

Lecture 25: Making code more efficient

Approaches to faster code

- Do as little as possible
- Vectorise
- Avoid copies

Do as little as possible

```
1 n <- 100000
2 cols <- 150
3 data_mat <- matrix(rnorm(n * cols, mean = 5), ncol = cols)
4 data <- as.data.frame(data_mat)
5
6 bench::mark(
7   means <- colMeans(data_mat),
8   means <- colMeans(data),
9   check = F
10 )
```

A tibble: 2 × 6

expression	min	median	`itr/sec`	mem_alloc
<code><bch:expr></code>	<code><bch:tm></code>	<code><bch:tm></code>	<code><dbl></code>	<code><bch:byt></code>
<code><dbl></code>				
1 means <- colMeans(data_mat)	434ms	435ms	2.30	25.4KB
0				
2 means <- colMeans(data)	452ms	453ms	2.21	114.5MB
2.21				

Avoid copies

The code below samples 100 observations from a $N(0, 1)$ distribution:

```
1 x <- c()
2 for(i in 1:100){
3   x <- c(x, rnorm(1))
4 }
```

How could I make this code more efficient?

Avoid copies

```
1  loop_1 <- function(n){
2    x <- c()
3    for(i in 1:n){
4      x <- c(x, rnorm(1))
5    }
6    return(x)
7  }
8
9  loop_2 <- function(n){
10   x <- rep(NA, n)
11   for(i in 1:n){
12     x[i] <- rnorm(1)
13   }
14   return(x)
15 }
```

Avoid copies

```
1 bench::mark(  
2   loop_1(100),  
3   loop_2(100),  
4   check = F  
5 )
```

```
# A tibble: 2 × 6
```

	expression	min	median	`itr/sec`	mem_alloc	`gc/sec`
	<bch:expr>	<bch:tm>	<bch:tm>	<dbl>	<bch:byt>	<dbl>
1	loop_1(100)	139µs	159µs	5604.	318KB	11.6
2	loop_2(100)	111µs	116µs	7735.	272KB	11.6

Avoid copies

```
1 bench::mark(  
2   loop_1(10000),  
3   loop_2(10000),  
4   check = F  
5 )
```

A tibble: 2 × 6

	expression	min	median	`itr/sec`	mem_alloc	`gc/sec`
	<bch:expr>	<bch:tm>	<bch:tm>	<dbl>	<bch:byt>	<dbl>
1	loop_1(10000)	76.7ms	81.5ms	11.2	406.3MB	24.2
2	loop_2(10000)	11.2ms	11.4ms	75.2	24.5MB	9.89

Vectorise

The code below samples 100 observations from a $N(0, 1)$ distribution:

```
1 x <- rep(NA, 100)
2 for(i in 1:100){
3   x[i] <- rnorm(1)
4 }
```

How could I make this code more efficient?

Vectorise

```
1 for_loop_sample <- function(n){
2   x <- rep(NA, n)
3   for(i in 1:n){
4     x[i] <- rnorm(1)
5   }
6 }
7
8 bench::mark(
9   x <- for_loop_sample(100),
10  x <- rnorm(100),
11  check=F
12 )
```

A tibble: 2 × 6

expression	min	median	`itr/sec`	mem_alloc	
<code><bch:expr></code>	<code><bch:tm></code>	<code><bch:tm></code>	<code><dbl></code>	<code><bch:byt></code>	
<code><dbl></code>					
1 x <- for_loop_sample(100)	111.21µs	114.67µs	8546.	271.04KB	
13.6					
2 x <- rnorm(100)	5.42µs	6.29µs	155260.	3.32KB	0

Other options

- Different data structures / algorithms
- Parallelization
- Rewrite code in C++

Class activity

https://sta279-s24.github.io/class_activities/ca_lecture_25.html

