Vectorized version of random walk script

David Zelený

December 20, 2015

The original function to generate random walk was based on the `for` loop of steps - in each step, random value (either -1 or 1) was generated, and added to the previous values (check Creating a loop and function using random walk example). The script of `random.walk` function looks like this:

```
random.walk <- function (no.steps)
{
  steps.rw <- vector (mode = 'numeric', length = no.steps)
  for (k in seq (2, no.steps))
  {
    steps.rw [k] <- steps.rw [k-1] + sample (c(-1,1),1)
  }
  steps.rw
}
```

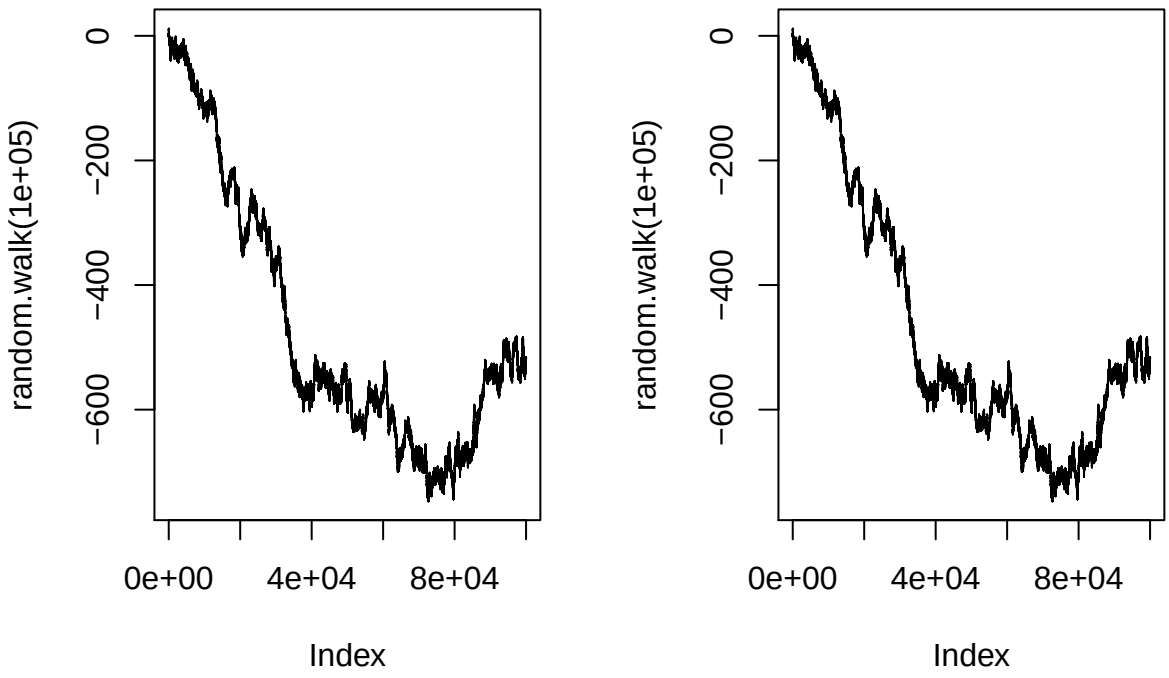
In this case, we may *vectorize* it - replace the loop by function(s) which can take as arguments the whole vectors. The solution is to first generate a vector of the length `no.steps`, which contains randomly selected values -1 or 1, and then use this vector in function `cumsum`, which returns the vector of cummulative sums - the vector of the same length as it's argument, where each element represents the cummulative sum of elements in the argument.

Vectorized option is:

```
random.walk.vec <- function (no.steps)
{
steps.rw <- sample (c(-1,1), size = no.steps-1, replace = T)
steps.rw <- c(0, steps.rw)
steps.rw <- cumsum (steps.rw)
return (steps.rw)
}
```

We may visually check that both functions produce exactly the same output. We will use `plot` function to draw the results of each random walk procedure applied on 100,000 steps (takes a while). Note the use of `set.seed` function - we need to setup the same random seed before running each function, to make sure that the generator of random values will start always from the same position:

```
par (mfrow = c(1,2))
set.seed (123)
plot (random.walk (100000), type = 'l')
set.seed (123)
plot (random.walk (100000), type = 'l')
```



And comparison of speed using `microbenchmark` function from `library (microbenchmark)` (note that here I decreased the number of steps from 100,000 to 10,000 to save time, since each function is repeated 100 times to obtain average times):

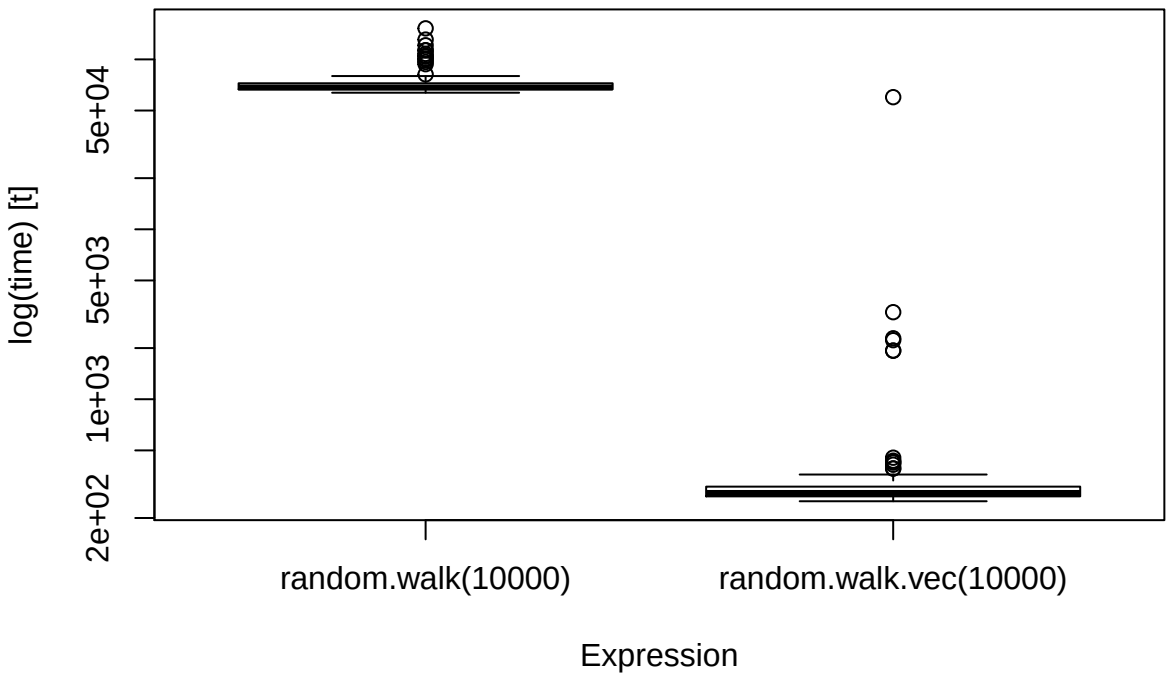
```
library (microbenchmark)
microbenchmark (random.walk (10000), random.walk.vec (10000))
```

```
## Unit: microseconds
##                    expr       min         lq       mean     median
##      random.walk(10000) 65883.925 69421.9590 86688.6819 76589.3250
##  random.walk.vec(10000)   257.009   283.7365   339.7092   308.9665
##          uq        max neval cld
##  91481.6940 209848.336   100   b
##    360.2825    827.901   100  a
```

Note that the vectorized version (the second row) is in average 226 times faster than the function using `for` loop! To calculate this number, I took the column `mean` in the table from `microbenchmark`: in average one run of `random.walk (10000)` takes 49341 microseconds = 0.049 seconds, while `random.walk.vec (10000)` takes only 218 microseconds = 0.0002 seconds, 49340/218 = 226. This value is changable - depends on the speed of your computer, and also on the number of steps the function has to do.

And because we like figures, we may draw the comparison also as a boxplot using `boxplot.microbenchmark` function from the same library:

```
boxplot (microbenchmark (random.walk (10000), random.walk.vec (10000)))
```


log(time) [t]
2e+02
1e+03
5e+03
5e+04
random.walk(10000)
random.walk.vec(10000)
Expression