Unit and Performance Testing of Scientific Software Using MATLAB®

BOJANA KOTESKA, MONIKA SIMJANOSKA, IVANA JACHEVA, FROSINA KRSTESKA and ANASTAS MISHEV, Ss. Cyril and Methodius University

In this paper we report our activities on performing testing of scientific software for calculating ECG-derived heart rate (HR)and respiratory rate (RR) by using Matlab®. The aim of this software is to aid the triage process in the emergency medicine which is crucial for ranging the priority of the injured victims in mass casualty situations based on the severity of their condition. One challenge of this paper is to perform unit testing in Matlab by using the Input Space Partitioning method. For that purpose, we created sets of test values for the ECG signals and we modified them according to our needs. By using the Profiler tool we tested the performance of the algorithm functions.

1. INTRODUCTION

Scientific software solves problems in various (scientific) fields by applying computational practices [Kelly et al. 2008]. Its multidisciplinary nature makes it more complex, but it provides great opportunities and advantages for scientists in many different scientific fields. Scientists usually have a large amount of data to process [Wilson et al. 2014], many calculations, lots of requests to handle, and automating the process by creating software makes their work easier, increases productivity, quality and sustainability [Wiedemann 2013].

Scientific software is based on models, experimentation and observation of the results [Joppa et al. 2013]. Tests are very much like experiments and the obtained results are observed later. That is how the scientists test their hypotheses. They run experiments, measure results and analyze the data.

Scientific software testing is a hard and challenging task due to the complexity and the lack of test oracles [Kanewala and Chen 2018; Lin et al. 2018]. Challenges are categorized according to the specific testing activities: test case development, producing expected test case output values, test execution, test result interpretation, cultural differences between scientists and the software engineering community, limited understanding of testing process, not applying known testing methods, etc [Kanewala and Bieman 2014].

In this paper, we report our activities on performing testing of a scientific software for calculating ECG-derived heart rate (HR) and respiratory rate (RR) by using Matlab®. We try to identify specific challenges, proposed solutions, and unsolved problems faced when testing scientific software. The aim of this software is to aid the triage process in the emergency medicine which is crucial for ranging the priority of the injured victims in mass casualty situations based on the severity of their condition

This work is supported by the Faculty of Computer Science and Engineering, Skopje, North Macedonia.
Ivana Jacheva, Bojana Koteska, Frosina Krsteska, Monika Simjanoska, and Anastas Mishev, FCSE, Rugjer Boshkovikj 16, P.O. Box 393 1000, Skopje; email: ivana.jaceva@students.finki.ukim.mk; bojana.koteska@finki.ukim.mk; frosina.krsteska@students.finki.ukim.mk; monika.simjanoska@finki.ukim.mk; anastas.mishev@finki.ukim.mk.

In: Z. Budimac and B. Koteska (eds.): Proceedings of the SQAMIA 2019: 8th Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, Ohrid, North Macedonia, 22–25. September 2019. Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

[Hogan and Brown 2014], whether they are man-made, natural or hybrid disasters. When performed manually, an efficient triage process takes less than 30 seconds. In order to optimize the process when there are hundreds of injured people, the challenge is to reduce the triage time and the number of medical persons needed. The software we describe in Section 2 extracts heart rate and respiratory rate from ECG signal. This optimization can be achieved by using the benefits of the biosensor technologies to extract the vital signs needed for the triage.

The other sections are organized as follows. Section 3 provides a comprehensive explanation of the testing methodology and results: definition of test cases, testing preparations and execution and analyses of the results of the executed tests. The final Section 4 concludes the paper.

2. DESCRIPTION OF SOFTWARE FOR CALCULATING ECG-DERIVED HR AND RR

The software is developed as a part of the triage procedure for determining a patient's condition [Gursky and Hrečkovski 2012]. It is designed for low-power wearable biosensor and it uses only an ECG signal to estimate automatically the HR and RR as crucial parts of the primary triage. The goal of this software is to perform efficient real-time processing of ECG data in terms of the power-demanding Bluetooth connection with the biosensor and the data transmission. The accuracy of the algorithm is published in [Simjanoska et al. 2018].

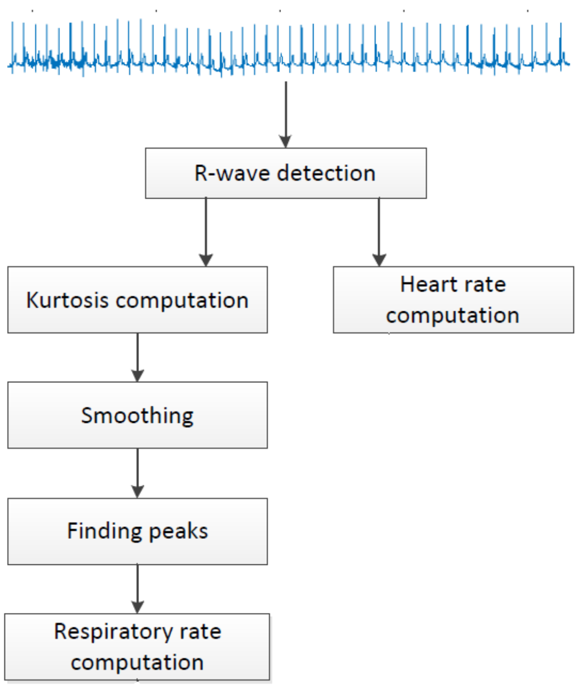


Fig. 1. Software Methodology

As depicted in Fig. 1, the input of the algorithm is a raw ECG signal. The calculation of the HR is performed by R-peak detection performed by using the the Pan Tompkins algorithm [Pan and Tompkins 1985]. HR is calculated according to the following equation:

$$HR = (\frac{signal\_length}{number\_of\_R\_peaks})^{-1} * 60 \tag{1}$$

The obtained R-peaks are used to estimate the RR. The kurtosis computation technique is used for measuring the peakedness of the signal's distribution. The locations of the local maxima are needed for the smoothing method upon which a peak finder method is applied to find the local maxima (peaks) of the ECG signal. The peaks represent a number of respirations according to which the respiratory rate is calculated:

$$RR = (\frac{signal\_length}{number\_of\_respirations})^{-1} * 60 \tag{2}$$

The implementation of the proposed algorithm is realized in Matlab. It has 493 lines of code in total.

The software contains two main functions: *h3r* and *pan_tompkin*. The first one, *h3r*, has two arguments needed for the calculation of the estimated values of RR and HR. The first argument is a raw ECG signal, represented in vector format, which contains an array of decimal values and the second argument is the measurement frequency (number of measurements in a second).

The second function, *pan_tompkin*, uses three arguments for calculating the qrs_amp_raw - amplitude of R-waves, qrs_i_raw - the index of R-waves and delay - a number of samples in which the signal is delayed due to the filtering. The arguments used by *pan_tompkin* are ECG - raw ECG vector signal, fs - sampling frequency and gr-flag - flag for plotting.

3. METHODOLOGY AND RESULTS

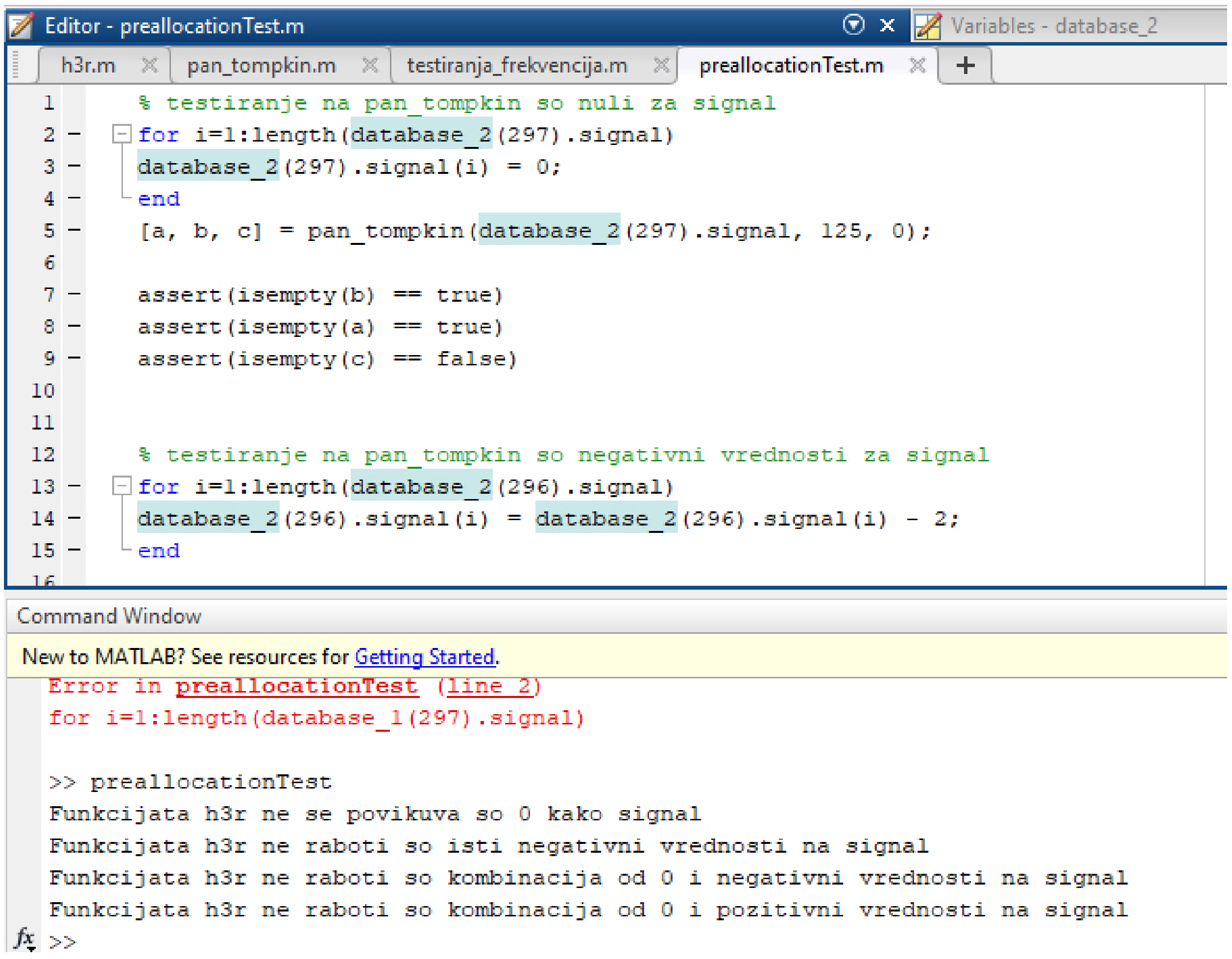
3.1 Unit Testing

h3r and *pan_tompkin* functions are the two main software units. Unit testing purpose is to assess the software units produced by the implementation phase. It represents the "lowest" level of testing [Ammann and Offutt 2016]. Unit testing improves the quality of science and engineering software and it focuses on small units of code (class, module, method, or function). In order to have effective unit testing, the procedure should be automated so that entire test suites can be run quickly and easily. The verifying of the individual units is helpful for verifying overall system behavior. The focus on the smaller software part leads to modular and more maintainable code [Eddins 2009].

When testing a scientific software, it is very important to think about the nature of the software and to be familiar with the scientific field. The metrics should be well defined and most often it is easily noticed that they are surreal. For example, a respiratory rate can't be 1000 breaths per minute. That's why the testers should know the possible result value ranges, so that they can determine the input domain. The input domain should consists of as much as possible inputs (valid and invalid) that could be taken by the program. The tester should choose test cases wisely since the input values could be infinite. Program correctness can be proved by testing all possible input values, but we can only test limited set of inputs (known as test cases). One method to determine the inputs for a specific variable is to use Input Space Partitioning (ISP) - input or output data is grouped or partitioned into sets of data that we expect to behave similarly and will help us with creating test cases. We used Functionality-based ISP [Ammann and Offutt 2016].

Matlab provides multiple tools for unit testing [The MathWorks, Inc 2019].

h3r function's first argument is the raw ECG signal vector. According to the database of ECG signals, the usual values are floating points values in range 0-10. The purpose of ISP is to create partitions of the input values and to test the software behavior when the input values are outside the usual range also. For the ECG vector, the following test cases were tested:



```
% testiranje na pan_tompkin so nuli za signal
for i=1:length(database_2(297).signal)
database_2(297).signal(i) = 0;
end
[a, b, c] = pan_tompkin(database_2(297).signal, 125, 0);

assert(isempty(b) == true)
assert(isempty(a) == true)
assert(isempty(c) == false)


% testiranje na pan_tompkin so negativni vrednosti za signal
for i=1:length(database_2(296).signal)
database_2(296).signal(i) = database_2(296).signal(i) - 2;
end
```

```
Error in preallocationTest (line 2)
for i=1:length(database_1(297).signal)

>> preallocationTest
Funkcijata h3r ne se povikuva so 0 kako signal
Funkcijata h3r ne raboti so isti negativni vrednosti na signal
Funkcijata h3r ne raboti so kombinacija od 0 i negativni vrednosti na signal
Funkcijata h3r ne raboti so kombinacija od 0 i pozitivni vrednosti na signal
>>
```

Fig. 2. Unit Test Results

—empty vector;

—vector of zeros;

—vector with negative float values;

—vector with positive floating values;

—vector with positive floating values greater than 10;

—vectors with combinations of negative values and 0s;

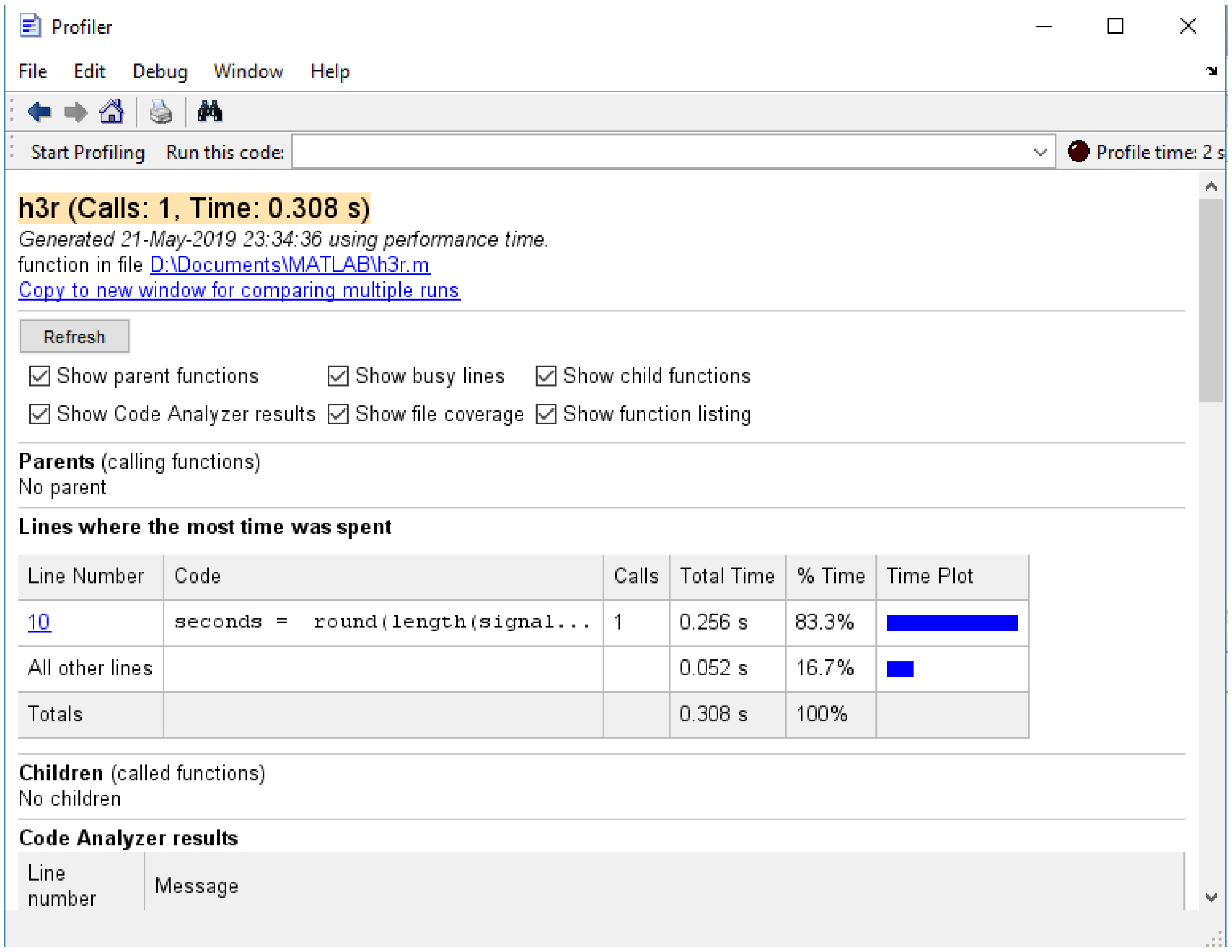
—vectors with combinations of positive values and 0s;

—vectors with combinations of values (negative, positive and 0).

We used the same strategy to test the other function as well, because both functions use the same type of argument - the raw ECG signal vector. For the frequency parameter and for the gr flag we also made ISP (for the frequency we have the correct value of 125, then 0, negative number and value greater than 125). For the gr flag we have 0, 1, negative value and value greater than 1. Each one of these checks represents a single test, which can be run separately.

In Matlab it is possible to define unit test suites. We combined all the tests in one file (test suite), and we ran the file once which ran all the tests automatically one by one.

By using the assertion functions from the matlab.unittest.qualifications.Assertable class, we compared the output HR and RR values with the expected ones and if the test passes, that means the assertion is true and the values from the test are the values we've been expecting to get.



h3r (Calls: 1, Time: 0.308 s)
Generated 21-May-2019 23:34:36 using performance time.
function in file D:\Documents\MATLAB\h3r.m
Copy to new window for comparing multiple runs

☑ Show parent functions ☑ Show busy lines ☑ Show child functions
☑ Show Code Analyzer results ☑ Show file coverage ☑ Show function listing

Parents (calling functions)
No parent

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
10	`seconds = round(length(signal...`	1	0.256 s	83.3%	■■■■■
All other lines			0.052 s	16.7%	■
Totals			0.308 s	100%	

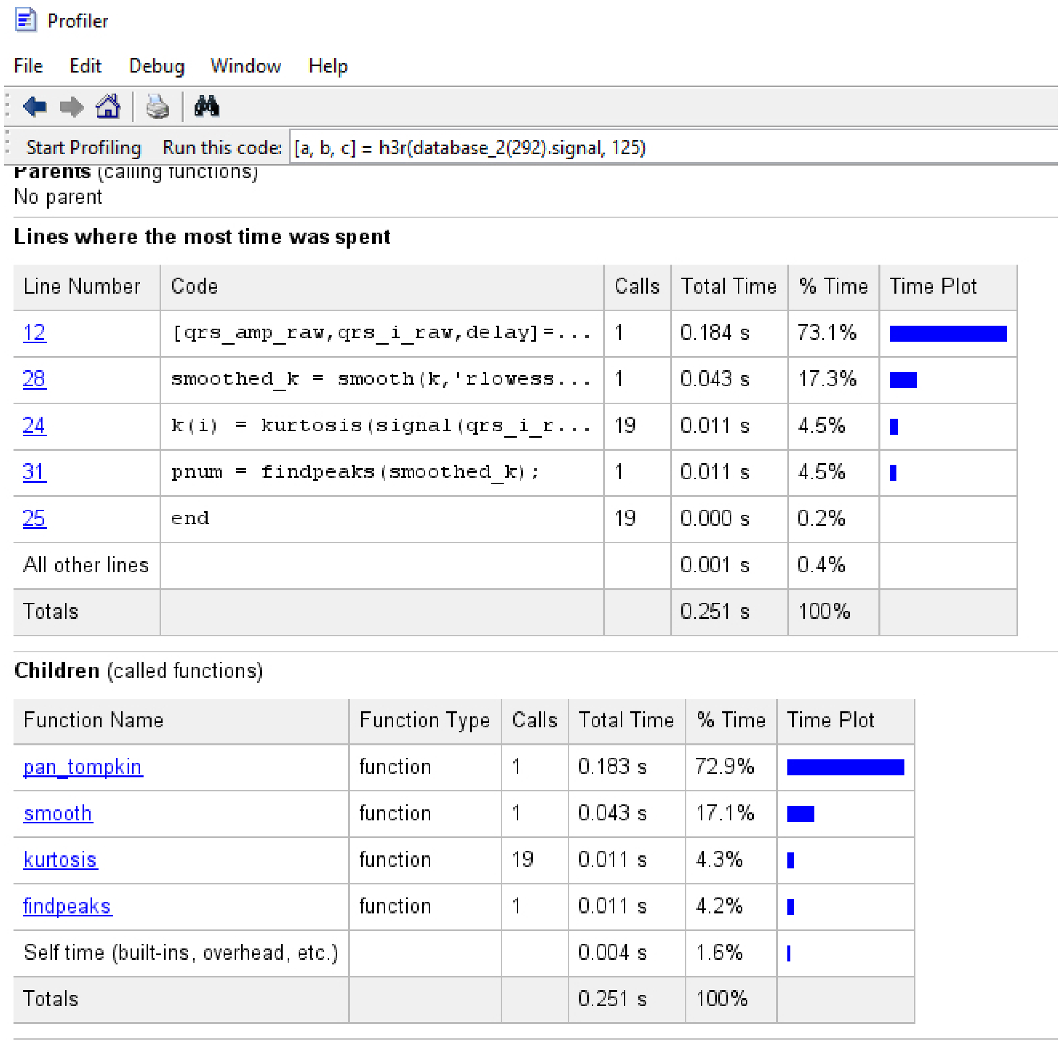
Children (called functions)
No children

Code Analyzer results

Line number	Message

Fig. 3. Performance testing of the hr3 function.

To make test running easy, Matlab provides function *runtests* which runs all the tests in the current folder, gathers them into a test suite, runs the test suite and returns the results as a TestResult object. In our project, we have several test files: one with the test cases for signal vector values, one for testing the functions with different values for frequency, etc. Figure 2 shows the output from the first test file - preallocationTest. As the figure shows, the output is shown in the Command Window. In Matlab, if the test case passed, it doesn't show any output. So, the output shown here is only by the test cases that failed. The output is presented in details, telling where and why the test case failed: function *h3r* can not provide results for signal contains only 0s, it does not work with negative values, combination of 0s and negative values, combination of 0s and positive values, etc.


Profiler
File Edit Debug Window Help
Start Profiling Run this code: [a, b, c] = h3r(database_2(292).signal, 125)

Parents (calling functions)
No parent

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
12	`[qrs_amp_raw,qrs_i_raw,delay]=...`	1	0.184 s	73.1%	▇▇▇▇▇▇
28	`smoothed_k = smooth(k,'rlowess...`	1	0.043 s	17.3%	▇
24	`k(i) = kurtosis(signal(qrs_i_r...`	19	0.011 s	4.5%	▮
31	`pnum = findpeaks(smoothed_k);`	1	0.011 s	4.5%	▮
25	`end`	19	0.000 s	0.2%	
All other lines			0.001 s	0.4%	
Totals			0.251 s	100%	

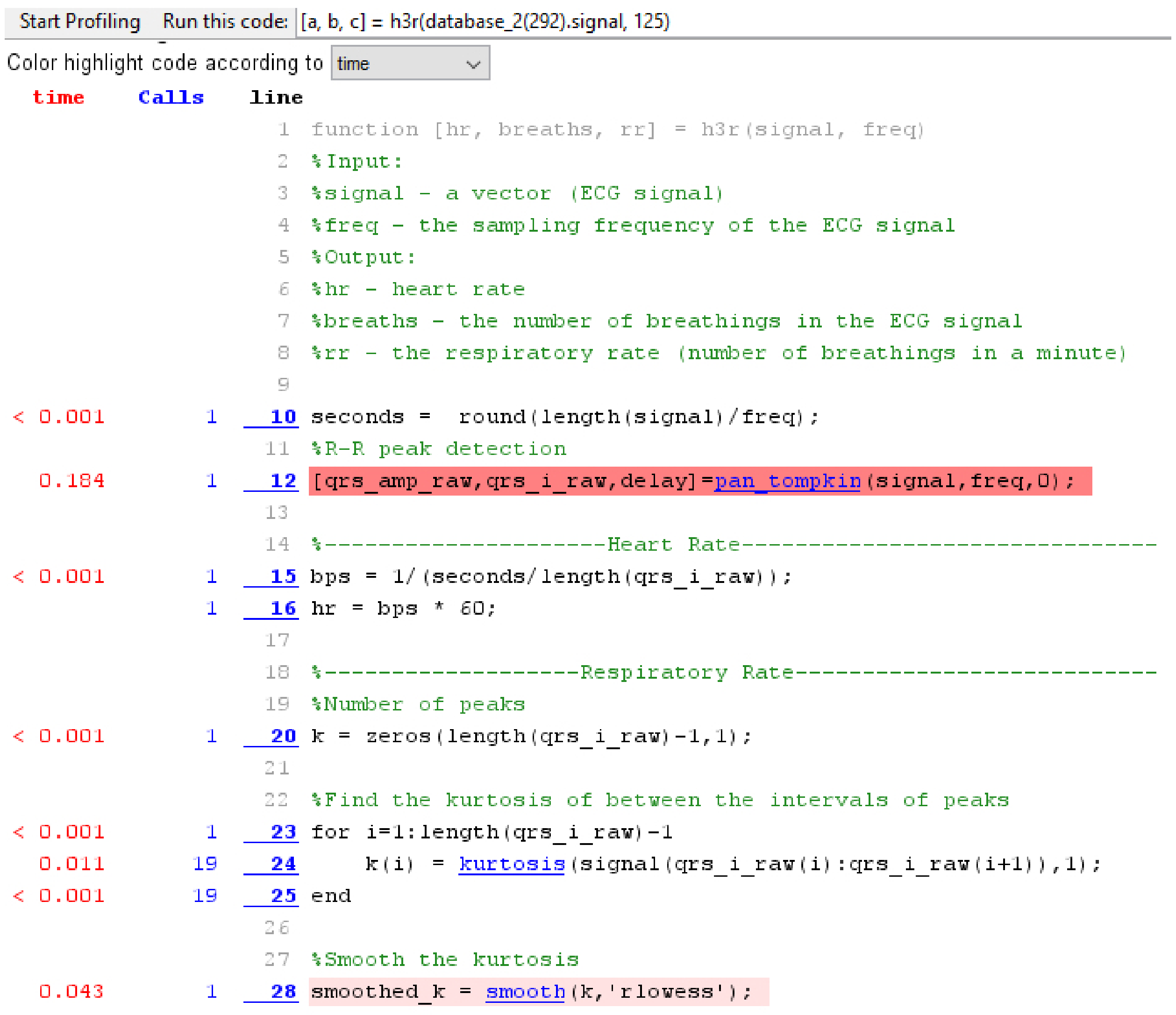
Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
pan_tompkin	function	1	0.183 s	72.9%	▇▇▇▇▇▇
smooth	function	1	0.043 s	17.1%	▇
kurtosis	function	19	0.011 s	4.3%	▮
findpeaks	function	1	0.011 s	4.2%	▮
Self time (built-ins, overhead, etc.)			0.004 s	1.6%	▏
Totals			0.251 s	100%	

Fig. 4. Performance testing of the *pan_tompkin* function.

3.2 Performance testing

Performance testing is a non-functional testing which checks the behavior of the system when it is under significant load. In a case of performance testing the software system is evaluated from a user's perspective, and is typically assessed in terms of throughput, stimulus response time, or both. Performance testing could be used to assess the level of system availability also [Vokolos and Weyuker 1998]. The goal of performance testing is to identify the performance bottleneck, to make comparison



```
[a, b, c] = h3r(database_2(292).signal, 125)
Color highlight code according to time
```

```
  time      Calls   line
                      1  function [hr, breaths, rr] = h3r(signal, freq)
                      2  %Input:
                      3  %signal - a vector (ECG signal)
                      4  %freq - the sampling frequency of the ECG signal
                      5  %Output:
                      6  %hr - heart rate
                      7  %breaths - the number of breathings in the ECG signal
                      8  %rr - the respiratory rate (number of breathings in a minute)
                      9
< 0.001         1    10  seconds =  round(length(signal)/freq);
                     11  %R-R peak detection
  0.184         1    12  [qrs_amp_raw,qrs_i_raw,delay]=pan_tompkin(signal,freq,0);
                     13
                     14  %----------------------Heart Rate--------------------------------
< 0.001         1    15  bps = 1/(seconds/length(qrs_i_raw));
                1    16  hr = bps * 60;
                     17
                     18  %--------------------Respiratory Rate----------------------------
                     19  %Number of peaks
< 0.001         1    20  k = zeros(length(qrs_i_raw)-1,1);
                     21
                     22  %Find the kurtosis of between the intervals of peaks
< 0.001         1    23  for i=1:length(qrs_i_raw)-1
  0.011        19    24      k(i) = kurtosis(signal(qrs_i_raw(i):qrs_i_raw(i+1)),1);
< 0.001        19    25  end
                     26
                     27  %Smooth the kurtosis
  0.043         1    28  smoothed_k = smooth(k,'rlowess');
```

Fig. 5. Perfromance evaluation of each code line.

of the performance, etc. Usually, the performance testing is made by using a benchmark - a program or workload designed to be representative of the typical software system usage [Pan 1999].

In the earlier versions of Matlab, in order to measure the code's performance, there was a testing framework, which included different performance measurement-oriented features. The newest version comes with a tool called Profiler, which automates the work. It provides details about the performance of a specific function: how much time did it take to execute the function, the percentage of the summed up time of executing the part of the program which that separate unit used, whether a line has been executed and if so, how many times, the full code coverage etc.

Figure 3 shows the results of the execution time of the *h3r* function. It gives information about the number of calls and total execution time for each children function call.

Figure 4 shows the executing time of the *pan_tompkin* function.

With the Profiler Tool, we can analyse each function call separately and see more detailed information about its executing, like it's shown in Fig. 5. The results show that *pan_tompkin* function took most of the execution time (0.184s or 73.1% of the total execution time).

4. CONCLUSION

Testing of the scientific software must become a standard part of the development process. Not because we only want to implement the correct way of development process as specified in the literature, but also we should consider the fact that many of the scientific software programs are connected to people's health and can be categorized as critical software programs. Many of the testing methods designed for commercial scientific software can be adapted to scientific software testing. Testing should be considered from different aspects also. For example, even if the scientific program code produces accurate results, problems with performance can exist.

In this paper we made unit and performance testing of a scientific software for calculating ECG-derived heart rate (HR) and respiratory rate (RR) designed to aid the triage process in the emergency medicine which is crucial for ranging the priority of the injured victims in mass casualty situations based on the severity of their condition. The testing was done in Matlab.

Multiple unit tests were created to test the functionality of the algorithm. Matlab provides a very user friendly testing interface. Most of the things are fully automated and only function parameters are required for testing. Unit testing and input space partitioning method helped us to find several function errors, especially with test cases with boundary values. For e.g. ECG signal with zeros, signal with negative values and signals with different lengths helped us to add exceptions in the code.

Performance testing helped us to check the execution times of the functions. We performed tests with different signal lengths in order to increase the data load. That was useful to think about the code optimization which is left as our future work.

REFERENCES

Paul Ammann and Jeff Offutt. 2016. *Introduction to software testing*. Cambridge University Press.

Steven L Eddins. 2009. Automated software testing for matlab. *Computing in science & engineering* 11, 6 (2009), 48–55.

Elin A Gursky and Boris Hrečkovski. 2012. *Handbook for Pandemic and Mass-casualty Planning and Response*. Vol. 100. IOS Press.

David E Hogan and Travis Brown. 2014. Utility of vital signs in mass casualty-disaster triage. *Western journal of emergency medicine* 15, 7 (2014), 732.

Lucas N Joppa, Greg McInerny, Richard Harper, Lara Salido, Kenji Takeda, Kenton O'hara, David Gavaghan, and Stephen Emmott. 2013. Troubling trends in scientific software use. *Science* 340, 6134 (2013), 814–815.

Upulee Kanewala and James M Bieman. 2014. Testing scientific software: A systematic literature review. *Information and software technology* 56, 10 (2014), 1219–1232.

Upulee Kanewala and Tsong Yueh Chen. 2018. Metamorphic Testing: A Simple Yet Effective Approach for Testing Scientific Software. *Computing in Science & Engineering* 21, 1 (2018), 66–72.

Diane Kelly, Rebecca Sanders, and others. 2008. Assessing the quality of scientific software. In *First International Workshop on Software Engineering for Computational Science and Engineering*. Citeseer.

Xuanyi Lin, Michelle Simon, and Nan Niu. 2018. Hierarchical metamorphic relations for testing scientific software. In *Proceedings of the International Workshop on Software Engineering for Science*. ACM, 1–8.

Jiantao Pan. 1999. Software testing. *Dependable Embedded Systems* 5 (1999), 2006.

Jiapu Pan and Willis J Tompkins. 1985. A real-time QRS detection algorithm. *IEEE Trans. Biomed. Eng* 32, 3 (1985), 230–236.

Monika Simjanoska, Bojana Koteska, Ana Madevska Bogdanova, Nevena Ackovska, Vladimir Trajkovik, and Magdalena Kostoska. 2018. Automated triage parameters estimation from ECG. *Technology and Health Care* Preprint (2018), 1–4.

The MathWorks, Inc. 2019. Testing Frameworks in Matlab. (2019). https://www.mathworks.com/help/matlab/matlab-unit-test-framework.html

Filippos I Vokolos and Elaine J Weyuker. 1998. Performance testing of software systems. In *Proceedings of the 1st International Workshop on Software and Performance*. ACM, 80–87.

Christin Wiedemann. 2013. Applying the scientific method to software testing. (2013). https://searchsoftwarequality.techtarget.com/feature/Applying-the-scientific-method-to-software-testing

Greg Wilson, Dhavide A Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven HD Haddock, Kathryn D Huff, Ian M Mitchell, Mark D Plumbley, and others. 2014. Best practices for scientific computing. *PLoS biology* 12, 1 (2014), e1001745.