On-line interactive programs for solving and visualization of scientific problems

Damir Krstinić and Ivan Slapničar
Faculty of Electrical Engineering, Mechanical Engineering
and Naval Architecture
University of Split
R. Boškovića b.b., 21000 Split, Croatia
E-mail: damir.krstinic@fesb.hr, ivan.slapnicar@fesb.hr

Abstract— **We present interactive programs for solving and visualizing the solutions of scientific problems. Considered problems require small amount of input and output data, but are computationally demanding. This is ideal for using Internet since the communication is minimal and powerful servers, including parallel computers, are used for computation. Presented programs solve boundary value problems with finite elements method, perform Fourier analysis (FRed), and numerical computation and visualization (Octave Online). Programs are easy-to-use via graphical user interface, and have high educational value. The programs, developed within projects financed by the Croatian Ministry of Science and Technology, are accessible at `http://bolek.fesb.hr/czr/en/on-line.php3`.**

I. Introduction

TODAY'S scientific and engineering calculations requires large amount of memory and CPU time. While large companies and scientific institutions can provide computers needed by this calculations, for smaller institutions price is often a limiting factor. Even if one or few of stronger computers can be obtained, they are usually not accessible by everyone, like students or dislocated departments and employees. If application is CPU time demanding, but requires relatively small amount of input and output data, making it accessible through the Internet or intranet is a good solution for this problem.

We have developed several applications for solution and visualization of scientific and engineering problems. These applications are *Boundary Value Problem* solver which solves boundary value problems with finite elements method, *FRed* which performs Fourier analysis, and *Octave On-line* for general numerical computations and visualization. These programs are developed within several projects financed by Croatian Ministry of Science and Technology in cooperation with the Center for Scientific Computing of the University of Split. Programs have intuitive and easy to use graphical user interface, and are accessible at `http://bolek.fesb.hr/czr/en/on-line.php3`.

The rest of the paper is organized as follows: in Section II we describe the technical aspects which are used in realization of all three programs, and in Section III we describe these programs in detail and give some examples.

II. Technical requirements

SEVERAL tools are used for creation of these programs; all of them freely available on the Internet. On our servers we are using Linux operating system (version SuSE 6.3, see `http://www.suse.de`) and Apache HTTP server (version 1.3, see `http://www.apache.org`). Any other UNIX operating systems should be just fine. The connection between the web server (user input) and the executable code is performed via the hypertext preprocessor PHP [7]. PHP is a server side language, with syntax similar to the one of C, which can be conveniently used to transform data from and to browser and in automatic generation of HTML code.

Computational part of the programs is programmed in GNU Octave [2]. GNU Octave is a high-level language primarily intended for numerical computations. It uses language that is mostly compatible with Matlab. For better performance, for some calculations we have developed some add-on routines for GNU Octave in C/C++ language. For compilation of C/C++ source code we have used standard GNU gcc/g++ compilers as shipped with the Linux.

Output images are created with Gnuplot [1], and JavaScript [3] is used in some parts of the user interface.

At the client side, no special software is required. Any operating system and any newer web browser is fine. This is another advantage of our programs, since users can have just inexpensive terminal equipment and still be able to perform demanding computations.

III. Program description

WHEN a user opens any of our programs, input form is generated by PHP. After the user is over with input, the data processing is started by selecting `Graph` or `Compute` button. Another PHP program is now called. This program parses the input data and checks for errors. The input data is then formated and a program which performs actual computation is called. Output data is flushed directly to the browser, so usually there is no need for any temporary data storage on the server. This is important since the program are not limited to one user at the time. Of course, large number of users at the same time can overload the CPU or memory resources, which will manifest itself through slower performance of computation. As the programs run on

a public server, the number of simultaneous users is limited for system protection, but there is no priority system. In more demanding professional applications, this could be achieved through closed user group with password protection, and CPU time reservation.

A. Octave On-line

IN the on-line version of GNU Octave, the user can directly enter Octave commands in the command-input entry field, or upload already prepared Octave program files (m-files) from user's own hard disk. After uploading, the uploaded file appears in command-input entry field and can be edited again. When the editing is over, the user execute commands on the server side. Output of the program contains both text and images mixed in the output window. All of the numeric (textual) output can be saved to automatically generated local file. After each run, the user can change parts of the program and start the computation again. The user can also the commands to a local file.

For the ease of use, the user has on line access to the complete Octave manual in two forms, through the GNU Info System and in the HTML version. We have also provided a help interface through help on particular command can be obtained.

The program needs some temporary server side data, but this is completely irrelevant for the user since all of the output is generated and flushed to the browser automatically and the user needs not worry about the data flow.

Since the program is being executed on the server, and the user is provided with the fully featured programming language this application has necessary security checking in order to avoid the possible attacks.

B. FRed

THE acronym *FRed* stands for "Fourierov red" which is the Croatian translation of "Fourier Series". *FRed* computes the coefficients of the Fourier expansion of the periodic function $f(x)$ which is defined on the interval $[a, b]$ and then periodically extended to the real axis. The Fourier expansion of f is given by [6, §6.12]

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cos\frac{k\pi x}{L} + b_k \sin\frac{k\pi x}{L},$$

where $L = (b-a)/2$. The Fourier coefficients a_k and b_k are given by

$$\begin{aligned} a_0 &= \int_a^b f(x)dx, \\ a_k &= \int_a^b f(x)\cos\frac{k\pi x}{L}dx, \\ b_k &= \int_a^b f(x)\sin\frac{k\pi x}{L}dx. \end{aligned}$$

The above integrals are numerically computed with the trapezoidal rule [6, §7.2]: for $n+1$ equidistant integration nodes $x_0, x_1, \ldots, x_n$ where $x_0 = a$ and $x_n = b$, we have

$$\int_a^b f(x)dx = \frac{b-a}{n}\left(\frac{f(x_0)+f(x_n)}{2} + \sum_{i=1}^{n-1} f(x_i)\right).$$

Since the program uses numerical integration, the Fourier coefficients can be computed if the above integrals are not elementary solvable, as well.

The program is used as follows. User is provided with the input form for entering required data. By selecting `Graph` button, the input is checked for errors and properly formated, and the Octave m-file which performs the actual computation is called. Since the integrals are computed with the trapezoidal formula, the results are more accurate if the required number of series' terms m is sufficiently small with respect to the number of integration nodes n.

The primary output is graphical, consisting of two images. The first image displays the graph of the given function and its approximation with the first m terms of the Fourier series. The second image displays the distribution of the first m harmonics $h_k = \sqrt{a_k^2 + b_k^2}$. The user can also choose to display numerical values of the computed coefficients.

Since *FRed* is at the present stage mainly intended for students the number of integration nodes is limited to 999, and the number of coefficients is limited to 99. However, these can easily be extended if needed.

As an example, in Fig. 1 we show the computation of the first 61 Fourier coefficients ($m = 30$) for the function $f(x) = \sqrt(x)$ on the interval $[0, 1]$ by using 300 integration nodes.

In the future development we plan to use the Fast Fourier Transform [6, §6.13] to compute the necessary integrals, and to use parallel processors for computation. This should considerably speed up the execution, and enable the use of larger number of integration nodes. Another planned improvement it to computed the bounds for the errors in computed coefficients by Richardson's extrapolation [6, §7.1].

C. Finite elements method for the boundary value problem

THE program finds the solution $u(x)$ of a second order differential equation:

$$-[a(x)u(x)']' + b(x)u(x) = f(x), \quad 0 \le x \le L \qquad (1)$$

subject to one of the four combinations of Dirichlet or Von Neumann boundary conditions:

1. $u(0) = u_0, \quad u(L) = u_L$
2. $u'(0) = u'_0, \quad u(L) = u_L$
3. $u(0) = u_0, \quad u'(L) = u'_L$
4. $u'(0) = u'_0, \quad u'(L) = u'_L$

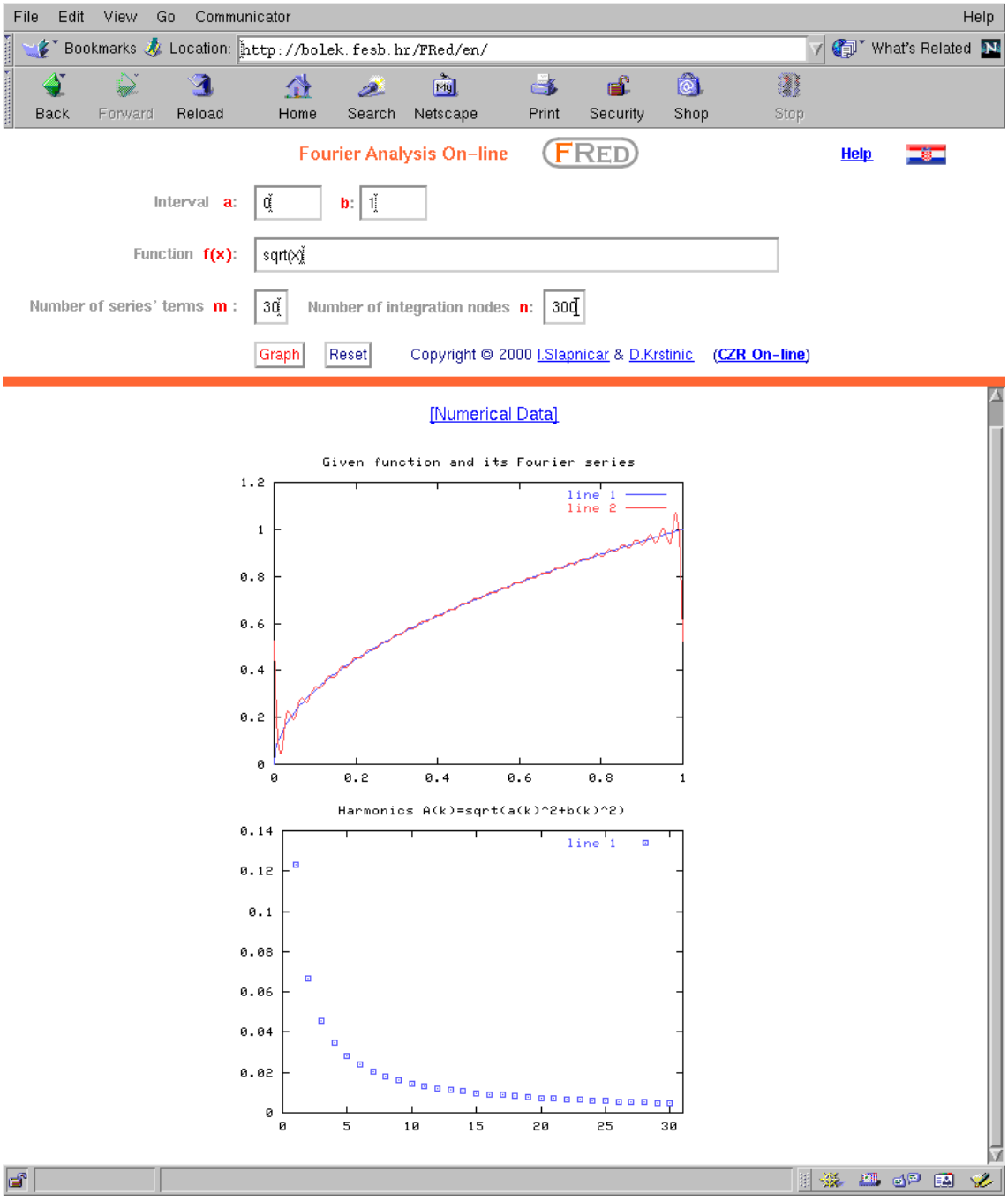

Fig. 1. Computation of Fourier coefficients

where

$f(x)$	is a real function
$a(x) > 0$	is a positive function
$b(x) \geq 0$	is a non negative function

The last combination of the boundary conditions has physical meaning only if $b(x) > 0$.

The equation (1) has many applications. For example, the solution u gives the state of the equilibrium (displacement) of a thin wire which satisfies given boundary conditions. In this case the variables in (1) have the following meaning:

L	is the length of the wire,
$u(0)$	is the position of the wire on the left side ($x = 0$),
$u(L)$	is the position of the wire on the right side ($x = L$),
$u'(0)$	is the slope of the tangent on the left side ($x = 0$),
$u'(L)$	is the slope of the tangent on the right side of ($x = L$),

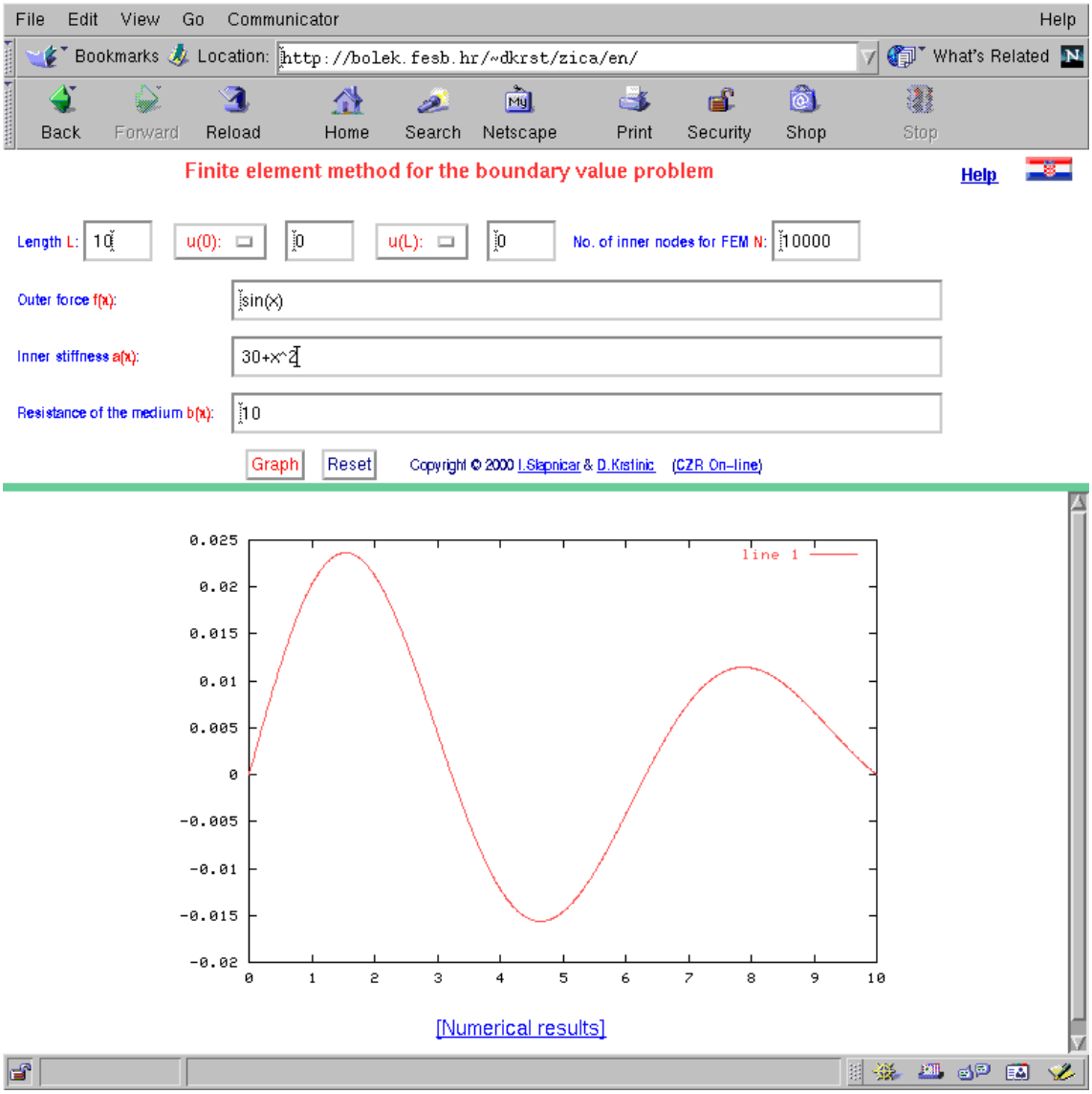

Fig. 2. Displacement of a thin wire

$f(x)$ is the distribution of the outer transversal force that acts on the the wire,

$a(x)$ is the distribution of the inner stiffness of the wire,

$b(x)$ is the distribution of the resistance of the elastic medium.

The numerical solution of the above differential equation is computed by using finite elements method, as described in [4], [6]. We have made some improvements in system coefficients generation in order to additionally speed up the method. The finite element method generates a tridiagonal symmetric positive definite system of linear equations. The main program which computes the coefficients and solves the resulting linear system is written in GNU Octave. Since GNU Octave does not provide a special solver for the symmetric tridiagonal systems, we incorporated the appropriate routine *dptsv* for solving tridiagonal positive definite systems from *LAPACK* [5] into Octave. By using *dptsv* great improvements where achieved in maximum system dimension and speed. Currently, the program solves problems up to 10^6 discretization nodes, and on our server, dual Intel Pentium III 600MHz with 256 MB RAM, such computations takes about 10 seconds.

As an example, in Fig. 2 we show the displacement of a thin wire fixed on both ends, $u(0) = u(L) = 0$, subjected to the sinusoidal transversal outer force, $f(x) = \sin x$, with the inner stiffness distribution given by $a(x) = 30 + x^2$. The resistance of the elastic medium is constant, $b(x) = 10$. The computation is performed for 10000 discretization nodes.

Notice that one advantage of using a high-level language such as Octave is the availability of large number of predefined functions, which simplifies the use of our program. In particular, by using functions such as *floor* or *ceil* the user can easily define Dirac-type functions

which are non-zero only at a certain point. For example, the expression

$$3 * floor(x/0.299) * floor(-(x-1)/0.699)$$

defines the function on interval $[0, 1]$ which is equal to 3 at the point $x = 0.3$ (actually, in a neighborhood of that point), and zero otherwise. Such forces typically appear in many real-world applications.

We are now developing parallel version of this program which will enable us to solve greater problems and increase the computation speed. We are also developing similar program for more interesting two dimensional problems.

IV. Conclusion

THE described and adopted technology enables the development of server-side applications for solving complex scientific problems. The described programs enable users to solve and model demanding engineering problems, for which they just need the Internet connection. The potential of the described approach is enormous and the work on developing these and further applications is in progress. In particular, we plan to enable parallel computation on the server-side, which will considerably increase performance.

Acknowledgments

We would like to thank Professor Krešimir Veselić, Fernuniversität Hagen, Germany, for encouraging us in writing Octave On-line and for helpful suggestions concerning the implementation of the finite element method for the boundary value problem. We also thank John W. Eaton from the Department of Chemical Engineering, University of Wisconsin–Madison, and Tom Weichmann, Buffalo State College, for their comments which helped us deal with some security issues in Octave On-line.

References

[1] Thomas Williams and Colin Kelley, *Gnuplot, An Interactive Plotting Program*, 1998, `http://www.gnuplot.org`.

[2] John W. Eaton, *GNU Octave*, 1997, `http://www.octave.org`.

[3] *JavaScript Guide*, Netscape Communications Corp., 2000, `http://developer.netscape.com`.

[4] Ibrahim Aganović and Krešimir Veselić, *Jednadžbe matematičke fizike*, Školska knjiga, Zagreb, 1985.

[5] E. Anderson et al., *LAPACK Users' Guide*, SIAM, Philadelphia, 1995.

[6] David Kincaid and Ward Cheney, *Numerical Analysis*, Brooks/Cole, Pacific Groove, 1996.

[7] Stig S. Bakken, *PHP Manual*, PHP Documentation Group, 2000, `http://www.php.net`.