محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه دوم

حسین باغیشنی

دانشگاه شاهرود

عملگرهای شرطی

بعضی موارد باید به کامپیوتر گفته شود که تصمیم بگیرد کدام عمل را انجام دهد:

$$|x| = \begin{cases} x & x \geq ۰ \\ -x & x < ۰ \end{cases}$$

یا

$$\psi(x) = \begin{cases} x^۲ & |x| \leq ۱ \\ ۲|x| - ۱ & |x| > ۱ \end{cases}$$

عملگرهای شرطی

بعضی موارد باید به کامپیوتر گفته شود که تصمیم بگیرد کدام عمل را انجام دهد:

$$|x| = \begin{cases} x & x \geq ۰ \\ -x & x < ۰ \end{cases}$$

یا

$$\psi(x) = \begin{cases} x^۲ & |x| \leq ۱ \\ ۲|x| - ۱ & |x| > ۱ \end{cases}$$

تمرین: تابع $\psi(x)$ را در R رسم کنید

ساده‌ترین عملگر شرطی: if

```
if (x >= 0) {
    x
} else {
   -x
}
```

شرط موجود در if نیازمند برگرداندن مقدار $TRUE$ یا $FALSE$ است.

ساده‌ترین عملگر شرطی: if

```
if (x >= 0) {
    x
} else {
   -x
}
```

شرط موجود در if نیازمند برگرداندن مقدار $TRUE$ یا $FALSE$ است.

شرط $else$ اختیاری است.

تمام مقادیر عددی معتبر به جز ۰، $TRUE$ محسوب می‌شوند و ۰، $FALSE$ محسوب می‌شود. اکثر مقادیر غیرعددی مسدود می‌شود (خطا خواهد داد):

```
> if(1) {"Truth!"} else {"Falsehood!"}
[1] "Truth!"
> if(-1) {"Truth!"} else {"Falsehood!"}
[1] "Truth!"
> if(0) {"Truth!"} else {"Falsehood!"}
[1] "Falsehood!"
> if("TRUE") {"Truth!"} else {"Falsehood!"}
[1] "Truth!"
> if("TRUTH") {"Truth!"} else {"Falsehood!"}
Error in if ("TRUTH") { : argument is not interpretable as logical
> if("c") {"Truth!"} else {"Falsehood!"}
Error in if ("c") { : argument is not interpretable as logical
> if(NULL) {"Truth!"} else {"Falsehood!"}
Error in if (NULL) { : argument is of length zero
> if(NA) {"Truth!"} else {"Falsehood!"}
Error in if (NA) { : missing value where TRUE/FALSE needed
> if(NaN) {"Truth!"} else {"Falsehood!"}
Error in if (NaN) { : argument is not interpretable as logical
```

عملگرهای منطقی: تکی یا دوگانه؟

عملگرهای منطقی & و | مانند عملگرهای حسابی هستند، یعنی بر روی بردارها عنصر عنصر عمل می‌کنند:

```
> c(TRUE,TRUE) & c(TRUE,FALSE)
[1] TRUE FALSE
```

برای کنترل حلقه‌های برنامه، معمولاً برنامه نیاز به مقادیر بولی تکی دارد. به این معنی که نیاز داریم به کامپیوتر بگوییم چیزی که نیاز نداریم را محاسبه نکن.

عملگرهای منطقی: تکی یا دوگانه؟

عملگرهای منطقی & و | مانند عملگرهای حسابی هستند، یعنی بر روی بردارها عنصر عنصر عمل می‌کنند:

```
> c(TRUE,TRUE) & c(TRUE,FALSE)
[1] TRUE FALSE
```

برای کنترل حلقه‌های برنامه، معمولاً برنامه نیاز به مقادیر بولی تکی دارد. به این معنی که نیاز داریم به کامپیوتر بگوییم چیزی که نیاز نداریم را محاسبه نکن.

راه حل استفاده از && و || است: از چپ به راست حرکت کن و وقتی به جواب رسیدی متوقف شو.

```
> (0>0) & ("c"+1)
Error in "c" + 1 : non-numeric argument to binary operator
> (0>0) && ("c"+1)
[1] FALSE
```

با مثال بالا، تصور کنید برای مولفه دوم محاسبات پیچیده‌ای نیاز باشد. R از آن صرفنظر می‌کند زیرا نیازی به محاسبه آن نیست!

اگر بر روی بردارها عمل شود، عملگرهای بولی دوتایی اعضای اول هر بردار را برای مقایسه در نظر می‌گیرند:

```
> c(FALSE,FALSE) | c(TRUE,FALSE)
[1] TRUE FALSE
> c(FALSE,FALSE) || c(TRUE,FALSE)
[1] TRUE
> c(FALSE,FALSE) || c(FALSE,TRUE)
[1] FALSE
```

به طور کلی: از && و || برای کنترل برنامه‌ها استفاده کنید و سعی کنید ورودی آن‌ها آرگومان‌های برداری نباشند

عملگرهای شرطی آشیانه‌ای

عملگرهای شرطی می‌توانند به صورت آشیانه‌ای به کار روند:

```
if (x^2 < 1) {
   x^2
} else {
   if (x >= 0) {
   x
   } else {
   -x
   }
}
```

به جای عمل خسته‌کننده آشیانه‌ای کردن عملگرهای $if/else$، می‌توان از $switch$ استفاده کرد (چگونه؟)

تکرار: انجام چیزهایی مشابه برای چندین بار

تکرار عملی یکسان (یا خیلی مشابه) به چندین بار مشخص:

```
n <- 10
table.of.logarithms <- vector(length=n)
table.of.logarithms
for (i in 1:n) {
   table.of.logarithms[i] <- log(i)
}
table.of.logarithms
```

حلقه for یک شمارنده را (در اینجا i) به کمک مقادیر یک بردار (در اینجا n : ۱) افزایش می‌دهد و بدنه حلقه را مادامی که کل بردار اجرا شود، تکرار می‌کند

تکرار: انجام چیزهایی مشابه برای چندین بار

تکرار عملی یکسان (یا خیلی مشابه) به چندین بار مشخص:

```
n <- 10
table.of.logarithms <- vector(length=n)
table.of.logarithms
for (i in 1:n) {
   table.of.logarithms[i] <- log(i)
}
table.of.logarithms
```

حلقه for یک شمارنده را (در اینجا i) به کمک مقادیر یک بردار (در اینجا n : ۱) افزایش می‌دهد و بدنه حلقه را مادامی که کل بردار اجرا شود، تکرار می‌کند

توجه: روشی بهتر و کارآمدتر برای انجام این وظیفه وجود دارد.

ترکیب *for* و *if*

```
x <- c(-5,7,-8,0)
y <- vector(length=length(x))
for (i in 1:length(x)) {
   if (x[i] >= 0) {
      y[i] <- x[i]
   } else {
      y[i] <- -x[i]
   }
}
y # now c(5,7,8,0)
```

توجه: روشی بهتر و کارآمدتر برای انجام این وظیفه وجود دارد.

while: تکرار شرطی

```
while (x > (1+1e-06)) {
   x <- sqrt(x)
}
```

شرط موجود در *while* باید مانند *if* یک مقدار بولی تکی $TRUE/FALSE$ باشد.

while: تکرار شرطی

```
while (x > (1+1e-06)) {
   x <- sqrt(x)
}
```

شرط موجود در *while* باید مانند *if* یک مقدار بولی تکی $TRUE/FALSE$ باشد.

حلقه مادامی که ورودی شرط $FALSE$ شود، اجرا می‌شود.

- اگر شرط همیشه $TRUE$ باشد، حلقه تا ابد تکرار می‌شود.

while: تکرار شرطی

```
while (x > (1+1e-06)) {
   x <- sqrt(x)
}
```

شرط موجود در *while* باید مانند *if* یک مقدار بولی تکی $TRUE/FALSE$ باشد.

حلقه مادامی که ورودی شرط $FALSE$ شود، اجرا می‌شود.

- اگر شرط همیشه $TRUE$ باشد، حلقه تا ابد تکرار می‌شود.
- هیچ وقت شروع نمی‌شود، مگر این که شرط با $TRUE$ آغاز شود.

تمرین: چطور یک حلقه *for* را با *while* جایگزین می‌کنید؟

تکرارهای غیرشرطی

```
repeat {
   print("Help! I am hba, trapped in an endless loop!")
}
```

یا مفیدتر از آن به صورت زیر است:

```
repeat {
   if (watched) {
     next()
   }
   print("Help! I am Dr. Morris Culpepper, trapped in an endless loo
   if (rescued) {
     break()
   }
}
```

همیشه حداقل یک بار وارد حلقه می‌شویم، حتی اگر $rescued$ برابر $TRUE$ باشد.

تکرارهای غیرشرطی

```
repeat {
   print("Help! I am hba, trapped in an endless loop!")
}
```

یا مفیدتر از آن به صورت زیر است:

```
repeat {
   if (watched) {
     next()
   }
   print("Help! I am Dr. Morris Culpepper, trapped in an endless loo
   if (rescued) {
     break()
   }
}
```

همیشه حداقل یک بار وارد حلقه می‌شویم، حتی اگر $rescued$ برابر $TRUE$ باشد.
$break()$ ما را از حلقه خارج می‌کند و $next()$ از مابقی بدنه (در اینجا if) صرفنظر می‌کند و به حلقه برمی‌گردد

تخصیص منابع: دوباره

یادآوری:

- تولید اتومبیل و کامیون توسط کارخانه با فولاد و ساعت کار کارگران
- منابع موجود شامل ۱۶۰۰ ساعت کار و ۷۰ تن فولاد با تولید ۱۰ اتومبیل و ۲۰ کامیون به طور کامل به کار گرفته می‌شوند
- مساله به طور دقیق با برنامه‌ریزی خطی حل می‌شود
- فرض کنید چیزی از جبر خطی نمی‌دانیم و مهم نیست چقدر از منابع، مادامی که یک حد اولیه موجود باشد، مصرف شود
- یافتن جواب با در نظر گرفتن یک طرح اولیه شروع شده و با بسط آن مادامی که قیدها رخ دهند، به دست می‌آید

```
factory <- matrix(c(40,1,60,3),nrow=2,
    dimnames=list(c("labor","steel"),c("cars","trucks")))
available <- c(1600,70); names(available) <- rownames(factory)
slack <- c(8,1); names(slack) <- rownames(factory)
output <-c(30,20); names(output) <- colnames(factory)
```

چطور عمل می‌کند؟

```
passes <- 0 # How many times have we been around the loop?
repeat {
passes <- passes + 1
needed <- factory %*% output # What do we need for that output level?
# If we're not using too much, and are within the slack, we're done
if (all(needed <= available) &&
all((available - needed) <= slack)) {
break()
}
# If we're using too much of everything, cut back by 10%
if (all(needed > available)) {
output <- output * 0.9
next()
}
# If we're using too little of everything, increase by 10%
if (all(needed < available)) {
output <- output * 1.1
next()
}
# If we're using too much of some resources but not others, randomly
# tweak the plan by up to 10%
output <- output * (1+runif(length(output),min=-0.1,max=0.1))
}
```

خروجی بعد از شروع با ۳۰ اتومبیل و ۲۰ کامیون:

```
> round(output,1)
cars trucks
10.4 19.7
> round(needed,1)
[,1]
labor 1596.1
steel 69.4
> passes
[1] 3452
```

یعنی کد نوشته‌شده، طرح اولیه را ۳۴۵۲ بار تصحیح می‌کند و نتیجه را با حالت واقعی ۱۰ اتومبیل و ۲۰ کامیون مقایسه کنید.

پرهیز از تکرارسازی

نرم‌افزار R روش‌های متفاوتی برای فرار از تکرارسازی دارد که مبتنی بر عمل کردن بر روی کل اشیاء می‌باشند، به طوری که

- به طور مفهومی ساده‌ترند

پرهیز از تکرارسازی

نرم‌افزار R روش‌های متفاوتی برای فرار از تکرارسازی دارد که مبتنی بر عمل کردن بر روی کل اشیاء می‌باشند، به طوری که

- به طور مفهومی ساده‌ترند
- کدهای ساده‌تری خواهند داشت

پرهیز از تکرارسازی

نرم‌افزار R روش‌های متفاوتی برای فرار از تکرارسازی دارد که مبتنی بر عمل کردن بر روی کل اشیاء می‌باشند، به طوری که

- به طور مفهومی ساده‌ترند
- کدهای ساده‌تری خواهند داشت
- سریعتر هستند (گاهی کمی سریع و گاهی خیلی خیلی سریعتر)

اکثر این روش‌ها در مورد محاسبات با بردارسازی می‌باشد

مثال

اکثر زبان‌های برنامه‌نویسی چطوری دو بردار a و b را با هم جمع می‌کنند:

```
c <- vector(length(a))
for (i in 1:length(a)) {
   c[i] <- a[i] + b[i]
}
```

و R چطوری دو بردار را با هم جمع می‌کند:

```
c <- a+b
```

مزایا:

- وضوح: علامت + در مورد چیزی است که در حال انجام آن هستیم
- خلاصه‌سازی: این عملگر آنچه را که کامپیوتر انجام می‌دهد، پنهان می‌کند
- کوتاه کردن: یک خط به جای چهار خط
- سرعت

معایب:

- باید یاد بگیرید در مورد کل اشیاء فکر کنید نه قسمت‌هایی از آن‌ها

محاسبات برداری‌شده

بسیاری از توابع به طور خودکار برای بردارسازی تنظیم شده‌اند:

```
abs(x) # Absolute value of each element in x
log(x) # Logarithm of each element in x
```

شرطی‌سازی با $ifelse()$:

```
ifelse(x<0,-x,x) # Pretty much the same as abs(x)
ifelse(x^2>1,abs(x),x^2)
```

تکرارسازی

- $rep(x, n)$: x را n بار تکرار می‌کند

تکرارسازی

- $rep(x, n)$: x را n بار تکرار می‌کند
- $seq()$: یک دنباله تولید می‌کند

آرایه‌ها با ساختارهای تکرارشده

همه ترکیبات مقادیری از بردارها با $expand.grid()$ قابل انجام است:

```
> expand.grid(v1=c("lions","tigers"),v2=c(0.1,1.1))
      v1  v2
1  lions 0.1
2 tigers 0.1
3  lions 1.1
4 tigers 1.1
```

یک ساختار داده تولید می‌کند به طوری که می‌توانید انواع متفاوتی از داده‌ها را با هم ترکیب کنید

آرایه‌ها با ساختارهای تکرارشده

همه ترکیبات مقادیری از بردارها با $expand.grid()$ قابل انجام است:

```
> expand.grid(v1=c("lions","tigers"),v2=c(0.1,1.1))
     v1  v2
1  lions 0.1
2 tigers 0.1
3  lions 1.1
4 tigers 1.1
```

یک ساختار داده تولید می‌کند به طوری که می‌توانید انواع متفاوتی از داده‌ها را با هم ترکیب کنید

بیشتر از یک بردار ورودی مناسب است

ترکیب ورودی‌های یک تابع: *outer*

```
:er(c(1,3,5),c(2,3,7),'*')
     [,1] [,2] [,3]
[1,]    2    3    7
[2,]    6    9   21
[3,]   10   15   35
```

برای عملگرها نیاز به علامت " هست.
البته برای دستور بالا، می‌توان از عملگر مخصوص خود استفاده کرد

```
c(1,3,5) %o% c(2,3,7)
```

با *outer* هر تابع برداری‌شده دو مقداره، قابل کاربرد است:

```
> outer(c(1024,1000),c(2,10),log)
          [,1]   [,2]
[1,] 10.000000 3.0103
[2,]  9.965784 3.0000
```

$replicate()$: یک چیز دقیقا یکسان را چندین بار انجام می‌دهد

```
# Take a sample of size 1000 from the standard exponential
rexp(1000,rate=1)
# Take the mean of such a sample
mean(rexp(1000,rate=1))
# Draw 1000 such samples, and take the mean of each one
replicate(1000,mean(rexp(1000),rate=1))
# Plot the histogram of sample means
hist(replicate(1000,mean(rexp(1000,rate=1))))

# Equivalent to that last, but dumb
sample.means <- vector(length=1000)
for (i in 1:length(sample.means)) {
sample.means[i] <- mean(rexp(1000,rate=1))
}
hist(sample.means)
```