A Tutor Program for a Numerical Methods Course for Engineers

Cuneyt Sert

Department of Mechanical Engineering
Middle East Technical University
06531 Ankara, Turkey

Abstract

A software is developed to serve as a learning and practice tool for the students of an undergraduate numerical methods course. It can also be used by the teachers of such a course in preparing class notes and exam questions. It consists of several modules, each covering a separate topic such as finding roots of nonlinear equations, solving systems of linear algebraic equations, optimization, curve fitting, differentiation, integration, and solving ordinary differential equations. The user can run typical numerical methods problems and get detailed numerical and visual output of the solution. The software has a graphical user interface for user interaction. It is an active software that allows to run problems that are designed by the user, instead of running pre-solved problems.

1. Introduction

Numerical methods are used to solve algebraic representations of physical problems. They are based on simple, well-defined algorithms, but contain rigorous algebraic operations, suitable for computer implementation. Based on this fact, one can idetify two typical properties of an undergraduate level numerical methods course: 1. Their underlying mathematics is usually straightforward. 2. They should be accompanied with some sort of a computer tool. The selection of the proper computer tool is important in the success of a numerical methods course. Typical choices are:

- Computer languages such as Fortran, Pascal, C, etc[1,2].
- Computer Algebra Systems such as Matlab, Maple, Mathcad, Mathematica, etc[3].
- Course specific softwares such as the ones that come with numerical methods textbooks.

The software described in this paper is of the third kind. Its capabilities will be described in the following sections.

2. EasyNumerics Software

The software described in this paper is called EasyNumerics[4]. It is currently being used in teaching a third year numerical methods course in the Mechanical Engineering Department of Middle East Technical University. It is written using the Tcl/Tk scripting language[5]. There are two reasons behind this choice. First one is the power of Tcl/Tk in creating Graphical User Interfaces (GUIs)[6]. Second one is the platform independence of Tcl/Tk. One can use the source code in almost every known computer platform with very little (or no) changes. EasyNumerics is available freely from its website[4]. Students and instructors can download and use it for educational and academic purposes.

Figure 1 shows the opening menu of EasyNumerics and summarizes its capabilities. As shown in Fig. 1, EasyNumerics has seven modules. First one is a function plotter used to plot single variable functions in two-dimensions. Second module is used for finding the roots of nonlinear equations. Third module is for solving sets of linear algebraic equations as well as for matrix operations like calculating the determinant or taking the inverse of a matrix. Another module is used for single variable unconstrained optimization problems. Next one is about curve fitting, which includes linear least squares regresssion and interpolation. Than comes a module for numerical differentiation and integration. Last module is used for solving first order ordinary differential equations.

2a. Function Plotting Module

Function Plotting module gives a general tool to generate graphs of single variable functions. It can be used to get a general insight for many numerical problems. For example in finding the roots of a nonlinear equation using iterative methods, a plot of the function is valuable for getting good initial estimates. Many of the other modules of EasyNumerics uses the function plotting module for visual represenation of the results.

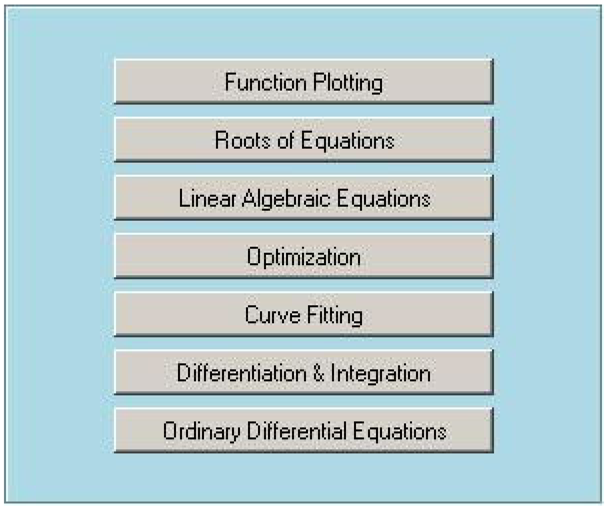
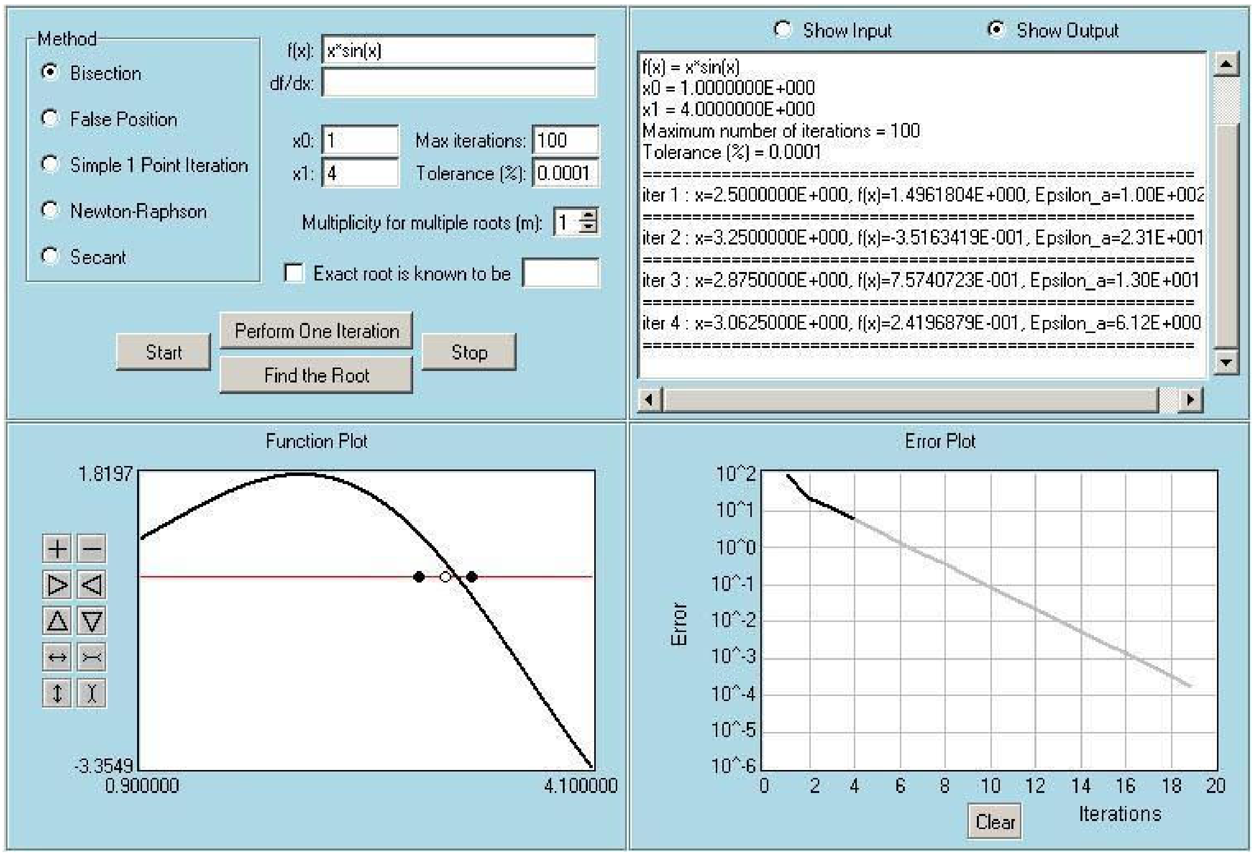

Function Plotting
Roots of Equations
Linear Algebraic Equations
Optimization
Curve Fitting
Differentiation & Integration
Ordinary Differential Equations

Figure 1. Opening menu of EasyNumerics

2b. Roots of Equations Module

Roots of Equations module is used to find the roots of nonlinear equations. A sample run of this module, used to find the first positive root of $f(x) = x\,sin(x)$, is shown in Fig. 2. The user starts by selecting the method to be used, either a bracketing method (bisection and false position) or an open method (simple one-point iteration, Newton-Raphson and secant). Than the function itself, starting estimates, maximum iteration number and the tolerance value need to be entered. If the exact root is specified, true errors can be calculated. Otherwise approximate errors are calculated. If the root is suspected to be a multiple root, than it can be mentioned to speed up the convergence. After inputting all necessary data, the user can perform one iteration at a time and watch the progress of the solution in the Function Plot window. The function will be plotted and the previous and present roots will be shown. In the Function plot window of Fig. 2, the results after the fourth iteration are shown. The two outer points surrounding the actual root are the results obtained from previous iterations. The bisection method that is used for this sample run calculates the new estimate for the root as the arithmetic average of these two points, which is shown as the white middle point. Results of the run can also be seen in the Output window. As seen from Fig. 2, estimated root, function value at the estimated root and the relative percent error are calculated after each iteration. The convergence history of this iterative solution can be followed from the Error Plot window, where the calculated errors are plotted. The gray line corresponds to a previous solution of the same problem. The plot for the current run is shown in black, which goes upto the fourth iteration.



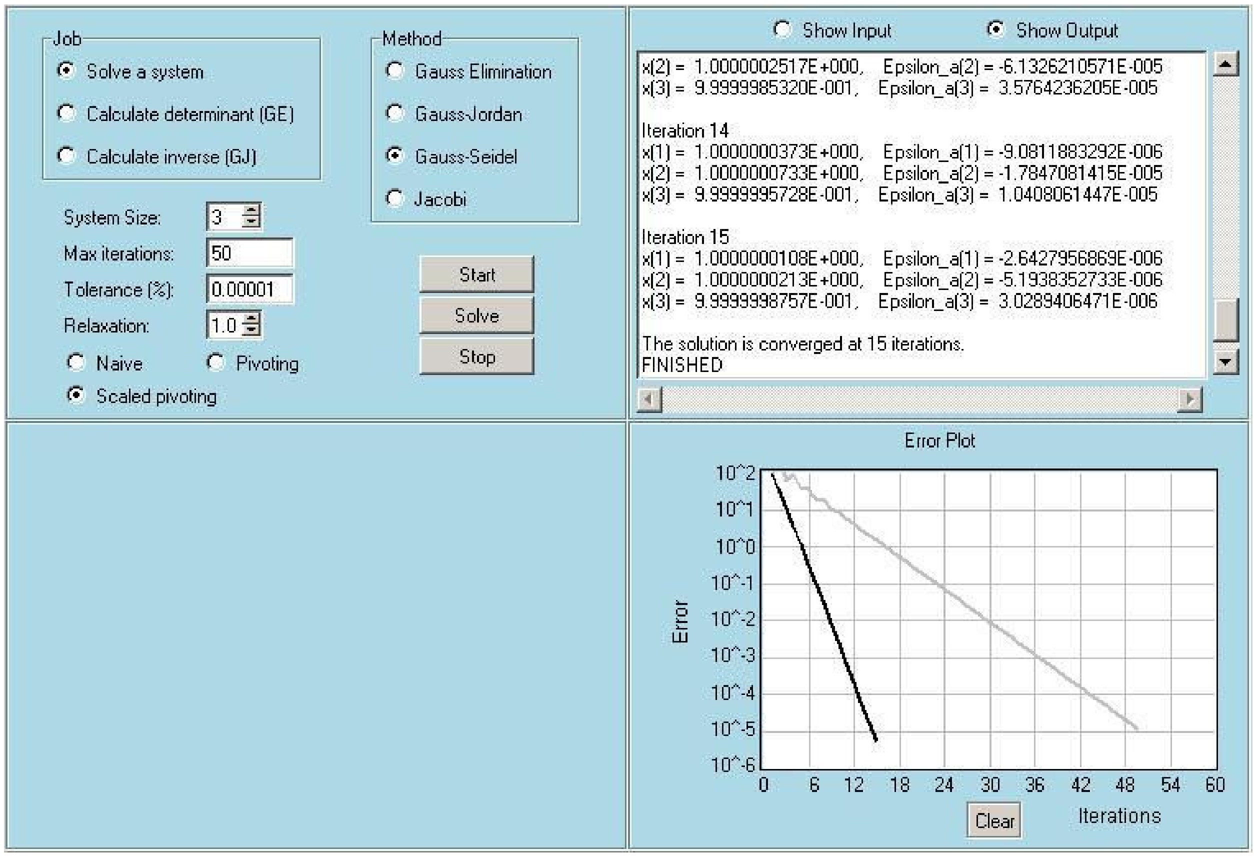
f(x): x*sin(x)
df/dx:
x0: 1
x1: 4
Max iterations: 100
Tolerance (%): 0.0001
Multiplicity for multiple roots (m): 1
Exact root is known to be
Method: [✓ Bisection]

```
f(x) = x*sin(x)
x0 = 1.0000000E+000
x1 = 4.0000000E+000
Maximum number of iterations = 100
Tolerance (%) = 0.0001
========================================================
iter 1 : x=2.5000000E+000, f(x)=1.4961804E+000, Epsilon_a=1.00E+002
========================================================
iter 2 : x=3.2500000E+000, f(x)=-3.5163419E-001, Epsilon_a=2.31E+001
========================================================
iter 3 : x=2.8750000E+000, f(x)=7.5740723E-001, Epsilon_a=1.30E+001
========================================================
iter 4 : x=3.0625000E+000, f(x)=2.4196879E-001, Epsilon_a=6.12E+000
========================================================
```

Figure 2. A sample run of the Roots of Equations module

2c. Linear Algebraic Equations Module

Linear Algebraic Equations module is used to solve systems of equations. It can also be used to calculate the determinant or the inverse of a matrix. A sample run of this module is shown in Fig. 3, which shows the solution of a 3x3 system using the Gauss-Seidel technique. To solve a system of equations, the user starts by selecting a method, either an elimination method (Gauss or Gauss-Jordan) or an iterative method (Gauss-Seidel or Jacobi). Than the system size should be specified and the appropriate boxes for the use of pivoting or scaled-pivoting should be checked depending on the choice. For iterative methods the user needs to provide the maximum number of iterations and a tolerance value. For iterative methods you can also specify a relaxation parameter to speed up the convergence. The entries of the coefficient matrix, right hand side vector and the initial guess vector are input using the Input window, which cannot be seen in Fig. 2. For the solution of a system of equations using an iterative method, the results after each iteration are shown in the Output window. In the sample run shown in Fig. 2, the convergence is achieved after 15 iterations and the results of the last three iterations can be seen. Error Plot window shows the convergence history. Black line corresponds to the Gauss-Seidel solution. Gray line is the convergence history for the same problem obtained from a previous run using the Jacobi method. Error Plot window allows plotting multiple lines like this, which gives a chance to compare of the performances of several methods. The user can also study the affects of the initial guess values or the relaxation parameter on the convergence by solving the same problem many times with different input parameters.



Job: [✓ Solve a system] Calculate determinant (GE) Calculate inverse (GJ)

Method: Gauss Elimination Gauss-Jordan [✓ Gauss-Seidel] Jacobi

System Size: 3

Max iterations: 50

Tolerance (%): 0.00001

Relaxation: 1.0

Naive Pivoting [✓ Scaled pivoting]

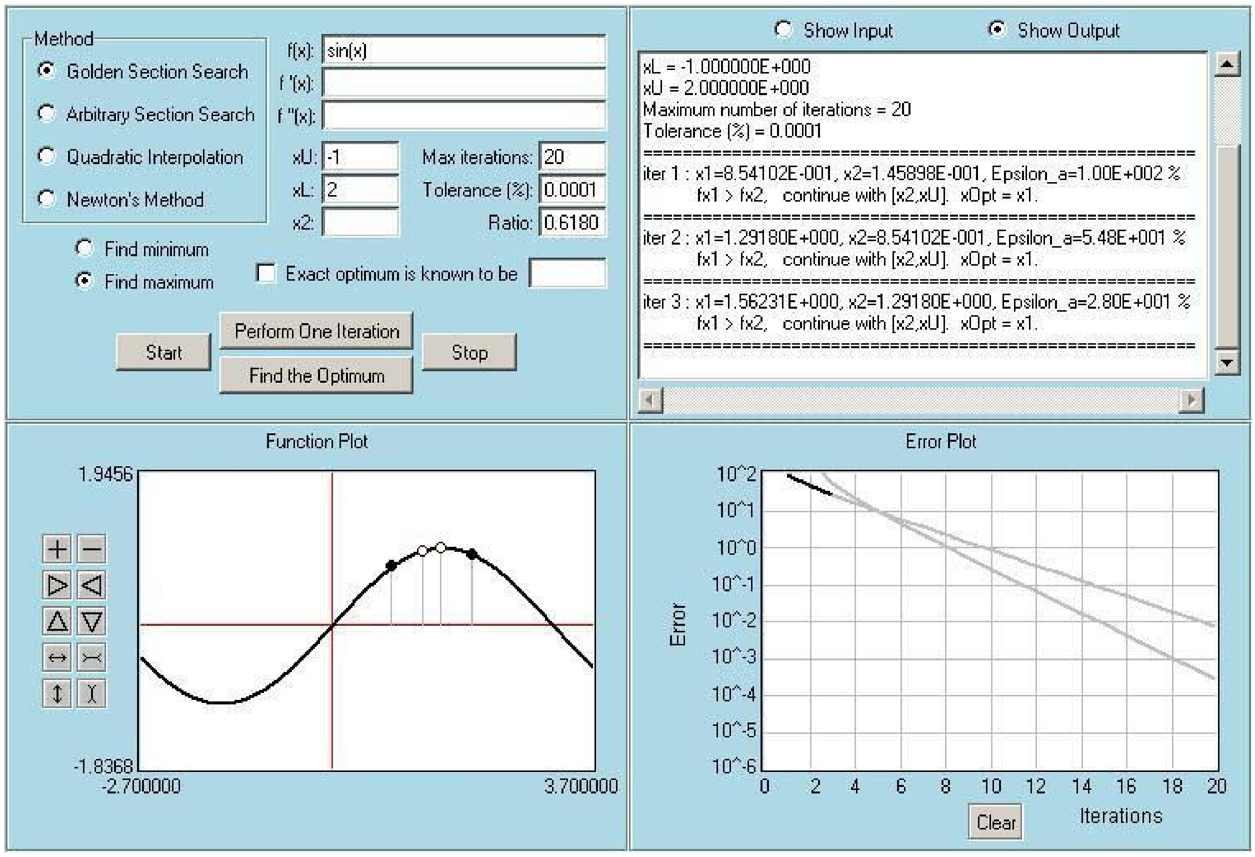
Show Input [✓ Show Output]

```
x(2) =  1.0000002517E+000,   Epsilon_a(2) = -6.1326210571E-005
x(3) =  9.9999985320E-001,   Epsilon_a(3) =  3.5764236205E-005

Iteration 14
x(1) =  1.0000000373E+000,   Epsilon_a(1) = -9.0811883292E-006
x(2) =  1.0000000733E+000,   Epsilon_a(2) = -1.7847081415E-005
x(3) =  9.9999995728E-001,   Epsilon_a(3) =  1.0408061447E-005

Iteration 15
x(1) =  1.0000000108E+000,   Epsilon_a(1) = -2.6427956869E-006
x(2) =  1.0000000213E+000,   Epsilon_a(2) = -5.1938352733E-006
x(3) =  9.9999998757E-001,   Epsilon_a(3) =  3.0289406471E-006

The solution is converged at 15 iterations.
FINISHED
```

Figure 3. A sample run of the Linear Algebraic Equations module

2d. Optimization Module

Optimization module is used for the solution of one-dimensional unconstrained optimization problems. A sample run of this module, used to find the first positive maximum of $f(x) = sin(x)$, is shown in Fig. 4. The solution starts by selecting a method among four possible choices. Arbitrary section search method is very similar to the golden section search, but instead of the golden ratio the user is free to select any ratio. The sample run shown in Fig. 4 uses the golden section search method, which is a bracketing method. For this method the user needs to provide two starting values that should bracket the desired optimum point. For this iterative method, maximum iteration number and the required tolerance need to be specified. The algorithm of the golden section search method also needs the information of whether the extreme point that user is searching for is a minimum or a maximum. After entering all the information the solution can be started and the results appear in the Output and Function Plot windows. Fig. 4 shows the solution after the third iteration. Golden section search method is based on finding an optimum point by reducing a specified interval from both sides by an amount proportional to the golden ratio. As seen from the Function Plot window, the two outer points came from the previous iteration and they set the starting interval of the fourth iteration. Two inner points are calculated during the third iteration and the interval will be reduced for the next iteration according to the details shown in the Output window. Error plot window shows two lines, one for this sample run, and the other for a run with the arbitrary section search method using a ratio of 0.5. Although the golden section search method uses less number of operations, its convergence is slower.


Method: [✓ Golden Section Search] Arbitrary Section Search / Quadratic Interpolation / Newton's Method

f(x): sin(x)
f '(x):
f ''(x):
xU: -1
xL: 2
x2:
Max iterations: 20
Tolerance (%): 0.0001
Ratio: 0.6180

Find minimum / [✓ Find maximum]

Exact optimum is known to be

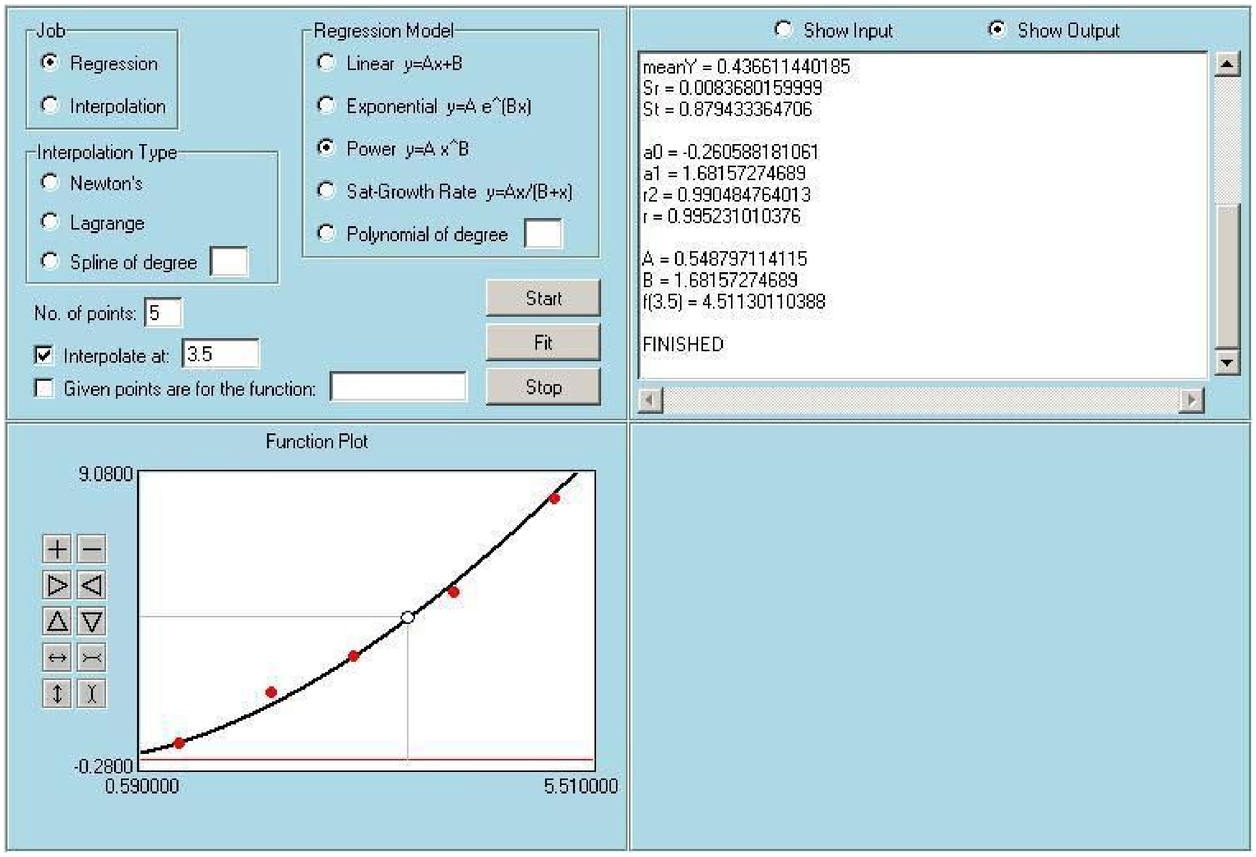
Show Input / [✓ Show Output]

```
xL = -1.000000E+000
xU = 2.000000E+000
Maximum number of iterations = 20
Tolerance (%) = 0.0001
==============================================================
iter 1 : x1=8.54102E-001, x2=1.45898E-001, Epsilon_a=1.00E+002 %
         fx1 > fx2,   continue with [x2,xU].  xOpt = x1.
==============================================================
iter 2 : x1=1.29180E+000, x2=8.54102E-001, Epsilon_a=5.48E+001 %
         fx1 > fx2,   continue with [x2,xU].  xOpt = x1.
==============================================================
iter 3 : x1=1.56231E+000, x2=1.29180E+000, Epsilon_a=2.80E+001 %
         fx1 > fx2,   continue with [x2,xU].  xOpt = x1.
==============================================================
```

Figure 4. A sample run of the Optimization module

2e. Curve Fitting Module

Curve Fitting module is used to find a function representation of a set of discrete points by using regression and interpolation techniques. A sample run of this module, used to fit a power function to a set of five points is shown in Fig. 5. First the user needs to select the type of curve fitting that should be performed, regression or interpolation. Than a model equation should be specified for regression and and interpolation type should be selected to perform interpolation. All possible choices can be seen in Fig. 5. Spline interpolation is available for degrees of 1, 2 and 3. In the sample run shown in Fig. 5, linear least squares regression will be used to fit a power function through a set of five points. The x and y values of these five points are entered in the Input window, which cannot be seen in Fig. 5. As the user hits the Fit button, the results of the regression are shown in the Output window. Details like the sum and the mean of x and y values, sum of the squares of the residuals between the given and fitted y values (S_r), sum of the squares of the residuals between the given y values and the mean (S_t), correlation coefficient (r), aswell as the the constants A and B appearing in the power equation are calculated. The specified data points and the fitted power equation can be seen in the Function Plot window. A quick visual check can be done for the quality of the fit. The performance of different model equations can easily be compared by performing one fit after the other. In this sample run, the user is specifically interested in the interpolated value at *x*=3.5 and the result for this can be seen both in the Output and the Function Plot windows.


Job: [✓ Regression] Interpolation

Interpolation Type: Newton's, Lagrange, Spline of degree

Regression Model: Linear y=Ax+B, Exponential y=A e^(Bx), [✓ Power y=A x^B], Sat-Growth Rate y=Ax/(B+x), Polynomial of degree

No. of points: 5

[✓] Interpolate at: 3.5

[ ] Given points are for the function:

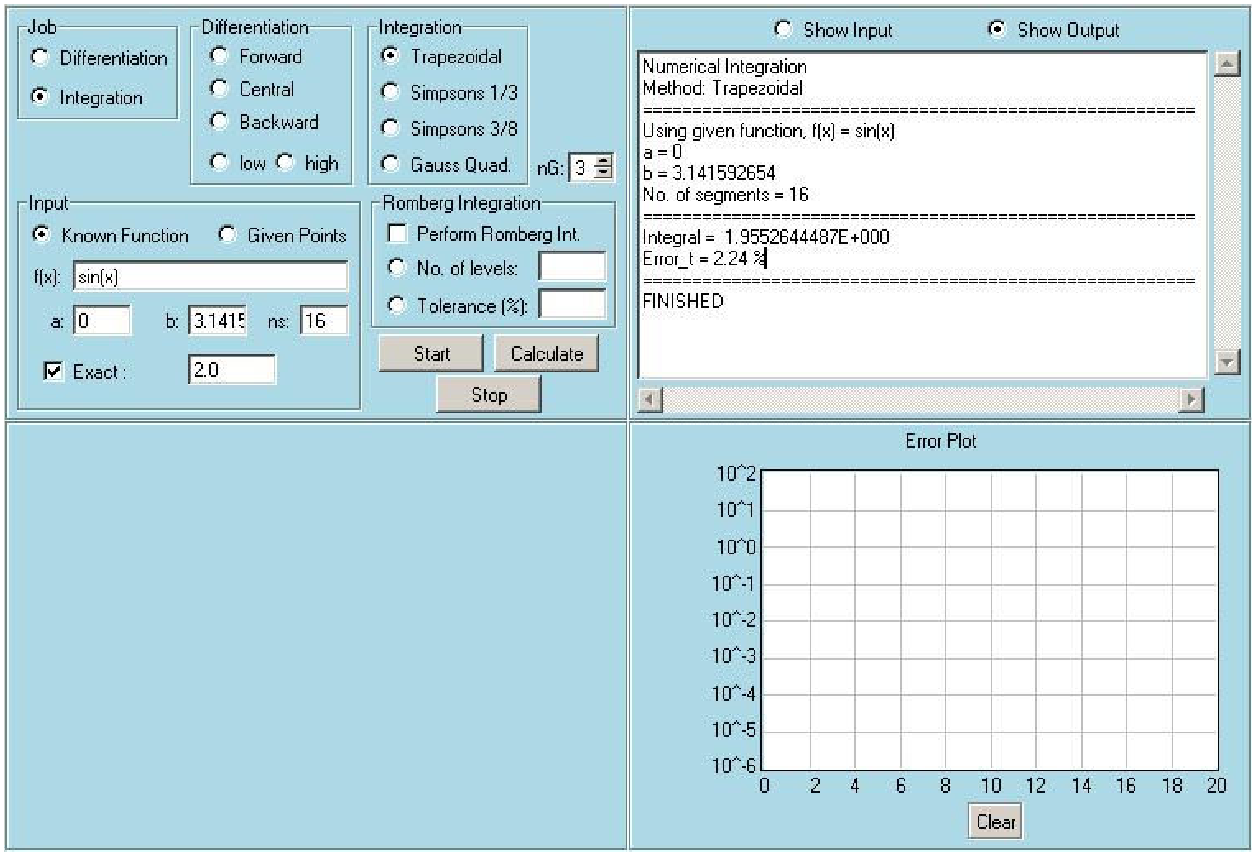
Show Input [✓ Show Output]

```
meanY = 0.436611440185
Sr = 0.0083680159999
St = 0.879433364706

a0 = -0.260588181061
a1 = 1.68157274689
r2 = 0.990484764013
r = 0.995231010376

A = 0.548797114115
B = 1.68157274689
f(3.5) = 4.51130110388

FINISHED
```

Figure 5. A sample run of the Curve Fitting module

2f. Differentiation and Integration Module

This module is used to perform numerical differentiation and integration. A sample run of this module, used to integrate *sin(x)* in the interval [0,π] using the Trapezoidal rule with 16 segments is shown in Fig. 6. The module can be used for forward, central and backward differentiation of two different orders. 1st and 2nd order formulas are used for forward and backward differentiation, whereas 2nd and 4th order formulas are used for central differentiation. Discrete data points can be entered by the user or they can be calculated from a known function. For integration of discrete data points, there are three possibilities, Trapezoidal rule and two different Simpson's rules. If the function is known, Gauss quadrature can be used for better accuracy. Another possibility for integrating known functions is the Romberg technique, the base of which is taken to be the Trapezoidal rule. The main advantage of the Romberg integration is its user defined stopping criteria. The user can select the levels of Romberg integration to be performed or specify a tolerance value as a stopping criteria of Romberg steps. In the sample run shown in Fig. 6, 16 segment Trapezoidal rule is used which resulted in a true percent relative error of 2.24 %. Although not shown in Fig. 6, the same integral can be calculated with a single segment three point Gauss quadrature formula with a true percent relative error of 0.07 %. The latter is more accurate and it uses less number of operations. For performance comparison purposes, it would be useful to know the number of floating point operations performed for each calculation, which can be implemented in the future releases of EasyNumerics.



Job: Differentiation [✓ Integration]

Integration: [✓ Trapezoidal] nG: 3

Input: [✓ Known Function] Given Points

f(x): sin(x)

a: 0 b: 3.1415 ns: 16

[✓ Exact :] 2.0

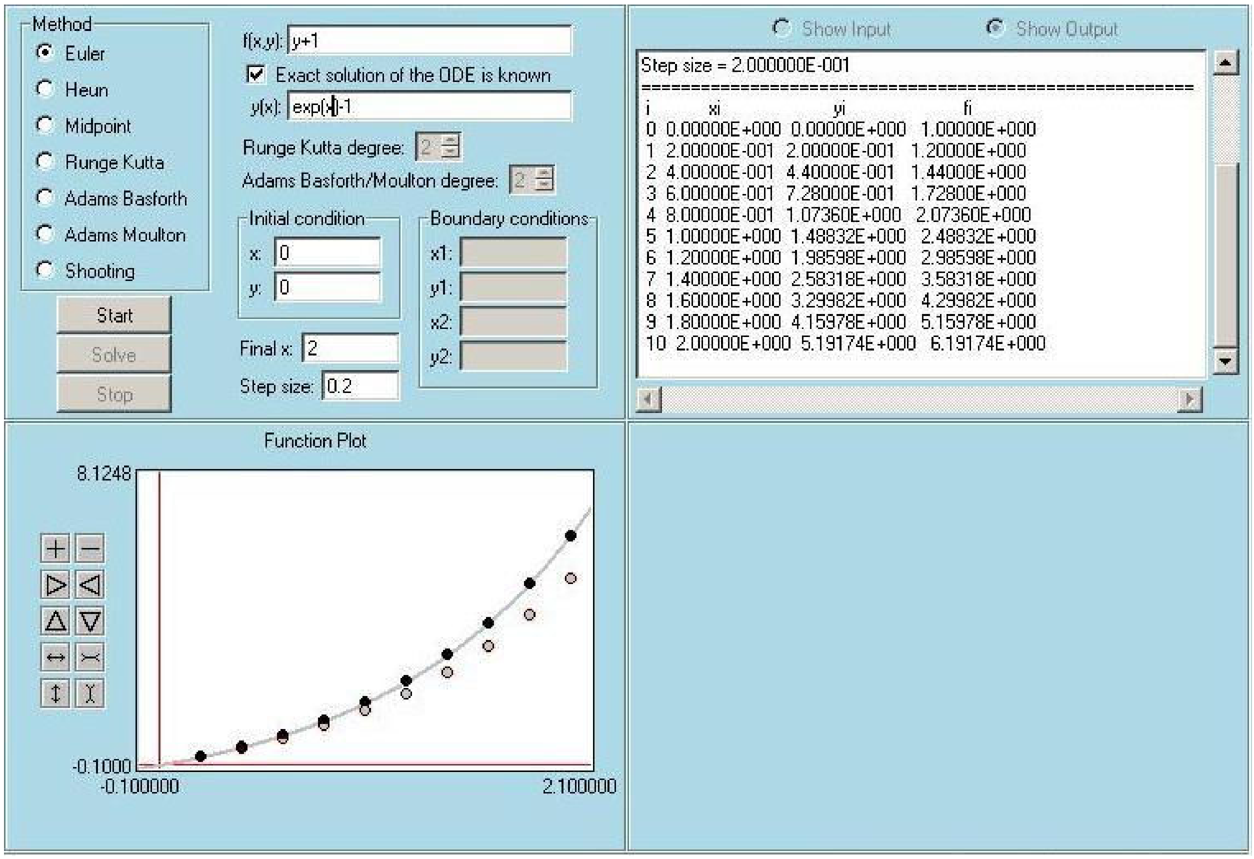
[✓ Show Output]

```
Numerical Integration
Method: Trapezoidal
=====================================================
Using given function, f(x) = sin(x)
a = 0
b = 3.141592654
No. of segments = 16
=====================================================
Integral =  1.9552644487E+000
Error_t = 2.24 %
=====================================================
FINISHED
```

Figure 6. A sample run of the Differentiation and Integration module

2g. Ordinary Differential Equations Module

This module is used to solve first order Ordinary Differential Equations (ODEs) in the form of *dy/dx=f(x,y)*. Figure 7 shows a sample run of this module that solves *dy/dx=y*+1 with the initial condition of *y(0)*=-1. With this module it is possible to solve initial value problems using six different methods, which can be seen in Fig. 7. Some of them are single step, some are multistep, some are self-starting and some are non-self-starting methods. There is also the shooting method that can be used to solve boundary value problems in a trial and error fashion. After selecting the method, initial or boundary conditions and the step size, the user can start the solution. The results can be seen in the Output and the Function Plot windows. For the sample run shown in Fig. 7, the Euler method is used to integrate $y'=y+1$ from $x=0$ to $x=2$, with a step size of 0.2. This requires computations of the known function *f* and the unknown *y* at ten steps. Results can be seen in the Output window. Computed *y* values are also plotted in the Function Plot window. Gray points are for this sample run, which uses the Euler method. Gray line is the given exact solution. Black points are for a previous run of the same problem with the 4th order Runge-Kutta method. As expected, this higher order method gives better results than the first order Euler method. When the exact solution is known, as in this example, exact errors can be computed, although this is not implemented in the current version of EasyNumerics. As demonstrated, the performance of different methods and the the affect of the step size can be studied easily.



f(x,y): y+1
[✓] Exact solution of the ODE is known
y(x): exp(x)-1
Runge Kutta degree: 2
Adams Basforth/Moulton degree: 2
Initial condition
x: 0
y: 0
Boundary conditions
x1:
y1:
x2:
y2:
Final x: 2
Step size: 0.2

```
Step size = 2.000000E-001
==========================================================
i        xi              yi              fi
0  0.00000E+000  0.00000E+000  1.00000E+000
1  2.00000E-001  2.00000E-001  1.20000E+000
2  4.00000E-001  4.40000E-001  1.44000E+000
3  6.00000E-001  7.28000E-001  1.72800E+000
4  8.00000E-001  1.07360E+000  2.07360E+000
5  1.00000E+000  1.48832E+000  2.48832E+000
6  1.20000E+000  1.98598E+000  2.98598E+000
7  1.40000E+000  2.58318E+000  3.58318E+000
8  1.60000E+000  3.29982E+000  4.29982E+000
9  1.80000E+000  4.15978E+000  5.15978E+000
10  2.00000E+000  5.19174E+000  6.19174E+000
```

Figure 7. A sample run of the ODE module

3. Conclusions

In this study a software called EasyNumerics is introduced. This software is intended to be used by the students of an entry level numerical methods course as a learning and practice tool. It can also be used by the instructors of such a course as an in-class demonstration program and as a tool to prepare and test assignment questions. EasyNumerics is an active software. It allows the solution of user designed problems. It is also a visual software and to achieve this it uses carefully designed Graphical User Interfaces. In this paper the capabilities of EasyNumerics are explained with screenshots that shows the solutions of many sample problems. EasyNumerics is a software under development and its current version can freely be downloaded for educational and academic purposes. Detailed help files and movies that shows the solution of sample problems can be found in its website[4].

Bibliography

1 Chapra S.C. and Canale R.P. *Numerical Methods for Engineers with Software and Programming Applications*, 4th ed., McGraw-Hill, 2002.
2 Cheney W. And Kincaid D. *Numerical Mathematics and Computing*, 5th ed., Thomson Brooks-Cole, 2004.
3 Mathews J.H. and Fink K.D. *Numerical Methods Using MATLAB*, 4th ed., Pearson-Prentice Hall, 2004.
4 http://www.me.metu.edu.tr/cuneyt/EasyNumerics
5 http://tcl.tk
6 Welch B.B. *Practical Programming in Tcl and Tk*, 3rd ed., Prentice Hall, 2000.

Biography

CUNEYT SERT is an assistant professor at the Mechanical Engineering Department of Middle East Technical University, Ankara, Turkey. His research interests include computational fluid dynamics, micro-fluidic transport, active-visual software development and distant education. Dr. Sert's contact information is available at his academic website; www.me.metu.edu.tr/cuneyt