محاسبات آماری پیشرفته

ترم اول سال تحصیلی ۹۳

جلسه سوم

حسین باغیشنی

دانشگاه شاهرود

چرا توابع؟

ساختارهای داده، مقادیر مرتبط با هم را در یک شیء گرد هم می‌آورند

چرا توابع؟

ساختارهای داده، مقادیر مرتبط با هم را در یک شیء گرد هم می‌آورند

توابع، دستورهای مرتبط با هم را در یک شیء به هم پیوند می‌دهند.

```
my.clever.function <- function(an.argument,another.argument) {
    # clever calculations
    return(important.result)
}
```

```
# "Robust" loss function, for outlier-resistant regression
# Inputs: vector of numbers (x)
# Outputs: vector with x^2 for small entries, 2|x|-1 for large ones
psi.1 <- function(x) {
    psi <- ifelse(x^2 > 1, 2*abs(x)-1, x^2)
    return(psi)
}

> z <- c(-0.5,-5,0.9,9)
> psi.1(z)
[1] 0.25 9.00 0.81 17.00
```

به قسمت‌های مختلف تابع نوشته‌شده دقت کنید.

رابط‌ها: ورودی یا آرگومان‌های تابع، خروجی یا مقادیر برگشت‌شده

رابط‌ها: ورودی یا آرگومان‌های تابع، خروجی یا مقادیر برگشت‌شده

در بدنه تابع، توابع دیگری مانند $ifelse()$ و $abs(x)$ و همچنین عملگرهایی مانند $>$ فراخوانی شده‌اند.

رابط‌ها: ورودی یا آرگومان‌های تابع، خروجی یا مقادیر برگشت‌شده

در بدنه تابع، توابع دیگری مانند $ifelse()$ و $abs(x)$ و همچنین عملگرهایی مانند $>$ فراخوانی شده‌اند.

دستور $return()$ نشان‌دهنده خروجی تابع است

رابط‌ها: ورودی یا آرگومان‌های تابع، خروجی یا مقادیر برگشت‌شده

در بدنه تابع، توابع دیگری مانند $ifelse()$ و $abs(x)$ و همچنین عملگرهایی مانند $>$ فراخوانی شده‌اند.

دستور $return()$ نشان‌دهنده خروجی تابع است

توجه: همیشه یک خط توضیح در مورد هدف تابع نوشته‌شده، فهرست کردن آرگومان‌های تابع، و خروجی‌های آن خیلی مفید خواهد بود

آرگومان‌های نام‌گذاری‌شده و پیش‌فرض

```
# "Robust" loss function, for outlier-resistant regression
# Inputs: vector of numbers (x), scale for crossover (c)
# Outputs: vector with x^2 for small entries, 2c|x|-c^2 for large ones
psi.2 <- function(x,c=1) {
psi <- ifelse(x^2 > c^2, 2*c*abs(x)-c^2, x^2)
return(psi)
}
> identical(psi.1(z), psi.2(z,c=1))
[1] TRUE
```

چنانچه نام یک آرگومان تابع در استفاده از آن تابع نباشد، مقادیر پیش‌فرض در نظر گرفته می‌شوند.

```
> identical(psi.2(z,c=1), psi.2(z))
[1] TRUE
```

آرگومان‌های نام‌دارشده، چنانچه به طور واضح برچسب خورده باشد، به هر ترتیب دلخواهی قابل استفاده است

```
> identical(psi.2(x=z,c=2), psi.2(c=2,x=z))
[1] TRUE
```

بررسی آرگومان‌ها

یک مشکل: مشاهده نتایج عجیب زمانی که آرگومان‌های مورد استفاده مورد انتظار نیستند

```
> psi.2(x=z,c=c(1,1,1,10))
[1] 0.25 9.00 0.81 81.00
> psi.2(x=z,c=-1)
[1] 0.25 -11.00 0.81 -19.00
```

پاسخ: در کد نوشته شده چند خط برای چک کردن این گونه آرگومان‌ها بنویسید

```
psi.3 <- function(x,c=1) {
# Scale should be a single positive number
stopifnot(length(c) == 1,c>0)
psi <- ifelse(x^2 > c^2, 2*c*abs(x)-c^2, x^2)
return(psi)
}
```

آرگومان‌های $stopifnot()$ عبارت‌هایی هستند که باید ارزیابی همه آن‌ها $TRUE$ باشد. به محض رخداد اولین $FALSE$ اجرا، همراه با پیغام خطا، متوقف می‌شود.

یک تابع چه می‌تواند ببیند و انجام دهد؟

نام آرگومان‌ها ویژه درون تابع هستند و در یک محیط بزرگتر باطل می‌شوند. در واقع تغییراتی که در درون تابع انجام می‌شوند، به خارج آن سرایت نمی‌کنند.

```
> x <- 7
> y <- c("A","C","G","T","U")
> adder <- function(y) { x<- x+y; return(x) }
> adder(1)
[1] 8
> x
[1] 7
> y
[1] "A" "C" "G" "T" "U"
```

جستجو در محیط R

هر نامی که در تابع تعریف نشده باشد، مقدار آن در محیط، جستجو می‌شود.

جستجو در محیط R

هر نامی که در تابع تعریف نشده باشد، مقدار آن در محیط، جستجو می‌شود.

چیزی که مهم است مقدار این گونه نام‌ها در هنگام فراخوانی تابع است نه هنگام تعریف کردن آن

```
> circle.area <- function(r) { return(pi*r^2) }
> circle.area(c(1,2,3))
[1] 3.141593 12.566371 28.274334
> truepi <- pi
> pi <- 3
> circle.area(c(1,2,3))
[1] 3 12 27
> pi <- truepi # Restore sanity
> circle.area(c(1,2,3))
[1] 3.141593 12.566371 28.274334
```

به رابط‌ها احترام بگذارید

- رابط‌ها یک محیط کنترل‌شده را برای کد R ما فراهم می‌کنند

به رابط‌ها احترام بگذارید

- رابط‌ها یک محیط کنترل‌شده را برای کد R ما فراهم می‌کنند
- تمام اطلاعاتی را که یک تابع نیاز دارد، با آرگومان‌ها به آن بدهید. این یک تمرین خوب است. در این صورت شانس وقوع خطا یا گیج شدن کم می‌شود

به رابط‌ها احترام بگذارید

- رابط‌ها یک محیط کنترل‌شده را برای کد R ما فراهم می‌کنند
- تمام اطلاعاتی را که یک تابع نیاز دارد، با آرگومان‌ها به آن بدهید. این یک تمرین خوب است. در این صورت شانس وقوع خطا یا گیج شدن کم می‌شود
- استثناء هم وجود دارد: شناخته‌شده‌های عمومی مثل π

به رابط‌ها احترام بگذارید

- رابط‌ها یک محیط کنترل‌شده را برای کد R ما فراهم می‌کنند
- تمام اطلاعاتی را که یک تابع نیاز دارد، با آرگومان‌ها به آن بدهید. این یک تمرین خوب است. در این صورت شانس وقوع خطا یا گیج شدن کم می‌شود
- استثناء هم وجود دارد: شناخته‌شده‌های عمومی مثل π
- به طور مشابه، خروجی تابع باید فقط از طریق مقادیر برگشت‌شده قابل دسترس باشند

مثال: رگرسیون غیرخطی

مراحل برآورد پارامترهای یک مدل رگرسیونی ساده مانند $Y = \beta_0 + \beta_1 X + \epsilon$ را به روش کمترین توان‌های دوم خطا، می‌دانیم. در این مدل خطی، برای بهترین برآوردهای β_0 و β_1، فرمول‌های دقیق وجود دارند. اما مشابه چنین فرمول‌هایی برای مدل‌های غیرخطی وجود ندارند.

مثال: رگرسیون غیرخطی

مراحل برآورد پارامترهای یک مدل رگرسیونی ساده مانند $Y = \beta_0 + \beta_1 X + \epsilon$ را به روش کمترین توان‌های دوم خطا، می‌دانیم. در این مدل خطی، برای بهترین برآوردهای β_0 و β_1، فرمول‌های دقیق وجود دارند. اما مشابه چنین فرمول‌هایی برای مدل‌های غیرخطی وجود ندارند.

جواب‌ها فرم بسته ندارند و می‌توان آن‌ها را به صورت تکراری به دست آورد.

مثال: رگرسیون غیرخطی

مراحل برآورد پارامترهای یک مدل رگرسیونی ساده مانند $Y = \beta_0 + \beta_1 X + \epsilon$ را به روش کمترین توان‌های دوم خطا، می‌دانیم. در این مدل خطی، برای بهترین برآوردهای β_0 و β_1، فرمول‌های دقیق وجود دارند. اما مشابه چنین فرمول‌هایی برای مدل‌های غیرخطی وجود ندارند.

جواب‌ها فرم بسته ندارند و می‌توان آن‌ها را به صورت تکراری به دست آورد.

اخیراً وجود یک رابطه غیرخطی بین تعداد جمعیت یک شهر، N، و تولید ناخالص شهری، Y، به صورت زیر، پیشنهاد شده است:

$$Y = y_0 N^a + \epsilon_0$$

به این مدل، یک مدل قانون توانی می‌گویند.

مثال: رگرسیون غیرخطی

مراحل برآورد پارامترهای یک مدل رگرسیونی ساده مانند $Y = \beta_0 + \beta_1 X + \epsilon$ را به روش کمترین توان‌های دوم خطا، می‌دانیم. در این مدل خطی، برای بهترین برآوردهای β_0 و β_1، فرمول‌های دقیق وجود دارند. اما مشابه چنین فرمول‌هایی برای مدل‌های غیرخطی وجود ندارند.

جواب‌ها فرم بسته ندارند و می‌توان آن‌ها را به صورت تکراری به دست آورد.

اخیراً وجود یک رابطه غیرخطی بین تعداد جمعیت یک شهر، N، و تولید ناخالص شهری، Y ، به صورت زیر، پیشنهاد شده است:

$$Y = y_0 N^a + \epsilon.$$

به این مدل، یک مدل قانون توانی می‌گویند.

می‌توان پارامترها را با می‌نیمم کردن MSE به دست آورد

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - y_0 N_i^a)^2,$$

که در آن y_0 معلوم و برابر ۶۶۱۱ دلار می‌باشد.

ادامه مثال

می‌توان پارامتر a را با تقریب زدن مشتق MSE و به روش تکراری به دست آورد (؟)

$$
\begin{aligned}
MSE(a) &= \frac{1}{n}\sum_{i=1}^{n}(Y_i - y_0 N_i^a)^2 \\
MSE'(a) &\approx \frac{MSE(a+h) - MSE(a)}{h} \\
a_{t+1} - a_t &\propto -MSE'(a)
\end{aligned}
$$

```
maximum.iterations <- 100
deriv.step <- 1/1000
step.scale <- 1e-12
stopping.deriv <- 1/100
iteration <- 0
deriv <- Inf
a <- 0.15
while ((iteration < maximum.iterations) && (deriv > stopping.deriv)) {
iteration <- iteration + 1
mse.1 <- mean((gmp$pcgmp - 6611*gmp$pop^a)^2)
mse.2 <- mean((gmp$pcgmp - 6611*gmp$pop^(a+deriv.step))^2)
deriv <- (mse.2 - mse.1)/deriv.step
a <- a - step.scale*deriv
}
list(a=a,iterations=iteration,converged=(iteration < maximum.iterations))
```

مشکلات کد نوشته‌شده

- غیرقابل انعطاف: برای تغییر حدس اولیه برای a باید آن را ویرایش کرد و سپس کد را کپی‌ـ پیست کرد و دوباره آن را اجرا کرد

مشکلات کد نوشته‌شده

- غیرقابل انعطاف: برای تغییر حدس اولیه برای a باید آن را ویرایش کرد و سپس کد را کپی‌ـ پیست کرد و دوباره آن را اجرا کرد
- پاک کردن خطا: چنانچه مجموعه داده تغییر کند، باید دوباره ویرایش و اجرای دوباره انجام شود و امیدوار باشیم همه ویرایش‌ها درست انجام شده‌اند

با تبدیل این کد به یک تابع و بهتر کردن آن، این مشکلات مرتفع خواهند شد.

اولین تلاش

```
estimate.scaling.exponent.1 <- function(a) {
maximum.iterations <- 100
deriv.step <- 1/1000
step.scale <- 1e-12
stopping.deriv <- 1/100
iteration <- 0
deriv <- Inf
while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
iteration <- iteration + 1
mse.1 <- mean((gmp$pcgmp - 6611*gmp$pop^a)^2)
mse.2 <- mean((gmp$pcgmp - 6611*gmp$pop^(a+deriv.step))^2)
deriv <- (mse.2 - mse.1)/deriv.step
a <- a - step.scale*deriv
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

مشکل: همه اعدادی است که عجیب هم به نظر می‌رسند

پاسخ: به عنوان آرگومان تابع با مقادیر پیش‌فرض تعریف شوند

```
estimate.scaling.exponent.2 <- function(a, y0=6611, maximum.iterations=100,
deriv.step = 1/100, step.scale = 1e-12, stopping.deriv = 1/100) {
iteration <- 0
deriv <- Inf
while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
iteration <- iteration + 1
mse.1 <- mean((gmp$pcgmp - y0*gmp$pop^a)^2)
mse.2 <- mean((gmp$pcgmp - y0*gmp$pop^(a+deriv.step))^2)
deriv <- (mse.2 - mse.1)/deriv.step
a <- a - step.scale*deriv
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

تمرین: با مقادیر متفاوتی از $deriv.step$ برنامه را اجرا کنید

```
estimate.scaling.exponent.2 <- function(a, y0=6611, maximum.iterations=100,
deriv.step = 1/100, step.scale = 1e-12, stopping.deriv = 1/100) {
iteration <- 0
deriv <- Inf
while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
iteration <- iteration + 1
mse.1 <- mean((gmp$pcgmp - y0*gmp$pop^a)^2)
mse.2 <- mean((gmp$pcgmp - y0*gmp$pop^(a+deriv.step))^2)
deriv <- (mse.2 - mse.1)/deriv.step
a <- a - step.scale*deriv
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

تمرین: با مقادیر متفاوتی از $deriv.step$ برنامه را اجرا کنید

مشکل: چرا محاسبات مشابه برای MSE دو بار انجام می‌شود؟

پاسخ: آن را به صورت یک تابع معرفی کن

```
estimate.scaling.exponent.3 <- function(a, y0=6611, maximum.iterations=100,
deriv.step = 1/100, step.scale = 1e-12, stopping.deriv = 1/100) {
iteration <- 0
deriv <- Inf
mse <- function(a) { mean((gmp$pcgmp - y0*gmp$pop^a)^2) }
while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
iteration <- iteration + 1
deriv <- (mse(a+deriv.step) - mse(a))/deriv.step
a <- a - step.scale*deriv
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

دقت کنید تابع mse درون تابع $estimate.scaling.exponent$ قرار گرفته است و به محیط R تعلق ندارد. یعنی با فراخوانی آن در محیط بزرگتر R پیغام خطا خواهید دید. اما مثلا مقدار y_0 را می‌توانید ببینید.

مشکل: این تابع فقط برای مجموعه داده gmp نوشته شده است. اگر بخواهیم با مجموعه داده دیگری نتایج را مقایسه کنیم، باید تابع دیگری بنویسیم؟

پاسخ: افزودن آرگومان‌های بیشتر با مقادیر پیش‌فرض

```
estimate.scaling.exponent.4 <- function(a, y0=6611, response=gmp$pcgmp,
predictor = gmp$pop, maximum.iterations=100, deriv.step = 1/100,
step.scale = 1e-12, stopping.deriv = 1/100) {
iteration <- 0
deriv <- Inf
mse <- function(a) { mean((response - y0*predictor^a)^2) }
while ((iteration < maximum.iterations) && (abs(deriv) > stopping.deriv)) {
iteration <- iteration + 1
deriv <- (mse(a+deriv.step) - mse(a))/deriv.step
a <- a - step.scale*deriv
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

می‌توان حلقه $while()$ را به حلقه $for()$ تبدیل کرد

```
estimate.scaling.exponent.5 <- function(a, y0=6611, response=gmp$pcgmp,
predictor = gmp$pop, maximum.iterations=100, deriv.step = 1/100,
step.scale = 1e-12, stopping.deriv = 1/100) {
mse <- function(a) { mean((response - y0*predictor^a)^2) }
for (iteration in 1:maximum.iterations) {
deriv <- (mse(a+deriv.step) - mse(a))/deriv.step
a <- a - step.scale*deriv
if (abs(deriv) <= stopping.deriv) { break() }
}
fit <- list(a=a,iterations=iteration,
converged=(iteration < maximum.iterations))
return(fit)
}
```

چرا باید بیشتر از یک تابع نوشت؟

- چون بیشتر از یک مساله وجود دارد

چرا باید بیشتر از یک تابع نوشت؟

- چون بیشتر از یک مساله وجود دارد
- مساله می‌تواند خیلی مشکل‌تر از آن باشد که در یک مرحله قابل حل باشد

چرا باید بیشتر از یک تابع نوشت؟

- چون بیشتر از یک مساله وجود دارد
- مساله می‌تواند خیلی مشکل‌تر از آن باشد که در یک مرحله قابل حل باشد
- می‌توانید مسایل مشابه را حل کنید

چرا باید بیشتر از یک تابع نوشت؟

- چون بیشتر از یک مساله وجود دارد
- مساله می‌تواند خیلی مشکل‌تر از آن باشد که در یک مرحله قابل حل باشد
- می‌توانید مسایل مشابه را حل کنید
- می‌توان مساله را به قسمت‌های کوچکتر تجزیه کرد و برای هر قسمت یک تابع نوشت

نوشتن توابع چندگانه مرتبط

آماردان‌ها دوست دارند با مدل‌هایشان خیلی کارها انجام دهند: برآورد، پیش‌گویی، آزمون فرضیه، نمایش بصری، شبیه‌سازی و غیره

نوشتن توابع چندگانه مرتبط

آماردان‌ها دوست دارند با مدل‌هایشان خیلی کارها انجام دهند: برآورد، پیش‌گویی، آزمون فرضیه، نمایش بصری، شبیه‌سازی و غیره

بنابراین باید توابع چندگانه‌ای برای انجام آن‌ها نوشته شوند

نوشتن توابع چندگانه مرتبط

آماردان‌ها دوست دارند با مدل‌هایشان خیلی کارها انجام دهند: برآورد، پیش‌گویی، آزمون فرضیه، نمایش بصری، شبیه‌سازی و غیره

بنابراین باید توابع چندگانه‌ای برای انجام آن‌ها نوشته شوند

برای این منظور، باید مدل را به عنوان یک شیء تعریف کنیم و در توابع مرتبط با هم باید یک سازگاری در رابط‌ها وجود داشته باشد:

- توابع نوشته شده برای یک نوع مشابه از اشیاء، باید آرگومان‌های مشابه داشته باشند و یک ساختار مشابه را در نظر بگیرند

نوشتن توابع چندگانه مرتبط

آماردان‌ها دوست دارند با مدل‌هایشان خیلی کارها انجام دهند: برآورد، پیش‌گویی، آزمون فرضیه، نمایش بصری، شبیه‌سازی و غیره

بنابراین باید توابع چندگانه‌ای برای انجام آن‌ها نوشته شوند

برای این منظور، باید مدل را به عنوان یک شیء تعریف کنیم و در توابع مرتبط با هم باید یک سازگاری در رابط‌ها وجود داشته باشد:

- توابع نوشته شده برای یک نوع مشابه از اشیاء، باید آرگومان‌های مشابه داشته باشند و یک ساختار مشابه را در نظر بگیرند
- توابعی که برای انجام یک کار مشابه نوشته می‌شوند باید آرگومان‌های مشابه داشته باشند و مقادیر خروجی مشابهی گزارش دهند

در نهایت همه توابع مرتبط با هم را در یک فایل تکی قرار دهید و برای توضیح هر کدام چند خط توضیح بنویسید

ادامه مثال مدل رگرسیون

پیش‌گویی مقادیر جدید با استفاده از مدل برازش‌شده

```
# Predict response values from a power-law scaling model
# Inputs: fitted power-law model (object), vector of values at which to make
# predictions at (newdata)
# Outputs: vector of predicted response values
predict.plm <- function(object, newdata) {
# Check that object has the right components
stopifnot("a" %in% names(object), "y0" %in% names(object))
a <- object$a
y0 <- object$y0
# Sanity check the inputs
stopifnot(is.numeric(a),length(a)==1)
stopifnot(is.numeric(y0),length(y0)==1)
stopifnot(is.numeric(newdata))
return(y0*newdata^a) # Actual calculation and return
}
```

```
# Plot fitted curve from power law model over specified range
# Inputs: list containing parameters (plm), start and end of range (from, to)
# Outputs: TRUE, silently, if successful
# Side-effect: Makes the plot
plot.plm.1 <- function(plm,from,to) {
# Take sanity-checking of parameters as read
y0 <- plm$y0 # Extract parameters
a <- plm$a
f <- function(x) { return(y0*x^a) }
curve(f(x),from=from,to=to)
# Return with no visible value on the terminal
invisible(TRUE)
}
```

وقتی یک تابع، تابع دیگری را فراخوانی می‌کند، از ... به عنوان یک فرا آرگومان استفاده می‌شود تا ورودی‌های نامشخص را به تابع فراخوانی‌شده واگذار کند

```
plot.plm.2 <- function(plm,...) {
y0 <- plm$y0
a <- plm$a
f <- function(x) { return(y0*x^a) }
# from and to are possible arguments to curve()
curve(f(x),...)
invisible(TRUE)
}
```

زیرتوابع

مسایل بزرگ را با تقسیم کردن آن‌ها به چند زیرمساله کوچکتر حل کنید

- فهم ساده‌تر

زیرتوابع

مسایل بزرگ را با تقسیم کردن آن‌ها به چند زیرمساله کوچکتر حل کنید

- فهم ساده‌تر
- اصلاح و بهبود دادن راحت‌تر

زیرتوابع

مسایل بزرگ را با تقسیم کردن آن‌ها به چند زیرمساله کوچکتر حل کنید

- فهم ساده‌تر
- اصلاح و بهبود دادن راحت‌تر
- طراحی ساده‌تر

یک قانون سرانگشتی: تابعی طولانی‌تر از یک صفحه، احتمالا بیش از حد طولانی است

زیرتوابع یا توابع جداگانه؟

زیرتوابع درون یک تابع دیگر تعریف شده‌اند.

زیرتوابع یا توابع جداگانه؟

زیرتوابع درون یک تابع دیگر تعریف شده‌اند.

مزیت‌های زیرتوابع: سادگی کد نوشته‌شده، دسترسی به متغیرهای محلی و مغشوش نکردن محیط R است

زیرتوابع یا توابع جداگانه؟

زیرتوابع درون یک تابع دیگر تعریف شده‌اند.

مزیت‌های زیرتوابع: سادگی کد نوشته‌شده، دسترسی به متغیرهای محلی و مغشوش نکردن محیط R است

معایب آن‌ها: ارزیابی هر بار آن‌ها در هر بار فراخوانی تابع بزرگتر است و این که در محیط بزرگتر نمی‌توان به آن‌ها دسترسی داشت

زیرتوابع یا توابع جداگانه؟

زیرتوابع درون یک تابع دیگر تعریف شده‌اند.

مزیت‌های زیرتوابع: سادگی کد نوشته‌شده، دسترسی به متغیرهای محلی و مغشوش نکردن محیط R است

معایب آن‌ها: ارزیابی هر بار آن‌ها در هر بار فراخوانی تابع بزرگتر است و این که در محیط بزرگتر نمی‌توان به آن‌ها دسترسی داشت

جانشین: نوشتن جداگانه آن‌ها و تعبیه همگی در یک فایل و یکی کردن آن‌هاست.

مثال

```
plot.plm.3 <- function(plm,from,to,n=101,...) {
x <- seq(from=from,to=to,length.out=n)
y <- predict.plm(object=plm,newdata=x)
plot(x,y,...)
invisible(TRUE)
}
```

مثال تخصیص منابع

```
planner <- function(output,factory,available,slack,tweak=0.1) {
needed <- plan.needs(output,factory)
if (all(needed <= available) && all(available-needed <= slack)) {
return(list(output=output,needed=needed))
}
else {
output <- adjust.plan(output,needed,available,tweak)
return(planner(output,factory,available,slack))
}
}
plan.needs <- function(output,factory) { factory %*% output }
adjust.plan <- function(output,needed,available,tweak) {
if (all(needed >= available)) { return(output*(1-tweak)) }
if (all(needed < available)) { return((1+tweak)) }
return(output*runif(n=length(output),min=1-tweak,max=1+tweak))
}
```