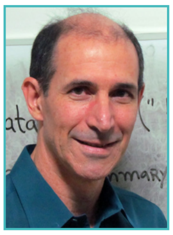
Large-scale statistical analysis of genome data with Ruby and R: skipping interface libraries



Sergio R. P. Line✉, **Ana P. de Souza, Luciana S. Mofatto**
Department of Morphology, Piracicaba Dental School, Piracicaba, SP, Brazil

Received 30 January 2014; **Accepted** 29 April 2014; **Published** 26 May 2014

Line SRP *et al.* (2014) *EMBnet.journal 20*, e753. http://dx.doi.org/10.14806/ej.20.0.753

Competing Interests: none

Abstract
Ruby is a dynamic interpreted, open source, object-oriented programming language with an elegant syntax and a focus on simplicity and productivity. One factor that may hinder the dissemination of Ruby, among academic and technological communities, is that it does not contain built-in methods for statistical analysis and graph creation. Statistical analysis with numerical data generated by Ruby scripts is traditionally performed by storing data to a file, which is read into another software environment for statistical analysis, using a package such as R. In order to circumvent this limitation, libraries have been created to perform statistical analysis with Ruby. These have not gained popularity, possibly owing to its limited statistical methods and relative complex usage. In this paper, we describe a simple and dynamic procedure to connect Ruby and R scripts. We show that this approach can be used for large-scale genome-data processing and statistical analysis. Its usage is simpler than interface libraries, as it does not require the creation of methods or routines other than those already existing in R and Ruby.

Inroduction
The development of high-throughput DNA sequencing techniques has led to an exponential increase in the volume of data available for analysis. This has opened new frontiers in medical and biological studies, and boosted interest in the genome research arena. However, in many cases, the analysis of large volumes of data can only be performed by computationally intensive and complex methods that include computer processing and statistical analysis of sequences. Developing statistical and programming skills is a major challenge, and frequently a discouraging factor for students and researchers in the biomedical area.

Ruby is a dynamic interpreted, open source, object-oriented programming language with a focus on productivity (Flanagan and Matsumoto, 2008); it is characterised by an elegant syntax and simplicity, it is natural to read and easy to learn (Aerts, 2009). The wide range of built-in methods for string manipulation, reflection and meta-programming capabilities make this language especially suitable for bioinformatics, where the size and complexity of codes can hinder readability (Aerts, 2009). There are several Ruby implementations available, such as JRuby (which runs on the Java Virtual Machine), Rubinius (an alternative implementation written in Ruby and C) and the standard reference C implementation, which is now on stable version 1.9. Because of these characteristics, Ruby is an increasingly popular programming language, and has been among the most popular interpreted languages (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html). One factor that may hinder the dissemination of Ruby, especially among academic and technological communities, is the fact that it does not contain built-in methods for statistical analysis and graph creation. In the past few years, Ruby libraries (gems) for statistical analysis have been created (*i.e.*, RSRuby, RinRuby and Statsample). Statsample[1] has a limited number of statistical methods, while RSRuby (Gutteridge, 2008) and RinRuby (Dahl

1 ruby-statsample.rubyforge.org/

Table 1. Examples of R and RSRuby syntaxes for statistical tests.

Statistical test	R	RSRuby
t-test	t.test(a,b)	r.t _ test(a,b)
Correlation test	cor(a,b, method = "spearman")	r.cor(a,b, :method => "spearman")

and Crawford, 2009) integrate Ruby with R. R is a scripting language and environment developed for statistical computing, with outstanding capacities for graphics generation (R Development Core Team, 2013). It has an extensive library of routines, with hundreds of contributors, it has been heavily used and is widely accepted by the scientific community.

The fact that RSRuby is a C extension for Ruby makes it much faster than RinRuby, which is 100% Ruby implemented. RSRuby, however, has some disadvantages, as: i) it is not available for alternative implementations of Ruby (*e.g.*, JRuby); ii) it is dependent on operating system, Ruby implementation and R version; and iii) downloading may not be trivial for people with no formal training in informatics. The main drawback of RSRuby is that it does not have the full capacity of R (*i.e.*, *p* values for correlation analysis cannot be directly obtained), and the transformations between R and Ruby are not trivial, as in many cases the methods for statistical tests have quite different syntaxes (Table 1).

RinRuby and R methods have the same syntax, and all the parameters from statistical analysis can be transformed in Ruby objects. In fact, RinRuby seems to be the less complicated bridge between R and Ruby. RinRuby, however, requires the assignment of variables to connect Ruby to R, a procedure that has to be repeated many times in complex codes. Limited by the factors listed above, the libraries for statistical analysis with Ruby have not gained much popularity, and it seems that the most common procedure to perform statistics from data generated by Ruby scripts is the storage of data in files, and later access of files through the R (or other statistical packages) console or command line (Chang, 2012). This approach, however, is time consuming and may not be feasible in high-throughput analysis of data, where hundreds or thousands of statistical tests have to be performed on a given data-set.

Ruby has the capability to execute an R script from within a Ruby script. This can be achieved using the Ruby system method (system("data")) to run an R code using the batch mode (R CMD BATCH script.R). Perhaps the easiest procedure would be to create a file (.csv or .txt) with the numeric data generated by a Ruby code, and subsequently to run an R script using batch processing (system("R CMD BATCH script.R")). In this approach, two separate scripts are created. The first is a Ruby script that generates and stores data in a file, where numeric values for each group or treatment are stored in distinct columns. Execution is then transferred to an R script that contains the commands for the statistical analysis. In our view, this is the simplest way to link Ruby and R. One possible drawback for this approach is the creation of the file to be accessed by the R script, which can delay the execution of the script, especially in files generated from very large data-sets.

In this report, we demonstrate the use of the Ruby method for statistical analysis of large-scale human coding sequences with R, and compare its time performance with RinRuby.

Methods and Scripts

Procedures

The scripts were run on Ruby version 1.9.3[2], R version 2.14.2[3] and Linux operating system Ubuntu version 12.04[4]. The performances of the Ruby system and RinRuby were compared analysing 29,064 human coding sequences from the The Consensus CDS (CCDS) project[5]. Sequences were stored in a single file (CCDSfinal.txt) in FastA format. The sequences were analysed as follows:

1. the CCDSfinal.txt file was opened, and each coding sequence was sequentially read;
2. the size, number of bases (A, C, G, T) and CpGs (cytosine followed by guanine) of the coding sequences were printed to a .csv file (Ruby system approach) or stored in arrays (RinRuby) (Supplementary file 1[6]);

2 www.ruby-lang.org/en/downloads
3 www.r-project.org
4 www.ubuntu.com/download
5 www.ncbi.nlm.nih.gov/projects/CCDS
6 journal.embnet.org/index.php/embnetjournal/article/downloadSuppFile/753/969

3. the data from the .csv files and arrays were used to plot frequency charts of bases (A, C, G and T) versus the frequency of CpGs. The Spearman rank correlation coefficients between coding sequence size and number of CpGs, and the respective p-values, were also calculated.

The time required to accomplish these procedures was obtained by the mean of five program runs, and was measured using the Ruby benchmarking method. The scripts are reported below.

Ruby system approach

Ruby script

```
File.open(“results.csv”,”w”)
def countCpG(seq)
       count = 0; cpg = 0
       while count < seq.size - 1
              cpg += 1 if (seq[count] +
seq[count +1]) == “CG”
              count += 1
       end; cpg
end

File.open(“CCDSfinal.txt”,”r”).each do
|line|
       unless line[0] == “>”
              File.open(“results.
csv”,”a”) do |f|
              f.print line.chomp.
size,”,”,line.count(“A”),”,”,line.
count(“C”),”,”,line.count(“G”),”,”,line.
count(“T”),”,”,countCpG(line),”\n”
              end
       end
end
system(“R CMD BATCH statistics.R”)
```

R script

```
file =read.csv(“results.csv”)
   par(mfrow = c(2,2))
   par(mar= c(4.5,5,4,4))
   bases = c(“A”,”C”,”G”,”T”)
for(i in 2:5){
   plot(file[,i]/
file[,1],(file[,6]/file[,1]),xlab =
paste(“frequency”,bases[i -1]), ylab =
“frequency CpG”, col = “red”,cex.lab = 2)
   lines(loess.smooth(file[,i]/
file[,1],file[,6]/file[,1], span= 3/5,
family=”gaussian”), lwd = 2)}
corr = cor.test(file[,6],file[,1],method
= “spearman”)
print(paste(corr$estimate,corr$p.value))
```

Results and Discussion

Figure 1 shows a graph of the frequency of each base (number of specific bases in a coding sequence/size of coding sequence) versus the frequency of CpG (number of CpGs in a coding sequence/size of respective coding sequence). CpG dinucleotides may be enzymatically methylated, and this chemical modification can modulate gene expression (Robertson, 2005). As can be seen, the relationship between CpG and base frequency is not linear.

The processing and statistical analysis of CCDS project coding sequences was completed in an average of 32.32 s with the Ruby system against 37.48 s with RinRuby (t-test p = 1.3e-07). The Ruby approach was also faster when we doubled the genes, or when only half of the genes were analysed (Figure 2). The scripts using the Ruby system approach had 795 characters altogether (Ruby and R scripts) against 1722 characters of RinRuby, which requires the assignment of variables to connect Ruby to R.

The Ruby system approach also works in Windows, where it requires the path to the R directory installation before the R CMD BATCH (*i.e.*, system(“path R CMD BATCH script.R”)) and in Mac OS X system in a way similar to that described here for the Linux operating system. This approach can probably be performed with any

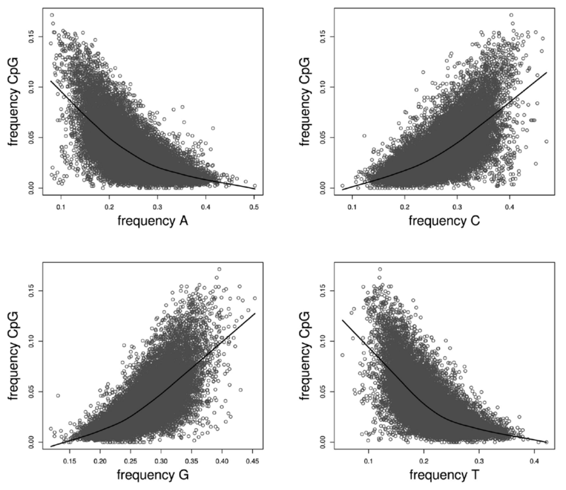


Figure 1. Plot of the frequency of each base (number of specific bases in a coding sequence/size of coding sequence) versus the frequency of CpGs (number of CpGs in a coding sequence/size of its coding sequence). The fitting line was obtained with the LOESS (locally weighted scatterplot smoothing) function of R.

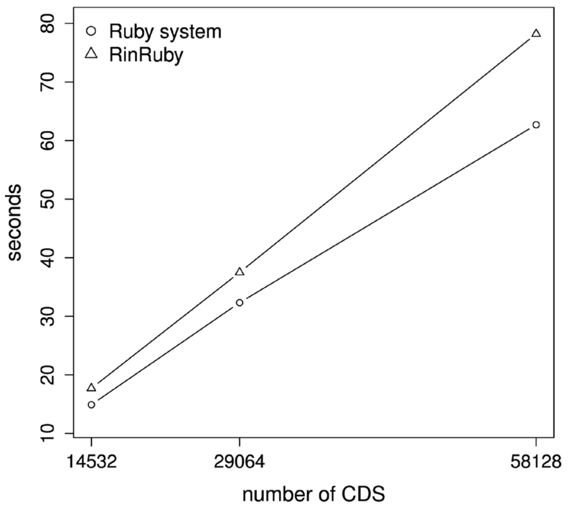


Figure 2. Time performance of the Ruby system and RinRuby approaches. Note that the Ruby system approach is faster than RinRuby, and that the time difference increases as the number of coding sequences increases. Number of CDS = number of coding sequences.

other computer language that has the capabilities to execute R scripts from within its native script.

Our analyses show that the Ruby system can be used for large-scale processing and statistical analysis of DNA sequencing data. In our view, this approach has three main advantages over other procedures: i) it avoids the installation of interface libraries; ii) it is simpler to use, as it does not require the creation of methods or routines, other than those already existent in R and Ruby; iii) the codes are shorter and more readable, as it does not require the assignment of variables to connect Ruby to R. We hope that the simplicity of the approach presented in this paper will incentive the use of Ruby in bioinformatics, as well as in other academic and technology fields, especially by professionals and students with no formal training in informatics.

Acknowledgements

LSM was supported by Fundacao de Amparo a Pesquisa do Estado de Sao Paulo-FAPESP.

References

Aerts J, Law A (2009) An introduction to scripting in Ruby for biologists. *BMC Bioinformatics* **10**, 221. http://dx.doi.org/10.1186/1471-2105-10-221

Chang SS (2012) *Exploring everyday things with R and Ruby.* O´Reilly Media. Sebastopol.

Dahl DB, Crawford S (2009) RinRuby: Accessing the R interpreter from pure Ruby. *J. Statist. Software*, **29**(4), 1-18.

Flanagan D, Matsumoto Y (2008) *The Ruby Programming Language*. O´Reilly Media, Sebastopol.

Gutteridge A (2008) RSRuby: A bridge between Ruby and the R interpreted language. Ruby package version 0.5.1. http://rubyforge.org/projects/rsruby (accessed 7 January 2014).

R Development Core Team (2013) The R project for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. http://www.r-project.org/ (accessed 6 January 2014).

Robertson K (2005) DNA methylation and human disease. *Nat Rev Genet* **6**, 597-610. http://dx.doi.org/10.1038/nrg1655