Root-Finding Methods in Two and Three Dimensions

Robert P. LeBoeuf

December 23, 2009

1 What is Root Finding?

Root finding is the process of finding solutions of a function $f(x) = 0$. As we learned in high school algebra, this is relatively easy with polynomials. A line's root can be found just by setting $f(x) = 0$ and solving with simple algebra. The quadratic formula gives us the way to solve for the roots of any parabola, and even allows us to find complex roots if they exist. Furthermore, there is also a cubic formula for solving for any cubic and a quartic formula for solving for any quartic. For polynomials greater than degree four and for any other continuous function where the roots cannot be solved for by algebraic means, we must use other more general methods to find the roots of the function. Well known important root finding methods will be discussed along with their pros and cons, the functions they work with best, and what are good ways to go about finding roots of certain functions.

2 The importance of root finding

Finding where the function crosses the x-axis can be very important. It can give us a better picture of the function so that complete picture of the function can be formed. It is used by calculators to find roots of numbers. Root finding methods can also determine the exact point where functions go from positive to negative, and can also be used in business to find break even points.

3 The Intermediate Value Theorem

The Intermediate value theorem is an intuitive theorem, but what it states is the basis for why certain root finding methods will work. It states that as long as there is a change in sign of the values of $f(x)$ over the interval $[a, b]$ over a continuous function, there is a root over the interval $[a, b]$. Basically, if a continuous function changes sign between two distinct points, then we must conclude that the function has crossed the x-axis.

4 Root Finding Methods in the Real Plane

Root finding methods in the real Cartesian coordinate system are pretty straightforward methods to finding roots. We are looking for where the graph of the function $f(x)$ crosses the x-axis in the real Cartesian coordinate plane. There are non iterative methods for finding these values in special cases such as the quadratic formula, but for functions where no distinct method is present, the general methods described below are used to find the root.

4.1 The Bisection Method

The bisection method is one of the oldest and most reliable ways of finding the roots of a continuous function. Because of this however, it is not a very fast method and requires the Intermediate Value Theorem to hold true over an interval provided in order to converge to a root. Other methods,described later take initial "guesses" of where the root is where there is not necessarily a sign change. Take a function $f(x)$ and an interval $[a, b]$ over which there is a change in sign. The bisection method will then find $f(c)$, where c is the point halfway between a and b. Now we find the sign of $f(c)$. If we are

lucky, $f(c) = 0$. If we are not lucky, then we repeat these steps over the interval $[a, c]$ if the sign of $f(c)$ is opposite the sign of $f(a)$, we will repeat over the interval $[b, c]$ if the sign of $f(c)$ is opposite the sign of $f(b)$. Below is a figure showing how the bisection method iterates: The bisection method is

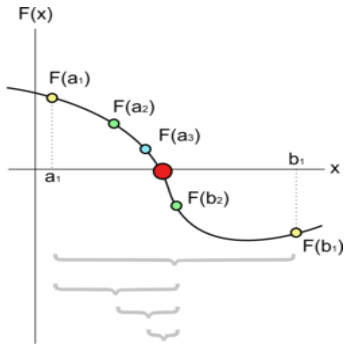

Figure 1: The Bisection Method

defined as having a convergence factor of 1, making it the slowest root-finding algorithm, but it is also the most reliable root-finding algorithm, and is used in cases where other faster algorithms will fail.

4.2 The Secant Method

The secant method is slightly different than the bisection method. It takes two input values over which there is a sign change, but instead of bisecting the difference between them after each iteration, it takes the points on the function at those initial guesses and constructs a line (secant line) connecting the two. Where this line crosses the x-axis is the next guess, and will replace the initial value with the same sign, and repeat the process. The method can be thought of in the following way: let a and b be the initial guesses, and let c be the next guess determined by the method: $c = a - f(a)\frac{a-b}{f(a)-f(b)}$.

4.3 Newton's Method

Newton's Method was developed in the 17th century by Sir Isaac Newton,one of the inventors of the calculus. Unlike the methods described above, Newton's method involves the taking of a derivative of a function. Newton's method takes an initial input or "guess" of where the root may be. Unlike

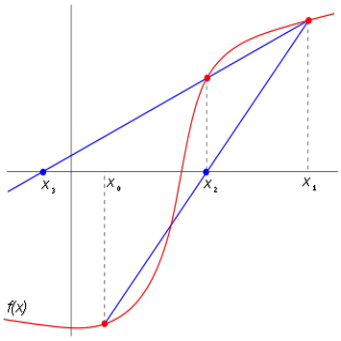

Figure 2: The Secant Method

the bisection method or secant method, Newton's method does not physically take an interval, but it computes a better guess as to where the root may be, and that better guess will converge to a root. Newton's method works like this: Let a be the initial guess, and let b be the better guess. By Newton's method: $b = a - \frac{f(a)}{f'(a)}$. Some will say that Newton's method is very fast and has a convergence factor of 2. However, for every one step of Newton's method, two steps of the secant method can be done, because Newton's method requires the taking of a derivative and then finding two function evaluations. This being true, it is more proper to consider Newton's method as having a convergence factor of $\sqrt{2}$ (about 1.41.) Below is a graphical representation of Newton's Method:

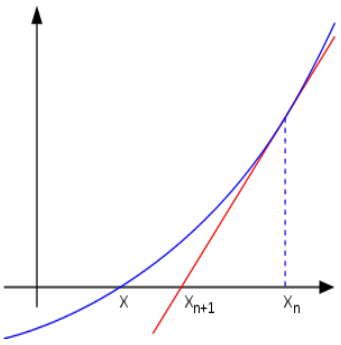

Figure 3: Newton's Method

4.3.1 Newton's Method vs. Secant Method

The secant method in most cases will be faster than Newton's method because of one major factor. Newton's method requires an evaluation of a value in two functions: the actual fucntion and its derivative, whereas the secant function only requires that the function be evaluated once at each point. However, Newton's method can still be the method of choice for certain tasks. For example, Newton's method works much better with polynomials, especially those in the form $f(x) = x^2 - c$ where c is a constant. In fact, calculators use Newton's method when finding square roots. For example, let's say we would like to find $\sqrt{3}$. The calculator would use Newton's method and solve for the equation $x^2 - 3 = 0$. It is programmed to choose likely values (such as values less than $|3|$ obviously), and it will find the root extremely simply as the function will converge very quickly.

4.3.2 Strange Behavior with Newton's Method

Because Newton's method has an evaluation with a derivative in the denominator, guesses close to where the derivative is equal to zero will not converge. There is also another interesting phenomenon when dealing with Newton's method. One day Profs. Saeja Kim and Adam Hauschnekt brought something interesting to my attention. Consider the following equation: $f(x) = x^3 - 5x$, and choose 1 as the initial guess of where the root is. If we iterate Newton's method once, we will get -1 as our next guess. If we iterate again, we will get 1 as our next best guess, and this pattern will repeat. $x_0 = 1$ $x_1 = -x_0 = -1$and $x_2 = -x_1 = x_0 = 1$. In fact, I discovered that all functions of this form: $f(x) = x^3 - a^2x$ will have an initial value $x_0 = \frac{a^2}{\sqrt{5}}$ in which Newton's method will not converge. I have also found that the functions of which this strange behavior of Newton's method exists are all odd functions, which are functions of the form $-f(x) = f(-x)$. Prof. Leon showed that where a change in concavity occurs (where the second derivative is equal to 0 or undefined) there will exist 2 singular points where this behavior occurs.

4.3.3 Practical Uses

Newton's method is now considered by most mathematicians to be inferior to the secant method because of what can be a tedious task of having two function evaluations. That along with the potentially bad starting points

makes it unattractive to mathematicians for practical uses. The main use of Newton's method is for finding squareroots on a calculator. On calculators (or in Matlab), if I wanted to find $\sqrt{2}$, I'd hit a button, the calculator would tell me that it's about 1.4142 and that's the end of it. But how is the calculator solving that? The calculator is actually finding the positive root of the equation $f(x) = x^2 - 2$ using Newton's method. Newton's method is uesd because of the simplicity of the function's derivative. The derivative of this equation is simply $f'(x) = 2x$. Newton's method is superior to the secant method in this instance because of the simplicity of the derivaitve of quadratic functions in the form of $f(x) = x^2 - c$ where c is a constant.

4.4 Muller's Method

Muller's method is somewhat similar to the secant method, but instead approximates the next root using a polynomial of degree 2. Muller's method also employs the quadratic formula, and because of this complex roots of real functions can be found using Muller's method. The formula for Muller's method is as follows: let $q = \frac{a-b}{b-c}$, $A = qf(a) - q(1+q)f(b) + q^2 f(c)$, $B = (2q+1)f(a) - (1+q)^2 f(b) + q^2 f(c)$ $C = (1+q)f(a)$. The next iteration is $d = a - (a-b)\frac{2C}{max(B \pm \sqrt{B^2 - 4AC})}$. Muller's method converges quickly, having a convergence factor of 1.6. Muller's method is the method of choice if a function may have complex roots, but if it is unlikely that it will, then usually the secant method or a different method that is slightly faster is preferable. Since Muller's method incorporates the quadratic formula, real guesses can converge to complex roots. Recall the discriminant from the quadratic formula: $b^2 - 4ac$. When the discriminant is negative, complex roots result because the square root of the discriminant is taken. Muller's method will still iterate with the new complex guesses and will eventually converge to the correct complex root. If you are working in the real plane with no sign change over an interval and believe there may be complex roots, Muller's Method is the method of choice.

4.5 Inverse Quadratic Interpolation

Inverse quadratic interpolation is very similar to Muller's method, except that it uses an inverse parabola in the form of $f(y) = ay^2 + by + c$ and finds where the sideways parabola crosses the x-axis and uses that point as the next point for the guess. This way, it will be slightly faster than Muller's

method, because the maximum value of the quadratic formula does not need to be computed since the inverse parabola will cross the x-axis in only one point as opposed to the two points it would cross in Muller's method. The rate of convergence is also slightly faster, at 1.8 instead of the 1.6 of Muller's method. If a value is chosen very far from the actual root though, the method may not converge or may converge much slower than expected. This method has a couple of drawbacks: it cannot compute complex roots like Muller's Method because it will always cross the x-axis in one place, and if the three initial guesses chosen are very far from a root, the method will not converge. But it is also one of the fastest methods for computing a root, and for that reason it is widely used today.

4.6 Brent's Algorithm

I have described various well known very good root finding algorithms that are widely used by mathematicians today. Each root finding method has its pros and cons. The bisection method will always work, as long as there is a sign change, but it is also the slowest method. Newton's method converges extremely quickly, but requires two function evaluations and the finding of a derivative. Muller's Method is fast and can even find complex roots, but has a lot of computation and will not converge if the guess is a point too far away from the root. Thses various methods raise the following questions: Which method is best for a particular function and should any of the methods be combined for a particular function? A mathematician by the name of Richard Brent developed an algorithm in 1973 that answers these questions. Brent's algorithm chooses between three methods of root finding: bisection, secant and inverse quadratic interpolation. It uses the best aspect of the three methods and chooses between the best one to use at each step. If inverse quadratic interpolation would converge outside of the interval where a sign change exists, then either bisection or secant would be the best method to use at that particular step. So, given the sign of the midpoint of the interval, if the secant method would converge at a point further away from the midpoint, then we would use bisection. Brent's algorithm uses the best aspects of each method to create a method that converges quickly, has very few drawbacks and will always converge to a root no matter the case. This algorithm is said to be one of the best root finding algorithms out there, and is state of the art for use of root finding today. The Matlab fzero function (Matlab's root finding function) incorporates the algorithm as part of it.

4.6.1 Drawback to Brent's Algorithm

One drawback to Brent's algorithm is that it cannot find complex roots of functions. A way to correct this is to use Muller's method if there is no sign change over a given interval of the function. Muller's method will construct the parabola, find the roots of that parabola, and keep iterating until a root is reached. So instead of trying to find an interval over which there is a root, Muller's method can be used instead to reach a root. If the root does not really exist over the interval, then Muller's method will not converge.

4.6.2 Personal Improvement to Brent's Algorithm

I suggest the following improvement to Brent's algorithm. The one main drawback to Brent's algorithm is that it requires a sign change over an interval and for that reason cannot compute complex roots. So what if instead of trying to find another interval over which there is a sign change, we compute Muller's method using the two inputs that Brent's algorithm requires for its method and create a new input halfway between the initial inputs in Brent's algorithm and excecute Muller's method for one iteration. If a complex root is found, we will continute to converge using Muller's method and if not, we will go back to using the other methods in Brent's algorithm. If the user inputs an interval over which there is no sign change, we could run one iteration of Muller's method to see if the function converges to a complex root. If it begins to do so, then we can continue running Muller's method until it converges. If it does not, then we can ask the user to input a different interval.

5 Three Dimensional Root Finding Methods

Root finding in three dimensions is somewhat different from root finding on the xy plane. In the xy plane we are solving for where a function crosses the x-axis, where its y value is zero. We have one equation and one unknown, so it is solvable. In 3 space however, if we set $f(x, y)$ equal to zero and solve for y, we can acheive a function (or relation, the result of solving for y may not pass the vertical line test) in the xy plane that will describe all of the points where the function in 3D will pass through the xy plane. To talk about root finding methods in 3D, we must have two equations as a function of both x and y and the method will converge to a point on the xy plane where both

functions will intersect. Visually, the two functions in 3D will have no height at that point. There are two main methods used to find this.

5.1 The importance of 3D root finding

Finding the intersection of two functions on the xy plane is very important because it solves a system of two non linear equations with two unknowns. This can be difficult to do by hand and sometimes inpossible. Using iterative algorithms like those that will be described below can give us a general way to solve any two non linear functions in 3D. In the buisness world, if we have two equations that take two variables and we would like to find the break even point of both functions, 3D root finding would be the way to go. It also tells us where the functions in 3D have no height, which is a critical piece of information when graphing these functions and describing them.

5.2 2D function finding

In 3D, if we would like to find where a single function $f(x, y)$ crosses the xy plane, we need only set $f(x, y)$ equal to zero and solve for y. For example, let $f(x, y) = 3xy^2 - 1$. Then, $3xy^2 - 1 = 0$ and solving for y, we get $y = \sqrt{\frac{1}{3x}}$. Matlab is usually not very good with symbolic algebra, so we need to do this step out by hand. Sometimes this can be very tedious, and sometimes it could be impossible to solve for y. When it is, we can set one of the variables as a constant, and find where the function will be zero when $x = 3$ for example. If we do this, we can find a bunch of points on the xy plane and use polynomial interpolation to fit a polynomial of a high degree to the set of points to find an approximate function where the surface will cross the xy plane.

5.3 Bisection in 3D

I have attempted to excecute the bisection method in 3D, but it does not appear to work. I do not know if there is an extention of the Mean Value Theorem in 3D. I have researched this topic but to no avail. Any feedback on a bisection method in three dimensions that works would be greatly appreciated.

5.4 Newton's method in 3D

Recall Newton's method for functions in R2: $b = a - \frac{f(a)}{f'(a)}$, where b is the better guess and a is the initial guess, and we keep iterating to some tolerance where we are satisfied that the method has converged to the root. In three dimensions, the method is very similar. We need to take into account the fact that it is a function in three dimensions. To find the "derivative" in three dimensions means we need to take the x partial derivative and the y partial derivative of each function, and set them up in a Jacobian Matrix. A Jacobian matrix is a matrix that is set up like this: $(\ f\ )\, xfy$
$gxgy$
. Where fx, fy, gx, gy are the partial derivatives for the variables x and y for each function f and g. Let J be the Jacobiam Matrix for two functions in 3D, let F be the vector of the function f and g evaluated at the initial guess point (x, y), and let dp be the change in the point that the method converges to, the better guess. Then, $dp = J^{-}1 * -F$. This is Newton's method in 3D. After the method converges, the point that we end up with will cause both functions to yield a value of 0 when the point is plugged in. See the function labeled newton3D below in matlab for the m-file developed to excecute the method. If the Jacobian is singular, we obviously cannot excecute Newton's method. If it is singular however, the functions will never intersect on the xy plane at all. Since we take the Jacobian evaluation at the same point, if the Matrix has no inverse, it cannot converge to a root. I have graphed sets of functions on Graphing Calculator and have seen that in all cases if the Jacobian is singular, the functions will never cross each other on the xy plane.

5.4.1 Pros to Newton's Method

This method is the most widely used method for 3D root finding. It is very fast and converges quickly to a root. For solving nonlinear systems of equations, this method is the most widely used and sucessful. It has very few drawbacks, the main one being all of the function evaluations that are necessary.

5.4.2 Drawback to Newton's in 3D

A main drawback to Newton's Method in 3D is that it will not always converge to the root that is closest to it. It can converge to a root that is very far

away, and if it does this, the method will take longer than Broyden's method which is an extention of the secant method in 3D. As in 2D where we had the points that did not converge at all, this could also occur. If we start at a point in Newton's method where we let $dp_0 = -dp_1$ and solve for dp_0, we will get a point where Newton's method will bounce back and forth between two points and never iterate to the root. Furthermore, if we take points as a guess point that are very close to those fixed points, the method will still converge, but will converge much slower. It is very unlikely that this will occur, but it is still a drawback. It also requires six function evaluations, one for each function and one for each of the partial derivatives. This means that one iteration of the method will be long.

5.5 Broyden's Method

Broyden's method is a generalization of the secant method in 3D. In 2D, the secant method is excecuted by finding the difference approximation (approximating the derivative of the function f). Broyden's method will work similar to Newton's method except that instead of re-evaluating the Jacobian after each iteration, it just performs an update of rank one. The Jacobian is also evaluated using the difference approximation that's used in the secant method in 2D which is $f'(a) = \frac{f(a)-f(b)}{a-b}$. The Jacobian is determined using the finite difference approximation: $J * (a - b) = F(a) - F(b)$. J is then updated at each step like so: $J_n = J_{n-1} + \frac{\Delta F_n - J_{n-1}\Delta x_n}{\Delta x_n^2}\Delta x_n^T$. Where ΔF is the change in the function evaluations, Δx is the change in the point, or dp from Newton's method. The Jacobian is updated this way and then Newton's method is performed again, as stated above in the Newton's method in 3d. The good thing about this method is the fact that when the Jacobian is updated this way, we do not need to take another function evalutation, and the method itself takes a shorter ammount of time. However, since the Jacobian is not computed exactly, Broyden's method will converge slower than Newton's method will.

6 A side topic I worked on

I attempted to work on a side project of creating an m-file that would generate the first n elements of the Fibonacci sequence. I was unsucessful and am looking for advice as to how to go about doing this for futrue research in the

field.

7 Conclusions

Working with root-finding methods this summer and later 3D methods in the Fall has been very rewarding. I have learned much about how to find roots of specific functions, and I have learned why these methods work the way that they do. I have found little information about root finding with complex functions, but I will endeavor to learn more about what root finding in the complex plane is. For my future work with root finding, I will work with complex functions after learning more about them and becoming more adept with them. I will research into root finding methods with cubic convergence and attempt to better Brent's algorithm and possibly create an algorithm of my own. I will continue to research 3D root finding in my spare time and am looking into researching root finding methods in the complex plane, once I am more skilled with complex analysis.

Please direct comments and questions to the author,Robert P. LeBoeuf (Undergraduate Mathematics Major at UMASS Dartmouth), at RLeboeuf@umassd.edu. M-files that I have created for root finding thus far are listed below:

```
%MULLERBEEF (mfile)  for excecuting Muller's method.
%  Edit the m file and set fa fb and fc equal to a function in terms of a b
%  and c, but make the function exactly the same for fa fb and fc save the
%  name of the variable.  The rest of the function will excecute Muller's
%  method iterating 10 times.
clear
close
a=1;
b=2;
c=3;
f = inline('sin(x)+cos(x)', 'x');

%Check to see if ab or c is already the root
if f(a)==0
  root=a
  end
if f(b)==0
  root=b
```

```
  end
if f(c)==0
  root=c
  end


for k=1:100
temp=a;
temp2=b;
fa=f(a);
fb=f(b);
fc=f(c);
q=(a-b)/(b-c);
%defines q to be used later in the method for convenience

A=(q*fa)-(q*(1+q)*fb)+(q^2*fc);
%defines A to be used in the final quadratic formula evaluation
B=((2*q+1)*(fa))-((1+q)^2*fb)+(q^2*fc);
%defines B for quad formula
C=(1+q)*fa;
%defines c for quad formula

root=a-(a-b)*((2*C)/(max((B+sqrt(B.^2-(4*A.*C))),(B-sqrt(B.^2-(4*A.*C))))));

%set a tolerance
if (B+sqrt(B.^2-(4*A.*C)))
break
end

if (B-sqrt(B.^2-(4*A.*C)))
break
end


%rounds d to 0 if it is smaller than our tolerance of 10^-15
if root<10^-15
    root=0;
end
```

```
b=temp;
a=root;
c=temp2;
end
format long
root
```

```
%method for excecuting the bisection method
%Excecutes bisection method:  bisectleboeuf(f,a,b,tol)  where f is a
%defined function, a and bb are guess points of where the actual root is
%and tol is a tolerance set by the user to avoid machine rounding errors
%where divide by 0 could occur.

function [c]=bisectbeef(f,a,b,tol)

%checks for sign change over the interval [a,b]
if sign(f(a))==sign(f(b))
error('There is no sign change over the interval [a,b]')
end

%Checks to see if f(a) or f(b) is already 0, in which case we are done.
if f(a)==0
a
return;
end

if f(b)==0
    b
return;
end
```

```
%Excecutes the bisection method
for k=1:100

%Finds c so that it is the average from a to b
c=((a+b)/2);

%Will now see if the sign of c is=a or b?s sign, and iterate the next
%iteration accordingly depending on the sign of f(c).

if abs(f(c))<=tol
root=c
return;

else if sign(f(c))~=sign(f(a))
b=c;

else
a=c;

end

end
end
end



%Method for excecuting Inverse Quadratic Interpolation

% Finds the root of a function f using inverse quadratic interpolation for
% inputs a b c and function f.  f(a) f(b) and f(c) must not be equal.
% Function takes form of quadfit(fun,a,b,c)  taking the function, and three
% initial guesses as inputs and outputs the root r.  If function has complex roots
%method will not converge.
function [r]=quadfit(f,a,b,c)
%defines variables
fa=f(a);
fb=f(b);
```

```
fc=f(c);

%Checks to see if initial guess is the correct answer
if fa==0
    r=a;
    return
end

if fb==0
    r=b;
    return
end

if fc==0
    r=c;
    return
end

%Checks for a divide by 0 error
if fa==fb || fa==fc || fb==fc
    error('Divide by 0 error.  Choose different values for a b and c')
end
%Finds a b and c
A=(b-a)/(fb-fa);
B=(c-b)/(fc-fb);
C=(B-A)/(fc-fa);

%main loop excecuting method

for k=1:20
   r=a-B*fa+C*fa*fb;

   %re defining variables
   a=b;
   fa=fb;
   b=c;
   fb=fc;
   c=r;
```

```
   fc=f(r);


end
%Warning that convergence may not have occured

if abs(r)>=10^25
    'Warning:  method may not have converged.  Results could be unreliable'
end



%method for plotting iterations of newton's method.

%This method will excecute Newton's method and plot the tangent line at
%each iteration  newton plot takes 4 input values: function, derivitive of
%function, an initial root guess r and some space to plot x.
%plotnewton(f,df,r,xspace,).  MUST ENTER x.

function [p]=plotnewton(f,df,r,x)
fr=f(r);
dfr=df(r);
%Numerical Newton's Method
for k=1:20


p=r-(fr/dfr);

%Redefine variable for next iteration
r=p;
fr=f(p);
dfr=df(p);

plot(x,f(x),p,f(p),'*r','markersize',10)
pause(.5);
end
return;
```

```
%Method for excecuting the bisection method

%Excecutes bisection method:  bisectleboeuf(f,a,b,tol)  where f is a
%defined function, a and bb are guess points of where the actual root is
%and tol is a tolerance set by the user to avoid machine rounding errors
%where divide by 0 could occur.

function [c]=bisectbeef(f,a,b,tol)

%checks for sign change over the interval [a,b]
if sign(f(a))==sign(f(b))
error('There is no sign change over the interval [a,b]')
end

%Checks to see if f(a) or f(b) is already 0, in which case we are done.
if f(a)==0
a
return;
end

if f(b)==0
    b
return;
end

%Excecutes the bisection method
for k=1:100

%Finds c so that it is the average from a to b
c=((a+b)/2);

%Will now see if the sign of c is=a or b?s sign, and iterate the next
%iteration accordingly depending on the sign of f(c).

if abs(f(c))<=tol
root=c
return;
```

```
else if sign(f(c))~=sign(f(a))
b=c;

else
a=c;

end

end
end
end


%Method for excecuting Newton's Method

%newtonbeef[f,df,a] takes a function, its derivitive and an initial guess
%a as parameters and edvaluates the root.
function [b]=newtonbeef(f,df,a)
b=0;
fa=f(a);
dfa=df(a);
%Newton?s Method will excecute
for k=1:100
    temp=b;

b=a-(fa/dfa);

%Redefine variable for next iteration
a=b;
fa=f(b);
dfa=df(b);

end
end

%Newton's method in 3D
J=[dfx(r(1,1),r(2,1)) dfy(r(1,1),r(2,1));dgx(r(1,1),r(2,1)) dgy(r(1,1),r(2,1))]
F=[f(r(1,1),r(2,1)) g(r(1,1),r(2,1))]';
```

```
%Change if we choose a tolerance
if det(J)==0
   error('Jacobian is singular, this system has no solution')
end
JINV=(1/det(J))*[J(2,2) -J(1,2); -J(2,1) J(1,1)];
for k=1:100
    %dp=J^-1*-F;
    dp=JINV*-F;
    %re-defines variables
    r=r+dp;
    J=[dfx(r(1,1),r(2,1)) dfy(r(1,1),r(2,1));dgx(r(1,1),r(2,1)) dgy(r(1,1),r(2,1))
    JINV=(1/det(J))*[J(2,2) -J(1,2); -J(2,1) J(1,1)];
    F=[f(r(1,1),r(2,1)) g(r(1,1),r(2,1))]';

    %tolerence check
%if f(r(1,1),r(2,1))&&g(r(1,1),r(2,1))<=tol
 % k
  %return
%end

end

end
```