R Tips

Zepu Zhang

September 1, 2008

1. For simple timing, use `system.time`. To time a group of statements, enclose them in braces.

2. To find out the number of TRUE elements in a vector, e.g., `sum(x > 0.5)` is faster than `length(which(x > 0.5))`.

3. For a list, pre-allocate the full length helps slightly:

```
> system.time({
+   mat <- vector('list', 10000);
+   for (i in 1:10000) mat[[i]] <- matrix(1:10000, 100, 100)
+ })
   user  system elapsed
  1.220   0.024   1.242
>
>
> system.time({
+   mat <- list();
+   for (i in 1:10000) mat[[i]] <- matrix(1:10000, 100, 100)
+ })
   user  system elapsed
  1.741   0.000   1.740
```

   In contrast, using `mat <- append(mat, list(...))` would be very slow.

4. For long loops, move common operations, such as multiplication or summation by a constant, out of the loop. The time consumption can be appreciable.

5. Use `seq_along`.

6. Compare

```
> system.time({
+   M <- matrix(0, 100, 100);
+   for (i in 1:100000) M[,] <- matrix(1:10000, 100, 100)
+ })
   user  system elapsed
 19.041   0.000  19.041
```

```
>
>
> system.time({
+   M <- matrix(0, 100, 100);
+   for (i in 1:100000) M <- matrix(1:10000, 100, 100)
+ })
   user  system elapsed
  9.933   0.000   9.936
```

Compare

```
> system.time({
+   M <- matrix(0, 100, 100);
+   for (i in 1 : 10000) M <- M + matrix(10000, 100, 100)
+ })
   user  system elapsed
    1.5     0.0     1.5
>
>
> system.time({
+   M <- matrix(0, 100, 100);
+   for (i in 1 : 10000) M[,] <- M + matrix(10000, 100, 100)
+ })
   user  system elapsed
  2.456   0.000   2.454
```

7. Use `mapply` and `Vectorize` to vectorize a function that usually does not take vector arguments:

```
> mapply(
+   function(lo, hi) {integrate(dnorm, lo, hi)$value},
+   c(-Inf, 0.3),
+   c(0.3, Inf))
[1] 0.6179114 0.3820886
>
>
> Integrate <- Vectorize(
+   function(fn, lo, hi) integrate(fn, lo, hi)$value,
+   vectorize.args = c('lo', 'hi')
+ )
> Integrate(dnorm, c(-Inf, 0.3), c(0.3, Inf))
[1] 0.6179114 0.3820886
```