Wprowadzenie do języków Octave i R

autor Łukasz Stafiniak

Pomoc

- Pomoc na temat funkcji / zagadnienia

Octave	R

`> help nazwa`	`> help ("nazwa")`
	`> help (nazwa)`
	`> ?nazwa`

- Kategorie pomocy (lista pakietów i funkcji)

Octave	R

`> help`	`> library() #dostępne pakiety`
	`> search()  #załadowane pakiety`
	`> help(package="nazwa")`

- Przeszukiwanie pomocy

Octave	R

tylko nagłówki pomocy:
`> lookfor "klucz"`
cała pomoc:
`> lookfor -all "klucz"`
	`> help.search("klucz")`

Octave: znajdź w podręczniku	R: przekieruj pomoc do przeglądarki
`> doc "nazwa"`	`> options(browser="firefox")` `> help.start()`

- Przykładowe zastosowanie funkcji

Octave	R
`> example nazwa` `> example ('nazwa')`	`> example(nazwa)`

- Dostępne zmienne/obiekty

Octave	R

krótko:
`> who`
z opisem:
`> whos`
	`> objects()`
	`> ls()`

- W **Octave** dodając średnik blokujemy wydrukowanie rezultatu operacji. (Pisząc skrypt lub funkcję możemy monitorować ich działanie, a gdy wszystko jest OK, dodać wszędzie średniki.)
- Komentarz rozpoczyna się od znaku `#` (w Octave też od znaku `%`).

Podstawowe typy wartości

- W **Octave** mamy trzy podstawowe struktury danych: macierze, tablice dowolnych elementów (cell arrays) i struktury z notacją kropkową. Liczby, wektory i stringi funkcjonują jako macierze odpowiedniego wymiaru.
  - Wartości logiczne, znaki alfanum., liczby rzeczywiste i zespolone tworzą jeden typ danych. (Wektor ze znakiem alfanum. jest wyświetlany jako string.)
- **R** jest mocno typizowanym językiem. Posiada wektory, macierze, typ grupujący "factors", listy (odpowiednik cell arrays), "ramki danych" data frames: macierze, których kolumny mogą być różnych typów (np. łącząc typy wyliczeniowe i liczbowe), funkcje (jako wartości, jak w programowaniu funkcyjnym). Stringi są wektorami.
  - R rozróżnia typy, tzw. "mody": liczbowy, liczb zespolonych, logiczny, znaków alfanum., "raw" (danych binarnych); `mode(x)` zwraca mod danej wartości, np. `complex` dla wektora liczb zespolonych
  - obiekty w R mają też swoje klasy (`class(x)`) (niektóre funkcje działają odpowiednio do klasy obiektu)

- W Octave i R stringi ograniczamy bądź znakiem ", bądź '
- Specjalna wartość `NA` (not available) oznacza brakujące wartości. W Octave dostępna jest tylko dla typu liczbowego.
- W **R**, `is.typ(x)` sprawdza czy `x` jest typu `typ`, `as.typ(x)` konwertuje `x` do typu `typ`
- W **R** argumenty funkcji mają nazwy, których możemy użyć w czasie wywoływania: `f(nazwa=wartość)`

Podstawowe konstrukcje sterujące

- W **Octave** niektóre konstrukcje wymagają umieszczenia w jednej linii (np. przy wprowadzaniu macierzy koniec linii jest równoznaczny z `;`), możemy kontynuować linię dalej kończąc ją znakiem `\`
- Octave zaleca używać słów kluczowych `endif`, `endwhile`, `endfor` etc., ale akceptuje też `end`, jedyne akceptowane przez Matlaba.

Octave	R

if-else-end w osobnych liniach
statements mogą być wieloliniowe
`if cond`
`  statements1`
`else`
`  statements2`
`endif`
wersja jednolinijkowa
`if cond, statm1; else statm2; end`
	składnia jak w C:
	aby zgrupować instrukcje
	użyj `{...}`
	`if (cond) statement1`
	`else statement2`

`while cond`
`  statements`
`endwhile`
	`while (cond) statement`

- W Octave pętla `for` przebiega po kolumnach macierzy, a w R po kolejnych elementach wektora/macierzy.
  Standardowa pętla to `for i=1:n`/`for (i in 1:n)`.

Octave	R

`> s = 0;`
`> for i = [1,3;2,4]`
`  s = s + i;`
`endfor`
`> s`
`s =`
`  4`
`  6`
	`> s <- 0`
	`> y <- matrix (1:4, ncol=2, nrow=2)`
	`> for (i in y) s <- s+i`
	`> s`
	`[1] 10`

Funkcje, biblioteki, zapisywanie

- W **Octave** wystarczy dodać ścieżkę katalogu z plikami `.m` aby odpowiadające im funkcje lub skrypty były widoczne: `addpath('/do/katalogu')`. Widoczne są też pliki z katalogu bieżącego.
- W **R** ładujemy zainstalowany pakiet poprzez `library('nazwa')`.

Octave: instalowanie pakietu np. z Octave-forge
`> pkg install ~/Desktop/statistics-1.0.5.tar.gz`
R: downloadowanie i instalowanie pakietu z CRAN
`> install.packages('UsingR')`

- W **Octave** aby użyć zmiennej wewnątrz funkcji nie przekazując jej jako argumentu, musimy zarówno na zewnątrz, jak i w środku funkcji zadeklarować ją jako globalną: `global x`.

- W **Octave** mamy pliki skryptowe i pliki funkcyjne, obydwa z rozszerzeniem `.m`. Pierwszy blok komentarzy w pliku jest udostępniany przez komendę `help`. Plik skryptowy to dowolny ciąg poleceń Octave, odpalany jak komenda. Plik funkcyjny zaczyna się od polecenia `function` (OK też w trybie interakt.):

Octave: plik `dotprod.m`

```
function xydot = dotprod (x, y)

# usage: f (x, t)
#
# This function computes the dot product
# of vectors x and y.

xydot = x * y' ;

endfunction
```

W Octave zaleca się kończyć funkcje przez `endfunction`, w Matlabie nie kończy się funkcji. Nazwa pierwszej funkcji w pliku musi pokrywać się z nazwą pliku, pozostałe funkcje są lokalne (widoczne tylko w danym pliku).

- W **R** plikom skryptowym dajemy rozszerzenie `.R` i uruchamiamy funkcją `source`. (Stan środowiska o rozszerzeniu **.RData** wczytuje `load`.)

- W **R** funkcje, jak wszystkie inne wartości, przypisujemy nazwom:

R

```
> bslash <- function(X, y) {
   X <- qr(X)
   qr.coef(X, y)
  }
```

Nazwy ujęte w `%` są operatorami infiksowymi, np. `'%!%' <- function (X, y) {...}` i potem użycie `X %!% y`.

- Przed rozpoczęciem pracy, stwórz katalog dla projektu. Uruchom **R** w tym katalogu. Zakończ pracę komendą `q()`, pozwala zapisać stan sesji w bieżącym katalogu.

- W **Octave** mamy do dyspozycji uchwyty funkcji:

Octave

```
> f = @sin;
> feval (f, pi/4)
```

oraz funkcje anonimowe (w R mamy je "za darmo" bo funkcje to wartości):

Octave
`> quad (@(x) x.^2, 0, 10)`

Wektory, macierze, struktury

- W **Octave** macierze wprowadzamy w nawiasach kwadratowych `[1,2,3;4,5,6]` oddzielając elementy wiersza przecinkiem a wiersze średnikiem. W **R** polecenie `c` tworzy wektor konkatenując podane elementy, `matrix` wypełnia tworzoną macierz elementami z (np. wektora) danych kolumna po kolumnie (lub dla `byrow=TRUE`, wiersz po wierszu). W Octave składnia `[a:b]` jest zawsze dopuszczalnym wariantem składni `a:b`.

Sprawdź następujące przykłady:

Octave	R
`> x = [0.1, 0.2, 0.3];`	`> x <- c(0.1, 0.2, 0.3)`
`> [x, x]`	`> c(x, x)`
`> [x; x]`	`> matrix(c(x,x), 2, 3, byrow=TRUE)`
`> 1:5`	`> 1:5`
`> 1:pi:9`	`> seq(1, 9, by=pi)`
`> linspace(0, 5, 9)`	`> seq(0, 5, length=9)`

- W Octave macierze (w R tylko wektory) mają automatycznie dostosowywany rozmiar; wektory i macierze są indeksowane od 1.

Octave	R

`> x = 5;`
`> x(2) = 7;`
`> x(2,2) = 3`
`x =`
`5 7`
`0 3`
	`> x <- 5`
	`> x[3] <- 7`
	`> x`
	`[1]  5 NA  7`

- Przewymiarowywanie i konkatenacja macierzy. W R macierzami (matrices) nazywa się tylko tablice (arrays) dwuwymiarowe. `matrix` i `array` w R "recykluje" wartości jeśli ich brakuje w wektorze.

Octave	R

`> A = reshape (1:12, [2,2,3])`
`> A(2,1,3)`
`ans =`
`  10`
	`> A <- array(1:7,c(2,2,3))`
	`> A(2,1,3)`
	`[1] 3`

`> [1,2,3;4,5,6]'`	`> t(matrix(1:6,2,3))`
`> B = ((1:5)+1)'`	`> B <- as.matrix(1:5+1)`
`> [B-1,B]`	`> cbind(B-1,B)`
`> [B'-1; B']`	`> rbind(t(B)-1,t(B))`
`> A(:)  #spłaszczenie macierzy`	`> c(A)`

- Funkcje/operacje liczbowe i logiczne na wektorach i macierzach w R są zazwyczaj dokonywane składowa-po-składowej.

Octave	R
`> sin(B)`	`> sin(B)`
`> B .* B`	`> B * B`
`> B' * B`	`> t(B) %*% B`
`> B ./ B`	`> B / B`
`> (B * B') / B'`	
`> A\b`	`> solve(A,b)`
`(1:5)' * (1:5)`	`> 1:5 %o% 1:5`

- Przykłady generowania stringów

Octave	R
`> B(1:10,1)='X'` `> B(1:10,2)=char([1:10]+96)`	`paste(c('X','Y'),1:10,sep='')`

- Octave i R pozwalają na **indeksowanie wektorów i macierzy** przy pomocy:
  - wektora/macierzy z wartościami logicznymi: prawda oznacza "dołącz element z tej pozycji", fałsz oznacza "omiń"
  - wektora liczb całkowitych dodatnich oznaczających pozycje, oznacza "dołącz elementy z wymienionych pozycji, w tym porządku"
  - "operatora" selekcji całego wymiaru (wiersza, kolumny)

Octave	R
`> B(:,1)`	`> B[,1]`
`> B(2,:)`	`> B[2,]`

R pozwala:

  - użyć ujemnych liczb całkowitych na oznaczenie pozycji, które chcemy wykluczyć (np. `x[-(1:5)]` – z pominięciem poz. 1-5)
  - użyć stringów, jeśli pozycje wektora/macierzy są nazwane, np.

R

```
> fruit <- c(5, 10, 1, 20)
> names(fruit) <- c('orange','banana','apple','peach')
> lunch <- fruit[c('apple','orange')]
```

- specjalny mechanizm grupowania pozwala podzielić elementy wektora na klasy i działać osobno na tych klasach: gdy `x` to wektor klas przyporządkowujący pozycje klasom, a `y` to wartości liczbowe poszczególnych pozycji,

R: `x` – klasy obserwacji, `z` – wartości obserwacji	
`> fx = factor(x)` `> m = tapply (z, fx, mean)`	średnie wartości dla klas

`levels(fx)` zwraca wektor klas. `by` działa na całych data.frames. `tapply` i `by` zwracają listę wyników indeksowaną klasami.

- Wektory i macierze można modyfikować na zaindeksowanych pozycjach przy pomocy operatora przypisania, np.
`y(y<0) = 0`, odpow. `y[y<0] <- 0`

  - prawą stroną może być wektor/macierz odpowiadający rozmiarem wektorowi/macierzy powstałemu przez wybranie wskazanych pozycji, np. `y[y<0] <- -y[y<0]` albo `A(2:2:6,:)=y(1:2:5,:)`

- W Octave są "tablice komórkowe" cell arrays, w R są listy. Pętla `for` iteruje je tak samo jak macierze (Octave) / wektory (R).

Octave	R

`> A = {1,'alfa';'beta',2}`
`A =`
`{`
`  [1,1] = 1`
`  [2,1] = beta`
`  [1,2] = alfa`
`  [2,2] = 2`
`}`
`> A{1,2}`
`ans = alfa`
`> A{3,3} = 3  #A{3,1}=[]`
	`> A <- list(1,'alfa','beta',2)`
	`[[1]]`
	`[1] 1`
	
	`[[2]]`
	`[1] 'alfa'`
	`...`
	`> A[[3]]`
	`[1] 'beta'`
	`> A[[6]] <- 3  #A[[5]]=NULL`

- Octave ma obiekty typu "data structure", różne od macierzy/tablic. W R elementy list mogą mieć nazwy (ale ciągle są pozycyjne); indeksowanie "wektorowe" zwraca listę jedno-elementową. (Kropka jest częścią nazw (jak litery) w **R** – zamiast niej używa się $.)

Octave	R

`> x.a = 1`
`> x.b = [1,3;2,4]`
`> x.c = 'string'`
`x =`
`{`
`  a = 1`
`  b =`
`  1  3`
`  2  4`
`  c = string`
`}`
	`> x <- list(a=1,b=matrix(1:4,2,2))`
	`> x$c <- 'string'`
	`> x`
	`$a`
	`[1] 1`
	`$b`
	`  [,1] [,2]`
	`[1,]  1  3`
	`...`
	`> x[[3]]`
	`[1] 'string'`
	`> x[3]`
	`$c`
	`[1] 'string'`

- W **R** można skracać nazwę pozycji tak długo, dopóki jest jednoznaczna, np. `x$covariance` do `x$cov`.
- Konkatenacja "wektorowa" `c` skleja listy (również z nazwami; przy indeksowaniu brane jest pierwsze wystąpienie danej nazwy).
- W **R** można "przyczepić" listę z nazwami, udostępniając te nazwy w środowisku: `attach(x)`/`detach(x)`.
- W **Octave** `for` może iterować po parach klucz-wartość:

Octave

```
for [val, key] = expression
  statements
endfor
```

- Specjalną klasą list w **R** są "ramki danych" (data frames) (tworzenie: `data.frame(nazwa=wartość,...)`), np. dodanie do ramki kolumny ze stringami skonwertuje ją automatycznie do typu kategorycznego "factors".

Wczytywanie danych

- W **Octave** funkcje `load` i `save` pozwalają przechowywać dane w różnych formatach. Standardowo używamy rozszerzenia `.mat`. Nazwy zmiennych też zostaną zapamiętane i odtworzone.

Octave
`> A = [1:3; 4:6; 7:9];` `> save myfile.mat A`

`> load myfile.mat`
`> A`
`A = ...`

Składnia: `save [opcje] plik v1 v2 ...` gdzie opcje to:

  - `-ascii`: zapisz pojedynczą macierz 2d "czystym tekstem" bez żadnych meta-danych
  - `-binary`: zapisz w binarnym formacie Octave
  - `-text`: zapisz w tekstowym formacie Octave (czytelny, rozwlekły)
  - `-mat7-binary`, `-mat-binary`, `-mat4-bin`: format binarny Matlaba 7, 6, 4
  - `-hdf5`: binarny format HDF5

- Octave ma znane z C funkcje `fprintf` i `fscanf` do "ręcznej" obsługi plików
- **R** ma `read.table` do wczytywania danych stablicowanych
  - domyślny separator rozdziela pola białymi znakami, argument `sep` podaje zbiór separatorów, warianty `read.csv` (`sep=','`) rozdziela przecinkiem, `read.csv2` rozdziela średnikiem (przecinek w roli "kropki dziesiętnej")
  - pierwszy wiersz zawiera nazwy zmiennych (kolumn)
  - pierwszy wiersz jest o jeden krótszy od pozostałych, których pierwsza kolumna zawiera nazwy wierszy, chyba że podamy `header=TRUE`, wtedy nazwy wierszy są domyślne
  - możemy podać `header=FALSE` i ręcznie określić nazwy zmiennych/kolumn w `col.names`
  - podając wektor `colClasses` możemy wymusić typ zmiennej/kolumny; `NA` daje domyślny typ, więc możemy podać tylko wybrane kolumny (wstaw pola `NA`, bo wektor jest recyklowany)
    - ciekawymi "niedomyślnymi" typami są `'NULL'` – pomiń kolumnę, `'Data'`, `'POSIXct'`
  - kolumny znakowe są domyślnie konwertowane do typu kategorycznego "factors" (można to zmienić argumentem-wektorem logicznym

`as.is`)

- stringi mogą być wydzielane przy pomocy separatorów `quote` (domyślnie `quote='"'`)
- plik może zawierać komentarze, domyślnie po znaku `comment.char='#'`

R

```
> HousePrice <- read.table('house.data', header=TRUE)
```

- **R** jest dystrybuowane z wieloma zbiorami danych, które można wczytać komendą `data`

R

```
> data()  #wylistuj tylko z pakietu datasets
> data(package = .packages(all.available = TRUE))
> data('iris')
> attach(iris)
> tapply(Sepal.Length, Species, mean)
> iris2 <- iris; iris2$Species <- NULL
> by(iris2, Species, summary)
```

- zwykle dane zostaną wczytane do jednej zmiennej typu date.frame o nazwie takiej jak nazwa zbioru danych

- `data` rozpoznaje pliki `.R`/`.r` (wczytuje przez `source(...)`), `.RData`/`.rda` (wczytuje przez `load(...)`), `.tab`/`.txt`/`.TXT` (wczytuje przez `read.table(..., header=TRUE)`), `.csv`/`.CSV` (wczytuje przez `read.table(..., header=TRUE, sep=';')`)