Programování v R

2.3 Konverze různých typů objektů

Při psaní programů je velmi často třeba konvertovat jeden typ objektu na druhý. V R existuje řada funkcí pro tento účel:

as.numeric (x)

as.character (x)

as.expression (x)

as.matrix (x)

as.data.frame (x)

data.matrix (x)

Posledně uvedené tři funkce konvertují objekt `x` na matici nebo data frame. O něco přívětivější možnost převedení data framu na matici představuje funkce `data.matrix` , která konvertuje všechny proměnné data framu `x` na numerické a pak je spojí jako sloupce jediné matice.

is.numeric (x)

is.character (x)

is.logical (x)

is.matrix (x)

is.data.frame (x)

Testují typ daného objektu `x`

2.4 Vstup/výstup

print (x)

Zobrazí hodnotu vybraného objektu jazyka R.

cat (…, file="", sep="")

Funkce `cat()` zobrazí jeden nebo více objektů výrazně primitivnějším způsobem než `print()`, umožňuje však mnohem větší kontrolu nad způsobem zobrazování. Tato funkce sama o sobě neodřádkovává, pro nový řádek je potřeba explicitně uvést řetězec „`\n`“. Např.

```
> x<-5.8
> cat("Vysledek je ",x," N/m. \n")
Vysledek je 5.8 N/m.
```

Této funkce lze použít i pro zapisování dat do souboru na disk — viz dokumentace k R.

readline ()

Používá se ke vstupu dat z klávesnice.

```
> x<-readline("Zadej hodnotu x:\n")
Zadej hodnotu x:
5.8
> x
[1] "5.8"
```

Z tohoto příkladu je jasné, že vstup dat je vždy ve formě textového řetězce, a že hodnota numerické proměnné musí být případně konvertována následující funkcí:

```
> x<-as.numeric(x)
> x
[1] 5.8
```

data.entry (x)

Vyvolá zabudovaný editor, který lze použít pro vložení/editaci dat v jednoduchém spreadsheetu. Příkaz může být volán s parametrem x, specifikujícím objekt určený k editaci.

2.5 Podmíněné příkazy

Podmíněné provádění příkazů se provádí pomocí funkce:

if (*podmínka*) *výraz1* else *výraz2*

Pokud je splněna `podmínka`, provádí se instrukce ve `výrazu1`, v opačném případě ve `výrazu2`. Složitější výrazy vyžadují použití složených závorek:

```
if (x>2 & y<1){
    print(x)
    print(y)
}else{
    cat("x<=2 nebo y>=1!\n")
}
```

2.6 Příkazy cyklu

I když v R— na rozdíl od jazyků BASIC nebo PASCAL — se tomu lze obvykle vyhnout, je možné i zde používat příkazů cyklu.

for (*proměnná* in *výraz1*) *výraz2*

`Výraz2` se provádí pro hodnoty `proměnné` postupně definované pomocí `výrazu1`. Složitější výrazy je třeba uzavřít do složených závorek.

Kupříkladu kód:

```
for (f in (1:5)){
   cat("Druha mocnina",f)
   cat(" je",f^2,"\n")
}
```

Dává výstup:

```
Druha mocnina 1 je 1
Druha mocnina 2 je 4
Druha mocnina 3 je 9
Druha mocnina 4 je 16
Druha mocnina 5 je 25
```

while (*podmínka*) *výraz*

`výraz` se bude provádět dokud je splněna `podmínka`

repeat (*podmínka*) výraz

Poznámka: snažte se příkazům cyklu pokud možno vyhýbat — jejich provádění v R znatelně zpomaluje provádění programu.

Cvičení 2.4

o pomocí cyklu a funkce `par(mfrow=)` napište krátký program, který najednou zobrazí šest Harkerových diagramů (binárních diagramů SiO_2 vs.další oxidy hlavních prvků) podle Vaší volby. Použijte souboru s daty pro sázavskou suitu. Nezapomeňte na správný popis os.

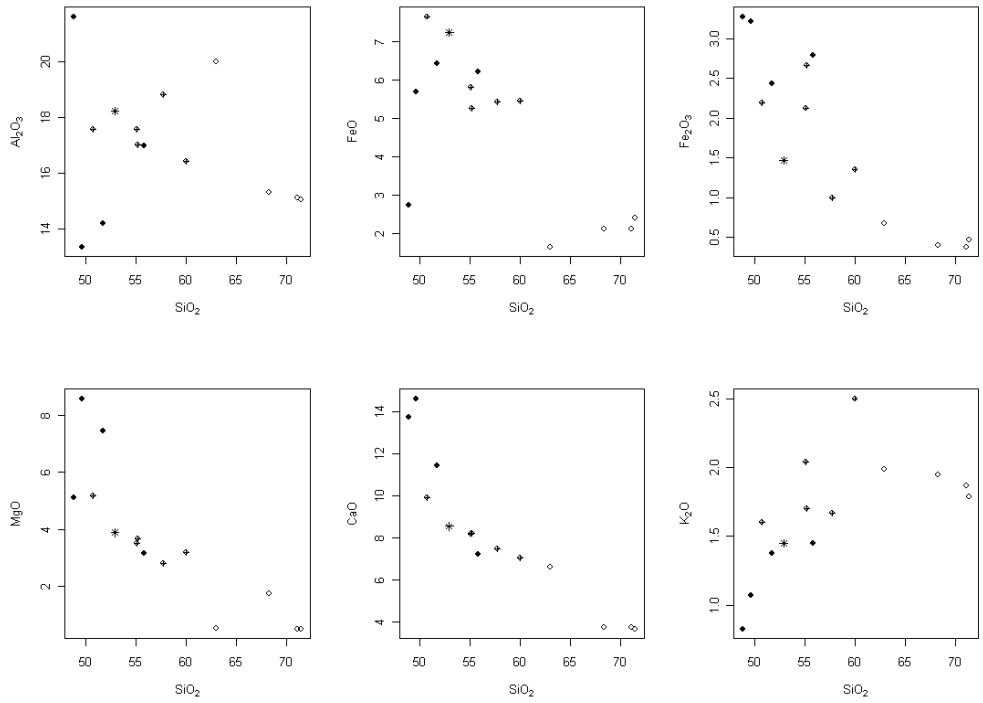
```
> WR<-read.table("sazava.data",sep="\t")
> WR<-as.matrix(WR[,-1])
> windows()
> par(mfrow=c(2,3))
> ee<-c("Al2O3","FeO","Fe2O3","MgO","CaO","K2O")
> for (f in ee){
>   plot(WR[,"SiO2"],WR[,f],xlab="SiO2",ylab=f,pch=WR[,"Symbol"])
> }
```

Nyní můžeme napsat o něco sofistikovanější formu programu, která bude využívat možnosti anotací pomocí funkce `expression`. Nejprve musíme ale znát, že konverze character vektoru x na výraz (angl. *expression*) se provádí funkcí `parse`:

```
> parse(text=as.expression(x))
```

and then the modified part of the code will be:

```
> lab<-c("Al[2]*O[3]","FeO","Fe[2]*O[3]","MgO","CaO","K[2]*O")
> for (f in 1:length(ee)){
>  plot(WR[,"SiO2"],WR[,ee[f]],xlab=expression(SiO[2]),
     ylab=parse(text=as.expression(lab[f])),pch=WR[,"Symbol"])
> }
> # Obr. 2.8
```



Obr. 2.8. Vybrané Harkerovy diagramy pro sázavskou suitu (Cvičení 2.4)

2.7 Uživatelské funkce

jméno*<- function (*argument1, argument2, …*) *výraz

Posledním příkazem v těle definice uživatelské funkce by měl být `return(x)`, s argumentem upřesňujícím jméno proměnné, kterou má funkce vracet. Jinak se bere ta, jíž byla naposledy přiřazena nějaká hodnota.

Příklad použití — funkce pro výpočet směrodatné odchylky (podle vzorce $S = \sqrt{\frac{\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2}{n}}$):

```
odchylka<-function(x){
      z<-sqrt(sum((x-mean(x))^2)/length(x))
      return(z)   # funkce vrací hodnotu proměnné z
      }
```

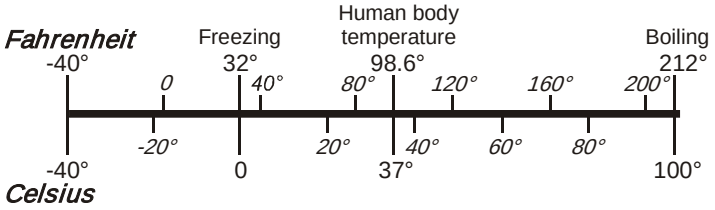
Důležité je si uvědomit, že proměnné použité ve funkci (v našem příkladu `x` a `z`) jsou lokální, tedy mimo ni neznámé.

Pokud načteme do paměti soubor `sazava.data`, můžeme pomocí funkce `odchylka()` spočítat třeba směrodatnou odchylku pro SiO_2:

```
>  WR<-read.table("sazava.data",sep="\t")
>  WR<-as.matrix(WR[,-1])
> odchylka(WR[,"SiO2"])
[1] 7.462887
```

Cvičení 2.5 *V anglosaských zemích se teploty stále hojně vyjadřují ve stupních Fahrenheita.*

- navrhněte funkci, která bude konvertovat stupně Fahrenheita na stupně Celsia
- vyzkoušejte si, jak funkce pracuje, na předcházejícím diagramu — kolik je ve stupních Celsia –40, 98.6 a 212 °F?
- vypište konverzní tabulku Fahrenheit → Celsius pro 0, 10, …220 °F a zobrazte data graficky

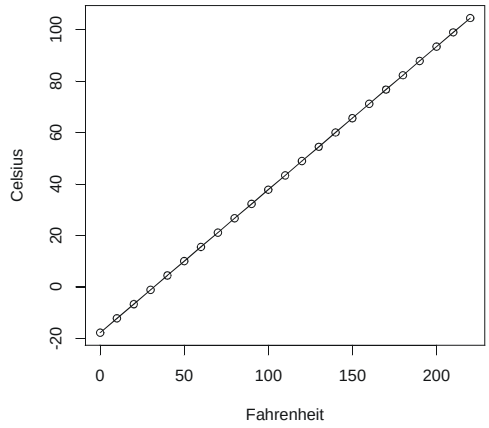


Obr. 2.9 Graficky vyjádřený vztah mezi Fahrenheitovou a Celsiovou stupnicí

```
Fahrenheit2Celsius<-function(x){
      a<-100/(212-32)
      b<-100-a*212
      z<-a*x+b
return(z)
}
```

```
> Fahrenheit2Celsius(c(-40,98.6,212))
[1] -40  37 100
```

```
> x<-seq(0,220,by=10)
> y<-Fahrenheit2Celsius(x)
> names(y)<-x
```



Obr. 2.10 Konverze Fahrenheit → Celsius

```
> y
         0         10         20         30         40         50         60
-17.777778 -12.222222  -6.666667  -1.111111   4.444444  10.000000  15.555556
        70         80         90        100        110        120        130
 21.111111  26.666667  32.222222  37.777778  43.333333  48.888889  54.444444
       140        150        160        170        180        190        200
 60.000000  65.555556  71.111111  76.666667  82.222222  87.777778  93.333333
       210        220
 98.888889 104.444444

> plot(x,y,xlab="Fahrenheit",ylab="Celsius",type="o")
> # Obr. 2.10
```

2.7.1 Argumenty funkcí

Existují dva způsoby, jak specifikovat parametry funkce jazyka R. Buď je lze uvádět v pořadí stejném, jakém byly definovány, když byla funkce navržena. Nebo je možno používat pojmenované parametry ve formě "`jméno.argumentu = hodnota`", a ty lze pak uvádět v náhodném pořadí.

Při psaní uživatelských funkcí lze uvádět implicitní hodnoty, jako v následujícím případě:

```
> my.plot<-function(x,y,pch="+",col="red"){
> …tělo funkce…
> }
```

a potom může být taková funkce volána řadou způsobů, například:

```
> my.plot(x,y)            # plots data as red crosses
> my.plot(x,y,"o")        # plots data as red circles
> my.plot(x,y,col="blue") # plots data as blue crosses
```

pak je také zřejmé, že příkaz:

```
> my.plot(x,y,"blue")
```

nebude fungovat správně. Dobré je si uvědomit, že v R má většina argumentů funkcí přiřazeny nějaké rozumné implicitní hodnoty. Ty obvykle stačí pro většinu aplikací a běžný uživatel nemusí o jejich existenci vůbec vědět. Když je třeba, příkaz `args` vypíše všechny argumenty dané funkce:

args (jméno_funkce)

kde `jméno_funkce` odkazuje na existující funkci, například:

```
> args("my.plot")
function (x, y, pch = "+", col = "red")
```

2.7.2 Přiřazení ve funkcích

Důležité je si uvědomit, že všechny proměnné definované v rámci uživatelské funkce (v příkladu funkce pro výpočet směrodatné odchylky to byly `x` a `z`) jsou lokální. To znamená, že všechna přiřazení jsou jen přechodné povahy, a po opuštění funkce jsou ztracena. Proto taky je třeba rozlišovat proměnné, byť se stejným jménem, ve funkci a v prostředí, z kterého byla funkce volána.
Ve (vzácných) případech kdy je třeba měnit hodnotu globální proměnné, lze použít operátor "`<<-`":

```
> x<<-"Hello"
```