Lecture 25: Making code more efficient

Approaches to faster code

- Do as little as possible
- Vectorise
- Avoid copies

Do as little as possible

remove extraneous steps if we're sure they are extraneous

```r
n <- 100000
cols <- 150
data_mat <- matrix(rnorm(n * cols, mean = 5), ncol = cols)
data <- as.data.frame(data_mat)

bench::mark(
  means <- colMeans(data_mat),
  means <- colMeans(data),
  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 means <- colMeans(data_mat)    434ms    435ms      2.30    25.4KB
0
2 means <- colMeans(data)        452ms    453ms      2.21   114.5MB
2.21
```

avoiding conversion from data frame to matrix

Avoid copies

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

create an empty length 0 vector

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

add a a new entry to the end of x, resave the object

How could I make this code more efficient?

- One option: use rnorm to take multiple samples directly
  rnorm(n) (n samples from Normal distribution)

- option 2: create vector of length 100 and then fill it in inside the for loop

Avoid copies

```r
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

growing vector

```r
bench::mark(
  loop_1(100),
  loop_2(100),
  check = F
)
```

allocating space ahead of time

```
# 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
```

slightly faster

less memory

Avoid copies

```r
bench::mark(
  loop_1(10000),
  loop_2(10000),
  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 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
```

↑ much faster

↑ much less memory

Vectorise

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

```r
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) 111.21µs 114.67µs     8546.  271.04KB
13.6
2 x <- rnorm(100)             5.42µs   6.29µs   155260.    3.32KB      0
```

↑ about 20x faster

↑ much less memory

Other options

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

Class activity

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