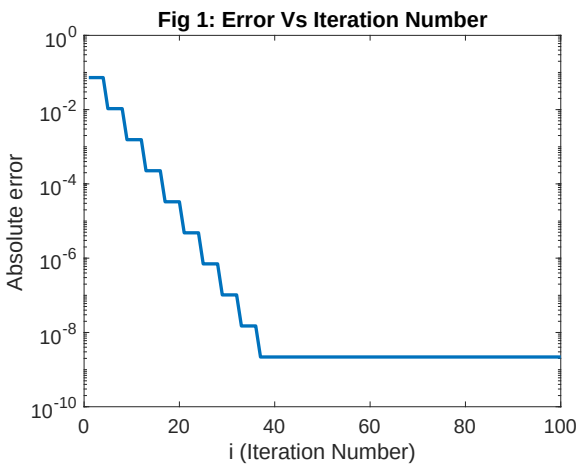
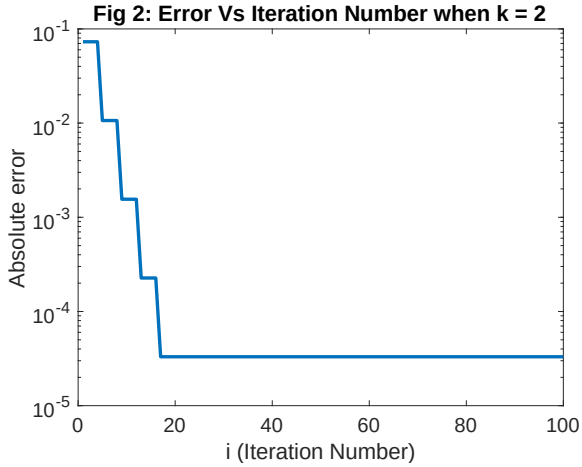
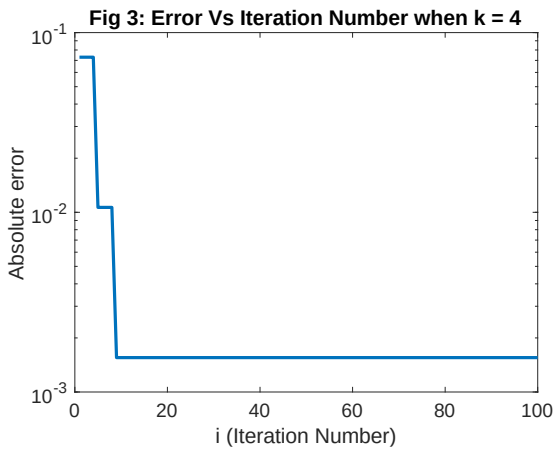
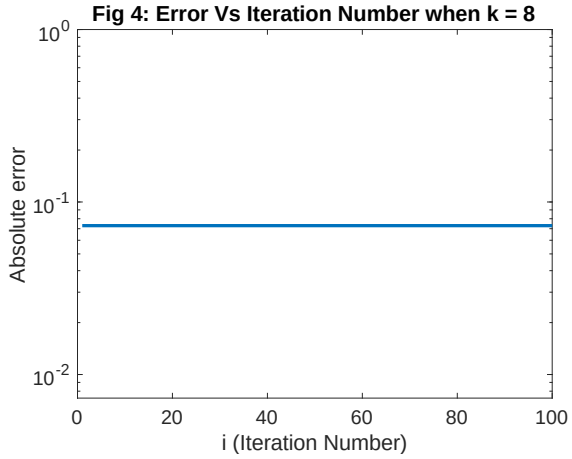




As depicted in Fig 1, when f(x) = -((cos(x) / (1 + x^2))) in the algorithm, the error decreases slowly with increasing i iteration numbers. The error starts out at a value of approximately 10^{-1} when i = 1 and decreases until it reaches a lower bound of 10^{-9} when i = 38. The accuracy of the algorithm increases with increasing i until it reaches its greatest when i = 38 when the error is at its smallest (the algorithm is most accurate for i >= 38). The algorithm is not efficient because it takes a large number of i iterations for the error to become small. Each decrease in error happens with 4 iterations rather than with every single iteration. The algorithm is not robust because the absolute error in the algorithm reaches a lower bound of 10^{-9} which is still nowhere near as small as the bound imposed by machine epsilon at 10^{-16}.





When f(x) is changed to –((cos(x^k) / (1 + x^{2k}))) as depicted in Fig 2, 3 and 4, the error will get larger with increasing k values until it reaches and remains at approximately $10^{-1.1}$ when k >= 8. The accuracy of the algorithm is very poor because it gets worse as k increases as shown by the increase in the error. Efficiency is good because as k increases, the algorithm takes less iterations to decrease the error down to its lower bound. For example, when k = 4 in Fig 2, the algorithm takes 10 iterations to drop the error down to its lower bound of 10^{-3} whereas when k = 8 in Fig 4, the algorithm already starts off with the error being at its lower bound of $10^{-1.1}$ when i = 1.

When we compare the algorithm when k = 1 and k = 4 in Fig 1 and 3 respectively, the function converges quicker to an incorrect point (larger error) when k = 4. Thus, the robustness of the algorithm deteriorates with increasing k values. If we consider the Taylor series expansion of f(x), larger k values means taking more derivatives of f. This results in many small terms close to 0 in the series when the derivatives are evaluated at x* and we incur a large round-off error when these terms are summed up. As (x – x*) gets closer to machine epsilon, floating-point issues arise because term coefficients get smaller resulting in smaller numbers for the terms in the series.

MATLAB Code

```
%Case 1: f(x) = -((cos(x) / (1 + x^2)))

f = @(x) -((cos(x) / (1+x.^2)));

min_xstar = 0;
gamma = ((3 - sqrt(5)) / 2);


triple = [-1, 1/2, 1];


N = 100;
i = 1:N;
error = zeros(N,1);

for j = 1:N

    an = triple(1);
    bn = triple(2);
    cn = triple(3);

    disp(triple);

    %1. Choose a new point x using the formula

    if ( (cn - bn) > (bn - an) )
        x = bn + gamma*(cn - bn);
    elseif ( (cn - bn) < (bn - an) )
        x = bn + gamma*(an - bn);
    end

    %2. Update the triple using the formula

    if (x < bn && f(x) < f(bn))
        triple(3) = bn;
        triple(2) = x;

    elseif( x > bn && f(x) < f(bn))
        triple(1) = bn;
        triple(2) = x;

    elseif( x < bn && f(x) > f(bn))
        triple(1) = x;

    elseif( x > bn && f(x) > f(bn))
        triple(3) = x;

    end
```

```
error(j) = abs(min_xstar - triple(2));

end

figure(1);
semilogy(i, error, 'linewidth', 3.0)
xlabel('i (Iteration Number)','fontsize',18)
ylabel('Absolute error','fontsize',18)
title('Fig 1: Error Vs Iteration Number','fontsize',18)
grid on;
set(gca,'fontsize',18)



% Case 2: f(x) = -((cos(x^k) / (1 + x^2*k)))

k = 2;
%k = 4;
%k = 8;
f = @(x) -((cos(x.^k) / (1 + x.^(2*k))));

min_xstar = 0;
gamma = ((3 - sqrt(5)) / 2);

triple = [-1, 1/2, 1];

N = 100;
i = 1:N;
error = zeros(N,1);

for j = 1:N

    an = triple(1);
    bn = triple(2);
    cn = triple(3);

    disp(triple);

    %1. Choose a new point x using the formula

    if ( (cn - bn) > (bn - an) )
        x = bn + gamma*(cn - bn);
    elseif ( (cn - bn) < (bn - an) )
        x = bn + gamma*(an - bn);
    end

    %2. Update the triple using the formula

    if (x < bn && f(x) < f(bn))
        triple(3) = bn;
        triple(2) = x;
```

```
    elseif( x > bn && f(x) < f(bn))
        triple(1) = bn;
        triple(2) = x;

    elseif( x < bn && f(x) > f(bn))
        triple(1) = x;

    elseif( x > bn && f(x) > f(bn))
        triple(3) = x;

    end

error(j) = abs(min_xstar - triple(2));

end

semilogy(i, error, 'linewidth', 3.0);
xlabel('i (Iteration Number)','fontsize',18);
ylabel('Absolute error','fontsize',18);
title(['Fig 4: Error Vs Iteration Number when k = '
num2str(k)],'fontsize',18);
grid on;
set(gca,'fontsize',18);
```