Pisanie funkcji w R

Chociaż napisaliśmy już różne funkcje w R, w tej części do pisania funkcji R będziemy podchodzić systematycznie.

Ogólne

Dobrym sposobem na poznanie funkcji lub napisanie nowej funkcji jest przyjrzenie się istniejącym. Jako przykład rozważmy, że chcielibyśmy napisać funkcję implementującą nową procedurę kreślenia. Więc zaczynamy od spojrzenia na istniejącą funkcję plot.

> plot ↲

function (x, y, …)

UseMethod("plot")

<bytecode: 0x000000000ee0c900>

<environment: namespace:graphics>

Nie jest to zbyt pomocne, dlatego podajemy instrukcję:

```
> methods(plot) ↲
  [1] plot.acf*              plot.data.frame*   plot.decomposed.ts*
  [4] plot.default           plot.dendrogram*   plot.density
  [7] plot.ecdf              plot.factor*       plot.formula*
 [10] plot.function          plot.hclust*       plot.histogram*
 [13] plot.HoltWinters*      plot.isoreg*       plot.lm
 [16] plot.medpolish*        plot.mlm plot.     ppr*
 [19] plot.prcomp*           plot.princomp*     plot.profile.nls*
 [22] plot.spec              plot.stepfun       plot.stl*
 [25] plot.table*            plot.ts            plot.tskernel*
 [28] plot.TukeyHSD
```

Funkcje niewidoczne są oznaczone gwiazdką

Jeśli zdecydujemy się przyjrzeć plot.default, możemy to zrobić przez

> plot.default ↲

function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL, log = "",

main = NULL, sub = NULL, xlab = NULL, ylab = NULL, ann = par("ann"),

axes = TRUE, frame.plot = axes, panel.first = NULL, panel.last = NULL,

asp = NA, …)

{

Widoczne są wiersze kodu

}

Ponieważ nasza nowa metoda kreślenia jest ukierunkowana na dane kategoryczne, decydujemy się raczej przyjrzeć plot.factor. Ale jest to funkcja oznaczona gwiazdką i dlatego nie jest widoczna:

```
> plot.factor

Error: object 'plot.factor' not found
```

Funkcje oznaczone gwiazdką można sprawdzić przy użyciu następującej metody:

```
> getAnywhere(plot.factor)

Znaleziono pojedynczy obiekt pasujący do 'plot.factor' .Znaleziono go w następujących miejscach zarejestrowano metodę S3 dla wykresu z grafiki przestrzeni nazw

namespace:graphics

with value

function (x, y, legend.text = NULL, …)

{

if (missing(y) || is.factor(y)) {

dargs <- list(…)

axisnames <- if (!is.null(dargs$axes))

dargs$axes

else if (!is.null(dargs$xaxt))

dargs$xaxt != "n"

else TRUE

}

if (missing(y)) {

barplot(table(x), axisnames = axisnames, …)

}

else if (is.factor(y)) {

if (is.null(legend.text))

spineplot(x, y, …)

else {

args <- c(list(x = x, y = y), list(…))

args$yaxlabels <- legend.text

do.call("spineplot", args)

}

}

else if (is.numeric(y))
```

boxplot(y ~ x, …)

else NextMethod("plot")

}

a) Jak są przypisywane wartości domyślne do argumentów funkcji?

b) Jakie jest domyślne zachowanie plot.factor()?

c) Jakie zadania można osiągnąć za pomocą pmatch() i co rozumie się przez częściowe dopasowanie? Co się stanie, jeśli plot.factor() zostanie wywołany z

(i) legend.text = 'AA=Agecat';

(ii) leg = 'AA=Agecat'? Explain.

d) Omów użycie funkcji missing().

e) Podaj przykład użycia funkcji stop(message= " ").

f) Podaj przykład użycia funkcji warning(message= " ").

g) Jakie jest zastosowanie funkcji warnings()?

h) Dlaczego można wywoływać funkcje bez podawania argumentów, np. q()?

i) Jeśli ciało funkcji składa się tylko z jednej instrukcji, nie jest konieczne ujmowanie jej w nawiasy klamrowe.

j) Konwencja jest taka, że jako wartość zwracaną przez funkcję używa się ostatniej ocenionej instrukcji. Jeśli kilka przedmiotów ma zostać zwróconych, zbierz je na liście.

k) Funkcja return() z pojedynczym obiektem lub listą obiektów jest przydatna do przerwania funkcji na pewnym pośrednim etapie i zwrócenia obiektu lub listy obiektów na tym konkretnym etapie. Zwykle robi się to, gdy funkcja jest w fazie rozwoju.

l) Czasami nie ma sensownej wartości do zwrócenia, np. kiedy funkcja jest napisana głównie po to, aby stworzyć jakiś wykres. W takich przypadkach funkcja invisible() może być użyta jako ostatnia instrukcja funkcji. Jako przykład użycia invisible() podaj następujące instrukcje:

> boxplot(rnorm(100), plot = TRUE)

> boxplot(rnorm(100), plot = FALSE)

Teraz spójrz na koniec funkcji boxplot.default(), aby zobaczyć, jak zaimplementowano invisible().

m) Biblioteki (pakiety) funkcji R. Dołączanie i odłączanie bibliotek do ścieżki wyszukiwania.

n) Tworzenie nowej funkcji za pomocą skryptów lub fix().

o) Edycja istniejącej funkcji za pomocą skryptów lub fix().

p) Zwróć uwagę, że podczas pisania funkcji wiersz można przerwać w dowolnym miejscu i kontynuować w kolejnym wierszu. Ostrzeżenie: Uważaj, aby nie umieścić punktu przerwania w miejscu, w którym oznacza on zakończenie instrukcji wykonywalnej. Wyjaśnić.

Pisanie nowej funkcji

Wyznaczanie indeksów elementów wektora lub macierzy spełniających określony warunek: funkcja where()

a) Napisz następującą funkcję za pomocą polecenia

> fix(where)

function(x, cond)

{ # Argument cond must evaluate to a logical value

if(!is.matrix(x))

seq(along = x)[cond]

else matrix(c(row(x)[cond], col(x)[cond]), ncol = 2)

}

b) Sprawdź zestaw danych o jakości powietrza za pomocą polecenia str(airquality).

c) Użyj funkcji where(), aby znaleźć wskaźniki (i) NA, (ii) wartości maksymalnej i (iii) wartości minimalnej w zbiorze danych o jakości powietrza.

d) Powtórz (c) używając wbudowanej funkcji which().

Sprawdzanie kolizji nazw obiektów

a) Co się stanie, jeśli obiekt R otrzyma taką samą nazwę jak istniejący obiekt?

b) Omów użycie funkcji apropos(), conflict(), find() i match() do nazewnictwa obiektów.

c) Pamiętaj, że gdy funkcja jest wywoływana, Oceniający najpierw szuka w środowisku globalnym funkcji o tej nazwie, a następnie w każdym z dołączonych pakietów lub baz dat w kolejności pokazanej przez search(). Ewaluator zazwyczaj przerywa wyszukiwanie, gdy nazwisko znajduje się po raz pierwszy. Jeśli dwa dołączone pakiety mają funkcje o tej samej nazwie, jeden z nich zamaskuje obiekt w drugim. Na przykład funkcja gam() istnieje w dwóch pakietach, gam i mgcv. Jeśli oba były dołączone, zwróci polecenie find("gam")

[1] "pakiet:gam" "pakiet:mgcv".

d) Operator :: może być użyty do uzyskania dostępu do zamierzonej wersji gam() za pomocą wywołania mgcv::gam() lub gam::gam().

e) Podczas pisania pakietów R przestrzeń nazw pakietu zapewnia inny mechanizm zapewniający, że używana jest poprawna wersja funkcji. Zauważ w związku z tym, że operator ::: może być użyty do uzyskania dostępu do obiektów, które nie są eksportowane.

Zmienne lokalne i środowiska ewaluacyjne

a) Gdzie jest przechowywany obiekt utworzony przez skrypt lub fix()?

b) Gdzie są przechowywane lokalne obiekty (obiekty, które są tworzone podczas wykonywania funkcji)?

c) Wyjaśnij, jak działa środowisko ewaluacyjne.

d) Co rozumie środowisko globalne?

e) Zapoznaj się z plikiem pomocy R w.r.t. operator <<-. Kiedy warto używać tego operatora? Jakie są niebezpieczeństwa związane z tym operatorem?

f) Co rozumie się przez zakres wyrażenia lub funkcji? Symbole występujące w ciele funkcji można podzielić na trzy klasy: parametry formalne, zmienne lokalne i zmienne swobodne. Formalne parametry funkcji to te, które pojawiają się w nawiasach oznaczających listę argumentów funkcji. Ich wartości są określane przez proces wiązania rzeczywistych argumentów funkcji z parametrami formalnymi. Zmienne lokalne są tworzone przez ocenę wyrażeń w treści funkcji. Zmienne, które nie są parametrami formalnymi ani zmienne lokalne nie są nazywane zmiennymi wolnymi. Wolne zmienne stają się zmiennymi lokalnymi, gdy są do nich przypisane. Rozważ poniższą definicję funkcji.

```
fun <- function(datvec) {

mean <- mean(datvec)

print(mean)

plot(datvec)

plot(Traffic)

}
```

W tej funkcji datvec jest parametrem formalnym, obiekt oznaczający po lewej stronie symbolu przypisania jest zmienną lokalną (nie mylić z funkcją mean() po prawej stronie symbolu przypisania), podczas gdy Traffic jest wolną zmienną. W języku R wolne powiązania zmiennych są rozwiązywane przez sprawdzenie najpierw środowiska, w którym utworzono funkcję. Nazywa się to zakresem leksykalnym. Jeśli następujące wywołanie funkcji zostanie wykonane z zachęty w katalogu roboczym fun(1:25), formalnemu parametrowi datvec w ciele funkcji przypisywana jest wartość 1:25 (rzeczywisty argument), a jego średnia jest przypisywana do lokalnego średnia obiektu. Jeśli wolny parametr Traffic zostanie znaleziony w środowisku globalnym lub w bazie danych na ścieżce wyszukiwania, zostanie utworzony wymagany graf, w przeciwnym razie do konsoli zostanie wysłany komunikat o błędzie.

Czyszczenie

a) Przestudiuj, jak używana jest funkcja on.exit(). Ta funkcja może służyć do resetowania opcji, które zostały zmienione podczas sesji R, do ich pierwotnych wartości po zakończeniu sesji lub zakończeniu funkcji z komunikatem o błędzie. Jest to również wygodne do usuwania plików tymczasowych.

b) Przestudiuj zastosowania funkcji .First() i .Last().

c) Napisz funkcję, która automatycznie otwiera okno wykresu z kwadratowym obszarem wykresu po rozpoczęciu sesji R.

Zmienna liczba argumentów: argument …

a) Rozważ następującą sytuację: Chcesz napisać funkcję do złożonego zadania. Na określonym etapie ma powstać wykres niektórych wyników pośrednich. Wymaga to, aby funkcja wywołująca zawierała wywołanie funkcji hist. Oto przykład fragmentu kodu do wykonania tego zadania:

```
complexfun <- function(datmat,colgraph)

{ datmat <- scale(datmat)

# Several lines of complex code here
```

hist(datmat, col = colgraph) }

Wywołanie takie jak complexfun(rnorm(1000), 'yellow') może teraz zostać wykonane w celu uzyskania pożądanego rezultatu. Problem polega na tym, że funkcja hist ma kilka argumentów, do których chciałbyś mieć dostęp, przekazując im odpowiednie rzeczywiste wartości za pomocą funkcji wywołującej complexfun. Zamiast uciekać się do dostarczenia kompletnego zestawu argumentów na liście argumentów complexfun, R zapewnia zgrabny sposób rozwiązania tej sytuacji: Argument … który działa jak każdy inny formalny argument, z wyjątkiem tego, że może reprezentować zmienną liczbę argumentów. Aby zobaczyć, jak działa argument … , zmień powyższą funkcję na:

complexfun2 <- function(datmat, …)

{ datmat <- scale(datmat)

Several lines of complex code here

hist(datmat, …) }

Argumenty reprezentowane przez argument … na liście argumentów funkcji hist są przekazywane do hist poprzez argument … występujący na liście argumentów funkcji complexfun2 :

> complexfun2(datmat = rnorm(1000), col = 'yellow',

probability = TRUE, xlim = c(-5,5))

b) Napisz funkcję, która pobierze maksymalną długość dowolnego z nieokreślonej liczby argumentów określonego trybu. Oto kolejna ilustracja użycia argument …:

maxlen <- function (mode.use="numeric", …)

{ my.list <- list(…)

out <- 0

for(x in my.list)

if(mode(x) == mode.use) out <- max(out,length(x))

out

}

Zauważ, że nazwany argument musi być określony jako taki w wywołaniu funkcji:

> maxlen(1:10, 1:15, 1:3, letters)

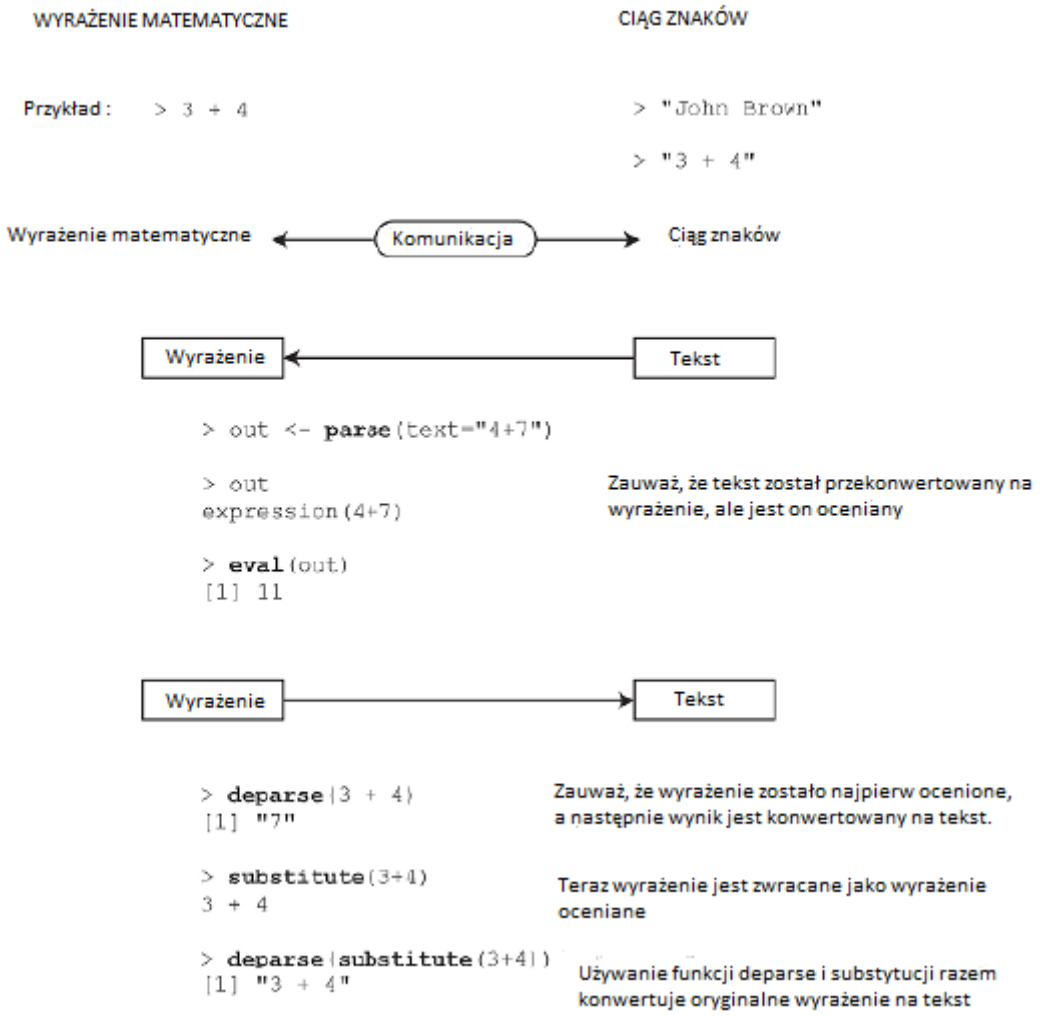
> maxlen(mode.use="numeric", 1:10, 1:15, 1:3, letters)

> maxlen(1:10, 1:15, 1:3, letters, mode.use="character")

> maxlen(mode.use="character", 1:10, 1:15, 1:3, letters)

Pobieranie nazw argumentów: Funkcje deparse() i replace()

Istnieje wiele praktycznych sytuacji wymagających konwersji wyrażeń matematycznych na ciągi znaków (tekst) lub odwrotnie, wymagających konwersji tekstu na wyrażenia matematyczne. Narzędzia (funkcje) dostarczone w R do osiągnięcia takich konwersji są podsumowane na rysunku



• Zadanie: napisz funkcję R, która wykreśli dwa wektory, używając jako etykiet osi nazw obiektów przekazanych jako argumenty do funkcji, a także używając tych nazw w głównym tytule wykresu.

Z rysunku wynika, że funkcja replace() przyjmuje wyrażenie jako argument i zwraca je bez wartości. Aby ocenić wartość zwracaną przez replace(), należy użyć funkcji eval(). Funkcja deparse() przyjmuje jako argument nieocenione wyrażenie i konwertuje je na ciąg znaków. Teraz jesteśmy gotowi do napisania następującej funkcji:

```
> labplot <- function (x,y)
{ xname <- deparse(substitute(x))
yname <- deparse(substitute(y))
plot(x,y, xlab=xname, ylab=yname, main = paste("Plot of",
yname,"versus", xname))
}
```

a) Przestudiuj i zilustruj użycie funkcji labplot() .

b) Z rysunku wynika również, że funkcja parse() działa odwrotnie niż deparse(), przekształcając ciąg znaków w wyrażenie nieocenione. To ostatnie nieocenione wyrażenie może być ocenione w razie potrzeby za pomocą funkcji eval().

Operatory

Wykonaj następującą instrukcję

> objects('package:base')[1:31]

w celu uzyskania przykładów operatorów dostępnych w R.

a) Operatory są specjalnymi funkcjami R. Omów to stwierdzenie. Pod jakimi względami operatory różnią się od zwykłych funkcji R?

b) Napisz operator %E%, aby określić odległość euklidesową między dwoma wektorami i podaj przykład jego użycia. Podpowiedź: przy tworzeniu operatorów z poprawką lub przy użyciu skryptów nazwę należy podać jako ciąg znaków np. fix("%E%").

Funkcje zastępcze

Wykonaj następującą instrukcję

> objects('package:base')

i zauważ, że niektóre nazwy obiektów pojawiają się w parach z nazwą jednego członka pary kończącą się na '<-' . Przykładami są „dim<-”, „levels<-”, „diag<-”, „names<-”, „rownames<-”, „colnames<-” i „dimnames<-”. Funkcje o nazwach kończących się na '<-' nazywane są funkcjami zastępczymi. Funkcja zastępująca pojawia się po lewej stronie symbolu przypisania używając nazwy bez znaku '<-' w celu zastąpienia zawartości obiektów pojawiających się na jego liście argumentów zawartością obiektu pojawiającego się po prawej stronie przypisania symbol np.:

> a <- rownames(X) # Function rownames in action.

> rownames(X) <- 1:nrow(X) # Replacement function 'rownames<-'

in action.

Jak można skontrolować obiekt „diag<-” i czy różni się on od obiektu „diag”? Porównaj wyniki następujących wywołań funkcji:

> getAnywhere('diag')

> getAnywhere('diag<-')

Pod jakimi względami funkcje zastępcze różnią się od innych funkcji? Aby napisać funkcję zastępującą, muszą być spełnione następujące zasady:

ja. nazwa funkcji musi kończyć się <-

ii. funkcja musi zwrócić cały obiekt z wprowadzonymi odpowiednimi zmianami

iii. końcowy argument funkcji odpowiadający danym zastępczym po prawej stronie przypisania musi mieć nazwę value

iv. zwykle istnieje funkcja towarzysząca o tej samej nazwie bez znaku „<-”.

Jako przykład, napisz funkcję zastępującą undefined(), która zastąpi brakujące wartości w obiekcie danych wartościami po jego prawej stronie:

"undefined<-" <- function (x, codes = numeric(), value)

{ if (length(codes) > 0) x[x %in% codes] <- NA

x[is.na(x)] <- value

x

}

Powyższą funkcję można utworzyć lub edytować za pomocą fix("undefined<-"). Zilustruj użycie undefined().

Wartości domyślne i leniwa ocena

a) Funkcja match.arg() jest przydatna do wybierania wartości domyślnej z jednej z zestawu możliwych wartości. Rozważmy następujący przykład:

> choice <- function(method=c("PCA","CVA","CA","NONLIN"))

{ match.arg(method) }

> choice()

[1] "PCA"

> choice("CVA")

[1] "CVA"

> choice("xx")

Error in match.arg(method) :

'arg' should be one of "PCA", "CVA", "CA", "NONLIN"

b) Funkcje w języku S podlegają zasadzie zwanej leniwą oceną, co oznacza, że wartość domyślna nie jest oceniana, dopóki nie jest rzeczywiście potrzebna w treści funkcji. W wyniku leniwej oceny w wywołaniu funkcji może się zdarzyć, że niektóre wartości domyślne nigdy nie zostaną ocenione.

Dynamiczne ładowanie procedur zewnętrznych

Skompilowany kod może działać w niektórych przypadkach znacznie szybciej niż odpowiedni kod w języku R. Funkcje .C() i .Fortran() umożliwiają użytkownikom korzystanie z programów napisanych w C lub Fortran w ich funkcjach R. Jak to się robi, zilustrowano poniżej. Przestudiuj uważnie ten przykład i w razie potrzeby zapoznaj się z plikami pomocy, aby uzyskać więcej informacji. Najpierw tworzona jest funkcja R, aby obliczyć iloczyn macierzy dwóch macierzy:

```
> matmult <- function (A,B)
{ if(ncol(A) != nrow(B)) stop("A and B not conformable with
respect to matrix multiplication \n")
n <- nrow(A)
q <- ncol(B)
Cmat <- matrix(NA, nrow=n, ncol=q)
for(i in 1:n)
{ for(j in 1:q) Cmat[i,j] <- sum(A[i,] * B[,j])
}
Cmat
}
```

Następnie napisany jest podprogram Fortran do wykonywania mnożenia macierzy. Kod Fortran dla tego podprogramu podano poniżej:

```
SUBROUTINE MATM (A1, A2B1, B2, A, B, OUT)
C This subroutine performs matrix multiplication.
C This should be improved with optimized code (such as
C from Linpack, etc.)
IMPLICIT NONE
INTEGER A1, A2B1, B2
DOUBLE PRECISION A(A1,A2B1), B(A2B1,B2), OUT(A1,B2)
C DUMMIES
INTEGER I, J, K
DO 300,J=1,B2
DO 200,I=1,A1
OUT(I,J)=0
DO 100,K=1,A2B1
OUT(I,J)=OUT(I,J)+A(I,K)*B(K,J)
100 CONTINUE
200 CONTINUE
300 CONTINUE
END
```

Następnie z podprogramu Fortran tworzona jest biblioteka dołączana dynamicznie (.dll). Najprostszym sposobem na to jest użycie polecenia R CMD SHLIB matm.f z powłoki DOS. Biblioteka dll jest dostępna jako C:\ matm32.dll (dla maszyn 32-bitowych) lub C:\matm64.dll (dla maszyn 64-bitowych). Teraz należy napisać funkcję R, w której wywoływany jest kod Fortran:

```
> matmult.Fortran <-function (A,B)

{ if(ncol(A) != nrow(B)) stop("A and B not conformable with

respect to matrix multiplication \n")

n <- nrow(A)

q <- ncol(B)

p <- ncol(A)

Cmat <- matrix(0, nrow=n, ncol=q)

storage.mode(A) <- "double"

storage.mode(B) <- "double"

storage.mode(Cmat) <- "double"

value <- .Fortran("matm", as.integer(n), as.integer(p),

as.integer(q), A, B, matprod=Cmat)

value$matprod }
```

Aby użyć matmult.Fortran(), należy załadować poprawną bibliotekę dll do katalogu roboczego za pomocą funkcji dyn.load():

```
> dyn.load("full path\\matm64.dll")

# Change 64 to 32 for 32bit machine
```

Porównaj odpowiedzi i czas wykonania matmult() i matmult.Fortran() dla macierzy o różnych rozmiarach.