MATH 6644
Homework 2

Stephan Boettcher

April 11, 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, this performance increase will not be as pronounced in other methods, such as the Chord Method.

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.

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

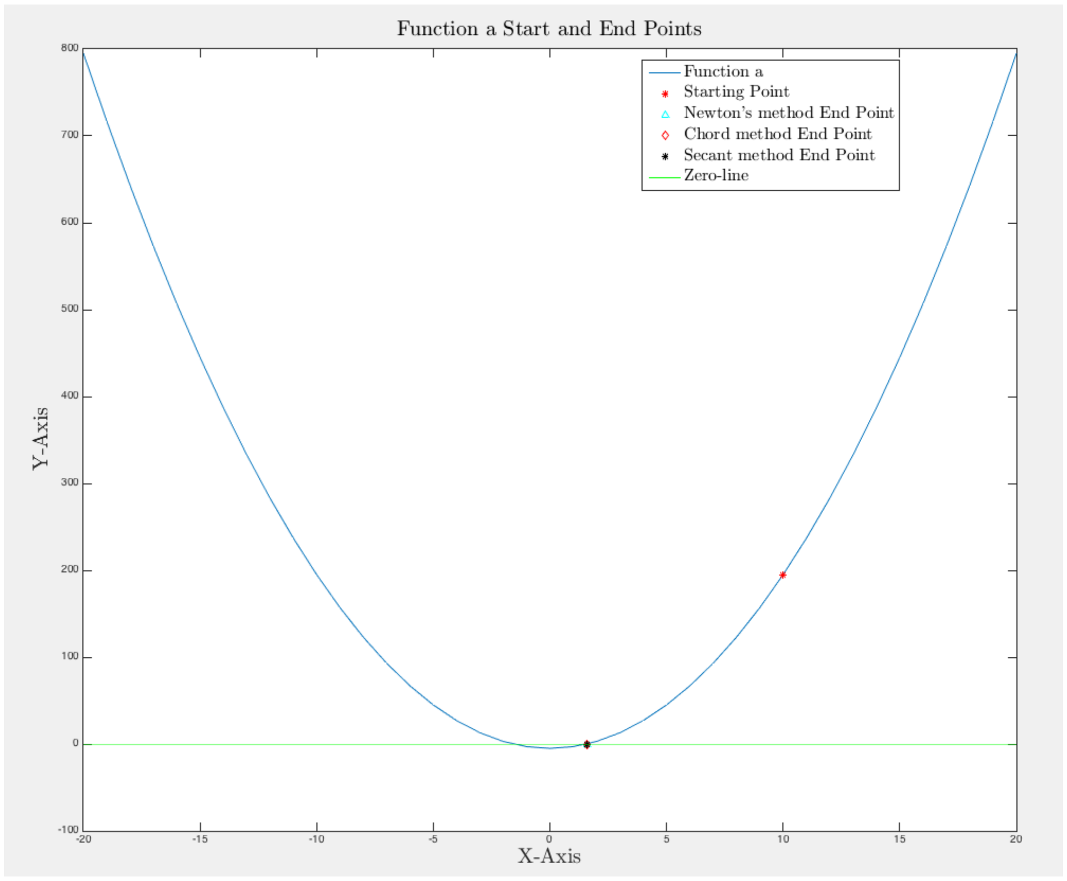

Figure 3: Initial conditions and final answer of Newton's Method, the Chord Method, and the Secant Method for Function a.

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.

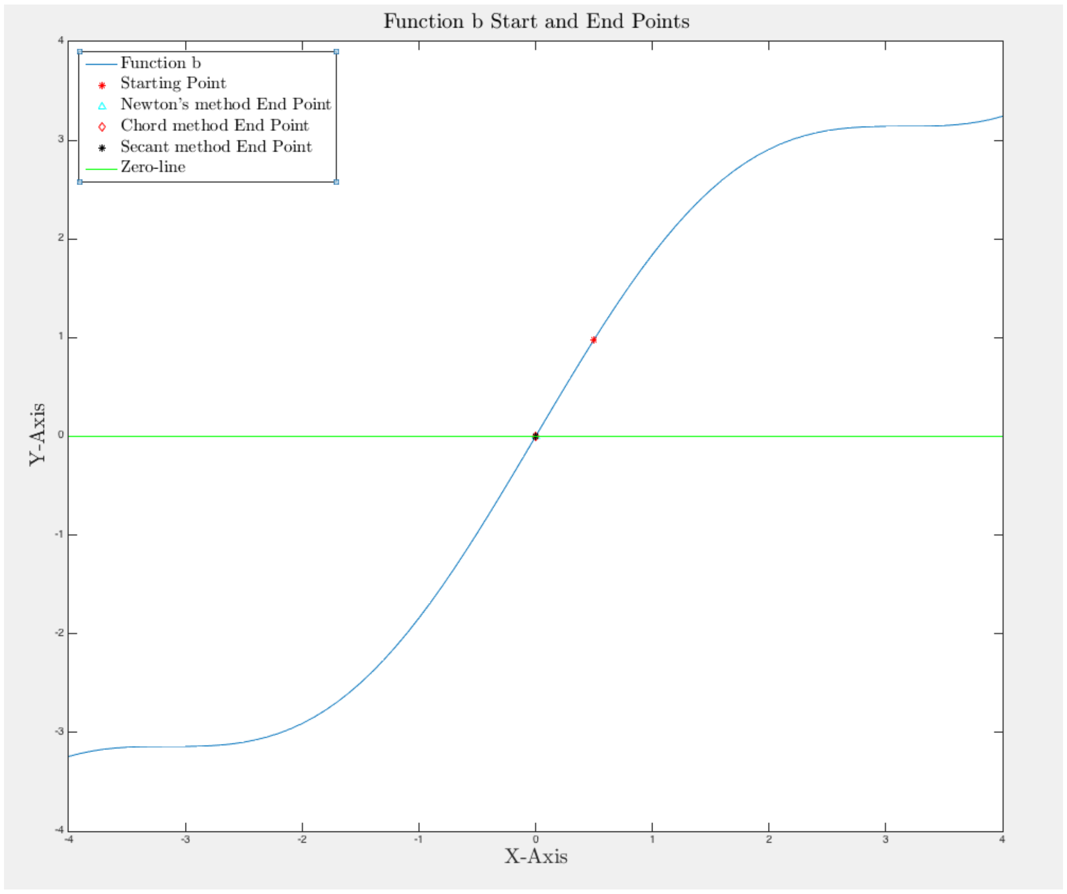

Figure 4: Initial conditions and final answer of Newton's Method, the Chord Method, and the Secant Method for Function b.

However, 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.

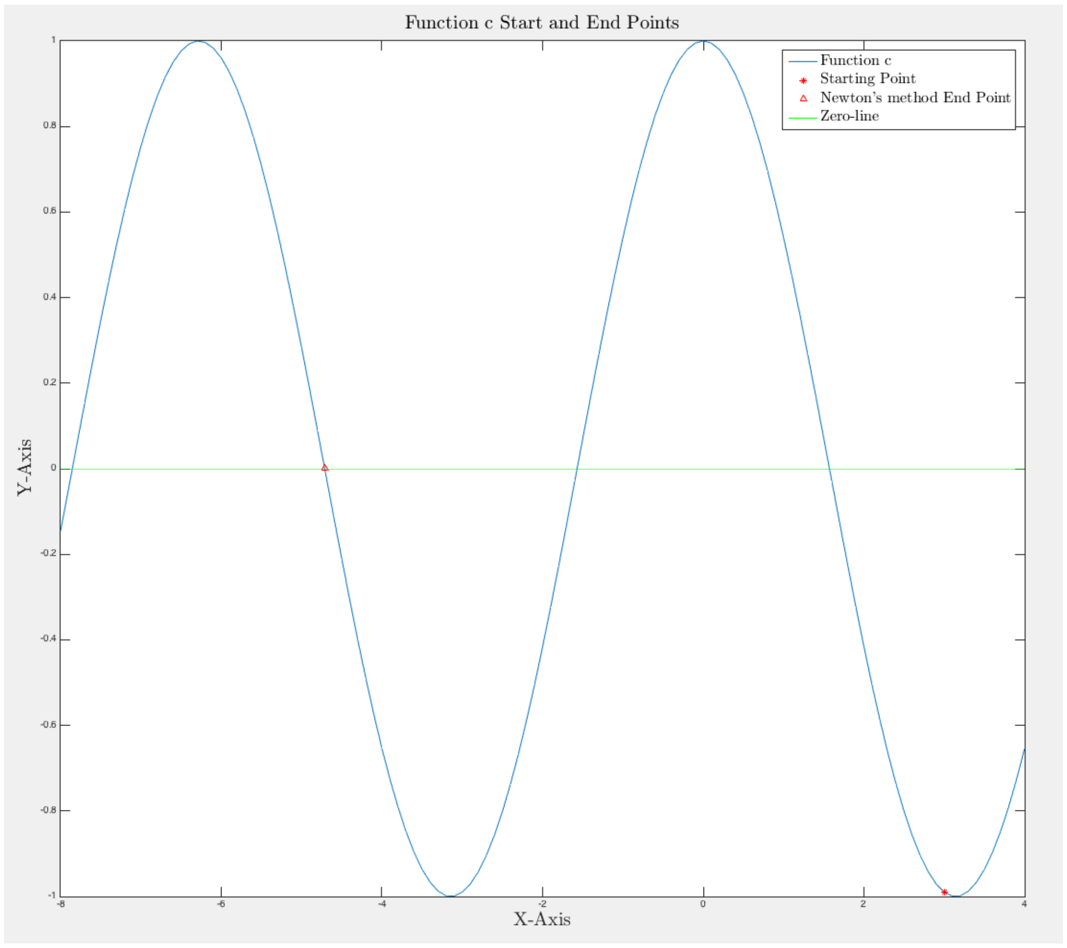

Figure 5: Initial conditions and final answer of Newton's Method, the Chord Method, and the Secant Method for Function c.

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 $||e_n|| \leq ||e_0||$, where ϵ is a small tolerance value. What is the number of floating point operations required to get this accuracy?

Question 4

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

Appendix A: Matlab code

AutoRun

```matlab
%% 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;



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

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


end


end
```