Může nabývat dvou logických hodnot `TRUE` nebo `FALSE`

Často vzniká porovnáním proměnných.

```
1  > x<-10;y<-20
2  > z<-x<y
3  > z
4  [1] TRUE
5  > class(z)
6  [1] "logical"
```

Typ dat `logical`

Jsou pro ně definovány všechny standardní logické operace

`&`	Logické `AND`
`|`	Logické `OR`
`!`	Negace

Typ dat `logical`

Ilustrace

```
> a<-TRUE;b<-FALSE
> a&b
[1] FALSE
> a|b
[1] TRUE
> !a;!b
[1] FALSE
[1] TRUE
```

Typ dat character

Slouží k ukládání řetězců znaků, řetězce se zadávají pomocí uvozovek.

```
> x<-"facina"
> class(x)
[1] "character"
#Ale take
> x<-as.character(3.1415926)
> x
[1] "3.1415926"
> class(x)
[1] "character"
```

Typ dat character

Řetězce znaků lze spojit pomocí `paste()`

```
1  > name<-"Donald"
2  > surname<-"Knuth"
3  > paste(name,surname)
4  [1] "Donald␣Knuth"
5  # Chceme-li
6  > paste(name,surname,sep=",")
7  [1] "Donald,Knuth"
```

Typ dat character

Jak vytvořit spojení bez mezer?

Typ dat character

Jak vytvořit spojení bez mezer?

Oddělovač ve funkci `paste()` definujeme jako prázdný řetězec, tj. definujeme `sep=""`.

Typ dat character

Jak vytvořit spojení bez mezer?

Oddělovač ve funkci `paste()` definujeme jako prázdný řetězec, tj. definujeme `sep=""`.

```
paste(name,surname,sep="")
```

Typ dat character

Někdy je užitečné získat formátovaný výstup pomocí funkce `sprintf()`.

Jeho syntaxe je stejná jako v jazyce C

Formátovací značky

`s`	znakový řetězec, položky `NA` jsou převedeny na "NA".
`d,i`	číselné hodnoty.
`o`	integer v osmičkové notaci.
`x,X`	Celé číslo v hexadecimálním zápisu s použitím stejné velikosti pro `a-f` jako v kódu.
`f`	Hodnota s dvojnásobnou přesností v pevné řádové čárce. Počet desetinných míst je určen přesností, výchozí hodnota je 6,
`e,E`	D Hodnota s dvojnásobnou přesností, v exponenciální notaci, s použitím stejné velikosti pro e jako v kódu.

Typ dat character – formátovaný výstup

```
> sprintf("%s␣has␣%i␣dogs","John",3)
[1] "John␣has␣3␣dogs"
> sprintf("Number␣pi␣equals␣%f",pi)
[1] "Number␣pi␣equals␣3.141593"
> sprintf("Number␣pi␣equals␣%0.12f",pi)
[1] "Number␣pi␣equals␣3.141592653590"
> sprintf("10!␣in␣exponential␣%e",factorial(10))
[1] "10!␣in␣exponential␣3.628800e+06"
sprintf("100␣in␣octal␣notation␣%o",100)
[1] "100␣in␣octal␣notation␣144"
> sprintf("1000␣in␣hexadecimal␣notation␣%X",1000)
[1] "1000␣in␣hexadecimal␣notation␣3E8"
```

Typ dat character – funkce sub()

Pokud chceme nahradit část řetězce jiným podřetězcem, použijeme funkci `sub()`.

Je důležité dbát na jednoznačnost podřetězce, protože se nahrazuje pouze první výskyt.

Podívejme se na příklad

```
> z<-"Here␣is␣my␣brother␣and␣my␣sister"
> sub("my","your",z)
[1] "Here␣is␣your␣brother␣and␣my␣sister"
> sub("my␣sister","your␣sister",z)
[1] "Here␣is␣my␣brother␣and␣your␣sister"
```

Tato funkce se liší od `sub()` tím, že `gsub()` postupně nahrazuje všechny výskyty odpovídajícího podřetězce.

Podívejme se na změnu v předchozím příkladu.

```
1 > gsub("my","your",z)
2 [1] "Here␣is␣your␣brother␣and␣your␣sister"
```

Vektory

Vektor je nejjednodušší datová struktura.

Lze ji charakterizovat jako posloupnost prvků stejného datového typu.

Jednotlivé hodnoty obsažené ve vektoru se označují jako komponenty.

Počet složek vektoru se označuje jako jeho délka.

Vektory

Vektor `v` je vytvořen pomocí funkce `c()`.

Jeho délka se zjistí pomocí funkce `length()`

```
1 > v<-c(1,3,5,7,9)
2 > length(v)
3 [1] 5
```

Vektory

Vektor logických hodnot

```
1 > v<-c(TRUE,TRUE,FALSE,TRUE,FALSE)
2 >v
3 [1] TRUE TRUE FALSE TRUE FALSE
```

Vektor prvků typu character

```
1 > a<-c("aa","bb","cc","dd","ee","ff")
2 > a
3 [1] "aa" "bb" "cc" "dd" "ee" "ff"
```

Vektory

Vektory lze slučovat pomocí „combine“ `c()`

```
1  >a<-c(1,2,3)
2  >b<-c(4,5,6)
3  >c(b,a)
4  [1] 4 5 6 1 2 3
5  # Viz prepsani komponent
6  > a<-c("a", "b", "c")
7  > c(a,b)
8  [1] "a" "b" "c" "4" "5" "6"
```

Vektory – aritmetika

Vektorová aritmetika je implementována po jednotlivých komponentách.

Aritmetické operace jsou implementovány po složkách.

- `+` přičtení čísla ke všem komponentám nebo sčítání vektorů po komponentách
- `-` odečte číslo od všech složek nebo odečte vektory složku po složce,
- `*` násobení všech složek číslem nebo násobení vektorů po složkách,
- `/` dělení všech složek číslem nebo dělení vektorů po složkách.

Vektory – aritmetika

```
> v<-c(1,3,5,7,9)
> u<-c(10,20,30,40,50)
> u+v
[1] 11 23 35 47 59
> u-v
[1] 9 17 25 33 41
> 5*v
[1] 5 15 25 35 45
> u*v
[1] 10 60 150 280 450
> u/5
[1] 2 4 6 8 10
> u/v
[1] 10.000000 6.666667 6.000000 5.714286 5.555556
```

Vektory – aritmetika

Upozornění na recyklační pravidlo

Pokud se délky vektorů neshodují, použije kratší je použit cyklicky opakovaně.

Toto pravidlo je omezeno podmínkou, že délka delšího vektoru je celistvým násobkem kratšího. Pokud ne, operace se neprovede.

Vektory – aritmetika

```
> v<-c(10,20,30)
> u<-1:9
> u+v
[1] 11 22 33 14 25 36 17 28 39
# Cislo je vektor delky 1
> b<-c(1,2,3,4)
> 5*b
[1] 5 10 15 20
```

Vektory – výběr komponent

Složky, které chceme z vektoru vybrat, určíme pomocí indexů v závorkách []

```
1  > v<-1:10
2  > v[3:5]
3  [1] 3 4 5
```

Poznámka

Operátor : definuje rozsah čísel od prvního po druhé.

Vektory – výběr komponent

Komponenty můžeme vybírat také pomocí vektoru logických hodnot.

Délka obou vektorů by měla být stejná, jinak se předpokládá, že zbývající pozice jsou TRUE.

```
1  > u<-2*1:6
2  > L<-c(FALSE,TRUE,TRUE,FALSE,FALSE,TRUE)
3  > u[L]
4  [1]  4  6 12
```

Vektory – výběr komponent

Vybrané komponenty nemusí být v souvislé posloupnosti indexů.

Definujeme je pomocí funkce `c()`

```
1 > a<-c("aa","bb","cc","dd","ee","ff")
2 > a[c(2,3,5)]
3 [1] "bb" "cc" "ee"
4 # Opakovane indexy
5 > a[c(2,2,3,5)]
6 [1] "bb" "bb" "cc" "ee"
```

Vektory – přiřazení názvů komponentám

Komponentám můžeme přiřadit vhodné názvy.

Názvy definujeme pomocí funkce `names()`.

```
1  > v<-c("Donald","Knuth")
2  > names(v)<-c("Name","Surname")
3  > v
4      Name   Surname
5  "Donald"  "Knuth"
```

Vektory – přiřazení názvů komponentám

Po přiřazení názvů komponentám je můžeme vybrat pomocí těchto názvů.

V předchozím příkladu můžeme použít

```
1 > v["Surname"]
2 Surname
3 "Knuth"
```

Matrice

Matice je dvourozměrná tabulka dat stejného typu uspořádaná do obdélníkového schématu.

Vytvoříme ji pomocí funkce `matrix()` s následujícími argumenty

`vector` obsahuje prvky matice,

`nrow` je celočíselná hodnota, která udává počet řádků v matici,

`ncol` je celočíselná hodnota, která určuje počet sloupců v matici,

`byrow` je logická hodnota, která určuje, zda má být matice vyplněna po řádcích (`byrows=TRUE`) nebo po sloupcích (`byrows=FALSE`), její výchozí hodnota je `FALSE`,

`dimnames` je seznam vektorů typu `character`, které obsahují nepovinná označení řádků a sloupců.

Matrix – příklad zadání

vfill

```
> A<-matrix(3:8,nrow=3,ncol=2,byrow=TRUE)
> A
     [,1] [,2]
[1,]    3    4
[2,]    5    6
[3,]    7    8
> B<-matrix(3:8,nrow=3,ncol=2,byrow=FALSE)
> B
     [,1] [,2]
[1,]    3    6
[2,]    4    7
[3,]    5    8
```

Matice – přístup k prvkům

Jednotlivé prvky matice jsou přístupné pomocí dvojice indexů oddělených čárkami v závorkách.

```
1  > A[2,2]
2  [1] 6
```

Matice – přístup k prvkům

Jednotlivé prvky matice jsou přístupné pomocí dvojice indexů oddělených čárkami v závorkách.

```
> A[2,2]
[1] 6
```

Vynechání jednoho z indexů vede k extrakci řádku nebo sloupce.

```
> A[,1]
[1] 3 5 7
> B[2,]
[1] 4 7
```

Matice – výběr podmatice

Řádky a sloupce definujeme pomocí funkce `c()`.

```
> C<-matrix(1:12,nrow=3)
> C
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> C[c(1,3),c(2,4)]
     [,1] [,2]
[1,]    4   10
[2,]    6   12
```

Matice – přiřazení názvů

Řádkům a sloupcům přiřazujeme názvy pomocí funkcí `dimnames()` a `list()`.

```
> dimnames(A)<-list(c("row1","row2","row3"),
+ c("col1","col2"))
> A
     col1 col2
row1    3    4
row2    5    6
row3    7    8

> A["row2","col1"]
[1] 5
```

Matice – transpozice

Matrici můžeme transponovat pomocí funkce `t()`

```
1  > B<-matrix(3:8,nrow=3,ncol=2,byrow=FALSE)
2  > t(B)
3       [,1] [,2] [,3]
4  [1,] 3 4 5
5  [2,] 6 7 8
```

Ostatní funkce jsou definovány v balíčku `matlib`.

Matice – operace

Jsou definovány po jednotlivých složkách

Je to důležité při násobení matic. Běžná operace $*$ znamená násobení prvků na stejných pozicích.

Standardní násobení matic z lineární algebry je definováno jako operace %*%.

Matice – operace

Porovnejme

```
1  > C<-B[c(1,2),c(1,2)]
2  > C*C
3        [,1] [,2]
4  [1,] 9 36
5  [2,] 16 49
6  > C%*%C
7        [,1] [,2]
8  [1,] 33 60
9  [2,] 40 73
```

Matice – slučování

Aby bylo možné matice sloučit, musí mít stejný počet řádků nebo sloupců.

Pokud mají stejný počet řádků, můžeme sloučit sloupce pomocí funkce `cbind()`.

```
> cbind(B,diag(c(1,2,5))
     [,1] [,2] [,3] [,4] [,5]
[1,]  3  6  1  0  0
[2,]  4  7  0  2  0
[3,]  5  8  0  0  5
```

Poznámka

Všimněte si funkce `diag()`. Vytvoří diagonální matici se zadaným vektorem na diagonále.

Matice – slučování

Pokud mají matice stejný počet sloupců, můžeme je sloučit pomocí funkce `rbind()`.

```
> C<-matrix(1:12,nrow=3)
> rbind(C,diag(c(1,2,5,7))[c(2,4),])
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
[4,]    0    2    0    0
[5,]    0    0    0    7
```

Array

Pole `array` jsou zobecněním maticové datové struktury.

Ve skutečnosti se jedná o více než dvourozměrné matice.

Pole můžeme vytvořit pomocí funkce `array()`.

Syntaxe této funkce je následující

```
name<-array(vector, dimensions,dimnames)
```

Pole – vytváření

Ukažme si vytvoření pole o rozměrech $3 \times 4 \times 3$.

Pro lepší orientaci v poli nejprve vytvořme názvy jednotlivých dimenzí.

```
1 > dim1<-c("A1", "A2", "A3")
2 > dim2<-c("B1", "B2", "B3", "B4")
3 > dim3<-c("C1", "C2", "C3")
```

Pole – vytvoření, pokračování

Nyní vytvoříme pole z, jenž obsahuje celá čísla od 1 do 36 ($3 \times 4 \times 3$ je 36).

```
> z<-array(1:36,c(3,4,3),
dimnames=list(dim1,dim2,dim3))
```

Chcete-li zobrazit strukturu pole, zadejte v prostředí R příkaz z.

Výstup je příliš dlouhý, než aby se zobrazil v prezentaci.

Pole – přístup k prvkům

K prvkům pole se přistupuje pomocí hranatých závorek ve stejném režimu jako u matic.

```
> z[2,3,1]
[1] 8
> z[2:3,2:3,2]
    B2 B3
A2 17 20
A3 18 21
```

Struktura data frame

Data frame je nejběžnější struktura pro ukládání dat.

Umožňuje ukládat sloupcové vektory různých datových typů.

Data frramy se vytvářejí pomocí funkce `data.frame()`, jejíž obecná syntaxe je následující

```
1 > name<-data.frame(col1,col2,col3, ...)
```

Data frame – vytváření

Vytvoříme krátký datový rámec obsahující údaje o střelbě basketbalistů.

```
> playerID<-c(1,2,3,4)
> position<-c("forward","guard","forward","center")
> attempted<-c(12,6,10,15)
> made<-c(7,4,6,12)
> players<-data.frame(playerID,position,attempted,made)
> players
  playerID position attempted made
1        1  forward        12    7
2        2    guard         6    4
3        3  forward        10    6
4        4   center        15   12
```

Data frame – přístup k buňkám

Existuje více způsobů, jak přistupovat k jednotlivým buňkám datového rámce.

Používání indexů

```
1  > players[1:2]
2    position playerID
3  1 1 rorward
4  2 2 guard
5  3 3 forward
6  4 4 center
```

Data frame – přístup k buňkám

Další možností je použití názvů sloupců.

Názvy sloupců jsou zadány jako vektor typu `character`.

```
1  > players[c("playerID", "attempted", "made")]
2    playerID attempted made
3  1 1 12 7
4  2 2 6 4
5  3 3 10 6
6  4 4 15 12
```

Data frame – přístup k buňkám

Třetí možností je použít $ značení.

Skládá se z názvu data framu na první pozici a názvu sloupce na druhé pozici, oddělených znakem $.

```
1 > players$position
2 [1] forward guard   forward center
3 Levels: center forward guard
```

Data frame – operátor dvojitých závorek

Pro přístup k jednomu sloupci použijte dvojitou hranatou závorku `[[]]`

Porovnejte tyto dva výroky

```
> players[4]
   made
1  7
2  4
3  6
4  12
```

```
> players[[4]]
[1] 7 4 6 12
```

Data frame – operátor dvojitých závorek

Operátor dvojité závorky je ekvivalentní s použitím čárky v operátoru jednoduché závorky.

```
> players[,4]
[1] 7 4 6 12
> players[, "made"]
[1] 7 4 6 12
```

Data frame – přiřazení názvů řádkům

Používáme funkci `row.names()`, jejímž argumentem je vektor typu `character`.

```
> row.names(players)<-c("Player1","Player2","Player3","Player4")
> players
         playerID position attempted made
Player1         1  forward        12    7
Player2         2    guard         6    4
Player3         3  forward        10    6
Player4         4   center        15   12
```

Data frame – přiřazení názvů řádkům

Nyní můžeme extrahovat řádky buď podle indexů, nebo podle názvů.

```
> players[3,]
         playerID position attempted made
Player3        3  forward        10    6
> players["Player3",]
         playerID position attempted made
Player3        3  forward        10    6
```

Data frame – výběr řádků

Pokud chceme získat více než jeden řádek, použijeme celočíselný vektor.

```
> players[c(1,3),]
         playerID position attempted made
Player1         1  forward        12    7
Player3         3  forward        10    6
> players[2:4,]
         playerID position attempted made
Player2         2    guard         6    4
Player3         3  forward        10    6
Player4         4   center        15   12
```

Data frame – přiřazení názvů řádkům

Pokud musíme často opisovat název datového rámce, může to být nepohodlné.

Pomocí funkce `attach()` přidáme datový rámec do vyhledávací cesty.

To nám umožňuje psát pouze názvy sloupců.

Pokud chceme datový rámec z vyhledávací cesty odstranit, použijeme jednoduše funkci `detach()`.

Data frame – attach() ukázka

Po připojení datového rámce players můžeme snadno vypočítat procenta jednotlivých hráčů.

```
> attach(players)
The following objects are masked _by_ .GlobalEnv:

    attempted, made, playerID, position

> 100*made/attempted
[1] 58.33333 66.66667 60.00000 80.00000
```

Data frame – alternativa k `attach()`

Alternativou připojení data framu k vyhledávací cestě je použití funkce `with()`.

```
> with(players, {
+ 100*made/attempted}
+ )
[1] 58.33333 66.66667 60.00000 80.00000
```

Data frame – slučování

Často potřebujeme sloučit data ze dvou nebo více datových sad.

Používáme funkci `merge()`.

Argumenty jsou názvy dvou data framů, ježé se mají sloučit.

Třetí argument `by='''column_name'` definuje určující proměnnou pro sloučení dat.

Data frame – slučování dat

Pro demonstraci sloučení nejprve vytvoříme nový data frame `rebounds`.

```
1 > offensive<-c(5,2,3,10)
2 > defensive<-c(6,3,8,12)
3 > rebounds<-data.frame(playerID,defensive,offensive)
4 > row.names(rebounds)<-c("Player1","Player2","Player3",
5 + "Player4")
```

Data frame – slučování dat

Nyní jsme připraveni sloučit data framy `players` a `rebounds`.

```
> new_players<-merge(players,rebounds,by="playerID")
> new_players
  playerID position attempted made defensive offensive
1        1  forward        12    7         6         5
2        2    guard         6    4         3         2
3        3  forward        10    6         8         3
4        4   center        15   12        12        10
```

Data frame – sloučení dat

Alternativou je přidání řádků do existujícího datového rámce pomocí funkce `rbind()`.

Argumenty jsou názvy dvou datových rámců.

Pro ilustraci připravíme nový datový rámec `players2`

```
1 > position<-c("center","guard","forward")
2 > attempted<-c(14,8,12)
3 > made<-c(10,5,8)
4 > players2<-data.frame(playerID,made,attempted,position)
5 > row.names(players2)<-c("Player5","Player6","Player7")
```

Data frame – slučování dat

Nyní tyto datové rámce sloučíme.

```
> more_players<-rbind(players,players2)
> more_players
         playerID position attempted made
Player1         1  forward        12    7
Player2         2    guard         6    4
Player3         3  forward        10    6
Player4         4   center        15   12
Player5         5   center        14   10
Player6         6    guard         8    5
Player7         7  forward        12    8
```

Seznam - List

Seznamy představují nejsložitější datovou strukturu.

Představují uspořádané kolekce objektů.

K vytvoření seznamu použijeme funkci `list()`. Jeho syntaxe je jednoduchá:

```
\list(objekt1,objekt2,...)
```

Jeho argumenty jsou názvy existujících objektů

Seznamy

Volitelně pojmenujte objekty ve vytvořeném seznamu:

```
\list(name1=objekt1,name2=objekt2,...)
```

Seznamy

Z našich stávajících data frmů players a players2 vytvoříme seznam pojmenovaný NBA.

```
> NBA<-list(club="Bulls",city="Chicago",Players=players)
> NBA
$club
[1] "Bulls"

$city
[1] "Chicago"

$Players
         playerID position attempted made
Player1         1  forward        12    7
Player2         2    guard         6    4
Player3         3  forward        10    6
Player4         4   center        15   12
```

Seznamy

Nyní můžeme přidat další člen seznamu pomocí funkce concatenate `c()`

```
1 > NBA<-c(NBA,list(club="Celtics",city="Boston",Players=players2))
2 > NBA
```

Výstup je příliš dlouhý na to, abychom ho zde ukázali, prohlédněte si ho přímo v R.

Poznámka

Tato funkce spojí všechny argumenty do jedné vektorové struktury. V tomto případě to znamená, že druhý klub dostal v novém seznamu pozice 4 až 6, zatímco prvek s dvojitým indexem `[2,1]` v seznamu neexistuje.

Seznamy – přístup k prvkům

Je nutno rozlišovat mezi operátory jednoduchých a dvojitých závorek.

Vyzkoušejte následující příkazy (některé výstupy jsou příliš dlouhé na to, abychom je zde mohli zobrazit)

```
> NBA[3]
> NBA[[3]]
> NBA[3][2]
$<NA>
NULL
> NBA[[3]][2,]
         playerID position attempted made
Player2         2    guard         6    4
```

Seznamy – úprava prvků

Zápis ve dvojitých závorkách umožňuje přímou modifikaci prvků seznamu.

```
> NBA[[3]][2,]
         playerID position attempted made
Player2         2    guard         6    4
> NBA[[3]][2,3]<-c(7)
> NBA[[3]][2,]
         playerID position attempted made
Player2         2    guard         7    4
```

Vstup z klávesnice

Nejjednodušší metoda (ale také časově nejnáročnější pro rozsáhlé vzorky)

Pracujeme se ve dvou krocích

- Vytvořte prázdný datový rámec s názvy a typy proměnných, které chceme uložit.
- Otevření jednoduchého editoru dat pomocí funkce `edit()`, jejímž argumentem je název datového rámce, který chceme upravovat.

Vstup z klávesnice

Vytvoříme prázdný data frame s názvem `mydata` se čtyřmi proměnnými: `name`, která má typ `character`, a třemi číselnými proměnnými `age`, `height` a `weight`.

```
1 > mydata<-data.frame(name=character(0),age=numeric(0),
2 + height=numeric(0),weight=numeric(0))
3 > mydata<-edit(mydata)
```

Poznámka

Všimněte si, že přiřazení jako `numeric(0)` a `character(0)` vytvoří proměnnou daného typu, ale bez dat.

Vstupní data ze souboru `.csv`

Hodnoty oddělené čárkou, jeden z nejpoužívanějších datových formátů.

První řádek může, ale nemusí obsahovat názvy sloupců.

Příklad struktury souboru

```
Column1,Column2,Column3
A,10,0.11
B,20,0.22
C,30,0.33
```

Vstupní data ze souboru .csv

Předpokládáme, že data jsou uložena v souboru `mydata.csv`.

Data importujeme pomocí funkce `read.csv()`.

```
> mydata<-read.csv("mydata.csv")
> class(mydata)
[1] "data.frame"
> mydata
  Column1 Column2 Column3
1       A      10    0.11
2       B      20    0.22
3       C      30    0.33
```

Vstupní data ze souboru `.csv`

Nepovinné argumenty funkce `read.csv()`.

- `header` Logická hodnota, určuje, zda vstupní soubor obsahuje názvy proměnných jako první řádek, výchozí hodnota `TRUE`.
- `sep` určuje znak oddělující položky, výchozí hodnota je čárka,
- `dec` určuje znak použitý v souboru pro desetinná místa, výchozí hodnota je `.`, zmiňme také funkci `read.csv2()`, která používá čárku pro desetinná místa a středník jako oddělovač.
- `skip=n` určuje počet řádků, které se mají přeskočit před zahájením čtení dat. Tato možnost je užitečná pro datové tabulky s prázdnými řádky nebo textovými popisy na začátku souborů.
- `stringsAsFactors`, což je logická hodnota, která určuje, zda se řetězce převedou na faktory, pokud chcete převodu zabránit, nastavte ji na `FALSE`.
- `row.names` vektor s názvy řádků.

Zápis dat do souboru .csv

R umí vytvořit soubor csv z existujícího data framu.

Používáme funkci `write.csv()` nebo `write.csv2()`, která používá čárku jako desetinnou tečku a středník jako oddělovač.

Běžná syntaxe

```
write.csv(object,file="file_name",...options)
```

`object` je povinný argument obsahující název data framu, který chceme uložit, a `file_name` je název (nebo úplná cesta) souboru.

Zápis dat do souboru `.csv`

Vybrané možnosti funkce `write.csv()`

- `append`, což je logická hodnota, která udává, zda se výstup připojí na konec souboru. Výchozí hodnota je `FALSE` a všechny existující soubory s tímto názvem budou přepsány.
- `sep` definuje znak oddělovače položek. Hodnoty v každém řádku `object` jsou odděleny tímto znakem.
- `dec` řetězec, který se použije pro desetinné čárky v číselných nebo složených sloupcích, musí to být jeden znak. Výchozí hodnotou je desetinná tečka.
- `row.names` Logická hodnota určující, zda se mají zapsat názvy řádků `object`.

Vstupní data ze souborů Excel

Existuje několik balíčků, které umožňují importovat data přímo ze souborů aplikace Excel. Uveďme si některé z nich:

- `xlsx`,
- `XLconnect`
- `readxl`

Excel 2007 a novější verze používají formát `xlsx`, proto zde uvedeme balí4ek `xlsx`.

Vstupní data ze souborů Excel

Balíček nainstalujeme obvyklým příkazem:

```
install.packages("xlsx")
```

Pokud jej chceme použít v aktuálním pracovním prostoru, načteme jej standardním způsobem:

```
library("xlsx")
```

Vstupní data ze souborů Excel

Tento balíček poskytuje dvě funkce pro načtení obsahu pracovního sešitu Excelu do R data.frame: `read.xlsx()` a `read.xlsx2()`.

Rozdíl mezi těmito dvěma funkcemi je následující:

- `read.xlsx()` zachovává datový typ, typ proměnné odpovídá každému sloupci v pracovním listu, ale je pomalý pro velké datové soubory (pracovní list s více než 100 000 buňkami). `read.xlsx2()` je rychlejší pro velké soubory.

Vstupní data ze souborů Excel

Obě funkce mají podobnou syntaxi:

```
read.xlsx(file, sheetIndex, header=TRUE, colClasses=NA)
read.xlsx2(file, sheetIndex, header=TRUE, colClasses="character")
```

Jejich argumenty mají následující význam:

- `file` je název souboru, který obsahuje tabulku. Pokud se soubor nenachází v pracovním adresáři, je třeba zadat úplnou cestu.
- `sheetIndex` je číslo udávající index listu, který se má načíst. Lze jej nahradit argumentem `jméno listu`, zadaným jako řetězec znaků s názvem listu.
- `header`logická hodnota. Pokud `header=TRUE`, použije se první řádek jako konvence pro pojmenování proměnných.
- `colClasses` znakový vektor reprezentující třídu každého sloupce.
- `startRow`, `endRow` čísla určující index počátečního a posledního načítaného řádku.

Zápis dat do souborů Excel

Balíček `xlsx` poskytuje dvě funkce zápisu `write.xlsx()` a `write.xlsx2()`.

Obecná syntaxe

```
write.xlsx(x, file, sheetName="Sheet1", col.names=TRUE,
row.names=TRUE, append=FALSE)

write.xlsx2(x, file, sheetName="Sheet1",col.names=TRUE,
row.names=TRUE, append=FALSE)
```

Zápis dat do souborů Excel

Jejich argumenty mají následující význam:

- `x` data frame, který se zapíše do sešitu.
- `souborjméno` (nebo cesta) výstupního souboru.
- `sheetName` řetězec znaků s názvem listu.
- `col.names` Logická hodnota, určuje, zda se mají do souboru zapsat názvy sloupců `x`.
- `row.names` Logická hodnota, určuje, zda se do souboru zapíší názvy řádků `x`.
- `append` logická hodnota, určuje, zda má být `x` připojen k existujícímu souboru, pokud `FALSE`, přepíše existující soubor stejnou cestou.

Čtení dat ze souborů JSON

JSON (JavaScript Object Notation) je jednoduchý formát pro výměnu dat.

Abychom mohli načítat soubory JSON do R, musíme nejprve nainstalovat nebo načíst balíček `rjson`.

Můžeme použít funkci `fromJSON()`

Použití závisí na umístění souboru `.json`.

```
data<-fromJSON(file="jméno souboru.json")
data<-fromJSON(file="URL na soubor json")
```

V obou případech je objekt `data` uložen jako seznam. Pro další analýzu můžeme data převést pomocí funkce `as.data.frame()`.

Zápis dat do souborů JSONI

Musí být provedeno ve dvou krocích.

V prvním kroku musíme připravit objekt JSON a ve druhém kroku jej zapíšeme do souboru.

Pro vytvoření objektu JSON použijeme funkci `toJSON()`:

```
dataJSON<toJSON(data)
```

Pak použijeme funkci `write()`.

```
write(dataJSON, "filename.json")
```

Statistika a programování v R

III. Rozdělení pravděpodobností v R

Náhodný výběr

Standardní funkce `sample()` se syntaxí

```
sample(x, size, replace, prob)
```

Argumenty

- `x` je vektor nebo soubor dat, ze kterého je vzorek vybrán,
- `size` je velikost vzorku,
- `replace` je logická hodnota, která určuje, zda se hodnoty ve vzorku opakují, nebo ne,
- `prob` je vektor pravděpodobností položek.

Náhodný vzorek – příklady

Nejjednodušší je použít pouze první argument

```
> sample(6)
[1] 4 3 5 1 6 2
> sample(4:10)
[1] 9 7 5 4 8 10 6
> sample(c(1,3,5,7,9))
[1] 9 5 7 3 1
```

Náhodný vzorek – příklady

Druhý argument určuje velikost vzorku

5 náhodně vybraných celých čísel od 1 do 40

```
1  > sample(1:40,5)
2  [1] 30 35 34 5 29
```

Náhodný vzorek – příklady

Simulace padesátinásobného hodu kostkou

```
1 > sample(6,50)
2 Error in sample.int(x, size, replace, prob) :
3  cannot take a sample larger than the population when 'replace = FALSE'
```

Chyba, protože velikost vzorku přesahuje délku vektoru dat, z něhož se má vzorek odebrat.

Náhodný vzorek – příklady

Simulujeme padesátinásobný hod kostkou - pro opakování hodnot musíme nastavit argument `replace`.

```
> vzorek (6 , 50 , replace = TRUE )
 [1] 6 5 3 3 5 5 4 6 3 1 3 2 ...
[39] 2 6 6 6 4 2 2 5 1 6 1 5
```

Náhodný vzorek – příklady

Můžeme rovněž simulovat házení falešnou mincí s vyšší četností hlav.

Předpokládejme, že hlava padá dvakrát častěji než znak, a nastavíme argument `prob=c(2/3,1/3)`.

```
> sample (c("head","tail"),20,replace = TRUE,prob=c(2/3,1/3))
 [1] "head" "tail" "head" "head" "head" "head" "tail" "head" "head" "tail"
[11] "head" "tail" "head" "head" "tail" "head" "head" "tail" "tail" "head"
```

Náhodné vzorky – zajištění stejného výsledku

Pokud vezmeme vzorky, budou náhodné a budou se měnit vždy, když použijeme funkci `vzorek()`.

Pokud potřebujeme rekonstruovat stejný vzorek, můžeme použít funkci `set.seed()`

```
1 > set.seed(3)
2 > sample(6)
3 [1] 2 5 6 1 4 3
4 > set.seed(3)
5 > sample(6)
6 [1] 2 5 6 1 4 3
```

Diskrétní rozdělení

Pravděpodobnosti jsou určeny seznamem pravděpodobností diskrétních výsledků, známým jako pravděpodobnostní funkce.

Označíme-li množinu všech možných hodnot diskrétní náhodné veličiny X jako H , můžeme zavést pravděpodobnostní funkci $p(x)$ podle vzorce

$$p(x) = \mathbb{P}(X = x), \; x \in H. \tag{1}$$

Diskrétní rozdělení

Uvedeme některá z nich:

- Bernoulliho rozdělení,
- binomické rozdělení,
- geometrické rozdělení,
- hypergeometrické rozdělení,
- negativní binomické rozdělení,
- Poissonovo rozdělení.

Bernoulliho rozdělení

Máme k dispozici čtyři funkce:

- `rbern(n,prob)`, kde `n` je počet pozorování a `prob` je pravděpodobnost náhodné události A (úspěch v experimentu). Generuje vektor `0` a `1` vybraný z Bernoulliho rozdělení s danou pravděpodobností.
- `pbern(q, prob, lower.tail = TRUE, log.p = FALSE)`
- `dbern(x, prob, log = FALSE)`
- `qbern(p, prob, lower.tail = TRUE, log.p = FALSE)`

Binomické rozdělení

Pro práci s binomickým rozdělením jsou implementovány 4 funkce:

- `rbinom(n,prob)`, kde `n` je počet pozorování, `p` je pravděpodobnost úspěchu. Tato funkce generuje n náhodných veličin s danou pravděpodobností. `pbinom(x, n, k)`, kde `n` je celkový počet pokusů, `p` je pravděpodobnost úspěchu, `x` je hodnota, pro niž má být pravděpodobnost určena. `dbinom(x, n, p)`, kde `n` je celkový počet pokusů, `p` je pravděpodobnost úspěchu, `x` je hodnota, pro kterou má být pravděpodobnost určena. `qbinom(prob, n, p)`, kde `prob` je pravděpodobnost, `n` je celkový počet pokusů a `p` je pravděpodobnost úspěchu v jednom pokusu. Tato funkce slouží k určení n-tého kvantilu, tj. určí k takové, že $P(X \leq k)$.

Binomické rozdělení – příklady

Príklad.
Předpokládejme, že v testu je dvacet otázek s výběrem odpovědi. Každá otázka má pět možných odpovědí a pouze jedna z nich je správná. Určete pravděpodobnost nejvýše šesti správných odpovědí, pokud se student pokusí odpovědět na každou otázku náhodně.

Binomické rozdělení – řešení 1

Pravděpodobnost správné náhodné odpovědi na otázku je $\frac{1}{5} = 0,2$.

Pravděpodobnost přesně 6 správných odpovědí lze určit pomocí funkce `dbinom()`

```
1 > dbinom(6,20,0.2)
2 [1] 0.1090997
```

Binomické rozdělení – řešení 1

Pomocí funkce `dbinom()` s $x = 0, \ldots .6$ zjistíme pravděpodobnost šesti nebo méně správných odpovědí v náhodných experimentech a výsledky sečteme.

Dostáváme tedy:

```
1  > dbinom(0,20,0.2) + dbinom(1,20,0.2) + dbinom(2,20,0.2)+
2  + dbinom(3,20,0.2) + dbinom(4,20,0.2) + dbinom(5,20,0.2)+
3  + dbinomial(6,20,0.2)
4  [1] 0.9133075
```

Binomické rozdělení – řešení 2

Alternativně můžeme použít distribuční funkci pro binomické rozdělení `pbinom()`.

Dostaneme stejnou hodnotu

```
1 > pbinom(6,20,0.2)
2 [1] 0.9133075
```

Binomické rozdělení – pokračování příkladu

Príklad.
Student složí zkoušku úspěšně, pokud v testu správně odpoví na více než 10 otázek. Jaká je pravděpodobnost, že student složí zkoušku, pokud odpovídá na otázky náhodně?

Binomické rozdělení – řešení

Protože hledáme pravděpodobnost $\mathbb{P}(X > 10)$. Použijeme v tomto případě funkci `pbinom()`, avšak s volbou `lower.tail=FALSE`.

Dostaneme tak

```
1 > pbinom(10,20,0.2,lower.tail=FALSE)
2 [1] 0.0005634137
```

Binomické rozdělení – příklad 2

Príklad.
Předpokládejme, že jsme v továrně zodpovědní za kvalitu. Denně vyrobíme 250 zařízení. Vadné zařízení musí být opraveno. Víme, že míra závad je 2%. Nasimulujme si, kolik zařízení musíme každý den v tomto týdnu opravit.

Binomické rozdělení – řešení

Pomocí funkce `rbinom()` vygenerujeme náhodný vzorek z binomického rozdělení s počtem pokusů $n = 250$ a pravděpodobností úspěchu $p = 0,02$.

Takže máme

```
1 > rbinom(7,250,0.02)
2 [1] 2 5 3 9 5 9 5
```

Binomické rozdělení – příklad 3

Príklad.
Předpokládejme, že testujeme lék, který má 80% úspěšnost. Každá studie má 30 pacientů. Kolik pacientů se nachází v dolním 10% procentní skupině pozitivních výsledků? Uveďme jednotlivé decily v tomto testu léčby.

Binomické rozdělení – řešení

10% úspěšných studií bude mít 0 až 21 pacientů s pozitivní reakcí na tuto léčbu. To určíme pomocí funkce `qbinom()`:

```
1  > qbinom(0,1,30,0,8)
2  [1] 21
```

Pro získání každého decilu v tomto léčebném testu zadáme

```
1  > qbinom(seq(0.1,1,0.1),30,0.8)
2   [1] 21 22 23 24 24 25 25 26 27 30
```

Hypergeometrické rozdělení

Čtyři funkce pro práci s hypergeometrickým rozdělením v R:

- `rhyper(N, m, n, k)`, obecně označuje funkci generování náhodných čísel při zadaných parametrech a velikosti vzorku,
- `phyper(x, m, n, k)` definuje hypergeometrickou distribuční funkci,
- `dhyper(x, m, n, k)` definuje pravděpodobnostní funkci hypergeometrického rozdělení,
- `qhyper(N, m, n, k)` je kvantilová funkce hypergeometrického rozdělení, která se používá k určení posloupnosti pravděpodobností mezi 0 a 1.

Zde `x` představuje soubor hodnot, `m` velikost populace, `n` počet vzorků, `k` počet položek v populaci a `N` hypergeometricky rozdělené hodnoty.

Hypergeometrické rozdělení – příklad 1

Príklad.
Bude zvolen pětičlenný výbor ze skupiny čítající 10 žen a 8 mužů. Jaká je pravděpodobnost, že výbor bude složen ze 3 žen a 2 mužů? Jaká je pravděpodobnost, že ve výboru bude většina žen?

Hypergeometrické rozdělení – řešení

Podle požadavku je $x = 3$ žen ve výboru, $m = 10$ celkový počet žen ve skupině, $n = 8$ celkový počet mužů ve skupině a $k = 5$ počet členů výboru.

Proto máme

```
1 > dhyper(3,10,8,5)
2 [1] 0.3921569
```

Hypergeometrické rozdělení – řešení

Ženy mohou mít ve výboru většinu, pokud je v něm 5, 4 nebo 3 ženy, případně pokud jsou v něm nejvýše 2 muži.

Můžeme použít součet hodnot funkce `dhyper()`:

```
1 > dhyper(5,10,8,5)+dhyper(4,10,8,5)+dhyper(3,10,8,5)
2 [1] 0.6176471
```

Alternativně můžeme tuto pravděpodobnost vypočítat pomocí funkce `phyper()`, kde $x = 2$ muži v komisi, $m = 8$ celkový počet mužů ve skupině, $n = 10$ celkový počet žen ve skupině a $k = 5$ počet členů komise.

```
1 > phyper(2,8,10,5)
2 [1] 0.6176471
```

Hypergeometrické rozdělení – příklad 2

Príklad.
Předpokládejme, že v zásilce 100 DVD přehrávačů je deset vadných přehrávačů. Inspektor náhodně vybere 15 kusů ke kontrole. Simulujme, kolik vadných hráčů bude vybráno v posloupnosti 10 kontrol.

Hypergeometrické rozdělení – řešení

Zásilka obsahuje $m = 10$ vadných DVD přehrávačů a $n = 90$ vadných DVD přehrávačů a inspektor náhodně vybere $k = 15$, kontrola se opakuje $N = 10$ krát.

K simulaci jejich výsledků používáme funkci `rhyper()`.

Dostáváme tedy:

```
1  > rhyper(10,10,90,15)
2   [1] 4 1 1 0 2 0 1 2 3 2
```

Negativní binomické rozdělení

Čtyři funkce pro práci s negativním binomickým rozdělením v R:

- `rnbinom(N,n,prob)`, kde `n` je počet pokusů, `N` je velikost vzorku, `prob` je pravděpodobnost úspěchu. Tato funkce generuje N náhodných veličin s danou pravděpodobností.
- `pnbinom(x, n, p)`, slouží k výpočtu hodnoty distribuční funkce záporného binomického rozdělení. Zde `x` je počet selhání před `n`tým úspěchem a `p` je pravděpodobnost úspěchu.
- `dnbinom(x, n, p)` je pravděpodobnost `x` selhání před `n`tým úspěchem (všimněte si rozdílu), když pravděpodobnost úspěchu je `p`.
- `qnbinom(x, n, p)` se používá k výpočtu hodnoty kvantilové funkce negativního binomického rozdělení. Zde `x` je vektor požadovaných úrovní kvantilů, `n` je celkový počet pokusů a `p` je pravděpodobnost úspěchu na pokus.

Negativní binomické rozdělení – příklady

Príklad.
Ropná společnost provádí geologickou studii, která ukazuje, že průzkumný ropný vrt by měl mít 20% šanci na nalezení ropy. Jaká je pravděpodobnost, že první objev bude ve třetím vrtu? Jaká je pravděpodobnost, že třetí úspěšný vrt bude při sedmém vrtání?

Negativní binomické rozdělení – řešení

Potřebujeme určit $\mathbb{P}(X = 2)$ s $n = 14$.

Všimněte si, že se technicky jedná o geometrickou náhodnou veličinu, protože hledáme pouze jeden úspěch.

s ohledem na implementaci `dnbinom()` zadáme `x=2` selhání před `n=1` úspěch a `p=0,2`.

Takže máme

```
1   > dnbinom(2,1,0.2)
2   [1] 0.128
```

U druhé otázky jsme zvolili `x=4` neúspěšných pokusů před `n=3` úspěšnými pokusy.

```
1   > dnbinom(4,3,0.2)
2   [1] 0.049152
```

Poissonovo rozdělení

Čtyři funkce pro práci s Poissonovým rozdělením v R:

- `dpois(x,l)` vypočítá hodnotu pravděpodobnostní funkce $\mathbb{P}(X = x)$ Poissonova rozdělení s parametrem λ implementovaným jako argument `l`.
- `ppois(x,l)` počítá distribuční funkci náhodné veličiny, která se řídí Poissonovým rozdělením. Určuje pravděpodobnost $\mathbb{P}(X \leq x)$, argument `l` je parametr rozdělení. Při zadání dalšího argumentu `lower.tail=FALSE` dostaneme pravděpodobnost $\mathbb{P}(X > x)$.
- `rpois(k,l)` slouží ke generování náhodných čísel z daného Poissonova rozdělení, `k` je počet potřebných náhodných čísel a `l` je parametr rozdělení.
- `qpois(q,l)` se používá pro generování kvantilů daného Poissonova rozdělení, `q` je vektor potřebných kvantilových úrovní a `l` je parametr rozdělení.

Poissonovo rozdělení – příklady

Príklad.
Na určité řece dochází k povodním v průměru jednou za 100 let. Vypočítejte pravděpodobnost $k = 0, 1, 2, 3, 4, 5,$ nebo 6 v intervalu 100 let.

Poissonovo rozdělení – řešení

K povodni dochází jednou za 100 let, takže ji můžeme považovat za vzácnou událost a počet povodní se řídí Poissonovým rozdělením.

Používáme funkci `ppois()` pro `x` , což je vektor celých čísel od 0 do 6, a parametr `l` rovný 1 povodni každých 100 let.

Máme

```
1  > x<-seq(0:6)
2  > dpois(x,1)
3  [1] 3.678794e-01 1.839397e-01 6.131324e-02 1.532831e-02
4  + 3.065662e-03
5  [6] 5.109437e-04 7.299195e-05
```

Poissonovo rozdělení – příklad 2

Príklad.
Prodejce životního pojištění prodá v průměru 3 pojistky životního pojištění týdně. Vypočítejme pravděpodobnost, že v daném týdnu uzavře nějaké pojistky.

Poissonovo rozdělení – řešení

„Několik pojistek“ znamená „1 nebo více pojistek“

Potřebujeme vypočítat pravděpodobnost $\mathbb{P}(X > 0) = 1 - \mathbb{P}(X \leq 0)$.

Parametr rozdělení je `l=3`

Použijeme funkci `ppois()` s volitelným argumentem `lower.tail` nastaveným na `FALSE`.

```
1 > ppois(0,3,lower.tail=FALSE)
2 [1] 0.9502129
```

Alternativně můžeme použít `dpois()`:

```
1 > 1-dpois(0,3)
2 [1] 0.9502129
```

Poissonovo rozdělení – příklad 3

Príklad.
Společnost vyrábí 300 elektromotorů denně. Pravděpodobnost, že je elektromotor vadný, je 0,01. Simulujme počet vadných motorů vyrobených každý den během pracovního týdne.

Poissonovo rozdělení – řešení

Průměrný počet chyb při denní výrobě 300 motorů je $\lambda = 0,01 \times 300 = 3$.

Pro generování denního počtu chyb používáme funkci `rpois()` s argumenty `k=5` pracovních dnů a `l=3`.

Dostaneme tedy

```
1 > rpois(5,3)
2 [1] 3 3 4 2 2
```

Poissonovo rozdělení – příklad 4

Príklad.
Uvažujme počítačový systém s Poissonovým proudem příchozích úloh. s průměrnou rychlostí 2 požadavky za minutu. Jaký je maximální počet úloh, které by měly přijít za jednu minutu se spolehlivostí 90% ?

Poissonovo rozdělení – řešení

Nalezení maxima příchodů se spolehlivostí alespoň 90% znamená nalezení 90% kvantilu.

Použíjeme funkci `qpois()` s argumenty `q=0,9` a `l=2` průměrné požadavky za minutu.

Dostaneme tedy

```
1 > qpois(0,9,2)
2 [1] 4
```

Spojitá rozdělení

Uveďme si některé z nich:

- rovnonoměrné rozdělení,
- exponenciální rozdělení,
- normální rozdělení,
- Studentovo rozdělení t,
- Chí-kvadrát rozdělení,
- Fisherovo rozdělení F.

V R je implementováno mnoho dalších rozdělení.

Spojitá rozdělení

Uveďme si některé z nich:

- rovnonoměrné rozdělení,
- exponenciální rozdělení,
- normální rozdělení,
- Studentovo rozdělení t,
- Chí-kvadrát rozdělení,
- Fisherovo rozdělení F.

V R je implementováno mnoho dalších rozdělení.

Budeme se zabývat, třemi „modrými“ rozděleními.

Rovnoměrné rozdělení

Čtyři funkce pro práci s rovnoměrným rozdělením v R:

- `dunif()`, která definuje funkci hustoty, jejímiž argumenty jsou vektor `x` a parametry `min` a `max` rozdělení,
- `punif()`, která definuje distribuční funkci, jejími argumenty jsou vektor `x` a parametry `min` a `max` distribuce,
- `qunif()`, která poskytuje kvantilovou funkci, jejímiž argumenty jsou kvantily `q` a parametry `min` a `max` rozdělení,
- `runif()`, který generuje náhodné hodnoty proměnné, jeho argumenty jsou velikost vzorku `n` a parametry `min` a `max` rozdělení.

Rovnoměrné rozdělení – příklad

Príklad.
Předpokládejme, že tramvaje odjíždějí ze zastávky v pravidelných pětiminutových intervalech. Vypočítáme, jaká je pravděpodobnost, že cestující bude čekat:

a) více než 3 minuty,
b) nejvýše než 1,5 minuty,

pokud přijede na zastávku v náhodný okamžik.

Rovnoměrné rozdělení – řešení a)

Čekací doba je náhodná veličina, která se řídí rovnoměrným rozdělením s parametry $a = 0$ a $b = 5$.

Pravděpodobnost, že cestující bude čekat déle než 3 minuty, je tedy $\mathbb{P}(X > 3) = 1 - F(3)$.

Dostaneme

```
1 > 1-punif(3,min=0,max=5)
2 [1] 0.4
```

Rovnoměrné rozdělení – řešení b)

Otázka (b) se týká pravděpodobnosti $\mathbb{P}(X \leq 1,5) = F(1,5)$.

Požadovaný výsledek dostaneme jako `punif(1,5,min=0,max=5)`, takže pravděpodobnost je $0,3$.

```
1 > punif(1.5,min=0,max=5)
2 [1] 0.3
```

Rovnoměrné rozdělení – simulace

Situaci můžeme simulovat pomocí funkce `runif()`.

Zvětšením velikosti vzorku můžeme také ilustrovat, jak zvýšení počtu náhodných experimentů vede k lepší aproximaci rozdělení.

Chcete-li zobrazit graf, spusťte kód

```
1 par(mfrow = c(3, 1))
2 hist(runif(10,min=0,max=5))
3 hist(runif(100,min=0,max=5))
4 hist(runif(1000,min=0,max=5))
```

Exponenciální rozdělení

Čtyři funkce pro práci s exponenciálním rozdělením v jazyce R:

- `dexp()`, která představuje funkci hustoty, jejímiž argumenty jsou vektor `x` a parametr `rate` rozdělení,
- `pexp()`, což je distribuční funkce, jejímiž argumenty jsou vektor `x` a parametr `rate` distribuce,
- `qexp()`, která určuje kvantilovou funkci, jejími argumenty jsou kvantily `q` a parametr `rate` rozdělení,
- `rexp()`, která generuje náhodné hodnoty proměnné, jejími argumenty jsou velikost vzorku `n` a parametr `rate` rozdělení.

Exponenciální rozdělení – příklad

Príklad.
Předpokládejme, že průměrná doba obsluhy u pokladny v supermarketu je tři minuty. Zjistěte pravděpodobnost, že pokladní dokončí obsluhu zákazníka za:

a) méně než dvě minuty,
b) více než pět minut.

Exponenciální rozdělení – řešení

Průměrná doba obsluhy u pokladny se rovná převrácené hodnotě četnostiobsluhy,

Četnost obsloužených zákazníků je tedy {*frac*13 zákazníka za minutu. Odpovědí na otázku (a) je tedy pravděpodobnost $\mathbb{P}(X < 2)$.

```
1 > pexp(1/3,2)
2 [1] 0.4865829
```

Odpověď na otázku (b) je pravděpodobnost $\mathbb{P}(X > 5)$.

```
1 > pexp(1/3,2)
2 [1] 0.4865829
```

Exponenciální rozdělení – příklad 2

Príklad.
Je známo, že poruchy určitého typu elektronického zařízení mají exponenciální rozdělení se střední dobou 30 měsíců, než se zařízení porouchá. Zjistěme pravděpodobnost, že.

a) náhodně vybrané zařízení se porouchá během prvního roku (12 měsíců),
b) náhodně vybrané zařízení vydrží déle než 6 let (72 měsíců).

Exponenciální rozdělení – řešení a)

Náhodnou veličinu, která představuje dobu do poruchy zařízení, označujeme X.

Potřebujeme odpovědět na otázku, jaká je pravděpodobnost $\mathbb{P}(X < 12)$, jestliže náhodná veličina X má exponenciální rozdělení s parametrem $\lambda = 1/30$.

Výsledek získáme příkazem:

```
1 > pexp(1/30,12)
2 [1] 0.32968
```

Exponenciální rozdělení – řešení b)

Abychom mohli odpovědět na otázku (b), musíme zjistit pravděpodobnost $\mathbb{P}(X \geq 70)$.

Abychom získali odpověď pomocí funkce `pexp()`, musíme nastavit argument `lower.tail=FALSE`.

Dostaneme

```
1 > pexp(1/30,72,lower.tail=FALSE)
2 [1] 0.09071795
```

Exponenciální rozdělení – kvantily

Pro ilustraci významu kvantilů zjistíme dobu, za kterou bude 60% zařízení nefunkčních.

Použijeme funkci `qexp()`, jak ukazuje následující příkaz:

```
1  > qexp(0,6,1/30)
2  [1] 27.48872
```

60% zařízení se tedy porouchá přibližně za 27,5 měsíce.

Normální rozdělení

Čtyři funkce pro práci s normálním rozdělením v R:

- `dnormf()`, což je funkce hustoty, jejímiž argumenty jsou vektor `x` a parametry `mean` a `sd` rozdělení,
- `pnorm()`, která představuje distribuční funkci, jejími argumenty jsou vektor `x` a parametry `mean` a `sd` rozdělení
- `qnorm()`, která představuje kvantilovou funkci, jejímiž argumenty jsou kvantily `q` a parametry `mean` a `sd` rozdělení,
- `rnorm()`, který generuje náhodné hodnoty proměnné, jeho argumenty jsou velikost vzorku `n` a parametry `mean` a `sd` rozdělení.

Normální rozdělení – příklad

Príklad.
Předpokládejme, že výsledky přijímacích zkoušek na vysokou školu mají normální rozdělení. Průměrné skóre tohoto testu je 70 bodů a směrodatná odchylka je 10 bodů. Jaké je procento studentů

a) , kteří u zkoušky dosáhli alespoň 85 bodů,
b) , kteří ve zkoušce získali nejvýše 60 bodů.

Normální rozdělení – řešení a)

Použijeme funkci `pnorm()` normálního rozdělení se střední hodnotou 70 a směrodatnou odchylkou 10.
Zajímá nás $\mathbb{P}(X \geq 85)$, horní konec normálního rozdělení. Proto používáme logický parametr `lower.tail=FALSE`.

Máme

```
1 > pnorm(85, mean=70, sd=10, lower.tail=FALSE)
2 [1] 0.0668072
```

Normální rozdělení – řešení b)

Abychom mohli odpovědět na otázku (b), musíme vypočítat pravděpodobnost $\mathbb{P}(X < 60)$.

Opět použijeme funkci `pnorm()`:

```
1 > pnorm(60, mean=70, sd=10)
2 [1] 0.1586553
```

Normální rozdělení – příklad 2

Príklad.
Podle údajů z www.uvzsr.sk byla průměrná výška 18-letých chlapců na Slovensku v roce 2011 179 cm se směrodatnou odchylkou 6,68 cm. Předpokládejme, že výška je normálně rozdělena, a určeme pravděpodobnost, že náhodně vybraný chlapec ve věku 18 let je

a) vyšší než 200 cm,
b) menší než 160 cm.

Normální rozdělení – řešení

Označme náhodnou veličinu, která popisuje výšku, jako X,

Abychom mohli odpovědět na otázku (a), musíme vypočítat pravděpodobnost $\mathbb{P}(X \geq 200)$.

V bodě b) potřebujeme zjistit $\mathbb{P}((\,) X < 160)$.

Pomocí funkce `pnorm()` získáme hodnotu

```
1 > pnorm(200,179,6.68,lower.tail=FALSE)
2 [1] 0.000834096
3 > pnorm(160,179,6.68)
4 [1] 0.002225376
```

Statistika a programování v R

IV. Programování v R

Funkce

Téměř všechny akce v jazyce R se provádějí pomocí funkcí.

Je implementována bohatá škála vestavěných funkcí.

Uživatel může definovat další funkce

Vestavěné funkce lze rozdělit na

- matematické funkce,
- řetězcové functions,
- specializované statistické a pravděpodobnostní funkce,
- další užitečné funkce.

Matematické funkce

O některých z nich jsme se již zmínili v lekci 1.

Zde uvedeme několik dalších podrobností

Logaritmická funkce `log()` vypočítá přirozený logaritmus jako výchozí hodnotu.

Pro získání logaritmu s libovolným základem je třeba deklarovat argument `base` funkce `log()`.

```
1 > log(4)
2 [1] 1.386294
3 > log(4,base=2)
4 [1] 2
```

Matematické funkce

Trigonometrické funkce pracují s argumentem zadaným v radiánech.

Při použití stupňů musíme hodnotu transformovat jako $r = \frac{\pi \cdot \alpha}{180}$, kde r je nová míra v radiánech a α je stará hodnota ve stupních, nebo můžeme také použít funkci `deg2rad()` z balíčku `REdaS`.

```
> library(REdaS)
> sin(90)
[1] 0.8939967
> sin(deg2rad(90))
[1] 1
```

```
> tan(45)
[1] 1.619775
> tan(deg2rad(45))
[1] 1
```

Matematické funkce – komplexní čísla

Funkce pro výpočty s komplexními čísly

- `Re(z)` Reálná část z.
- `Im(z)` Imaginární část z.
- `Mod(z)` Modul z.
- `Arg(z)` Argument z.
- `Conj(z)` komplexně sdružené číslo $\overline{z}$.

Řetězcové funkce

Funkce `nchar()` určuje velikost každého prvku vektoru znaků.

```
1  > z<-c("yellow","black","white")
2  > nchar(z)
3  [1] 6 5 5
4  > str<-"This␣is␣a␣long␣string"
5  > nchar(str)
6  [1] 21
```

řetězcové funkce

Argument `keepNA` je logická hodnota, která určuje, zda se má vrátit `NA` tam, kde je `NA` obsaženo v `x`.

```
1 > z<-c("",NULL, "black",NA)
2 > nchar(z,keepNA=TRUE)
3 [1] 0 5 NA
4 > nchar(z,keepNA=FALSE)
5 [1] 0 5 2
```

Řetězcové funkce

Seznam řetězcových funkcí

- `nchar()` Počet znaků v řetězci.
- `substr()` Výběr nebo nahrazení podřetězců.
- `grep()` Vyhledání vzoru v řetězci.
- `strsplit()` Vytvoří řetězec v daném bodě rozdělení.
- `sub()` Vyhledá vzor v řetězci a nahradí jej.
- `paste()` Spojí řetězce pomocí zadaného oddělovače.
- `toupper()` Převede řetězec na velká písmena.
- `tolower()` Převede řetězec na malá písmena.

Řetězcové funkce

Chcete-li v řetězci najít zadaný vzor, použijte funkci grep()

```
> str <- c('abcd','bdcd','abcdabcd')
> pattern<- 'abc'
> grep(pattern, str)
[1] 1 3
> pattern<- 'Abc'
> grep(pattern, str)
integer(0)
> grep(pattern, str,ignore.case=TRUE)
[1] 1 3
> pattern<- 'a*'
> grep(pattern, str)
[1] 1 2 3
> grep(pattern, str,fixed=TRUE)
integer(0)
```

Řetězcové funkce

Chcete-li nalezený vzor nahradit jiným řetězcem, použijte funkci `sub()`.

```
1 > str<-"Bohemia␣does␣not␣use␣EURO␣currency"
2 > str<-sub("Bohemia","Czechia",str)
3 > str
4 [1] "Czechia␣does␣not␣use␣EURO␣currency"
```

K dispozici je nepovinný argument `ignore.case`, logická hodnota.

Řetězcové funkce

Další funkcí pro manipulaci s textovým řetězcem je `substr()`.

Máá tři argumenty: textový řetězec `x` a `start` a `stop` pro deklaraci pozice prvního a posledního znaku, jenž má být vybrán nebo nahrazen.

```
1 > str<-"Bohemia␣does␣not␣use␣EURO␣currency"
2 > substr(str,1,7)
3 [1] "Bohemia"
4 > substr(str, 1, 5)<-"Czech"
5 > str
6 [1] "Czechia␣does␣not␣use␣EURO␣currency"
```

Řetězcové funkce

Funkce `strsplit()` rozdělí prvky znakového vektoru `x` na pozice definované druhým argumentem `split`.

```
> strsplit(str,"")
[[1]]
[1]"C" "z" "e" "c" "h" "i" "a" "␣" "d" "o" "e" "s" "␣" "n" "o" "t" "␣" "u"
[20] "e" "␣" "E" "U" "R" "O" "␣" "c" "u" "r" "r" "e" "n" "c" "y"
> strsplit(str, "␣")
[[1]]
[1] "Czechia" "does" "not" "use" "EURO" "currency"
> strsplit(str, "e")
[[1]]
[1] "Cz" "chia␣do" "s␣not␣us" "␣EURO␣curr" "ncy"
```

Řetězcové funkce

Funkce `strsplit()` rozdělí prvky znakového vektoru `x` na pozice definované druhým argumentem `split`.

O funkci `paste()` pro spojování řetězců jsme se již zmínili.

Argumenty jsou řetězce, které mají být spojeny, a `sep`, který definuje jejich oddělovač.

```
1 > paste("x",1:4,sep="")
2 [1] "x1" "x2" "x3" "x4"
3 > paste("Today␣is",date(),sep="␣")
4 [1] "Today␣is␣Tue␣Apr␣27␣10:39:55␣2021"
5 > paste(c("a","b"),1:4,sep="/")
6 [1] "a/1" "b/2" "a/3" "b/4"
```

Řetězcové funkce

Dvě související funkce `toupper()` a `tolower()` transformují zadaný řetězec na velká a malá písmena.

```
> toupper(str)
[1] "CZECHIA␣DOES␣NOT␣USE␣EURO␣CURRENCY"
> tolower(str)
[1] "czechia␣does␣not␣use␣euro␣currency"
```

Elementární statistické funkce

- `mean()` Průměrná hodnota vzorku.
- `median()` Medián vzorku.
- `sd()` Směrodajná odchylka.
- `var()` Výběový rozptyl .
- `mad()` Absolutní odchylka mediánu.
- `quantile()` Výběrové kvantily, implicitní jsou kvartily.
- `range()` Rozsah hodnot.
- `sum()` Součet prvků vektoru.
- `min()` Minimum.
- `max()` Maximum.

Elementární statistické funkce – `mean()` nepovinné argumenty

`trim`, který určuje procento nejvyšších a nejnižších hodnot, které jsou z výpočtu vynechány, a vrací tak ořezaný průměr.

Druhý nepovinný argument `na.rm` je logická hodnota, která určuje, zda se mají před pokračováním výpočtu odstranit hodnoty `NA`.

```
1  > x<-c(1,3,5,10,12)
2  > mean(x)
3  [1] 6.2
4  > mean(x,trim=0,2)
5  [1] 6
```

```
1  > x<-c(1,5,2,12,NA,3,6)
2  > mean(x)
3  [1] NA
4  > mean(x,na.rm=TRUE)
5  [1] 4.833333
6  > mean(x,na.rm=TRUE,trim=0.17)
7  [1] 4
```

Elementární statistické funkce – quantiles()

Výchozím výsledkem jsou kvartily

Pro zadání pravděpodobnostních úrovní pro kvantily je třeba zadat nepovinný argument prob ve formě číselného vektoru.

```
1  > delay<-c(0,9,0,42,14,0,11)
2  > quantile(delay)
3    0% 25% 50% 75% 100%
4   0.0 0.0 9.0 12.5 42.0
5   > quantile(delay,prob=c(0,0,33,0,67,1))
6     0% 33% 67% 100%
7   0.00 0.00 11.06 42.00
```

Elementární statistické funkce – `mad()`

Absolutní mediánová odchylka je robustní míra variability jednorozměrného vzorku kvantitativních dat.

Pro vzorek $X_1, \ldots, X_n$ je definován vzorcem:

$$\mathrm{MAD}(X) = \mathrm{median}\{|X_i - \overline{X}|\}$$

```
1  > mad(delay)
2  [1] 13.3434
```

Užitečné funkce – seq()

Funkce `seq()` generuje posloupnost čísel začínající `from` a končící `to`. Poslední argument `by` určuje krok posloupnosti.

```
> seq(10)
 [1] 1 2 3 4 5 6 7 8 9 10
> seq(5,15)
 [1] 5 6 7 8 9 10 11 12 13 14 15
> seq(5,15,2)
[1] 5 7 9 11 13 15
```

Užitečné funkce – rep()

Funkce rep() má dva argumenty, vektor x, který se má opakovat, a počet cyklů opakování n.

```
> rep(1,10)
 [1] 1 1 1 1 1 1 1 1 1 1
> rep(c(1,3),4)
[1] 1 3 1 3 1 3 1 3
> rep("hello",3)
[1] "hello" "hello" "hello"
```

Užitečné funkce – `sort()` a `order()`

Funkce `sort()` a `order()` jsou spojeny s uspořádáním prvků vektoru `x`.

`sort()` poskytuje vzestupně seřazené hodnoty, zatímco `order()` poskytuje indexy seřazených komponent v původním vektoru.

```
1  > x<-c(5,2,10,3,7,8)
2  > sort(x)
3  [1]  2  3  5  7  8 10
4  > order(x)
5  [1]  2  4  1  5  6  3
```

Užitečné funkce – `rev()`

Dává vektor `x` v opačném pořadí prvků

```
> rev(x)
[1]  8  7  3 10  2  5
> rev(sort(x))
[1] 10  8  7  5  3  2
```

Podmíněné příkazy – `if`

`if()` příkaz provádí operace na základě jednoduché podmínky

`if (podmínka) {příkaz, který se provede, pokud podmínka platí}`

Více než jeden výrok musí být v závorkách

```
1  > x<-5
2  > if(x%%2){print("Odd number")}
3  [1] "Odd number"
4  > x<-6
5  > if(x%%2){print("Odd number")}
6  >
```

Podmíněné příkazy – příkaz `if ... else`

Toto rozšíření příkazu `if` má obecnou syntaxi ve tvaru:

```
if (test_expression) {
příkaz1
} else {
příkaz2
}
```

```
1 > x<-5
2 > if(x%%2){print("Odd number")} else {print ("Even number")}
3 [1] "Odd number"
4 > x<-10
5 > if(x%%2){print("Odd number")} else {print ("Even number")}
6 [1] "Even number"
```

Podmíněné příkazy – příkaz `if ... else`

Úroveň řízení můžeme dále přizpůsobit vnořením příkazu else if. Pomocí else if můžeme přidat libovolný počet podmínek. Syntaxe je následující:

```
if (condition1) {
    příkaz1
    } else if (podmínka2) {
    příkaz2
    } else if (podmínka3) {
    prříkaz3
    } else {
    příkaz4
}
```

Podmíněné příkazy –příklad `if... else`

Príklad.
Sazby DPH se liší v závislosti na zakoupeném produktu. Předpokládejme, že máme tři různé druhy výrobků s různými sazbami DPH (které na Slovensku skutečně platí):

Kategorie	**Zboží**	**DPH**
A	Roušky, respirátory (ve skutečnosti osvobozené od DPH)	0%
B	Vybrané potraviny, knihy, časopisy, léky	10%
C	Vše ostatní	20%

Napište příkaz, který použije správnou sazbu DPH na výrobek, který si zákazník zakoupil.

Podmíněné příkazy – if ... else řešení

```
> category <- "B"
> price<-50
> if (category =="A"){
  cat("A␣vat␣rate␣of␣0%␣is␣applied.","The␣total␣price␣is",price *1.00)
} else if (category =="B"){
    cat("A␣vat␣rate␣of␣10%␣is␣applied.","The␣total␣price␣is",price *1.10)
} else {
    cat("A␣vat␣rate␣of␣20%␣is␣applied.","The␣total␣price␣is",price *1.20)
}
A vat rate of 10% is applied. The total price is 55
```

Podmíněné příkazy – `ifelse`

Příkazy `if` a `if ... else` by se neměly používat, pokud je v podmínce vyhodnocován vektor.

Příkaz `if` vyhodnotí podmínku pouze pro první prvek vektoru.

```
1  > x<-c(5,4,3,2,1)
2  > if(x>3){x*2}
```

lze očekávat, že výsledek bude `10,8,3,2,1`. Skutečný výsledek je však následující:

```
1  [1] 10  8  6  4  2
2  Warning message:
3  In if (x > 3) { :
4    the condition has length > 1 and only the first element
5     will be used
```

Podmíněné příkazy – `ifelse`

Abychom získali očekávaný výsledek, musíme použít příkaz `ifelse` s obecnou syntaxí:

```
ifelse(podmínka, výraz1, výraz2)
```

```
1  > ifelse(x>3,2*x,x)
2  [1] 10  8  3  2  1
```

Podmíněné příkazy – switch

`switch()` testuje výraz oproti prvkům seznamu. Každá hodnota v seznamu se nazývá case

Syntaxe funkce `switch()`:

```
switch (výraz, seznam)
```

```
1 > x<-10
2 > switch(x%%2+1, "even", "odd")
3 [1] "even"
4 > x<-9
5 > switch(x%%2+1, "even", "odd")
6 [1] "odd"
```

Podmíněné příkazy – switch

Pokud je výrazem řetězec znaků, funkce switch() vrátí hodnotu na základě názvu prvku.

```
> x <- "a"
> switch(x, "a"="apple", "b"="banana", "c"="cherry")
[1] "apple"
> x <- "c"
> switch(x, "a"="apple", "b"="banana", "c"="cherry")
[1] "cherry"
```

Podmíněné příkazy – `switch`

V případě vícenásobné shody je vrácena hodnota prvního odpovídajícího prvku.

Můžeme definovat výchozí hodnotu, která bude vrácena, pokud nebude nalezena shoda.

```
> x <- "a"
> switch(x, "a"="apple", "a"="apricot", "a"="avocado")
[1] "apple"
> x <- "x"
> switch(x, "a"="apple", "b"="banana", "c"="cherry","some␣fruit")
[1] "some␣fruit"
```

Cyklus – `for`

Cyklus `for` nám umožňuje pevný počet opakování příkazu nebo bloku příkazů.

Obecná syntaxe cyklu `for` je následující:

```
for (val in sequence)
{
příkaz
}
```

kde `sequence` je vektor a `val` nabývá během cyklu každé z hodnot sequence.

Cyklus – for

```
> x<-c(2,5,10,8,6,3,12)
> limit<-mean(x)
> count<-0
> for(i v x){
if (i>limit) count<-count+1
  }
> count
[1] 3
```

Cyklus – `for`

Cyklus můžeme zastavit dříve, než projde všechny položky, pomocí příkazu `break`.

```
> x<-c(2,4,6,5,8,10,11,12,14,20)
> for (i in x){
    if(i%%2==1) {break}
  print(i/2)
  }
[1] 1
[1] 2
[1] 3
```

Cyklus – `for`

Pomocí `next` můžeme přeskočit iteraci, aniž bychom ukončili cyklus.

```
> x<-c(2,4,5,8,11,20)
> for (i in x) {
+ if(i%%2==1) {next}
+ print(i/2)
+ }
[1] 1
[1] 2
[1] 4
[1] 10
```

Cyklus – `while`

Užitečné, když chceme opakovat příkaz nebo blok příkazů, dokud není splněna určitá podmínka.

```
while (podmínka){
 příkazy
}
```

Cyklus – `while`

Pomocí cyklu `while` simulujte hod kostkou, dokud nepadne prvních šest bodů.

```
> roll<-0
> while(roll!=6){
  roll<-sample(1:6,1)
  print(roll)
  }
[1] 1
[1] 4
[1] 3
[1] 4
[1] 6
```

Cyklus– repeat

Podobně jako cyklus `while`, avšak blok příkazů se provede alespoň jednou bez ohledu na splnění podmínky.

```
repeat{
příkaz
}
```

V cyklu repeat se nekontroluje podmínka pro ukončení cykluy. Podmínku musíme explicitně vložit do těla cyklu a příkazem break cyklus ukončit.

Cyklus – repeat

Pomocí cyklu repeat simulujte hod kostkou až do prvního hodu šesti bodů.

```
> repeat{
  roll<-sample(1:6,1)
  print(roll)
  if(roll==6){break}
  }
[1] 5
[1] 2
[1] 1
[1] 5
[1] 6
```

Uživatelem definovaná funkce

Obecná struktura funkce je

```
myfunction_name <- function(arg1, arg2, ... ){
příkazy
return(objekt)
}
```

Uživatelem definovaná funkce

Jednotlivé složky funkce jsou:

- **Název funkce**, což je skutečný název funkce. Je uložen v prostředí R jako objekt s tímto názvem.
- **Argumenty**, což jsou zástupné znaky. Při volání funkce předáváme hodnoty argumentů. Argumenty jsou nepovinné, to znamená, že funkce nemusí obsahovat žádné argumenty. Argumenty mohou mít také výchozí hodnoty.
- **Tělo funkce**, která obsahuje sadu příkazů definujících, co funkce dělá. Tělo funkce se nachází uvnitř složených závorek {}.
- **Návratová hodnota**, což je poslední výraz v těle funkce, který se vyhodnocuje.

Uživatelem definovaná funkce

Definujme funkci `cubes()`, která vypíše třetí mocninu čísel v posloupnosti.

```
1  cubes <- function(a) {
2     for(i in 1:a) {
3        b <- i^3
4        print(b)
5     }
6  }
```

```
1  > cubes(6)
2  [1] 1
3  [1] 8
4  [1] 27
5  [1] 64
6  [1] 125
7  [1] 216
```

Uživatelem definovaná funkce

Funkci můžeme definovat bez argumentů. Za těchto okolností vytváří posloupnost třetích mocnin konstantní délky.

```
1  cubes <- function() {
2     for(i v 1:5) {
3        b <- i^3
4        print(b)
5     }
6  }
```

```
1  > cubes()
2  [1] 1
3  [1] 8
4  [1] 27
5  [1] 64
6  [1] 125
```

Uživatelem definovaná funkce

Argumenty volání funkce lze zadávat ve stejném pořadí, v jakém jsou definovány ve funkci.

```
1  cubes <- function(start,end) {
2     for(i in start:end) {
3        b <- i^3
4        print(b)
5     }
6  }
```

```
1  > cubes(12,10)
2  [1] 1728
3  [1] 1331
4  [1] 1000
```

Uživatelem definovaná funkce

Alternativně můžeme funkci volat podle názvů argumentů

```
> cubes(end=12,start=10)
[1] 1000
[1] 1331
[1] 1728
```

Uživatelem definovaná funkce

Funkci cubes() můžeme nadefinovat s výchozími argumenty

```
1  cubes <- function(start=1,end=10) {
2     for(i in start:end) {
3        b <- i^3
4        print(b)
5     }
6  }
```

```
1  > cubes(end=4)
2  [1] 1
3  [1] 8
4  [1] 27
5  [1] 64
```

Uživatelem definovaná funkce

Co se stane, když chceme do proměnné vložit hodnotu `cubes(2,2)`?

```
1 > z<-cubes(2,2)
2 [1] 8
3 > z
4 NULL
```

Proměnná `z` neobsahuje žádnou hodnotu

Uživatelem definovaná funkce

Funkci musíme definovat pomocí návratové hodnoty `return()`

```
1  cubes <- function(start=1,end=10) {
2     for(i in start:end) {
3        b <- i^3
4        return(b)
5     }
6  }
7  > z<-cubes(2,2)
8  > z
9  [1] 8
```

Uživatelem definovaná funkce

Funkce cubes() ve skutečnosti vrací pouze jednu hodnotu

Pokud chceme výsledek rozšířit na celý rozsah, musíme výstupní proměnnou definovat jako vektor.

```
1 cubes <- function(start=1,end=10) {
2     b<-vektor() # inicializace vektoru
3    for(i in start:end) {
4         b[i-start+1]<-i^3 # zmena indexu
5    }
6    return(b)
7 }
```

Uživatelem definovaná funkce

Nyní získáme úplnou posloupnost třetích mocnin v uvedeném rozsahu:

```
1 > z<-cubes(4,8)
2 > z
3 [1] 64 125 216 343 512
```

Uživatelem definovaná funkce

V programování v jazyce R funkce nevracejí více hodnot.

Můžeme však vytvořit seznam, který obsahuje více objektů, které má funkce vrátit.

```
1  powers<-funkce(start=1,end=10) {
2          b<-vektor()
3          c<-vektor()
4          for(i in start:end) {
5          b[i-start+1]<-i^2
6          c[i-start+1]<-i^3
7     }
8     out<-list(b,c)
9     return(out)
10 }
```

Uživatelem definovaná funkce

Nyní ji můžeme použít k získání výstupu ve formě seznamu

```
1  > powers(1,5)
2  [[1]]
3  [1]  1  4  9  16  25
4
5  [[2]]
6  [1]  1  8  27  64  125
```

Spouštění skriptů v R

Skript R je jednoduše textový soubor obsahující (téměř) stejné příkazy, jaké byste zadali při psaní

Lze jej vytvořit v libovolném jednoduchém textovém editoru a uložit s příponou `.R`.

Pro spuštění skriptu v systému Linux existují v zásadě dva příkazy.

```
Rscript filename.R
```

který je preferován. Starší příkaz je

```
R CMD BATCH název souboru.R
```

Statistika a programování v R

V. Základní grafika v R

Bodové grafy

Vytvoříme je jednoduše pomocí funkce `plot()`.

Nejjednodušeji funkce přijímá dva argumenty `x` a `y`.

Tyto proměnné jsou vektory, které obsahují hodnoty, jež chceme vykreslit.

Délka vektorů musí být stejná.

Bodové grafy

Príklad.
Předpokládejme, že místní zmrzlinárna sleduje, kolik zmrzliny prodá v závislosti na polední teplotě v daný den. Zde jsou jejich údaje za posledních 10 dní:

Temperature	28	30.2	32	31	29.5	26	31.5	30	29	34
Tržby (€)	540	560	530	570	525	490	530	530	500	580

Bodové grafy

Nejprve definujeme dva číselné vektory:

`x`, který obsahuje teploty

`y`, který bude představovat denní objem prodeje

Pak nakreslíme bodový graf

```
1  > x<-c(28,30.2,32,31,29.5,26,31.5,30,29,34)
2  > y<-c(540,560,530,570,525,490,530,530,500,580)
3  > plot(x,y)
```

Jak uložit graf

Pomocí příkazu `dev.copy()` můžeme obsah grafického okna uložit do souboru, aniž bychom museli znovu zadávat příkazy.

Pro vytvoření souboru `newplot.png` z našeho grafu zadáme:

```
1 > dev.copy(png,'newplot.png')
2 > dev.off()
```

Jak uložit obrázek

Případně můžeme výstup z obrazovky přesměrovat do souboru.

Můžeme použít funkce

`pdf()`	Vektor `pdf` formát, nejlepší volba při použití s `pdflatex`
`svg()`	Vektorový `svg` formát, snadná změna velikosti.
`postscript()`	Vektorový postscriptový formát `ps`, snadná změna velikosti.
`png()`	Bitmapový formát s vysokým rozlišením, nelze bez ztráty měnit velikosti.
`jpeg()`	Komprimovaný formát bitmapy, nemění velikost beze ztrát.
`bmp()`	Bitmapový formát s vysokým rozlišením, nemění velikost bezeztrátově.
`tiff()`	Bitmapový formát s vysokým rozlišením, nemění velikost bezeztrátově.

Možnosti ukládání grafů

`filename`	Název uloženého souboru, v případě potřeby s úplnou cestou.
`width`	Šířka výsledného grafu, výchozí hodnota 7 in.
`height`	Výška výsledného grafu, výchozí hodnota 7 in.
`res`	rozlišení obrázku, platí pro formáty bitmap, výchozí 72 dpi.
`units`	Jednotky míry.
`bg`	Barva pozadí.
`fg`	Barva popředí.
`family`	Použité písmo (výchozí Helvetica).

Modifikace grafu – značkovací body

Značka bodu je dána hodnotou argumentu `pch` funkce `plot()`.

Možné hodnoty

□ pch=0	○ pch=1	△ pch=2	+ pch=3	× pch=4
◇ pch=5	▽ pch=6	⊠ pch=7	✳ pch=8	⨁ pch=9
⊕ pch=10	✡ pch=11	⊞ pch=12	⊗ pch=13	⧄ pch=14
■ pch=15	● pch=16	▲ pch=17	♦ pch=18	● pch=19
• pch=20	○ pch=21	□ pch=22	◇ pch=23	△ pch=24

Modifikace grafu –značky bodů

Zkusme upravit náš graf

```
1 > plot(x,y,pch=17)
2 > plot(x,y,pch=1)
```

Modifikace grafu –značky bodů

Zkusme upravit náš graf

```
1 > plot(x,y,pch=17)
2 > plot(x,y,pch=1)
```

Co upravit dále?

Modifikace grafu –značky bodů

Zkusme upravit náš graf

```
1  > plot(x,y,pch=17)
2  > plot(x,y,pch=1)
```

Co upravit dále? Typ čáry spojující body

Modifikace grafu – typ spojnice

Typ spojnice bodů se nastavuje pomocí argumentu `type` funkce `plot()`.

Možné hodnoty

`p`	Bodový graf, výchozí hodnota.
`l`	Souvislá čára.
`b`	Souvislá čára s body.
`c`	Části spojitých čar s vynechanými body.
`o`	Části souvislých čar, body překresleny.
`h`	Graf podobný histogramu.
`s`	Schodovitý graf.

Modifikace grafu – typ čáry

Zkusme upravit náš graf

```
> plot(x,y,type="l")
> dev.off()
> plot(x,y,type="s")
> dev.off()
> plot(x,y,pch=17,type="b")
> dev.off()
```

Modifikace grafu – typ čáry

Zkusme upravit náš graf

```
> plot(x,y,type="l")
> dev.off()
> plot(x,y,type="s")
> dev.off()
> plot(x,y,pch=17,type="b")
> dev.off()
```

Co upravit dále?

Modifikace grafu – typ čáry

Zkusme upravit náš graf

```
> plot(x,y,type="l")
> dev.off()
> plot(x,y,type="s")
> dev.off()
> plot(x,y,pch=17,type="b")
> dev.off()
```

Co upravit dále? Styl spojovací linky

Modifikace grafu – styl spojnice

Styl čáry se nastavuje pomocí argumentu `lty` funkce `plot()`.

Možné hodnoty

1	Tlustá čára (výchozí).	2	Čárkovaná čára.
3	Tečkovaná čára.	4	Tečky a čárky.
5	Dlouhé čárky.	6	Dlouhá a krátká dvojité přerušované čárky.

Šířka čáry se nastavuje pomocí argumentu `lwd` funkce `plot()`.

Modifikace grafu – styl spojnice

Zkusme upravit náš graf

```
1 > plot(x,y,type="l",lty=5)
2 > dev.off()
3 > plot(x,y,type="l",lty=1,lwd=2)
4 > dev.off()
```

Modifikace grafu – styl spojnice

Zkusme upravit náš graf

```
1 > plot(x,y,type="l",lty=5)
2 > dev.off()
3 > plot(x,y,type="l",lty=1,lwd=2)
4 > dev.off()
```

Co upravit dále?

Modifikace grafu – styl spojnice

Zkusme upravit náš graf

```
1 > plot(x,y,type="l",lty=5)
2 > dev.off()
3 > plot(x,y,type="l",lty=1,lwd=2)
4 > dev.off()
```

Co upravit dále? Barvu

Modifikace grafu – barvení

Než začneme, jeden problém:

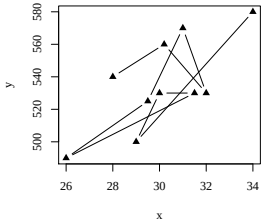
```
1  > plot(x,y,pch=17,type="l")
```

Modifikace grafu – barvení

Než začneme, jeden problém:

```
1  > plot(x,y,pch=17,type="l")
```

Dává

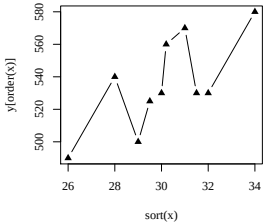


Modifikace grafu – barvení

Než začneme, jeden problém:

```
1  > plot(x,y,pch=17,type="l")
```

Ale my bychom chtěli

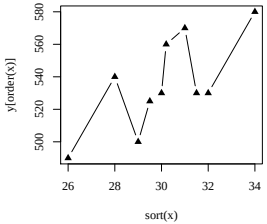


Modifikace grafu – barvení

Než začneme, jeden problém:

```
1  > plot(x,y,pch=17,type="l")
```

Ale my bychom chtěli



Jak to zařídit?

Modifikace grafu – styl čáry

Odpověď

Použijte `sort()` a `order()`

```
1  > plot(sort(x),y[order(x)],pch=17,type="b")
2  > dev.off()
```

Modifikace grafu – barvení

Barvy můžeme upravit pomocí

- názvu barev položek, například `col=red`
- číslo barvy položky, například `col=636`
- podle hexadecimálního kódu (v režimu RGB), například `col="#FFCC00"`

Seznam dostupných barev lze získat jako výstup funkce `colors()`.

Modifikace grafu – barvení

Zkusme

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col="red")
2 > dev.off()
3 > plot(sort(x),y[order(x)],pch=17,type="b",col=636)
4 > dev.off()
5 > plot(sort(x),y[order(x)],pch=17,type="b",col="#FFCC00")
6 > dev.off()
```

Modifikace grafu – barvení

Další možnosti obarvení jsou

`col.axis`	Barva popisků os.	`col.lab`	Barva označení os.
`col.main`	Barva hlavního nadpisu.	`col.sub`	Barva podnadpisu.
`bg`	Barva výplně znaku.	`fg`	Barva podkladu grafu.

Modifikace grafu – barvení

Zkusme

```
>plot (sort(x),y[order(x)],lty =1, type ="b",col="aquamarine",lwd =2,
  col.axis ="violet",col.main ="green",main ="Main␣title",fg="red",
  col.lab="coral3",pch=17)
> dev.off()
>par(bg="beige")
>plot (sort(x),y[order(x)],lty =1, type ="b",col=30,lwd =2,
  col.axis ="darkmagenta", col.main ="blue3",col.sub="blue2",
  main ="Main␣title", sub="Subtitle", fg="red",col.lab="coral4",pch=17)
> dev.off()
```

Modifikace grafu – barvení

Barvy jako vektory

Hodnotu argumentu `col` můžeme nastavit jako vektor.

Barvy z vektoru se pak pravidelně střídají.

Můžeme také použít funkci `rainbow()` s předem definovanou posloupností barev.

Modifikace grafu – barvení

Zkusme

```
1  > plot(sort(x),y[order(x)],pch=17,type="b",
2    col=c("red", "blue")
3  > dev.off()
4  >plot(sort(x),y[order(x)],pch=17,type="b",col=rainbow(5))
5  > dev.off()
```

Úpravy grafu – názvy a titulky

Základní kreslicí funkce v R obsahují argument `main`, který umožňuje přidat ke grafu nadpis.

Můžeme také použít argument `sub` pro přidání podnadpisu, který bude umístěn pod grafem.

Alternativním způsobem, jak do grafu přidat nadpis a podnadpis, je použití funkce `title()`.

Úprava grafu – názvy a titulky

Zkusme

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=rainbow(4))
2 > title(main="Icecream␣sales",col.main="red")
3 > title(sub="Temperature",col.sub="blue",adj=1,line=2)
4 > dev.off()
```

Úprava grafu – přidání textu do grafu

Do nakresleného grafu můžeme přidat libovolný text pomocí funkcí `text()` a `mtext()`.

Funkce `text()` umístí zadaný text na libovolné místo v kreslicí oblasti, funkce `mtext()` umístí text na okraje.

Funkce `text()` přijímá dva další argumenty:

- `location` definuje souřadnice `x` a `y`, kde bude text umístěn. Souřadnice musí být zadány jako první dva argumenty funkce.
- `pos` určuje umístění ve vztahu k aktuální poloze, 1=dolů, 2=doleva, 3=nahoru a 4=doprava. Definování pozice jako `locator(1)` umožňuje umístění textu pomocí myši.

Úprava grafu – přidání textu do grafu

Zkusme

```
1  > plot(sort(x),y[order(x)],pch=17,type="b",col=30)
2  > title(main="Icecream␣sales",col.main="red")
3  > title(sub="Temperature",col.sub="blue",adj=1,line=2)
4  > text(c(28,32),c(560,500),c("Text1","Text2"),pos=1,col="red")
5  > dev.off()
```

Úprava grafu – přidání textu do grafu

Funkce `mtext()` má další dva argumenty:

- `side` určuje stranu plochy grafu, na kterou umístíme textový popisek, 1=dole, 2=vlevo, 3=nahoře a 4=vpravo.
- `line` určuje číslo řádku, na kterém bude umístěn popisek. Řádky jsou číslovány od 0.

Zkusme

```
1
2 > plot(sort(x),y[order(x)],pch=17,type="b",
3 +   col=30,xlab="",ylab="")
4 > mtext("Temperature",side=1,line=2,adj=1)
5 > mtext("Sales",side=2,line=2)
```

Úprava grafu – uzpůsobení os

Pokud chceme odstranit rámeček grafu, nastavíme argument kreslicí funkce `axes=FALSE`.

Nové osy přidáme pomocí funkce `axes()`.

Argument funkce `axis()` určuje stranu grafu, na kterou bude osa přidána.

Jako obvykle čísla určují strany: 1=spodní, 2=levá, 3=horní a 4=pravá.

Zkusme

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE)
2 > axis(1)
3 > axis(2)
```

Úprava grafu – nastavení os

Další možností přizpůsobení je změna barev os. Toho můžeme dosáhnout nastavením volitelných argumentů funkce `axis()`:

- `col` určuje barvu osy,
- `col.ticks` určuje barvu dělících bodů,
- `col.axis` určuje barvu značek.

Zkusme

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE)
2 > axis(1,col="blue",col.ticks="red",col.axis=555)
3 > axis(2,col="deepskyblue2",col.ticks=444,col.axis="red")
```

Další nastavení os

Můžeme také:

- určit počet značek se zadanými počátečními a koncovými hodnotami,
- upravit délku a orientaci značek,
- otáčet popisky značek,
- přizpůsobit popisy značek,
- odstranit značky,
- přidat menší značky pomocí `Hmisc` .

Další přizpůsobení os – rozdělení intervalů

Argumenty `xaxp` a `yaxp` umožňují nastavit polohu dělících značek na osách `x` a `y`.

Jejich hodnoty jsou nastaveny jako vektory `c(start,end,regions)`, `start` a `end` definují počáteční a koncové hodnoty na každé ose a `regions` definuje počet intervalů, na něž je osa rozdělena.

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE)
2 > axis(1,col="blue",col.ticks="red",col.axis=555,xaxp=c(26,34,8))
3 > axis(2,col="blue",col.ticks="red",col.axis=555,yaxp=c(490,580,9))
```

Další nastavení os – délka a orientace značek

Argument `tck` umožňuje nastavit délku a orientaci dělících značek.

Jeho kladná hodnota orientuje značky dovnitř kreslící plochy, zatímco záporné hodnoty orientují značky vně kreslící plochy. Čím větší je absolutní hodnota, tím delší jsou značky. Výchozí hodnota je `tck=-0,05`.

Rotace je omožněna argumentem `las`, který může nabývat jedné ze čtyř hodnot:

- `las=0` popisky jsou rovnoběžné s osou (výchozí),
- `las=1` všechny popisky jsou vodorovné,
- `las=2` popisky jsou kolmé na osu,
- `las=3` všechny popisky jsou svislé.

Další nastavení osy - délka a orientace značek

Zkuste

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE)
2 > axis(1,col="blue",xaxp=c(26,34,8),tck=0.02,las=3)
3 > axis(2,col="blue",yaxp=c(490,580,9),tck=0.02,las=2)
```

Poznámka

Oddělovače můžeme zcela odstranit nastavením argumentů `xaxt="n "` pro osu `x` nebo `yaxt="n "` pro osu `y`.

Další nastavení osy – popis dělících značek

Popisy dělících značek lze změnit pomocí argumentu `labels` funkce `axis()`.

Pro správné umístění štítků musíme nastavit jejich pozici pomocí argumentu `at`.

```
1  > plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE)
2  > axis(1,col="blue",at=seq(round(min(x)),round(max(x)),by=1),
3  + labels=0:8)
4  > axis(2,col="blue",yaxp=c(490,580,9),tck=0.02,las=2)
```

Rozsah os a přizpůsobení

Rozsah hodnot pro osy lze definovat pomocí nepovinných argumentů `xlim` a `ylim` funkce `plot()`.

Hranice jsou zadány jako vektory ve tvaru `c(start,end)`

Můžeme také transformovat osy do logaritmického měřítka tak, že nastavíme argument `log` na hodnotu osy, kterou plánujeme přizpůsobit.

`log="x "` nastaví logaritmickou stupnici na osu `x`,
`log="y "` nastaví logaritmickou stupnici na osu `y` a
`log="xy "` transformuje obě osy do logaritmického měřítka.

Rozsah os a přizpůsobení

Zkuste

```
1 > plot(sort(x),y[order(x)],pch=17,type="b",col=30,axes=FALSE,
2 + ylim=c(400,600))
3 > axis(1,col="blue",at=seq(round(min(x)),round(max(x)),by=1),
4 + labels=0:8)
5 > axis(2,col="blue",yaxp=c(490,580,9),tck=0.02,las=2)
```

Duální svislé osy

Príklad.
Do jednoho grafu chceme zakreslit dvě charakteristiky zdravotního stavu pacientů, teplotu a krevní tlak.

Duální svislé osy

Príklad.
Do jednoho grafu chceme zakreslit dvě charakteristiky zdravotního stavu pacientů, teplotu a krevní tlak.

V proměnných y a z máme uloženy údaje o 100 pacientech, přičemž proměnná x obsahuje posloupnost identifikátorů pacientů, čísla od 1 do 100.

Duální svislé osy

Nejprve upravíme okraje oblasti kreslení pomocí `par(mar = c(3, 4, 2, 4))`.

Poté vykreslíme bodový graf naměřených teplot.

Důležitým krokem je nastavení nového grafu pomocí `par(new=TRUE)`. Nyní jsme připraveni vykreslit druhou sadu dat modře, bez rámečků a bez os.

duální osa `y` je vykreslena pomocí funkce `axis(4)` na pravé straně grafu.

Duální svislé osy

```
x<-1:100 # generovani dat
y <- runif(100, min=35, max=40)
z <- y+10*runif(100,min=7,max=12)
par(mar = c(3, 4, 2, 4))
plot(x, y, pch = 19, ylab = "Temperature")
par(new=TRUE)
plot(x, z, col = 4, pch = 19,
     axes = FALSE, # Zadne osy
     bty = "n", # Zadny ramecek
     xlab = "", ylab = "")
osa(4)
mtext("Blood preasure", side = 4, line = 3, col = 4)
```

Křivky

Jednou z mnoha užitečných funkcí v R je `curve()`.

Je to malá šikovná funkce, která umožňuje vykreslovat křivky, např. grafy funkcí.

Funkce `curve()` přijímá jako první argument výraz v syntaxi R.

Například,

```
curve(x^2)
curve(x^2,xlim=c(-2,2),col="red",lwd=2)
```

Zobrazení dvou nebo více křivek v jednom grafu

Použijeme funkci `curve()` s argumentem `add=TRUE`.

Například

```
curve(x^2)
curve(sqrt(x),col="red",lwd=2,add=TRUE)
```

Zobrazení dvou nebo více křivek v jednom grafu

Použití funkce `curve()` není omezeno na její samostatné použití.

Je možné vykreslit nějaká data a pak přes ně pomocí funkce `curve()` nakreslit libovolnou čáru.

```
1 set.seed(1)
2 x <- rnorm(100)
3 y <- x^2 + rnorm(100)
4 plot(y ~ x)
5 curve(x^2,add=TRUE)
```

Přidání legendy

Funkce `legend()` umožňuje přidat ke grafům legendu.

Některé argumenty:

- `x,y` pozice v kreslicí oblasti definovaná souřadnicemi v grafu,
- `legend`vektor řetězců pro popis v legendě,
- `col` vektor barev použitých v grafu,
- `pch` vektor tvarů značek použitých v grafu,
- `lty` vektor typů čar použitých v grafu,
- `ncol` počet sloupců použitých v legendě, výchozí hodnota je jeden sloupec.

Doplnění legendy

Príklad.
Vytvořme uživatelskou funkci `gonplot()`, která vykreslí grafy $\sin x$ a $\cos x$ v rozpětí $(-10; 10)$ ve dvou barvách a různých typech čar. Poté ke grafu přidáme legendu.

Doplnění legendy

Uživatelem definovaná funkce

```
1 gonplot <- function() {
2     curve(sin(x),xlim=c(-10,10),col="red",lwd=2,type="l",
3     ylab="sin␣x",xlab="",ylim=c(-1,2))
4     curve(cos(x),xlim=c(-10,10),col="blue",lwd=2,type="l",lty=2,
5     ylab="sin␣x",xlab="",add=TRUE)
6 }
```

Doplnění legendy

Zobrazení grafu a přidání legendy

```
1 gonplot()
2 legend(x="topright", # pozice
3        legend = c("sin x", "cos x"), # text legendy
4        lty = c(1, 2), # Typy cary
5        col = c("red", "blue"), # Barvy car
6        lwd = 2)
```

Doplnění legendy – poznámka

Argument `x` pozice může být nastaven na jednu z hodnot:

`top`, `topleft`, `topright`, `bottom`, `bottomleft`, `bottomright`, `left`, `right` nebo `center`.

Tento scénář nevyžaduje nastavení argumentu `y`, protože poloha legendy je zadána slovně.

Sloupcové grafy

Sloupcový graf zobrazuje kategoriální data pomocí obdélníkových sloupců, jejichž výška nebo délka je úměrná hodnotám, které představují.

R používá funkci pro vytváření sloupcových grafů

```
barplot(H,xlab,ylab,title, names.arg,col)
```

Parametry použité ve funkci jsou následující:

- `H` je vektor nebo matice obsahující číselné hodnoty použité ve sloupcovém grafu,
- `xlab` je označení osy x,
- `ylab`je označení osy y,
- `title` je nadpis sloupcového grafu,
- `names.arg` je vektor popisů, které se zobrazují pod každým sloupcem,
- `col` slouží k přiřazení barev sloupcům v grafu.

Sloupcové grafy

Předpokládejme, že vektor x obsahuje denní prodeje určitých výrobků. Objem prodeje lze znázornit v podobě sloupcového grafu.

```
1  x<-c(2000,2400,1400,2600)
2  barplot(x)
```

Sloupcové grafy – vodorovné sloupce

Nastavte argument `horiz=T` na hodnotu true.

```
1  barplot(x,horiz=T)
```

Sloupcové grafy – obarvení a popisky sloupců

Pro přiřazení názvů sloupcům používáme parametr `names.arg` sloupcového grafu.

Dále definujeme hodnoty parametrů

- `xlab` a `ylab` pro názvy os,
- `col` a `border` pro obarvení sloupců, a
- `main` pro definování nadpisu grafu

Je to podobné jsko u funkce `plot()`.

Sloupcové grafy – obarvení a popisky sloupců

Nechť náš vektor x představuje denní prodej nějakého ovoce.

Jejich jména nastavíme jako vektor goods a použijeme jej k přiřazení jmen sloupcům.

```
1 goods<-c("orange","banana","apple","plum")
2 barplot(x,names.arg=goods,xlab="Fruit",ylab="Sales",
3 col="cyan",main="Monthly␣sale",border="black")
```

Sloupcové grafy – obarvení a popisky sloupců

Graf můžeme přizpůsobit různými barvami sloupců

Požadované barvy nastavíme jako vektor `colours` a použijeme jej jako hodnotu argumentu `col`.

Argument `border` určuje barvu ohraničení sloupců.

```
1 colours<-c("orange","yellow","red","blue")
2 barplot(x,names.arg=goods,xlab="Fruit",ylab="Sales",
3 col=colours,main="Monthly␣sale",border="black")
```

Sloupcové grafy – s proporcemi

Pomocí matice vstupních hodnot můžeme podíly ve sloupcích barevně odlišit a označit.

Rozsah objemů prodeje ve vektoru `x` za více měsíců.

Tyto informace pak prezentujeme graficky.

Nejprve nastavíme

```
1 months<-c("Jan", "Feb", "Mar", "Apr")
2 x<-matrix(c(2000,2400,1400,2600,1800,2200,1600,2400,2100,
3 2300,1500,2400,2400, 1800,1200,2200),nrow=4,ncol=4)
```

Sloupcové grafy – s proporcemi

Nyní jsme připraveni nakreslit graf s proporcionálními podíly

Přidáme také legendu

```
1 barplot(x, main = "Sale volumes", names.arg = months,
2 xlab = "Month", ylab = "Sales",
3 col = colours, ylim=c(0,11000))
4 legend("topright", goods, fill = colours,ncol=2)
```

Sloupcové grafy – s proporcemi

Stejné informace můžeme také prezentovat seskupením sloupců v grafu.

Nastavíme argument `beside=T`

```
1 barplot(x, beside=T,main = "Sale volumes",
2 names.arg = months, xlab = "Month", ylab = "Sales",
3 col = colours, ylim=c(0,3000))
4 legend("topright", goods, fill = colours,ncol=2)
```

Sloupcové grafy – vyplnění texturami

Namísto barev můžeme sloupce vyplnit texturami.

Nejjednodušší jsou rovnoběžné čáry

Hustotu čar můžeme regulovat argumentem `density`, jehož hodnota je vektor o délce rovné počtu čar.

Podobně můžeme zadáním argumentu `angle` jako vektoru, jehož délka je rovna počtu sloupců, určit úhel výplňových čar.

Sloupcové grafy – vyplnění čarami

```
1 x<-c(2000,2400,1400,2600)
2 barplot(x,density=c(5,10,20,30), angle=c(0,30,60,90),
3 col="blue",names.arg=goods,main="Sale␣volumes",
4 xlab="Fruit",ylab="Sales")
```

Sloupcové grafy – vyplnění křížením čar

```
angle1<-c(0,30,60,90)
angle2<-c(90,120,150,0)
barplot(x,density=c(10,15,20,25), angle=angle1,beside = TRUE,
 main="Sale volumes", col = colours,names.arg=months, xlab = "Month",
 ylab = "Sales",ylim=c(0,3000))
barplot(x,density=c(10,15,20,25), angle=angle2,beside = TRUE,
col = colours, add=TRUE)
legend("topright", goods, ncol=2, fill=colours, angle=angle1,
 density=c(10,15,20,25))
legend("topright", goods, ncol=2, fill=colours, angle=angle2,
 density=c(10,15,20,25))
```

Histogramy

Histogram je zobrazení přibližného rozložení číselných dat.

Zobrazuje četnosti hodnot rozdělených do intervalů.

Histogram je podobný sloupcovému grafu s tím rozdílem, že sdružuje hodnoty do souvislých intervalů.

Histogramy poskytují přibližný obraz hustoty základního rozdělení dat.

Histogramy

Histogram lze vytvořit pomocí funkce `hist()` v R.

```
barplot(H,xlab,ylab,title, names.arg,col)
```

Parametry použité ve funkci jsou následující:

- `data` je vektor obsahující číselné hodnoty použité v histogramu,
- `main` je nadpis grafu,
- `col` slouží k nastavení barvy sloupců,
- `border` slouží k nastavení barvy rámečku každého sloupce,
- `xlab` určuje popis osy x,
- `xlim` určuje rozsahu hodnot na ose x,
- `ylim` slouží k určení rozsahu hodnot na ose y,
- `breaks` slouží k určení šířky každého sloupce.

Histogramy - příklad

Príklad.
Ilustrujme vykreslení histogramů na příkladu hodu kostkou. Předpokládejme, že hodíme dvěma kostkami 10 000 krát a zajímá nás součet hozených bodů.

Histogramy - příklad

Nejprve simulujeme hod kostkou:

```
1 dice1<-sample(1:6,replace=T,10000)
2 dice2<-sample(1:6,replace=T,10000)
3 c<-dice1+dice2
```

Nyní můžemezobrazit histogram součtů pomocí funkce `hist()`:

```
1 hist(c,breaks=1.5:12.5, main="Rolling␣2␣dice",
2 xlab="two␣dice", ylab="Frequency")
```

Histogramy - příklad

Centrální limitní věta, známá z teorie pravděpodobnosti, říká, že v mnoha situacích, kdy se sčítají nezávislé náhodné veličiny, jejich průměrný součet konverguje k normálnímu rozdělení (neformálně ke zvonové křivce), i když původní veličiny samy o sobě normálně rozděleny nejsou.

To lze dokumentovat na histogramu zobrazením křivky hustoty normálního rozdělení do stejného grafu jako histogram.

```
1 hist(c,breaks=1.5:12.5, main="Rolling␣2␣dice",
2 xlab="two␣dice", ylab="Frequency")
3 curve(dnorm(x,mean(c),sd(c))*10000,col="red",add=T)
```

Kruhové diagramy

Kruhový diagram je graf pro jednu kategoriální proměnnou a je alternativou sloupcového grafu.

Kruhový diagram (nebo koláčový diagram) je graf ve tvaru kruhu rozděleného na výseče, jež znázorňují poměry mezi veličinami.

V kruhovém diagramu je délka oblouku každého dílku (a tedy i jeho středový úhel a plocha) úměrná množství, které představuje.

Kruhové diagramy

Základní syntaxe pro vytvoření kruhového diagramu v prostředí R je následující:

```
pie(data, labels, radius, main, col, clockwise)
```

Význam argumentů:

- `data` je vektor obsahující číselné hodnoty použité v kruhovém diagramu,
- `labels`se používá k popisu dílků,
- `radius` označuje poloměr kruhu diagramu (hodnota mezi -1 a $+1$),
- `main` označuje nadpis grafu,
- `col` označuje paletu barev,
- `clockwise` je logická hodnota, která určuje, zda jsou plátky vykresleny ve směru nebo proti směru hodinových ručiček.

Kruhový diagram–příklad

Príklad.
Předpokládejme, že chceme znázornit podíly měsíčních výdajů domácnosti pomocí kruhového diagramu. Uvažujeme tyto kategorie výdajů: bydlení, jídlo, oblečení, zábava a ostatní.

Hodnoty, které použijeme jako parametry kruhového diagramu:

```
1 data<-c(200,300,100,80,150)
2 labels<-c("housing","food","clothing","entertainment","other")
3 pie(data,labels,main="Monthly␣expenses")
```

Kruhový diagram – úprava barev

Pro změnu barev v grafu použijeme funkci `rainbow()`, která definuje paletu barev.
Jeho argumenty jsou následující:

- `n` počet barev (≥ 1), které mají být v paletě,
- `s,v` „sytost“ a „hodnota“ pro přidání popisu k barvám
- `start` (upravený) odstín v $\langle 0; 1\rangle$, od kterého začíná vybraná duha,
- `end` (uopravený) odstín v $\langle 0; 1\rangle$, ve kterém duha končí,
- `gamma` gama korekce pro každou barvu, `(r,g,b)` v prostoru RGB (se všemi hodnotami v $\langle 0; 1\rangle$), výsledná barva odpovídá $(r^\gamma, g^\gamma, b^\gamma)$,
- `alfa` průhlednost, číslo ve tvaru $\langle 0; 1\rangle$, (0 znamená průhledný a 1 znamená neprůhledný).

Kruhový diagram–použití `rainbow()`

```
1 description<-paste(labels,"\n",data,sep="")
2 pie(data,description,main="Monthly␣expenses",
3 col=rainbow(length(data)))
```

Poznámka

Změnili jsme také popisky. K jejich názvům jsme přidali číselné hodnoty.

Kruhový diagram - další vylepšení

Jako další vylepšení můžeme požadovat popisy s procenty a zobrazení grafu s 3D efektem.

Nejprve musíme přepočítat procenta a přidat výsledky do popisů. Pro získání procent jako celých číslech použijeme funkci `trunc()`.

Pak můžeme vytvořit koláčový graf (tentokrát pomocí palety) `heat.colors()`.

```
1 description<-paste(labels,"\n",trunc(100*data/sum(data)),
2 "%", sep="")
3 pie(data,description,main="Monthly␣expenses",
4 col=heat.colors(length(data)))
```

Kruhový diagram - další vylepšení

Pro získání 3D efektu v grafu je potřeba použít balíček `plotrix`.

Používáme `pie3D()` grafy s 3D efektem.

```
1  library("plotrix")
2  pie3D(data,labels=description,main="Monthly␣expenses",
3   col=rainbow(length(data)))
```

Kruhový diagram – oddělení částí

Vzhled 3D grafu můžeme dále upravit pomocí parametrů

- `height`, jenž určuje výšku 3D koláče (výchozí hodnota je 0,1)
- `theta`, která mění úhel pohledu (výchozí hodnota je $\frac{\pi}{6}$).
- `explode`, který definuje rozdělení částí koláče

```
1 pie3D(data,labels=description,main="Monthly␣expenses",
2  col=terrain.colors(length(data)),height=0.2,theta=1.5,
3 explode=0.1)
```

Všimněme si použití palety `terrain.colors`

Vějířový graf

Užitečnou alternativou ke kruhovým diagramům je `fan.plot()` definovaný v balíčku `plotrix`.

Umožňuje vizuálně porovnat sektory kruhového diagramu.

Vějířový graf můžeme přizpůsobit zadáním dalších argumentů:

- `max.span` úhel maximálního sektoru v radiánech. Ve výchozím nastavení se `data` škáluje tak, aby se součet rovnal 2π.
- `ticks` počet políček, která by se objevila, kdyby byly sektory v kruhovém diagramu. Výchozí nastavení je bez políček.

Vějířový graf

Ilustrace vějířového grafu

```
1 fan.plot(data,labels=description,main="Monthly␣expenses",
2  col=rainbow(length(data)),max.span=pi,ticks=max(data))
```

Vějířový graf

Ilustrace vějířového grafu

```
1 fan.plot(data,labels=description,main="Monthly␣expenses",
2  col=rainbow(length(data)),max.span=pi,ticks=max(data))
```

Nevýhodou vějířového grafu je velký bílý prostor nad grafem.

Tento prostor můžeme odstranit nastavením nového grafického zařízení s výškou a šířkou definovanou uživatelem.

Nové grafické okno otevřeme pomocí funkce `new.dev()`. Velikost okna definujeme pomocí argumentů `height` a `width`.

Vějířový graf

```
1  dev.new(width=10,height=5,unit="cm")
2  fan.plot(data,labels=description,main="Monthly␣expenses",
3   col=rainbow(length(data)),max.span=pi,ticks=max(data))
```

Krabicový graf

Krabicové grafy se v R vytvářejí pomocí funkce `boxplot()`. Základní syntaxe pro vytvoření boxplotu v R je:

```
boxplot(x, data, notch, varwidth, names, main)
```

Význam parametrů je následující:

- `x` je vektor nebo vzorec,
- `data` je data frame.
- `notch` je logická hodnota. Nastavte jako TRUE, pokud chcete vykreslit zářez.
- `varwidth` je logická hodnota. Nastavte jako TRUE, aby se vykreslila šířka rámečku úměrná velikosti vzorku,
- `names` jsou popisky skupin, které jsou vytištěny pod každým boxplotem,
- `main` slouží k zadání nadpisu grafu.

Karbicový graf – příklad

Príklad.
Předpokládejme, že máme statistiky z basketbalového zápasu v datovém souboru `hráči.csv`. Tento datový soubor obsahuje identifikátory hráčů, jejich pozici, počet střeleckých pokusů a počet úspěšných střeleckých pokusů. Pomocí boxplotů porovnejme body získané na jednotlivých pozicích.

```
1  players<-read.csv("players.csv")
2  boxplot(made~position,data=players,
3  xlab="Position",ylab="Points␣gained",
4  main="Scoring␣by␣position")
```

Krabicový graf

Stejně jako u ostatních typů grafů můžeme vzhled grafu přizpůsobit.

Ilustrujeme obarvení grafu a šířku políček upravíme tak, aby byla úměrná velikosti vzorku, a to nastavením `varwidth=TRUE`.

```
1 boxplot(made~position,data=players,
2 xlab="Position",ylab="Points␣gained",
3 main="Scoring␣by␣position",col="cyan",varwidth=TRUE)
```

Krabicový graf

Nastavením logické proměnné `horizontal` na hodnotu `TRUE` můžeme grafy v boxplotu otáčet.

Kromě toho se barvy mohou v jednotlivých boxech lišit.

```
1 boxplot(made~position,data=players,
2 xlab="Position",ylab="Points␣gained",
3 main="Scoring␣by␣position",
4 col="col=c("blue","cyan","green"),
5 varwidth=TRUE,horizontal=TRUE)
```

Q-Q graf

Kvantilový graf (zkráceně Q-Q graf) je grafický nástroj, který nám pomáhá posoudit, zda soubor dat věrohodně odpovídá nějakému teoretickému rozdělení, například normálnímu nebo exponenciálnímu rozdělení.

Pokud například provádíme statistickou analýzu, která předpokládá, že naše závislá proměnná je normálně rozdělena, můžeme k ověření tohoto předpokladu použít normální Q-Q graf.

Jedná se pouze o vizuální kontrolu, nikoli o exaktní důkaz, ale umožňuje nám na první pohled zjistit, zda je náš předpoklad věrohodný, a pokud ne, jak je předpoklad porušen a které hodnoty dat k porušení přispívají.

Q-Q graf

Q-Q graf je v podstatě bodový graf vytvořený vynesením dvou sad kvantilů proti sobě.

Pokud obě sady kvantilů pocházejí ze stejného rozdělení, nachází se body přibližně na přímce.

Q-Q grafy vezmou naše vzorky, seřadí je vzestupně a pak je vynesou do grafu proti kvantilům navrženého teoretického rozdělení.

Počet kvantilů je zvolen tak, aby odpovídal velikosti našeho vzorku.

Q-Q graf

V R máme dvě funkce pro vytváření Q-Q grafů:

`qqnorm()` vytvoří normální Q-Q graf (což znamená, že navržené teoretické rozdělení je normální),

`qqqplot()` umožňuje vytvořit Q-Q graf pro porovnání dvou datových souborů.

S funkcí `qqnorm()` souvisí funkce `qqline()`, která vykreslí přímku „teoretického“, standardně normálního kvantilového grafu, který prochází kvantilovými 'probs', standardně prvním a třetím kvartilem.

Ilustrace Q-Q grafu

Nejprve vytvoříme vzorek z normálního rozdělení

V dalším kroku ho porovnáme s teoretickým rozdělením

```
1 x<-rnorm(100,mean=10,sd=1)
2 qqnorm(x)
3 qqline(x, col="steelblue", lwd=2)
```

Ilustrace Q-Q grafu

Pro ilustraci situace, kdy vzorek nepochází z předpokládaného rozdělení, vygenerujeme vzorek z exponenciálního rozdělení.

```
1 x<-rexp(100,rate=1/10)
2 qqnorm(x)
3 qqline(x, col="steelblue", lwd=2)
```

Ilustrace Q-Q grafu

Pokud dva náhodné vzorky pocházejí ze stejného typu rozdělení, vytvoříme pro srovnání dva vektory `x` a `y`.

Na tyto vzorky pak použijeme funkci `qqplot()`.

```
1 x<-rnorm(100,mean=10,sd=1)
2 y<-rnorm(100,mean=5,sd=3)
3 qqplot(x,y,main="Q-Q␣plot␣for␣two␣samples")
```

Ilustrace Q-Q grafu

Funkce `qqqplot()` nespolupracuje s funkcí `qqline()`

Chceme-li přidat pomocnou přímku, použijeme funkci `abline()` společně s funkcí `sort()`.

```
1 x<-rnorm(100,mean=10,sd=1)
2 y<-rnorm(100,mean=5,sd=3)
3 qqplot(x,y,main="Q-Q␣plot␣for␣two␣samples")
4 abline(lm(sort(y) ~ sort(x)), col = "steelblue", lwd = 2)
```

Funkce `lm()` vytvoří model lineární závislosti a poskytne koeficienty potřebné k vykreslení přímky.

Q-Q graf

Funkci `qqplot()` lze použít k porovnání vzorku s jakýmkoli teoretickým rozdělením.

Vytvoříme vektor kvantilů teoretického rozdělení o stejné délce jako daný vzorek a tento vektor pak použijeme jako druhý soubor dat vstupující do funkce `qqplot()`.

```
1  x<-rexp(100,rate=1/10)
2  y<-qexp(seq(0,1,by=0.01),rate=1)
3  qqplot(x,y,main="exponential␣Q-Q␣plot")
4  abline(lm(sort(y[1:100]) ~ sort(x)), col = "steelblue",
5   lwd = 2)
```

Více grafů v jednom obrázku

V prostředí R můžeme grafy kombinovat užitím grafických parametrů `mfrow` a `mfcol`.

Stačí zadat vektor, který určuje počet řádků a počet sloupců, které plánujeme vytvořit.

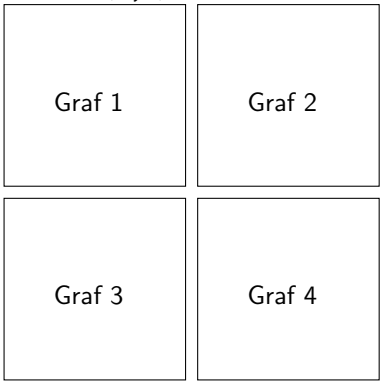
Rozhodnutí o tom, který parametr grafu použijeme, závisí na tom, jak chceme mít grafy uspořádané:

- `mfrow` grafy budou uspořádány po řádcích,
- `mfcol` grafy budou uspořádány po sloupcích.

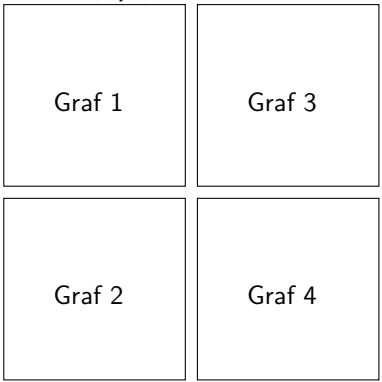
Toto nastavení se používá jako argument funkce `par()`, která upravuje parametry grafického zařízení.

Více grafů v jednom grafu

`mfrow=c(2,2)`



`mcol=c(2,2)`



Více grafů v jednom grafu–ilustrace

```
1  set.seed(5)
2  x <- rexp(80)
3  # Dva radky, dva sloupce
4  par(mfrow = c(2, 2))
5  # Grafy
6  hist(x, main = "Histogram") # Vlevo nahore
7  boxplot(x, main = "Box plot") # Vpravo nahore
8  plot(x, main = "Scatter plot") # Vlevo dole
9  pie(table(round(x)), main = "Kolacovy graf") # Vpravo dole
10 # Zpet na puvodni graficke zarizeni
11 par(mfrow = c(1, 1))
```

Více grafů v jednom grafu - složitější struktura

Často potřebujeme vytvořit obraz se složitější strukturou.

V takových situacích musíme použít funkci `layout()`. Tato funkce přijímá čtyři důležité argumenty:

- `mat` matrix, kde každá hodnota představuje umístění obrázků.
- `widths` vektor pro šířky sloupců. Můžeme je také zadat v centimetrech pomocí funkce `lcm()`.
- `heights` vektor pro výšku sloupců. Můžeme je také zadat v centimetrech pomocí funkce `lcm()`.
- `respect` logická hodnota nebo matice vyplněná 0 a 1 stejných rozměrů jako `mat` pro určení, zda se mají respektovat vztahy mezi šířkami a výškami.

Více grafů v jednom grafu - složitější struktura

Před přidáním grafů můžeme zobrazit náhled rozvržení pomocí funkce `layout.show()`.

```
l <- layout(matrix(c(1, 2, 2, # Prvni, druhy,
                     3, 3, 4), # treti a ctvrty graf
            nrow=2,
            ncol = 3,
            byrow = TRUE))
layout.show(l)
```

Více grafů v jednom grafu - složitější struktura

Tuto metodu ilustrujeme na bodovém grafu implementovaném s okraji ve formě histogramu a krabicového grafu.

```
l <- layout(matrix(c(2, 0, 1, 3),
                   nrow = 2, ncol = 2,
                   byrow = TRUE),
            widths = c(9, 3),
            heights  = c(3, 9), respect = TRUE)
plot(x, main = "Scatter plot")
hist(x, main = "Histogram")
boxplot(x, main = "Box plot")
```

Statistika a programování v R

VI. Výběrové charakteristiky

Průměr

Pro výpočet průměru musíme použít `as.numeric()`, protože `cars[1,]` poskytuje hodnoty ve formátu seznamu.

Pro získání průměrné hodnoty měsíčního počtu nově registrovaných osobních automobilů (v tisících) v letech 2017-18 tedy použijeme kód:

```
1 cars<-read.csv2("macrostat.csv",header=FALSE,sep=";")
2 mean(as.numeric(cars[1,]))
3 [1] 8.090125
```

Průměr

Často se stává, že hodnoty statistického znaku, který nás zajímá, jsou uspořádány v posloupnosti absolutních četností.

V tomto případě upravíme vztah (??) pro výpočet výběrového průměru do tvaru:

$$\overline{x} = \frac{x_1 \cdot n_1 + x_2 \cdot n_2 + \cdots + x_k \cdot n_k}{n_1 + n_2 + \cdots + n_k} = \frac{\sum_{i=1}^{k} x_i \cdot n_i}{\sum_{i=1}^{k} n_i}. \tag{2}$$

kde x_i představují hodnoty proměnné a n_i jejich absolutní četnosti.

Průměr

V tomto případě musíme definovat vlastní funkci pro výpočet střední hodnoty.

Jako vstupní hodnoty zadáme dva vektory. První vektor obsahuje hodnoty, kterých náhodná veličina nabývá, a druhý je vektor jejich četností.

Před provedením výpočtu podle vztahu (2) je nutno ověřit, zda mají oba vektory stejnou délku.

Průměr

Odpovídající funkci `mean2()` pak lze definovat takto:

```
1  mean2<-function(arg1,arg2){
2      if (length(arg1)==length(arg2)){
3          s<-sum(arg1*arg2)/sum(arg2)
4      }
5      else{s<-c("Arguments␣are␣not␣of␣equal␣length")}
6      return(s)
7  }
```

Průměr

Použití právě definované funkce `mean2()` můžeme ilustrovat na proměnné, která nabývá hodnot z množiny $\{1, 2, \ldots, 10\}$.

Absolutní četnosti těchto hodnot můžeme generovat pomocí Poissonova rozdělení.

Možné hodnoty jsou uloženy ve vektoru `a` a jejich absolutní četnosti ve vektoru `b`.

```
1  a<-c(1,2,3,4,5,6,7,8,9,10)
2  b<-rpois(10,20)
3  mean2(a,b)
4  5.38613861386139
```

Medián

K určení mediánu je v prostředí R implementována funkce `median()`.

Můžeme tedy jednoduše zjistit medián nově registrovaných osobních automobilů během měsíce pomocí kódu

```
1 > median(as.numeric(cars[1,]))
2 [1] 8.2425
```

Kvantily

Medián, definovaný v předchozím oddíle, rozděluje vzorek na dvě stejně pravděpodobné podmnožiny.

Obecně můžeme vzorek rozdělit na libovolný počet q stejně pravděpodobných částí. Tyto hodnoty se nazývají q-kvantily a k-tý q-kvantil pro náhodnou veličinu X je určen vztahem

$$\mathbb{P}(X < x) \leq \frac{k}{q}. \tag{3}$$

Kvantily

Pro určeníní kvantilů je v R implementována funkce `quantile()`. Bez zadání volitelných parametrů dává minimum vzorku, první kvartil, medián, třetí kvartil a maximum vzorku.

Můžeme si to ilustrovat na datech COVID-19, stažených z oficiálních stránek slovenské vlády https://korona.gov.sk.

```
1  data<-read.csv("https://mapa.covid.chat/export/csv",
2     header=T,sep=";")
3  > quantile(data[,4])
4     0%    25%    50%    75%   100%
5      0     30    232   1737  15278
6  >
```

Kvantily

Můžeme také nastavit některé volitelné argumenty funkce `quantile()`:

- `probs` číselný vektor pravděpodobností s hodnotami v $\langle 0, 1 \rangle$, který definuje úrovně pravděpodobnosti pro požadované kvantily,
- `na.rm` logická hodnota, je-li TRUE, jsou z `data` před výpočtem kvantilů odstraněny všechny NA a NaN,
- `names` logická hodnota, je-li TRUE, má výsledek atribut names. Pro zrychlení při mnoha `probs` nastavte na `FALSE`.

Kvantily

Ilustrujeme to na určení decilů denních přírůstků

```
> quantile(data[,4],probs=seq(0,1,by=0.1))
    0%   10%   20%   30%   40%   50%   60%   70%   80%   90%  100%
     0     6    20    43    91   232   642  1293  2034  3041 15278
>
```

Variační rozpětí

Variační rozpětí je v prostředí jazyka R výstupem funkce `range()` I Jejím výstupem jsou dvě hodnoty - největší a nejmenší hodnota ve vzorku.

Abychom mohli vyjádřit rozsah variace jako jedinou hodnotu podle definice (**??**), použijeme funkce `max()` a `min()`.

Variační rozpětí

Ilustrace zdrojového kódu

```
1  > x<-c(5,10,12,4,16,8,9)
2  > range(x)
3  [1]  4 16
4  > R<-max(x)-min(x)
5  > R
6  [1] 12
7  >
```

Mezikvartilové rozpětí

V jazyce R je implementováno jako funkce `IQR()`

```
1  > x<-c(5,10,12,4,16,8,9)
2  > IQR(x)
3  [1] 4.5
4  >
```

Střední absolutní odchylka

V jazyce R je implementována jako funkce `mad()`

```
1  > x<-c(5,10,12,4,16,8,9)
2  > mad(x)
3  [1] 4.4478
4  >
```

Rozptyl a standardní odchylka

Funkce `var()` a `sd()` musíme používat opatrně.

Jejich výsledkem je nestranný odhad rozptylu a směrodatné odchylky celé populace.

Chceme-li vypočítat výběrový rozptyl podle vztahu (**??**), musíme definovat vlastní funkci, což ilustrujeme v následujícím zdrojovém kódu.

Rozptyl a standardní odchylka

```
> variance<-function(x) sum((x-mean(x))^2)/length(x)
> stdev<-function(x) sqrt(variance(x))
> variance(x)
[1] 14.40816
> stdev(x)
[1] 3.795809
> var(x) # porovnejte vysledky
[1] 16.80952
> sd(x)
[1] 4.099942
```

Variační koeficient

Variační koeficient je statistická míra relativního rozptylu datových bodů ve vztahu k průměru.

Variační koeficient CV je definován jako poměr směrodatné odchylky s k průměru $\overline{x}$.

$$CV = \frac{s}{\overline{x}}. \tag{4}$$

Variační koeficient se často vyjadřuje v procentech.

Variační koeficient

Variační koeficient není v jazyce R implementován jako funkcie

Lze ho určit pomocí známých funkcí nebo definovat funkci vlastní

```
1 > cv<-function(x) variance(x)/mean(x) * 100
2 > cv(x)
3 [1] 157.5893
```

Šikmost

Šikmost je mírou asymetrie rozdělení nebo souboru dat.

Šikmost γ_1 definujeme jako

$$\gamma_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^3}{s^3}. \tag{5}$$

Šikmost a špičatost

Pro výpočet šikmosti a špičatosti v R potřebujeme balíček `moments`.

V tomto balíčku jsou definovány funkce `skewness()` a `kurtosis()`.

```
> library(moments)
> skewness(x)
[1] 0.3598295
> kurtosis(x)
[1] 2.252963
>
```

Statistika a programování v R

VII. Odhady parametrů

Bodové odhady

Metody

V tomto kurzu představíme dvě metody konstrukce bodových odhadů:

- metoda momentů,
- metoda maximální pravděpodobnosti.

Předpokládejme, že máme vzorek $X_1, \ldots, X_n$ z rozdělení, který závisí na vektoru parametrů $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$.

Intervaly spolehlivosti

Example

Předpokládejme, že se uskutečnil průzkum na 250 náhodně vybraných lidech, aby se zjistilo zda vlastní tablet. Z 250 dotázaných 98 uvedlo, že vlastní tablet. Pomocí 95 % hladiny spolehlivosti vypočtěte odhad intervalu spolehlivosti pro skutečný podíl lidí, kteří vlastní tablet.

Intervaly spolehlivosti

Řešení: Nejprve vypočítáme nevychýlený bodový odhad pravděpodobnosti p jako $\hat{p} = \frac{98}{250}$ a položíme $\hat{q} = 1 - \hat{p}$.

Nyní můžeme vypočítat hranice intervalu spolehlivosti pomocí funkce `qnorm()`.

Intervaly spolehlivosti

```
> n<-250
> p<-98/n
> q<-1-p
> c<-qnorm((1+alpha)/2,0,1)
> lower.bound<-p-c*sqrt(p*q/n)
> upper.bound<-p+c*sqrt(p*q/n)
> print(c(lower.bound,upper.bound))
[1] 0.3314836 0.4525164
```

Získali jsme tedy 95 % interval spolehlivosti $(0,3315; 0,4525)$ pro podíl lidí vlastnících tablet.

Děkuji za pozornost.

Statistika a programování v R

Aleš Kozubík