stsci4520_lab7

Nick Gembs

3/8/2023

```
# let's look at the exp function for an example
# we'll compare it to using exponentiation
n <- 1000
x <- rnorm(n)
y <- 10*rnorm(n)
e <- exp(1)

library("bench")
# spend a little time reading the documentation

# this can be alternatively called with the "cute" syntax bench::mark( exp(x) )
result <- mark( exp(x), e^x )
# this contains a table with information about how long it takes
# exp(x) is a little bit faster than e^x
result
```

```
## # A tibble: 2 x 6
##   expression      min   median `itr/sec` mem_alloc `gc/sec`
##   <bch:expr> <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
## 1 exp(x)       75.8us    121us     7275.    7.86KB        0
## 2 e^x         177.1us    288us     3063.    7.86KB        0
```

```
# run this block of code a few times to get a sense of whether the results are consistent

# let's do y and compare
result <- mark( exp(y), e^y )
result
```

```
## # A tibble: 2 x 6
##   expression      min   median `itr/sec` mem_alloc `gc/sec`
##   <bch:expr> <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
## 1 exp(y)       47.6us   92.6us     8176.    7.86KB        0
## 2 e^y         201.5us  321.6us     2191.    7.86KB        0
```

```
# Exercise: read the documentation for "press" and use it to
# benchmark exp(x) against e^x for a range of values of n,
# such as n = 10, n = 100, n = 1000, n = 10000
```

```r
# the exponential function has a series expansion:
# exp(x) = 1 + x/1! + x^2/2! + x^3/3! + ...

# let's try to beat the exp function using the series expansion
my_exp1 <- function(x){
  z <- rep(1,length(x))
  y <- x
  for(j in 1:3){
    z <- z + y/factorial(j)
    y <- y*x
  }
  return(z)
}

# this only works for small x
x <- 0.01*rnorm(n)
mark( exp(x), e^x, my_exp1(x) )
```

```
## # A tibble: 3 x 6
##   expression      min   median `itr/sec` mem_alloc `gc/sec`
##   <bch:expr> <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
## 1 exp(x)       68.4us    126us     7212.    7.86KB     2.06
## 2 e^x         145.9us    232us     3802.    7.86KB     0
## 3 my_exp1(x)   11.6us     42us    15242.  101.94KB    14.5
```

```r
n <- 1000
# Exercise: see if you can write your own version of my_exp that beats exp
my_exp2 <- function(x){
  # fill in your code here

  z <- 1 + x + x*x*((3+x)/6)

  return(z)
}

x <- 0.01*rnorm(n)
mark( exp(x), e^x, my_exp1(x), my_exp2(x) )
```

```
## # A tibble: 4 x 6
##   expression      min   median `itr/sec` mem_alloc `gc/sec`
##   <bch:expr> <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
## 1 exp(x)         75us  108.7us     8043.    7.86KB     2.06
## 2 e^x           156us  222.8us     3738.    7.86KB     0
## 3 my_exp1(x)   15.6us   42.3us    15268.   55.02KB    14.7
## 4 my_exp2(x)    5.5us   16.4us    35579.   23.58KB    17.8
```

```r
n <- 999999
x <- 0.01*rnorm(n)

t1 <- proc.time()
ans = exp(x)
t2 <- proc.time()
print("exp(x) time:")
```

```
## [1] "exp(x) time:"
```

```
print(t2-t1)
```

```
##    user  system elapsed
##    0.11    0.02    0.12
```

```
t1 <- proc.time()
ans1 = my_exp1(x)
t2 <- proc.time()
print("my_exp1(x) time:")
```

```
## [1] "my_exp1(x) time:"
```

```
print(t2-t1)
```

```
##    user  system elapsed
##    0.03    0.01    0.05
```

```
t1 <- proc.time()
ans2 = my_exp2(x)
t2 <- proc.time()
print("my_exp2(x) time:")
```

```
## [1] "my_exp2(x) time:"
```

```
print(t2-t1)
```

```
##    user  system elapsed
##    0.01    0.02    0.03
```

```
# sorting strings vs factors vs numbers
hot <- read.csv("C:/Users/Nick/Downloads/Hot_100.csv")

# pay attention to the units!
mark(
  order( hot$chart_date ),
  order( as.Date( hot$chart_date ) ),
  order( as.factor(hot$chart_date) ),
  order( as.numeric(as.factor(hot$chart_date) ) )
)
```

```
## Warning: Some expressions had a GC in every iteration; so filtering is
## disabled.
```

```
## # A tibble: 4 x 6
##   expression                                   min   median itr/s~1 mem_a~2
##   <bch:expr>                                <bch:tm> <bch:tm>   <dbl> <bch:b>
## 1 order(hot$chart_date)                       8.07s    8.07s   0.124  1.28MB
```

```
## 2 order(as.Date(hot$chart_date))                    5.1s     5.1s   0.196 43.67MB
## 3 order(as.factor(hot$chart_date))                48.71ms  69.98ms  14.2   12.05MB
## 4 order(as.numeric(as.factor(hot$chart_date)))    87.83ms  94.45ms   9.14  13.31MB
## # ... with 1 more variable: `gc/sec` <dbl>, and abbreviated variable names
## #   1: `itr/sec`, 2: mem_alloc
```

factors and numbers return total time in milliseconds, while string and Date return in seconds. numbers were the quickest to sort, followed by factors, then dates, and finally, strings.

```
# how much time is spent on the actual ordering?
hot_string <- hot$chart_date
hot_date <- as.Date( hot$chart_date )
hot_fact <- as.factor( hot$chart_date )
hot_number <- as.numeric( as.factor( hot$chart_date ) )
mark( order(hot_string), order(hot_date), order(hot_fact), order(hot_number) )
```

```
## # A tibble: 4 x 6
##   expression             min   median `itr/sec` mem_alloc `gc/sec`
##   <bch:expr>        <bch:tm> <bch:tm>     <dbl> <bch:byt>    <dbl>
## 1 order(hot_string)     9.5m     9.5m   0.00175    1.28MB     0
## 2 order(hot_date)    19.85ms  24.82ms  39.0        3.85MB     2.17
## 3 order(hot_fact)     2.72ms   3.27ms 273.         2.59MB    12.3
## 4 order(hot_number)  12.31ms  13.25ms  66.7        1.28MB     2.08
```

The results were different for the actual ordering. factors, dates, and numbers return total time in milliseconds, while string returns in seconds. factors were the quickest to order, followed by dates, then numbers, and finally, strings.