The Ruby Language in biomedicine: a short review and selected examples

Maciej Goliński[1], **Agnieszka Kitlas Golińska**[1]

[1] Department of Medical Informatics, Institute of Computer Science, University of Bialystok, Poland

Abstract. Ruby is a dynamic programming language – both in relation to the typing discipline and in the increasing number of fields of usage. It may be very useful in biomedical data studies. In this paper we present some facts about the Ruby language together with the example of how to use Ruby in EMG signal analysis. We also review selected applications of the Ruby language in biomedical studies and present our original work. Our goal was to check if in Ruby one can write programs for basic signal analysis (power spectrum based on Fourier transform and energy of signal), which could give accurate results. We also compare computation times using Ruby language and MatLab. We conclude that the Ruby language gives the same results as MatLab and although computation times are worse than in MatLab, we propos using the Ruby language. It is a simple, efficient tool and in contrast to the expensive MatLab – free of charge.

Introduction

Software plays an important role in biomedical studies. There are many programs, which are used, but they are mostly very expensive. One of the widely used programs is MatLab, language of technical computing, developed by MathWorks. We propose the use of the Ruby language – simple, efficient and free of charge tool [10, 13].

Some facts about Ruby language

The Ruby language was designed in Japan in 1995 [13]. Its creator, Yukihiro Matsumoto, attempted a merge of his favourite programming languages, i.e. Perl, Smalltalk, Eiffel, Ada, and Lisp [4]. His objective was a language with a focus on the ease of use by humans, not by computers, which means that it is a language with a very high level of abstraction. Dynamic typing model, in this case "duck typing", makes writing even simpler. The language contains a lot of syntactic sugar [5].

ISBN 978–83–7431–336–0 ISSN 0860-150X

Ruby's main paradigm is object-oriented programming, which is currently the most popular way to write applications. It may also serve as a bridge to functional programming, supposedly the next major programming paradigm. It is one of the few languages that implements full object-oriented paradigm, which means that even numbers and classes are objects [3]. Some people, mostly from the Java community, say it's too much of a generalization, but it is just another property, that makes the language easy to use and general-purpose.

One of the core strengths of Ruby is its reflectivity [13]. All classes are open and can be modified at any moment by a programmer. Ruby supports single inheritance, but programmers can include modules to expand classes' behavior. This style of programming is called a mixin. Metaprogramming is a popular and efficient way to write applications, and it's supported by Ruby [11]. It increases productivity and encourages the programmers to re-use their code. The language is widely used as a scripting language, which helps in OS administration, XML processing or loading CSV files. Ruby can also be embedded in HTML code. It allows creating dynamic, server-side-evaluated web pages. Ruby on Rails is the so-called killer app for Ruby [4], which allows creating functional database-based web applications in a matter of minutes. Ruby may not be the fastest language to compute, but it is time-saving for the programmers.

A short review of application of Ruby language in biomedicine

The Ruby language is very useful in biomedical informatics. The position [1] is an introduction to Ruby for biomedical researchers. It contains a discussion of many applications covering the most common computational tasks in the field of biomedicine.

While browsing many scientific article databases, we could only find that Ruby is generally used as a helper tool, not the main one.

In the paper [12] a Ruby script is used to load 5880 anonymized magnetic resonance in studies (almost 2 million images).

Lim in his article [9] introduces a novel method of introducing bioinformatics to students in a Programming Languages class. Ruby allowed the teacher to present basic concepts of string manipulation in a way that is familiar to programmers.

In the paper [7] the authors present BioRuby, an open-source bioinformatics library as a functional way to simplify work for bioinformatics researchers. The article is an introduction to BioRuby by demonstrating

a few key features of the library, eg. handling FASTA file format, fetching a KEGG graph with the KEGG API, using the interactive environment and availability for all platforms and Java Virtual Machine.

So far, the Ruby language is not often used in the field of medicine. There are a few applications in signal analysis, so we can expect more in the future.

Basics of the Ruby language – how to use Ruby in signal analysis

Ruby is a fully object-oriented language. It means that everything is an object: numbers, strings, nil (empty value), or even classes themselves (contrary to eg. C++ or Java). Thanks to this, it is possible to call methods for any variable. To find out, what is the class of any given object, the method "`class`" can be called. To get a list of methods available for an object, the method "`methods`" may be used.

Examples:

```
number = 5
number.class      #returns the class of a variable, Fixnum here
number.methods    #returns a list of method available for
                                                    the object
nil.class         #returns NilClass
"text".class.clas #returns the class of the String class - Class
```

To print something on screen, we can use one of three instructions: "`print`", "`puts`", or "`p`":

- `print` – simply print a string on screen,
- `puts` – print and add the end of line symbol,
- `p` – print a more detailed information about an object (ie. it calls the "`inspect`" method) and adds the end of line symbol.

Examples:

```
print "Hello!\n"
puts "Hello!"
p "Hello!"
```

The method "`gets`" is used for input. The plus sign is used for concatenation of strings.

Example:

```
print "Input text: "
print "Inputed: " + gets
```

The variables' names in Ruby can consist of letters, numbers, and underline, but must begin with a letter. To assign a value to a variable, the "=" sign is used. There is no need to declare the type of a variable in Ruby.

Example:

```
print "What's your name?"
name = gets
puts "Hello, " + name
```

The variables can be used in operations.

Example:

```
x = 7
y = 3
sum = x + y
puts "The sum: " + sum.to_s
```

To concatenate a string with a number, we must call the "to_s" conversion method.

In Ruby, a table can contain objects of different classes.

Example:

```
table = [3, "a", 2.5, ["x", "y", "z"]]
```

To add another object to a table, the "<<" operator can be used.

Example:

```
table << "object"
```

This operator will append an object at the end of the table.

Code blocks

A code block is an unnamed function. It can be passed as a parameter to a method. A code block is a fragment between curly braces or the keywords "do" and "end". The local variables of a block are declared between two vertical lines. One of the most frequently used method that accepts a block is "each", which is used for iteration of a collection.

Example:

```
tab = [1, 2, 3, 4, 5, 6, 7, 8, 9]
sum = 0
tab.each {|i| sum += i}
puts "The result: #{sum}"
```

Another usage of code blocks is repetition.

This example prints the string three times:

```
3.times {puts "Hello, Ruby!"}
```

Blocks allow code to be loop-free, which makes debugging much easier.

Code block is an easy way of handling files, which in most languages is quite tiresome. Using traditional technique would look something like this:

```
file = File.open("file_name", "r+")
p file
file.close
```

With code blocks it would be much easier and shorter:

```
File.open("file_name") {|file| p file}
```

There is an additional advantage to this method. We don't have to remember to close the file, because Ruby will do it for us. To get a line of the contents of a file, the method "`gets`" may be called. This method, however, returns a string, so we have to convert it to a float using "`to_f`". Then the "`while`" loop would be useful to get the entire contents, line by line.

Application of code blocks in our work

In our programs we used code blocks to load the signals:

```
data=[]
File.open("signal.txt") do |f|
  while line=f.gets
    data<<line.to_f
  end
end
```

We have the signal from file signal.txt in a table data. Now it's only a matter of passing this table to the method that performs the discrete Fourier transform. The result is written to another file, in the following way:

```
File.open("signalout.txt", "w+") {|out| out<<dft(data)}
```

Methods in EMG signal analysis using the Ruby language and MatLab

We used two basic, widely used methods in signal analysis – power spectrum based on the Fourier transform (frequency domain analysis) and energy of signal (time domain analysis). We wrote programs for these methods in Ruby (version 1.9) and MatLab (version 2006).

Power spectrum is based on the Fourier transform. In MatLab we used built-in "`fft`" function (fast Fourier transform algorithm) for the Fourier transform, which is implemented on environment level. In the Ruby language

there aren't any built-in functions for signal processing and analysis (only basic mathematical functions), so we wrote a program for Fourier transform ourselves using definition of Fourier transform – our written function "`dft`" and also – to compare computing time – "`fft`" function (fast Fourier transform algorithm). Both in Matlab and in Ruby we calculated energy using simple iterators and arithmetic operators.

Power spectrum

Power spectrum is defined as [2, 14]:

$$P_{xx}(\omega) := |\hat{x}(\omega)|^2 = \hat{x}(\omega)\overline{\hat{x}(\omega)} \tag{1}$$

where x is complex conjugate of x and

$$\hat{x}(\omega) := \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt \tag{2}$$

is the Fourier transform. It is very important transform in signal processing and analysis, because it yields the information about frequencies occurring in signals and we can compute the dominant frequency for signals using this method.

In our experiment we used a MatLab function "`fft`", which calculates the Fourier transform using the fast Fourier transform algorithm:

```
s=load('e2.txt');
x=fft(s);
```

and a Ruby function "`dft`" (written by us) to calculate the discrete Fourier transform:

```
File.open("signal.txt", "w+") {|out| out << dft(data)}
```

Then (in both languages) we calculated the absolute value squared, using built-in functions.

Signal energy

Signal energy is also an important parameter in signal processing and analysis. It is defined as in [14]:

$$E_x := \int_{-\infty}^{\infty} x^2(t)dt \tag{3}$$

Using energy of signal we can classify signals and this is a basic parameter of signal.

In the Ruby application, the energy of a signal was calculated by writing a method "`energy`" which uses simple iterators and arithmetic operators:

```
def energy(signal)
  signal2=signal.map {|x| x*x}
  signal2.inject(0) {|sum, x| sum+x}
end

data=[]
File.open("signal.txt") do |f|
   while line=f.gets
     data<<line.to_f
   end
end
puts energy(data)
```

MatLab is a language created for matrix-handling, so the code was easy to write:

```
s=load('signal.txt');
sum(s.^2)
```

Both, with the signal energy and the power spectrum, the signals were loaded from a file, and the result was written to a file.

Selected EMG signals

We selected three EMG recordings for our studies: from a healthy subject, one with myopathy and one with neuropathy. EMG records were obtained using 25 mm concentric needle electrode placed in tibialis anterior muscle. Subjects dorsiflexed the foot gently against resistance and then relaxed. All signals were obtained from Physionet [6]. Every signal had 8192 samples.

Results of EMG signal analysis using the Ruby language and MatLab

We obtained exactly the same results using MatLab and the Ruby language for dominant frequency (see [Tab. 1]). Also dominant frequency for all three EMG signals was very low. We couldn't distinguish between these signals using this Fourier based method. So Fourier transform based methods aren't perfect, especially for these biomedical signals, where we have frequently nonlinearity, nonstationarity and low frequency. In modern literature on biomedical signal analysis there are voices that suggest new methods, based on nonlinear dynamics [8]. Nevertheless, Fourier transform based methods are frequently first selected methods in any signal analysis, so we also want to present our results. In the future we plan to used

some of the nonlinear methods (among others from chaos theory and fractal geometry) to these EMG signals to find out more about their nature.

Tab. 1. Values of dominant frequency for selected EMG signals

Selected EMG signal	Dominant frequency (Hz)
Healthy subject	0.063
Subject with neuropathy	0.062
Subject with myopathy	0.058

In [Tab. 2] we present obtained values of signal energy (exactly the same results using MatLab and Ruby). The highest values are for a signal from the subject with neuropathy, the lowest – for a signal from a healthy subject. Using signal energy we may distinguish a healthy subject from unhealthy subjects. Here we can see that in our pathological cases the signal is strengthened.

Tab. 2. Values of signals energy for selected EMG signals

Selected EMG signal	Values of signal energy
Healthy subject	21.485
Subject with neuropathy	417.070
Subject with myopathy	39.672

As see above, we have obtained some results which may suggest that Fourier transform based methods are not adequate to these signals and that in our pathological states energy of the biomedical signal is rising. Of course, these results need confirmation on larger groups of subjects, therefore we are very careful in our conclusions. Here, we have focused on the application of the Ruby language, but in the future we plan to investigate these matters more thoroughly.

Comparison of computation times using Ruby language and MatLab

In [Tab. 3] and [Tab. 4] we present a comparison of computation times of the Fourier transform and signal energy programs executed in MatLab and implemented in the Ruby language.

Tab. 3. Comparison of computation times of the Fourier transform using MatLab built-in "`fft`" function and our written "`fft`" function in the Ruby language (our signal length presented in bold font)

Signal Length (samples)	Computation time (s)	
	Ruby	MatLab
0	0.000	0.00000065
256	0.015	0.00027823
512	0.034	0.00000660
1024	0.081	0.00001123
2048	0.156	0.00002081
4096	0.329	0.00004239
8192	**0.699**	**0.00013816**
16384	1.526	0.00043267
32768	3.311	0.00108538
65536	7.066	0.00314686

Tab. 4. Comparison of computation times of signal energy using MatLab and the Ruby language (our signal length presented in bold font)

Signal Length (samples)	Computation time (s)	
	Ruby	MatLab
0	0.000000	0.0000013
256	0.000094	0.0000035
512	0.000187	0.0000045
1024	0.000421	0.0000054
2048	0.000952	0.0000099
4096	0.001591	0.0000134
8192	**0.002948**	**0.0000234**
16384	0.005616	0.0000446
32768	0.011092	0.0001032
65536	0.024242	0.0003448

Computation times dependency is nearly linear in all considered cases for commonly used signal lengths. Although Ruby's execution time seems poor in the comparison, it should be remembered that it is an interpreted

language, while MatLab is compiled and highly optimized. And last, but not least, Ruby is free of charge, while MatLab is an expensive tool.

Conclusions

We have concluded that the Ruby language gives the same results as MatLab and although computation times are worse than in MatLab, we propose using the Ruby language. It is a simple, efficient and – in contrast to expensive MatLab – free of charge tool. The fact, that it is a second slower (for the common signal sizes) is not that much of a problem for scientific usage. The delay is negligible. In the future we plan to create Biomedical Signal Analysis Library in Ruby and we expect more applications of Ruby in biomedicine studies generally.

REFERENCES

[1] Berman J. J., Ruby Programming for Medicine and Biology, Jones & Bartlett Pub, 2007.

[2] Challis R. E., Kitney R. I., Biomedical signal processing (in four parts). Part 3: the power spectrum and coherence function, Medical & Biological Engineering & Computing, 29, pp. 225–241, 1991.

[3] Fitzgerald M. J., Learning Ruby, O'Reilly Media, Sebastopol, 2007.

[4] Flanagan D., Matsumoto Y., The Ruby Programming Language, O'Reilly Media, Sebastopol, 2008.

[5] Fulton H., The Ruby Way, Second Edition: Solutions and Techniques in Ruby Programming, Addison-Wesley Professional, Boston, 2006.

[6] Goldberger A. L., Amaral L. A. N., Glass L., et al., PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals, Circulation, 101 (23), pp. e215–e220, 2000. [Circulation Electronic Pages; http://circ.ahajournals.org/cgi/content/full/101/23/e215]

[7] Goto N, Prins P., Nakao M., et al., BioRuby: bioinformatics software for the Ruby programming language, Bioinformatics, 26 (20), pp. 2617–2619, 2010.

[8] Klonowski W., Application of new non-linear dynamics methods in biosignal analysis, Proceedings of the World Medical Conference, Prague, Czech Republic, 26–28.09.2011, pp. 180–187, 2011.

[9] Lim D., A Ruby in the rough: using VHLLs in bioinformatics, Journal of Computing Sciences in Colleges, 21 (6), pp. 108–116, 2006.

[10] Olsen R., Eloquent Ruby, Addison-Wesley Professional, Boston, 2011.

[11] Perrotta P., Metaprogramming Ruby: Program Like the Ruby Pros, Pragmatic Bookshelf, 2010.

[12] Rascovsky S. J., Delgado J. A., Sanz A., et al., Informatics in Radiology: Use of CouchDB for Document-based Storage of DICOM Objects, Radiographics, 32 (3), pp. 913–927, 2012.
[13] Thomas D., Fowler Ch., Hunt A., Programming Ruby 1.9: The Pragmatic Programmers' Guide, Pragmatic Bookshelf, 2009.
[14] Zieliński T. P., Cyfrowe przetwarzanie sygnałów: od teorii do zastosowań, Wydawnictwa Komunikacji i Łączności, Warszawa, 2005.