MS4014 Numerical Anaylsis Assignment

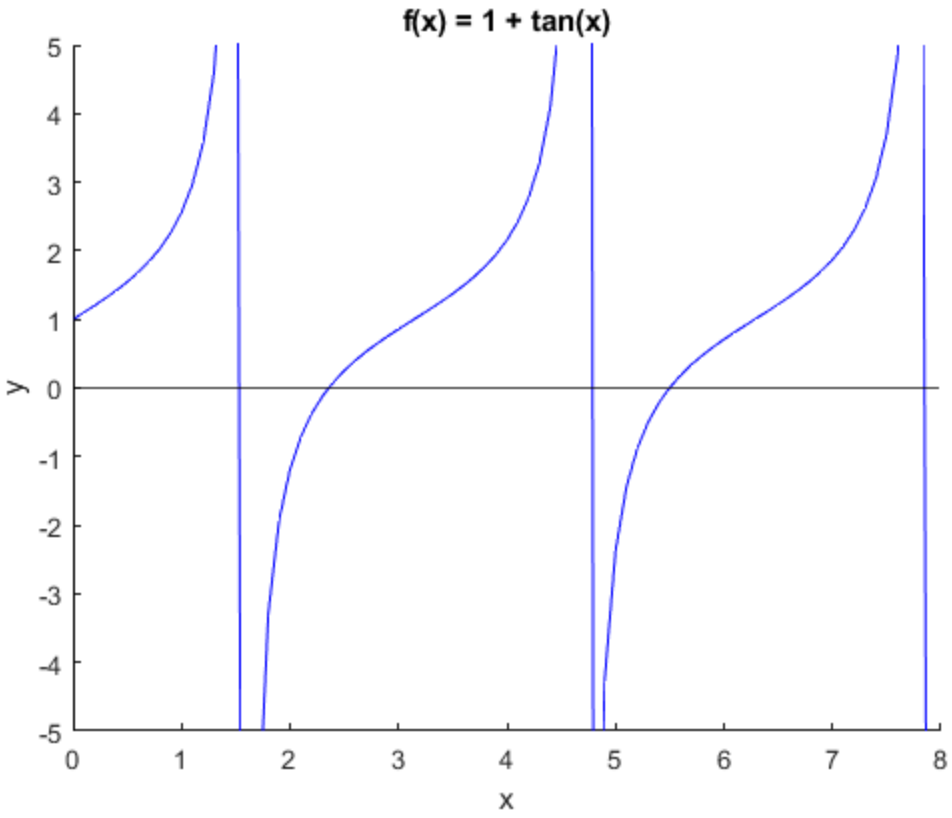
Table of Contents

12/04/2022 20237154 Orla Fitzmaurice

Q 1 (d)

Here is my plot of 1 + tan(x)

```
 hold on;

title("f(x) = 1 + tan(x)");
xlabel("x");
ylabel("y");


yline(0,'k-');
xline(0,'k-');

x = 0:0.1:8;
ylim([-5,5])

y = 1 + tan(x);

plot(x,y,'b')

hold off;
```



Here is my solution to Q1 where you can see the roots x1 = 2.356194490192345 with 6 iterations, and x2 = 5.497787143782138 with 4 iterations

```
format long
f = @(x) 1 + tan(x);
df = @(x) (sec(x))^2;
d2f = @(x) 2*((sec(x))^2)*tan(x);

[x1,iters1] = Halley(2.5,f,df,d2f,1e-12)
[x2,iters2] = Halley(5.5,f,df,d2f,1e-12)
```

```
x1 =

   2.356194490192345


iters1 =

     6


x2 =

   5.497787143782138
```

```
iters2 =

     4
```

Q2

Q2 (a)

Here is my solution to Q2 (a), I have found one set of roots for the system x1 = [0.401077045775061; 1.235485200613828; 3.363437753611112] with 7 iterations.

As you can see from my output, I have found the same two roots twice using different initial guesses. I left this in because no matter what initial guess I tried I kept getting the same roots, just found using different number of iterations.

```
format long
f = @(x) [x(1)+x(2)+x(3)-5;x(1)^2 + x(2)^2 + x(3)^2 - 13;x(1)^3 +
x(2)^3 + x(3)^3 - 40];
J = @(x)
[1,1,1;2*x(1),2*x(2),2*x(3);3*(x(1)^2),3*(x(2)^2),3*(x(3)^2)];

%Roots 1
x1 =  [1;2;3];
[x1,iters] = multnewton(x1,f,J,10e-10,200)

x2 =  [-10;-11;-12];
[x2,iters] = multnewton(x2,f,J,10e-10,200)


x1 =

   0.401077045775061
   1.235485200613828
   3.363437753611112


iters =

     7


x2 =

   0.401077045775061
   1.235485200613827
   3.363437753611112


iters =

    22
```

Q 2 c

Here is my solution to Q2 (c), I used the broyden function I wrote to find four roots for the system: x1 = [-1.597406862934387;2.223704557326811] with 2000 iterations, x2 =[2.633645959250824;0.289513037043148] with 79 iterations, x3 =[1.443556044179248; 1.504516792352986] with 189 iterations, x4 =[-0.709373573280753;-2.092222758400996] with 107 iterations

```
format long
f = @(x) [x(1)^2 + x(1)*(x(2)^3)-7;3*(x(1)^2)*x(2)-(x(2)^3)-6];
J = @(x)
[2*x(1)+(x(2)^3),3*x(1)*(x(2)^2);6*x(1)*x(2),3*(x(1)^2)-3*(x(2)^2)];

%Root 1
x1 =  [-1;3];
[x1,iters] = broyden(x1,f,J,10e-10,2000)

%Root 2
 x2 =  [1;0];
[x2,iters] = broyden(x2,f,J,10e-10,2000)
%Root 3
x3 =  [3;4];
[x3,iters] = broyden(x3,f,J,10e-10,2000)

%Root 4
 x4 =  [-1;-1];
[x4,iters] = broyden(x4,f,J,10e-10,2000)
```

```
x1 =

  -1.597406862934387
   2.223704557326811


iters =

        2000


x2 =

   2.633645959250824
   0.289513037043148


iters =

    79


x3 =
```

```
   1.443556044179248
   1.504516792352986


iters =

   185


x4 =

  -0.709373573280753
  -2.092222758400996


iters =

   107
```

Q3

Here are my solutions to Q3 I used the Jacobi function to find the solution to the system to be xJ =[2.999999046325684;1.999999046325684;0.999999046325684] with 14 iterations.

I used the Gauss Seidel function to find the solution to the system to be xG =[2.999999642372131;1.999999821186066;0.999999955296516] with 8 iterations.

```
%I used the SOR function to find the solution to the system to be
% xS1 = [2.999999992326067;1.999999997219903;1.000000000142202] with
% w=1.05 and 7 iterations and
% xS2 =[3.000000163644878;1.999998913695293;1.000000635667327] with
w=1.8
% and 65 iterations.
%
% When w = 1.8 the SOR method needs way more iterations to converge
%
%
% I found the condition number of A to be 2.571428571428571.
% Since k>1, is large, the matrix is ill-conditioned and
% it can be possible for the residual to be small and the
% error to be large so it can be very diffucult to find
% the correct solution using Gaussian Elimination because a
% small error with have a big effect

A = [4,-1,0;-1,4,-1;0,-1,4];
b = [10;4;2];
x0 = [0;0;0];

xJ = Jacobi(A,b,x0,10e-7)
xG = GaussSeidel(A,b,x0,10e-7)
xS1 = SOR(A,b,x0,1.05,10e-7)
```

```
 xS2 = SOR(A,b,x0,1.8,10e-7)

r = norm(A,inf);
t = norm(inv(A),inf);

%Condition Number of A
k = r*t


iters =

    14


xJ =

   2.999999046325684
   1.999999046325684
   0.999999046325684


iters =

     8


xG =

   2.999999642372131
   1.999999821186066
   0.999999955296516


iters =

     7


xS1 =

   2.999999992326067
   1.999999997219903
   1.000000000142202


iters =

    65


xS2 =

   3.000000163644878
   1.999998913695293
```

```
   1.000000635667327


k =

   2.571428571428571
```

Code

Here are the functions I wrote for this assignment.

For Q1, I wrote this function called Halley.m

```
function [x,iters] = Halley(x0,f,df,d2f,tol)
% Halley's method for f(x) = 0.
%
% input: x0  - initial guess
%        f   - function f
%        df  - derivative of the function f
%        d2f  - 2nd derivative of the function f
%        tol - error tolerance
%
% output: x - the approximate root f(x) = 0
%         iters - the number of iterations required
%

xg = x0;
incr = 1;
iters = 0;


while (abs(incr) > tol)
    incr = -f(xg)/( df(xg)-( (f(xg)*d2f(xg))/2*df(xg) ) );
    xg = xg + incr;

    iters = iters + 1;
end

% Final value
x = xg;
```

For Q2 (a), I adopted this function from labs we completed called multnewton.m

```
function [x,iters] = multnewton(x0,f,J,tol,steps)
% Newton's method for system f(x) = 0.
%
% input: x0  - initial guess vector
%        f   - function f
%        J  - Jacobian
%        tol - error tolerance
```

```
%        steps - the max number of iterations the function is
%                allowed to perform => prevents an infinite loop
%
% output: x - the approximate root f(x) = 0, in vector form
%         iters - the number of iterations required
%

xg = x0;
incr = 1;
iters = 0;

while (abs(incr) > tol && iters<steps)
    Dx = J(xg)\f(xg); % solve for increment
    xg = xg - Dx; % add on to get new guess
    incr = norm(Dx,inf);
    iters = iters+1;
end

x = xg;
```

For Q2 (b), I wrote this function called broyden.m

```
function [x,iters] = broyden(x0,f,J,tol,steps)
% Broyden's method for system f(x) = 0.
%
% input: x0  - initial guess vector
%        f   - function f
%        J  - Jacobian
%        tol - error tolerance
%        steps - the max number of iterations the function is
%                allowed to perform => prevents an infinite loop
%
% output: x - the approximate root f(x) = 0, in vector form
%         iters - the number of iterations required
%
%
incr = 1;
iters = 0;
B = J(x0);

while (abs(incr) > tol && iters<steps)
   s = -B\f(x0);
   x1 = x0 + s;
   y = f(x1) - f(x0);
   r = ((y-B*s)*s')/((s')*s);
   B = B + r;

    incr = norm(r,inf);
    iters = iters+1;
end

x = x1;
```

For Q3, I adopted the function from labs we completed called Jacobi.m

```
function [x,iters] = Jacobi(A,b,x0,tol)

% Solve linear system Ax = b
% using Jacobi iteration
% A is an n by n matrix
% b is an n by 1 vector
% x0 is an n by 1 vector (initial guess)
% x is an n by 1 vector
% tol is error tolerance
%

[n, n] = size(A);     % Find size of matrix A

xold = zeros(n,1);
xnew = zeros(n,1);
xold = x0;
incr = 1;
iters =  0;


while (abs(incr)>tol)

    for i = 1:n
        xnew(i) = (1/A(i,i))*(b(i) - sum(A(i,:)*xold) +
 A(i,i)*xold(i));
    end

    %Errror of Jacobi using max norm
    incr = norm(xnew-xold,inf)/norm(xnew,inf);

    xold = xnew;

    iters = iters +1;
end

x = xnew;
iters
```

For Q3, I adopted the function from labs we completed called GaussSeidel.m

```
function [x,iters]  = GaussSeidel(A,b,x0,tol)

% Solve linear system Ax = b
% using Gauss Seidel iteration
% A is an n by n matrix
% b is an n by 1 vector
% x0 is an n by 1 vector (initial guess)
% x is an n by 1 vector
```

```
% tol is error tolerance

[n, n] = size(A);     % Find size of matrix A

x = zeros(n,1);
incr = 1;
iters = 0;


while (abs(incr)>tol)
    z = x;
    for i = 1:n
      x(i) = (1/A(i,i))*(b(i) - sum(A(i,1:i-1)*x(1:i-1)) - sum(A(i,i
+1:n)*x(i+1:n)));

    end

    %Errror of Gauss Seidel using max norm
   incr = norm(x-z,inf)/norm(x,inf);

    x';
    iters = iters+1;
end
iters
```

For Q3, I adopted the function from labs we completed called SOR.m

```
function [x,iters] = SOR(A,b,x0,omega,tol)

% Solve linear system Ax = b
% using SOR iteration
% A is an n by n matrix
% b is an n by 1 vector
% x0 is an n by 1 vector (initial guess)
% x is an n by 1 vector
% tol is error tolerance
% omega = weighting parameter, 1 < omega < 2

[n, n] = size(A);     % Find size of matrix A

x = zeros(n,1);
x = x0;
incr = 1;
iters = 0;

while (abs(incr)>tol)
        z = x;

    for i = 1:n
      x(i) = (1-omega)*x(i) + (omega/A(i,i))*(b(i) -
 sum(A(i,1:i-1)*x(1:i-1)) - sum(A(i,i+1:n)*x(i+1:n)));
    end
```

```
  %Error of SOR using max norm
  incr = norm(x-z,inf)/norm(x,inf);
  iters = iters +1;

end
iters
```

Published with MATLAB® R2021a