Algorithm XXX: MANBIS—A C++ Mathematical Software Package for Locating and Computing Efficiently Many Roots of a Function: Usage and Practical Issues

DIMITRA-NEFELI A. ZOTTOU, DIMITRIS J. KAVVADIAS, FROSSO S. MAKRI, and MICHAEL N. VRAHATIS, University of Patras

This article describes the user interface of MANBIS, a C++ mathematical software package for tackling the problem of computing the roots of a function when the number of roots is very large (of the order of hundreds or thousands). This problem has attracted increasing attention in recent years because of the broad variety of applications in various fields of science and technology. MANBIS applies the bisection method in order to obtain an approximate root according to a predetermined accuracy. The theory behind the MANBIS algorithm is given in the accompanying paper [Zottou *et al.* 2017a], along with its features and advantages. In this paper we give the user interface to MANBIS and present several details of its implementation, as well as examples of its usage.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General–*Numerical algorithms*; G.4 [**Mathematical Software**]: *Algorithm design and analysis*

General Terms: Algorithms

Additional Key Words and Phrases: Rootfinding, zerofinding, many zeroes, expected behavior, bisection based methods, imprecise function values, counting and computing roots, very large problems

ACM Reference Format:
Dimitra-Nefeli A. Zottou, Dimitris J. Kavvadias, Frosso S. Makri, and Michael N. Vrahatis, 2017b. Algorithm XXX: MANBIS—A C++ mathematical software package for locating and computing efficiently many roots of a Function: Usage and practical issues. *ACM Trans. Math. Softw.* 1, 1, Article 2 (January 2017), 14 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

A brief description of the numerical methods employed in MANBIS is given in the accompanying paper [Zottou *et al.* 2017a] while the complete description along with its theoretical development and the necessary proofs are given in the paper [Kavvadias *et al.* 2005]. Here we give the user interface to MANBIS and give implementation details, as well as examples of its usage.

In this article we describe a set of eight files written in C++, collectively called MANBIS. This mathematical software package computes a user-given percentage $q \in (0, 100)$ of the total number of roots of a continuous function $f : \mathbb{R} \to \mathbb{R}$ within the user-defined interval $[a, b] \subset \mathbb{R}$.

In Section 2 the usage description of the package is provided, the input parameters and the termination criteria are described and two simple representative examples for the usage of MANBIS are presented in details, in order to illustrate in practice the processes. The description of the software is provided in Section 3 while in Section 4 sample applications for testing purposes are illustrated and the corresponding

Authors' address: Department of Mathematics, University of Patras, GR-26110 Patras, Greece; email: {nefeli.zottou; djk; makri; vrahatis}@math.upatras.gr.

DOI: http://dx.doi.org/10.1145/0000000.0000000

discussion is given. Finally, a synopsis and some concluding remarks are presented in Section 5.

2. USAGE DESCRIPTION

The package MANBIS consists of a set of eight C++ source files, three of which must be edited by the user to embed the function to be analysed. The description information related to these files appears in Section 3.

The package requires a C++ compiler in order to be executed. All the files of MANBIS, including the three files provided by the user, must be stored in the same folder. Whenever the user desires to add a new function, MANBIS must be compiled again, in order to incorporate the formula into the program. If the user desires to run a different function, which has already been embedded, then only the file `main.cpp` must be compiled again. Any number of functions can be defined here. Recompilation is not necessary for a different run of the same function.

The required steps for a typical run are as follows:

Step 1: Add the new formula of the function in the file `Functions.cpp`. The syntax of this file must be as follows:

```
# include "Functions.h"
long double function_name(long double variable_name) {
   return formula of function;}
```

The user must replace the strings `function_name` and `variable_name` with an appropriate function name, and the string `formula_of_function` with the formula of the function coded in C ++. More complicated functions can be defined using the appropriate code lines written in C++. If additional variables are needed they can be defined at this stage using any valid C++ name. In order to create the formula $f(x)$, the user may freely use any available mathematical function in C++.

Step 2: Add the prototype of the new function in the file `Functions.h`. The syntax of this file must be as follows:

```
# include <iostream>
# include <cmath>
long double function_name(long double variable_name);
```

The user can add more formulas at the end of the file `Functions.cpp`. The prototypes of the additional functions should also be declared in the file `Functions.h`.

Step 3: The user must set the name of the function in the file `main.cpp`, in order to specify which formula will be analysed by MANBIS. The syntax of the file `main.cpp` must be as follows:

```
# include <iostream>
# include "Parameters.h"
# include "Manbis.h"
# include "Functions.h"
using namespace std;
int main(int argc, char* argv[]){
   Parameters parameters(argc, argv);
   Manbis man(parameters, function_name);
   man.FindRoots();
   return 0;
}
```

Step 4: Issue the following commands at the command line:

```
  g++ main.cpp Functions.cpp Parameters.cpp Manbis.cpp -o
nameofexecutable.out
  ./nameofexecutable.out parameters list
```

2.1. Input Parameters and Termination Criteria

The parameters that MANBIS accepts are twelve with only three of them being mandatory. These are the left and right endpoints of the considered interval and the percentage of the roots that the user desires to obtain. The specified parameters must be given at the command line next to the name of the executable. The available options are described below:

(1) `(--digits)` or `(-a)`: Number of digits for the output computed results. This is an optional parameter with a non-negative integer value. The default value is 8.

(2) `(--time)` or `(-t)`: Flag for returning or not the execution time. This is an optional parameter. Valid values are 0 and 1. The value 1 gives the execution time while the default value is 0.

(3) `(--accuracy)` or `(-e)`: Precision of the computed roots. This is an optional parameter with an integer value greater than 0 and less than 16 for 32-bit OS or less than 32 for 64-bit OS. The default value is 4.

(4) `(--xleft)` or `(-l)`: Left endpoint of the initial interval. This is a **mandatory** real parameter.

(5) `(--xright)` or `(-r)`: Right endpoint of the initial interval. This is a **mandatory** real parameter whose value must be greater than the value of `(--xleft)`.

(6) `(--uncutfiles)` or `(-u)`: Flag that controls the construction of the files "SubsByLength.txt" and "SubsByXLeft.txt". This is an optional parameter. If its value is 1 then the file "SubsByLength.txt" will contain the unexamined subintervals sorted by length while the file "SubsByXLeft.txt" will contain the unexamined subintervals sorted by their left endpoint. The default value is 0 (no construction).

(7) `(--percentage)` or `(-q)`: Percentage of the roots that the program will attempt to compute. This is a **mandatory** parameter with a real value greater than 0 and less than or equal to 100 (see Subsection 2.2 of the article [Zottou *et al.* 2017a]).

(8) `(--probability)` or `(-p)`: Probability of the confidence interval. This is an optional parameter. Valid values are 0.95, 0.99, 0.90 or 0.80. MANBIS estimates the total number of roots by calculating two estimations of their number, a lower and an upper one. As the program proceeds these values are updated and their difference decreases. Thus, the parameter `(-p)` is the probability that the actual number of roots lies between these two values. The reported value when the program ends is their sum divided by two (see Subsection 2.2 of the article [Zottou *et al.* 2017a]).

(9) `(--noroots)` or `(-n)`: Using the value 1 the roots are printed in the file Result.txt while using the value 0 only their number is printed.

(10) `(--statdiff)` or `(-s)`: Difference between two consecutive statistical estimations. This is an optional parameter, which controls the minimum allowed difference between two consecutive estimations. If this minimum difference is not achieved the algorithm will not terminate even if it estimates that the required percentage of roots has been computed. Allowed values are in the interval [0.1,1.5] and default is 0.5. Smaller values will generally result to better estimations but also to longer runs.

(11) `(--currentest)` or `(-c)`: Flag for enabling the display of the progress of the program in real time. This is an optional parameter that may assist the user to decide whether to prematurely terminate the program if it takes too long. Valid values are 0 and 1 with the default being 0 (not enabled). When enabled, the progress of the program is displayed in real time. In particular, the following information is displayed on the screen from left to right:
   i. The current estimation of the total number of roots. This becomes more reliable as the program proceeds and stabilizes when sufficient information becomes available.
   ii. The absolute number of the required roots. This is the previous value multiplied by the required fraction of roots.
   iii. The discovered number of roots.
   iv. The fraction of the discovered roots.
   v. The real time (in seconds) to discover 100 roots. This increases as the program proceeds as the calculation of new roots becomes more and more costly.

(12) `(--help)` or `(-h)`: It gives a brief description of the program usage.

The previous step is equivalent to using the `make` command, with the exception that in the later case the executable name is by default `manbis.out`. Thus, the usage of the `makefile` is as follows:

```
make
./manbis.out parameters
```

Furthermore, the file "makefile" contains two additional commands which remove some files that the user might want to delete after the execution of MANBIS. The first command is the following:

```
make clean1
```

This command removes the three files: "SubsByLength.txt", "SubsByXLeft.txt" and "Results.txt". The second command is the following:

```
make clean
```

This command removes all the executable files.

2.2. Usage Examples

We next give two simple representative examples for the usage of MANBIS.

Example 1. Calculate the roots of the function $f(x) = \sin(x^3)\cos(x^2) - 0.02$.
   Task 1: The considered interval is $(0, 10)$, the required accuracy is determined using 5 decimal digits and the required percentage is 80%.
   Task 2: The same function as above (no additional compilation is required), the considered interval is $(0, 5)$, the required accuracy is determined using 6 decimal digits and the required percentage is 90%. On the output only the number of the roots in the "Result.txt" is provided. Also, the unexamined subintervals are required to be sorted by their left endpoint as well as by there length in the files "SubsByXLeft.txt" and "SubsByLength.txt" respectively.

The corresponding steps for *Task 1* are as follows:

Step 1: The user has to add the new formula of the function in the file Functions.cpp. For instance in the following lines the above function is called "example1":

```
# include "Functions.h"
long double example1(long double x) {
   return sin( pow(x, 3) ) * cos( pow(x, 2) ) - 0.02 ;
}
```

Step 2: The user has to add the prototype of the function in the file `Functions.h`. The syntax of this file must be as follows:

```
# include <iostream>
# include <cmath>
long double example1(long double x);
```

Step 3: The user should set the name of the function in the file `main.cpp`. The syntax of the file must be as follows:

```
# include <iostream>
# include "Parameters.h"
# include "Manbis.h"
# include "Functions.h"
using namespace std;
int main(int argc, char* argv[]){
   Parameters parameters(argc, argv);
   Manbis man(parameters, example1);
   man.FindRoots();
   return 0;
}//End of main
```

Step 4: Issue the following command at the command line:

```
g++ main.cpp Functions.cpp Parameters.cpp Manbis.cpp -o manbis.out
./manbis.out -l 0 -r 10 -e 5 -q 80
```

After the execution of this command the following lines will be displayed at the screen:

```
Accuracy: 1e-05
Initial interval: (0,10)
Percentage: 80%
Number of digits of output roots:8
```

After the termination of MANBIS the file "Results.txt" will be created, which contains the computed roots. The format of the output is as follows (we give here only the first eight lines):

```
Requested percentage is 80%
Requested accuracy is 1e-05
Number of the roots that the program has calculated is 333
Estimated percentage of the calculated roots is 83%
Estimated total number of roots in the interval (0,10) is 404.
The roots are:
0.27169466
1.2447667
     ⋮
```

In order to perform *Task 2* the user just needs to give the following command (no other steps are required):

```
./manbis.out -l 0 -r 5 -e 6 -q 90 -n 0 -u 1
```

Notice that, in this case, the file "Result.txt" does not contain the roots but only the following text:

```
Requested percentage is 90%
Requested accuracy is 1e-06
Number of the roots that the program has calculated is 46
Estimated percentage of the calculated roots is 91%
Estimated total number of roots in the interval (0,5) is 52.
```

Also, in this case, after the termination of the program two more files have been created. Thus, the first five lines of the file "SubsByXLeft.txt" are the following:

```
The uncut odd subintervals are 1155
Sorted by xleft
1)= (0 , 0.009765625) length=0.009765625
2)= (0.009765625 , 0.01953125) length=0.009765625
3)= (0.01953125 , 0.029296875) length=0.009765625
      ⋮
```

while the first five lines of the file "SubsByLength.txt" are the following:

```
The uncut even subintervals are 1155
Sorted by length
1)= (4.6582031 , 4.6679688) length=0.009765625
2)= (0.25390625 , 0.26367188) length=0.009765625
3)= (1.2304688 , 1.2402344) length=0.009765625
      ⋮
```

A useful option especially when dealing with large instances, is the `-c` switch. This enables the display of the progress of the program in real time. The screen looks as follows:

```
Current    | Requested | Discovered | Discovered | Avg time of computing
estimation | #roots    | #roots     | fraction   | 100 roots

--341-------341----------337----------99%----------0.268843sec-----------
```

Displayed values are explained in the previous paragraph. The main purpose of this option is to help the user to obtain an idea of the size of the problem, the current progress of the program and possibly decide to terminate the program by pressing `Ctrl+C` if the number of discovered roots is sufficient or if the program is taking too long to terminate. No loss of calculations occurs in this way and the calculated roots can be found in the specified file.

Example 2. Compute the roots of the function $f(x) = \cos(x)$ using the Taylor's series expansion. The considered interval is $(-5, 5)$, the required accuracy is determined using 8 decimal digits, the required percentage is 85% as well as all the results are printed out using 16 decimal digits.

The purpose of this example is to demonstrate a definition of a more complicated function as well as to demonstrate the usage of the `make` command and how the user can add the formula of the second example without deleting the example1 in case of future examination.

The corresponding steps are as follows:

Step 1: The user can add the following lines after the end of the function example1:

```
long double example2(long double x) {
long double f=1;
double  fct=2;
double  sqrx=x*x;
int  index=2;
double  factor=0;
for(int i=1;i<100;i++){
    factor=sqrx/fct;
    if(i%2){
        f-=factor;
    }else{
        f+=factor;
    }
    index++;
    fct*=index;
    index++;
    fct*=index;
    sqrx*=x*x;
}
return f;
}
```

Step 2: The user should add at the end of the file Functions.h the following line code:

```
long double example2(long double x);
```

Step 3: The user must slightly change the file `main.cpp` from the previous example, as follows:

```
# include <iostream>
# include "Parameters.h"
# include "Manbis.h"
# include "Functions.h"
using namespace std;
int main(int argc, char* argv[]){
   Parameters parameters(argc, argv);
   Manbis man(parameters, example2);
   man.FindRoots();
   return 0;
}//End of main
```

Step 4: Issue the following commands at the command line:

```
make
./manbis.out -l -5 -r 5 -e 8 -q 85 -a 16
```

Remark 2.1. Notice that the above command `./manbis.out -l -5 -r 5 -e 8 -q 85 -a 16` is equivalent to the command `./manbis.out --xleftl -5 --xright 5 --accuracy 8 --percentage 85 --digits 16`.

In this case, the file "Results.txt" contains the following lines where all the numbers have been printed out using 16 decimal digits:

```
Requested percentage is 85%
Requested accuracy is 1e-08
Number of the roots that the program has calculated is 4
Estimated percentage of the calculated roots is 89%
Estimated total number of roots in the interval (-5,5) is 5.
The roots are:
-4.712388981133699
-1.570796326268464
1.570796326268464
4.712388981133699
```

3. DESCRIPTION OF THE SOFTWARE

The package MANBIS contains about $1,700$ lines of code, 50 percent of which are comments. MANBIS is coded in standard C++ and has been tested on Microsoft Windows, GNU/Linux and MAC OS X. The total storage required by MANBIS is $80.5\,k$B while the average executable file size is $109.0\,k$B depending on the function which the user has set in the text file.

MANBIS consists of the following files:

(1) **Functions.cpp**: Contains the definitions of the user's formulas.
(2) **Functions.h**: Contains the prototypes of the user's functions. Any function defined in Functions.cpp should also be prototyped here.
(3) **Parameters.cpp**: Reads and checks the parameters supplied by the user and performs some important initializations.
(4) **Parameters.h**: Contains the prototypes of the functions in the file Parameters.cpp.
(5) **Manbis.cpp**: Contains the code of some functions of the program.
(6) **Manbis.h**: Contains the prototypes of the functions in the file Manbis.cpp.
(7) **main.cpp**: Contains the main function which calls MANBIS with the function to be analysed.

Next we briefly describe each one of the above files.

Functions.cpp: Contains the user's functions. Any number of functions can be coded here. The return value of these functions should be a long double. The name of the functions can be any valid name of C++. Finally, any of these functions has one long double parameter.

(1) `long double name_users_function(long double variable_name)`: It receives as input a real number and returns the corresponding value of the function.

Functions.h: This file contains all the prototypes of the user's functions, which have been created in the file Functions.cpp. If the user needs to include a non-standard library or header file in order to define his function, he should manually edit this file and add a line at the beginning with the appropriate include directive.

Parameters.h: Contains the prototypes of the functions in the file Parameters.cpp.

Parameters.cpp: Reads and checks the parameters supplied by the user and performs some initializations.

Manbis.h: Contains the prototypes of the functions in the file Manbis.cpp.

Manbis.cpp: This file contains the implementations of the following functions:

(1) `void PrimordialStep()`: This function performs some initializations.

(2) `void CleanRoots(std::map<long double, long long unsigned int>& myroots, long double e, long double medial)`: The purpose of this function is to assure that only one copy of a certain root will be stored. Thus, this function will retain the copy of the root using the highest precision.

(3) `void StatisticStopingCriterion()`: This function implements the stopping criterion of the algorithm. If the discovered roots as a percentage of the estimated number of roots exceeds the user's supplied percentage then the algorithm will terminate.

(4) `void CutOdds()`: Subdivides an "odd subinterval", that is the subinterval which its endpoints obtain opposite signs of the function values. It also calls the functions "StatisticStoppingCriterion" and "CleanRoots".

(5) `void CutEven()`: Subdivides an "even subinterval", that is the subinterval which its endpoints obtain the same sign of the function values. It also calls the functions "StatisticStoppingCriterion" and "CleanRoots".

(6) `void PrintResults()`: Prints out the computed roots in the file named "Results.txt".

(7) `int GetExistOdd()`: This function returns the value 1 if at least one subinterval with opposite signs at its endpoints has been created during the execution of the program.

(8) `long long unsigned int GetNumberOfOdd()`: Returns at every iteration the number of subintervals with opposite signs at their endpoints, which have not been examined.

(9) `long long unsigned int GetNumberOfEven()`: Returns at every iteration the number of the subintervals with the same sign at their endpoints, which have not been examined.

(10) `void PrintChoices()`: Displays on the screen some messages regarding the user's choices.

(11) `int Bolzano(long double x1,long double x2, long double (*function_)(long double ) )` : The parameters of this function are two real values x1 and x2 and a pointer to the formula of the function and returns -1 if the function values f(x1) and f(x2) obtain opposite signs, otherwise it returns 1.

(12) `void UserInterruption(int sig)`: This function terminates the program when `Ctrl+C` is pressed. Before termination the calculated roots up to that point are saved in the specified file.

(13) `void FindRoots()`: This is the main program which evokes all the above functions. It requires the user determined parameters and it gives the text file "Result.txt" as well as the text files "SubsByLength.txt" and "SubsByXLeft.txt" only if this is requested by the user. Also, it performs the following procedures:

- P1: Call PrintChoices to output the choices of the user.
- P2: Call PrimordialStep(), which bisects the interval $[a, b]$ into two equal subintervals and initialize MANBIS.
- P3: Call GetExistOdd() in order to initialize the flag `exist_odd`, which confirms the existence of at least one "odd subinterval".

P4: Call GetNumberOfOdd() in order to initialize the `number_of_odd` subintervals.

P5: Call GetNumberOfEven() in order to initialize the `number_of_even` subintervals.

P6: If an odd subinterval exists it calls CutOdds() to subdivide the current interval and if the statistics criterion is fulfilled it performs the procedure P11, otherwise it performs the next procedure.

P7: Update the variables `exist_odd`, `number_of_odd`, `number_of_even` by calling the functions GetExistOdd(), GetNumberOfOdd(), GetNumberOfEven(), respectively.

P8: If an odd subinterval exists it performs the procedure P6, otherwise it performs the next procedure.

P9: If an even subinterval exists it calls EvenOdds() to subdivide the current interval and if the statistics criterion is fulfilled it performs the procedure P11, otherwise it performs the next procedure.

P10: Update the variables `exist_odd`, `number_of_odd`, `number_of_even` by calling the functions GetExistOdd(), GetNumberOfOdd(), GetNumberOfEven(), respectively and perform the procedure P6.

P11: Call PrintResults() to write the outputs to the text files and terminate MANBIS.

4. SAMPLE APPLICATIONS

The main assumption on which the total number of roots is estimated is that the roots of the function are uniformly distributed in the considered interval. However, this information is not a priori available. On the other hand, this is a natural choice when no additional information about the distribution of the roots of the function is given. Our experience is that the proposed approach can also be efficiently applied in cases where the distribution of the roots is not uniform and, in general, where the distribution is not a priori known.

Firstly, in order to test the accuracy of the statistics that the code calculates we have provided a function whose roots are known to be uniformly distributed in the given interval. The following example describes such an *artificial* function. The term artificial refers to the fact that the roots were beforehand determined by randomly selecting a predefined number of points in the considered interval. Thus, using the build-in C++ function rand() in the interval $(0, 1)$ we have produced the points r_i, $i = 1, 2, \ldots, n$ and we have considered these points to be the roots of the following function:

$$f(x) = \prod_{i=1}^{n}(x - r_i), \quad \text{where } r_i \text{ are random numbers within the interval } (0, 1). \tag{1}$$

We ran two tests using function (1), with $n = 100$ and $n = 1000$. Tables I and II provide a synopsis of the results. We have named the corresponding functions “fun100” and “fun1000” respectively and they have been included in the files Functions.cpp and Functions.h for testing purpose. We analyse the above functions in the interval (0,1) with accuracy 8 decimal digits. A fraction of the results with different required percentage is given in table I and table II for both “fun100” and “fun1000” respectively. The labels of the columns are as follows: “P” indicates the required percentage of the roots which has been set by the user, “S” is the allowed difference of estimations in two consecutive samplings, also set by the user, “NCR” is the number of the roots that have been calculated by MANBIS, “EP” is the estimated percentage of the roots that has been calculated by MANBIS, “NER” is the total number of roots that has been

Table I: Runs of MANBIS for test function (1),where $n = 100$.

P	S	NCR	EP	NER	SEC
60%	0.5	60	73%	84	0.015
65%	0.5	60	73%	84	0.015
80%	0.5	76	84%	91	0.015
90%	0.5	88	91%	97	0.015
99%	0.5	96	100%	97	0.110

estimated by MANBIS and "SEC" is the corresponding execution time in seconds required by MANBIS in order to compute the roots. For this issue we have conducted 100 independent experiments and we have calculated the mean execution time in seconds.

For the "fun100" the line below sets the required percentage to 90

```
./manbis.out -l 0 -r 1 -e 8 -q 90
```

With the above parameters MANBIS has computed 91 roots. The total computation burden for each root was low and it was consistent with $\lceil \log_2(b-a)\,\varepsilon^{-1} \rceil$ [Zottou *et al.* 2017a, Relation (2)]. Notice that the requested percentage was 90% which means that 90 roots should had been computed out of the 100 existing, while 88 have been computed by MANBIS. This is so, because the estimation, after the computation of 88 roots (as shown in column "EP"), was that the interval contained 97 roots and consequently the discovered 88 constitute a percentage slightly higher than 90%, which is the reason why the program terminated at this point.

Tables I and II show that the number of estimated roots approaches the real number as the number of computed roots increases. Also, we observe that although the number of the roots is relatively small, MANBIS behaved predictably and reliably. In the example "fun100", when the percentage was set to a value less than 65%, MANBIS will calculate again EP=73%. This is so, because, in this case the information required for the statistical analysis is not sufficient.

As it is pointed out in [Zottou *et al.* 2017a] our approach draws its strength from the fact that the roots are expected to be many. Thus, in order to verify also experimentally this statement we increased the number of roots of the function (1) to 1000. To this end, by using again the build-in C++ function rand() in the interval $(0, 1)$ we have produced the points r_i, $i = 1, 2, \ldots, n$, where $n = 1000$ and we have considered these points to be the roots of the function (1). We have named the corresponding function "fun1000" and we have included it into the package for testing purposes.

In Table II, results for the function (1) obtained by performing five different runs of MANBIS using various required percentages and the same accuracy of 8 decimal digits are exhibited. In this case, we observe that the results are more satisfactory.

Our final example deals with a case where the roots are not uniformly distributed using the following function:

$$f(x) = \cos(x^2)\sin(x^2) - 0.2\,. \tag{2}$$

This is a transcendental function with approximately 32.200 not uniformly distributed roots within the considered interval $(-200.5, 100)$. We have named the corresponding function "fun" and we have also included it into the package for testing purposes. In Table III, results for function (2) obtained by performing different runs of MANBIS with various percentages and the same accuracy of 8 decimal digits are exhibited. In this case the user has to be stricter with the requested percentage. This is because, for

Table II: Runs of MANBIS for test function (1),where $n = 1000$.

P	S	NCR	EP	NER	SEC
60%	0.5	626	65%	973	0.234
65%	0.5	782	80%	987	0.298
80%	0.5	874	89%	984	0.390
90%	0.5	926	95%	983	0.509
99%	0.5	990	100%	998	2.282

low percentages, MANBIS may terminate at an earlier stage due to the fact that the gathered data are not sufficient for a good estimation of the actual number of roots. From the above results we observe that the number of estimated roots approaches its actual number as the number of computed roots increases. Also, in this case, we observe that for low percentages the statistical estimation failed. In such a case the user is advised to repeat the procedure more than once with different required percentage and/or smaller values for the -s parameter. For example, the first row of Table III demonstrates that the default value `-s=0.5` with goal percentage 30% lead to a large underestimation of the actual number of roots. More runs were tried with higher percentages which give better estimations of the roots in the given interval. Especially for the percentage 30% two more runs are exhibited in lines 2 and 3 with a smaller value of `-s` parameter (0.3 or 0.4 respectively). These lines exhibit that the underestimation of the first line was corrected. The new estimation is closer to the real value as the value of `-s` parameter is smaller, closer to the lower limit 0..1. Note however, that in the final column the execution time is higher as the `-s` value is getting smaller.

All the results detailed in this section were obtained on an OS Microsoft Windows 8.1, System type 64-bit Operating System, x64-bases processor, Processor Intel(R)Core(TM)i5-4210U CPU @ 2.40 GHz, Installed memory (RAM) 4,00 GB. The compiler we used was the gcc version 4.9.3 applied on Cygwin 6.3, which is a Unix-like environment and command-line interface for Microsoft Windows.

5. SYNOPSIS AND CONCLUDING REMARKS

In this article, a C++ mathematical software package called MANBIS has been described. This package computes a user-given percentage of the total number of roots according to the user-given accuracy of the user-defined continuous real function of one variable within the user-given interval.

MANBIS applies a bisection method in order to obtain the approximate roots. Thus, the only computable information required is the algebraic signs of the given function which is the smallest amount of information (one bit of information) necessary for the purpose needed. Furthermore, the number of iterations required by the bisection method in order to compute an approximate root to a predetermined accuracy is known a priori.

It is important to point out that MANBIS can also be applied in cases where the roots are not uniformly distributed in the considered interval or when the distribution of the roots is unknown. In general, this information is not available a priori. However, a natural choice when no additional information about the distribution of the roots of the function is given is to consider the roots uniformly distributed in the defined interval.

Table III: Runs of MANBIS with different percentage for the test function "fun" (function (2)).

P	S	NCR	EP	NER	SEC
30%	0.5	1912	33%	5842	0.256
30%	0.4	7128	43%	16777	0.908
30%	0.3	31960	88%	36635	4.448
33%	0.5	3814	35%	11172	0.490
40%	0.5	7128	43%	16777	0.908
43%	0.5	28318	44%	65488	3.439
60%	0.5	31960	73%	43833	3.965
65%	0.5	31960	73%	43833	3.952
80%	0.5	31960	88%	36635	4.467
90%	0.5	31960	94%	34083	6.001
95%	0.5	31960	97%	32976	8.452
97%	0.5	31960	99%	32458	14.425
99%	0.5	31960	100%	32207	28.509

Our experience is that MANBIS is efficient for tackling specific root-finding problem even when the roots are not uniformly distributed. In this case, however, the statistical estimation may fail. In such a case the user is advised to repeat the procedure more than once with different required percentage and observe possible variations in the estimations. Additionally, the user is advised to control the allowed difference of estimations in two consecutive sampling. In general, larger percentages are to be trusted more, even in the case of the known uniform distribution. The estimation with the largest number of root should be kept.

In conclusion, the main issues of the proposed package can be summarised as follows: a) To the best of our knowledge, MANBIS is the first package for computing very "cheaply" a percentage of roots when the size of the problem is very large. b) MANBIS proceeds solely by requiring only the smallest amount of information (one bit of information) necessary for the purpose needed, and not any additional information. c) MANBIS is capable of estimating without any additional function computational cost the total number of roots within the user-given interval. This estimation is improved after each iteration, by taking into consideration the new computed roots, and d) MANBIS is capable of computing (beforehand) the total computational cost required for approximating a root with a given accuracy. Thus, since it also estimates the total number of roots, it is able to easily estimate at each stage the total computational burden required for computing all the roots.

ACKNOWLEDGMENTS

We wish to express our appreciation and thanks to Dr Tim Hopkins and the referees for their constructive and valuable comments.

REFERENCES

KAVVADIAS, D. J., MAKRI F. S., AND VRAHATIS, M. N. 2005. Efficiently computing many roots of a function. *SIAM J. Sci. Comput. 27*, 1, 93–107.

ZOTTOU, D.-N. A., KAVVADIAS, D. J., MAKRI, F. S., AND VRAHATIS, M. N. 2017a. Algorithm XXX: MANBIS—A C++ mathematical software package for locating and computing efficiently many roots of a function: Theoretical issues. *ibid.*

Received March 2016; revised September 2016; accepted October 2007