Lecture 25: Making code more efficient

Approaches to faster code

- Do as little as possible
- Vectorise
- Avoid copies

Do as little as possible

(ence extraneous step if you're sure they are extraneous)

```
 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
`gc/sec`
  <bch:expr>                  <bch:tm> <bch:tm>     <dbl> <bch:byt>
<dbl>
1 means <- colMeans(data_mat)    437ms    437ms      2.29   25.4KB
0
2 means <- colMeans(data)        456ms    458ms      2.18   114.5MB
2.18
```

Avoid copies

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

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

How could I make this code more efficient?

- Growing vector is slow
  (every time I add a new entry I have to create a new object & allocate memory)

Alternative: create a vector of the right length then fill it in

Avoid copies

```
loop_1 <- function(n){
  x <- c()
  for(i in 1:n){
    x <- c(x, rnorm(1))
  }
  return(x)
}

loop_2 <- function(n){
  x <- rep(NA, n)
  for(i in 1:n){
    x[i] <- rnorm(1)
  }
  return(x)
}
```

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)    143µs    161µs     5552.     318KB     11.6
2 loop_2(100)    114µs    118µs     7529.     272KB     11.5
```

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)   85.7ms  104.8ms      9.22   406.3MB    20.0
2 loop_2(10000)   11.8ms   12.2ms     70.6     24.5MB     9.80
```

much faster

much less memory

Vectorise

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

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

How could I make this code more efficient?

rnorm(100)

Vectorise

```r
for_loop_sample <- function(n){
  x <- rep(NA, n)
  for(i in 1:n){
    x[i] <- rnorm(1)
  }
}

bench::mark(
  x <- for_loop_sample(100),
  x <- rnorm(100),
  check=F
)
```

```
# A tibble: 2 × 6
  expression                      min   median `itr/sec` mem_alloc
`gc/sec`
  <bch:expr>                  <bch:tm> <bch:tm>     <dbl> <bch:byt>
<dbl>
1 x <- for_loop_sample(100) 113.92µs 119.04µs     8069.  271.04KB
11.2
2 x <- rnorm(100)             5.42µs   6.33µs   151517.    3.32KB
15.2
```

about 20× faster

Other options

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

Class activity

https://sta279-f23.github.io/class_activities/ca_lecture_25.html