American Journal of Computational and Applied Mathematics 2019, 9(1): 6-11
DOI: 10.5923/j.ajcam.20190901.02

Marching Method: A New Numerical Method for Finding Roots of Algebraic and Transcendental Equations

J. A. Ugboh, I. M. Esuabana*

Department of Mathematics, University of Calabar, Calabar, Cross River State, Nigeria

Abstract A new method for finding approximate roots (one at a time) of Nonlinear Algebraic or Transcendental Equations is developed similar in form with the bisection method. Given such an equation say, $f(x)=0$, with a root, say α on the interval $[a,b]=[a_0,b_0]$, or the succeeding interval $[a_n,b_n]$ is partitioned into ten equal subintervals. The function is then evaluated at the mid-point x_5 and compared with $f(a)=f(a_0)$ or with $f(a_n)$ for the *nth* iteration. If $f(a_0)f(x_5)>0$ or $f(a_n)f(x_5)>0$, the root is to the right of x_5 otherwise it is to its left. A search is then conducted to determine which of the remaining five subintervals contains the root. The midpoint of such an interval is the approximate root α_{n+1}, $n=0,1,2,3\cdots$. After the nth iteration, the approximate root α_{n+1} is the midpoint of the interval whose length is $2^{-n}10^{-n}(L_0)$; where $L_0=b_0-a_0$ and $[a_0,b_0]$ is the initial interval. A theorem that gurantees the convergence of the approximate roots to the actual root was stated and proved. Half the length of the succeeding interval $[a_n,b_n]$ serves as upper bound for the error in the approximation. An algorithm was developed and a program coded in JAVA was used to solve seven test problems. The results show that the method yielded approximations which are all correct to ten decimal places after ten iterations; and correct to fifteen decimal places after fifteen or fourteen iterations.

Keywords Nonlinear, Algebraic, Transcendental, Equations, Roots, Iterations, Approximations, Errors

1. Introduction

Given a non linear algebraic or transcendental equation of the form: $f(x)=0$; one often resort to numerical methods where analytic solution are not easy to come by. A good number of such methods are available and one of the simplest is the bisection method. However, the rate of convergence for the bisection method is very slow and so not suitable in a situation where high precision is required. It is therefore necessary to search for a method that is as simple but yeild highly accurate results in a few number of iterations. In this work, we present a method that is simple with high precision in a few number of iterations, machine implementable and guarantee convergence to desired root.

The marching method involves partitioning the given interval $[a,b]=[a_0,b_0]$, or the succeeding interval $[a_n,b_n]$ into ten subintervals of equal size with $a_0=x_0<x_1<x_2<\cdots<x_9<x_{10}=b_0$ or $a_n=x_0<x_1<x_2<\cdots<x_9<x_{10}=b_n$; one of which must contain the root or one of the end points must be the root. For succeeding intervals $I_n=[a_n,b_n]$ at the n^{th} iteration, the lower and upper end points becomes respectively a_n and b_n. A search is conducted from the midpoint (x_5) of the interval either to the left or right to determine the subinterval which contains the root. Given $f(x)=0$ on $[a,b]$ with $f(a)f(b)<0$, and midpoint $x_5=\frac{a+b}{2}=\frac{a_0+b_0}{2}$ in the first iteration; (or $x_5=\frac{a_n+b_n}{2}$ for the *nth* iteration); the following decision rule applies: if $f(x_5)=0$, the root is x_5 and we stop; otherwise, we search to the right of the midpoint if $f(x_5)$ has same sign as $f(a_0)$ or $f(a_n)$; and to the left otherwise. In either case, a maximum of four searches is required. It is worthy of note that the search might just be once!

Once the subinterval, $[x_k,x_{k+1}]$ or $[x_{k-1},x_k]$ containing the root is found, the approximate root α_{n+1}, $n=0,1,2,3...$ is taken as the midpoint of that subinterval leading to a succeeding interval $I_1=[a_1,b_1]$ of length $L_1=\frac{b_0-a_0}{20}$ or

* Corresponding author:
esuabanaita@gmail.com (I. M. Esuabana)
Published online at http://journal.sapub.org/ajcam

$I_n = [a_n, b_n]$ of length $L_n = \frac{b_n - a_n}{20}$. Thus, in the first iteration, an upper bound for the error in approximation is $\frac{1}{2}L_1 = \frac{b_0 - a_0}{40}$.

After *n* iterations, the upper bound for the error is expected to be half of the length $[a_n, b_n]$, that is, the error bound after n iterations is $\frac{1}{2}\left(\frac{b_n - a_n}{20}\right) = 2^{-(n+1)}.10^{-n}.L_0$; where L_0 is the length of the starting interval. It is simple, machine implementable (can be coded in many high level languages), converges very fast to desired root and conserve space. However, it is a bit more difficult when compared to the bisection method due to the search. In any case, with high precision computers, the very high rate of convergence compensates for the extra work. An algorithm and a model software program in JAVA will be developed for the method. Seven test problems have been earmark for machine solution.

2. Preliminaries

A number of methods have been developed to solve nonlinear algebraic and/or transcendental equations of the form $f(x) = 0$. The bisection method, secant method, Newton-Raphson method, regular falsi method and the general iteration technique are among the most commonly used ([1], [5], [7]). Slow convergence is one of the major drawbacks of most of these methods (except for Newton-Raphson). Newton- Raphson method though converges fast, has the flaw of requiring existence of first derivative and evaluation of two functional values per step ([3], [5], [6]]. Nonlinear equations have become indispensible tools among scientist in various disciplines hence, faster methods of solutions are in high demand. With the advent of high speed/precision computers and sophistication in proggamming, new methods of solutions are emerging. The efficiency, consistency, simplicity and economy of such a method are the measuring indexes ([6], [8], [4]).

Lemma 2.1 *Let* $f : X \to Y$ *be any given real-valued function continuous on the closed and bounded interval* $[a,b]$ *such that* $f(a).f(b) < 0$*; then there exists at least* $\alpha \in (a,b)$ *such that* $f(\alpha) = 0$. *[3], [2], [6]*

The above theorem, a consequence of intermediate value theorem, enables us determine interval where a root of the equation $f(x) = 0$ exists. However, if the function is highly oscillatory, there is a possibility of several roots in that interval. In other to avert this, the end points are taken close enough or a rough graph of $y = f(x)$ may be sketched to determine a rough point where the curve crosses the x axis.

3. Methodology

Suppose we desire to find the root of the algebraic or transcendental equation say, $f(x) = 0$ on the closed interval $[a,b]$ where b and a are close enough to avoid the existence of more than one root of f. The marching method involves the following steps.

Step (i) Evaluate $f(a), f(b)$ and confirm that $f(a)f(b) < 0$. Set $a = a_0$ and $b = b_0$ and error level $\varepsilon = 10^{-p}$, $p \in \boldsymbol{N}$.

Step (ii) Partition the interval $[a,b] = [a_0, b_0]$ (or the succeeding interval $[a_n, b_n]$) into ten equal parts with $a_0 = x_0 < x_1 < x_2 < \cdots < x_9 < x_{10} = b_0$ or $a_n = x_0 < x_1 < x_2 < \cdots < x_9 < x_{10} = b_n$.

In the first iteration we take $x_k = a_0 + k\frac{(b_0 - a_0)}{10}$, where $k = 0, 1, 2 \cdots 10$.

In general, we take $x_k = a_n + k\frac{(b_n - a_n)}{10}$ for the n^{th} iteration, where $k = 0, 1, 2 \cdots 10$ and $n = 0, 1, 2, 3, \cdots$

Step (iii): Conducting the search. Evaluate $f(x_5)$, if it is equal to zero you have the root,or $|b - a| \leq \varepsilon$, stop, desired root is x_5; otherwise if it has the same sign as $f(x_0) = f(a_n)$ then the search is to the right of x_5, and if of opposite sign, the search is to the left of x_5. The search is conducted as follows:

(a) Suppose $f(x_5)$ and $f(a)$ have the same sign, we evaluate f at $x_6, x_7, x_8,$ or x_9 and for any $k = 6, 7, 8$ or 9 which $f(x_k) = 0$, x_k is the root, otherwise,

if $f(x_k)$ has opposite sign with $f(x_5)$; we have found the interval containing the root to be:

$[x_{k-1}, x_k]$, if not, the last interval which is $[x_9, x_{10}]$ must contain the root. The approximate root is then:

$$\alpha = \frac{x_{k-1} + x_k}{2}; k = 6,7,8,9 \text{ or } 10;\ n = 0,1,2,3,\cdots \quad (1)$$

End points of new interval: Next evaluate $f(\alpha_{n+1})$; if $f(\alpha_{n+1}) = 0$ or $|\alpha_{n+1} - \alpha_n| \leq \varepsilon$; $n = 0, 1, 2, 3 \cdots$ or $n = N$ maximum number of iterations, the root is α_{n+1}, stop, else, if $f(\alpha_{n+1})$ has same sign with $f(x_{k-1})$ then $a_{n+1} = \alpha_{n+1}$, and $b_{n+1} = x_k$, otherwise, $b_{n+1} = \alpha_{n+1}$, and $a_{n+1} = x_{k-1}$. New or succeeding interval $I_{n+1} = [a_{n+1}, b_{n+1}]$; go to step (ii).

(b) On the other hand, if $f(x_5)$ and $f(a_n)$ have opposite signs, we search to the left of x_5 by evaluating f

at x_1, x_2, x_3 or x_4 . For $k = 1, 2, 3,$ or 4 ; should $f(x_k) = 0$, we have the root as x_k, otherwise if $f(x_k)$ has same sign as $f(a_n)$, then the interval $[x_k, x_{k+1}]$ contains the root. Where none of these function values has same sign as $f(x_0) = f(a_n)$, then the root must be in the first interval $[x_0, x_1]$. The approximate root is then:

$$\alpha = \frac{(x_k + x_{k+1})}{2}; \text{ where } k = 0,1,2,3, \text{ or } 4; \quad \text{n} = 0,1,2,3,\cdots \tag{2}$$

End points of new interval: Next evaluate $f(\alpha_{n+1})$; if $f(\alpha_{n+1}) = 0$ or $|\alpha_{n+1} - \alpha_n| \le \varepsilon$; n = 0, 1, 2, 3 $\cdots$ or n = N maximum number of iterations, the root is α_{n+1}, stop, else, if $f(\alpha_{n+1})$ has same sign with $f(x_{k-1})$ then $a_{n+1} = \alpha_{n+1}$, and $b_{n+1} = x_k$, otherwise, $b_{n+1} = \alpha_{n+1}$, and $a_{n+1} = x_{k-1}$. New or succeeding interval $I_{n+1} = [a_{n+1}, b_{n+1}]$; go to step (ii).

(c) In summary, at any new iteration level we have the points of partition: $x_k = a + \frac{k(b-a)}{10}$; where $k = 0,1,2\cdots 10$. The midpoint of the interval $[a,b]$ is always x_5.

The **approximate root** is:

$$\alpha_{n+1} = \begin{cases} \frac{(x_k + x_{k+1})}{2} & \text{if search is to the left of } x_5,\ 0 \le k \le 4 \\ \frac{(x_{k-1} + x_k)}{2} & \text{if search is to the left of } x_5;\ 6 \le k \le 10 \end{cases} \tag{3}$$

where n = 0, 1, 2, 3,$\cdots$

New Interval: Next evaluate $f(\alpha_{n+1})$ and if zero you have the root, otherwise, If the search was to the left of x_5 and $f(\alpha_{n+1})$ has same sign with $f(x_k)$ then $a_{n+1} = \alpha_{n+1}$, and $b_{n+1} = x_{k+1}$, else, $b_{n+1} = \alpha_{n+1}$, and $a_{n+1} = x_k$; go to step (ii).

If the search was to the right of x_5 and $f(\alpha_{n+1})$ has same sign with $f(x_k)$ then $a_{n+1} = \alpha_{n+1}$, and $b_{n+1} = x_k$, otherwise, $b_{n+1} = \alpha_{n+1}$, and $a_{n+1} = x_{k-1}$; go to step (ii).

The processes in steps (ii) to (iii) and (a) or (b) is continued until we have: $f(x) = 0$ or $|b - a| < \varepsilon$ or after the desired number of iterations; where ε is the tolerable error.

In all, seven test problems will be solved comprising of four polynomials and three transcendentals.

Theorem 3.1 *Let* $f(x)$ *be any function continuous at all points of the interval* $[a,b]$ *with* $f(a)f(b) < 0$, *then the iteration* $\alpha_n = \frac{(x_k + x_{k+1})_n}{2}$ *or* $\alpha_n = \frac{(x_{k-1} + x_k)_n}{2}$ *conerges to the root* $\alpha \in (a,b)$, *where* $n = 1,2,3,\cdots$

Proof. f is continuous on the bounded interval $[a,b]$ so with $f(a)$ and $f(b)$ having opposite signs, we have a root $\alpha \in (a,b)$ by theorem 2.1 . For the first iteration, the interval containing the root is of length $\frac{L}{2(10)} = L_1$; where L is the initial or starting length; (i.e. $L = b - a$). To obtain the next interval containing the root, the above interval is partioned into ten and the mid point of the new subinterval containing the root is used as the next approximation. Thus, the length of the second interval is $\frac{1}{2(10)}$ of $\frac{L}{2(10)}$ giving the length $L_2 = (\frac{1}{2(10)})(\frac{L}{2(10)}) = \frac{L}{2^2(10^2)}$. Continuing in this manner, at the n^{th} iteration, the length of the interval containing the root will be $L_n = \frac{L}{2^n(10^n)}$. As $n \to \infty, L_n \to 0$ meaning that $|\alpha_{n+1} - \alpha_n| < L_n \to 0$. Since $L_n \to 0$ and the root $\alpha \in (a,b)$ with $b - a < L_n \to 0$; α_n must converge to the root $\alpha \in (a,b)$.

3.1. Algorithm

```
Function sameSign(x, y) {
   if( (x < 0 AND y < 0 ) OR ( x > 0 AND y > 0) )
         return true;
   else return false;
}
Note: x_k = a_n + k(b_n − a_n)/10

Function Search(lowerLimit, upperLimit) {
   if( sameSign( f(x_5), f(a_n)) )
     let direction = "right";
   else let direction = "left";

   for i = lowerLimit to upperLimit AND
interval_not_found:
         if(direction is "right"):
           if(Not sameSign(f(x_5), f(x_i)):
                α_{n+1} = (x_{i−1} + x_i)/2
         if(sameSign(f(α_{n+1}), f(x_{i−1})):
         //update a, and b
         a_{n+1} = α_{n+1},   b_{n+1} = x_i
         else:
         a_{n+1} = x_{i−1}, b_{n+1} = α_{n+1}
         else if(direction is "left"):
         if(Note sameSign(f(x_0), f(x_i)):
         α_{n+1} = (x_{i−1} + x_i)/2
         if(sameSign(f(α_{n+1}), f(x_i)):
            a_{n+1} = x_{i−1}, b_{n+1} = α_{n+1}
         else:
```

$a_{n+1} = \alpha_{n+1}, \quad b_{n+1} = x_i$
if(interval_was_not_found):
if(direction is "right"):
$\alpha_{n+1} = \frac{(x_9+x_{10})}{2}$
if(sameSign(f(x_9), f(α_{n+1})):
$a_{n+1} = \alpha_{n+1}, \quad b_{n+1} = x_9$
else:
$a_{n+1} = x_9, \; b_{n+1} = \alpha_{n+1}$
else if(direction is "left"):
$\alpha_{n+1} = \frac{(x_1+x_0)}{2}$
if(sameSign(f(x_0), f(x_1)):
$a_{n+1} = \alpha_{n+1}, \quad b_{n+1} = x_1$
else:
$a_{n+1} = x_0, \; b_{n+1} = \alpha_{n+1}$
return α_{n+1};
}

CONDITIONF FOR ITERATION:
$f(\alpha_{n+1}) \neq 0$ AND $|\alpha_{n+1} - \alpha_n| > \varepsilon$ OR $n < N$

3.2. Comparative Analysis of Steps/Error

The method is about a modification of the well known bisection method. In each iteration step, the marching method partition the given interval $[a,b] = [a_n,b_n]$ into ten sub intervals, evaluate functioal value of the mid point and at most four others, [but could be only one] (to its left or right) which it compare with that of the lower limit $f(a_n)$. With the excution of each step the length of the succeeding interval is reduced to $\frac{1}{20}$ of the previous interval $[a_n,b_n]$. At the end of each step condition for succeeding interval which is excatly as in the bisection method is implimented. On the other hand, in the bisection method, the interval is halfed per each iterative step, mid point functional value compared with one of the end points $f(a_n)$ or $f(b_n)$ in each iteration step. At each of the step succeeding interval conditions are also implimented.

Four steps in the bisection method can reduce the length of the interval $[a_n,b_n]$. by a factor of $\frac{1}{2^4}$.

To surpass the accuracy level of the marching method, five iterations of the bisection method will be required. From the search algorithm for the marching method, with one, two or three searches per step, the matching method will involve less work and post more accurate results, [i,e, 75 percent of the time]. After *n* iterations the length of the succeeding interval for the bisection method is: $2^{-n} * (b_0 - a_0)$; while that of the marching method is: $2^{-n} * 10^{-n} * (b_0 - a_0)$.

If that of the bisection is further divided by 8, we still have:

$$\frac{1}{8} * \left[2^{-n}(b_0 - a_0)\right] > 2^{-n} * 10^{-n}(b_0 - a_0).$$

Error: Like in other brackecting methods, the maximum absolute error in the marching method is half the length of the *nth* interval. Thus after n iterations, the maximum absolute error in the method is:

$$\varepsilon_a = |\alpha_{n+1} - \alpha| \leq \frac{1}{2}\left[2^{-n} * 10^{-n}(b_0 - a_0)\right].$$

The convergence rate is linear. If we use the maximum or upper bound for the error at the n^{th} iteration and denote it by ε_n then we have (Chapra and Canale, 2006, Mathews, 1987)

$$\frac{\varepsilon_{n+1}}{\varepsilon_n} = \frac{\frac{1}{2}[2^{-n-1} * 10^{-n-1}(b_0 - a_0)]}{\frac{1}{2}[2^{-n} * 10^{-n}(b_0 - a_0)]} = \frac{1}{20}$$

$$\Rightarrow \frac{\varepsilon_{n+1}}{\varepsilon_n} = K \Rightarrow \varepsilon_{n+1} = K * \varepsilon_n,$$

where $K = \frac{1}{20}$ is a constant, hence the convergence is linear. For the bisection method the constant $K = \frac{1}{2}$. It follows that the bisection method redues the bracketing interval by half while the marching method reduces same interval by a factor of 20.

4. Results

The following test problems were solved with the marching method using a program coded in JAVA.

Problem 1: Use the matching method to find the root of $f(x) = 3x^2 + x - 10$ in $[1,2]$; using the first 15 iterations. Out put the successive intervals and the approximate root in each interval.

Table 1. Results showing approximate roots of $f(x) = 3x^2 + x - 10$ in $[1,2]$; obtained from a JAVA coding

j	*a*	*b*	*x*
1	1.6500000000000001	1.895	1.6500000000000001
2	1.6622500000000002	1.6855250000000002	1.6622500000000002
3	1.6657412500000004	1.6696980000000003	1.6657412500000004
4	1.6665326000000005	1.6667304375000005	1.6667304375000005
5	1.6666611943750005	1.6667096645625006	1.6666611943750005
6	1.6666660413937504	1.6666684649031254	1.6666684649031254
7	1.6666666472710943	1.6666671925607035	1.6666666472710943
8	1.6666666472710943	1.6666666745355747	1.6666666745355747
9	1.6666666663562306	1.6666666677194546	1.6666666677194546
10	1.6666666666288754	1.6666666666970367	1.6666666666970367
11	1.666666666666364	1.6666666666847676	1.666666666666364
12	1.666666666666364	1.6666666666672842	1.6666666666672842
13	1.66666666666664	1.666666666666686	1.666666666666686
14	1.6666666666666652	1.6666666666666776	1.6666666666666652
15	1.6666666666666665	1.666666666666667	1.666666666666667

Root = 1.6666666666666667. Observe that the root is correct to 10 decimal places at the 10^{th} iteration and to 15

decimal places at the 15th. The exact root is $2\frac{2}{3}$.

Problem 2: Use the matching method to find the root of $f(x) = x^2 + 3x + 2$ in $[-3, -1.5]$; using the first 14 iterations. Output the successive intervals and the approximate root in each interval.

Table 2. Results showing approximate roots of $f(x) = x^2 + 3x + 2$ in $[-3, -1.5]$ obtained from a JAVA coding

j	*a*	*b*	*x*
1	–2.025	–1.6575	–2.025
2	–2.0066249999999997	–1.9717124999999998	–2.0066249999999997
3	–2.0013881249999996	–1.9954529999999997	–2.0013881249999996
4	–2.0002010999999995	–1.9999043437499995	–1.9999043437499995
5	–2.0000082084374995	–1.9999355031562496	–2.0000082084374995
6	–2.0000009379093746	–1.999997302645312	–1.999997302645312
7	–2.000000029093359	–1.9999992111589449	–2.000000029093359
8	–2.000000029093359	–1.9999999881966382	–1.9999999881966382
9	–2.000000000465654	–1.9999999984208183	–1.9999999984208183
10	–2.000000000056687	–1.9999999999544453	–1.9999999999544453
11	–2.000000000000454	–1.9999999999728488	–2.000000000000454
12	–2.000000000000454	–1.9999999999990736	–1.9999999999990736
13	–2.00000000000004	–1.999999999999971	–1.999999999999971
14	–2.0000000000000018	–1.9999999999999833	–2.0000000000000018
15	–2.000000000000001	–1.9999999999999991	–2.000000000000001

Root = -2.000000000000001. The exact root in this interval is - 2. Observe that the root is correct to 10 decimal places with the 10th iteration and to 15 decimal places at the 15th.

Problem 3: By using the marching method, obtain an approximate root of $f(x) = 8x^2 + 3x - 5$ in $[0,1]$ using the first 15 iterations. Out put the successive intervals and the approximate root in each interval.

Table 3. Results showing approximate roots of $f(x) = 8x^2 + 3x - 5$ in $[0,1]$; obtained from a JAVA coding

j	*a*	*b*	*x*
1	0.6000000000000001	0.6500000000000001	0.6500000000000001
2	0.6225	0.6252500000000001	0.6225
3	0.6237375000000001	0.6238887500000001	0.6237375000000001
4	0.6238055625000001	0.62381388125	0.6238055625000001
5	0.6238093059375001	0.6238097634687501	0.6238093059375001
6	0.6238095118265625	0.6238095369907812	0.6238095118265625
7	0.623809523150461	0.623809524534493	0.623809523150461
8	0.6238095237732755	0.6238095238493973	0.6238095237732755
9	0.6238095238075303	0.623809523811717	0.6238095238075303
10	0.6238095238094143	0.6238095238096446	0.6238095238094143
11	0.623809523809518	0.6238095238095307	0.623809523809518
12	0.6238095238095238	0.6238095238095245	0.6238095238095238
13	0.623809523809524	0.623809523809524	0.623809523809524

Root = 0.623809523809524. From the 13th iteration the root is correct to 15 decimal places! Excat root unknown but is in $[0,1]$.

Problem 4: By using the marching method obtain an approximate root for $f(x) = x\sin x + \cos x$ in $[0, \pi]$; [x in radians] using the first 15 iterations. Output end points of successive interval and the approximate roots in each interval.

Table 4. Results showing approximate roots of $f(x) = x\sin(x) + \cos(x)$ on $[0, \pi]$; obtained from coding in JAVA

j	*a*	*b*	*x*
1	2.7475000000000005	3.0615	2.7475000000000005
2	2.7946000000000004	2.84798	2.7946000000000004
3	2.7972690000000004	2.8023401000000003	2.7972690000000004
4	2.7982832200000005	2.7985367750000005	2.7985367750000005
5	2.7982958977500005	2.7983199854750005	2.7982958977500005
6	2.79831878108875	2.7983199854750005	2.79831878108875
7	2.7983199252556883	2.7983199854750005	2.7983199252556883
8	2.798319982464035	2.7983199854750005	2.798319982464035
9	2.7983199853244525	2.7983199854750005	2.7983199853244525
10	2.798319985467473	2.7983199854750005	2.798319985467473
11	2.798319985474624	2.7983199854750005	2.798319985474624
12	2.798319985474982	2.7983199854750005	2.798319985474982
13	2.7983199854749996	2.7983199854750005	2.7983199854749996
14	2.7983199854750005	2.7983199854750005	2.7983199854750005

Root = 2.7983199854750005 correct to 15 decimal places after 14 iterations. Excat root not known.

Problem 5: Obtain an approximate root of $f(x) = x^6 + 2x^4 - x^3 - 1$ on $[0,1]$; with the marching method using the first 15 iterations. Out put the end points of successive intervals and the approximate root in each.

Table 5. Results showing approximate roots of $f(x) = x^6 + 2x^4 - x^3 - 1$ on $[0,1]$; obtained from a JAVA coding

j	*a*	*b*	*x*
1	0.8500000000000001	0.985	0.8500000000000001
2	0.8770000000000001	0.88375	0.88375
3	0.8827375000000001	0.8836487500000001	0.8827375000000001
4	0.8827830625	0.8828696312500001	0.8827830625
5	0.8827873909375	0.88279561496875	0.8827873909375
6	0.8827886245421875	0.8827900226275001	0.8827886245421875
7	0.8827886944464531	0.8827888272645579	0.8827886944464531
8	0.8827887409327898	0.882788775465497	0.8827887409327898
9	0.8827887461126959	0.8827887519832561	0.8827887461126959
10	0.8827887461126959	0.8827887464062238	0.8827887464062238
11	0.8827887463636623	0.8827887463915475	0.8827887463915475
12	0.8827887463803934	0.8827887463817876	0.8827887463817876
13	0.8827887463815854	0.8827887463817179	0.8827887463817179
14	0.8827887463816987	0.8827887463817112	0.8827887463817112
15	0.8827887463817081	0.8827887463817106	0.8827887463817081

Root = 0.8827887463817081 which is correct to 15 decimal places at the 15th iteration. Exact root not known.

Problem 6: Determine the approximate root of $f(x) = \tan(\pi - x) - x$ on the interval $[(1.6, 3]$ where x is in radians with the first 16 iterations.

Table 6. Results showing approximate roots of $f(x) = \tan(\pi - x) - x$ on $[(1.6, 3]$; obtained from a JAVA coding

j	*a*	*b*	*x*
1	2.02	2.09	2.09
2	2.027	2.0305	2.0305
3	2.028925	2.02987	2.028925
4	2.02906675	2.0292274	2.02906675
5	2.0290908475	2.029118158	2.0290908475
6	2.029097675125	2.0291038199875002	2.029097675125
7	2.02909828961125	2.029098596854375	2.029098596854375
8	2.029098581492219	2.029098596854375	2.029098581492219
9	2.0290985868689737	2.0290985908631343	2.0290985868689737
10	2.029098587068682	2.0290985874481273	2.029098587068682
11	2.0290985870876543	2.0290985871237015	2.0290985870876543
12	2.0290985870894565	2.029098587092881	2.0290985870894565
13	2.02909858709271	2.029098587092881	2.02909858709271
14	2.0290985870928724	2.029098587092881	2.0290985870928724
15	2.0290985870928804	2.029098587092881	2.0290985870928804
16	2.029098587092881	2.029098587092881	2.029098587092881

Root = 2.029098587092881 after 16 iterations. Exact root not known but in $[1.6, 3]$.

Problem 7: Determine the approximate root of $f(x) = xe^x - 2x - 1$ on the interval $[1, 2]$ correct to 15 decimal places or using the first 15 iterations of the marching method.

Table 7. Results showing approximate roots of $f(x) = xe^x - 2x - 1$ on $[1, 2]$; obtained from a JAVA coding

j	*a*	*b*	*x*
1	1.05	1.145	1.05
2	1.07375	1.095125	1.07375
3	1.07481875	1.0768493749999999	1.07481875
4	1.0750218125	1.07512334375	1.07512334375
5	1.0750268890624999	1.0750365345312498	1.0750268890624999
6	1.0750360522578122	1.0750365345312498	1.0750360522578122
7	1.0750365104175779	1.0750365345312498	1.0750365104175779
8	1.0750365333255663	1.0750365345312498	1.0750365333255663
9	1.0750365344709656	1.0750365345312498	1.0750365344709656
10	1.0750365345282356	1.0750365345312498	1.0750365345282356
11	1.075036534531099	1.0750365345312498	1.075036534531099
12	1.0750365345312423	1.0750365345312498	1.0750365345312423
13	1.0750365345312494	1.0750365345312498	1.0750365345312494
14	1.0750365345312498	1.0750365345312498	1.0750365345312498

Root = 1.0750365345312498 correct to 15 decimal places after 14 iterations. Exact root in $[1, 2]$ is not known.

5. Conclusions

The matching method is designed to determine a root of nonlinear equation at a time. Given that a root exists on the interval $[a, b]$, i.e. $f(a)f(b) < 0$, the interval is partitioned into ten equal parts with $a = x_0$ and $b = x_{10}$. Either one of the end points of the subintervals is the root or one of the subintervals contains the root. A search is conducted to determine this interval. This search might involve evaluation of at most four functional values or might just be one or two per step! The approximate root is taken as the mid-point of the new interval and if it is not the root, the process is repeated until the root is found or the stopping criterion is met. An algorithm was developed and a program coded in JAVA was used to solve seven test problems involving polynomials and transcendental functions. Results are tabulated the tables give the intervals and the approximate roots for each step. Results obtained show a very high precision of about ten decimal places in ten iterations which is equivalent to one decimal per step. For few decimal places or iterations, computation with calculator can be used.

The Marching Method ensures that the approximation converges to the desired root, highly accurate, economical in terms of space and time, easy to program and can be used to solve nonlinear algebraic or transcendental equations. The program coding can be done in any other high level language that is capable of searching.

REFERENCES

[1] Ele, B. I. and Ugboh, J. A.. *Numerical Computations for Science and Engineering with C++ Programming Basics*. Radiant Ventures Press, Calabar – Nigeria, 2017.

[2] Gaughan, E. G., *Introduction to Analysis*. Brooks/Cole Publishing company, California, 1993.

[3] Grewal, B. S., *Numerical Methods in Engineering and Science*. Khanna Publishers, Delhi, 1997.

[4] Haeussler, E. F., Paul, R. S., and Wood, R. J., *Introductory Mathematical Analysis for Business, Economics, and Life and Social Sciences*. Printice-Hall, Boston, 2011.

[5] Mathews, J. H., Numerical Methods for Computer Science, Engineering and Mathematics, Printice-Hall, New Jersey, 1987.

[6] Olayi, G. A., *Introduction to Numerical Analysis*. ABU Press, Zaria – Nigeria, 2000.

[7] Riley, K. F., P., H. M., and Bence, S. J., *Mathematical Methods for Physics and Engineering*, Cambridge University Press, United Kingdom, 1998.

[8] Won, Y. Y., *Applied Numerical Methods Using MATLAB*, John Wiley Interscience, New York, 2005.