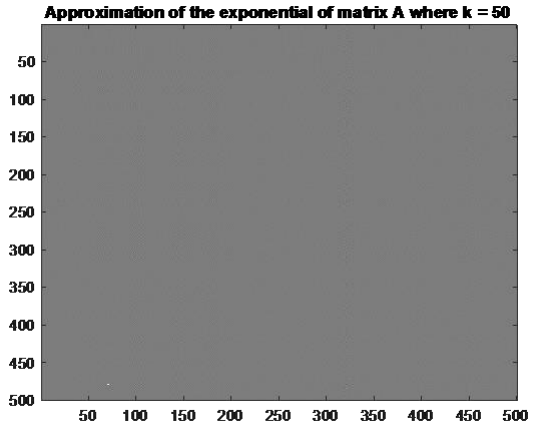
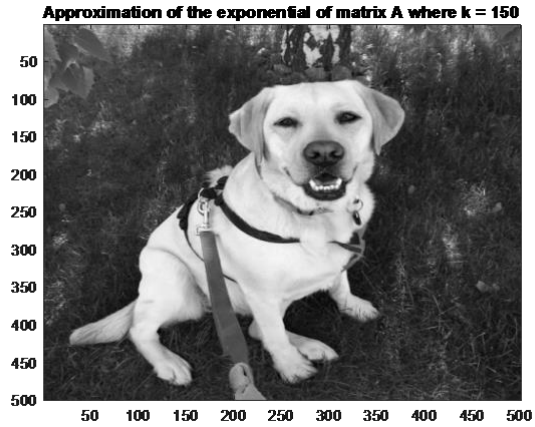
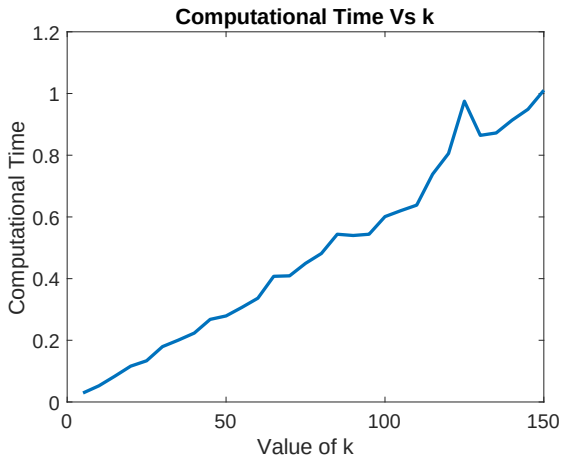
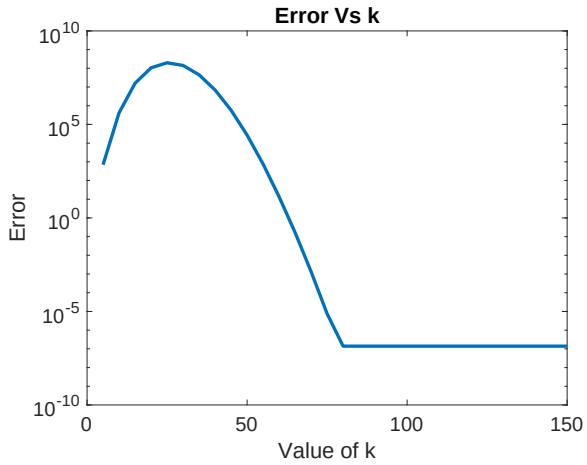
(a)





As depicted in the top left figure, output accuracy is very low when the exponential of matrix A is approximated using only the summation of the first 50 terms of the Taylor series expansion. When the number of terms in the Taylor series expansion is increased to 150, however, we get a more accurate approximation of the exponential of matrix A as shown in the top right figure. Therefore, we can deduce that approximating matrix A with more terms in the Taylor series expansion increases approximation accuracy and thus yields higher output accuracy. The trade-off, however, is that an algorithm approximating matrix A with more terms will take longer to run.





(b) As shown in the Computational Time vs k figure, the computational time depends on k by increasing almost linearly as the value of k increases. This agrees with counting the number of flops in my algorithm because firstly, my algorithm iteratively initializes expAk to a 500x500 matrix along with a 500x500 identity matrix I_mat. Secondly in a nested loop, my algorithm takes the sum of expAk and I_mat iteratively which takes 1 flop per summation. Thirdly, I_mat is set to A * I_mat divided by the current iteration number. This third step is the matrix multiplication part of the algorithm and will take n*n*(2n-1) = 500*500*(2*500-1) flops. Because n = 500, the total number of flops in my algorithm will be 250000*(999)*k which will grow linearly as k increases.

(c) As depicted in the Error vs k figure, the approximation error is very large when k < 75. The error decreases significantly as k increases from ~25 to 75. The accuracy of my algorithm is reasonable because k represents the number of terms in the Taylor series expansion and a larger k value means a better approximation of the exponential of matrix A. A better approximation means a lower error which is consistent with the Error vs k figure. With regards to robustness, my algorithm is not very robust in comparison to the machine epsilon because the error decreases to and balances out at a value of 10^{-4} which is still not as robust as 10^{-16} (value of the machine epsilon).

MATLAB Code

```
Part a

%CA3matrix will be matrix A
load("CA3matrix.mat");


%Case 1: k = 50
I_mat = eye(500,500);
expAk = zeros(500,500);
expAk = expAk + I_mat;

k = 50;

for i = 1:k
    expAk = expAk + (A^i)/factorial(i);
end

imagesc(real(expAk));
colormap gray;
title('Approximation of the exponential of matrix A where k = 50');
set(gca,'fontsize',14);

%Case 2: k = 150
expAk = zeros(500,500);
expAk = expAk + I_mat;

k = 150;

for i = 1:150
    expAk = expAk + (A^i)/factorial(i);
end

imagesc(real(expAk));
colormap gray;
title('Approximation of the exponential of matrix A where k = 150');
set(gca,'fontsize',14);

Part b


load('CA3matrix.mat');
k = 5:5:150;
times = zeros(length(k),1);

for i = 1:length(k)
    tic;
    I_mat = eye(500,500);
    expAk = zeros(500,500);

    for j = 1:k(i)
```

```
        expAk = expAk + I_mat;
        I_mat = (A*I_mat) / j;
    end
    times(i) = toc;
end



plot(k,times,'linewidth',3.0)
xlabel('Value of k','fontsize',18);
ylabel('Computational Time','fontsize',18);
title('Computational Time Vs k');
grid on;
set(gca,'fontsize',18);

Part c

load('CA3matrix.mat');
k = 5:5:150;
expA = expm(A);
errors = zeros(length(k),1);

for i = 1:length(k)
    I_mat = eye(500,500);
    expAk = zeros(500,500);

    for j = 1:k(i)
        expAk = expAk + I_mat;
        I_mat = (A*I_mat)/j;
    end
    errors(i) = norm(expA - expAk)/ norm(expA);

end

semilogy(k,errors,'linewidth',3.0)
xlabel('Value of k','fontsize',18);
ylabel('Error','fontsize',18);
title('Error Vs k');
grid on;
set(gca,'fontsize',18);
```