Implementation of Numerical Solutions for Nonlinear Equations using MATLAB

Hnin Hnin Khaing¹ & Yee Yee Htun²

¹Hnin Hnin Khaing & ²Yee Yee Htun
¹Lecturer, Department of Engineering Mathematics, TU (Maubin), Ayarwaddy Division
²Lecturer, Department of Engineering Mathematics, TU (Hmawbi), Yangon Division
Myanmar

Abstract: Problems that most frequently encountered are nonlinear equations in sciences and engineering problems. In this paper, we will focus on MATLAB solutions to nonlinear equations by studying various methods. In this paper, the numerical methods for solving nonlinear equations using MATLAB can be carried out. This presents the most widely used iterative methods for nonlinear equations and MATLAB features for finding numerical solutions. The numerical examples are considered and implemented in this paper.

Keywords: nonlinear equations, MATLAB, numerical methods, iteratives methods.

1. INTRODUCTION

Compare to linear algebraic equations, most frequently encountered are nonlinear equations in sciences and engineering problems. Solving nonlinear equations could be computationally expensive; therefore, the solving of approximate linear equations was indispensable especially in the early times when the computers were not powerful enough. Today, with the rapid development of computing technology, solving directly nonlinear equations is becoming increasingly important. In this paper, various methods of solving nonlinear equations problems are studied and we will focus on MATLAB solutions to nonlinear equations. The three methods of solutions to nonlinear algebraic equations will be presented in this technical approach paper. The graphical method for nonlinear equations with one and two unknown variables can be analysis with polynomial equations. Numerical solutions to nonlinear equations and nonlinear matrix equations can also be implemented in this paper. (Dingyü Xue, 2009).

Graphical and numerical methods will be presented and the simplest solution is implemented using MATLAB. In this paper, the linear programming, quadratic programming and general nonlinear programming will be studied and MATLAB-based solutions will be carried out. As a procedure, the theory background of Bisection Method, Secant Method and Newton Raphson Method of solving nonlinear equations problems are studied and carried out. Then, to be approached to simplest and fast way, MATLAB instructions relative to nonlinear equations solving process are also studied and tested. Actually, various possible problems are applied to three methods and tested. But, only the main points
or examples are expressed in this paper as a portion of Mathematical approach to Engineering Problems.

2. NUMERICAL METHODS FOR NONLINEAR EQUATIONS

2.1 Bisection method

The bisection method is derived from the Intermediate Value Theorem, which states that if a continuous function \( f \), with an interval \([a, b]\) as its domain, takes values \( f(a) \) and \( f(b) \) at each end of the interval and assume \( f : [a, b] \to \mathbb{R} \) is a continuous function and that \( f(a)f(b) < 0 \). Also, assume that the interval \([a_n, b_n]\) contains only one root \( \alpha \) of \( f \). Then the idea is that at each step, we bisect the interval, so that \( f \) has opposite signs at the endpoints of the interval. This will produce a sequence of embedded intervals \([a_n, b_n]\), such that for every \( n \in \mathbb{N} \), \( \alpha \in [a_n, b_n] \) and \( f(a_n)f(b_n) < 0 \).

The idea behind the Intermediate Value Theorem can be stated: when we have two points connected by a continuous curve, as shown in Picture 1:

- one point below the line
- the other point above the line
- then there will be at least one place where the curve crosses the line.

The first step in iteration is to calculate the mid-point of the interval \([a, b]\). If \( c \) be the mid-point of the interval, it can be defined as: \( c = (a+b)/2 \). The function is evaluated at ‘c’, which means \( f(c) \) is calculated.

- \( f(c) = 0 : c \) is the required root of the equation.
- \( f(b) * f(c) > 0 : \) if the product of \( f(b) \) and \( f(c) \) is positive, the root lies in the interval \([a, c]\).
- \( f(b) * f(c) < 0 : \) if the product of \( f(b) \) and \( f(c) \) is negative, the root lies in the interval \([b, c]\).

In the second iteration, the Intermediate value theorem is applied either in \([a, c]\) or \([b, c]\), depending on the location of roots. And then, the iteration process is repeated by updating new values of \( a \) and \( b \). Picture 2 shows how iteration done in bisection method.
2.2 Secant Method

This method uses two initial guesses and finds the root of a function through interpolation approach. For each successive iteration, two of the most recent guesses can be used as two most recent fresh values to find out the next approximation. Features of Secant Method can be shortly expressed as:

- No. of initial guesses – 2
- Type – open bracket
- Rate of convergence – faster
- Convergence – super linear
- Accuracy – good
- Approach – interpolation
- Programming effort – tedious

The Procedure of Secant Method can be shown as flowchart as in Picture 3 compare with Newton Raphson Method and this will give the algorithm to implement in MATLAB.

(a) Secant Method Algorithm
(b) Newton Raphson Method Algorithm

Picture 3. Comparison of Secant Method and Newton Raphson (www.codewithc.com)

2.3 Newton Raphson Method

The theoretical and mathematical background behind Newton-Raphson method and its MATLAB program (or program in any programming language) is approximation of the given function by tangent line with the help of derivative, after choosing a guess value of root which is reasonably close to the actual root. The x-intercept of the tangent is calculated by using elementary algebra, and this calculated x-intercept is typically better approximation to the root of the function. This procedure is repeated till the root of desired accuracy is found. Lets now go through a short mathematical background of Newton’s method. For this, consider a real value function f(x) as shown in the Picture 3(c).
Let’s try the example problem using Newton-Raphson method, solving it numerically. The function is to be corrected to 9 decimal places. For a given function: \( x^3 - x - 1 = 0 \), which is differentiable,

The first derivative of \( f(x) \) is \( f'(x) = 3x^2 - 1 \).

Let us determine the guess value:

\[
\begin{align*}
f(1) &= 1 - 1 - 1 = -1 \\
f(2) &= 8 - 2 - 1 = 5
\end{align*}
\]

Therefore, the root lies in the interval \([1, 2]\). So, assume \( x_1 = 1.5 \) as the initial guess root of the function \( f(x) = x^3 - x - 1 \).

\[
\begin{align*}
f(1.5) &= 1.5^3 - 1.5 - 1 = 0.875 \\
f''(1.5) &= 3 \times 1.5^2 - 1 = 5.750
\end{align*}
\]

Using Newton’s iteration formula:

\[
x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 1.5 - \frac{0.875}{5.750} = 1.34782600
\]

The iteration for \( x_3, x_4, \ldots \) is done similarly.

3. MATLAB FUNCTIONS FOR SOLVING NONLINEAR EQUATIONS

3.1 Function “roots”

The Syntax : \( r = \text{roots}(p) \) returns the roots of the polynomial represented by \( p \) as a column vector. Input \( p \) is a vector containing \( n+1 \) polynomial coefficients, starting with the coefficient of \( x^n \). A coefficient of 0 indicates an intermediate power that is not present in the equation. This function can study as shown in Picture 4(a). For example: \( p = [3 \ 2 -2] \) represents the polynomial \( 3x^2 + 2x - 2 \). The roots function solves polynomial equations of the form \( p_1x_n + \ldots + p_nx + p_{n+1} = 0 \). Polynomial equations contain a single variable with nonnegative exponents.

3.2 Function: “fzero”

Root of nonlinear function (fzero) can be write in syntax as:

\[
x = \text{fzero}(\text{fun}, x0), x = \text{fzero}(\text{fun}, x0, \text{options}), x = \text{fzero}(\text{problem}) \quad \text{and} \quad [x, \text{fval}, \text{exitflag}, \text{output}] = \text{fzero}(__).
\]
We can find the real root (with initial guess \(x_0=1\)) instead of using “roots()” function mentioned in 3.1, with:

\[
\text{>> } x = \text{fzero}(f,1)\\
\text{x = 1.3247}
\]

The example of using function “fzero” is shown in Picture 4(b).

### 3.3 Function: “inline”

The constructing of inline object can be scriptable in MATLAB as the fucncitons: inline (expr), inline (expr,arg1,arg2,...) or inline (expr,n). “inline (expr)” constructs an inline function object from the MATLAB expression contained in “expr”. The input argument to the inline function is automatically determined by searching expr for an isolated lower case alphabetic character, other than i or j, that is not part of a word formed from several alphabetic characters. If no such character exists, x is used. If the character is not unique, the one closest to x is used. If two characters are found, the one later in the alphabet is chosen. “inline (expr,arg1,arg2,...)” constructs an inline function whose input arguments are specified by arg1, arg2,... Multicharacter symbol names may be used. “inline (expr,n)” where n is a scalar, constructs an inline function whose input arguments are x, P1, P2, ...

Three commands related to inline allow you to examine an inline function object and determine how it was created. “char(fun)” converts the inline function into a character array. This is identical to “formula(fun)”. “argnames(fun)” returns the names of the input arguments of the inline object fun as a cell array of character vectors. “formula(fun)” returns the formula for the inline object fun. A fourth command “vectorize(fun)” inserts a . before any ^, * or / in the formula for fun. The result is a vectorized version of the “inline” function. (www.mathworks.com)

### 3.4 Function “num2str”, “abs”, “plot”

The simple functions of “num2str” which convert number to string, “abs” which convert the value to be absolute or real only and “plot” which make data to graph or curve are also studied and apply in solution of nonlinear equations of MATLAB. Picture 5 shows example expression of these functions which can be studied by help of MATLAB.

![Picture 5. Study on MATLAB Functions](image-url)
4. IMPLEMENTATION RESULTS

4.1 M-Scripts for Three Methods

Input data are made as variable for any nonlinear equations, not only by getting answer, plotting graph process was followed at the end of program. The m-script for each method was carried out as in the following Table 1:

<table>
<thead>
<tr>
<th>% Bisection Method</th>
<th>%Secant Method</th>
<th>%Newton Raphson Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>s=input( 'Enter function with right hand side zero:' , 's');</td>
<td>a=input( 'Enter function:','s');</td>
<td>a=input( 'Enter the function in the form of variable x:','s');</td>
</tr>
<tr>
<td>f=inline(s);</td>
<td>f=inline(a);</td>
<td></td>
</tr>
<tr>
<td>a=input( 'Enter the first value of guess interval:');</td>
<td>x(1)=input( 'Enter first point of guess interval:');</td>
<td>x(1)=input( 'Enter Initial Guess:');</td>
</tr>
<tr>
<td>b=input( 'Enter the end value of guess interval:');</td>
<td>x(2)=input( 'Enter second point of guess interval:');</td>
<td>error=input( 'Enter allowed Error:');</td>
</tr>
<tr>
<td>eps=input( 'Enter the allowed error:');</td>
<td>n=input( 'Enter allowed Error in calculation:');</td>
<td></td>
</tr>
<tr>
<td>if f(b)*f(a)&gt;0</td>
<td>iteration=0;</td>
<td></td>
</tr>
</tbody>
</table>
| fprintf( 'The guess is incorrect! Enter new guesses
'); | for i=3:1000 |
| a=input( 'Enter the first value of guess interval:\n'); | x(i) = x(i-1) - (f(x(i-1)))*(x(i-1) - x(i-2))/(f(x(i-1)) - f(x(i-2))); |
| b=input( 'Enter the end value of guess interval:\n'); | iteration=iteration+1; |
| else end | if abs((x(i)-x(i-1))/x(i))*100<n |
| for i=1:100 | root=x(i); |
| c=(a+b)/2; | n_it=i; |
| if f(c)>0 | break |
| b=c; | end |
| else a=c; end end | if err(i)<error |
| a=1; b=2; p=c; | break |
| for i=1:100 | end |
| c=(a+b)/2; er(i)=f(c)-f(p); | f=inline(a); |
| if f(c)>0 | fprime= inline(d); |
| b=c; else a=c; end end | for i=1:100 |
| if er(i)<eps end | x(i+1)=x(i)-((f(x(i))/fprime(x(i)))); |
| str = [ 'The required root of the equation is: ' , num2str(xr), ' ' ]; | err(i)=abs((x(i+1)-x(i))/x(i)); |
| fprintf( 'The number of iteration is %d',iteration); | if err(i)<error |
| plot(x); | break |
| title( 'Plot of error' ) | end |
| xlabel( 'Number of iterations' ) | root=x(i); |
| ylabel( 'Error' ) | fprintf( 'Root of given equation is %f',root) |
| grid | plot(err); |
| | title( 'Plot of error' );
| | xlabel( 'Number of iterations' );
| | ylabel( 'Error' );
| | grid |
4.2 Simulation Results

The simulation results are shown in Picture 6 to Picture 8. We can study, analyze, compare and solve fastly and simply with our implemented MATLAB program for any nonlinear equations.

(a) m-file Running Results
(b) Error Plot

Picture 6. Bisection Method Results

(a) m-file Running Results
(b) Error Plot

Picture 7. Secant Method Results

(a) m-file Running Results
(b) Error Plot

Picture 8. Newton Raphson Method Results

(a) m-file Running Results
(b) Error Plot

Picture 9. Results Comparison for \((x^3 - x - 1 = 0)\)

5. DISCUSSION, CONCLUSION AND RECOMMENDATION

The implementation of Numerical Solutions for Nonlinear Equations are sucessfully done in this research paper using MATLAB. By this practicle approach, we can easily compare and analyse any nonlinear equations which are mostly represented in real-world Engineering process. How we can
compare and used each method of Bisection, Secant and Newton Raphson’s can be express as shown in Picture 9 for a given same nonlinear equation of $x^3 - x - 1 = 0$. We can proof that Bisection Method was simplest form. For Secant Method, it is faster than other numerical methods, except the Newton Raphson method and there is no need to find the derivative of the function as in Newton-Raphson method. If the function is not differentiable, Newton’s method cannot be applied. By this research paper, we can clearly analyze how nonlinear equations are easily be solved using various method by the help of MATLAB.

6. REFERENCE
2. Sanda Micula, “NONLINEAR EQUATIONS IN MATLAB”, Department of Mathematics and Computer Science, Babeș-Bolyai University, Cluș-Napoca, smicula@math.ubbcluj.ro.