ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 5, Special Issue 13, October 2016**

# Zeroing on zeros of polynomials

Nibir Pal [1], Susobhan Ghosh [2]

U.G. Student, Department of Computer Science & Engineering, Kalyani Government Engineering College, Kalyani, West Bengal, India[1]

U.G. Student, Department of Computer Science & Engineering, Kalyani Government Engineering College, Kalyani, West Bengal, India [2]

**ABSTRACT**: There are a lot of root finding algorithms to find zeroes of polynomials. 2nd degree and 3rd degree polynomials can be solved very easily, but higher degree complex polynomials of nth degree, n>3, one requires efficient algorithms to find the zeroes. This paper proposes an approach for finding zeroes of polynomials using Graeffe's Algorithm, Bairstow's algorithm and Bisection method. Graeffe's algorithm focuses on squaring the roots of the original polynomial, and then effectively approximating the polynomial using its coefficients. Bairstow algorithm on the other hand finds a quadratic factor of the polynomial, and thus finds 2 roots in each instance. The roots found are then narrowed down using bisection method.
The efficiency of the method is tested and the results are encouraging.

**KEYWORDS**: polynomial roots, root finding algorithm, Bairstow's algorithm, Graeffe's algorithm, multiple roots, complex roots, Ruffini's method, Bisection.

## I. INTRODUCTION

Polynomial zeroes have a very long history [1], and a multitude of work have been proposed [2]. Finding solutions to equations has many great direct implications in modern era like in solving complex equations in physics, finger print scanning etc., and finding the roots of polynomials is one which is sought after. Lower degree polynomials can easily be solved, but higher ones need special algorithms and approach to narrow the roots down to a small interval which can be as small as $10^{-6}$.

Because of the complexity in finding roots of higher degree polynomials, one has options of using a multitude of algorithms. Newton-Ralphson's [3] method is one which deals with the derivate of the polynomial and a point, finding the tangent and then finding its intersection point with x-axis. Such a method has limitations like unable to compute roots of changing concavity/convexity and can to and fro in a graph. Bisection method [4]alone takes a lot of time to compute and reach to the answer as it performs binary search operation, and also finding the end values having opposite signs would take time. The regula-falsi method [5] has tendency of failing to converge under some naïve circumstances. Brent's method [6] on the other hand is a very handy and fast method, using the combination of bisection method, the secant method [7] and inverse quadratic interpolation [8], and choosing which is best for a situation. Finding roots one by one using Newton's method [9], Halley's method [10], Laguerre's method [11] etc. and consecutively dividing the polynomial using Horner's method following Ruffini's rule is also a way to solve polynomials.

Thus one can use a couple of algorithms together and fine tune the approximate answers using Bisection and finding the inflection points. The first algorithm which has been discussed in this paper, is a very fast algorithm, namely the Graeffe's Algorithm [12]. It results can be approximate, which can be made more accurate by using Bisection over the yielded results, which has also been discussed. The remainder of the paper consists of multiple same roots detection which can be done by finding the inflection point and to the degree till which the derivative is 0. The thus found roots can be shown as factors in the form (x-root), which can then be factorized from the main polynomial. This is done using Ruffini's Method [13], which has been discussed next. The resulting polynomial can then be solved by Bairstow's Algorithm [14], which has been discussed at the end..

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 13, October 2016

## II. GRAEFFE-BISECTION- DERIVATIVE-RUFFINI-BAIRSTOW (GBDRB) ALGORITHM

The algorithm uses a combination of four different algorithms and one derivate check starting with Graeffe's Algorithm - to approximate real roots, followed by bisection method - to increase accuracy of roots, a derivate check to determine inflection points and thus the repeated roots, Ruffini's method - to find the factorized polynomial after finding the real roots, and Bairstow's Algorithm – to ultimately find all the remaining imaginary roots. Thus, combining these methods, one can use the GBDRB algorithm as described here and find all the roots of a polynomial.

Given a polynomial $f(x) = x^n + a_1x^{n-1} + \cdots + a_n$
We need to find the solution to f(x)=0.
We can also represent the polynomial as

$$f(x) = (x - x_1)(x - x_2)(x - x_3) \dots \dots \dots \dots (x - x_n)$$

### 1. *GRAEFFE'S ALGORITHM*

Graeffe's Algorithm consecutively squares the roots of the polynomial to make it such that, one root is negligible in comparison to the other, and then uses Vieta's relations to approximate the roots.

As per Graeffe's algorithm, one finds

$$f(-x) = (-1)^n(x + x_1)(x + x_2)(x + x_3) \dots \dots \dots \dots (x + x_n)$$

The roots have changed their signs. We define,

$$p(x^2) = f(x) * f(-x) = (-1)^n(x^2 - x_1^2)(x^2 - x_2^2)(x^2 - x_3^2) \dots \dots \dots \dots (x^2 - x_n^2).$$

So we can write, $p(x) = (-1)^n(x - x_1^2)(x - x_2^2)(x - x_3^2) \dots \dots \dots \dots (x - x_n^2)$

$$= x^n + b_1x^{n-1} + \cdots + b_n$$

Comparing it to $(x) = x^n + a_1x^{n-1} + \cdots + a_n$ , the coefficients are related by

$$b_k = (-1)^k a_k^2 + 2\sum_{j=0}^{k-1}(-1)^j a_j a_{2k-j}, \text{ with } a_0=b_0=1$$

If we repeat the root squaring process k times, we get,

$$p^k(z) = z^n + a_1^k z^{n-1} + \cdots + a_n^k$$

This polynomial of degree n has roots $z_1 = x_1^{2^k}, z_2 = x_2^{2^k}, z_3 = x_3^{2^k}, \dots, z_n = x_n^{2^k}$

The roots and the coefficients are then related using Vieta's relations, $a_0=1$

$$a_1^k = -(z_1 + z_2 + \cdots + z_n)$$

$$a_2^k = (z_1z_2 + z_2z_3 + \cdots + z_{n-1}z_n)$$

.
.
.

$$a_n^k = (-1)^n(z_1z_2z_3 \dots z_{n-1}z_n)$$

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 13, October 2016

Given the number of iterations by which we have repeated the root squaring process (in our case 6 times, which means a factor of 64 as power), we can assume the roots have been separated sufficiently such that we can ignore the other remaining terms. Using this assumption, we can approximate the roots of the equation as per:

$$a_1^k = -z_1,\ a_2^k = (z_1 z_2) \text{ and so on…..}$$

Which implies:

$$z_1 = x_1^{2^k} = -a_1^k,\ z_2 = x_2^{2^k} = \frac{-a_2^k}{a_1^k},\ \ldots\ldots\ldots\ldots,\ z_n = x_n^{2^k} = \frac{-a_n^k}{a_{n-1}^k}$$

Now, by using logarithms or simple solving of $x^n = k$ will yield approximate values of $x_1$, $x_2$, $x_3$… $x_n$. One can use a power of 64 or higher to yield results of precision of less than $10^{-6}$. One has to note that, this holds good only for real roots. For complex roots, one has to successively compute square roots of $q^{m-1}(x)$ which is time consuming (for checking which root holds) and bisection doesn't support complex numbers, so gaining accuracy at each step is not feasible. The approximate real roots from this algorithm can be put into Bisection method to gain accuracy, whereas the complex roots are tough to handle.

**Algorithm – Graeffe**

**Input –**1. Degree of polynomial n
2. Array A [ ] containing the coefficients of the polynomial

**Output** - Array B [ ] containing coefficients of the polynomial whose roots are raised to the power 64 of the original polynomial.

```
1: for i=1 to 6
2:     for j=1 to n
3:         Update B[j]= A[j] * A[j]
4:         Assign l=2, which changes its sign alternately for each term
5:         Assign k=1
6:         while ( j – k ) >= 0 and ( j + k ) <= n
7:             Update l = l *(-1)
8:             Update k = k+1
9:             Update B[j] = B[j] + ( l * A[j-k] * A[j+k] )
```

The complexity of the Graeffe's Algorithm as deployed comes out to be O ($n^2$)

## 2. *BISECTION METHOD*

Bisection method is extremely useful in finding roots upto the desired accuracy within a fixed upper and lower bound. Given an upper bound **a**and a lower bound **b**, within which only one root of the polynomial is situated, we define the mid-point between **a**and **b**as

$$\text{mid} = (a+b)/2$$

This process effectively divides the region between **a** and **b** in two halves. Each region is checked for the presence of roots by the condition

$$f(\text{mid})*f(a) < 0$$

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 13, October 2016

since it is known that the value of a function f(x) changes sign at the root.

The region where the root lies is identified and the value of **a**and **b**are changed to be the upper and lower bound respectively of this region. The process is repeated till the desired accuracy is reached.

**Algorithm – Bisection**

```
1: for i =1 to 20
2:         Assign mid= (a+b)/2;
3:         if f(mid)*f(a) < 0
4:                 Update b=mid
5:                 end
6:         else
7:                 Update a=mid
8:                 end
9: return mid
```

### *3. DERIVATIVE CHECK FOR INFLECTION POINTS*

The derivative method implemented in this review calculates the degree up to which the derivative of the given function at the given point is zero consecutively, and thus can determine equal roots. It accepts the point x=a, at which the derivative is required to be found.

$$f(x) = x^n + a_1x^{n-1} + \cdots + a_n$$

The derivative is calculated as

$$f'(x) = nx^{n-1} + a_1(n-1)x^{n-2} + \cdots + a_{n-1}$$

Putting the value of ***a*** in this expression, we find the value of the first derivative at ***a***. If this value is zero, we calculate the second order derivative and check if its zero. The process is repeated till the derivative becomes non-zero after a particular degree. The function returns the degree up to which the derivative is zero consecutively.

For a given root, its multiplicity is given by the number of times the derivative of the function is zero at that point. The return value determines the degree of multiplicity of the particular root and thus eliminates the need of checking the same root more than once. It also helps in determining the sign of roots.

**Algorithm – Derivative Check**

**Input**- 1.array coef[ ] containing coefficients of the given polynomial of degree n
2. Value of x at which the derivative of the polynomial is to be calculated
**Output**- an integer to represent till how many degrees the derivative of the polynomial is zero at the Given value of x. e.g.- 0 if f'(x) is not zero. 1 if f'(x) is zero. 2 if f'(x) and f''(x) is zero ….etc.

```
1: Assign s=0
2: Assign b=n
3: Assign c=0
4: while s = 0
5:         for i=0 to b
6:                 Update coef [i] = coef[i]*(b-i)
```

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 13, October 2016

```
7:              Update s = s + (coef[i] * pow(a,(b-i)))
8:      Update b=b-1
9:      If s = 0
10:             Update c = c+1
11:             endif
12: return c
```

## 4. RUFFINI'S METHOD

Polynomials may have real roots as well as complex roots, and as per the algorithm's implementation of Graeffe's algorithm, finding complex roots is a complex process. The resulting polynomial after dividing the polynomial by its factors of the form (x – real_root) is the polynomial which has the complex roots of the original polynomial. This polynomial can then be solved by Bairstow's algorithm or another efficient complex root finding algorithm to find the complex roots of the original polynomial.

To find the polynomial having just the complex roots of the original polynomial of degree n, we implement Ruffini's method.

The rule establishes a method for dividing the polynomial

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

by the binomial

$$Q(x) = x - r$$

to obtain the quotient polynomial

$$R(x) = b_{n-1} x^{n-1} + b_{n-2} x^{n-2} + \cdots + b_1 x + b_0,$$

To divide $P(x)$ by $Q(x)$:

1. The coefficients of $P(x)$ are taken down in order.

```
P(x)|        aₙ        aₙ₋₁        ...        a₁        a₀
    |
  r |
----|-------------------------------------------------------
    |
    |
```

2. The first coefficient of Q(x) is the same as P(x)

```
P(x)|        aₙ        aₙ₋₁        ...        a₁        a₀
    |
  r |
```

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 5, Special Issue 13, October 2016**

```
----|-------------------------------------------------------
    |          a_n
    |
Q(x)|       = b_n-1
```

3. To find the other coefficients, multiply the previous coefficient of Q(x) with **r** and add it to the current coefficient of P(x)

$$A[j] = (A[j-1] * root[i]) + A[j]$$

```
P(x)|          a_n          a_n-1          ...          a_1          a_0
    |
  r |                        b_n-1r
----|-------------------------------------------------------
    |          a_n       a_n-1+(b_n-1r)
    |
Q(x)|       = b_n-1       = b_n-2
    |
```

4. Repeat step 3 until no numbers remain

```
P(x)|          a_n          a_n-1          ...          a_1          a_0
    |
  r |                        b_n-1r         ...          b_1r         b_0r
----|-------------------------------------------------------------------
    |          a_n       a_n-1+(b_n-1r)     ...       a_1+b_1r       a_0+b_0r
    |
Q(x)|       = b_n-1       = b_n-2          ...       = b_0          = s
    |
```

The $b_i$ values are the coefficients of the result ($R(x)$) polynomial, the degree of which is one less than that of $P(x)$. The final value obtained, $s$, is the remainder.After division by each root, the degree of the polynomial is decremented by 1.

$$n = n-1$$

The above process is repeated till all the real roots are exhausted.

---

**Algorithm – Ruffini**

---

**Input:** The coefficients of a polynomial A= { $a_0$, $a_1$, $a_2$, ……., $a_n$ }, its degree n and its real roots in an array root having k roots
**Output:** The coefficients of the polynomial of degree n-k A= { $a_0$, $a_1$, $a_2$, ……., $a_{n-k}$ } after being divided by ( x – root[i] ), i = 1 to k

```
1: for i=1 to k
2:         for j=1 to n
3:                 A[j] = (A[j-1] * root[i]) + A[j]
4:         Update n = n-1
```

---

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 13, October 2016

### 5. *BAIRSTOW'S ALGORITHM*

Bairstow's Algorithm is an iterative root finding algorithm, which can be used to find both real and complex roots of a polynomial. It is based on the idea of synthetic division of the given polynomial by a quadratic function and can be used to find all the roots of a polynomial by recursive calls. Since it divides the polynomial by a quadratic, at each step, it finds two roots of the polynomial.

As for the polynomial f(x), the GBDRB algorithm has already found the real roots using Graeffe's algorithm and got the polynomial having complex roots only using Ruffini's method.

Bairstow's method divides the polynomial f(x) by a factor p(x) = $x^2 - rx - s$ and the quotient is a polynomial g(x) of degree n-2

$$g(x) = b_0 x^{n-2} + b_1 x^{n-3} + \cdots + b_{n-3} x + b_{n-2}$$

The remainder is a linear polynomial $r(x) = b_{n-1}(x - r) + b_n$

Since, the quotient g(x) and r(x) are found out by synthetic division, the coefficients can be related by the recurrence relation:

$$b_0 = a_0$$

$$b_1 = a_1 + r * b_0$$

$$b_i = a_i + r * b_{i-1} + s * b_{i-2}, for\ all\ i = 2\ to\ n$$

If p(x) is an exact factor of f(x), then r(x) should be equal to 0. And roots of p(x) are thus roots of f(x). The factor p(x) = $x^2 - rx - s$ is based on guess values of r and s, and so, Bairstow's method primarily focuses on reducing the relative errors in r and s such that r(x) becomes 0. For such values of r and s, Bairstow's method follows a similar approach as to Newton Ralphson's method.

Since, $b_n$ and $b_{n-1}$ are both functions of r and s, we can have Taylor expansion of $b_n$ and $b_{n-1}$ as:

$$b_{n-1}(r + \Delta r, s + \Delta s) = b_{n-1} + \frac{\partial b_{n-1}}{\partial r}\Delta r + \frac{\partial b_{n-1}}{\partial s}\Delta s + O(\Delta r^2, \Delta s^2)$$

$$b_n(r + \Delta r, s + \Delta s) = b_n + \frac{\partial b_n}{\partial r}\Delta r + \frac{\partial b_n}{\partial s}\Delta s + O(\Delta r^2, \Delta s^2)$$

The second term $O(\Delta r^2, \Delta s^2)$ and further higher terms can be neglected as $\Delta r^2$and $\Delta s^2$ tends to 0. To further improve the approximation value of r and s, we can equate both the equations to 0, and we find:

$$\frac{\partial b_{n-1}}{\partial r}\Delta r + \frac{\partial b_{n-1}}{\partial s}\Delta s = -b_{n-1}$$

$$\frac{\partial b_n}{\partial r}\Delta r + \frac{\partial b_n}{\partial s}\Delta s = -b_n$$

These system of equations need partial derivatives of $b_n$ and $b_{n-1}$ with respect to r and s to be solved. Bairstow has shown that these partial derivatives can be obtained by synthetic division of g(x), which results in the similar previous recurrence relation with $a_i$ replaced by $b_i$ and $b_i$by$c_i$:

$$c_0 = b_0$$

$$c_1 = b_1 + r * c_0$$

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 13, October 2016

$$c_i = b_i + r * c_{i-1} + s * c_{i-2}, for\ all\ i = 2\ to\ n$$

Where $\frac{\partial b_n}{\partial r} = c_{n-1}, \frac{\partial b_n}{\partial s} = \frac{\partial b_{n-1}}{\partial r} = c_{n-2}, and\ \frac{\partial b_{n-1}}{\partial s} = c_{n-3}$

Therefore, the previous system of equations can be written as

$$c_{n-2}\Delta r + c_{n-3}\Delta s = -b_{n-1}$$

$$c_{n-1}\Delta r + c_{n-2}\Delta s = -b_n$$

These sets of equations can be solved to find out the values of $\Delta r\ and\ \Delta s$. The approximate errors in the assumed values are given by:

$$\in_{a,r} = \left|\frac{\Delta r}{r}\right| * 100$$

$$\in_{a,s} = \left|\frac{\Delta s}{s}\right| * 100$$

As long as $\in_{a,r} > \in_s$ and $\in_{a,r} > \in_s$, where $\in_s$ is the iteration stopping error, we repeat the process with updated values of$(r + \Delta r, s + \Delta s)$. When iteration stopping error is finally larger than the percentage errors, one can calculate the roots of p(x) using Sridhar Acharya's rule, which may yield real or complex roots. These roots will also be roots of f(x).Thus after each execution of Bairstow algorithm with multiple iterations depending upon stopping error, we get two roots at max, and the quotient g(x) of n-2 degree can again be passed through the algorithm, i.e. recursive call.

Given the time it takes to narrow down on the correct values of r and s, and just one quadratic takes multiple iterations, it is time consuming in some cases, and this algorithm only uses it to finding complex solutions of polynomials.

**Algorithm – Bairstow**

**Input**: A polynomial of degree n having coefficients A= { $a_0$, $a_1$, $a_2$, ……., $a_n$ }
**Output:** A polynomial $x^2$-rx-s and its roots with accuracy of $10^{-6}$, such that, it perfectly divides the polynomial taken as input, and the quotient polynomial being stored again in A with degree n-2, and recursive call to find all roots of input polynomial

1: **if** n==0
2: **return**
3: **elseif** n==1
4: Root is -A[1]
5: **return**
6: **elseif** n==2
7: Assign Discriminant = A[1]*A[1] - 4*A[0]*A[2]
8: Roots are ( -A[0] ± squareroot(Discriminant) )/2
9: **return**
10: **elseif** n>2
11: Assign r = A[1]/A[0] and s = A[2]/A[0]
12: Take delr and dels, which are small increments of r and s, and arrays B and C, B being the quotient.
13: Take Er and Es, which are percentage errors in r and s and assign Er=100 and Es=100
14: **while** Er < $10^{-6}$ and Es < $10^{-6}$
15: Assign B[0] = A[0] and C[0] = A[0]

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 5, Special Issue 13, October 2016

16: Assign B[1] = A[1]+(r * B[0]) and C[1] = B[1]+(r * C[0])
17: **for** i=2 to n
18: Assign B[i] = A[i] + (r * B[i-1])+(s * B[i-2])
19: Assign C[i] = B[i] + (r * C[i-1])+(s * C[i-2])
20: Assign delr = $\frac{((B[n]*C[n-3])-(B[n-1]*C[n-2]))}{((C[n-2]*C[n-2])-(C[n-3]*C[n-1]))}$
21. Assign dels = $-\frac{B[n]+(C[n-1]*delr)}{C[n-2]}$
22. Assign r = r + delr
23. Assign s = s + dels
24. Assign Er = $\left|\frac{delr}{r}\right| * 100$
25. Assign Es = $\left|\frac{dels}{r}\right| * 100$
26. Assign Discriminant = $r^2$+4s
27. Roots are (r ± squareroot(Discriminant) )/2
28. Update n = n-2
29. **for** i=0 to n
30. A[i] = B[i]
31. **if** n>0
32. Recursive call this method with updated A and n as input
33. **endif**

## III. CONCLUSION

We have combined multiple root finding algorithm and methods and formed a suitable cluster to find roots of nth degree polynomials with high precision. We have found out that Graeffe's Algorithm is best suited to find the real roots and their approximate values, and has a complexity of O ($n^2$). One can find accuracy of the real roots found here with Bisection, but approximating complex roots from the algorithm is hard to implement, although achievable. Also, Graeffe's algorithm can't solve polynomials of the form $x^n + x^{(n-1)} + ... + x^2 + x + 1$. But it can be easily found out by simple methods. Bisection method is pretty good at attaining accuracy of roots, but as a standalone, it would often be a problem to select initial values for which the polynomial has opposite signs. Using derivative to find inflection points is a good way of learning where equal roots are present and in what frequency. Bairstow's Algorithm has fast convergence after a few iterations, but each step has multiple iterations which increases time consumption, and depends on initial values of r and s. A particular kind of instability is observed when the polynomial has odd degree and only one real root. Quadratic factors that have a small value at this real root tend to diverge to infinity.

**Acknowledgement:** We thank Professor Kousik Dasgupta for inspiring discussions about researching in a topic, and guiding us in the right direction on this subject and the comments, that led to this improved presentation of the topic.

## REFERENCES

[1] http://en.wikipedia.org/wiki/Root-finding_algorithm
[2] http://www.google.com/patents/US8358817
[3] http://en.wikipedia.org/wiki/Newton's_method, Introduction to Numerical Analysis by Amritava Ghosh, 2004, ISBN- 81-87504-67-6, Newton-Raphson method, pg-180.
[4] http://en.wikipedia.org/wiki/Bisection_method, Introduction to Numerical Analysis by Amritava Ghosh, 2004, ISBN- 81-87504-67-6, Method of Bisection, pg-172.
[5] http://en.wikipedia.org/wiki/False_position_method, Introduction to Numerical Analysis by Amritava Ghosh, 2004, ISBN- 81-87504-67-6, Regula-falsi method, pg-192.
[6] http://en.wikipedia.org/wiki/Brent's_method, Optimal Iterative Process for Root-Finding by Richard Brent, Shmuel Winograd and Philip Wolfe. Mathematical Sciences Department, IBM Watson research center, Yorktown Heights, New York. August 3, 1972.
[7] http://en.wikipedia.org/wiki/Secant_method, Introduction to Numerical Analysis by Amritava Ghosh, 2004, ISBN- 81-87504-67-6, Secant method, pg-188.

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

*(An ISO 3297: 2007 Certified Organization)*

**Vol. 5, Special Issue 13, October 2016**

[8] http://en.wikipedia.org/wiki/Inverse_quadratic_interpolation, Introduction to Numerical Analysis by Amritava Ghosh, 2004, ISBN- 81-87504-67-6, Inverse Interpolation method, pg-185.
[9] http://en.wikipedia.org/wiki/Halley's_method, On the Geometry of Halley's Method by T. R. Scavo and J. B. Thoo The American Mathematical MonthlyVol. 102, No. 5 (May, 1995), pp. 417-426. Published by: Mathematical Association of America. Article Stable URL: http://www.jstor.org/stable/2975033
[10] http://en.wikipedia.org/wiki/Laguerre's_method
[11] http://en.wikipedia.org/wiki/Horner's_method
[12] http://en.wikipedia.org/wiki/Graeffe%27s_method, Introduction to Numerical Analysis by Amritava Ghosh, 2004, ISBN- 81-87504-67-6, Graeffe's method for algebraic equations, pg-196.
[13] http://en.wikipedia.org/wiki/Ruffini's_rule
[14] http://en.wikipedia.org/wiki/Bairstow's_method