Including GNU Octave in a numerical programming C++ rapid development software context for research models

Including GNU Octave in a numerical programming C++ rapid development software context for research models

E Oanta[1] and A Pescaru[1, 2]
[1]Constanta Maritime University, Faculty of Naval Electro-Mechanics, 104 Mircea cel Batran Street, 900663, Constanta, Romania
[2]Constanta Maritime University, Faculty of Navigation, 104 Mircea cel Batran Street, 900663, Constanta, Romania

E-mail: alex_pescaru@yahoo.com

Abstract. The paper is included in the strategy to use API facilities of CAD/CAE commercial applications and dedicated programming languages, such as GNU Octave, in order to create a set of instruments useful for the rapid development of the hybrid models. In this way, an exploratory investigation is presented regarding the most appropriate methods to use GNU Octave scientific programming facilities in C++ applications. Four software demonstrators are included in the paper, which may be used as an inspirational environment. Each program presents relevant aspects regarding the C++ API. The first demonstrator presents two methods to start the Octave interpreter, depending on its current version, and the use of the 'feval()' function for 'gcd()', 'acos()' and 'inv()'. The type of the input and output values of 'feval()' are 'octave_value_list', being necessary appropriate conversion functions. The second program is dedicated to the built-in-functions, being presented three case studies. The third demonstrator presents the conversion of the values from a user defined C++ type to the Octave types, i.e. the conversion of a two dimensional array into a 'Matrix' Octave type,which is tested using the 'norm()' function.The fourth program uses 'source_file(OCT_script_file)', which executes the commands in the 'OCT_script_file'. The content of this file is written from a string and it is either a set of Octave commands, or the name of another Octave script file which is executed. We have solutions to minimise the access time spent due to the frequently write operation on the disk. We plan to continue our tests in order to find the best methods to use the GNU plotting facilities to develop graphical interpreters.

1. Introduction

The research of the complex phenomena requires creative investigation approaches which maximise the accuracy of the results and minimise the expenses. These objectives are easily fulfilled if a hybrid model of the phenomenon is conceived. In this way, a hybrid model consists of theoretical sub-models which may be either analytical, or numerical; the calibration and the validation of the theoretical models' results and the verification of their accuracy may be achieved using experimental studies, [1], which are an intrinsic part of the hybrid model. The various studies of a hybrid model require both, dedicated solvers and interfaces between the studies. One can notice that solvers based on analytical or general numerical methods may be used in any sub-model of any nature, theoretical or experimental. The key factor in the easily development of the hybrid models is to create in advance the necessary software components, which may be regarded as 'Lego' pieces, [2], to be used at a later stage in various 'construction games', i.e. creation of computer aided original models. A particular aspect

regards the large flexibility of the general numerical methods, whose algorithms may be used as an inspiring environment for the development of particular solvers. For instance, the computing of the linear expansion coefficient of an aluminium alloy cylinder block and of the temperatures in some measurement points, [3], were solved using a procedural model based on the bisection method. The approximation based on spline functions was used to deduce the laws of variation of the isostatic curves, starting from the isoclinic curves in photoelasticimetry, [4].

2. Hybrid models computing context

The general directions to be followed in order to create a large set of instruments for the rapid development of the hybrid models are: atomization, 'librarization', parameterization, automatization and integration, each of them being presented in [2].

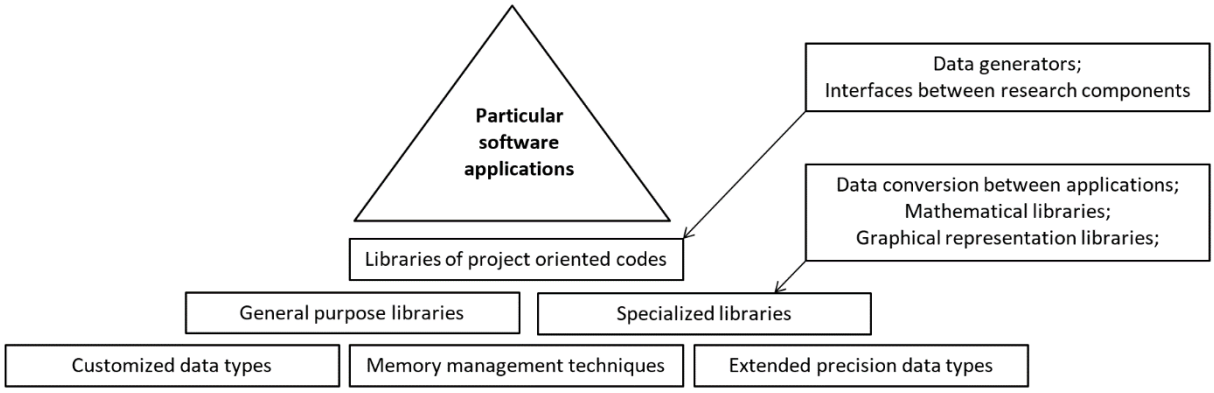

Figure 1. Hierarchy of hybrid models' software components in mechanical engineering, [5].

The hierarchy of software components is presented in figure 1, the general numerical methods libraries, i.e. the 'mathematical' libraries being included in the 'specialized libraries' type of components.

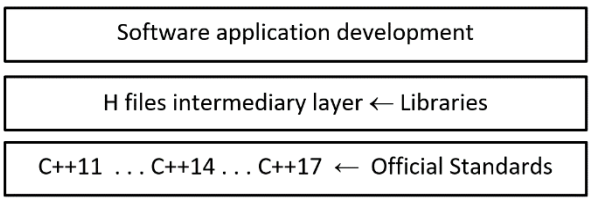

Figure 2. Usefulness of the intermediary layer of C++ header files, [5].

The libraries are implemented as header files, in this way being reached a high degree of reusability. Moreover, the high level applications either remain unchanged, or may have very few modifications when a C++ new standard is considered. In this way the maintenance of the software regards only some of the basic general purpose header files.

In this context, the 'mathematical' libraries consist either of links to external software applications which solve linear algebra problems, such as GNU Octave, or of original solvers which use particular data types, original memory management techniques and arbitrary precision software libraries, as it is presented in figure 1.

Until now we used GNU Octave as a standalone application whose output data is linked to other software components using several techniques: interfaces based on CSV text files, vector graphics and metaprogramming methods. The latest facilities of the spline interpolation data processor are presented in figure 3, [6].

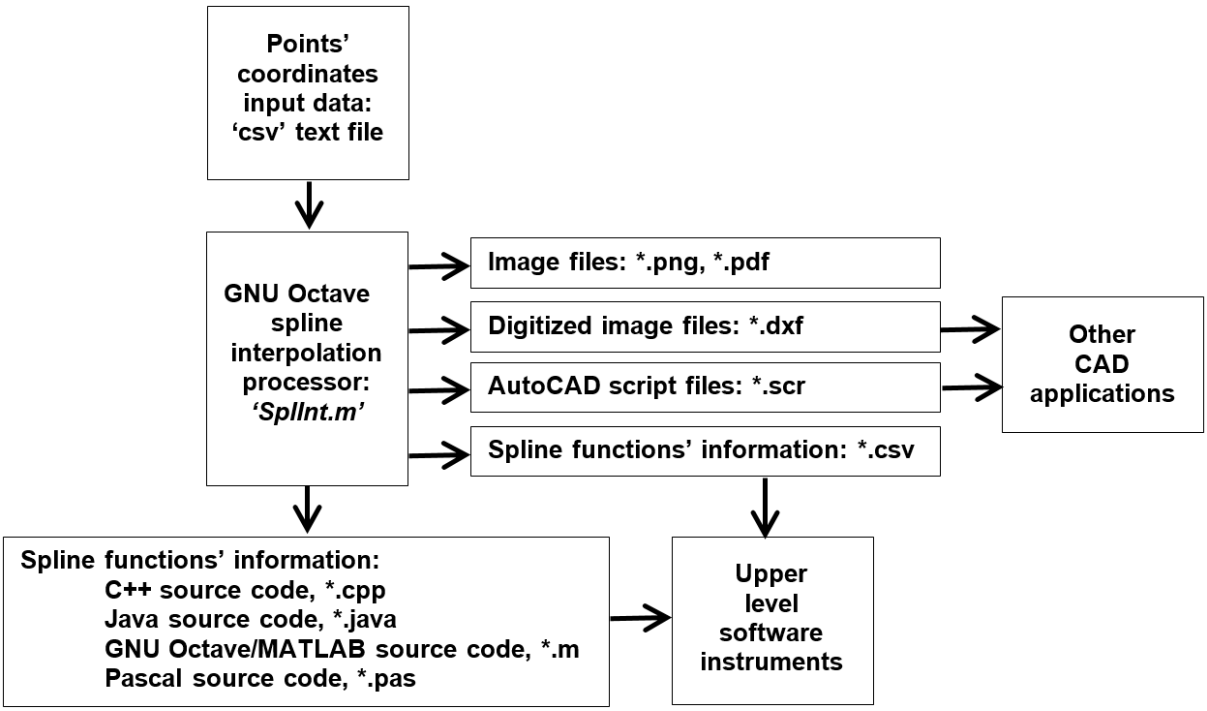

Figure 3. Usefulness of the intermediary layer of C++ header files, [6].

In order to have a higher degree of integration with the hybrid models' software development environment, the C++ API methods are explored.

3. Software demonstrators regarding the use of C++ front end for GNU Octave

In order to acquire knowledge regarding the set of methods to be used to develop a C++ interface to the GNU Octave facilities, several test programs were conceived.

The computer codes have a large number of comments in order to be very explicit and to easily replicate the Eclipse settings in the future projects.

The first computer code presented in appendix A was inspired by [7], and it was modified in order to verify several ways to call the 'feval()' function. The initialization of the Octave interpreter allows the analyst to use script files, oct-files and built-in functions.

The 'echo' flag variable is used to trace the execution. The test program may be run for 3 Octave functions: 'gcd()', 'acos()' and 'inv()'. As it can be noticed, 'feval()' has three input parameters. The first one is a string which represents the name of the Octave function to be tested. The second parameter is of 'octave_value_list' type. The third input parameter is 'nargout'. The 'feval()' function returns an 'octave_value_list' type result. The definition of an 'octave_value_list' uses an index which counts the values of the list. This index starts from 0, as it can be noticed in the 'for' loops. To write the output values of the 'feval()' function, they must be converted into the appropriate type. In this way, for the result of the 'gcd()' function is used the 'int_value()' function, for the result of the 'acos()' function is used the 'double_value()' function and for the 'inv()' function is used the 'matrix_value()' function.

In comparison with the Octave environment where the matrices' indexes, i.e. the line index and the column index of an element of a matrix, are starting from 1, in the programming mode they start from 0, otherwise the matrix is considered 'unknown'.

The second computer code is presented in appendix B and it tests the methods to use the built-in functions, [7]. The basic rule is that for a 'function_name' in GNU Octave there is a corresponding function in C++ API designated 'Ffunction_name'. For instance, in this computer code there are tested the 'norm()', 'inv()' and'det()' Octave functions whose corresponding functions are 'Fnorm()',

'Finv()' and 'Fdet()'. The 'builtin-defun-decls.h' header file contains the declarations of the built-in functions.

An important remark in this code is that *the C++ 'double' type values may be directly assigned to GNU Octave variables, therefore no conversion functions are necessary*.

The third computer code is presented in appendix C. It tests the conversion from a two dimensional array of double type values defined in C++ in a GNU Octave 'Matrix' type whose norm is computed. This example connects the C++ libraries already created which contain user-defined types to the C++ API of GNU Octave.

The fourth computer code presents an interesting method which uses as input a string which contains either a series of Octave commands, or a script name. Three tests were run and the computer codes, together with the results are presented in appendix D.The input string is written in a text file that is executed by Octave. This test program was inspired by [8], where a valuable code is presented, however in a compact form but which may be easily understood. The functionalities of the OctParser class presented in [8] were used to create the software demonstrator presented in appendix D.

We also noticed that the run of a GNU Octave script should start with 'clear -all' in order to completely delete the all the components of the symbol table: local and global user-defined variables and functions.

The software demonstrators were compiled, linked and run from the Eclipse environment and also from the according command line window, i.e. Command Prompt in Windows and Terminal in Ubuntu.

4. Discussion

Testing the C++ API capabilities in GNU Octave is a long run concern of the authors. The first successfully tests were run a few years ago when the current version was 4.0.0. At presents, working with the latest version, 5.1.0., is demanding because several changes were done in the successive GNU Octave versions.

One of our recent objective is to create multi-platform applications, therefore there was chosen a 'Long Term Support', aka LTS, Linux operating system, i.e. Ubuntu. One of the problems is that Ubuntu installs by default the 4.2.2 version of GNU Octave. Fortunately, all the programs installed by default in Ubuntu are 64 bit applications.

Other problem we had to face was in Windows, where some of the important 32bit dynamic link libraries used by us were not updated to 64 bit. Therefore, we had to use only 32 bit applications in order to preserve all the facilities offered by the already developed software.

To conclude, some of the important aspects regarding the complexity of the problem solved by us are: distinct operating systems, different versions of GNU Octave and 32 bit vs. 64 bit applications. The lack of documentation regarding the C++ API for GNU Octave was other reason we approached this problem in order to offer useful practical solutions for scientific programming analysts.

The initial tests of the computer codes in 5.1.0. returned the error "The procedure entry point glBegin@4 could not be located in the dynamic link library C:\Octave\Octave-5.1.0.0\mingw32\bin\libgl2ps.dll". The dependencies were verified, special care being dedicated to the 'GL', i.e. the graphical library. Several 'opengl*.dll'files were found in the computer and the simplest solution was to copy the 'opengl32.dll' (36770 KB) dynamic link library from 'C:\Octave\Octave-5.1.0.0\mingw32\bin' in the Debug folder of the Eclipse current project where the executable program is generated.

The details regarding the Eclipse settings, i.e. 'Include directories', 'Libraries' and 'Library Paths' are presented in the computer code as remarks. In this way, when a new version of a program is tested, the helpful information may be easily found.

As it can be noticed from the software solutions presented in the appendices, there are sequences defined for Linux and GNU Octave version 4.2.2 and sequences defined for Windows and GNU Octave version 5.1.0. In this way, in appendix A there may be noticed that the interpreter is started in two ways, according to the GNU Octave version. There are tested Octave functions which return

various types of data, integers, doubles and matrices. The software demonstrators present the methods used to handle the input and the output data for these functions in a C++ computer code.

Beside the method which uses the interpreter, there may be also used built-in functions, appendix B. This method was also used for three case studies, the result being firstly extracted from the output of the functions whose type is 'octave_value_list' and then converted to the appropriate C++ type.

The software demonstrator presented in appendix C starts from two user-defined types, 'real_bt' which may be float or double and 'index_bt' which is an integer type. These types are defined in '../_Headers/basic_types.h' and they are used in all the upper level applications, therefore there may be easily modified top level components from the bottommost level. The problem was to test if a data conversion between these '_bt', i.e. 'basic types', to GNU Octave types is possible and how to do it. In the code there may be remarked the conversion of the indexes of the matrix from integer user defined type to 'octave_idx_type', for instance 'i_OctVal=static_cast<octave_idx_type>(i_lin)'. There may be also noticed that the 'real_bt' variable may be directly assign to an element of a GNU Octave 'Matrix'. The next stage is to test if the conversion is successfully, by applying a built-in function to the 'Matrix' data. This software demonstrator is useful as an inspiring environment for the future numerical programming projects where various data types, including matrices, will be passed from C++ to Octave and finally back to C++.

In appendix D is presented other original demonstrator which uses as input data a string which includes GNU Octave commands. The string is written in a text file designated 'OCT_script.m' which is executed using the 'source_file(OCT_script_file)' command. There were considered three case studies, the output data being written in a text file. An imperfection of the solution resides in the repetitive write operations on the disk. However, this aspect was previously solved in [9] where a method to handle the large matrices was presented and in [10] where a nested based self-identification solutions' domain algorithm was applied in the modelling of the ship hull loads. The frequently use of some files lead to the idea that a RAM drive must be created for the storage of these files. In this way the execution time was significantly reduced by accessing files using the RAM speed. This idea may be also applied in this case. The according solution for Linux is in progress.

5. Conclusions

The paper presents a series of tests regarding the C++ API facilities which 'open' GNU Octave scientific programming environment to the C++ programming universe.

The important aspects consist of the knowhow presented in the paper, the details being very important for the facile development of the front end C++ applications.

In the following research stages there will be investigated the most flexible methods to develop header files which use the knowledge acquired during the development of the software demonstrators previously presented. This means to extend the C++ library already developed and to use GNU Octave as a numerical solver integrated in C++ application. Another objective is to use the GNU Octave plotting facilities in order to develop graphical interpreters.

The paper is included in our general trend to create software libraries, i.e. header files in C++, in various programming languages in order to use the API facilities of the CAD/CAE, FEM commercial applications and of the scientific programming environments, [11]. In this way, the software instruments of a hybrid model may be easily developed.

Appendix A

In this appendix is presented the 'OCTAVE_Test_0002.cpp' computer code which tests the 'feval()' function.

```
/*
 *  Name:   OCTAVE_Test_0002.cpp
 *  Inspired by: https://octave.org/doc/v5.1.0/Standalone-Programs.html
 *  Tested on:  March 02, 2019
 */
```

```
#ifdef __linux__
// Linux header files
#include"/usr/include/c++/7/iostream"// std::cout, std::end
#include"/usr/include/octave-4.2.2/octave/oct.h"// Octave header
#include"/usr/include/octave-4.2.2/octave/octave.h"// Octave header
#include"/usr/include/octave-4.2.2/octave/parse.h"// Octave header
#include"/usr/include/octave-4.2.2/octave/interpreter.h"// Octave header
#elif _WIN32
// Windows header files
#include<iostream>
#include"octave/oct.h"
#include"octave/octave.h"
#include"octave/parse.h"
#include"octave/interpreter.h"
#else
// Other operating system
#endif

/*
>>>>> SETTINGS <<<<<
 In Windows:
   The path to the DLL-s from Octave is: C:\Octave\Octave-5.1.0.0\mingw32\bin
   This path must be added in the system Path variable.

   In Eclipse libraries: octave octinterp
   In Eclipse the 'Library Paths' are:
      C:\MinGW\lib
      C:\Octave\Octave-5.1.0.0\mingw32\lib\octave\5.1.0
      C:\Octave\Octave-5.1.0.0\mingw32\bin
   In Eclipse 'Include directories'
      C:\Octave\Octave-5.1.0.0\mingw32\include\octave-5.1.0

   Path of the EXE: C:\E\Workspace-Eclipse\OCTAVE_Test_0002\Debug

//-------------------------
 In Linux:

  IMPORTANT FOR ALL C++ OCTAVE PROJECTS:
   octave.h from usr/include/octave-4.2.2/octave/ has <str_vec.h> in line 33,
signaled as error.
   str_vec.h is in the same folder as octave.h
<str_vec.h> means it is a system header, which is not!
   I modified the access rights in order to allow 'write': sudo chmod +2 octave.h
   I changed line 33 of octave.h from <str_vec.h> to "str_vec.h" and the error
isn't reported anymore!

   In Eclipse libraries: octave, octinterp
   In Eclipse the 'Library Paths' are:
      /usr/include/octave-4.2.2/octave
      /usr/share/octave/4.2.2/etc/tests
      /usr/lib/x86_64-linux-gnu
   In Eclipse 'Include directories'
      /usr/include/octave-4.2.2/octave
      /usr/share/octave/4.2.2/etc/tests
      /usr/lib/x86_64-linux-gnu

   Path of the executable: /home/emil/eclipse-workspace/OCTAVE_Test_0002/Debug
   Executable name: OCTAVE_Test_0002
      command: chmod +777 OCTAVE_Test_0002
      command: ./OCTAVE_Test_0002

   ----------------------------
```

```
   Terminal and Command Prompt compile command with 'mkoctfile',
which is also in /usr/bin:
      mkoctfile --link-stand-alone OCTAVE_Test_0002.cpp -o OCTAVE_Test_0002

*/
intmain ( int argc, char* argv[] ) {
// Define variable for versions earlier than 4.4
#ifdef __linux__
// At present the default version installed in Ubuntu is 4.2.0
        #define Octave_version_4
#elif _WIN32
// At present the default version in Windows is 5.1.0.
#else
// Other operating system
#endif
//
int echo=1;     /* used to trace the execution */
int i_casestudy=3;/* this is 1, 2 or 3           */
//
if (echo > 0) {
      std::cout<<"Step 0010   i_casestudy="<<i_casestudy<<std::endl;
   }
switch (i_casestudy) {
case 1 : { std::cout<<"   Test the gcd() function!\n"; break; }
case 2 : { std::cout<<"   Test the acos() function!\n"; break; }
case 3 : { std::cout<<"   Test the inv() function!\n"; break; }
   }
if (echo > 0) { std::cout<<"Step 0020"<< std::endl; }
try {
if (echo > 0) { std::cout<<"Step 0030"<< std::endl; }
int status;
#ifdef Octave_version_4
// Create interpreter up to version 4.4
         string_vector OCT_argv (2);
         OCT_argv(0) = "embedded";
         OCT_argv(1) = "-q";
// This is for Octave 4.2.x
         status = !octave_main (2, OCT_argv.c_str_vec (), true);
#else
// Create interpreter from version 4.4
         octave::interpreter intrprtr;
         status = intrprtr.execute ();
#endif
if (echo > 0) { std::cout<<"Step 0040"<< std::endl; }
if (status != 0) { // Tell the interpreter that we're ready to execute commands:
         std::cerr <<"creating embedded Octave interpreter failed!"<< std::endl;
return(status);
      }
if (echo > 0) { std::cout<<"Step 0050"<< std::endl; }
      octave_value_list in_data_list, out_data_list;
switch (i_casestudy) {
case 1 : {
            octave_idx_type n = 2;
if (echo > 0) { std::cout<<"Step 0060"<< std::endl; }
for (octave_idx_type i = 0; i < n; i++) {
               in_data_list(i) = octave_value (5 * (i + 2));
               std::cout<<"   in_data_list("<<i<<")="
<<in_data_list(i).int_value()<<std::endl;
            }
if (echo > 0) { std::cout<<"Step 0070"<< std::endl; }
#ifdef __linux__
/* Linux code - Octave 4.2.2 */
// feval (OCT_function, in_data_list, nargout);
```

```
                out_data_list = feval ("gcd", in_data_list, 1);
#elif _WIN32
/* Windows 10 code - Octave 5.1.0 */
                out_data_list = octave::feval ("gcd", in_data_list, 1);
#else
// Other operating system
#endif
if (echo > 0) { std::cout<<"Step 0080"<< std::endl; }
if (out_data_list.length () > 0) {
                std::cout <<"GCD of ["
<< in_data_list(0).int_value ()
<<", "
<< in_data_list(1).int_value ()
<<"] is "<< out_data_list(0).int_value ()
<< std::endl;
             } else {
                std::cout <<"invalid\n";
             }
if (echo > 0) { std::cout<<"Step 0090"<< std::endl; }
break;
          }
case 2 : {
             in_data_list(0)=-1.0;
if (echo > 0) { std::cout<<"Step 0100"<< std::endl; }
#ifdef __linux__
/* Linux code - Octave 4.2.2 */
// feval (OCT_function, in_data_list, nargout);
                out_data_list = feval( "acos", in_data_list, 1);
#elif _WIN32
/* Windows 10 code - Octave 5.1.0 */
                out_data_list = octave::feval( "acos", in_data_list, 1);
#else
// Other operating system
#endif
if (out_data_list.length() > 0) {
                std::cout <<"acos of "
<< in_data_list(0).double_value()
<<" is "
<< out_data_list(0).double_value()
<< std::endl;
             } else {
                std::cout <<"invalid\n";
             }
if (echo > 0) { std::cout<<"Step 0110"<< std::endl; }
break;
          }
case 3 : {
             octave_idx_type n = 2;
             Matrix a_matrix = Matrix (n, n);
             a_matrix(0,0)=1.0;  a_matrix(0,1)=2.0;
             a_matrix(1,0)=-1.0; a_matrix(1,1)=3.0;
             in_data_list(0)=a_matrix;
if (echo > 0) { std::cout<<"Step 0120"<< std::endl; }
#ifdef __linux__
/* Linux code - Octave 4.2.2 */
                // feval (OCT_function, in_data_list, nargout);
                out_data_list = feval( "inv", in_data_list, 1);
#elif _WIN32
/* Windows 10 code - Octave 5.1.0 */
                out_data_list = octave::feval( "inv", in_data_list, 1);
#else
// Unknown operating system
#endif
```

```
if (out_data_list.length() > 0) {
               std::cout <<"inv of \n"
<< in_data_list(0).matrix_value()
<<" is \n"
<< out_data_list(0).matrix_value()
<< std::endl;
            } else {
               std::cout <<"invalid\n";
            }
if (echo > 0) { std::cout<<"Step 0130"<< std::endl; }
break;
         }
      } /* End of switch */
if (echo > 0) { std::cout<<"Step 0140"<< std::endl; }
   } // End of try
catch (const octave::exit_exception& ex) {
      std::cerr <<"Octave interpreter exited with status = "
<< ex.exit_status () << std::endl;
   }
catch (const octave::execution_exception&) {
      std::cerr <<"error encountered in Octave evaluator!"<< std::endl;
   }
if (echo > 0) { std::cout<<"Step 0150"<< std::endl; }
// The next #ifdef ... #endif commented sequence is an alternate to
// the  try ... catch  active sequence which follows.
//   #ifdef Octave_version_4
//      clean_up_and_exit(0);
//   #endif
try {
// Workaround to avoid the error thrown by clean_up_and_exit() in
// Octave version 4.2.2
      clean_up_and_exit (0);
   }
catch (const octave::exit_exception&) {
// clean_up_and_exit should already have done this . . .
if (echo > 0) { std::cout<<"Step 0160"<< std::endl; }
//exit(0);
   }
if (echo > 0) { std::cout<<"Step 0170"<< std::endl; }
return(0);
}
```

Appendix B

In this appendix is presented the 'OCTAVE_Test_0003.cpp' computer code which tests the methods used to call built-in functions from C++ API.

```
/*
 *  Name:    OCTAVE_Test_0003.cpp
 *  Inspired by: https://octave.org/doc/v5.1.0/Standalone-Programs.html
 *  Tested on:   March 04, 2019
 */
#ifdef __linux__
// Linux header files
#include"/usr/include/c++/7/iostream"// std::cout, std::end
#include"/usr/include/octave-4.2.2/octave/oct.h"
#include"/usr/include/octave-4.2.2/octave/octave.h"
#include"/usr/include/octave-4.2.2/octave/parse.h"
#include"/usr/include/octave-4.2.2/octave/interpreter.h"
#include"/usr/include/octave-4.2.2/octave/builtin-defun-decls.h"
#elif _WIN32
#include<iostream>// std::cout, std::end
#include"octave\oct.h"
```

```
#include"octave\octave.h"
#include"octave\parse.h"
#include"octave\interpreter.h"
#include"octave\builtin-defun-decls.h"
#else
// Other operating system
#endif

/*
>>>>> SETTINGS <<<<<
 In Windows:
   The path to the DLL-s from Octave is: C:\Octave\Octave-5.1.0.0\mingw32\bin
   This path must be added in the system Path variable.

   In Eclipse libraries: octave octinterp
   In Eclipse the 'Library Paths' are:
      C:\MinGW\lib
      C:\Octave\Octave-5.1.0.0\mingw32\lib\octave\5.1.0
      C:\Octave\Octave-5.1.0.0\mingw32\bin
   In Eclipse 'Include directories'
      C:\Octave\Octave-5.1.0.0\mingw32\include\octave-5.1.0

   Path of the EXE: C:\E\Workspace-Eclipse\OCTAVE_Test_0003\Debug

//------------------------
 In Linux:

  IMPORTANT FOR ALL C++ OCTAVE PROJECTS:
   octave.h from usr/include/octave-4.2.2/octave/ has <str_vec.h> in line 33,
signaled as error.
   str_vec.h is in the same folder as octave.h
<str_vec.h> means it is a system header, which is not!
   I modified the access rights in order to allow 'write': sudo chmod +2 octave.h
   I changed line 33 of octave.h from <str_vec.h> to "str_vec.h" and the error
isn't reported anymore!

   In Eclipse libraries: octave, octinterp
   In Eclipse the 'Library Paths' are:
      /usr/bin
      /usr/include/octave-4.2.2/octave/
   In Eclipse 'Include directories'
      /usr/include/octave-4.2.2/octave/

   Path of the executable: /home/emil/eclipse-workspace/OCTAVE_Test_0003/Debug
   Executable name: OCTAVE_Test_0003
      command: chmod +777 OCTAVE_Test_0003
      command: ./OCTAVE_Test_0003

   Terminal compile command with mkoctfile, which is also in /usr/bin:
      mkoctfile --link-stand-alone OCTAVE_Test_0003.cpp -o OCTAVE_Test_0003
*/

intmain(int argc, char* argv[]) {
int echo=1;          /* used to trace the execution */
int i_casestudy=3;   /* this is 1, 2 or 3           */
//
if (echo > 0) {
      std::cout<<"Step 0010   i_casestudy="<<i_casestudy<<std::endl;
   }
switch (i_casestudy) {
case 1 : { std::cout<<"   Test the norm() function!\n"; break; }
case 2 : { std::cout<<"   Test the inv() function!\n"; break; }
case 3 : { std::cout<<"   Test the det() function!\n"; break; }
```

```
   }
   octave_idx_type n = 2;
if (echo > 0) { std::cout<<"Step 0020"<< std::endl; }
   Matrix a_matrix       = Matrix (n, n);
   Matrix matrix_inverse = Matrix (n, n);
if (echo > 0) { std::cout<<"Step 0030"<< std::endl; }
double a_val;
for (octave_idx_type i = 0; i < n; i++) {
for (octave_idx_type j = 0; j < n; j++) {
if (echo > 0) { std::cout<<"Step 0040"<< std::endl; }
         a_val = (i + 1) * 10.0 + (j + 1);
if (echo > 0) { std::cout<<"Step 0050 a_val="<<a_val<< std::endl; }
         a_matrix(i,j) = a_val; // So Matrix may directly use the C++ double type
if (echo > 0) { std::cout<<"Step 0060
a_matrix("<<i<<","<<j<<")="<<a_matrix(i,j)<< std::endl; }
      }
   }
if (echo > 0) {
      std::cout <<"Step 0070 - Matrix a_matrix is "<< a_matrix << std::endl;
   }
if (echo > 0) { std::cout<<"Step 0080"<< std::endl; }

   OCTINTERP_API::octave_value_list in, out;

if (echo > 0) { std::cout<<"Step 0090"<< std::endl; }

   in(0) = a_matrix;
//in(0).matrix_type(a_matrix) = a_matrix;

if (echo > 0) { std::cout<<"Step 0100"<< std::endl; }

switch (i_casestudy) {
case 1 : {
//out = OCTINTERP_API::Fnorm (in, 1);
         out=Fnorm(in, 1);
if (echo > 0) { std::cout<<"Step 0110"<< std::endl; }
double norm_of_the_matrix = out(0).double_value ();
if (echo > 0) { std::cout<<"Step 0120"<< std::endl; }
         std::cout <<"This is the norm of the matrix:"<< std::endl
<< norm_of_the_matrix                 << std::endl;
if (echo > 0) { std::cout<<"Step 0130"<< std::endl; }
break;
      }
case 2 : {
         out=Finv(in, 1);
if (echo > 0) { std::cout<<"Step 0140"<< std::endl; }
         matrix_inverse = out(0).matrix_value ();
if (echo > 0) { std::cout<<"Step 0150"<< std::endl; }
         std::cout <<"This is the inverse of the initial matrix:"<< std::endl
<< matrix_inverse<< std::endl;
if (echo > 0) { std::cout<<"Step 0160"<< std::endl; }
break;
      }
case 3 : {
         out=Fdet(in, 1);
if (echo > 0) { std::cout<<"Step 0170"<< std::endl; }
double matrix_determinant = out(0).double_value ();
if (echo > 0) { std::cout<<"Step 0180"<< std::endl; }
         std::cout <<"This is the determinant of the initial matrix:"<< std::endl
<< matrix_determinant<< std::endl;
if (echo > 0) { std::cout<<"Step 0190"<< std::endl; }
break;
      }
```

```
   } /* End of switch */
return(0);
}
```

Appendix C

In this appendix is presented the 'OCTAVE_Test_0004.cpp' computer code which converts a double array of double type values defined in C++ in a Matrix Octave type for which the norm() function is applied.

```
/*
 *  Name:         OCTAVE_Test_0004.cpp
 *  Inspired by: https://octave.org/doc/v5.1.0/Standalone-Programs.html
 *  Tested on:   March 06, 2019
 */
#ifdef __linux__
// Linux header files
#include"/usr/include/c++/7/iostream"// std::cout, std::end
#include"/usr/include/octave-4.2.2/octave/oct.h"
#include"/usr/include/octave-4.2.2/octave/octave.h"
#include"/usr/include/octave-4.2.2/octave/parse.h"
#include"/usr/include/octave-4.2.2/octave/interpreter.h"
#include"/usr/include/octave-4.2.2/octave/builtin-defun-decls.h"
#include"../_Headers/basic_types.h"
#include"../_Headers/cancel.h"
#elif _WIN32
#include<iostream>// std::cout, std::end
#include"octave/oct.h"
#include"octave/builtin-defun-decls.h"
#include"../_Headers/basic_types.h"
#include"../_Headers/cancel.h"
#else
// Other operating system
#endif

/*
>>>>> SETTINGS <<<<<
 In Windows:
   The path to the DLL-s from Octave is: C:\Octave\Octave-5.1.0.0\mingw32\bin
   This path must be added in the system Path variable.

   In Eclipse libraries: octave octinterp
   In Eclipse the 'Library Paths' are:
      C:\MinGW\lib
      C:\Octave\Octave-5.1.0.0\mingw32\lib\octave\5.1.0
      C:\Octave\Octave-5.1.0.0\mingw32\bin
   In Eclipse 'Include directories'
      C:\Octave\Octave-5.1.0.0\mingw32\include\octave-5.1.0

   Path of the EXE: C:\E\Workspace-Eclipse\OCTAVE_Test_0004\Debug
//------------------------
 In Linux:

  IMPORTANT FOR ALL C++ OCTAVE PROJECTS:
   octave.h from usr/include/octave-4.2.2/octave/ has <str_vec.h> in line 33,
signaled as error.
   str_vec.h is in the same folder as octave.h
<str_vec.h> means it is a system header, which is not!
   I modified the access rights in order to allow 'write': sudo chmod +2 octave.h
   I changed line 33 of octave.h from <str_vec.h> to "str_vec.h" and the error
isn't reported anymore!

   In Eclipse libraries: octave, octinterp
```

```
   In Eclipse the 'Library Paths' are:
      /usr/bin /usr/share/octave/4.2.2/etc/tests/
   In Eclipse 'Include directories'
      /usr/bin /usr/include/octave-4.2.2/octave/

   Path of the executable: /home/emil/eclipse-workspace/OCTAVE_Test_0004/Debug
   Executable name: OCTAVE_Test_0004
      command: chmod +777 OCTAVE_Test_0004
      command: ./OCTAVE_Test_0004

   Terminal compile command with mkoctfile, which is also in /usr/bin:
      mkoctfile --link-stand-alone OCTAVE_Test_0004.cpp -o OCTAVE_Test_0004

*/
usingnamespace std;

staticconstint matrix_dim=5;

typedefreal_btreal_bt_matrix[matrix_dim][matrix_dim];
real_bt_matrix a_matrix;
index_bt a_matrix_dim;
Matrix OCT_matrix;

voidgenerate_sample_matrix () {
// Set the global variables which may be accessed from any sequence
   a_matrix_dim=3;
   a_matrix[0][0]=1.0; a_matrix[0][1]=3.0; a_matrix[0][2]=5.0;
   a_matrix[1][0]=2.0; a_matrix[1][1]=7.0; a_matrix[1][2]=4.0;
   a_matrix[2][0]=9.0; a_matrix[2][1]=8.0; a_matrix[2][2]=0.0;
}
//
voidconvert_a_matrix_to_OCT_matrix (flag_bt echo) {
double dbl_number;
   OCT_matrix = Matrix(a_matrix_dim, a_matrix_dim);
index_bt i_lin, j_col;
   octave_idx_type i_OctVal, j_OctVal;
for (i_lin=0; i_lin<a_matrix_dim; i_lin++) {
for (j_col=0; j_col<a_matrix_dim; j_col++) {
         dbl_number=a_matrix[i_lin][j_col];
         i_OctVal=static_cast<octave_idx_type>(i_lin);
         j_OctVal=static_cast<octave_idx_type>(j_col);
         OCT_matrix(i_OctVal, j_OctVal)=dbl_number;
      }
   }
if (echo > 0) {
      std::cout<<"convert1 -> This is a
matrix:"<<std::endl<<OCT_matrix<<std::endl;
   }
}
//
voidOCT_matrix_operation_norm(flag_bt echo) {
   octave_value_list in;
   in(0) = OCT_matrix;
   octave_value_list out = Fnorm (in, 1);
double norm_of_the_matrix = out(0).double_value ();
if (echo > 0) {
      std::cout <<"This is the norm of the matrix:"<< std::endl
<< norm_of_the_matrix                << std::endl;
   }
}
//
voidrun_test_norm() {
flag_bt echo=1;
```

```
   generate_sample_matrix();
   convert_a_matrix_to_OCT_matrix(echo);
   OCT_matrix_operation_norm(echo);
}
//
intmain(int argc, char* argv[]) {
  run_test_norm();
return(0);
}
```

Appendix D

In this appendix is presented the 'OCTAVE_Test_0006.cpp' computer codeand the results for 3 tests.

```
/*
 *  Name:        OCTAVE_Test_0006.cpp
 *  Inspired by: http://octave.1599824.n4.nabble.com/Running-a-script-file-
inside-c-code-tc4679185.html; author: Sumeet Kumar Sinha
 *  Tested on:   March 08, 2019
 */
#ifdef __linux__
// Linux header files
#include"/usr/include/c++/7/iostream"// std::cout, std::end
#include"/usr/include/c++/7/fstream"// for files
#include"/usr/include/c++/7/string"//
#include"/usr/include/c++/7/stdlib.h"//
#include"/usr/include/octave-4.2.2/octave/oct.h"// Octave header
#include"/usr/include/octave-4.2.2/octave/octave.h"// Octave header
#include"/usr/include/octave-4.2.2/octave/parse.h"
#include"/usr/include/octave-4.2.2/octave/interpreter.h"

#elif _WIN32
// Windows header files
#include<iostream>// std::cout, std::end
#include<fstream>
#include<string>
#include"octave\oct.h"// If the path is set in Eclipse
#include"octave\octave.h"// If the path is set in Eclipse
#include"octave\parse.h"// If the path is set in Eclipse
#include"octave\interpreter.h"// If the path is set in Eclipse
#else
// Other operating system
#endif
/*
>>>>> SETTINGS <<<<<
 In Windows:
   The path to the DLL-s from Octave is: C:\Octave\Octave-5.1.0.0\mingw32\bin
   This path must be added in the system Path variable.

   In Eclipse libraries: comdlg32 mingw32 octinterp octave
   In Eclipse the 'Library Paths' are:
      C:\MinGW\lib
      C:\Octave\Octave-5.1.0.0\mingw32\lib\octave\5.1.0
      C:\Octave\Octave-5.1.0.0\mingw32\bin
   In Eclipse 'Include directories'
      C:\Octave\Octave-5.1.0.0\mingw32\include\octave-5.1.0
      C:\Octave\Octave-5.1.0.0\mingw32\lib\octave\5.1.0

   Path of the EXE: C:\E\Workspace-Eclipse\OCTAVE_Test_0001\Debug
//------------------------
 In Linux:

  IMPORTANT FOR ALL C++ OCTAVE PROJECTS:
```

```
   octave.h from usr/include/octave-4.2.2/octave/ has <str_vec.h> in line 33,
signaled as error.
   str_vec.h is in the same folder as octave.h
<str_vec.h> means it is a system header, which is not!
   I modified the access rights in order to allow 'write': sudo chmod +2 octave.h
   I changed line 33 of octave.h from <str_vec.h> to "str_vec.h" and the error
isn't reported anymore!

   In Eclipse libraries: octave, octinterp
   In Eclipse the 'Library Paths' are:
      /usr/include/octave-4.2.2/octave
      /usr/share/octave/4.2.2/etc/tests
      /usr/lib/x86_64-linux-gnu
   In Eclipse 'Include directories'
      /usr/include/octave-4.2.2/octave
      /usr/share/octave/4.2.2/etc/tests
      /usr/lib/x86_64-linux-gnu

   Path of the executable: /home/emil/eclipse-workspace/OCTAVE_Test_0006/Debug
   Executable name: OCTAVE_Test_0006
      command: chmod +777 OCTAVE_Test_0006
      command: ./OCTAVE_Test_0006

   Terminal compile command with mkoctfile, which is also in /usr/bin:
      mkoctfile --link-stand-alone OCTAVE_Test_0006.cpp -o OCTAVE_Test_0006
*/
intmain(int argc, char* argv[]) {
int echo=1;          /* used to trace the execution */
int i_casestudy=3;   /* this is 1, 2 or 3           */
   std::string OCT_script_file="OCT_script.m";
//
if (echo > 0) { std::cout<<"Step 0010"<< std::endl; }
   std::string OCT_commands;
switch (i_casestudy) {
case 1 : {
         std::cout<<"   Test Octave common commands!\n";
         OCT_commands = "x=1; y=5; x+y; save a.txt ans;";
break; }
case 2 : {
         std::cout<<"   Test the inv() function!\n";
         OCT_commands = "a=[1 3; 2 -1]; b=inv(a); save a.txt b;";
break; }
case 3 : {
         std::cout<<"   Test the execution of the test_777.m script\n";
         OCT_commands = "test_777.m";
break; }
   }
if (echo > 0) { std::cout<<"Step 0020 - OCT_commands: '"<<OCT_commands<<"'"<<
std::endl; }
// Define variable for versions earlier than 4.4
#ifdef __linux__
// At present the default version in Linux is 4.2.0
#define Octave_version_4
if (echo > 0) { std::cout<<"Step 0030"<< std::endl; }
#elif _WIN32
// At present the default version in Windows is 5.1.0.
#else
// Other operating system
#endif
//
if (echo > 0) { std::cout<<"Step 0040"<< std::endl; }
int status;
#ifdef Octave_version_4
```

```
if (echo > 0) { std::cout<<"Step 0050"<< std::endl; }
// Define parameters for Octave interpreter for versions <= 4.2.0
      string_vector OCT_argv (3);
      OCT_argv(0) = "embedded";
      OCT_argv(1) = "-q";
      OCT_argv(2) = "--interactive";
// Create interpreter for Octave versions <= 4.2.0
      status = !octave_main (3, OCT_argv.c_str_vec (), true);
#else
if (echo > 0) { std::cout<<"Step 0060"<< std::endl; }
// Create interpreter for versions > 4.2.0
      octave::interpreter intrprtr;
      status = intrprtr.execute ();
#endif
if (echo > 0) { std::cout<<"Step 0070"<< std::endl; }
if (status != 0) {
        // Tell the interpreter that we're ready to execute commands:
      std::cerr <<"creating embedded Octave interpreter failed!"<< std::endl;
return(status);
   }
if (echo > 0) { std::cout<<"Step 0080"<< std::endl; }
//
   std::fstream fstream_outputfile;
constchar* txt_outfilename=OCT_script_file.c_str();
   fstream_outputfile.open(txt_outfilename, std::ios::out);
   fstream_outputfile<<OCT_commands;
   fstream_outputfile.close();
if (echo > 0) { std::cout<<"Step 0090"<< std::endl; }
#ifdef __linux__
// Linux code
int i_success; // 'system()' command return code
      std::string Linux_cmd;
      std::string folder_name;
//
      folder_name="/home/emil/eclipse-workspace/OCTAVE_Test_0006";
// folder_name="./";
//
      Linux_cmd="chmod a+rwx "; // a means 'for all'; + means 'activate';
read/write/execute
      Linux_cmd=Linux_cmd+folder_name;
if (echo > 0) { std::cout<<"Step 0100 - Linux_cmd: '"<<Linux_cmd<<"'"<<
std::endl; }
      i_success=system(Linux_cmd.c_str());
if (echo > 0) {
         std::cout<<"Step 0110 - Current folder changing permissions return code: ";
         std::cout<<i_success<< std::endl;
      }
// If necessary
      std::string file_permissions="666"; // For owner, group, other: 1=execute; 2=write; 4=read;
      Linux_cmd="chmod "+file_permissions+" ./"+OCT_script_file;
if (echo > 0) { std::cout<<"Step 0120 - Linux_cmd: '"<<Linux_cmd<<"'"<<
std::endl; }
      i_success=system(Linux_cmd.c_str());
if (echo > 0) {
         std::cout<<"Step 0130 - Changing permissions return code: "<<i_success<<
std::endl;
      }
      source_file(OCT_script_file,"",true);
//       int parse_status;
//       eval_string(OCT_commands,true,parse_status);
if (echo > 0) { std::cout<<"Step 0140"<< std::endl; }
#elif _WIN32
// Windows code
```

```
      octave::source_file(OCT_script_file);
if (echo > 0) { std::cout<<"Step 0150"<< std::endl; }
#else
// Other operating system
#endif
if (echo > 0) { std::cout<<"Step 0160"<< std::endl; }

#ifdef Octave_version_4
      clean_up_and_exit(0);
if (echo > 0) { std::cout<<"Step 0170"<< std::endl; }
#endif

/*
   try {
      // Workaround to avoid the error thrown by clean_up_and_exit() in
      // Octave version 4.2.2
      clean_up_and_exit(0);
   }
   catch (const octave::exit_exception&) {
      // clean_up_and_exit should already have done this . . .
      if (echo > 0) { std::cout<<"Step 0180"<< std::endl; }
      //exit(0);
   }
*/
if (echo > 0) { std::cout<<"Step 0190"<< std::endl; }
return(0);
}
```

a) Results for i_casestudy=1,
" x=1; y=5; x+y; save a.txt ans;"

Contents of 'OCT_script.m'

```
x=1; y=5; x+y; save a.txt ans;
```

Contents of 'a.txt'

```
# Created by Octave 5.1.0, . . .
# name: ans
# type: scalar
6
```

b) Results for i_casestudy=2,
" a=[1 3; 2 -1]; b=inv(a); save a.txt b;"

Contents of 'script'

```
a=[1 3; 2 -1]; b=inv(a); save a.txt b;
```

Contents of 'a.txt'

```
# Created by Octave 5.1.0, . . . .
# name: b
# type: matrix
# rows: 2
# columns: 2
 0.14285714285714285 0.4285714285714286
 0.2857142857142857 -0.14285714285714285
```

c) Results for i_casestudy=3, "test_777.m"

Contents of 'test_777.m'

```
a = [1 2; 4 -1]
b = inv(a)
x = rand (1, 10);
y = rand (1, 10);
save a.txt a
save b.txt b
save x.txt x y
```

Contents of 'a.txt'

```
# Created by Octave 5.1.0, . . . .
# name: a
# type: matrix
# rows: 2
# columns: 2
 1 2
 4 -1
```

Contents of 'b.txt'

```
# Created by Octave 5.1.0, . . . .
# name: b
# type: matrix
# rows: 2
# columns: 2
 0.1111111111111111 0.22222222222222221
 0.44444444444444442 -0.1111111111111111
```

Contents of 'x.txt'

```
# Created by Octave 5.1.0, . . . .
# name: x
# type: matrix
# rows: 1
# columns: 10
 0.53468986103028615 0.59526721302034358 0.48955503368307135 0.17389774257027163
0.32631027876714741 0.058678169145625111 0.94226966565386661 0.26581763031688427
0.20705764328990156 0.46018303831243451


# name: y
# type: matrix
# rows: 1
# columns: 10
 0.83858658334179692 0.51350899425659668 0.53152687095072249 0.14967622800209363
0.20589866978037369 0.018271409957827656 0.30391144167325362 0.67259826982753623
0.96700786918407333 0.14950365895740353
```

6. References

[1] Oanta E and Panait C 2014 *Advanced Material Research* Trans Tech Publications **837**(99) pp 141-146
[2] Oanta E, Raicu A and Panait C 2017 *IOP Conference Series: Materials Science and Engineering* **227**
[3] Oanta E and Taraza D 2000 *SAE 2000 World Congress Proceedings*
[4] Oanta E, Panait C, Barhalescu M, Sabau A, Dumitrache C and Dascalescu A-E 2015 *SPIE 9258 Advanced Topics in Optoelectronics, Microelectronics, and Nanotechnologies VII* **92582A**
[5] Oanta E 2018 *Hybrid modeling in mechanical engineering* Habilitation thesis September 21 2018 Doctoral School of Mechanical Engineering Constanta Maritime University
[6] Oanta E, Pescaru A and Micu A 2018 *SPIE 10977 Advanced Topics in Optoelectronics, Microelectronics, and Nanotechnologies IX* **109771R**
[7] GNU Octave Standalone Applications, https://octave.org/doc/v5.1.0/Standalone-Programs.html accessed on 17 March 2019
[8] Sinha S K 2016 *Running a script file inside c++ code* http://octave.1599824.n4.nabble.com/Running-a-script-file-inside-c-code-tc4679185.html accessed on 24 February 2019
[9] Oanta E and Nicolescu B 1999 *A Versatile PC-Based Method for the Processing of the Large*

Matrices Proceedings of the 19th Computers and Information in Engineering ASME Conference 2 457-464 paper DETC99/CIE-9059
[10] Oanta E, Tamas R and Paun M 2018 *SPIE 10977 Advanced Topics in Optoelectronics, Microelectronics and Nanotechnologies IX* **1097704**
[11] Oanta E, Panait C and Raicu A 2016 *SPIE 10010 Advanced Topics in Optoelectronics, Microelectronics, and Nanotechnologies IX* **100102O**

Acknowledgement
This work was supported by a grant of the Romanian Ministry of Research and Innovation, CCCDI-UEFISCDI, project number PN-III-P1-1.2-PCCDI-2017-04-04/31PCCDI/2018, acronym HORESEC, within PNCDI III.
The authors express their gratitude to the GNU Octave maintainers and to the GNU Octave community for their generosity in sharing their ideas. This paper is intended to become a 'helpful hand' for them.