`rowSums` × `apply` × `for`: Solution

David Zelený

```
# install.packages ('vegan')
library (vegan)
data (dune)

# The original 'rowSums' solution:
result.rowSums <- rowSums (dune > 0)

# Use of 'apply' function:
result.apply <- apply (dune > 0, 1, sum)

# Use of 'for' loop:
result.for <- vector (mode = 'numeric', length = nrow (dune))
names (result.for) <- rownames (dune)
for (ro in seq (1, nrow (dune)))
  result.for[ro] <- sum (dune[ro,] > 0)

# And results printed on screen:
result.rowSums
# 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
# 5 10 10 13 14 11 13 12 13 12  9  9 10  7  8  8  7  9  9  8

result.apply
# 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
# 5 10 10 13 14 11 13 12 13 12  9  9 10  7  8  8  7  9  9  8

result.for
# 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
# 5 10 10 13 14 11 13 12 13 12  9  9 10  7  8  8  7  9  9  8
```

How fast is each solution? You may use function `microbenchmark` from the package of the same name; it takes each expression in it's argument (expressions are separated by commas), repeats it 100-times, and counts average time it takes:

```
#install.packages ('microbenchmark')
library (microbenchmark)
library (vegan)
data (dune)
microbenchmark(result.rowSums <- rowSums (dune > 0),
               result.apply <- apply (dune > 0, 1, sum),
               {result.for <- vector (mode = 'numeric', length = nrow (dune))
                names (result.for) <- rownames (dune)
                for (ro in seq (1, nrow (dune)))
                  result.for[ro] <- sum (dune[ro,] > 0)}
               )
```

```
Unit: microseconds
...
...
     min       lq      mean   median       uq     max neval
   229.6   242.75   262.014   249.70   259.55   450.2   100
   289.6   302.80   329.014   318.80   332.30   527.9   100
 10679.6 11163.50 12097.821 11436.25 11840.15 18650.9   100
```

The last version (`for` loop) is the slowest - in average (column `mean`) one calculation takes 12098 microseconds = 12.1 millisecond = 0.0121 second, while the other two (`rowSums` and `apply` solution) are faster and quite similar to each other and take between 262-329 microseconds. The `for` loop solution is therefore more than 40-times slower than the other two solutions.