Assignment 2 Feedback

STA 242

May 22, 2015

Formatting Requirements

Please make sure you do the following for every assignment:

- Include the URL of your Bitbucket repository.
- Put margins on every page, including source code.[1]
- Use a white or light-colored background for source code.
- Format tables neatly (no raw output).
- Put your files in the `AssignmentN` directory.[2]

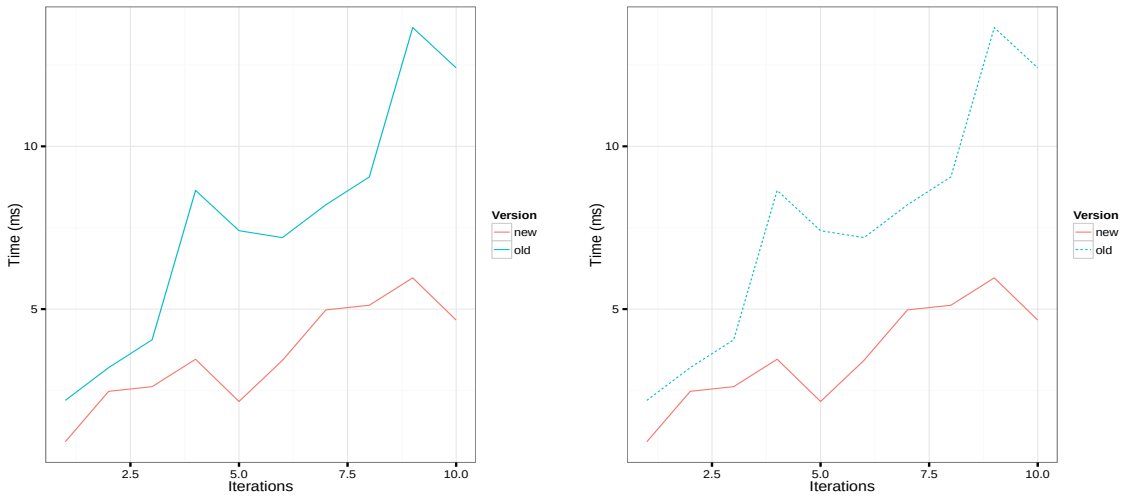
[1] Otherwise there's nowhere to write feedback.

[2] For instance, Assignment 5 goes in the `Assignment5/` directory.

Making Plots

Distinct Lines

Make plots print-friendly by using distinct line styles.



In the figure above, the lines on the left plot will look identical when printed in black and white. The right plot doesn't have this problem, because the line styles are distinct. Here's the code to produce the right plot:

```
plt = ggplot(data, aes(x=Iterations, y=Time))
# Always add units to labels!
plt = plt + labs(y="Time (ms)")
plt = plt + geom_line(aes(color=Version, linetype=Version))
```

```
5
6 plt
```

Even more distinct styles are possible by mixing line styles and point styles.

High Resolution

When printing, make sure your plots have a high enough resolution that details can be made out. The easiest way to do this is to always use the `pdf()` device for saving plots.

If you prefer to use a different graphics device, make sure the pixel size or `res` argument is suitably high. As an example, the American Statistical Association requires 300 dots per inch for publication figures.

Scientific Computing

Timing Code

When timing code, use multiple replications for each trial. If you're a statistics student and didn't do this, shame on you!

A single measurement could have substantial error due to other programs running on your computer. Because it's based on multiple replications, the mean will be more accurate. Reporting the number of replications and the standard deviation is also a good idea (unless, perhaps, it's very small).

Slow C Code

About five people reported that their C code was *slower* than their R code. For most of these, the C code was faster (as we'd expect), but they were using `matrix()` within their R wrapper function. It turns out that `matrix()` can be incredibly slow in many cases. Profiling is the best way to detect this.

Odds & Ends

- Using a log-scale improves plot readability when the data has a wide range.
- `which()` is rarely needed when subsetting.
- C code for R packages goes in the `src/` directory, not `SRC/` or any other variation.
- C and C++ are different languages. There seemed to be some confusion about this.