Vectorization in R

Pariksheet Nanda

March 29, 2016

1 Why vectorize?

The problem with using any high level programming languages is losing sight of how the language authors intend for you to use the language efficiently. Most often the conversation around vectorization is about improving the program's execution speed. However writing vectorized code is also beautiful, correct, code; to see why this is, we need to take a step back.

There are many ways to write programs to achieve a particular goal. In fact, this freedom to reach one's goal by several means is what makes programming an art form; there is enough room to express one's particular style without being restricted to a single 'correct' answer. However, while multiple programs can produce the same result, some programs are more correct than others. Compare these 2 result equivalent code snippets from chapter 3 of the *R Inferno*[1]:

```
> x <- rnorm(100)
```

```
> # C-style summation
> lsum <- 0
> for(i in 1:length(x)) {
+   lsum <- lsum + log(x[i])
+ }
```

```
> # Vectorized summation
> lsum <- sum(log(x))
```

More than being 'correct', code can be beautiful when it pays attention to 3 ideas:

Concise As the saying goes, software is complete not when nothing more can be added, but when nothing can be taken away.

Human legible Variable names and logic should be quickly understood. Avoid obscure shorthand. Comments are useful, but prefer to write code that reads so easily that comments are not needed to explain individual lines. Code appearance should follow standard practices and 'idioms'.

Unit Tested Bugs are an inevitable consequence of a growing program line count. Most functions should have tests to prove that they, infact, produce their intended results. Instead of throwing away short checks in the interpreter, they are preserved as unit tests so that bugs don't creep back in as code is sculpted and improved.

Thus, in addition to efficiency, vectorization is also about supporting these 3 ideas to writing beautiful and correct code.

2 Concepts

1. Application Programming Interface (API)
2. Vectors, List, Matrices, Data.Frames
3. Logical conditionals (&, |, !)
4. Vector Recycling Rule (a.k.a broadcasting in NumPy)
5. Profiling code with system.time()

[1] `http://www.burns-stat.com/pages/Tutor/R_inferno.pdf`

3 Exercises

1. Overload the multiplication operator to concatenate strings.

   ```
   > "abc" * "def" # should give "abcdef"
   ```

2. Subtract each element in the following array from it's left neighbor.

   ```
   > x = seq(4, 7) # should get "1 1 1"
   ```

3. Find the derivative of this function only using vectorization

   ```
   > x = seq(0, 2*pi, 0.01)
   > y = sin(x)
   > plot(x, y, type='l')
   ```

4. Write an averaging filter using vectorization and apply it to the parrot sample image, using the following pattern of averaging 5 neighboring pixels:

   $$\begin{matrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 1 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

   To do this you will beed the EBImage package:

   ```
   > source("https://bioconductor.org/biocLite.R")
   > biocLite("EBImage")
   ```

   Then open the sample image with:

   ```
   > library(EBImage)
   > f = system.file("images", "sample.png", package="EBImage")
   > img = readImage(f)
   > display(img)
   ```

4 References

1. Vectorization by Alyssa Frazee `http://alyssafrazee.com/vectorization.html`
2. Vectorization by Noam Ross `http://www.noamross.net/blog/2014/4/16/vectorization-in-r--why.html`
3. Vectoriation in R Inferno `http://www.burns-stat.com/pages/Tutor/R_inferno.pdf`
4. DPLYR Library `http://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf`
5. Some essential built-in functions (see pages 1 and 2) `https://cran.r-project.org/doc/contrib/Short-refcard.pdf`
6. Smaller subset of built-in functions (see pages 1 and 2) `https://www.ualberta.ca/~ahamann/teaching/renr690/R_Cheat_Data.pdf`
7. Matrix multiplication `http://www.statmethods.net/advstats/matrix.html`
8. Long list of functions grouped by category and operation `http://mathesaurus.sourceforge.net/octave-r.html`
9. Reading R source code `http://stackoverflow.com/a/19226817`