Cui Liu Cai
Math 128B Report

A More Reliable Equation Solver –My_fzero (Matlab Version)

My_fzero is a reliable and efficient root-finder program in Matlab version. The idea of how My_fzero works comes from combining the ideas of "Personal Calculator Has Key to Solve Any Equation $f(x) = 0$" by Professor William M. Kahan, "An Equation Solver for a Handheld Calculator" by Paul J. McClellan, and the Matlab root-finder, fzero.

Introduction

Finding zeros/roots of a given function f, that is, find a number a such that $f(a) = 0$, is the most important and basic of tasks in many different fields. A lot of problems in physics, chemistry, economics, statistics and mathematics field have to use the method of finding zeros of functions. Therefore, a good method of determining the roots/zeros of functions (including polynomials, equations with exponential terms, logarithm terms, trigonometry terms, etc.) is very important. This topic has captured a lot of mathematicians' interests for centuries.

As the world becomes more and more technology-oriented, the problems we are facing today become more and more complicated. The functions we have to solve are not only some simple equations any more. They are usually some very complicated functions (for instance, some computer programmed functions). And usually they are passed to us as a black box. So a smart, efficient, and robust root finder program is essential for us today.

For us, the ideal equation solver should be able to find all solutions of any equation provided by the user. However, this is impossible in general for today. A more realistic expectation is that the root finder should have several key abilities. First, it can find one root at a time for most equations, beginning with user provided initial guess or guesses not necessarily restricted to be very near to the solution. Second, it can flexibly find minimum or maximum or limitation instead of insisting to find a root where there might be no root at all. Third, it can provide understandable and reliable diagnostic information should the solver fail to find a solution, so that the user can rewrite $f(x)$ into a new one which is easier to solve, or the user can give different initial guess or guesses. These are the design objective for the equation solver, "My_fzero".

How things are today?

We have many documents about how to find roots for equations with one or more independent variables. For instance, "Iterative Method for the Solution of Equations" by J. F. Traub, "Solution of equations and Systems of Equations" by Ostrowski, "The

Numerical Treatment of a single Non-linear equation" by Householder, etc. Here we will only consider the equations with one independent variable.

There are already a lot of numerical root-finding methods. The most popular methods include Bisection Method, Brent's Method, False Position Method, Inverse Quadratic Method, Muller's Method, Newton's Method, Ridders' Method, Secant Method, etc. The complexities of these methods vary a lot. Each method has some advantage for some individual cases, but none of them is a heal-all. For example, Bisection Method is a very simple and robust method. Given two points a and b where the function $f(x)$ reverses sign, and suppose $f(x)$ is computable for all the points between a and b, Bisection Method grantees to find a root where $f(x)$ vanishes or find two successive machine representable points where $f(x)$ reverses sign. However Bisection Method is usually very slow. And the requirement of $f(x)$ reverses sign at the given two points is hard to satisfy because the user usually does not know what the function is. Another famous method, Newton's Method, is theoretically good but hard to be applied, because it is generally impossible to calculate the derivative for a user-defined function $f(x)$. Even when the calculation of derivative is achievable, the performance of Newton's Method will not be good until the guess is near enough to the root, which is almost as hard as finding the root.

Having the idea that none of these root-finding methods can fit all the cases, it is reasonable to think that a good root finder should be a combination of many methods, and it should have the ability to choose a proper method for each step automatically. My_fzero is designed following this concept.

What is My_fzero? What does My_fzero do? When does it work?

My_fzero is an automatic numerical real root finder which can solve for the real roots of one variable equations of the form

$$f(x) = 0 \tag{1}$$

where $f(x)$ is generally a non-linear function of x defined by the user. $f(x)$ can be as simple as a function like $f(x) = x - 1$, and it can also be a very complicated user provided program. Also, $f(x)$ can be continuous or discontinuous, and might not be computable for some regions. For an illustration of the usage of My_fzero, consider the following function

$$f(x) = x^3 - 2x - 3 \tag{2}$$

The user can run:

```
My_fzero(inline('x.^3 -2*x -3'),1)
```

where inline('x.^3 -2*x -3') means an inline function of $f(x)$, and the initial guess is 1. The result 1.89328919630450 is displayed after the user run the above command.

Although some equation solvers use or partly use direct solver[1] to analyze $f(x)$ and attempt to solve it by applying rules of algebra, in general, no equation solver can understand $f(x)$ or the program that defines $f(x)$. Based on this reason, My_fzero does not consider direct solving method at all. It just takes $f(x)$ as a black box, and iteratively executes it with its guess or guesses as the argument, beginning with the initial guess or guesses provided by the user. It will consider a proper model to fit $f(x)$ based on the previous results, and find the best guess to the root as the next guess based on this model. If everything goes well, My_fzero will get closer and closer to the root, until it finally finds a guess where $f(x)$ vanishes, which must then be the root.

However, will $f(x)$ always vanish at its root? Mathematically speaking, yes. But for a computer, it is not necessary to be true. Any computer has its precision, which means computer cannot represent all real numbers precisely. For example, computer can represent 1/2 as 0.5 precisely, but 1/3 will probably be represented as 0.33…3 within a computer. Thus, for a computer, only discrete points in the real axis are representable, and the density of these representable points near x is determined by the value of x and the precision of the computer. This is the root of the notorious round off problem.

For example, if

$$f(x) = x - 1 - eps/2 \tag{3}$$

where *eps* is a constant of the floating-point relative accuracy in that computer, which means the distance from 1.0 to the next representable number. In most recent computers, *eps* is about 2.22e-16, not very small in fact. It's easy to see that mathematically $f(x)$ has a root at $1 + eps/2$. But because the computer cannot represent $1 + eps/2$ precisely, and the nearest two representable points are 1 and $1 + eps$, the smallest absolute value of $f(x)$ is $eps/2$, which is about 1.11e-16, but $f(x)$ never vanishes. Taking this in mind, the equation solver can not insist to find a root where $f(x)$ vanishes; instead, if it can find two successive representable numbers where $f(x)$ reverses sign, it will also declare that a root is found.

For the below three most common and easiest cases, My_fzero guarantees to find a root[2]:

1. $f(x)$ is strictly monotonic, regardless of initial guesses;

[1] Direct solver: the equation solver attempts to solve the user defined equation directly by applying rules of algebra to isolate the unknown on one side of an equation. If it succeeds, the value of the other side of the equation is the solution to the equation. Paul J. McClellan, "An Equation Solver for a Handheld Calculator", Hewlett-Packard Journal, August 1987

[2] William M. Kahan, "Personal Calculator Has Key to Solve Any Equation f(x) = 0", Hewlett-Packard Journal December 1979

2. $\pm f(x)$ is strictly convex, and there is a representable point where $f(x)$ is 0, or there are two representable points where $f(x)$ reverses sign, regardless of initial guesses;

3. $f(x)$ reverses sign in the initial guesses a and b which means $f(a)$ and $f(b)$ has different signs, and $f(x)$ is computable for all numbers between a and b;

For other cases, My_fzero will mostly succeed, but may also fail to find a place where $f(x)$ vanishes or reverses sign, possibly because no such place exists. When it happens that My_fzero cannot find a root, rather than searching forever, it will stop where it seems $|f(x)|$ has reached to a local minimum or constant, or it will stop and declare that no root has been found. An exit flag is returned to inform the user whether it finds a root, finds two successive representable points where the function reverse sign, find a local minimum of $|f(x)|$, or there is no root found. Based on the exit flag and the result, the user can determine what to do next.

Although My_fzero is quite robust, it doesn't necessarily mean My_fzero will never get wrong. As illustrated above, the computer can represent only discrete points in the real axis, thus the behavior of $f(x)$ besides these representable points is not going to be revealed by any root finder. For example, let

$$f(x) = 25x^2 - 20x + (4 + eps) \tag{4}$$

It can be seen that mathematically speaking, $f(x)$ has no real root. But due to the round-off problem, the machine might return 0.4 as the root of $f(x)$, which is clearly wrong.

Furthermore, even if the computer can represent any real number without round off, any root finder can still be fooled easily. In order to fool any root-finder, first let the root-finder to solve for zeros of $f(x) \equiv 1$, and record the finitely many points x_1, x_2, Λ, x_n at which $f(x)$ was calculated to reach the decision that $f(x)$ never vanishes. Then we can form a function:

$$f(x) = g(x) \cdot \prod_{i=1}^{n}(x - x_i) + 1 \tag{5}$$

where $g(x)$ can be any function with no pole at x_1, x_2, Λ, x_n.

Since both functions take exactly the same value 1 at every sample argument, the root finder must decide the same way for both: both equations $f(x) = 0$ have no real root. But that is obviously wrong.

How does My_fzero compare with other root finders?

There are many equation solvers in the world. However, most of them are hampered by one or more limitations listed below:

1. They may insist to be given two initial guesses where the function reverses sign. This requirement is sometimes hard to satisfy because the user usually doesn't know exactly what the property of the function is. My_fzero can take one or two initial guesses, when $f(x)$ is real and finite for at least one of them, My_fzero will try its best to find a root no matter the initial guess is near or far from the root. But better guess will help My_fzero to find a root faster.

2. They may ask for one or more stopping criteria in case they will run forever. These criteria includes maximum steps or maximum time they might iterate no matter they find a root or not, or a tolerance that they will stop and claim a root when the distance between either the two points where the function reverses sign or any two successive guesses is less than the tolerance. These criteria are also hard to be defined in advance properly, because of the unknown property of $f(x)$. Using the maximum iteration steps or time criterion, they may claim no root when their search permit expires after a long search, but just a few more steps before they would have found a root. When using a tolerance, they may still run forever because that tolerance can never be reached, or they may claim a "root" is found no matter how silly it is. Furthermore, the tolerance should be relative to the root itself. For example, a root 0.1 ± 1 is not as precise as a root 100 ± 1. My_fzero does not have these problems. It will stop and claim a root only when it has found a point when $f(x)$ vanishes, or found two successive machine representable points where $f(x)$ reverses sign. My_fzero can go on searching for a long time for some cases like $f(x) = 1/x$, but will not run forever.

3. They may insist to find a root, even there is not root at all. My_fzero will stop when it thinks it has found a local minimum of $|f(x)|$.

4. They may be unable to handle out of domain problem. If an arithmetic error occurs during the evaluation of $f(x)$ for some x, then $f(x)$ is not defined for that value of x and we say that x lies outside the domain of the definition of $f(x)$. Usually, they may stop and claim that there is no root, just because they encounter the out of domain problem. My_fzero can properly deal with that problem.

5. They may simply return a root if they find one, or claim no root if they cannot find one. There isn't much more information for the user. My_fzero will return an exit flag indicating why it stopped. The reason can be that it has found a root, or it has found two successive representable points where $f(x)$ reverses sign, or it has found a local minimum of $|f(x)|$, or it found not root, etc. This exit flag can help the user to decide what to do next, like re-write the function in another format to see whether it

will be easier to solve, try out different guesses, etc. The user can even write a program to deal with it automatically, based on the exit flag. My_fzero will also return the last two guesses and the function values for these two guesses.

6. They may restrict $f(x)$. My_fzero has no requirement of what $f(x)$ is. $f(x)$ can be a build in function, an inline function, or in general a user provided function.

How does My_fzero work?

The method My_fzero employed is a combination of Bisection Method, modified Secant Method, Quadratic Method, and Brent's Method, which itself is also a combination of several root finding methods including Bisection Method, False Position Method (it is also called Root Bracketing Method), and Inverse Quadratic Method.

Fig.1 below shows the three blocks of My_fzero. First, My_fzero analyze the input arguments and initialize some variables, which include the search bounds with lowerbound $= -\infty$ and upperbound $= \infty$. If it finds some syntax error, it will exit at that point. Then, My_fzero will try to obtain two valid initial guesses, which means the values of $f(x)$ at these two points are real and finite. The user of My_fzero can input one or two initial guesses. If the user provides two identical guesses, one of them will be discarded. If no guess the user provides is valid, My_fzero will display some message and exit. If the user provides two guesses, and they are all valid, the program will go on to the main loop for root finding. Otherwise, the user has provided only one valid guess, say, a. In this case, My_fzero will try to find another valid guess b before it can actually begin to find a root. It will search the right side first to see whether it can find b. If $a = 0$, set $b = 1/50$, otherwise, set $b = a + |a|/50$. In this way, My_fzero will have another rough guess not far away from a. If $f(x)$ is out of domain at b, then use Bisection Method, i.e., let $b = (a+b)/2$. Any time $f(x)$ is out of domain at b, the search bound will be reset upperbound $=$ b. This process will continue until a valid guess is found, or there isn't any other machine representable point between a and b. If the latter case happens, My_fzero will try to find a valid point in the left side of a. The process will be the same, except that b will be initially set to -1/50 or $a - |a|/50$, and that lowebound $=$ b if out of domain happens.

If My_fzero has searched both sides and found no second valid point, it will return with an *exitflag* indicating it cannot find a second valid point. Otherwise, it has successfully finished the initial guesses block. If My_fzero happens to meet a point where $f(x)$ vanishes, which in fact seldom happens, it will luckily return with the root. Otherwise, it goes on to the main loop.

Although this method for obtaining initial guesses is simple, it's quite robust. Since we do not have much information up to this stage, Bisection Method is a reasonable choice. Other root finders might have some problem when trying to obtain two initial guesses. For example, the Matlab root finder, fzero, can also take one or two initial guesses. If two

guesses are provided, it insists that the function $f(x)$ must reverse sign at these two points, which in fact make it almost as difficult as finding the root itself. If only one guess is provided, fzero will try both side of a alternatively, multiplying the distance between a and the guess by a fix constant which is larger than 1, until it finds one b that $f(a)$ and $f(b)$ have different signs, or it meets an out of domain problem. However, this method is not trustworthy. For example, when $f(x)$ is computable anywhere and always has the same sign, given whatever single guess, fzero will run for ever, because it can never find two points where $f(x)$ reverses sign.

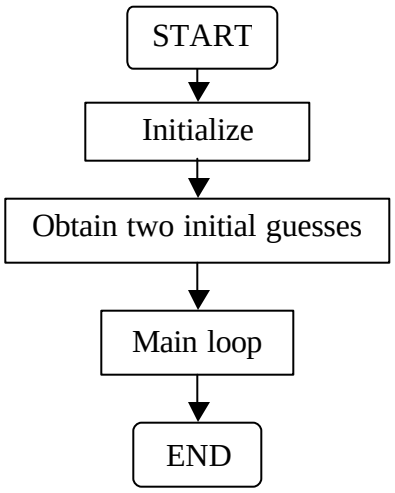

Fig.1 Diagram of My_fzero

Fig.2 shows the diagram for the main loop of My_fzero. For each loop, there must be two points a and b, and $f(b)$ is always the best value, which means $|f(b)| < |f(a)|$. There may also be a third point, depending on the previous result, which will be explained later. As the methods used for the case where $f(a)$ and $f(b)$ have the same sign and for the case where $f(a)$ and $f(b)$ have different signs are quite different, first of all, My_fzero will select a processing block based on this information. In each of the two blocks, My_fzero will try to find a proper next point c using the most suitable method, and calculate $f(c)$. When My_fzero reaches one of its exit conditions, (which means it successfully finds a root, or two successive machine representable points where $f(x)$ reverses sign, or a local minimum of $|f(x)|$, or it concludes that it can not find a root.) My_fzero will prepare the output arguments and then return within the main loop. If My_fzero has not reach any exit condition, it will record some information, and update the current guesses a and b, and also make sure that $|f(b)| < |f(a)|$.

Fig.3 and Fig.4 show the diagram of processing blocks for the same sign case and different signs case individually.

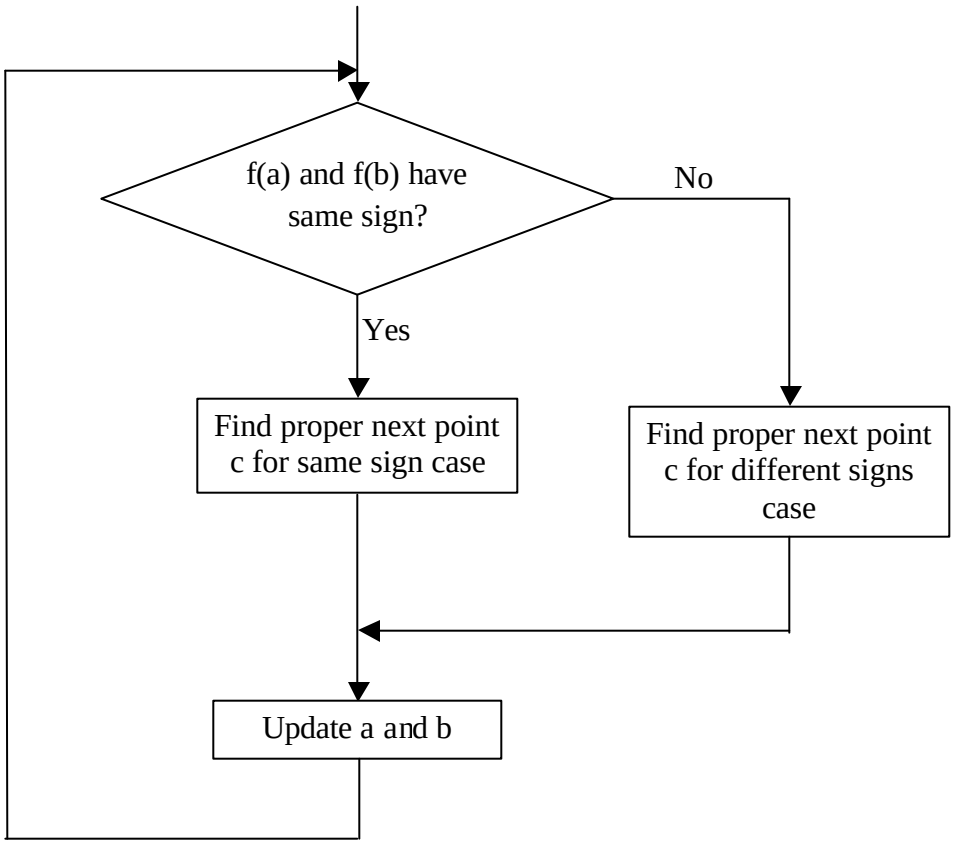

Fig.2 Diagram for the main loop

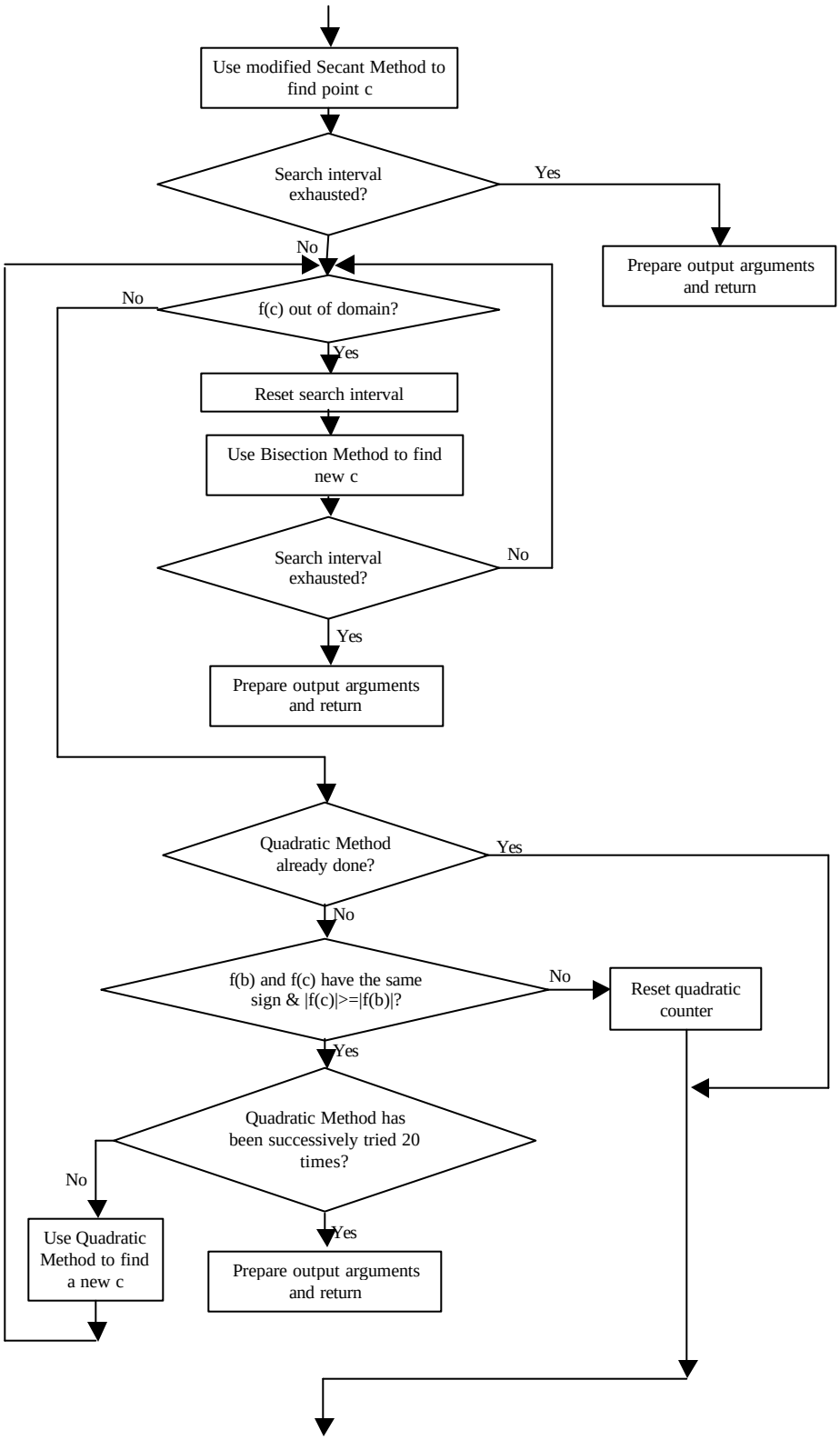

Fig.3 Diagram for the block dealing with the case where f(a) and f(b) have the same sign.

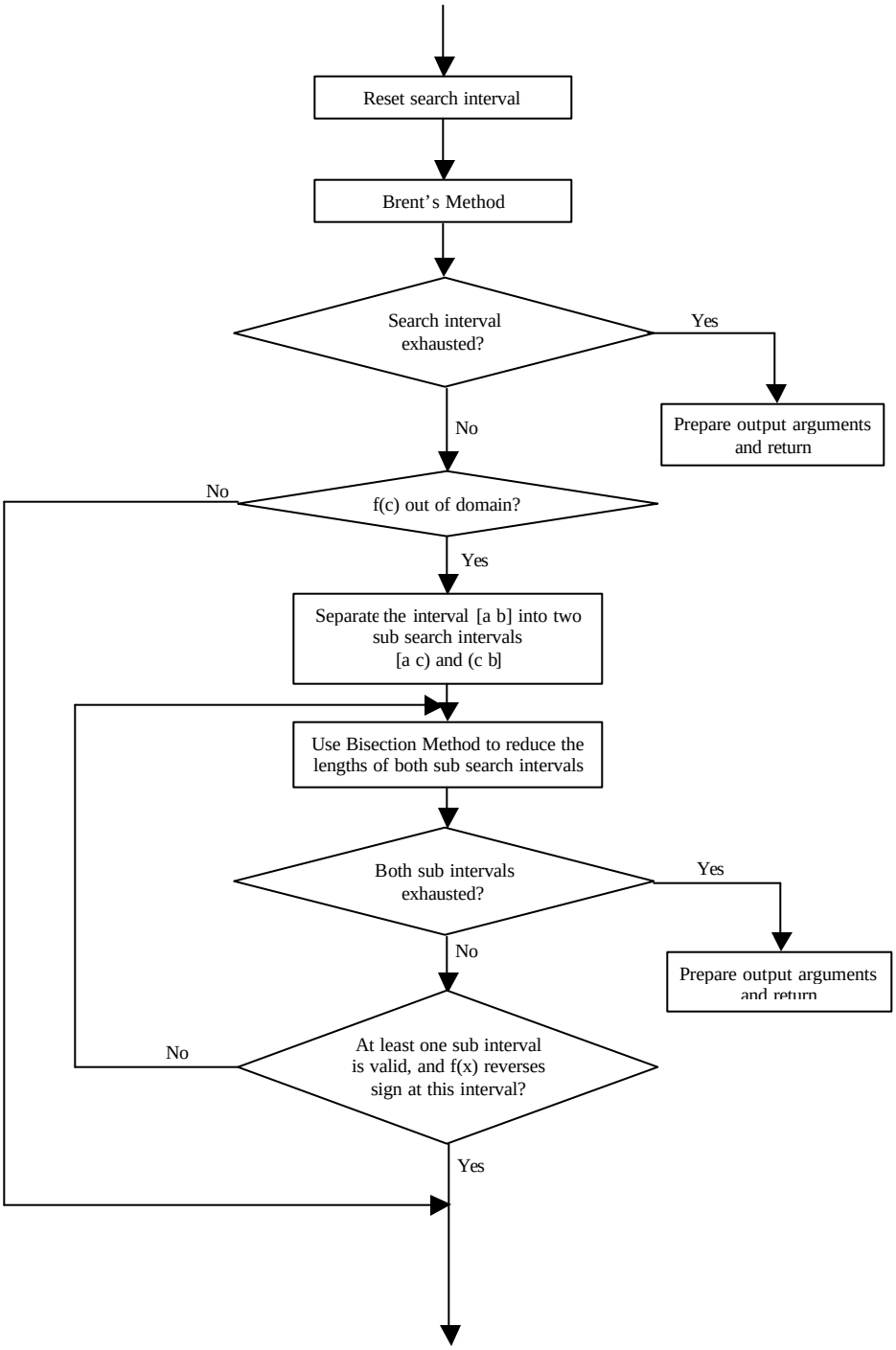

Fig.4 Diagram for the block dealing with the case where f(a) and f(b) have different signs.

- **Root finding method when f(a) and f(b) have the same sign**

The basic method used for the case where $f(a)$ and $f(b)$ have the same sign is modified Secant Method. Secant Method is a well-known powerful root finding method. Suppose we have the current two guesses a and b. As illustrated in Fig.5, the straight line cross $(a, f(a))$ and $(b, f(b))$ will usually cut the x-axis at $(c, 0)$. The value of c is given by the formula:

$$c = b - (b-a) \cdot f(b) / (f(b) - f(a)) \tag{6}$$

If a and b lie close enough to the root of $f(x)$ and the graph of $f(x)$ near the root is smooth enough, $f(x)$ is assumed to be approximately linear near the root, so we can see that c is much closer to the root than a and b. This means Secant Method can give a better approximation toward the root based on the current guesses. Furthermore, this approaching can be done in an iterative way, that is, each time let $a = b$ and $b = c$, and go on using Secant Method. In this way, the new guess will get closer and closer to the root until it finally reaches the root.

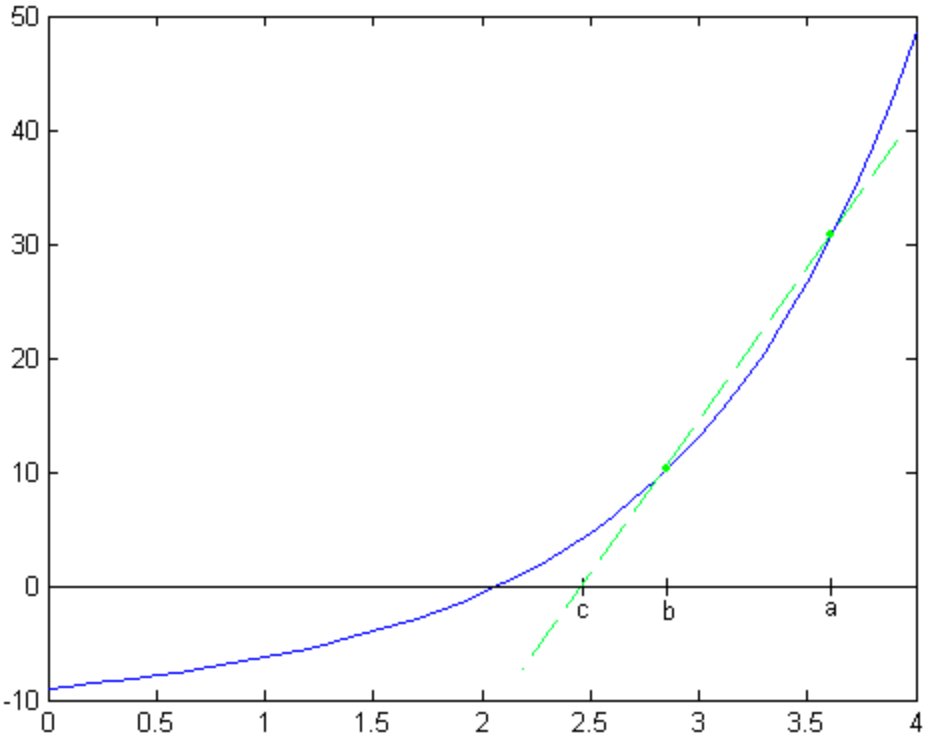

Fig.5 Secant Method

However, there are still some problems with Secant Method. When the difference between $f(a)$ and $f(b)$ is relatively small, the next point c may run far away from the

root, as illustrated in Fig.6. To avoid this problem, a constraint that c cannot run too far away at a time was added. That is, when $f(a) = f(b)$ or $|c-b| > 100 \cdot |b-a|$, let $|c-b| = 100 \cdot |b-a|$.

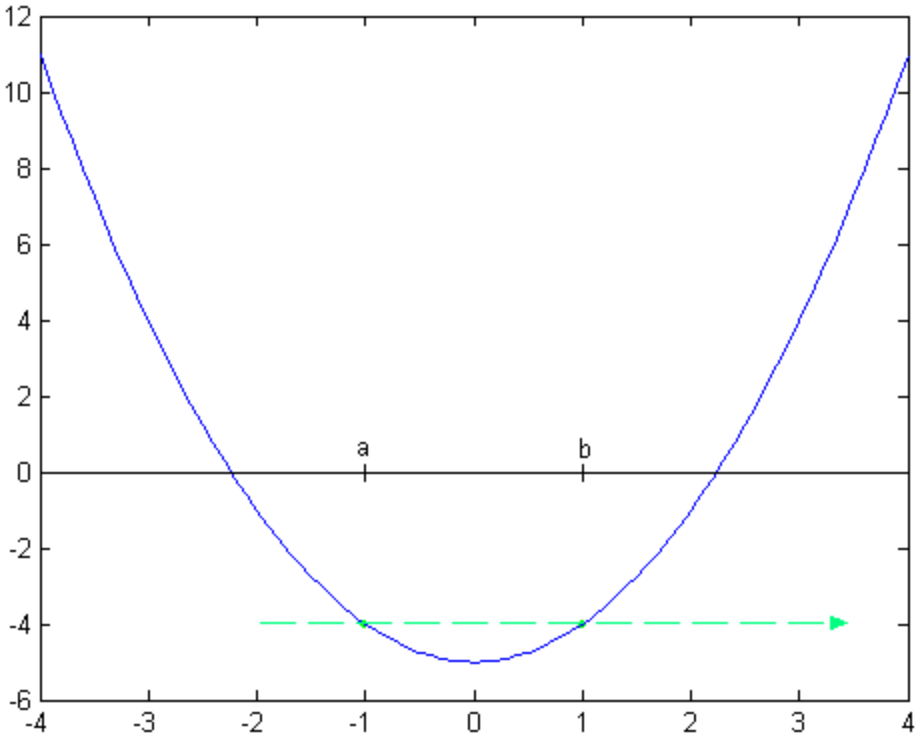

Fig.6 Secant Method when different between f(a) and f(b) is small

Another problem is that, when it happens that $|f(a)|$ is much larger than $|f(b)|$, as illustrated in Fig.7, round off problem may make c coincide with b. This may let Secant Method stop unsuccessfully. The modification used in My_fzero for this case is to let c be the next representable point of b.

To find the root, Secant Method requires a and b to be close enough to the root, otherwise it may possibly run away from the root, and even may never come back. But finding guesses close to the root usually can take a long time. After Secant Method finds good guesses near the root, usually it will converge to it very fast. Then when $f(x)$ becomes tiny, the rounding error becomes significant too. The relatively inaccurate $f(a)$ and $f(b)$ makes the quotient $f(b)/(f(b)-f(a))$ not very reliable. It's worth to point out that if $f(x)$ has some level of uncertainty, that is to say there is some "noise" when calculating $f(x)$, it's better for the user who defines $f(x)$ to make sure that $f(x)$ is forced to be 0 when the its absolute value is under the level of uncertainty. This is to make the root finding process more robust against noise.

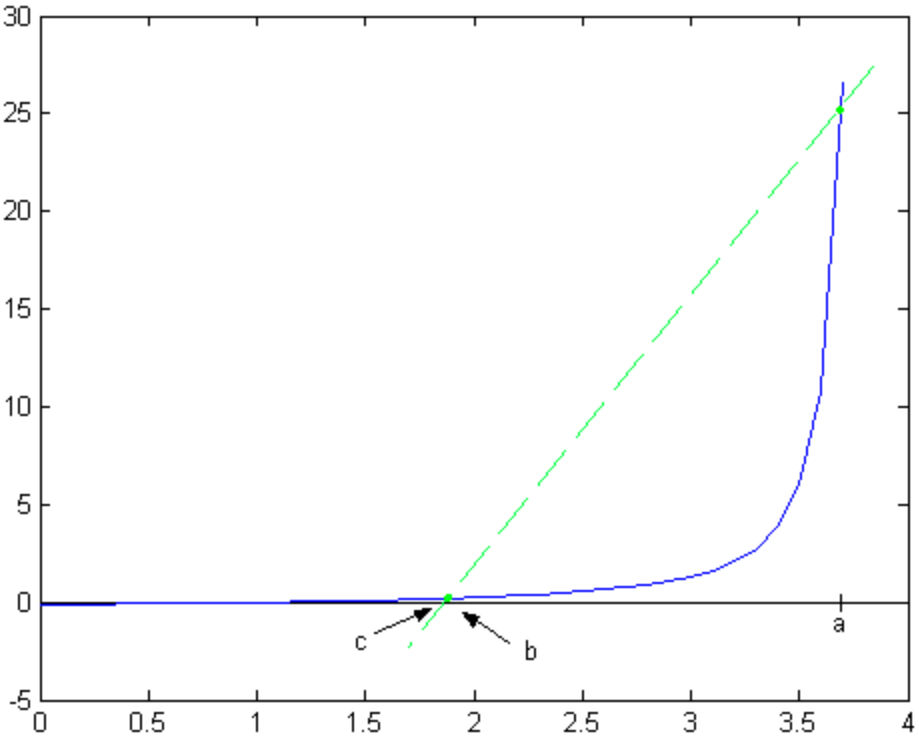

Fig.7 Round off problem makes Secant Method stop

Another thing worth to mention is that, when the two guesses are very near to the roots, $b-a$ and $f(b)$ are both so small that when calculating formula (6), $(b-a)\cdot f(b)$ may be first rounded off to 0, then $(b-a)\cdot f(b)/\big(f(b)-f(a)\big)$ will also be 0 not matter how small $f(b)-f(a)$ is. To avoid this problem, parentheses should be added to the formula as:

$$c = b-(b-a)\cdot\big(f(b)/\big(f(b)-f(a)\big)\big) \tag{7}$$

Sometimes Secant Method iteration cycles endlessly through estimations $a, b, c, d, a, b, c, d, \Lambda$, as illustrated in Fig.8 and Fig.9. My_fzero will not cycle like Fig.8, because when it finds two points where $f(x)$ reverses sign, it will give up Secant Method and begin to use Brent's Method. To avoid cycling like Fig.9, when My_fzero finds $|f(c)|$ is larger than $|f(b)|$, it uses Quadratic Method, trying to find a local minimum of $|f(x)|$. Here Quadratic Method means interpolating a quadratic through the three points $\big(a, f(a)\big)$, $\big(b, f(b)\big)$ and $\big(c, f(c)\big)$, and set d to the place where that quadratic's derivative vanishes. In effect, d is the highest or lowest point on the quadratic. Then, My_fzero will resume the Secant iteration using b and d as the next two guesses. At all times, b and $f(b)$ records the smallest $|f(x)|$ calculated so far.

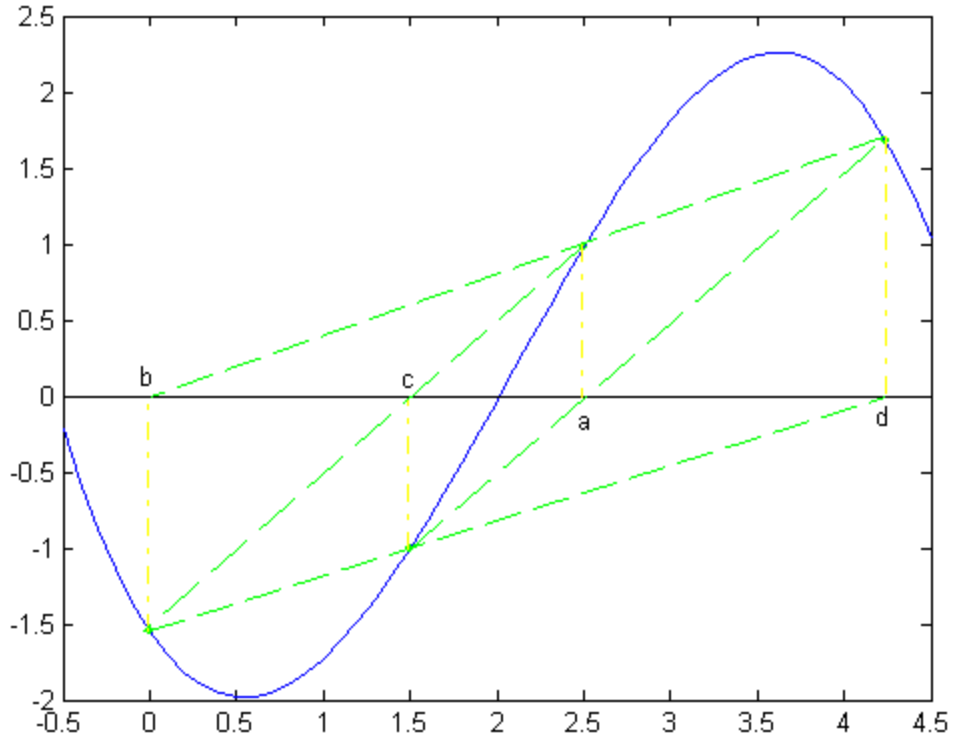

Fig.8 Cycling of Secant Method, (case 1)

Quadratic Method can keep Secant Method from cycling around a relatively shallow minimum of $|f(x)|$, as illustrated in Fig.10, because it will usually look elsewhere for the root. But Quadratic Method still cannot prevent the searching from trapped near relatively deep minimum of $|f(x)|$. To solve this, My_fzero does not insist to find a root. If $|f(b)|$ has not decreased for Q consecutive quadratic fits, My_fzero concludes that there is a local minimum of $|f(x)|$ at b, and stops searching. Q is a constant number. The larger Q, the larger chance My_fzero will avoid being trapped near a local minimum of $|f(x)|$, or the larger chance it will find a more precise local minimum. However, the calculation increases with larger Q. In My_fzero, Q is 20.

The whole process of a root finding iteration when $f(a)$ and $f(b)$ have the same sign is described below:

First, use Secant Method to find next guess c. If $|c-b| > 100 \cdot |b-a|$, let $c = b + 100 \cdot |b-a| \cdot sign(c-b)$. If c happens to coincide with b, let c be the next representable point of b. If c is out of search interval, let c be $(\mathrm{b} + \text{lowerbound})/2$ or $(\mathrm{b} + \text{upperbound})/2$, depending on to which side of the search interval c locates.

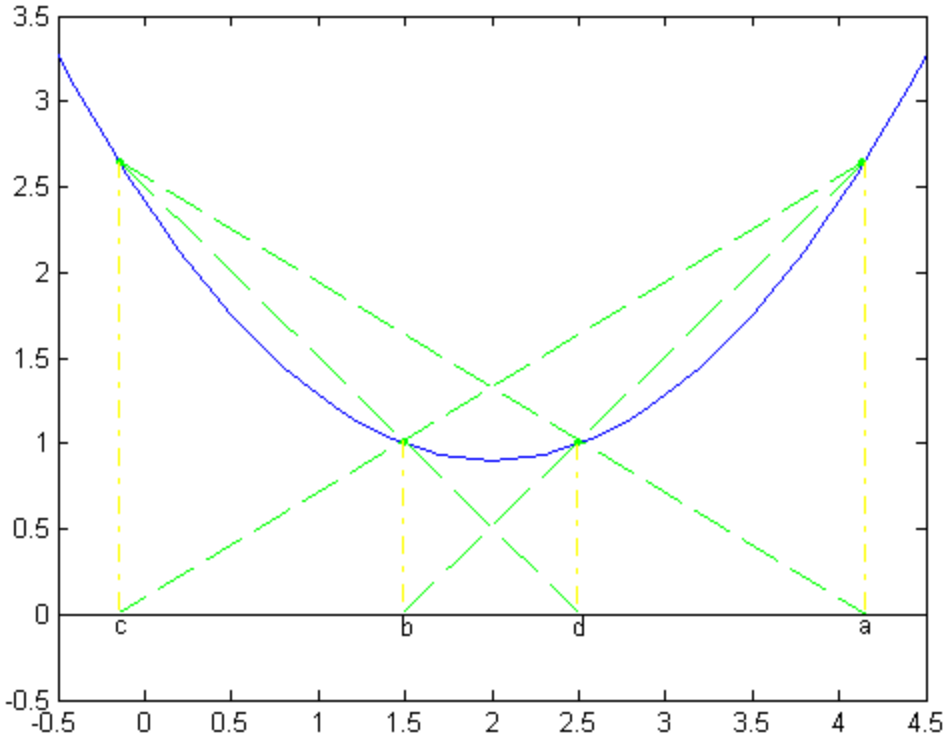

Fig.9 Cycling of Secant Method, (case 2)

Then, if the search interval has been exhausted, My_fzero meets its exit condition. It will decide whether that's because it cannot find a root, or because it has expanded the searching to infinite.

If the search interval has not been exhausted, My_fzero will calculate $f(c)$. If $f(c)$ is out of domain, Bisection Method will be used iteratively, reset the search interval and let $c=(b+c)/2$ in each iteration. This will not stop until the search interval has been exhausted, or $f(c)$ is within domain. If the search interval has been exhausted, My_fzero has failed to find a root. Otherwise, My_fzero has found a valid c, and it will go on the following steps.

If $f(c)$ and $f(b)$ have different signs, My_fzero will stop Secant Method and jump to the block which deal with the case that $f(x)$ reverses sign. Otherwise, My_fzero compares $|f(c)|$ and $|f(b)|$. If $|f(c)|<|f(b)|$, Secant Method has found a better approximation to the root, so it will go on to the next iteration. If $|f(c)|\geq|f(b)|$, Quadratic Method is used, and c is reset to the point where the derivation of the quadratic vanishes. Each time Quadratic Method is employed, a counter is decremented and tested. Each time Secant method finds $f(c)$ with decreasing magnitude, that counter

is reset to Q. When the decremented counter value is zero, My_fzero returns the last sample value as an approximate local minimum of $|f(x)|$. Otherwise, it will go on searching with the new guess c. However, this new c is still possible to be out of domain, so My_fzero will go back to the place where it calculate $f(c)$ and see whether $f(c)$ is out of domain. So there is a loop within each iteration, as illustrated in Fig.3. If $f(c)$ is within the domain, then no matter $|f(c)|$ is less than $|f(b)|$ or not, My_fzero will take it and go on to the next iteration.

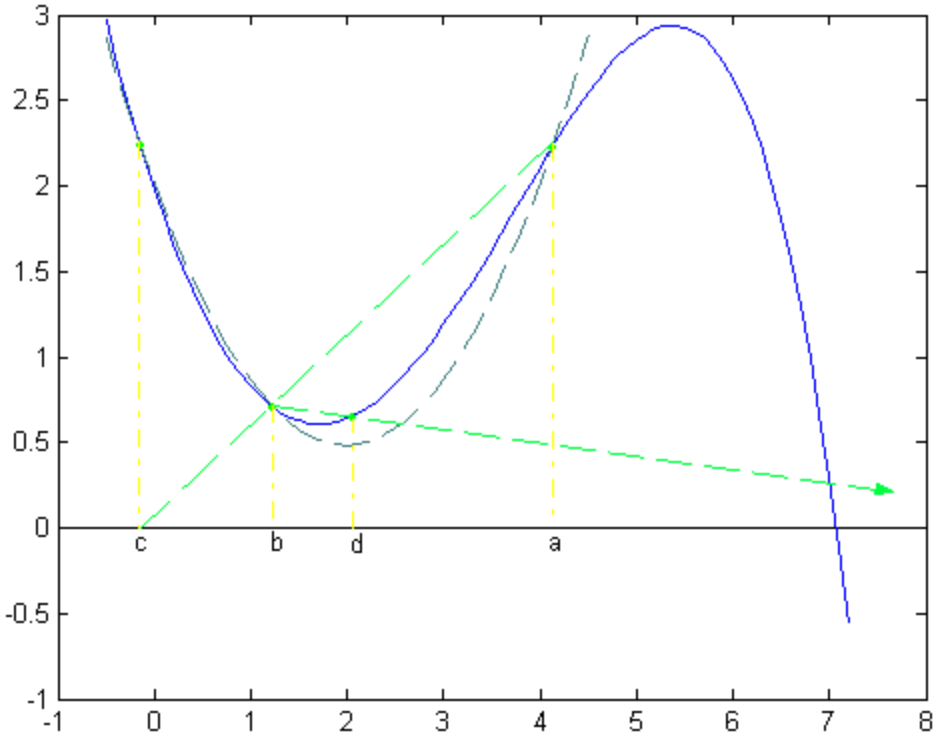

Fig.10 Quadratic Method

- **Root finding method when f(a) and f(b) have different signs**

The main method used in this block is Brent's Method, also known as van Wijngaarden-Deker-Brent Method.

Although Secant Method and other methods like False Position Method usually converge faster than Bisection Method, we still often find cases for which Bisection converges more rapidly. Bisection Method always halves the interval, while Secant Method and False Position Method can sometimes spend many iterations trying to get near to the root. A better way to do root finding is to combine these methods together. We can use a usually fast method while keeping track of whether it is actually converging or not. If not,

Bisection Method will be used as a back up to make sure the program at least converges linearly. Brent's Method is an excellent algorithm that works in the above way. It is a combination of False Position, Bisection, and Inverse Quadratic Method. False Position is similar to Secant Method, and it only differs in that False Position retains the known smallest interval brackets the root. Inverse Quadratic Method is similar to the Quadratic Method. It uses three points to fit an inverse quadratic function for which x is a quadratic function of y, as illustrated in Fig.11.

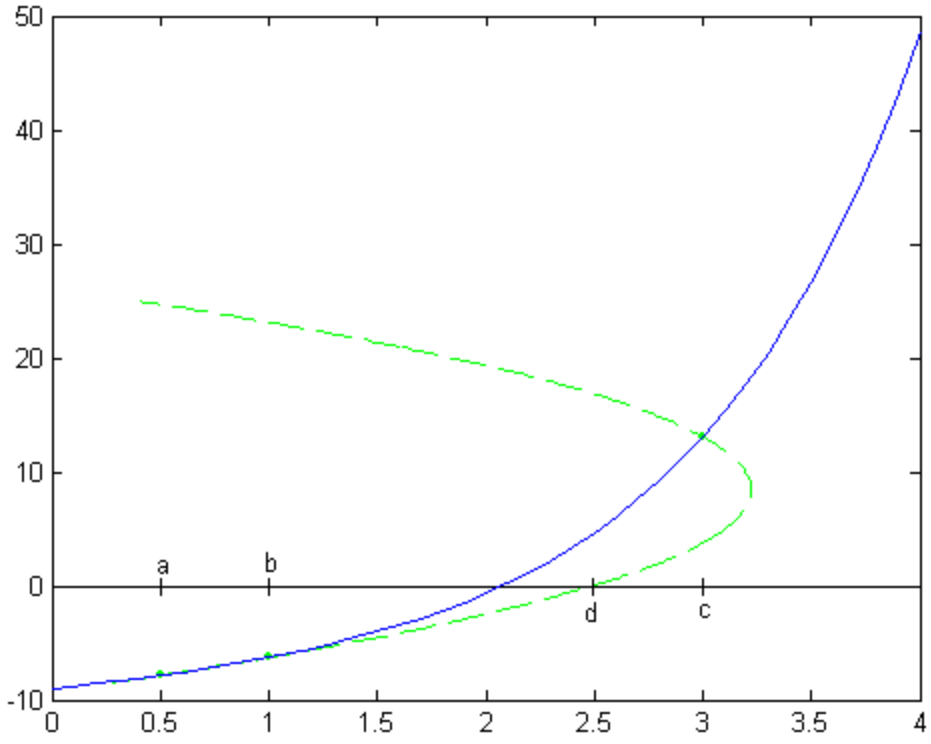

Fig.11 Inverse Quadratic Method

The function of the inverse quadratic can be written as:[3]

$$x = \frac{[y - f(a)]\cdot[y - f(b)]}{[f(c) - f(a)]\cdot[f(c) - f(b)]}\cdot c + \frac{[y - f(b)]\cdot[y - f(c)]}{[f(a) - f(b)]\cdot[f(a) - f(c)]}\cdot a + \frac{[y - f(c)]\cdot[y - f(a)]}{[f(b) - f(c)]\cdot[f(b) - f(a)]}\cdot b \qquad (8)$$

The point where the inverse quadratic crosses x-axis is taken as the next guess. Setting y to zero gives this point, which can be written as:

[3] William H. Press, "Numerical Recipes – the Art of Scientific Computing," Page 252, Cambridge University Press 1986

$$x = b + P/Q \tag{9}$$

Where, set:

$$R = f(b)/f(c), \qquad S = f(b)/f(a), \qquad T = f(a)/f(c) \tag{10}$$

And:

$$P = S \cdot [T \cdot (R - T) \cdot (c - b) - (1 - R) \cdot (b - a)] \tag{11}$$

$$Q = (T - 1) \cdot (R - 1) \cdot (S - 1) \tag{12}$$

And b and $f(b)$ records the smallest $|f(x)|$ calculated so far. Usually, P/Q ought to be a small correction. However, Inverse Quadratic method is not very stable. It will give very bad guess. Brent's Method avoids this problem by maintaining brackets on the root and checking whether the new guess calculated using Inverse Quadratic Method or False Position Method is within the brackets and is converging rapidly enough. If not, it will use Bisection Method instead. Thus, Brent's Method combines the robustness of Bisection Method with the higher speed of Inverse Quadratic Method and False Position Method.

The whole process of a root finding iteration when $f(a)$ and $f(b)$ have different signs is described below:

First, reset the search interval which is known the smallest interval which brackets the root. That is, $lowerbound = \min(a, b)$ and $upperbound = \max(a, b)$. Then, if $f(a)$ or $f(b)$ is infinite, use Bisection Method, otherwise use Brent's Method to find a new guess c. Brent's Method will choose one out of the three methods: Bisection Method, False Position Method and Inverse Quadratic Method. If the search interval has been exhausted, My_fzero has successfully found two successive machine representable points which bracket the root. Otherwise, My_fzero calculate $f(c)$ and see whether $f(c)$ is out of domain. If not, My_fzero will go on to the next iteration. If $f(c)$ is out of domain, the search interval $[lowerbound\ upperbound]$ is divided into two sub intervals $[lowerbound\ g_1)$ and $(g_2\ upperbound]$, where $g_1 = g_2 = c$ initially. Then, these two sub intervals are halved alternatively and iteratively using Bisection Method. In each time, let $c = (lowerbound + g_1)/2$ first. If $f(c)$ is out of domain, then let $g_1 = c$. Otherwise, if $f(c)$ and $f(lowerbound)$ have the same sign, let $lowerbound = c$. Then the right sub interval is processed in the same way. This loop will continue until both sub intervals are exhausted; or My_fzero finds a sub interval which brackets the root. If the intervals are exhausted, My_fzero fails to find any root. If it's the latter case, the sub interval which brackets the root will be kept, while the other sub interval will be discarded. Then My_fzero will resume root finding.

Examples and Comparison

In order to test the performance of My_fzero, some examples are given in this chapter. Also, several other methods are used as comparison. These methods (or program) include fzero, Bisection Method, Secant Method, False Position Method and Brent's Method. Because fzero uses Brent's Method after it finds two points where $f(x)$ reverses sign, the result of Brent's Method is the same as fzero if two initial guesses are provided. Consequently, it is not necessary to list Brent's Method in the tables. The program of fzero requires a tolerance, which is *eps* in default.

The numbers listed in these tables may be cut short if they are too long to be put in the tables. Also, My_fzero may output an interval that brackets the root, so there may be two values listed in the "X" and "fval" row. If a method is not applied, it will be left empty.

1. $f(x) = \exp(x) + x - 2$

The function is shown in Fig.12.

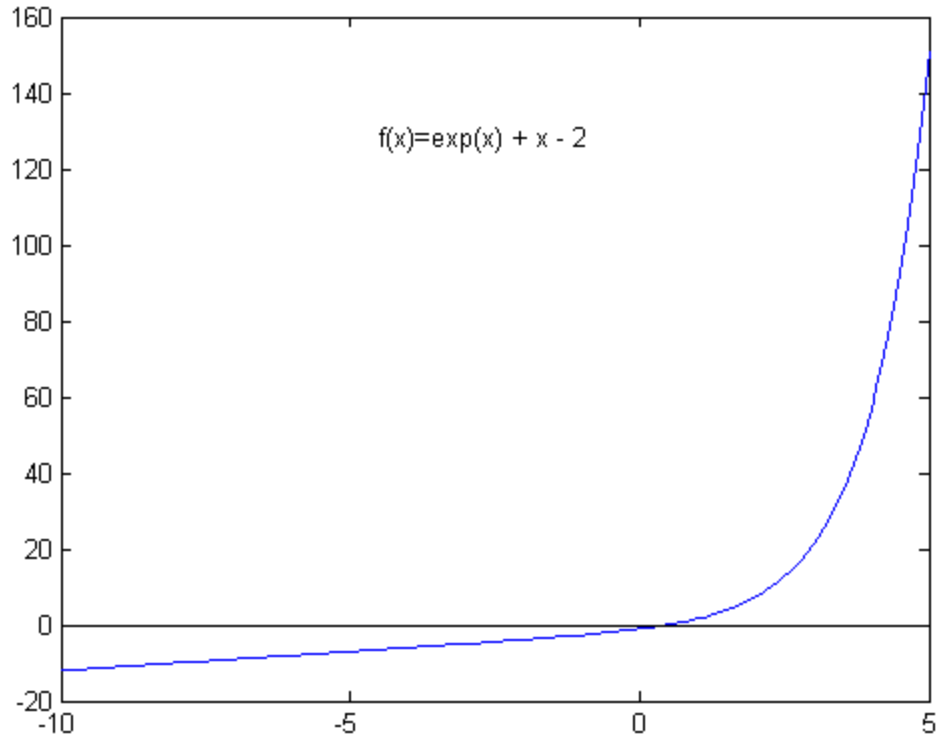

Fig.12 $f(x) = \exp(x) + x - 2$

The result is shown in Tab.1.

Initial_Guess(es) / Method		-5	4	-5, 4	-5, -3
My_fzero	X	0.44285440100239	0.44285440100239	0.44285440100239	0.44285440100239
	fval	0	0	0	0
	steps	11	13	11	12
fzero	X	0.44285440100239	0.44285440100239	0.44285440100239	
	fval	0	0	0	
	steps	33	27	11	
Secant	X			0.44285440100239	0.44285440100239
	fval			0	0
	steps			14	12
Bisection	X			0.44285440100239	
	fval			0	
	steps			58	
False Position	X			0.44285440100239	
	fval			-3.1086244689e-015	
	steps			207	

Tab.1 Comparison for $f(x) = \exp(x) + x - 2$

2. $f(x) = \exp(x) - 4x - (4 - 4\ln 4)$

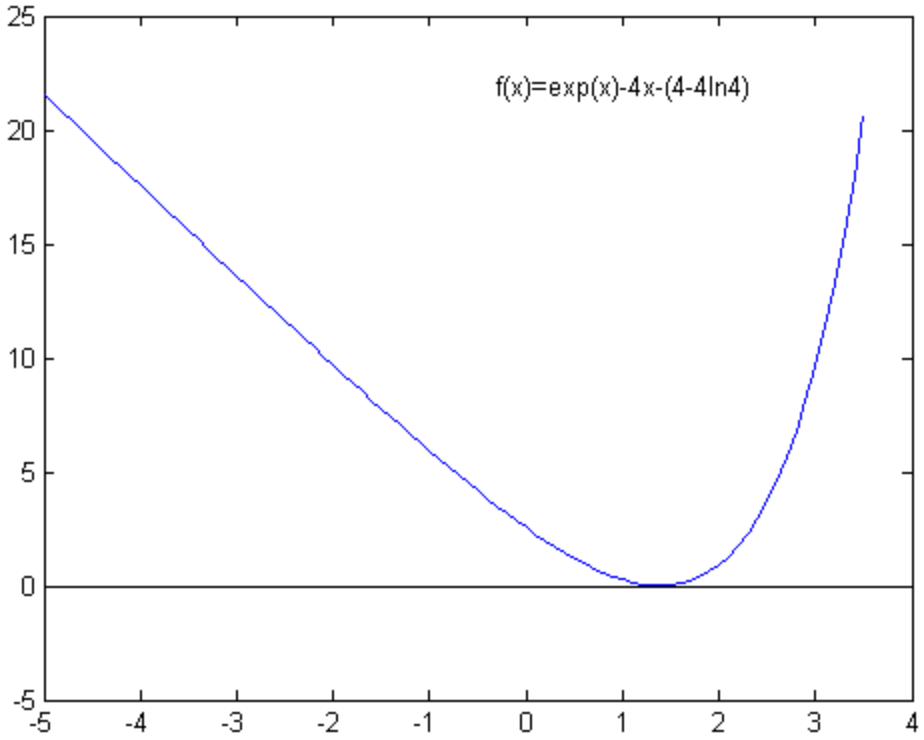

Fig.13 $f(x) = \exp(x) - 4x - (4 - 4\ln 4)$

Method \ Initial_Guess(es)		-3	2	-3, 2	-3,-2
My_fzero	X	1.38629435046769	1.38629437556774	1.38629435316725	1.38629434784790
	fval	0	0	0	0
	steps	41	39	40	40
fzero	X	NaN	NaN		
	fval	NaN	NaN		
	steps	55 (Failed)	58 (Failed)		
Secant	X			1.38629437500115	1.38629434784790
	fval			0	0
	steps			40	40

Tab.2 Comparison for $f(x) = \exp(x) - 4x - (4 - 4\ln 4)$

3. $f(x) = \exp(x) - 20x + 90$

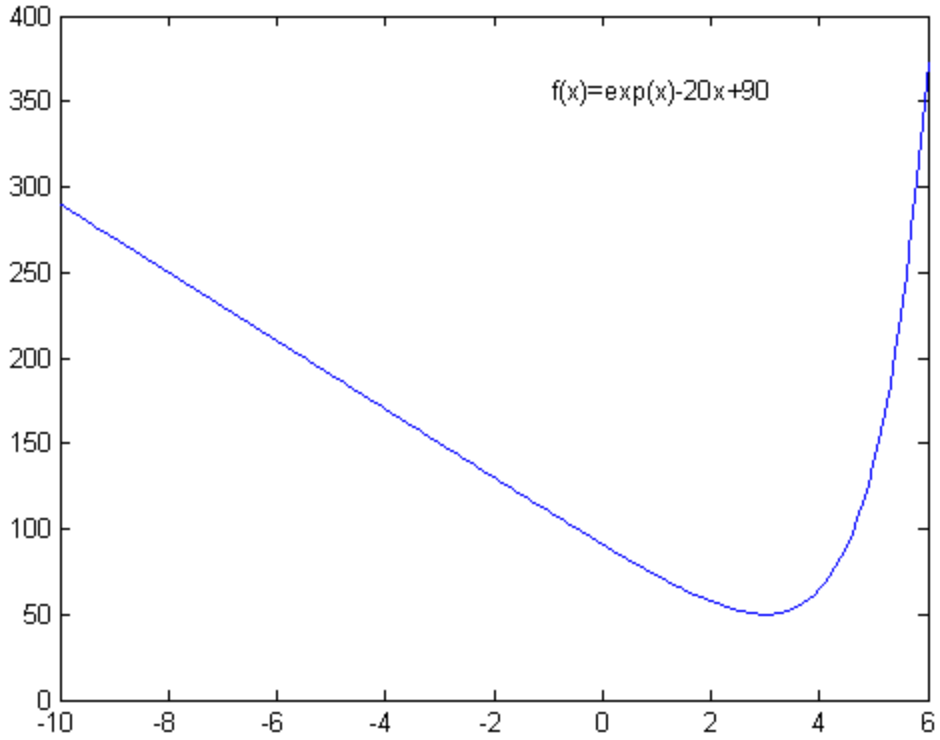

Fig.14 $f(x) = \exp(x) - 20x + 90$

Initial_Guess(es) / Method		-4	5	-6, -4	-2, 5
My_fzero	X	2.99380152969407	2.99452197374669	2.99573192325833	2.99175808342445
	fval	50.08539178265905	50.08536917126861	50.08535452892141	50.08551226176941
	steps	42 (min)	43 (min)	47 (min)	44 (min)
fzero	X	NaN	NaN		
	fval	NaN	NaN		
	steps	53 (Failed)	52 (Failed)		
Secant	X			-4.73465295951742	4.74423859164874
	fval			184.7018446875744	110.0354990955713
	steps			33 (Wrong)	178 (Wrong)

Tab.3 Comparison for $f(x) = \exp(x) - 20x + 90$

4. $f(x) = \exp(6x - x^4 - 1) - 1$

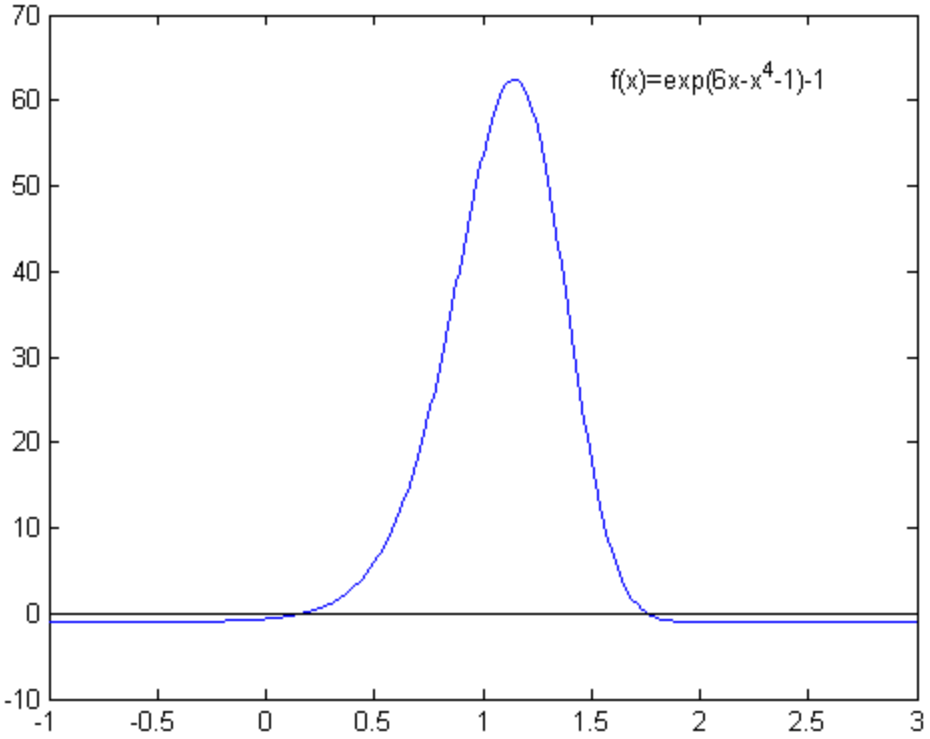

Fig.15 $f(x) = \exp(6x - x^4 - 1) - 1$

Initial_Guess(es) / Method		-0.5	1	3	1, 3
My_fzero	X	0.16679566609859	0.16679566609859	3.06000000000000	1.75777201824726
	fval	0	0	-1	0
	steps	12	13	41 (Min)	14
fzero	X	0.16679566609859	0.16679566609859	1.75777201824726	1.75777201824726
	fval	8.8817841970e-016	0	0	-1.776356839e-015
	steps	36	29	27	16
Secant	X				-Inf
	fval				-1
	steps				4
Bisection	X				1.7577720182472
	fval				0
	steps				54
False Position	X				1.75777201824726

	fval				-5.329070518e-015
	steps				183

Tab.4 Comparison for $f(x) = \exp(6x - x^4 - 1) - 1$

5. $f(x) = \ln(6x - x^4)$

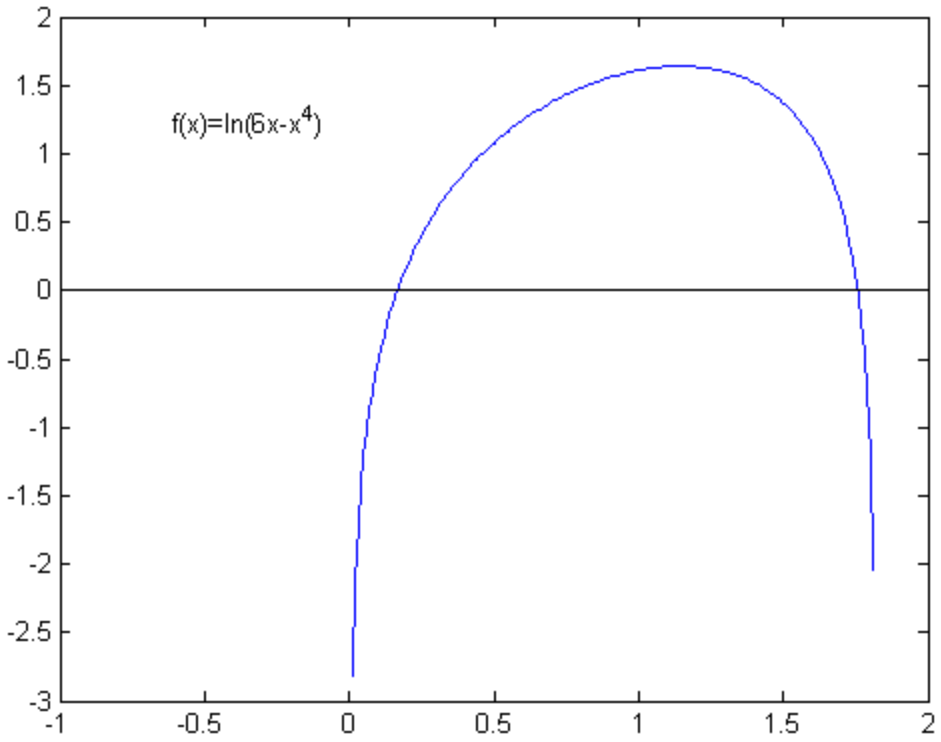

Fig.16 $f(x) = \ln(6x - x^4)$

Initial_Guess(es) / Method		1	0.1, 1	0.5, 1	0.1, 1.8
My_fzero	X	0.16679566609859	0.16679566609859	0.16679566609859	0.16679566609859
	fval	0	0	0	0
	steps	13	10	12	16
fzero	X	0.16679566609859	0.16679566609859		
	fval	0	0		
	steps	30	10		
Secant	X		NaN	NaN	NaN
	fval		NaN	NaN	NaN

	steps		4 (Failed)	3 (Failed)	3 (Failed)
Bisection	X		0.16679566609859		
	fval		0		
	steps		56		
False Position	X		0.16679566609859		
	fval		0		
	steps		28		

Tab.5 Comparison for $f(x) = \ln(6x - x^4)$

6. $f(x) = (x-1)^2 - 1$

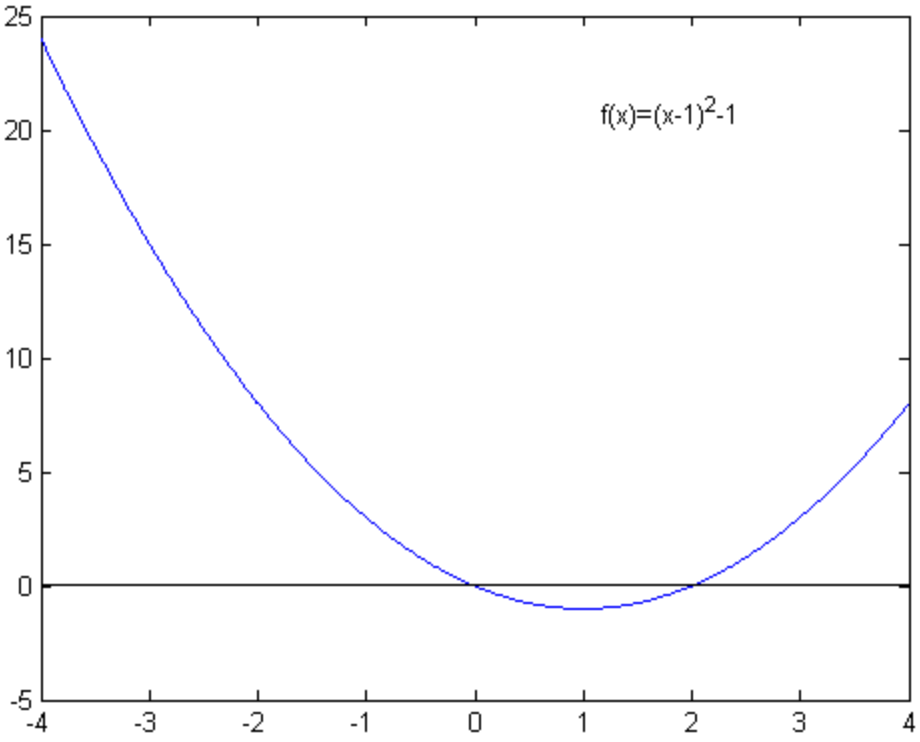

Fig.17 $f(x) = (x-1)^2 - 1$

Initial_Guess(es) / Method		-1	1	-2, 1	1, 4
My_fzero	X	-9.073655918e-017	2	3.5343903075e-017	2
	fval	0	0	0	0
	steps	9	9	11	11
fzero	X	-1.538729133e-017	-1.258661437e-017	3.5568126759e-017	2
	fval	0	0	0	0

	steps	31	32	11	11
Secant	X			-7.073198405e-018	2
	fval			0	0
	steps			13	11
Bisection	X			-1.110223024e-016	2
	fval			0	0
	steps			55	55
False Position	X			3.7088953354e-018	2.00000000000000
	fval			0	-8.881784197e-016
	steps			57	54

Tab.6 Comparison for $f(x) = (x-1)^2 - 1$

7. $f(x) = \sin(2p \cdot \exp(-x^2) + 0.1)$

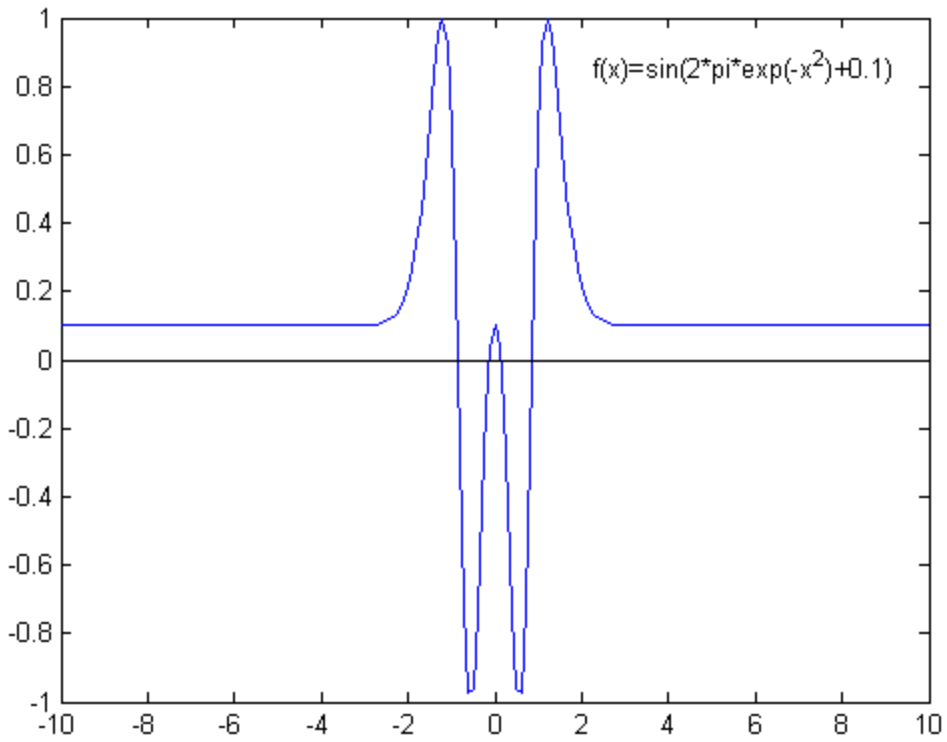

Fig.18 $f(x) = \sin(2p \cdot \exp(-x^2) + 0.1)$

Initial_Guess(es) / Method		-9	-0.5	-6, 0.5	0, 2
My_fzero	X	-8.82000000000000	0.12666296163542,	-0.85176040584, -	0.12666296163542,

			0.12666296163542	0.85176040584	0.12666296163542
	fval	0.09983341664683	-0.24492935e-15, 0.643249059e-15	0.1224646799e-15, -0.3216245299e-15	-0.2449293e-15, 0.6432490e-15
	steps	41 (Min)	18	12	19
fzero	X		-0.12666296163542	-0.85176040584857	
	fval		-2.44929359e-016	1.2246467991e-016	
	steps	Never Stop	31	14	
Secant	X				
	fval				
	steps			8 (Failed)	7 (Failed)
Bisection	X			-0.8517604058485, -0.8517604058485	
	fval			0.1224646799e-15, -0.3216245299e-15	
	steps			58	
False Position	X			-0.85176040584857	
	fval			1.224646799e-016	
	steps			21	

Tab.7 Comparison for $f(x) = \sin(2p \cdot \exp(-x^2) + 0.1)$

8. $f(x) = |x| \cdot \exp(-|x|) + 0.05$

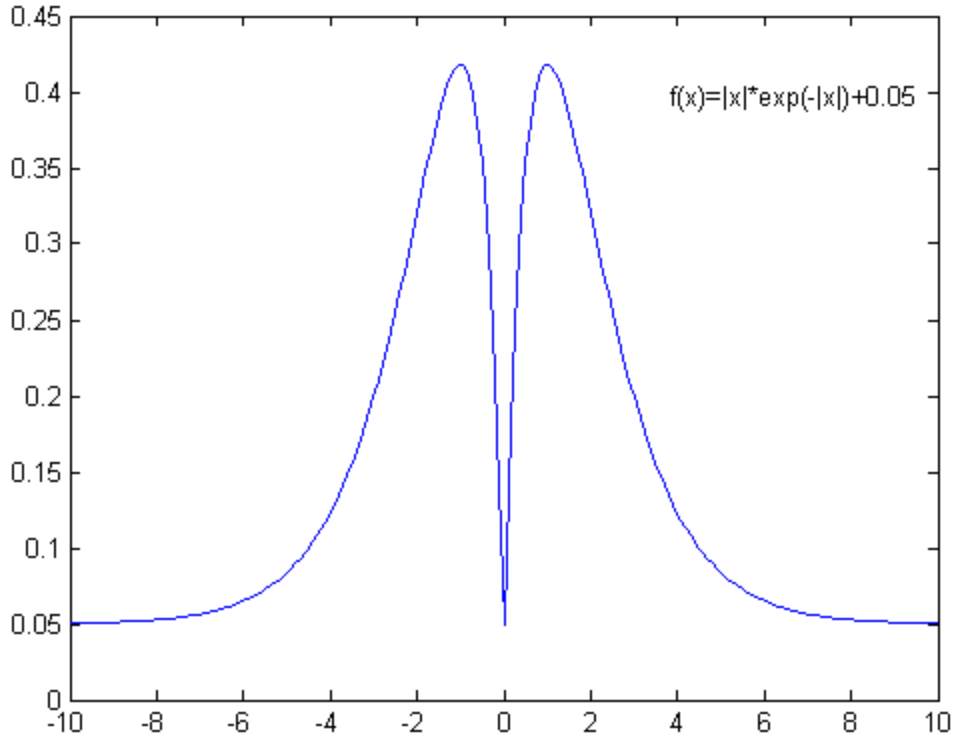

Fig.19 $f(x) = |x| \cdot \exp(-|x|) + 0.05$

Initial_Guess(es) / Method		-4	-0.2	-4, -0.2	-0.2, -0.1
My_fzero	X	-315.704551030901	0.00019712926687	-212.35589595391	148.92040406551
	fval	0.05000000000000	0.05019709041075	0.05000000000000	0.05000000000000
	steps	45 (Min)	50 (Min)	44 (Min)	44 (Min)
fzero	X	NaN	NaN		
	fval	NaN	NaN		
	steps	4111 (Failed)	4127 (Failed)		
Secant	X			NaN	NaN
	fval			NaN	NaN
	steps			7 (Failed)	7 (Failed)

Tab.8 Comparison for $f(x) = |x| \cdot \exp(-|x|) + 0.05$

9. $f(x) = 1/\sin(x)$

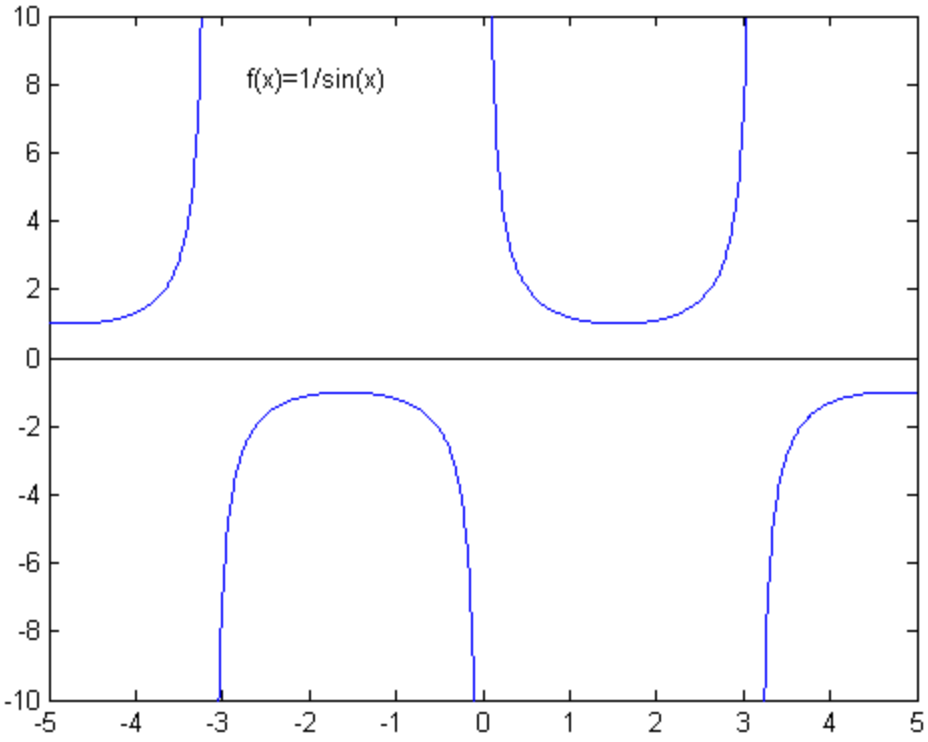

Fig.20 $f(x) = 1/\sin(x)$

Initial_Guess(es) / Method		-2	3.5	-3,-0.2	2, 4
My_fzero	X	0, 4.940656458e-324	6.28318530717959, 6.28318530717959	6.28318530717959, 6.28318530717959	3.14159265358979, 3.14159265358979
	fval	-Inf,Inf	0.6529855401e+15, -4.0828098382e+15	1.5546077909e+15, -4.0828098382e+15	-3.109215581e+15, 8.1656196765e+15
	steps	1371	73	69	67
fzero	X	-3.14159265358979	3.14159265358979		3.14159265358980
	fval	-6.87411693e+014	-6.0463434e+014		-4.7664865e+014
	steps	86	71		67
Secant	X			7.15444238e+015	1.35452182e+029
	fval			-1.42857839908636	1.89740919371289
	steps			638 (Wrong)	974 (Wrong)
Bisection	X				3.14159265358979, 3.14159265358979

	fval				8.1656196765e+15, -3.109215581e+15
	steps				54
False Position	X				3.14159265358979
	fval				-1.30597108e+015
	steps				123

Tab.9 Comparison for $f(x) = 1/\sin(x)$

10. $f(x) = sign(x) \cdot \left(\exp(-x^2) \cdot \sqrt{(x-1)\cdot(x+1)} + 0.002\right)$

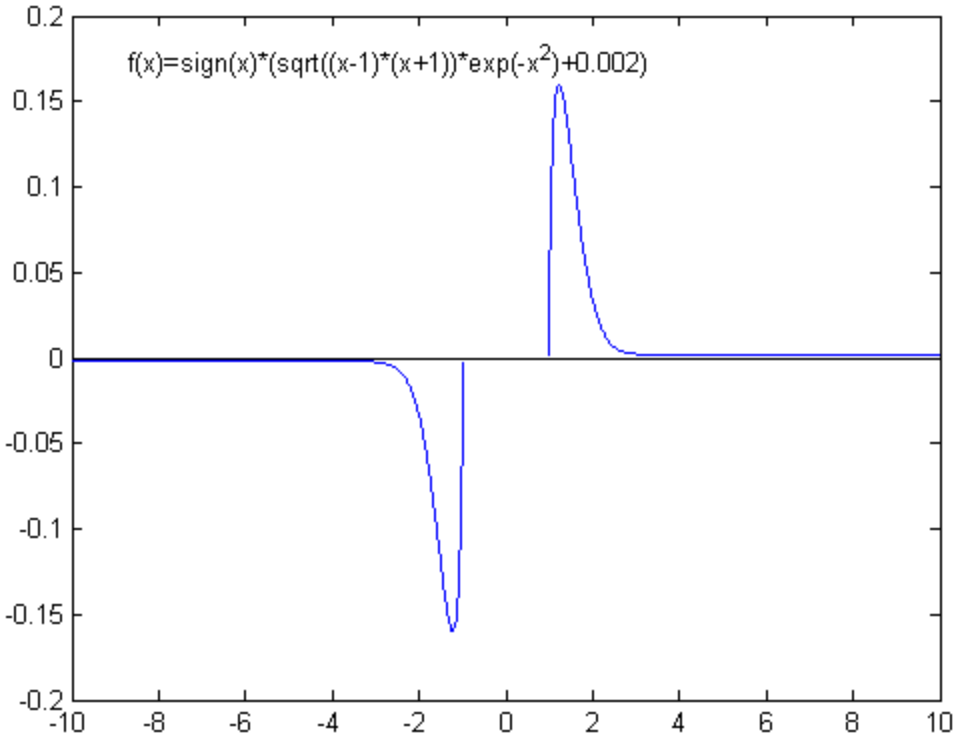

Fig.21 $f(x) = sign(x) \cdot \left(\exp(-x^2) \cdot \sqrt{(x-1)\cdot(x+1)} + 0.002\right)$

Initial_Guess(es) / Method	-2	-8	-2, -8	-2, 3

My_fzero	X	-1, 1	-1, 1	-1, 1	-1, 1
	fval	-0.00200000000, 0.00200000000	-0.00200000000, 0.00200000000	-0.00200000000, 0.00200000000	-0.00200000000, 0.00200000000
	steps	140	116	117	112
fzero	X	NaN	NaN		-0.529 - 0.034i
	fval	NaN	NaN		-0.000 - 0.642i
	steps	19 (Failed)	21 (Failed)		57 (Wrong)
Secant	X			NaN	NaN
	fval			NaN	NaN
	steps			4 (Failed)	8 (Failed)
Bisection	X				NaN
	fval				NaN
	steps				3 (Failed)
False Position	X				NaN
	fval				NaN
	steps				5 (Failed)

Tab.10 Comparison for $f(x) = sign(x) \cdot \left(\exp(-x^2) \cdot \sqrt{(x-1) \cdot (x+1)} + 0.002\right)$

11. $f(x) = \exp(x^2) - (1 + eps)$

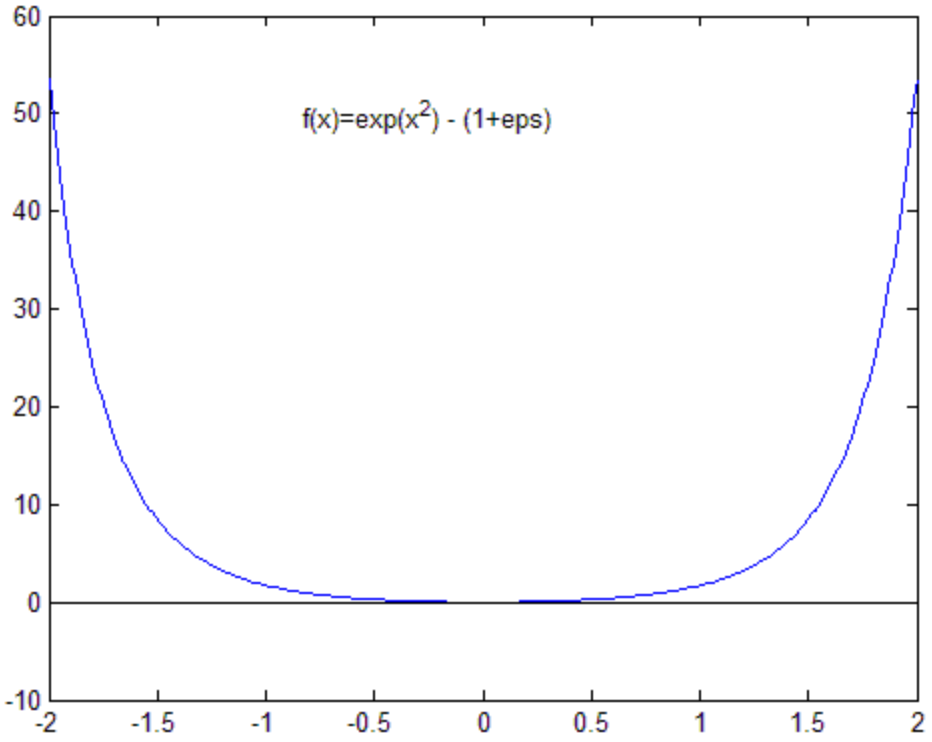

Fig.22 $f(x) = \exp(x^2) - (1+eps)$

Method \ Initial_Guess(es)		1	10	-4, 5	26, 26.64
My_fzero	X	1.648771146e-008	1.48799085e-008	-1.58398597e-008	1.61566267e-008
	fval	0	0	0	0
	steps	41	187	66	1024
fzero	X	NaN	NaN		
	fval	NaN	NaN		
	steps	41 (Failed)	26 (Failed)		
Secant	X			NaN	26
	fval			NaN	3.82886246e+293
	steps			68 (Failed)	2 (Wrong)

Tab.11 Comparison for $f(x) = \exp(x^2) - (1+eps)$

12. $f(x) = \sqrt{x} - 4$

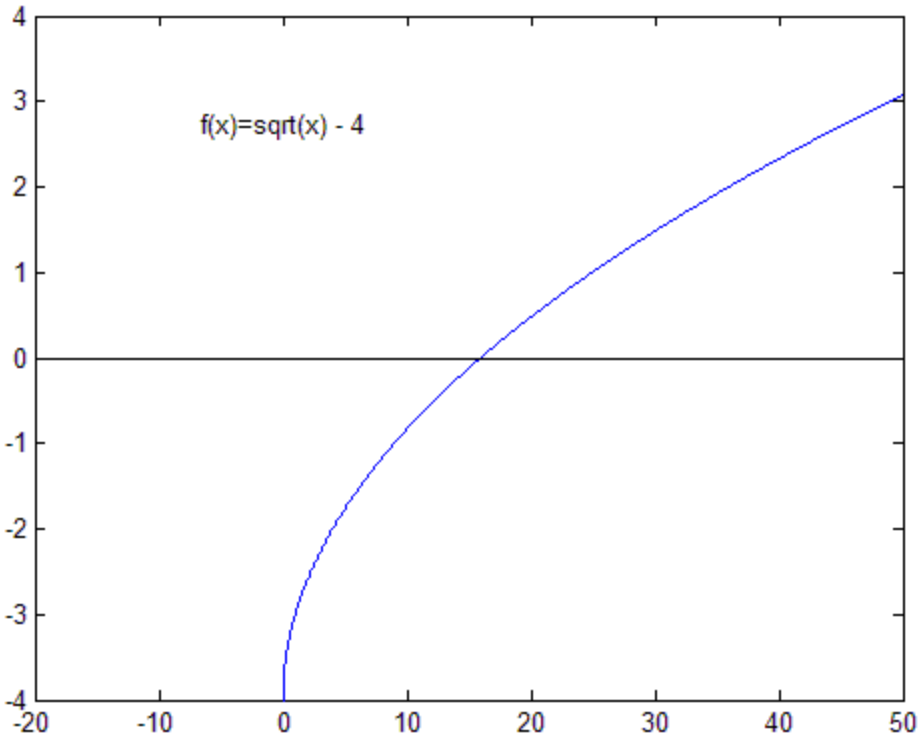

Fig.23 $f(x) = \sqrt{x} - 4$

Initial_Guess(es) / Method		0.5	777	0.5, 40	20, 30
My_fzero	X	16	16.00000000000000	16	16.00000000000000
	fval	0	0	0	0
	steps	11	13	4	8
fzero	X	NaN	NaN	16.00000000000000	
	fval	NaN	NaN	0	
	steps	23 (Failed)	23 (Failed)	4	
Secant	X			16.00000000000000	16.00000000000000
	fval			0	0
	steps			9	8
Bisection	X			16	
	fval			0	
	steps			55	
False Position	X			16.00000000000000	
	fval			0	

	steps			43	

Tab.12 Comparison for $f(x)=\sqrt{x}-4$

13. $f(x)=x\cdot\left(0.01-\sqrt{x^2-1}\right)$

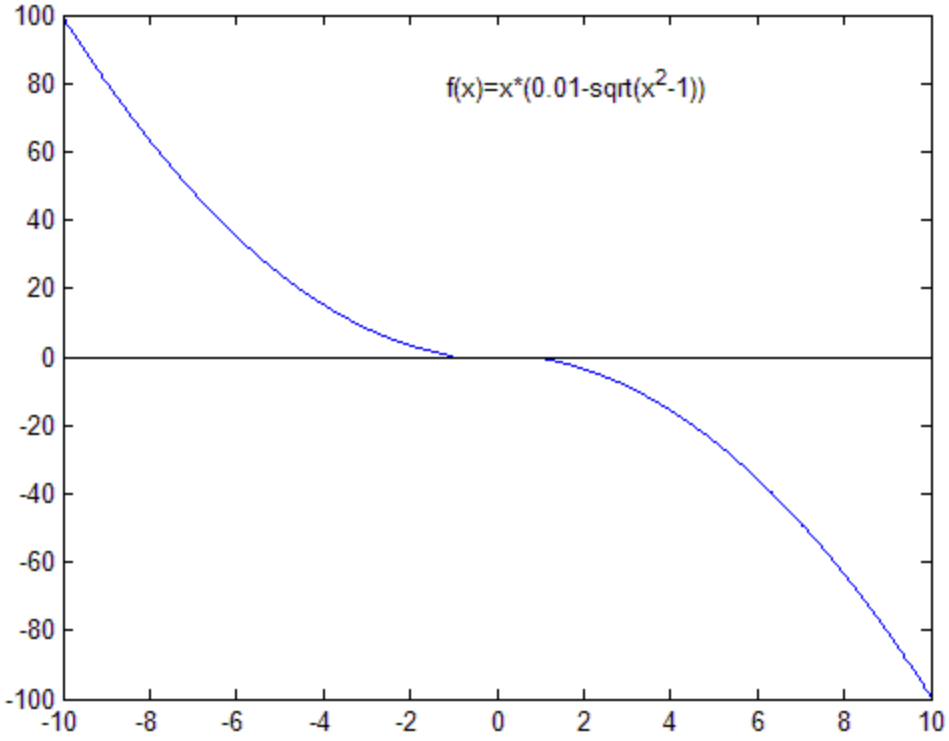

Fig.24 $f(x)=x\cdot\left(0.01-\sqrt{x^2-1}\right)$

Initial_Guess(es) / Method		-8	-50	-50, -8	-11, 10
My_fzero	X	-1.0000499987500, -1.0000499987500	-1.0000499987500, -1.0000499987500	-1.0000499987500, -1.0000499987500	-1.0000499987500, -1.0000499987500
	fval	0.105511156e-13, -0.116544549e-13	0.105511156e-13, -0.116544549e-13	0.105511156e-13, -0.116544549e-13	0.105511156e-13, -0.116544549e-13
	steps	58	62	54	73
fzero	X	NaN	0.9471 - 0.0038i		0.7322 + 0.0978i

	fval	NaN	-0.0000 + 0.3041i		-0.0000 - 0.5185i
	steps	21	83 (Wrong)		61 (Wrong)
Secant	X			NaN	NaN
	fval			NaN	NaN
	steps			8 (Failed)	3 (Failed)
Bisection	X				NaN
	fval				NaN
	steps				3 (Failed)
False Position	X				NaN
	fval				NaN
	steps				3 (Failed)

Tab.13 Comparison for $f(x) = x \cdot \left(0.01 - \sqrt{x^2 - 1}\right)$

Conclusion

In this report, a robust and efficient real root finder My_fzero for one variable equation $f(x) = 0$ is introduced. It uses a combination of many root finding methods: Bisection Method, modified Secant Method, Quadratic Method, and finally Brent's Method, which itself is also a combination of several root finding methods including Bisection Method, False Position Method and Inverse Quadratic Method. My_fzero automatically selects one of the methods which it considers to be the best for the current searching situation. It can also deal with out of domain problem, which makes it even robust.

My_fzero tries to find a root, beginning from one or two initial guesses provided by the user. Usually, it will find it quickly if such a root exists. Sometimes round off problem makes $f(x)$ never vanish. In this case, My_fzero tries to approach to the root as precise as possible. That is to say, it finds two successive machine representable points where $f(x)$ reverses sign.

However, My_fzero does not insist to find a root. This is mainly because a root does not always exist. When My_fzero can not find two points where $f(x)$ reverses sign, and can not find a point with decreasing magnitude of $f(x)$ for a number of successive iterations, it assumes it has found a local minimum of $|f(x)|$. This method prevents My_fzero from searching forever, when a root does not exist, or when My_fzero is dithering near a local minimum. But the cost is that My_fzero becomes more ready to stop near a local minimum, even if there is a root.

My_fzero does not require a tolerance. It tries to find the root as precise as possible. Although this is usually beneficial for the user, sometimes it will become slow, especially

when the root is 0 or infinite. Take $f(x) = 1/x$ as an example, given the initial guess 1, My_fzero iterates 1477 times before it finds the root at infinite. Another example is $f(x) = x^2$. Given the initial guess 1, it needs 776 iterations before it finds that $f(x)$ vanishes at 1.570382273913005e-162.

Future work may be directed to improving My_fzero for some cases where My_fzero is not efficient. For example, when solving $f(x) = x^2$, My_fzero uses Secant Method, while Secant Method converges slowly for this case. Also, more examples are necessary to test My_fzero for further improvement.

REFERENCES

[1] WILLIAM M. KAHAN: *Personal Calculator Has key to Solve Any Equation f(x)=0,* Hewlett-Packard Journal, December 1979

[2] PAUL J. McCLELLAN: *An Equation Solver for a Handheld Calculator*, Hewlett-Packard Journal, August 1987

[3] Matlab *"fzero.m"*, Version 5.3

[4] WILLIAM H. PRESS: *Numerical Recipes, the Art of Science Computing,* Cambridge University Press 1986

[5] A. M. OSTROWSKI: *Solution of Equations and Systems of Equations.* 2nd Edition, New York: Academic Press 1966

[6] J. F. Traub: *Iterative Methods for the Solution of Equations,* Englewood Cliffs, N. J. Prentice-Hall, Inc. 1964

[7] J. STORE: *Introduction to numerical Analysis,* Springer-Verlag New York Inc. 1980

Appendix

Syntax

x = My_fzero(fun, x0)
x = My_fzero(fun, x0, trace_level)
x = My_fzero(fun, x0, trace_level, p1, p2 …)
[x, fval] = My_fzero(…)
[x, fval, exitflag] = My_fzero(…)
[x, fval, exitflag, fcount] = My_fzero(…)

Description

x = My_fzero(fun, x0) tries to find a zero using the initial guess(es) x0. x0 can be a scalar or a vector of length two, which means My_fzero can have one or two initial guesses.

The value x is a root if My_fzero finds one, or x(1) and x(2) are two successive machine representable points where fun reverses sign, or x is a local minimum of |f(x)|, or x is the last point before My_fzero claims no root has been found. Usually, x(2) will keep the second latest point of guess. The meaning of x is indicated by exitflag.

x = My_fzero(fun, x0, trace_level) displays information according to trace_level:

x = My_fzero(fun, x0, trace_level, p1, p2 …) provides for additional arguments, p1, p2, etc., which passed to the objective function, fun.

[x, fval] = My_fzero(…) returns the value of the objective function fun at the solution.

[x, fval, exitflag] = My_fzero(…) returns a value exitflag that describes the exit condition, thus the meaning of x.

[x, fval, exitflag, fcount] = My_fzero(…) returns a fcount which indicates how many times the function fun has been executed.

Arguments

Input Argument:

fun The function whose zero is to be searched. fun is a function that accepts a scalar x and returns a scalar f, the objective function evaluated at x. The function fun can be specified as a function object:

x = My_fzero('Anyfun', x0)

where Anyfun is a Matlab function.
Fun can also be an inline function:

x = My_fzero(inline('exp(x)-3'), x0)

x0 Initial guess or guesses

trace_level

trace_level indicates what information to display. The value of trace_level can be:

0 no display.

1 display the final message (find a root, minimum, no root or error, etc.)

2 display information for each iteration.

p1, p2, …

Additional arguments for function fun.

Output Argument:

x Root or two successive representable points where fun reverses sign or local minimum of |fun| or the last point before exit.

fval The value of the function fun at x.

exitflag

A value indicates the exit condition. It can be:

0 Root found successfully.

1 Arguments error.

2 Second argument must be of length 1 or 2.

3 The input function failed.

4 All guess(s) is/are not valid or out of domain.

5 One guess is valid, but cannot find any other valid point near it.

6 Search already extended to +/-Inf. But the value of FunFcnIn at that infinite point is not a number. There might be a limit.

7 Search already extended to +/-Inf. There might be a limit.

8 Search interval exhausted during Secant extrapolation.

9 There might be a minimum magnitude.

10 Found an interval where the function reverses sign.

11 Search interval whether the function reverses sign exhausted during Bisection method.

Fcount

Number of times the function fun has been executed.

Algorithm

Please refer to the body of this report.