Evaluation of Modified Regula Falsi Methods

David (Jung Won) Yang

jwy273

Abstract

One dimensional numerical zero finding methods saw active development and research until the latter half of the twentieth century. A good zero finding method should be stable and efficient. The bisection method is very stable, but lacks efficiency. On the other the secant method can achieve fast rates of convergence but it lacks stability. Brent's method utilizes both methods and another interpolating method to provide the best of both words. Brent's method became perhaps the most widely used general zero finding method in the world. However, there is a family of methods labeled the modified regula falsi methods that also provide both efficiency and stability. We compare the efficiency and stability of Brent's method with a few modified regula falsi methods. In our test results, the performance of the regula falsi methods were comparable and even competitive with Brent's method. This is interesting and significant because the implementation of the modified regula falsi methods are much less complex than Brent's method.

I. Introduction

Zero finding or solving for the zero of a continuous function is a very fundamental problem that is ubiquitous in mathematics. It is often the case that the solution for a root of an equation does not have a closed form solution, or the closed form, analytical solution or the solution is too complex to evaluate. In such a case, a numerical solution is desirable or required. When thinking of numerical methods for zero finding, the most well known method is most likely Newton's method (in fact it is probably the most famous method of numerical analysis). Newton's method is a brilliant algorithm, that is commonly used, and it has many positive characteristics such as quadratic convergence, and the fact that it requires only one starting value. However, Newton's method cannot be used as a general zero finding algorithm because it requires that the function be differentiable, and convergence rests on many conditions. There have been many methods that have been proposed for general zero finding or methods that do not require a derivative, however, there has been very little talk about these methods for the past few decades. Currently, the most widely used zero finding algorithm is more likely Brent's method, which is the method that is implemented in Matlab's function fzero. There seems to be a consensus that this method is the optimal function. However, there is a family of methods called the regula falsi methods that rival the performance and stability of Brent's method. These methods are called modified regula falsi methods, and we will compare these methods to Brent's method in this paper.

II. Zero Finding Methods

The first and perhaps simplest method is the bisection method. The bisection method takes two endpoints of different signs for the initial values. Bisection does not require the functions values, only their signs. From the intermediate value theorem we know that a root must exist in the interval between the two end points. We divide the interval in half by selecting the midpoint and either take the interval to the right of the midpoint if the midpoint is negative, or the left of the interval if the midpoint is positive. Convergence is guaranteed, and bisection always convergences at the same rate. At every iteration bisection reduces the

interval of uncertainty (the interval where the zero can lie) by a factor of $\frac{1}{2}$ so the method has a linear rate of convergence. The benefit of bisection is that it is completely reliable, barring numerical errors, however the rate of convergence is not satisfactory.

The next method is the secant method, and this method, unlike bisection, utilizes the function values. The secant method takes two end points for the initial values (ideally you would want two endpoints with different signs like bisection, but it is not a requirement). Then a linear interpolation is made between the two end point, and the zero of the interpolating linear function is calculated. This zero then becomes an end point. To further explain, if the iterates x_{k-1} and x_k, and their respective function values $f(x_{k-1})$ and $f(x_k)$ are used to interpolate a linear function, and the zero of this linear function becomes x_{k+1}. This new iterate replaces $f(x_{k-1})$ as an endpoint. Under certain assumptions, it can be shown that the secant method has a superlinear rate of convergence, and for many cases this does indeed occur. However, the secant method is very sensitive to initial values, and is very unstable. One instance that frequently leads to issues is when both endpoints of the secant method have the same sign. When both endpoints have the same sign, the linear interpolant will not intersect the x axis between the two endpoints, and thus the new iterate will be extrapolated beyond the endpoints. Not only is this extrapolation unreliable, as we cannot know whether the next iterate is progressing in the right direction, but it can be very problematic for convergence. It's often the case that the extrapolation leads to the selection of an iterate that is very far away from the previous two iterates.

In an attempt to resolve the issues of instability of the secant method, the regula falsi method was developed. Unlike the secant method, the regula falsi methods requires that the two end points have different signs. When a new iterate, x_{k+1} is computed through linear interpolation, we do not replace the endpoint given by x_{k-1}, but rather replace the endpoint that has the same sign as x_{k+1}. This way, the two ends points at any given iteration will have opposite function values. This means that the interval of uncertainty is well defined like bisection, and linear interpolation between the endpoints will never lead to extrapolation outside of the interval. Regula falsi is indeed very reliable, and in addition its convergence can be super linear like the secant method. However, regula falsi performs very poorly in certain intervals, namely convex and concave intervals. In convex or concave intervals, one end point remains fixed, and the other endpoint converges to the root. The convergence or decreasing of the interval of uncertainty is very slow as one end point is not being reduced at all and convergence is only happening on one side of the root. This one sided convergence leads to very slow convergence in many cases. In fact regula falsi often displays a linear rate of convergence that is much slower than bisection.

Regula falsi has two very desirable features: stability, and possible secant-like super linear convergence. However, the fact that it can run into an extremely slow convergence pattern is a huge issue, which makes the method unreliable and thus keeps it from being used. However, the method did not become irrelevant as a family of methods were developed that sought to fix the issues with convergence in regula falsi. These methods are referred to as modified regula falsi methods in Sergio Galdino's paper "A Family of Regula Falsi Root-Finding Methods."

These regula falsi methods, or rather most zero finding methods, have largely been forgotten in the last decade. Considerations of the different one dimension zero finding methods seem to have come to an end after the development and implementation of Brent's method. This method is perhaps the most widely used general zero finding routine today as it is the routine that is run in Matlab's zero finding function fzero. After Brent's method was released, zero finding seemed to have been considered a solved problem. Brent's method was believed to have the reliability of a bracketing method, and also the converge of an open method such as the secant method or inverse quadratic interpolation. This paper seeks to compare Brent's method to show that the modified Regula Falsi methods should be considered as direct competitors to Brent's method.

III. Modified Regula Falsi

The modified regula falsi methods all aim to solve the issue of one sided convergence that is characteristic of regula falsi. The proposed solution is to prevent one sided convergence by making sure that a function value is not used more than once for linear interpolation. When regula falsi falls into one sided convergence, one of the end points remains fixed, and thus the function value corresponding to the fixed point also remains fixed. This means that the same function value is being used for interpolation. The modified regula falsi methods seek to solve this by assigning "fake" function values to the endpoints so that the fake function value is not continuously used. These function values are essentially a factor of the original fixed function value and this factor can be computed in several ways. Ned Anderson describes it well when he says that "the idea behind the Illinois method is to interrupt the regula falsi method whenever it is not the secant method"[4]. This description can be applied to all modified regula falsi methods, and even Algorithim A.

Here is a brief explanation of the general algorithm for the methods

---

Modified Regula Falsi:

1) Get two initial end points, that have different signs and make them the end points. Let us call the end points a and b.

2) Do linear interpolation to get the next iterate x

3) If x and a have different signs, assign x to b. If this is not the case, then x and b have different signs assign x to a.

4) From this point on we must check if an end point has been used twice for linear interpolation to compute the next iterate. If an end point has been used twice we will decrease the function value that corresponds to that end point by a factor, μ. This is what we mean when we say that we assigning a "fake" function value. This reduced function or "fake" function value is then used in the next linear interpolation.
5) For every subsequent iteration that the fixed end point is used, the function of that end point is further decreased by a factor of μ.
6) This continues until we get an iterate, x, that has a function value, f(x), that has the same sign as the function value of the fixed end point. Then we can replace the fixed end point with the a new end point x.

---

if we let x_{i+1}, x_i, and x_{i-1} be a sequence of iterates, and f_{i+1}, f_i, f_{i-1} be the corresponding function values. If $f_{i+1}f_i > 0$, then f_{i-1} must be used as an end point again. However, f_{i-1} has been used as an endpoint twice, so f_{i-1} is replaced with the value μf_{i-1}. This is the so called modified step.

The idea is that by decreasing the fixed function value, the zero of the interpolant will be closer to the end point that has been used consecutively. The smaller the function value is (in terms of absolute value), the closer the zero of the interpolant will be to the fixed end point. When we bring the zero of the interpolant (which becomes the next iterate) closer to the fixed end point there are two possible outcomes. We either speed up convergence by bringing the next iterate closer to the zero of the function, or we bring the next iterate to the same side as the fixed end point. By side we mean the side with respect to the zero of the function. If the next iterate is on the same side as the fixed end point, we can replace the fixed end point. Thus the issue of the fixed endpoint is in a way resolved.

The different modified regula falsi methods all follow the explanation that was given above. The only difference between the methods is the factor μ. There are many different choices for this factor. The most simple choice is $\frac{1}{2}$ which is used by Wheeler's method or the Illinois method. Then there is also the Pegasus method which uses a much more complicated factor that utilizes function values. The factor for the Pegasus

method is $\frac{fa*fb}{fb+fx}$ where fa and fb are the function values for the end points, and fx is the function value for the newly calculated iterate.

An error analysis for the Illinois method and for Pegasus method was given by M. Dowell, and P. Jarratt in their papers [2],[3]. In their error analysis for the Illinois method, they conclude that asymptotically, the efficiency and rate of convergence of the Pegasus method is superior to the Illinois method, and that both methods are superior to regula falsi. Asymptotic gives information about the performance of an algorithm, however, it cannot explain how the methods will perform in actuality. Not only is the performance asymptotic, but it often rests on assumptions. For example Newton's method asymptotically has quadratic convergence, but since it relies on many assumptions, convergence is often much slower.

IV. Algorithm A

Algorithm A was presented by Ned Anderson and Ake Bjorck in their paper [4]. The method can be considered another modified regula falsi method, however, the method is arguably more complex. Algorithm A is stated as a higher order method. The reason is because, the Algorithm computes a quantity using three functions values. However, this is no different from the Pegasus method which uses 3 function values as well.

Like the other regula falsi methods, the algorithm takes a step using the secant method unless an iterate was used twice. In other words, if if we let x_{i+1}, x_i, and x_{i-1} be a sequence of iterates, and f_{i+1}, f_i, f_{i-1} be the corresponding function values. If $f_{i+1}f_i > 0$, then a step of type "P" is taken to compute the next iterate x_{i+2}. The formula for calculating the iterate under a step of "P" is,

$$P : x_{i+2} = x_{i+1} - f_{i+1}/\bar{f}'_{i+1}$$

$\bar{f}_{i+1}$ is the interpolating parabola to the three previous points. If we look at the formula, it has the form of Newton's equation except the derivative of the function f is replaced by the derivative of the interpolating parabola.

The step of P can also be interpreted as a modification step that we know from the modified regula falsi methods. After doing some manipulation, it can be shown that the step of P is equivalent to doing linear interpolation after replacing f_{i-1}, the function value at the fixed/retained endpoint, with
$g = f_{i+1} + (x_{i-1} - x_{i+1})\bar{f}'_{i+1}$. Then after some further manipulation, it can be show that

$$g = \mu f_{i-1}, \quad where \quad \mu = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} / \frac{f_i - f_{i-1}}{x_i - x_{i-1}}$$

We can clearly see the form shared by the family of regula falsi methods.

In the step of P, it is possible for the calculated iterate x_{i+2} to not be within the interval of uncertainty set up by the previous two iterates or it is possible that $\bar{f}'_{i+1} = 0$. In order to keep x_{i+2} within the interval, and for $\bar{f}'_{i+1} \neq 0$, we must have $\mu > 0$. In the case that $\mu \leq 0$, we simple use the modified step from the Illinois method. To sum up, the method does a step of P whenever the conditions for the secant step aren't met, and does a step of Wheeler whenever the conditions for a step of P aren't met.

Anderson and Bjorck do an asymptotic analysis for Algorithm A, and in their error analysis they conclude that Algorithm A has superior efficiency compared to both Pegasus and the Illinois method. They also show that asymptotically the modification step from Wheeler's method is never taken.

V. Brent's Method

Brent's method is perhaps the most widely used one dimensional zero finding method. It is the method that is implemented in Matlab's zero finding function, fzero. Brent's method was developed by Richard P.

Brent and was published in his book, "Algorithms for Minimization without Derivatives." The predecessor to Brent's method was Dekker's method, which used bisection to safe guard against the instability of the secant method. In Dekker's method, a step of secant is taken unless the step of secant is not between $\frac{a+b}{2}$, and the previous iterate (a and b represents the end points of the interval that bracket a sign change). If the secant method is outside of this interval, and a step of bisection is taken instead. Brent's was an improvement to Dekker's method making it faster and safer. Brent's method makes several significant changes to Dekker's method. First, there is an extra variable e in Brent's method that keeps track of the ratio of the size of the steps between two successive steps. This variable is important because it is used to create extra situations in which bisection can be called. This helps prevents cases where the interpolation methods make very little progress or movement which is an issue with Dekker's method according to Wilkinson and Gu. Brent's method also adds another interpolation method. A step of inverse quadratic interpolation is taken if certain conditions are met. If these conditions are not met, then a step of secant is taken instead. Inverse quadratic interpolation is known to have a higher order of convergence than secant. The convergence near the zero is very fast, however, its global behavior is unpredictable. Brent's method is able to safeguard against the unpredictable behavior of inverse quadratic interpolation with bisection and secant.

In their paper, Wilkinson and Gu mentions that Brent's method performs poorly for very flat intervals [5]. However, they did not give any functions as examples except for the function they specifically constructed to display the n^2 convergence, where n is the number of operations it takes for bisection to converge.

The stopping criteria for Brent's method is :

```
% value of b, and c is on the opposite side of the zero from b.
% Convergence test and possible exit
m = 0.5*(c - b);
toler = 2.0*tol*max(abs(b),1.0);
if (abs(m) <= toler) || (fb == 0.0)
    break
end
```

The tolerance in Brent's method (in fzero) is a tolerance on the interval of uncertainty, and it is relative to the value at the endpoints.

VI. Experiment

Algorithm A, the Illinois method, and the Pegasus method were all implemented in Matlab. A test was implemented to compare the performance with Brent's method. We tested the performance on 48 different functions. The performance was measured by the number of function evaluations it took to meet the convergence criteria. We use the convergence criteria presented in Matlab's fzero. The convergence criteria is based on a tolerance of x rather than a tolerance of the function value $f(x)$. We set the tolerance to 10^{-16}. Most of the functions were from the research paper of Wilkins and Gu [5]. Some of the functions were taken from Ned Anderson, and Ake Bjorck's paper [4]. We would like to note that we are only testing simple zeros, so there must be different signs on each side of the root. In addition, we only selected intervals that contained exactly one zero. We also implemented the "step of delta", a technique used in Brent's method into the function into the other methods. The "step of delta" that was implemented was simpler than the one that is implemented in Matlab's fzero, and it was translated to Matlab from the Algol code in Ake Bjorck's paper.

VII. Results

Average Number of Function Evaluations

Alg A	Illinois	Pegasus	Brent
10.3571	11.8571	10.4643	9.5179

From the tests we see that all of the functions converged super linearly. We see that Brent's method had the lowest number of function evaluations on average. Brent's method was more efficient than Algorithm A and Pegasus by an average of around one function evaluation.

Here is a table for the number of function evaluations for each function. If a function was listed more than once, then different endpoints were used.

function:	Alg A	Illinois	Pegasus	Brent
sin(x) - 0.5:	9	11	10	10
2*x*exp(-7)+1-2*exp(-7*x):	11	13	11	11
x^2-(1-x)^20:	14	13	14	13
1+(1+(1-15)^4)*x-(1-15*x)^4:	3	3	3	2
-(x+sin(x))*exp(-x):	23	24	21	13
-(x+sin(x))*exp(abs(x)):	3	3	3	3
exp(-10*x)*(x-1)+x^10:	11	16	16	10
(2*x-1)/x:	6	15	15	4
sqrt(x) - cos(x):	8	11	8	8
3*(x+1)*(x-5)*(x-1):	12	13	11	10
3*(x+1)*(x-5)*(x-1):	10	12	10	10
x^3 -7*x^2 + 14*x -6:	9	11	9	9
x^3 -7*x^2 + 14*x -6:	12	14	12	14
x^4 -2*x^3 -4*x^2 + 4*x +4:	12	11	11	10
x^4 -2*x^3 -4*x^2 + 4*x +4:	9	10	9	9
x-2*(-x):	3	3	3	2
exp(x) - x^2 + 3*x -2:	7	8	7	8
2*x*cos(2*x) - (x+1)^2:	10	13	10	9
2*x*cos(2*x) - (x+1)^2:	14	14	13	10
3*x - exp(x):	11	13	11	10
x + 3*cos(x) - exp(x):	7	9	7	8
x^2 - 4*x + 4 -log(x):	11	11	8	10
x^2 - 4*x + 4 -log(x):	10	12	10	11
x + 1 -sin(pi*x):	9	8	9	5
x + 1 -sin(pi*x):	4	4	4	6
exp(x) -2 - cos(exp(x-2):	13	14	13	10
(x+2)*(x+1)^2 *x*(x-1)^3 *(x-2):	11	13	11	11
(x+2)*(x+1)^2 *x*(x-1)^3 *(x-2):	16	19	17	14
(x+2)*(x+1)^2 *x*(x-1)^3 *(x-2):	12	14	13	12
x^4 -3*x^2 -3:	9	10	8	8
x^3 -x -1:	12	13	12	10
pi + 5*sin(x/2) - x:	11	11	10	10
2^(-x)-x:	7	8	7	6
(2-exp(-x)+x^2)/3 -x:	13	13	14	12
5*x^(-2)+2 -x;:	10	13	10	9
exp(x)/3 -x:	11	13	11	10
exp(x)/3 -x:	10	12	9	9
5^(-x) -x:	10	13	12	9
5*(sin(x)+cos(x))-x:	12	13	12	9
-x^3 -cos(x):	17	15	15	14
-x^3 -2*x^2-5:	13	15	14	12

x^3 + 3*x^2 -1:	9	12	9	11
x- cos(x):	8	10	9	8
x - 8 - 2*sin(x):	12	15	12	11
exp(x) + 2^(-x) + 2*cos(x) -6:	9	10	9	9
log(x-1) + cos(x-1):	11	13	10	10
2*x*cos(2*x) - (x-2)^2:	9	9	10	9
exp(x) - 3*x^2:	12	14	12	13
sin(x) - exp(-x):	8	11	8	8
3*x -exp(x):	11	13	11	10
x +3*cos(x) -exp(x):	7	9	7	8
x^2 -4*x +4 -log(x):	11	11	8	10
x +1 -2*sin(pi*x):	11	11	10	10
x +1 -2*sin(pi*x):	9	12	9	11
exp(1/(x-10)^2)-1.2:	15	16	15	14
exp(x)+10-1000:	13	14	14	11

We can see that all of the functions are very stable and there was no case among the 46 functions where the methods did not converge or took a very large number of function evaluations to converge (large as in over 30 function evaluations). We see that it took the regula falsi methods an exceptionally large number of function evaluations to find the zero of the function $-(x + \sin x) * e^{-x}$, where as Brent's method did not have any exceptional difficultly with that function. The function is very convex around the zero, and that may have posed a some difficulty for the modified regula falsi methods, as the extreme convexity may have caused a lot of modified or "fake" function value steps, instead of the usual unmodified secant steps. However, for convex or concave intervals in general, convergence does not seem to be too slow. For example, the function $exp(x)+10-1000$ is convex, but convergence did not deviate much from the average convergence.

Here are the sequence of steps taken for Illinois and Pegasus for $-(x + \sin x) * e^{-x}$.
Illinois: MMMMMMMMUUMMMUUMUUMUUU
Pegasus: MMMMMMMUUMMUUMMUUMMU

Here is the same data except we implemented the "step of delta" into the modified regula falsi methods and Algorithm A.

Average Number of Function Evaluations

Alg A	Illinois	Pegasus	Brent
10.1964	11.7321	10.3214	9.5179

function:	Alg A	Illinois	Pegasus	Brent
sin(x) - 0.5:	9	11	10	10
2*x*exp(-7)+1-2*exp(-7*x):	11	13	11	11
x^2-(1-x)^20:	13	13	13	13
1+(1+(1-15)^4)*x-(1-15*x)^4:	3	3	3	2
-(x+sin(x))*exp(-x):	22	23	21	13
-(x+sin(x))*exp(abs(x)):	3	3	3	3
exp(-10*x)*(x-1)+x^10:	10	15	15	10
(2*x-1)/x:	6	15	15	4
sqrt(x) - cos(x):	8	11	8	8
3*(x+1)*(x-5)*(x-1):	12	13	11	10
3*(x+1)*(x-5)*(x-1):	10	12	10	10
x^3 -7*x^2 + 14*x -6:	9	11	9	9
x^3 -7*x^2 + 14*x -6:	12	14	12	14
x^4 -2*x^3 -4*x^2 + 4*x +4:	12	11	11	10

x^4 -2*x^3 -4*x^2 + 4*x +4:	9	10	9	9
x-2*(-x):	3	3	3	2
exp(x) - x^2 + 3*x -2:	7	8	7	8
2*x*cos(2*x) - (x+1)^2:	10	13	10	9
2*x*cos(2*x) - (x+1)^2:	11	12	10	10
3*x - exp(x):	11	13	11	10
x + 3*cos(x) - exp(x):	7	9	7	8
x^2 - 4*x + 4 -log(x):	11	11	8	10
x^2 - 4*x + 4 -log(x):	10	12	10	11
x + 1 -sin(pi*x):	9	8	9	5
x + 1 -sin(pi*x):	4	4	4	6
exp(x) -2 - cos(exp(x-2):	13	14	13	10
(x+2)*(x+1)^2 *x*(x-1)^3 *(x-2):	11	13	11	11
(x+2)*(x+1)^2 *x*(x-1)^3 *(x-2):	16	19	17	14
(x+2)*(x+1)^2 *x*(x-1)^3 *(x-2):	12	14	13	12
x^4 -3*x^2 -3:	9	10	8	8
x^3 -x -1:	12	13	12	10
pi + 5*sin(x/2) - x:	11	11	10	10
2^(-x)-x:	7	8	7	6
(2-exp(-x)+x^2)/3 -x:	13	13	14	12
5*x^(-2)+2 -x;:	10	13	10	9
exp(x)/3 -x:	11	13	11	10
exp(x)/3 -x:	10	12	9	9
5^(-x) -x:	10	13	12	9
5*(sin(x)+cos(x))-x:	10	11	10	9
-x^3 -cos(x):	16	14	14	14
-x^3 -2*x^2-5:	13	15	14	12
x^3 + 3*x^2 -1:	9	12	9	11
x- cos(x):	8	10	9	8
x - 8 - 2*sin(x):	12	15	12	11
exp(x) + 2^(-x) + 2*cos(x) -6:	9	10	9	9
log(x-1) + cos(x-1):	11	13	10	10
2*x*cos(2*x) - (x-2)^2:	9	9	10	9
exp(x) - 3*x^2:	12	14	12	13
sin(x) - exp(-x):	8	11	8	8
3*x -exp(x):	11	13	11	10
x +3*cos(x) -exp(x):	7	9	7	8
x^2 -4*x +4 -log(x):	11	11	8	10
x +1 -2*sin(pi*x):	11	11	10	10
x +1 -2*sin(pi*x):	9	12	9	11
exp(1/(x-10)^2)-1.2:	15	16	15	14
exp(x)+10-1000:	13	14	14	11

The "step of delta" decreased the average number of function evaluations by a small amount. The amount of improvement for the three methods is on average between .1 and .2 function evaluations.

VIII. Discussion

The results from the tests show that Brent's method is stable, consistent and fast, as expected. The other three methods were also stable, consistent and fast, however based on the tests, slower than Brent's method. However, the performance of Brent's method and the Pegasus method were good enough to be competitive with Brent's method. We selected the flat function $e^{1/(x-10)^2} - 1.2$, however, Brent's function did not show

slow convergences for this function. We tested the functions on many different functions, and many of the intervals were convex intervals. For example, we tested methods on the convex function $e^x + 10 - 1000$. Unlike the original regula falsi, the modified regula falsi methods, in general, did not have issues with convex or concave intervals.

In Cleave Molar's article, "Are we there yet" [6], he gives us a reason to revisit the modified regula falsi methods. In his article he speaks or the application of numerical zero finding techniques in software for differential equations. In the context of differential equations, we are dealing with vector valued functions, and thus we need to be able to expand the logic of zero finding methods to vector valued function. According to Molar, because of the complexity of the method and the number of components, Brent's method cannot be vectorized. However, the Illinois method, which is much simpler than Brent's method, can be vectorized. This may suggest that the other modified regula falsi methods that we discussed can be vectorized as well. Not only are the modified regula falsi methods competitive with Brent's method in performance, but there are potential applications and extensions for these methods in areas and problems for which Brent's method is not suitable.

Matlab Implementation of Methods

```
function [zero,funcCount] = modregfalsi(f,a,b,mew,tol,maxit)
%Input: f is a function handle for the function were are finding a zero
%for. a and b are two initial iterates that must differ in sign. mew is a
%function that computes the reduced or "fake" function value. tol is the
%tolerance in x for which we can consider the iterates as converged, maxit
%is the maximum number of iterates that the method can run.

fa=f(a); fb=f(b);
x = b; fx = fb;

%Let n be the number of function evaluations.
n=2;

i = 1;
while ((i<=maxit) && (abs(b-a)>tol) && (fx ~= 0))

    %convergence criteria
    m = 0.5*(b - a);
    toler = 2.0*tol*max(abs(b),1.0);
    if (abs(m) <= toler) || (fb == 0.0)
        break
    end

    i = i+1;
    %Step of secant.
    dx = -fx/(fb-fa) * (b-a);
    x = x+dx;

    %step of delta
    if abs(x-b) < tol; x = b - tol*sign(b-a); end

    fx = f(x);
    n = n+1;
    if fx*fb <0
        a = b; fa = fb;
    else
        fa = mew(fa,fb,fx);
    end
    b=x; fb=fx;
end
zero = x;
funcCount = n;
%format = 'n: %4d  f(x): %16e\n';
%fprintf(format,n,fx);
end
```

```
function [zero,funcCount] = AlgorithmA(f,a,b,tol,maxit)
%Input: f is a function handle for the function were are finding a zero
```

```
%for. a and b are two initial iterates that must differ in sign. tol is the
%tolerance in x for which we can consider the iterates as converged, maxit
%is the maximum number of iterates that the method can run.

fa = f(a); fb = f(b);
x = b; fx = fb;

%Let n be the number of function evaluations.
n=2;

i=1;
while (i<=maxit)&& (abs(b-a)>tol) && (fx ~= 0)

    %convergence criteria
    m = 0.5*(b - a);
     toler = 2.0*tol*max(abs(b),1.0);
     if (abs(m) <= toler) || (fb == 0.0)
         break
     end

     i = i+1;
     w1 = (fb-fa)/(b-a);
     x = b- fb/(w1);

     %step of delta
     if abs(x-b) < tol; x = b - tol*sign(b-a); end

     fx = f(x);
     n = n+1;
     if fx*fb <= 0
         a = b; fa = fb;
     else
         w2 = (fx-fb)/(x-b);
         g = w2/w1;
         %If factor is positive, then do hyperbolic interpolation.
         if g > 0
             fa = g*fa;
         %If factor is non-positive do wheeler. q
         else
             fa = fa/2;
         end
     end
     b = x; fb = fx;
end
zero = x;
funcCount = n;
%format = 'n: %4d  f(x): %16e\n';
%fprintf(format,n,fx);
end
```

```
function mew = pegasus(fa,fb,fx)
    mew = fa*fb/(fb+fx);
end
```

```
function mew = wheeler(fa,~,~)
    mew = (fa/2);
end
```

```
function Tests()
%Testing the performances of the algorithms on various functions.
    fun = {@f1,@f2,@f3,@f4,@f5,@f6,@f7,@f8,@f9,@f10,@f11,@f12,@f13,@f14,@f15,
        @f16,@f17,@f18,@f19,@f20,@f21,....
         @f22,@f23,@f24,@f25,@f26,@f27,@f28,@f29,@f30,@f31,@f32,@f33,@f34,@f35,
             @f36,@f37,@f38,@f39,@f40,@f41,....
         @f42,@f43,@f44,@f45,@f46};


    tol = 10^(-16); %Tolerance
    maxit = 1000; %Maximum Number of Itertions
    %Initialize the arrays that hold the function values
    fevals_A = zeros(55,1);
    fevals_wheeler = zeros(55,1);
    fevals_pegasus = zeros(55,1);
    fevals_brent = zeros(55,1);

    n = 0;
    %Run the methods and save the number of function evaluations
    for i = 1:length(fun)
        [end_pts,f] = fun{i}();
        [a,b,k] = end_pts();
        for j = 1:k
            n = n+1;
           [~,fevals_A(n)] = AlgorithmA(f,a(j),b(j),tol,maxit);
           [~,fevals_wheeler(n)] = modregfalsi(f,a(j),b(j),@wheeler,tol,maxit)
               ;
           [~,fevals_pegasus(n)] = modregfalsi(f,a(j),b(j),@pegasus,tol,maxit)
               ;
           options = optimset('FunValCheck','on','TolX',tol);
           [x,fval,exitflag,output] = fzero(f,[a(j),b(j)],options);
           fevals_brent(n) = output.funcCount;
        end
    end

    %Diplay a mean number of function evaluation
    avg_funcCount_A = mean(fevals_A)
    avg_funcCount_wheeler = mean(fevals_wheeler)
    avg_funcCount_pegasus = mean(fevals_pegasus)
    avg_funcCount_brent = mean(fevals_brent)

```

```
38      %Run a script that displays the number of function evaluations for each
39      %function.
40      Tests1
41  end
```

References

[1] Sergio Galdino, A Family of Regula Falsi Root-Finding Methods, CET. 1(2011), 514-517

[2] M. Dowell, P. Jarratt, The "Pegasus" Method For Computing the Root of an Equation, BIT, 12(1972), 503-508

[3] M. Dowell, P. Jarratt, A Modified Regula Falsi Method for Computing the Root of an Equation, BIT, 11(1971), 168-174

[4] Ned Anderson, Ake Bjorck, A New High Order Method of Regula Falsi Type for Computing a Root of an Equation, BIT, 13(1973), 253-264

[5] Gauta Wilkins, Ming Gu, A modified Brent's method for finding zeros of functions, Numerische Mathematik, 123(2013), No.1, 177-188

[6] Cleave Moler, Are we there yet?, Matlab News and Notes, 1997, 16-17