Acta Mathematica Nitriensia
Vol. 5, No. 2, p. 1 – 10
ISSN 2453-6083

# Root Approximation in Matlab Computational Environment

Viliam Ďuriš 1[a] – Peter Korman 2[b]

[a*]*Department of Mathematics Faculty of Natural Sciences Constantine the Philosopher University in Nitra, Tr. A. Hlinku 1, 949 74 Nitra, Slovakia*
[b]*Lear Corporation Slovakia s.r.o., Priemyselný park Nitra, Dolné Hony 1, 949 01, Nitra*

Received September 14, 2019; received in revised form September 23, 2019; accepted September 28, 2019

**Abstract**

The task of solving non-linear equation occurs in practically all engineering disciplines. In the case of one equation, it is always possible to approximate the root within the required accuracy and to use some convergent method. The article deals with three basic convergent methods for roots approximation, namely bisection, tangent method and chord method, which are implemented and tested on several tasks in solving non-linear equations in Matlab computational environment as a suitable tool for implementation of various numerical methods.

**Keywords**: root approximation, bisection, Newton's method, false-proposition, Matlab

**Classification**: 49M15, 65H04

## 1 Introduction

In numerical mathematics we often seek algorithmically approximate solution of analytically insoluble problems or problems for which the analytical solution is very lengthy. Root approximation is a problem that occurs very often in the natural sciences or engineering practice to solve various tasks. For approximation of roots, thus solving equations in the form $f(x) = 0$, there are several known numerical methods that are based on recursive formulas. Recurrence-based computation methods, where each member of a sequence is defined as a function of a previous formulas, are found at the beginning of mathematics by Babylonians or Greeks. The Babylonians used them to calculate the square root of positive numbers and the Greeks to approximate the number $\pi$ [1].

A suitable computational environment for the implementation of various algorithms of numerical mathematics is the Matlab programming environment (a term which was created as an abbreviation from the Matrix Laboratory) [2]. Matlab is a suitable and powerful language to work with any calculation in teaching, industry or research with extensive possibilities in the creation and use of various structures. The Matlab computational environment enables to perform various mathematical calculations, implement various algorithms, measure, analyse and statistically process various data, model and simulate different and even infeasible events, design various systems with user interface, create graphs and so on. Matlab is an object-oriented environment and directly utilizes the features of the operating system (e.g. when working with files), which simplifies the design phase as much as possible, so we can spend more time on the algorithm itself when implementing various numerical algorithms. Matlab

*Corresponding author; email: vduris@ukf.sk
DOI: 10.17846/AMN.2019.5.2.1-10

consists of a computational unit that performs various numeric operations with real or complex number matrices, a programming language used to write algorithms using language commands, a working environment that includes various command processing tools or data import and export, graphic system that allows you to create user environments, work with images or animations, libraries of mathematical functions, and various embedded algorithms such as global optimum search algorithms, API interface for implementing C or Fortran scripts, Toolboxes with various optional libraries of specialized functions (e.g. for fuzzy logic, neural networks, statistics), and an open architecture for implementing various systems. A very effective and standalone extension of the Matlab system is the graphical interactive Simulink program, which allows you to simulate and model various dynamic systems using graphics schemes using Matlab functions and commands.

## 2 Numerical methods for finding roots

In the Matlab computational environment, the roots of a polynomial function can be searched for directly by the built-in `roots` function, where the coefficients of the polynomial are determined by the vector. Consider, for example, function $f(x) = -x^4 + 3x^2 + 2x$ and find its roots. First, a coefficient vector must be created (which must also include zero coefficients in order not to reduce the degree of the polynomial).

```
>> c = [-1 0 3 2 0];
>> roots(c)

ans =
         0
    2.0000
   -1.0000
   -1.0000
```

Given function and its roots can be viewed in the chart (Figure 1).

```
x = -2.5:0.005:2.5;
y = -x.^4 + 3*x.^2 + 2*x;
h = plot(x, y);
set(h, 'LineWidth', 3);
grid on
```

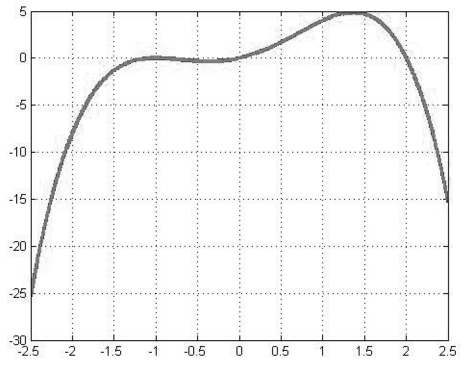

**Figure 1:** Function Graph $f(x) = -x^4 + 3x^2 + 2x$.

For any function (not only in the case of a polynomial), if an algebraic equation $f(x) = 0$ is given with real coefficients having within an interval $(a, b)$ exactly one real root $k$, we can approximate it with the ab ante desired accuracy $\varepsilon$ using the `fzero` function, also built-in directly in Matlab. The `fzero` function can be called with either a two-element vector representing the interval $(a, b)$ or only with a starting point $x_0$ from which the search is to be started. In this case, the `fzero` function first finds such an interval around the starting point at which the function $f(x)$ changes its sign. If it does not find such an interval, it returns `NaN`. However, if we know the appropriate interval $(a, b)$ beforehand, it is guaranteed that `fzero` function successfully returns a value close to the root of the equation. Now consider the function $f(x) = x^3 - 7x + 1$ on the interval $(0,1)$ (Figure 2) and find the solution to the equation $f(x) = 0$.

```
>> f = inline('x^3 - 7*x + 1');
>> k = fzero(f, [0 1])
k = 0.1433
```

If you need to see the exact steps of the `fzero` function, you can set their display using the `Display` parameter of the `options` [3] structure. The accuracy $\varepsilon$ is also possible to be set with `optimset` function.

```
>> setdsp = optimset('Display', 'iter');
>> k = fzero(f, [0 1], setdsp)
 Func-count     x          f(x)             Procedure
    2               0            1          initial
    3        0.166667    -0.162037          interpolation
    4        0.143426   -0.00103363         interpolation
    5        0.143277  5.13222e-007         interpolation
    6        0.143277 -4.73821e-012         interpolation
    7        0.143277 -2.22045e-016         interpolation
    8        0.143277 -2.22045e-016         interpolation
Zero found in the interval [0, 1]
k = 0.1433
```

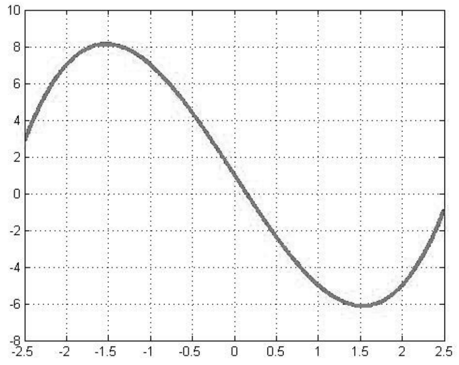

**Figure 2:** Function Graph $f(x) = x^3 - 7x + 1$.

We can also solve non-linear equations numerically using the method of half division of the interval (so-called bisection) [4]. Let the interval $\langle a, b\rangle$ be one of the intervals containing the root of the equation $f(x) = 0$. We will find a solution in this interval if $f(a) \cdot f(b) < 0$. Let's define a sequence of intervals $\langle a_n, b_n\rangle$ with the following properties (Figure 3):

1. $\langle a_1, b_1\rangle = \langle a, b\rangle$

2. Next, we take the middle of the interval $c_1 = \frac{a_1+b_1}{2}$ and if this point is the solution to that equation, we will terminate the process, otherwise $f(c_1) \neq 0$. Then either $f(a_1) \cdot f(c_1) < 0$ and then $\langle a_2, b_2 \rangle = \langle a_1, c_1 \rangle$, or $f(b_1) \cdot f(c_1) < 0$ and then $\langle a_2, b_2 \rangle = \langle c_1, b_1 \rangle$.
3. Suppose we defined such an interval $\langle a_n, b_n \rangle$ that $f(a_n) \cdot f(b_n) < 0$. We take the middle of the interval $c_n = \frac{a_n+b_n}{2}$ again. If this point is the solution of the given equation we will terminate the process, otherwise $f(c_n) \neq 0$. Then either $f(a_n) \cdot f(c_n) < 0$ and then $\langle a_{n+1}, b_{n+1} \rangle = \langle a_n, c_n \rangle$, or $f(b_n) \cdot f(c_n) < 0$ and then $\langle a_{n+1}, b_{n+1} \rangle = \langle c_n, b_n \rangle$.

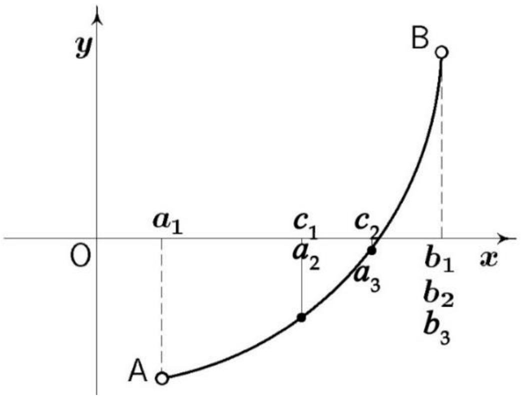

**Figure 3:** Method of bisection.

If the sequence $\{c_n\}$ has a finite number of members, then the last is the root of the equation $f(x) = 0$. If it is infinite, then it has a finite limit, which is the exact solution to our equation. At each step, the length of the interval was reduced by half, and thus for an approximate solution $c_n$ the estimate $|c_n - k| < \frac{b-a}{2^n}$ applies, where $k$ is the exact solution. To determine the approximate solution with accuracy $\varepsilon > 0$, then we terminate the process of dividing the interval when $|b_n - a_n| < 2\varepsilon$ and we will accept an approximate solution $c_n = \frac{a_n+b_n}{2}$. From the inequality $|c_n - k| < \frac{b-a}{2^n}$ (if its right side is less than $\varepsilon$) we get $n > \frac{1}{\ln 2} \ln\left(\frac{b-a}{\varepsilon}\right)$, which represents the number of iterations needed for accuracy $\varepsilon$. If the conditions are satisfied that the function is continuous and assumes different characters at the ends of a given interval, then this method is always convergent. The following function (such as the m-file `bisection.m`) presents the method of bisection in the Matlab environment.

```
function [f_root, term_type] = bisection(funct, tol, maxiter,
                                               a, b)
iterations = 0;
term_type = 0;
f_root = NaN;
f_a = feval(funct, a);
f_b = feval(funct, b);

while (f_a * f_b<0) && (iterations<maxiter) && ((b - a)>tol)
  c = (b + a) / 2;
  f_c = feval(funct, c);
  if (f_c * f_a) < 0
```

```
      b = c;
      f_b = f_c;
    else
      a = c;
      f_a = f_c;
    end
    iterations = iterations + 1;
  end
   if (iterations == maxiter)
    term_type = 1;
  else
    f_root = c;
    term_type = iterations;
  end
   end
```

Now we can find the root of the equation using the created function $f(x) = x - 2\,sin\,x^2$ on the interval $\langle 0.5,1\rangle$. We create the function $f(x)$ as a separate m-file to call as a real parameter of the formal `funct` parameter in the `bisection` function.

```
  function y = f(x)
  y = x - 2*sin(x.^2);
```

To call the `bisection` function, run the script (`bisect.m` file) specified below for accuracy $\varepsilon = 10^{-4}$ and a maximum of 50 iterations allowed.

```
  [f_root, term_type] = bisection('f', 1e-4, 50, 0.5, 1);
   if (term_type == 0)
    disp('The interval does not contain the root or contains
                                              more roots!')
  elseif (term_type == 1)
    disp('The root could not be found within the specified
                                       number of iterations!')
  else
    disp(['Root = ' num2str(f_root) ' found in '
                           num2str(term_type) ' iterations.'])
  end
```

We get the result (Figure 4):

```
  >> bisect
  Root = 0.50543 found in 13 iterations.
```

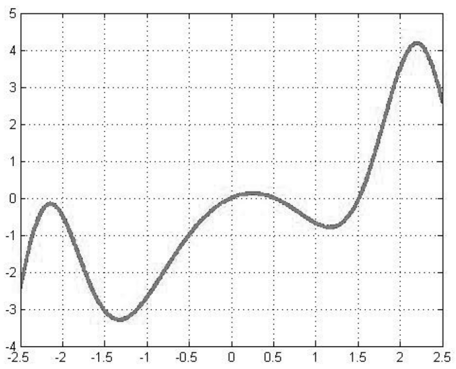

**Figure 4:** Function Graph $f(x) = x - 2\,sin\,x^2$.

Another option for finding the roots of a given function is to use the chord method (the so-called false-proposition method) [5]. Assume $f(a) \cdot f(b) < 0$. The chord method consists in replacing the function $f(x)$ in the interval $(a, b)$ by a chord given by points $A\big(a, f(a)\big)$, $B\big(b, f(b)\big)$, whose equation is $y - f(a) = \frac{f(b)-f(a)}{b-a}(x - a)$ (Figure 5).

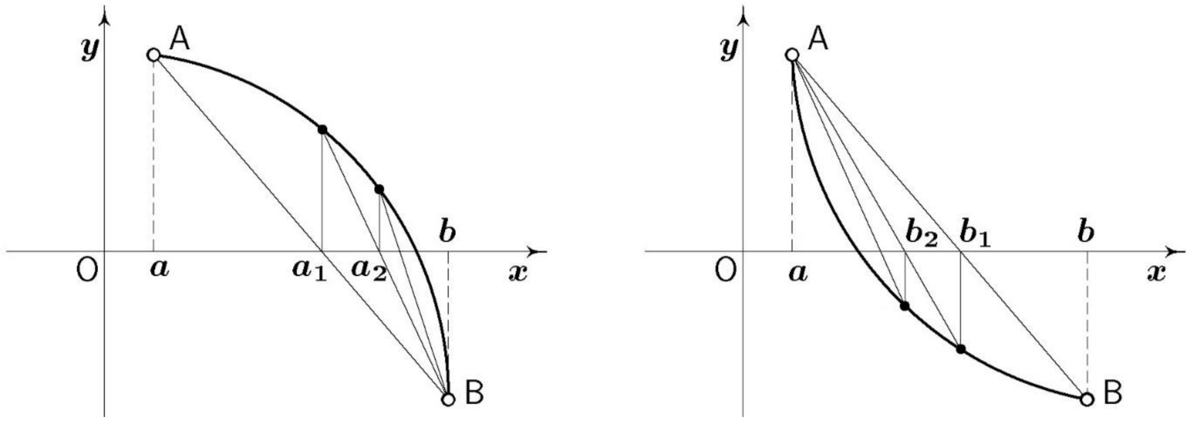

**Figure 5:** Method of chords.

Lay $y = 0$ and calculate the intersection point $x_1 = a - \frac{b-a}{f(b)-f(a)} f(a)$ with $x$ axis. If the sign of $f(x_1)$ equals the sign of $f(a)$, we lay $x_1 = a_1$. Proceed in this way and get approximations $a_2 = a_1 - \frac{b-a_1}{f(b)-f(a_1)} f(a_1)$ , $a_3$, $a_4$, ... that converge to the root $k$. If the sign of $f(x_1)$ equals the sign of $f(b)$, we lay $x_1 = b_1$, then $b_1 = a - \frac{b-a}{f(b)-f(a)} f(a)$ as well as in the next process we get approximations $b_2 = a - \frac{b_1-a}{f(b_1)-f(a)} f(a)$ , $b_3$ , $b_4$ , ... converging to the root. The chord method algorithm as a function written in Matlab then looks like [6]:

```
function f_root = false_proposition(f, tol, a, b)

fa = feval(f, a);
fb = feval(f, b);
 while (abs(fa) > tol) && (abs(fb) > tol)
  x1 = a - (b - a) / (fb - fa) * fa;
  fx1 = feval(f, x1);
  if (fx1 * fa) > 0
    a = x1;
  else
    b = x1;
  end
  fa = feval(f, a);
  fb = feval(f, b);
end
 f_root = x1;
 end
```

Now we find the root of the equation $x^3 - 5x^2 - 16x + 53 = 0$ in the interval $\langle 2,3 \rangle$ by the chord method with precision of $\varepsilon = 10^{-4}$.

```
>> format long
>> f = inline('x^3 - 5*x^2 - 16*x + 53');
>> f_root = false_proposition(f, 1.e-4, 2, 3)

f_root = 2.38347756851597
```

The tangent method (the Newton method) belongs among the important methods of roots approximation. The development of mechanics in the 17th century played its role in solving the problem of tangents, seeking their analytical expression and constructions. The correlation of physical factors of movement with the curve geometry of the moving point curve has crystallized into the concept that the direction of movement at each point of the trajectory is determined by the tangent to the curve at that point. The first concepts of tangents date back to antiquity [7], according to which the tangent line had one point in common with the curve. In modern differential geometry for higher-grade algebraic curves and transcendental curves, the original static concept of tangent has been replaced by the dynamic understanding of the tangent as the limit position of the secant with "infinitely close" intersections with the curve. In such a view of the tangent, various physical factors stemming from Galileo's theory of motion could have entered into the tangent theory. Isaac Newton (1642 - 1727), a mathematician and physicist, proved the solution to the tangent problem from the perspective of infinitesimal calculus by his methods and he described his methods of finding approximate algebraic solutions in 1699 in his work *De analysi per aequationes numero terminorum infinitas,* published 12 years later by British mathematician William Jones. In 1740, based on this work, British mathematician Thomas Simpson introduced a new iterative method for solving general non-linear equations.

Suppose a given function $y = f(x)$ has a derivative. We choose the initial root approximation $x_0$. We run the tangent to the graph of function $f$ through the point $[x_0, f(x_0)]$. Mark $x_1$ its intersection with the $x$ axis. Then we lead the tangent through the point $[x_1, f(x_1)]$. Mark $x_2$ its intersection with the $x$ axis. This proceeds further (Figure 6) [8].

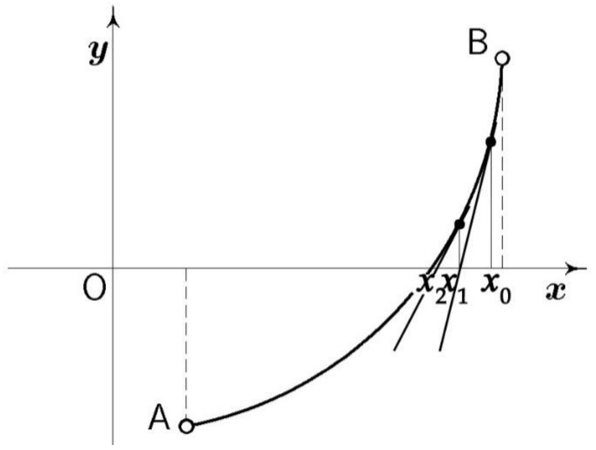

**Figure 6:** Newton method.

Suppose we know $x_k$ and we want to calculate a closer approximation $x_{k+1}$. We run the tangent through the point $[x_k, f(x_k)]$ to the curve $y = f(x)$. Substitute to the tangent equation

$$y = f(x_k) + f'(x_k)(x - x_k)$$

by $y = 0$ to obtain the intersection of the tangent line with the $x$ axis:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

We will now deduce the error of this method. Let $e_k = x_k - x^*$ is an error in the $k$-th step. Let us make Taylor expansion [9] $f(x^*)$ around $x_k$. Now suppose that the second derivative also exists. It is valid from the definition of Taylor expansion that its sum around the point equals the functional value at this point, so

$$0 = f(x^*) = f(x_k) + (x^* - x_k)f'(x_k) + \tfrac{1}{2}(x^* - x_k)^2 f''(\varphi),$$

where $\varphi$ is a point of the interval whose limit values are $x_k$ and $x^*$.
After simplification we get

$$-\frac{1}{2}(x^* - x_k)^2 \frac{f''(\varphi)}{f'(x_k)} = \frac{f(x_k)}{f'(x_k)} + (x^* - x_k)$$

$$-\frac{1}{2}(x^* - x_k)^2 \frac{f''(\varphi)}{f'(x_k)} = x^* - \left[x_k - \frac{f(x_k)}{f'(x_k)}\right] = x^* - x_{k+1}$$

$$\frac{1}{2}e_k^2 \frac{f''(\varphi)}{f'(x_k)} = e_{k+1}$$

Then

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^2} = \frac{|f''(\varphi)|}{|f'(x_k)|}$$

And we see that the Newton method converges quadratically. We can now find the root of the equation $f(x) = x - 2\sin x^2$ using the Newton method. For this purpose we created 3 functions in Matlab. The first represents the given function, the second represents its derivative:

```
  function y = f(x)
  y = x - 2*sin(x.^2);
  function y = f2(x)
  y = 1 - 4*x.*cos(x.^2);
```

The third function is the algorithm of the Newton method itself.

```
  function [f_root, term_type] = newton(tol, x, maxiter)
   iterations = 0;
   while (iterations < maxiter) && (abs(f(x)) > tol)
    x = x - f(x) / f2(x);
    iterations = iterations + 1;
  end
  if (iterations == maxiter)
    term_type = 1;
  else
    f_root = x;
    term_type = iterations;
  end
   end
```

We call the `newton` function by running the script ( `newton2.m` ) lower with precision $\varepsilon = 10^{-4}$ and a maximum of 50 iterations allowed.

```
[f_root, term_type] = newton(1e-4, 1, 50);

if (term_type == 1)
  disp('Root not found!')
else
  disp(['Root = ' num2str(f_root) ' found in '
                       num2str(term_type) ' iterations.'])
end
```

We get the result:

```
>> newton2
Root = 0.50548 found in 4 iterations.
```

We can see that compared to the bisection method, we have obtained a root approximation with a significantly lower number of iterations due to quadratic convergence. The Newton method can also diverge. However, the method always converges, provided that the initial approximation is sufficiently close to the root. By appropriate combining the bisection method with the Newton method, it is possible to construct a combined method that always converges.

## Conclusion

Mathematics provides a variety of analytical or algebraic tools to solve practical problems from different industries. However, many practical tasks result from the compilation of such equations or functions that are analytically insolvable or very difficult to solve. Thanks to numerical mathematics and computers, we can now construct algorithms for virtually every problem. These algorithms are able to find approximate (or even accurate) solutions within the accuracy we require, and our analytical task remains to determine the error estimate of the inaccurate solution. Root approximation means expressing the problem of solving the equation in a form useful in the field of iterative processes, which is also the bisection method, the chord method or the Newton method, obtaining a root value that is as accurate as possible to the true value. The roots approximation in the Matlab computational environment in terms of computations includes convergent methods, by means of which we can find solutions of non-linear equations with any degree of accuracy and thus Matlab proves to be a very suitable tool for implementing algorithms of various numerical methods.

## References

1. Pickover C. A. (2011). *The Math Book: From Pythagoras to the 57th Dimension, 250 Milestones in the History of Mathematics*. New York, NY: Sterling Publishing, ISBN: 9781402757969.

2. Higham D. J., Higham N. J. (2017). *MATLAB Guide, 3e*. USA: SIAM, ISBN: 9781611974652.

3. Mathworks. (2019). *Online documentation*. Available at: https://www.mathworks.com/help/matlab/ref/optimset.html, accessed 7th of June, 2019.

4. Otto S. R., Denier J. P. (2005). *An introduction to Programming and Numerical Methods in MATLAB*. London: Springer-Verlag London Limited, ISBN: 9781852339197.
5. Palumbíny D., Palumbíny O. (2002). *Algebra 2*. Nitra: Constantine The Philosopher University, ISBN: 8080505454.
6. Fulier J., Ďuriš V., Frantová P. (2007). *Systémy počítačovej algebry CAS vo vyučovaní matematiky*. Nitra: Constantine The Philosopher University, ISBN 9788080941390.
7. Čižmár J. (2017). *Dejiny matematiky – od najstarších čias po súčasnosť*. Bratislava: PERFECT, Slovak Republic, ISBN: 9788080468293.
8. Polyak B. T. (2007). *Newton's method and its use in optimization*. In: European Journal of Operational Research, Vol. 181, No. 3, p. 1086-1096, DOI: 10.1016/j.ejor.2005.06.076.
9. Plofker K. (2001). *The "Error" in the Indian "Taylor Series Approximation" to the Sine*. In: Historia Mathematica, Vol. 28, No. 4, p. 283–295, DOI:10.1006/hmat.2001.2331.