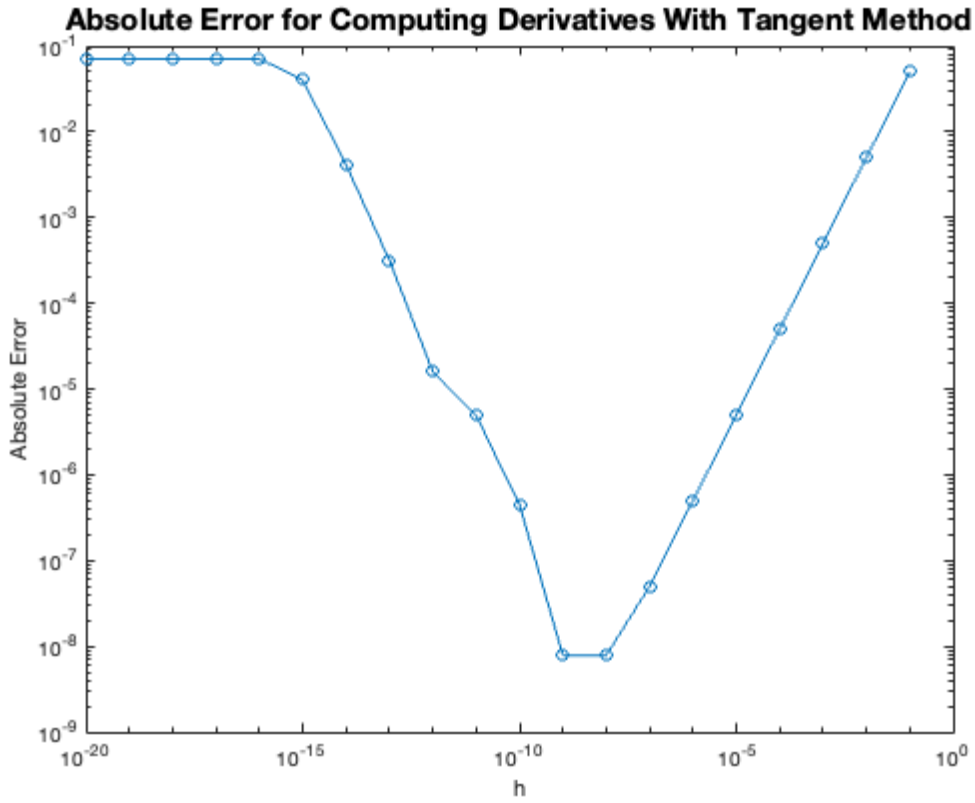
Contents


```
% MATH 512 Proj 1
```

Problem 3

```
y = [];
x0 = 1.5;
h = 10.^(-1.*[1:20]);

for i = 1:length(h)
    y(i) = (sin(x0+h(i))-sin(x0))/h(i);
end

abserror = abs(y-cos(1.5));
figure(1)

loglog(h,abserror,'o-')
title('Absolute Error for Computing Derivatives With Tangent Method',FontSize= 15)
xlabel('h')
ylabel('Absolute Error')

% comment: As we see from figure 1, when the difference, h, decreases, the
% absolute error decreases at first, until h = 1e-8, then the absolute
% error starts to increase. The error converges to a larger value than that
% of the estimation for h = 1e-1. We believe that the round-off error
% contributes the most to the overturn of the error. For the denominator,
% as h goes to 0, 1/h goes to infinity. We lose larger precision while
% truncating the tail of a larger number.
```

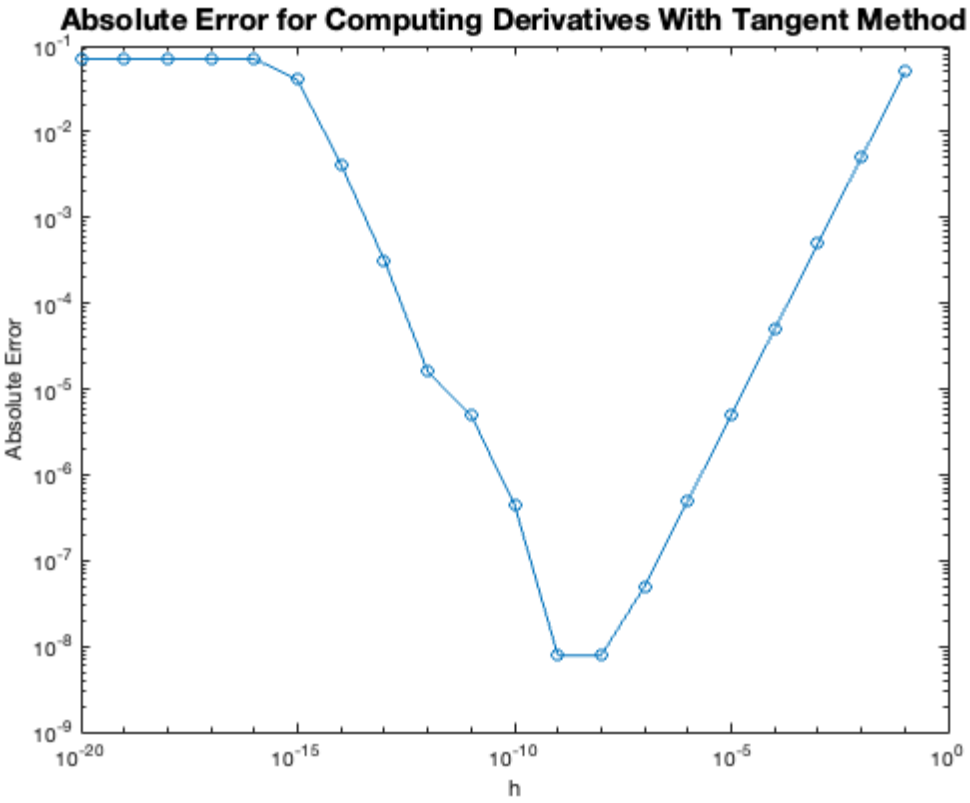


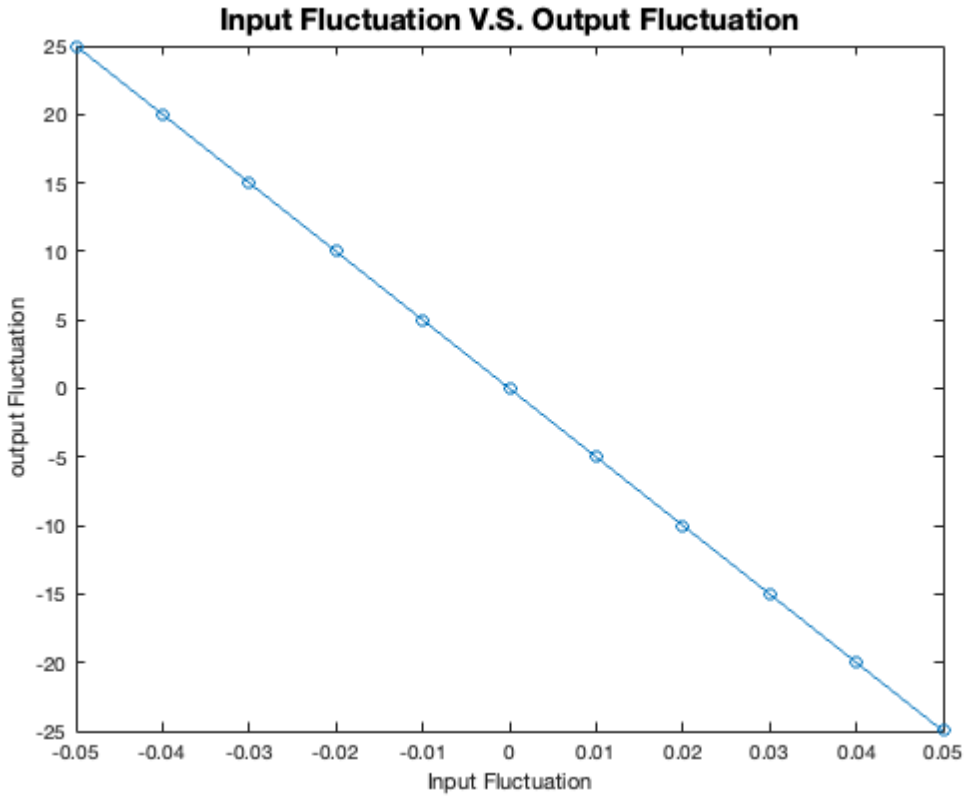
Problem 4

a) The main difficulty in solving the linear system is due to the fact that the matrix is very close to singularity. If we use LU decomposition to solve this linear system, we end up with (a^2-b^2)/a*y = b/a in the last row. Besides the round-off error in dividing a very small number (a^2-b^2)/a, the system is also unstable. If we fluctuate the column vector on the right hand side, say [1, epsilon]', the value for y would change by epsilon*a/(a^2-b^2), which could be much larger than epsilon. Now let's try some examples to illustrate the instability:

```
% example 1
a = 10; b = 9.999;
z = [1,0]';
A = [a,b;
     b,a];
x_y = A\z;
epsilon = 0.01*(-5:5);
x_y_fluc = zeros(2,length(epsilon));

z_fluc = z;

for i = 1:length(epsilon)
    z_fluc(2) = z(2)+epsilon(i);
    x_y_fluc(:,i) = A\z_fluc-x_y;
end

figure(2)
plot(epsilon,x_y_fluc(1,:),'-o')
title('Input Fluctuation V.S. Output Fluctuation',FontSize=15)
xlabel('Input Fluctuation')
ylabel('output Fluctuation')
% As figure 2 shown, the fluctuation of the output is much larger than the
% fluctuation in input.
```

```
% b) x+y = 1/(a+b) would be a stable algorithm given that a+b are not too
% small.

x_add_y = 1/(a+b);

% c) The statement is false. The problem of solving the linear system is
% ill-conditioned as we proved analytically in (a). While for computing x+y
% unless we specify what algorithm we use, we can't state that it is not
% ill-conditioned. For example, if we first solve the linear system and
% then compute x+y, the round-off would also affect our solution.
```





Problem 5

```
% Finding root of the function f(x) = sqrt(x)-1.1 by bisection. We can pick
% a = 0 and b = 2 to start with. f(0) = -1.1 and f(2) > 0.
```

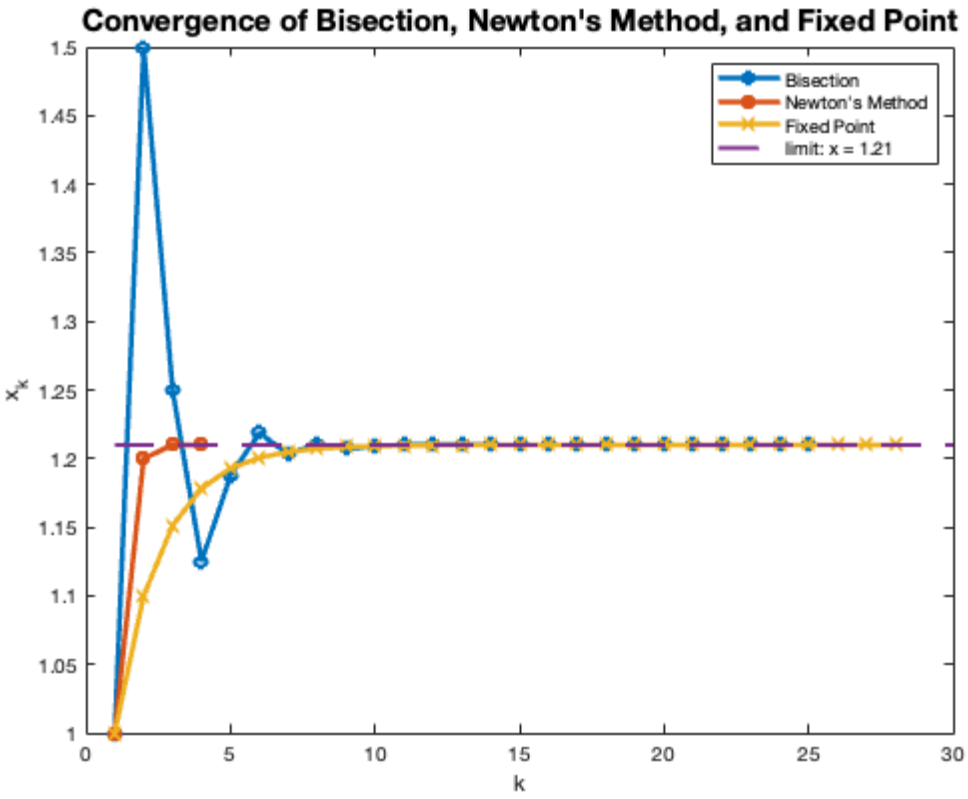
```
a = 0; b = 2;
iter = 1;
x_bisec = [1];
% see the definition of f1 in the last section
while abs(f1((a+b)/2)) > 1e-8
    if f1((a+b)/2) > 0
        b = (a+b)/2;
    else
        a = (a+b)/2;
    end
    iter = iter+1;
    x_bisec(iter) = (a+b)/2;
end

% a) total iteration: 24
disp(iter-1)
% it is close to our expectation which is -log2(2e-8)+1 ~ 26

% b) absolute error ~ 2.1458e-8
disp(abs(1.21-(a+b)/2))
% Convergence analysis can predict the range of the error. Given the
% iteration = 24, our interval length is 2^-23 ~ 1.2e-7, which upper-bounds
% the absolute error.
```

```
    24

   2.1458e-08
```

Problem 6

```
% Newton's method:
x_N = [1]; % start from 1
iter = 1;
while abs(f1(x_N(iter))) > 1e-8
    x_N(iter+1) = x_N(iter) - f1(x_N(iter))/df1(x_N(iter));
    iter = iter+1;
end


% Fixed point method:
% Note that we should try to pick a function that the
% derivative close to the fixed point is less than 1 (locally lipschiz). So
% we choose  f2(x) = x - f1(x) so that when f1(x) = 0, f2(x) = x. A
% divergent example would be g(x) = x + f1(x). In that case, fixed point
% method will diverence for any starting point except the fixed point
% itself.

% Start from 1
 x_fp = [1];
 iter = 1;
 while abs(f1(x_fp(iter))) > 1e-8
     x_fp(iter+1) = f2(x_fp(iter));
     iter = iter + 1;
 end

figure(3)
```

```
plot(x_bisec,'-o',LineWidth=2.5)
hold on
plot(x_N,'-s',LineWidth=2.5)
plot(x_fp,'-x',LineWidth=2.5)
plot(1.21*ones(1,30),'--',LineWidth=2)

xlabel('k')
ylabel('x_k')
title("Convergence of Bisection, Newton's Method, and Fixed Point",FontSize=15)
legend('Bisection',"Newton's Method",'Fixed Point','limit: x = 1.21')

% Comments: As figure 3 shown, bisection method can have larger error in
% the next iteration, although the range of the error is decreasing by 1/2
% every iteration. For Newton's method, it converges really fast when we
% picked a good starting point. In this case, x_0 = 1 is already very close
% to the root. For fixed point method, it appoarches the root gradually
% from one side with a decreasing convergence rate.
```

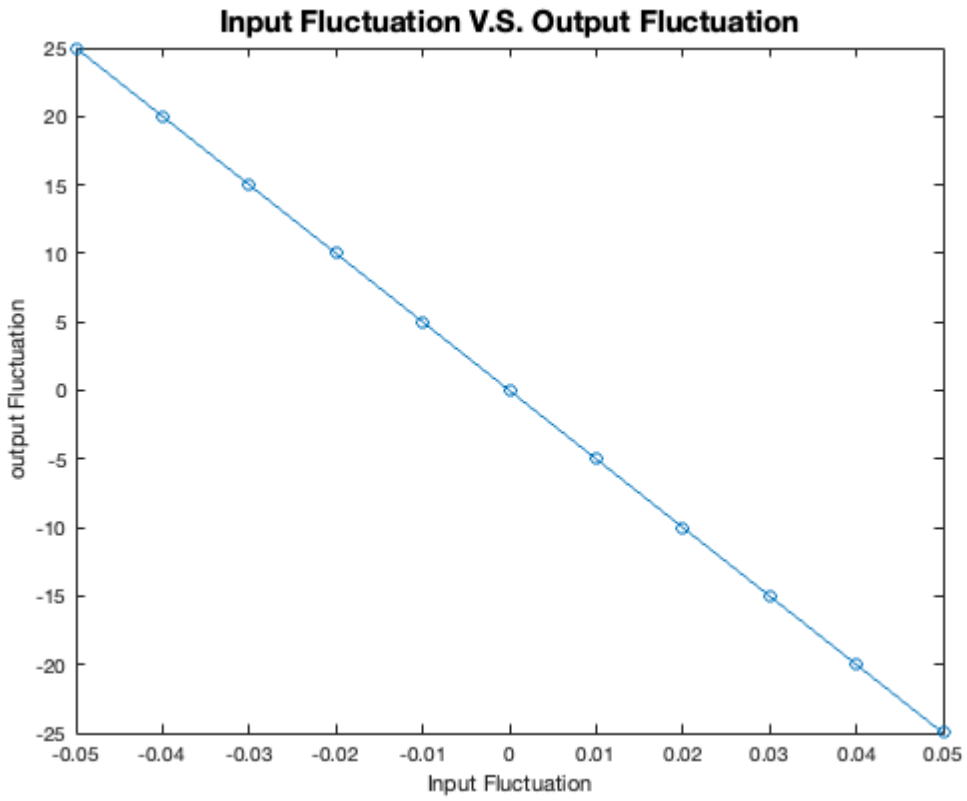


Problem 7

The definition of the function 'tridiag' is in the last section. Test function 'tridiag' with some examples.

```
n = 30;

A1 = rand(1)*10*rand(1,n-1); % lower diagonal
A2 = rand(1)*10*rand(1,n); % diagonal
A3 = rand(1)*10*rand(1,n-1); % upper diagonal
b = (rand(1)*20*rand(1,n))';

A = diag(A1,-1)+diag(A2)+ diag(A3,1);
```

```
x_matlab = A\b;
x_my = tridiag(A1,A2,A3,b);

disp(max(abs(x_my-x_matlab)))
```



Functions

Problem 7

```
function x = tridiag(A1,A2,A3,b)
    n = length(A2);
    if length(A1) ~= n-1 || length(A3) ~= n-1 || length(b) ~= n
        x = nan;
        return
    end
    x = zeros(n,1);
    % Gaussian elimination
    for i = 1:n-1
        A2(i+1) = A2(i+1) - A3(i)*A1(i)/A2(i);
        b(i+1) = b(i+1) - b(i)*A1(i)/A2(i);
    end

    % Back substitution
    x(n) = b(n)/A2(n);
    for i = n-1:-1:1
        x(i) = (b(i)-x(i+1)*A3(i))/ A2(i);
    end
end

% Problem 5 & 6
function y = df1(x)
```

```
    y = 1/2*x^(-1/2);
end

function y = f2(x)
    y = x-f1(x);
end

function y = f1(x)
    y = sqrt(x)-1.1;
end
```

```
    2.2027e-13
```

Published with MATLAB® R2021b