MATH 6644
Homework 2

Stephan Boettcher

April 14, 2015

Question 1

Can the performance of the Newton iteration be improved by a linear change of variables? That is, for nonsingular $N \times N$ matrices A and B, can the Newton iterates for $F(x) = 0$ and $AF(Bx) = 0$ show any performance difference when started at the same initial iterate? What about the chord method?

A linear change of variables is a technique used to reduce a difficult to a simpler one. This is commonly done by substituting values or expressions for ones that depend on other variables. By reducing the number of dependent variables in a set of expressions, the corresponding $N \times N$ A matrix becomes more sparse. For a given Newton iterate, $F(x) = 0$, the nonsingular A and B matrices can be used to modify the sparsity of the iterates, given by $AF(Bx) = 0$, to make them easier to evaluate. This would be beneficial to Newton's method as a more sparse iterate makes evaluating the Jacobian less expensive. With Newton's method, both the iterate, $AF(Bx_n)$ and the corresponding Jacobian are evaluated with each iteration. Thus, if they can be made less costly to evaluate, a performance increase can be expected. However, the degree of sparsity and resulting performance increase must offset the cost of the extra matrix-vector multiplications, which may quickly and completely offset the gains mentioned above. The other way the performance of Newton's method could be expected to increase is from a reduction in the number iterations necessary to converge. In order to evaluate this, we must first look at Newton's equation.

Newton's Method converges quadratically to x and the error of the function is governed by the equation:

$$||\vec{x}_{n+1} - \vec{x}_n|| = || - (F'(\vec{x}_n))^{-1} F(\vec{x}_n)||$$

Substituting in A and B into the above equation, we get:

$$||\vec{x}_{n+1} - \vec{x}_n|| = || - (ABF'(B\vec{x}_n))^{-1} AF(\vec{Bx_n})||$$

Since we know that the A matrix is non-singular, the A and A^{-1} will cancel. We also know that $||(F'(\vec{x}_n))^{-1} F(\vec{x}_n)||$ is a good estimator for $||\vec{x}_n - \vec{x}^*||$, we can write the above equation as:

$$||B^{-1}(B\vec{x}_{n+1} - B\vec{x}^*)|| \leq k||B^{-1}(B\vec{x}_n - B\vec{x}^*)||^2$$

where $\vec{x}^*$ is the solution, k is a constant, and $\vec{x}_n$, $\vec{x}_{n+1}$ are the values of $\vec{x}$ at steps n and $n+1$. At this point we note that the B and B^{-1} matrices cancel and any potential impact upon the convergence rate is nullified. From this analysis, Newton's method's convergence rate will not benefit from a linear change of variables. An almost identical analysis of the Chord method yields the answer to potential performance increases.

The Chord method evaluates the Jacobian matrix once, prior the the iteration section of the code. The remainder of the code uses a set LU-decomposed version of the Jacobian to iterate to a solution. Since the Jacobian does not change over the life of the Chord method, only as small performance increase would be realized. The chord method is also governed by the equation:

$$||\vec{x}_{n+1} - \vec{x}_n|| = || - (F'(\vec{x}_n))^{-1} F(\vec{x}_n)||$$

Substituting in A and B into the above equation, we get:

$$||\vec{x}_{n+1} - \vec{x}_n|| = || - (ABF'(B\vec{x}_0))^{-1} AF(B\vec{x}_n)||$$

Since we know that the A matrix is non-singular, the A and A^{-1} will once again cancel. Using the fact that $||(F'(\vec{x}_0))^{-1} F(\vec{x}_n)||$ is a good estimator for $||\vec{e}_n||\ ||\vec{e}_0|| = ||\vec{x}_n - \vec{x}^*||\ ||\vec{x}_0 - \vec{x}^*||$, we get:

$$||B^{-1}(B\vec{x}_{n+1} - B\vec{x}^*)|| \leq k||B^{-1}(B\vec{x}_0 - B\vec{x}^*)||\ ||B^{-1}(B\vec{x}_n - B\vec{x}^*)||$$

Once again, the non-singular B matrices cancel and have no impact upon the convergence rate of the Chord method.

To summarize, performing a linear change of variables on the Chord and Newton's methods has no impact upon the convergence rates of either methods. While there is a small possibility that performing a linear change of variables may improve the evaluation of the jacobian, it is likely that any performance gain would be offset by the cost of the matrix-vector multiplication required. Thus, it is not advisable to perform a linear change of variables.

Question 2

Write a program that solves single nonlinear equations with Newton's method, the chord method, and the secant method. For the the secant method, use $x_{-1} = 0.99x_0$. Apply your program to the following function/initial iterate combinations, document and explain your results:
(a)$f(x) = 2x^2 - 5; x_0 = 10$;
(b) $f(x) = sin(x) + x; x_0 = 0.5$;
(c) $f(x) = cos(x); x_0 = 3$

The Newton's method, the Chord Method, and the Secant Method were all programmed in Matlab and used as nonlinear solvers for functions a,b, and c using the initial conditions given. Each of these codes can be found in the attached documentation, or listed in Appendix A below. All three methods used the following stopping criteria:

$$||F(x)|| \leq \tau_r ||F(x_0)|| + \tau_a$$

where τ_r is the relative tolerance compared to the initial norm of the function, and τ_a is the absolute tolerance of the function. Both τ_r and τ_a were set to 10^{-6} for this Question.

The first function was defined as:

$$f(x) = 2x^2 - 5 \qquad x_0 = 10$$

This function is a basic parabola with a minima at $x = 0$, and $F(x^*) = 0$ at $x \approx \pm 1.5811$. The three methods were all used to find the minima, and agree to down to the $\approx 10^{-5}$ decimal place. The final x values of the three methods can be see in Figure 1. All three methods were able to converge to a common value for the first two functions. However, the periodicity of the cosine function in function c resulted in the failure of the Secant and Chord methods. Figure 2 shows the number of iterations required to converge for the three different methods. For Function a, Newton's method converged the quickest, but as can be seen by Figure 3, all three methods converged to the same point. The Chord method is a locally linearly convergent method and as a result, it took the longest to converge on the answer.

Function	Newton's	Chord Method	Secant Method
a	1.58113883	1.58113910	1.58113890
b	0.00000000	-0.00000000	0.00000000
c	-4.71238898	Did Not Converge	Did Not Converge

Figure 1: Final x values for the 3 functions

Function	Newton's	Chord Method	Secant Method
a	6	91	8
b	3	7	4
c	4	Did Not Converge	Did Not Converge

Figure 2: Number of Iterations required to converge

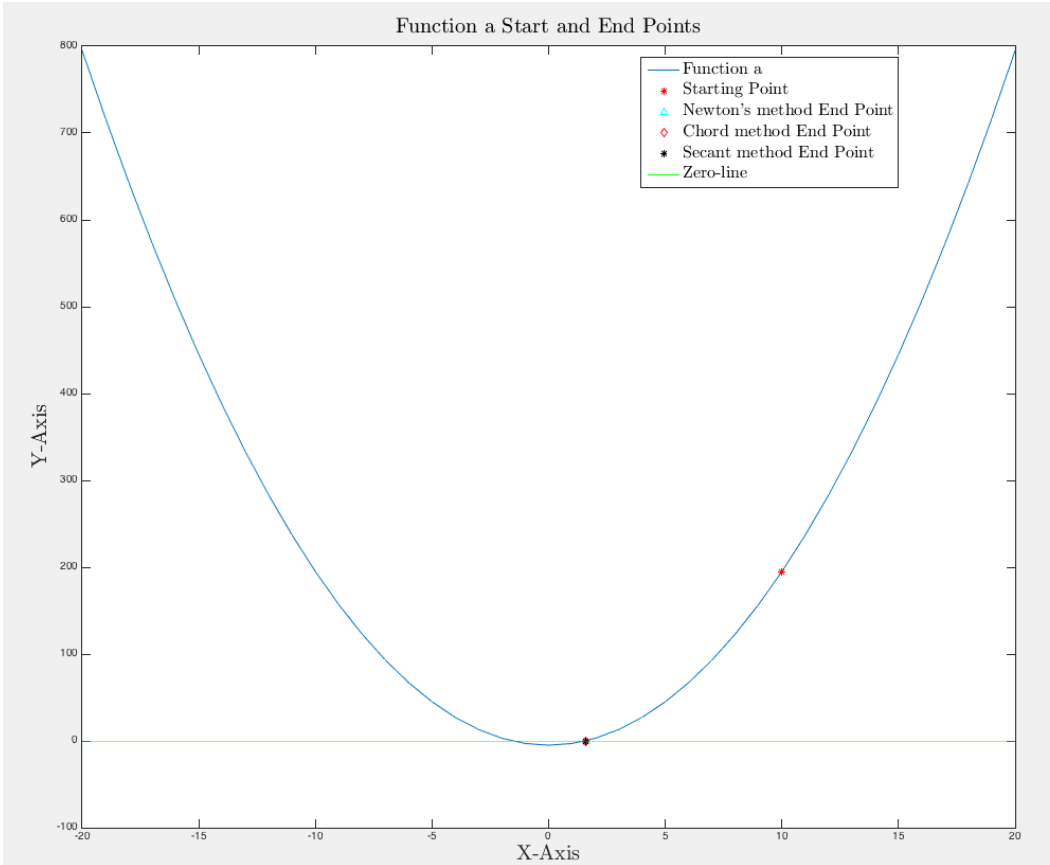

Figure 3: Initial conditions and final answer of Newton's Method, the Chord Method, and the Secant Method for Function a.

For Function b, once again, all three methods converged to the same point, as seen in Figure 4. As can be seen in Figure 2, Newton's method once again converges the fastest, but is only marginally quicker than the other two methods. For both Functions a and b, the starting point was close enough to the $F(x^*) = 0$ point that the methods were able to converge.

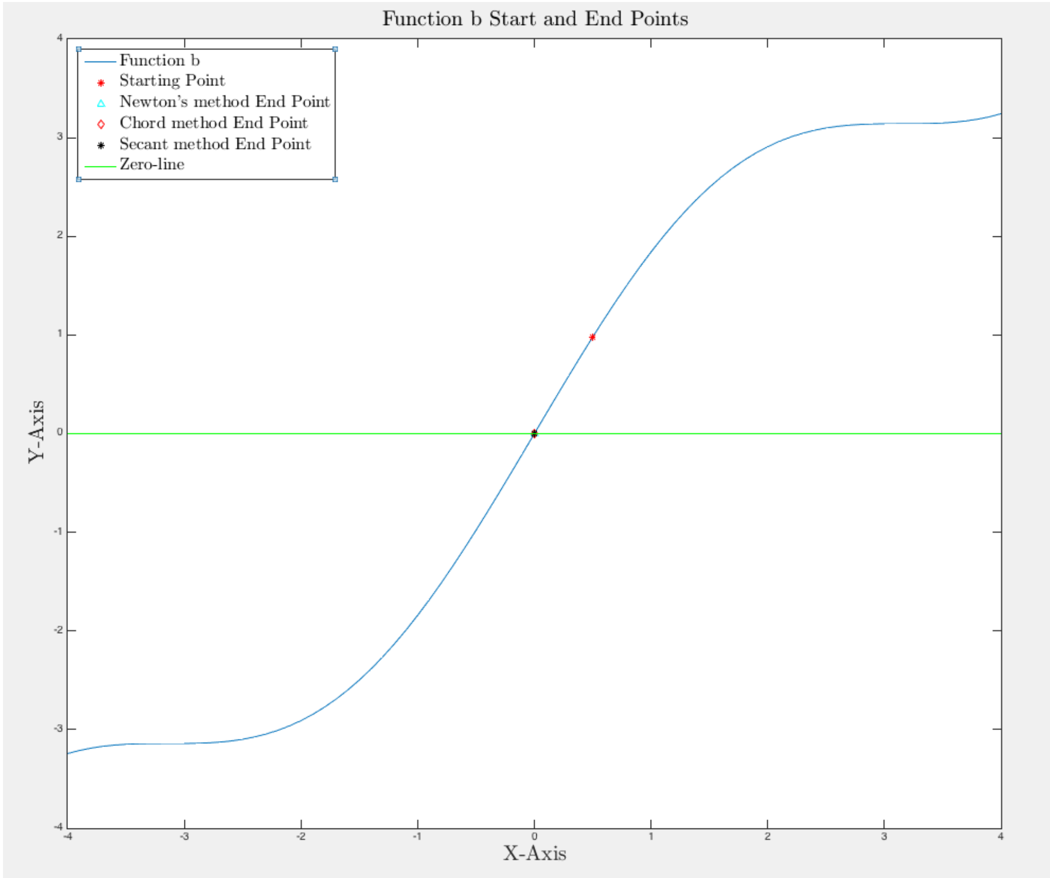

Figure 4: Initial conditions and final answer of Newton's Method, the Chord Method, and the Secant Method for Function b.

The Chord and Secant methods both had issues converging on Function c due to its periodic motion, as can be seen in Figures 1,2, and 5. While Newton's method did find a point where $F(x) = 0$, this was by no means the closest zero point. The Secant method diverged the fastest from the true solution, quickly reaching an x value equal to the computational precision of Matlab, which is 2^53. At this point, the method is still working, but Matlab is unable to correctly calculate the next step. When a_n is being calculated, a divide-by-0 error occurs and the program exits. The Secant method works by approximating the derivative, $F'(x)$ by using the previous step's x and $F(x)$ values. However, with a periodic function, errors in the derivative can and will send the function off in an incorrect direction. If any x point is closest to to an extreme on the y-axis, the Secant — and Chord Method for that matter — will go careening off into the distance. This doesn't mean that they won't eventually converge, but if they do it will be at a $F(x) = 0$ far from the start point.

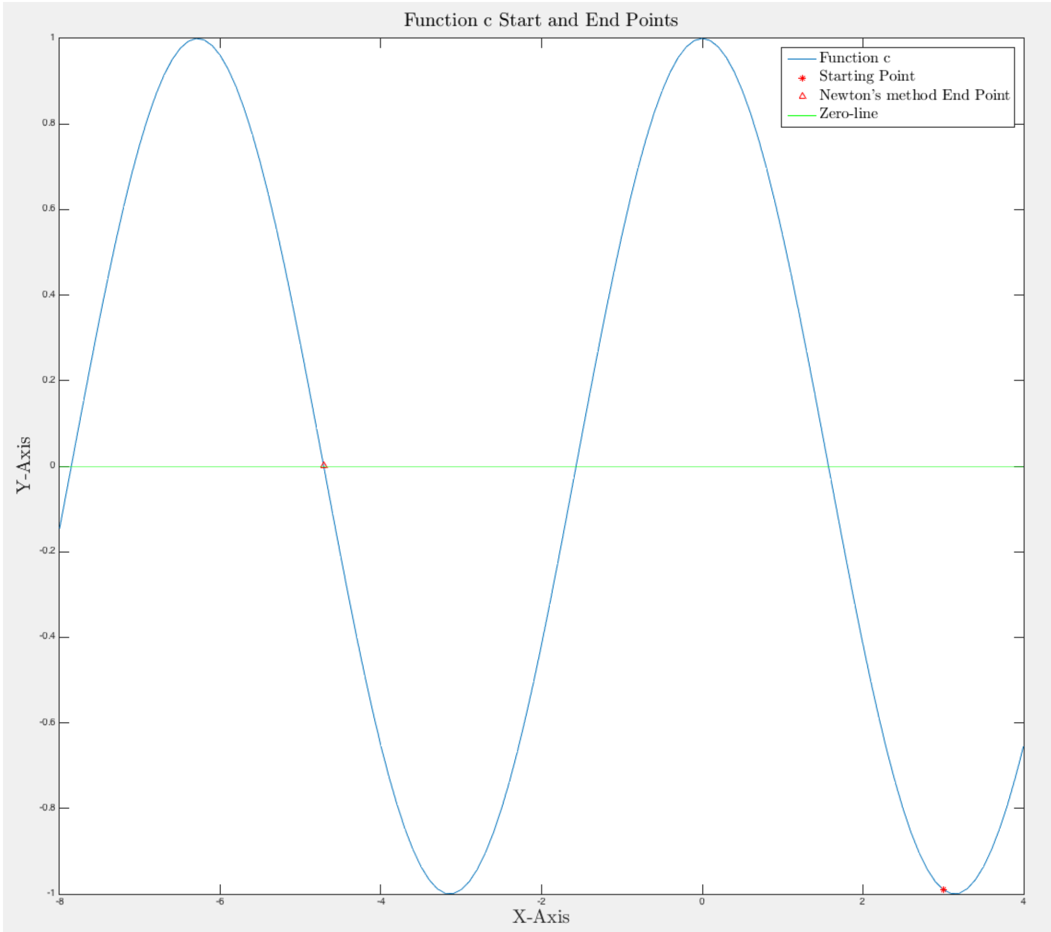

Figure 5: Initial conditions and final answer of Newton's Method, the Chord Method, and the Secant Method for Function c.

Question 3

Assume that the standard assumptions hold, that the cost of a function evaluation is $\mathcal{O}(N^2)$ floating-point operations, the cost of a Jacobian is $\mathcal{O}(N)$ function evaluations, and that x_0 is near enough to x so that the Newton iteration converges quadratically to x. Estimate what is the number of iteration needed to obtain $||e_n|| \leq \epsilon ||e_0||$, where ϵ is a small tolerance value. What is the number of floating point operations required to get this accuracy?

Newton's Method converges quadratically to x and the error of the function is governed by the equation:

$$||\vec{x}_{n+1} - \vec{x}^*|| \leq k||\vec{x}_n - \vec{x}^*||^2$$

where $\vec{x}^*$ is the solution, k is a constant, and $\vec{x}_n$, $\vec{x}_{n+1}$ are the values of $\vec{x}$ at steps n and $n+1$. Since $\vec{x}^*$ is the solution, $\vec{x}_n - \vec{x}^*$ can be written as:

$$||\vec{e}_{n+1}|| \leq k||\vec{e}_n||^2 \leq (k||\vec{e}_{n-1}||^2)^2 \leq \dots$$

We know from these equations and the problem statement that the Newton iteration converges quadratically and that the norm of the error is dependent upon the previous step's error. Thus, to get to $\vec{x}_n$, the error has converged by $(k||\vec{x}_0 - \vec{x}^*||^2)^{n+1} = ||\vec{e}_n||$. The number of iterations necessary to get to the ϵ tolerance is:

$$\frac{||\vec{e}_n||}{||\vec{e}_0||} = (k||\vec{x}_0 - \vec{x}^*||^2)^n = \epsilon$$

$$n = \frac{log(\epsilon)}{log(k||\vec{e}_0||^2)}$$

Assuming that evaluating the function, $F(x)$ costs $\mathcal{O}(N^2)$ FLOPS and evaluating the Jacobian is only $\mathcal{O}(N)$ FLOPS, the number of of FLOPS required to get to an accuracy of ϵ can be calculated. The breakdown of the FLOPS cost per step for Newton's method is given below:

$$\begin{aligned}
\vec{r} &= ||F(\vec{x})|| & N^2\ FLOPS \\
F_x &= \vec{r} & N\ FLOPS\ for\ an\ assignment \\
While\ ||F_x|| &> \tau_r \vec{r} + \tau_a & n\ iterations \\
J &= F'(\vec{x}) & N\ FLOPS \\
\vec{s} &= -F(\vec{x})(F'(\vec{x}))^{-1} & Depends\ on\ how\ \vec{s}\ is\ solved.\ See\ Below \\
\vec{x} &= \vec{x} + \vec{s} & N\ FLOPS \\
F_x &= F(\vec{x}) & N^2\ FLOPS \\
end &
\end{aligned}$$

The cost per iteration for Newton's method is highly dependent upon how $\vec{s}$ is solved. From Homework 1, we know that the Conjugate Gradient method requires $\approx \mathcal{O}(160N)$ FLOPS to reduce the error of the solver by a factor of 10^{-3}. This value will be used as a rough estimate for the remained of this problem. Thus, adding up all of the FLOPS, it requires $\mathcal{O}(N^2)$ FLOPS to set up the problem, and $\mathcal{O}(N^2+162N)$ FLOPS per n iterations. To get to the ϵ accuracy, it would take $\mathcal{O}((\frac{log(\epsilon)}{log(k||\vec{e}_0||^2)})(N^2 + 162N) + N^2)$ FLOPS.

Question 4

Answer the questions in the previous problem for the chord method.

Much of the previous problem's analysis carries over for this problem. One of the main differences between the Chord method and Newton's method is the Chord method converges at a linear rate in the area local to the solution. Because of this, the error equation presented in Problem 3 above changes to:

$$\frac{||\vec{e}_n||}{||\vec{e}_0||} = \frac{k||\vec{e}_0||||\vec{e}_n||}{k||\vec{e}_0||} = (||\vec{x}_0 - \vec{x}^*||)^n = \epsilon$$

$$n = \frac{log(\epsilon)}{log(k||\vec{e}_0||)}$$

Because the Chord method is linear, the quadratic term in the denominator changes into a linear term. Assuming that evaluating the function, $F(x)$ costs $\mathcal{O}(N^2)$ FLOPS and evaluating the Jacobian is only $\mathcal{O}(N)$ FLOPS, the number of of FLOPS required to get to an accuracy of ϵ can be calculated.

The breakdown of the FLOPS cost per step for the Chord method is given below:

$$
\begin{aligned}
\vec{r} &= ||F(\vec{x})|| && N^2\ FLOPS \\
F_x &= \vec{r} && N\ FLOPS\ for\ an\ assignment \\
J &= F'(\vec{x}) && N\ FLOPS \\
L,U\ decomposition\ of\ J & && \frac{N^3}{3}\ FLOPS\ for\ Cholesky\ factorization \\
While\ ||F_x|| &> \tau_r \vec{r} + \tau_a && n\ iterations \\
L\vec{y} &= -F(\vec{x}) && Forward\ Substitution\ N^2\ FLOPS \\
U\vec{s} &= \vec{y} && Back\ Substitution\ N^2\ FLOPS \\
\vec{x} &= \vec{x} + \vec{s} && N\ FLOPS \\
F_x &= F(\vec{x}) && N^2\ FLOPS \\
end &
\end{aligned}
$$

Once again, the cost per iteration is dependent on how $\vec{s}$ is solved. Because the L, U decomposition has been calculated prior to the start of the iteration section, the cost is only incurred once. While $\mathcal{O}(\frac{N^3}{3})$ is the worst-case for decomposing the Jacobian, in reality, this value is dependent on the sparsity of the matrix. If the Jacobian is very sparse, the cost for decomposing it will be much smaller. Inside the iteration section, the $\vec{s}$ was solved using forward and backward substitution, but they could be easily calculated using two sequential Conjugate Gradient calls.

The total cost of using the Chord method with Cholesky factorization of the Jacobian and Forward/Backward substitution is $\mathcal{O}((\frac{N^3}{3} + N^2 + 2N) + (\frac{log(\epsilon)}{log(k||\vec{e}_0||)})(3N^2 + N))$ FLOPS. This method, compared to Newton's method, has the potential for costing significantly more FLOPS to converge to a solution. However, if the Jacobian is extremely costly to evaluate, the Chord method has the advantage of only evaluating it once.

Appendix A: Matlab code

AutoRun

```
%% This is the auto running script for homework 2

atol=1e-8;
rtol=1e-8;
maxIt=10000;
%% Problem 2, part a

%Setup the problem
f1= @(x) 2*x.^2-5; % create a function handle for the problem
x0=10; %initialize the x val
xneg=0.99*x0; %for the secant method, x_(-1) val.
 fprintf('%s\n','Run 1');
[Nx1,newCount1]=Newton(f1,x0,atol,rtol);
[Cx1, Ccount1 ] = Chord( f1, x0,atol, rtol,maxIt);
[ Sx1, Scount1  ] = secant( f1, x0,xneg,atol, rtol,maxIt);
%% Problem 2, part b
 fprintf('%s\n','Run 2');

f2= @(x) sin(x)+x;
x0=0.5;
xneg=0.99*x0;

[Nx2,newCount2]=Newton(f2,x0,atol,rtol);
[Cx2, Ccount2 ] = Chord( f2, x0,atol, rtol,maxIt);
[ Sx2, Scount2  ] = secant( f2, x0,xneg,atol, rtol,maxIt);
%% Problem 2, part c

 fprintf('%s\n','Run 3');
f3= @(x) cos(x);
x0=03;
xneg=0.99*x0;


[Nx3,newCount3]=Newton(f3,x0,atol,rtol);
[Cx3, Ccount3 ] = Chord( f3, x0,atol, rtol,maxIt);
[ Sx3, Scount3  ] = secant( f3, x0,xneg,atol, rtol,maxIt);

%% Ploting section figure 1
figure;
x=-20:1:20;
plot(x,f1(x),'DisplayName','Function a')
hold on;
plot(10,f1(10),'r*','DisplayName','Starting Point');
plot(Nx1,f1(Nx1),'c^','DisplayName','Newton''s method End Point');
plot(Cx1,f1(Nx1),'rd','DisplayName','Chord method End Point');
plot(Sx1,f1(Nx1),'k*','DisplayName','Secant method End Point');
plot(x,zeros(length(x),1),'g','DisplayName','Zero-line');

% Create ylabel
ylabel({'Y-Axis'},'FontSize',20,'Interpreter','latex');

% Create xlabel
xlabel({'X-Axis'},'FontSize',20,'Interpreter','latex');
```

```

% Create title
title({'Function a Start and End Points'},'FontSize',20,...
    'Interpreter','latex');

% Create legend
legend1 = legend('show');
set(legend1,'Interpreter','latex','FontSize',16);

hold off;
%% Ploting section Figure 2
figure;
x=-4:.1:4;
plot(x,f2(x),'DisplayName','Function b');

hold on;

plot(0.5,f2(0.5),'r*','DisplayName','Starting Point');
plot(Nx2,f2(Nx2),'c^','DisplayName','Newton''s method End Point');
plot(Cx2,f2(Cx2),'rd','DisplayName','Chord method End Point');
plot(Sx2,f2(Sx2),'k*','DisplayName','Secant method End Point');

plot(x,zeros(length(x),1),'g','DisplayName','Zero-line');
% Create ylabel
ylabel({'Y-Axis'},'FontSize',20,'Interpreter','latex');

% Create xlabel
xlabel({'X-Axis'},'FontSize',20,'Interpreter','latex');

% Create title
title({'Function b Start and End Points'},'FontSize',20,...
    'Interpreter','latex');

% Create legend
legend1 = legend('show');
set(legend1,'Interpreter','latex','FontSize',16);

hold off;
%% Ploting section Figure 3
figure;
x=-8:.1:4;
plot(x,f3(x),'DisplayName','Function c');

hold on;

plot(03,f3(3),'r*','DisplayName','Starting Point');
plot(Nx3,f3(Nx3),'r^','DisplayName','Newton''s method End Point');
% plot(Cx2,f3(Cx2),'rd');
% plot(Sx3,f3(Sx3),'k*');
plot(x,zeros(length(x),1),'g','DisplayName','Zero-line');
% Create ylabel
ylabel({'Y-Axis'},'FontSize',20,'Interpreter','latex');

% Create xlabel
xlabel({'X-Axis'},'FontSize',20,'Interpreter','latex');

% Create title
title({'Function c Start and End Points'},'FontSize',20,...
    'Interpreter','latex');

```

```
% Create legend
legend1 = legend('show');
set(legend1,'Interpreter','latex','FontSize',16);

hold off;


%% Latex out
fprintf('function  &Newton''s & Chord Method & Secant Method %s','\\\hline')
fprintf('a&%.8f& %.8f& %.8f%s \n',Nx1,Cx1,Sx1,'\\\hline')
fprintf('b&%.8f& %.8f& %.8f%s \n',Nx2,Cx2,Sx2,'\\\hline')
fprintf('c&%.8f& %s& %s%s \n',Nx3,'Did Not Converge','Did Not Converge','\\\hline')
fprintf('\n');

fprintf('function  &Newton''s & Chord Method & Secant Method %s\n','\\\hline')
fprintf('a&%d& %d& %d%s \n',newCount1,Ccount1,Scount1,'\\\hline')
fprintf('b&%d& %d& %d%s \n',newCount2,Ccount2,Scount2,'\\\hline')
fprintf('c&%d& %s& %s%s \n',newCount3,'Did Not Converge','Did Not Converge','\\\hline')
fprintf('\n');
```

Newton's Method

```
function [ x, numIts ] = Newton( fhandle, x0,atol, rtol)
%NEWTON Newton's method for non linear systems of equations
%%This method takes in the function handle to the system that needs to be
%%solved, the inital x value, and the tolerance.
%%Since the homework only requires a single nonlinear equation to be
%%sloved, this method was not extended to cover a matrix.

r0=norm(fhandle(x0),inf);
x=x0;
fx=fhandle(x0);
numIts=0;
h=1e-5;

while norm(fx,inf)>rtol*r0+atol
    numIts=numIts+1;
    df=imag(fhandle(x+h*1i))/h;

    %% Remove this for final runtime calcs!
%      df2=(fhandle(x+h)-fx)/h;
%      dabs=abs(df-df2);
%      if(dabs>.001)
%          disp('ERROR WITH THE IMAGINARY STEP!!!');
%      end

    %% Solve for s. Since this is a 1-d problem, we can just devide by df.
    %if this was a multi-dimensional matrix, we would need to use a linear
    %solver to find s.

    s=-fx/df;
    x=x+s;
    fx=fhandle(x);


end
```

```


end
```

Chord Method

```
function [x, numIts ] = Chord( fhandle, x0,atol, rtol,maxIt)
%CHORD The Chord method for non linear systems of equations
%%This method takes in the function handle to the system that needs to be
%%solved, the inital x value, and the tolerance.

r0=norm(fhandle(x0),inf);
x=x0;
fx=fhandle(x0);
numIts=0;
h=1e-5;

    for i=1:length(x)
        jacobian(i)=imag(fhandle(x+h*1i))/h;
    end
    [l,u]=lu(jacobian);


while norm(fx,inf)>rtol*r0+atol && numIts<maxIt
    numIts=numIts+1;

    if(mod(numIts,10000)==0)
        fprintf('%s %d\n','Chord Method: Max number of iterations:',numIts);
    end


    %% Solve for s. Since this is a 1-d problem, we can just devide by df.
    %if this was a multi-dimensional matrix, we would need to use a linear
    %solver to find s.
    y=-fx/l;
    s=y/u;
    x=x+s;
    fx=fhandle(x);


end


end
```

Secant Method

```
function [ x, numIts  ] = secant( fhandle, x0,xneg,atol, rtol,maxIt)
%SECANT The secant method for non linear systems of equations
%%This method takes in the function handle to the system that needs to be
%%solved, the inital x value, and the tolerance.

r0=norm(fhandle(x0),inf);
```

```
x=x0;
x0=xneg;
fx=fhandle(x);
numIts=0;


while norm(fx,inf)>rtol*r0+atol && numIts<maxIt
    numIts=numIts+1;

  if(numIts==633)
      disp('');
  end

    a=(fx-fhandle(x0))/(x-x0);
    xn=x0;
    x0=x;
    x=x0-fx/a;
    fx=fhandle(x);


end


end
```