MA398 Assignment 1

1.

a) Proving absolute convergence in operator norm should be sufficient $\in$ finite complex vectorspace ~~Cn~~

$\|T\|_{op}$ - represents operator norm on T

st.

$$\|T\|_{op} = \sup_{\|x\| \leq 1} \|Tx\|$$

$\|x\|$ is norm on $\mathbb{C}^n$

By sub-multiplicitivity:

(i.e. $\|AB\|_m \leq \|A\|_m \|B\|_m$)

$$\|T^k\|_{op} \leq \|T\|_{op} \|T\|_{op} \ldots \underbrace{\|T\|_{op}}_{k \text{ times}}$$

$$= \|T\|_{op}^k$$

So (PTO)

So:

$$\left\| \sum_{k=0}^{+\infty} \frac{1}{k!} A^k \right\|_{op} \leq \sum_{k=0}^{+\infty} \frac{1}{k!} \|A^k\|_{op}$$

$$\leq \sum_{k=0}^{+\infty} \frac{\|A\|_{op}^k}{k!} < +\infty$$

(by standard convergence of exponential)

To be more thourough - to prove the convergence of the infinite sum, we must show that for some finite α, β with $\alpha \leq \beta$:

$$\left\| \sum_{k=\alpha}^{\beta} \frac{1}{k!} A^k \right\|_{op} \leq \sum_{k=\alpha}^{\beta} \frac{\|A^k\|_{op}}{k!} \leq \sum_{k=\alpha}^{\beta} \frac{\|A\|_{op}^k}{k!}$$

i.e. the finite ends of matrix exponential series* by the finite ends of the exponential series with complex arguments $\rightarrow 0$

* are dominated in the operator norm

b) $A = S^{-1} \Lambda S$

$$\exp(A) = \sum_{k=0}^{+\infty} \frac{1}{k!} A^k$$

$$\cancel{\exp} = I + A^1 + \frac{A^2}{2} + \frac{A^3}{3} + \dots$$

$$= I + (S^{-1}\Lambda S)^1 + \frac{(S^{-1}\Lambda S)^2}{2} + \frac{(S^{-1}\Lambda S)^3}{3} + \dots$$

$A^n = S^{-1} \Lambda^n S$

$$\Rightarrow \exp(A) = I + S^{-1}(\Lambda)^1 S + \frac{S^{-1}\Lambda^2 S}{2} + \frac{S^{-1}\Lambda^3 S}{3} + \dots \quad (*)$$

If we factorise S^{-1}, S:

$$\Rightarrow \exp(A) = S^{-1}\left(I + \Lambda + \frac{\Lambda^2}{2} + \frac{\Lambda^3}{3} + \dots\right) S$$

$$= S^{-1} \exp(\Lambda) S \quad (= \exp(A))$$

$$\Rightarrow \exp(A) = S^{-1}\left(\sum_{k=0}^{+\infty} \frac{1}{k!} \Lambda^k\right) S$$

c) Define $\alpha, \beta \in \mathbb{C}$

$$\exp(A(\alpha+\beta)) = \exp(A\alpha)\exp(A\beta)$$

$\Rightarrow$ from definition we get:

$$\left(I + A\alpha + \frac{(A\alpha)^2}{2} + \dots\right)\left(I + A\beta + \frac{(A\beta)^2}{2} + \dots\right)$$

$$= \left(\sum_{j=0}^{+\infty} \frac{(A\alpha)^j}{j!}\right)\left(\sum_{k=0}^{+\infty} \frac{(A\beta)^k}{k!}\right)$$

(cauchy product)

$$= \sum_{j=0}^{+\infty}\sum_{k=0}^{+\infty}\left(\frac{A^{j+k}\alpha^j\beta^k}{j!\,k!}\right)$$

Define $n = j+k \Rightarrow k = n-j$

$\Rightarrow \exp(A\alpha)\exp(A\beta)$

$$= \sum_{j=0}^{+\infty}\sum_{n-j}^{+\infty}\left(\frac{A^n\alpha^j\beta^{n-j}}{j!\,n-j!}\right) = \sum_{n=0}^{+\infty}\frac{A^n}{n!}\sum_{j=0}^{n}\left(\frac{n!\,\alpha^j\beta^{n-j}}{j!\,(n-j)!}\right)$$

$$= \sum_{n=0}^{+\infty}\frac{A^n(\alpha+\beta)^n}{n!} = \exp(A(\alpha+\beta))$$

PTO

Let's set $\alpha = 1$, $\beta = -1$

we get:

$$\exp(A)\exp(-A) = \exp(A(1+(-1)))$$

$$= \exp(\underline{0}) = I$$

$\Rightarrow$ $\exp(A)$ has an inverse $= \underline{\exp(-A)}$

d)

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

in $S^{-1} \Lambda S$ form:

$$\underset{S^{-1}}{\begin{pmatrix} -i & i \\ 1 & 1 \end{pmatrix}} \underset{\Lambda}{\begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix}} \underset{S}{\begin{pmatrix} \frac{i}{2} & \frac{1}{2} \\ -\frac{i}{2} & \frac{1}{2} \end{pmatrix}}$$

(This is actually $S \Lambda S^{-1}$ but $S \Lambda S^{-1} = S^{-1} \Lambda S$)

$$\exp(A) = S^{-1} \exp \cancel{A}(\Lambda) S$$

$$= \begin{pmatrix} -i & i \\ 1 & 1 \end{pmatrix} \exp \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix} \begin{pmatrix} \frac{i}{2} & \frac{1}{2} \\ -\frac{i}{2} & \frac{1}{2} \end{pmatrix}$$

$$= \begin{pmatrix} e^{i} & 0 \\ 0 & e^{-i} \end{pmatrix}$$

$$\cos x = \frac{e^{ix} + e^{-ix}}{2}$$

$$\sin x = \frac{e^{ix} - e^{-ix}}{2i}$$

$$\cos x + i \sin x = \cancel{2} e^{ix}$$

$$\cos x - i \sin x = \cancel{2} e^{-ix}$$

set $x = 1 \Rightarrow e^{i} = \cos 1 + i \sin 1$ (radians)

$e^{-i} = \cos 1 - i \sin 1$

PTO

$$\Rightarrow \exp(\Lambda) = \begin{pmatrix} \cos 1 + i\sin 1 & 0 \\ 0 & \cos 1 \cancel{+} - i\sin 1 \end{pmatrix}$$

$$\Rightarrow \exp(A) = \begin{pmatrix} -i & i \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \cos 1 + i\sin 1 & 0 \\ 0 & \cos 1 - i\sin 1 \end{pmatrix} \begin{pmatrix} \frac{i}{2} & \frac{1}{2} \\ -\frac{i}{2} & \frac{1}{2} \end{pmatrix}$$

$$\boxed{= \begin{pmatrix} \cos 1 & \sin 1 \\ -\sin 1 & \cos 1 \end{pmatrix}} \quad (*)$$

Geometrical meaning of $(*)$

This is ~~a~~ clockwise rotation of 1 radian

$$\exp(tA) = \sum_{k=0}^{+\infty} \frac{t^k A^k}{k!} = \begin{pmatrix} \cos t & \sin t \\ -\sin t & \cos t \end{pmatrix} \quad (**)$$

$\Rightarrow$ Geometrical meaning of $(**)$:

Clockwise rotation of t radians

e)

Type B $= \begin{pmatrix} 0 & b \\ -b & 0 \end{pmatrix}$

Trivially:

$$\exp(tB) = \begin{pmatrix} \cos(bt) & \sin(bt) \\ -\sin(bt) & \cos(bt) \end{pmatrix}$$

and Type C

$$\exp(tA) = \begin{pmatrix} e^{at}\cos(bt) & e^{at}\sin(bt) \\ -e^{at}\sin(bt) & e^{at}\cos(bt) \end{pmatrix}$$

---

f) MA398_Assignment1_Exercise1f.m

some notes about the code are commented in

```matlab
%initialisation
Y = [];
m = 20;
%if m >= 17, the truncation error is so small, it is determined as 0 by
%matlab *


for n = 0:m
    %more initialisation
    A = [0,1;-1,0];
    sum = [1,0;0,1];
    currentTerm = [1,0;0,1];
    %for loop to calculate finitely truncated exponential from definition
    for i = 1:n
        currentTerm = currentTerm * A / i;
        sum = sum + currentTerm;
        %Y is an array that will containt the truncation error, represented
        %as the difference in the norms of the expm value and the truncated
        %value
        Y(n) = abs(norm(expm(A)) - norm(sum));
        %Z(n) = abs(norm(expm(A) - sum));
    end
end

%array for x and creating log arrays
X = [];
logX = [];
logY = [];
for i = 1:m
    X(i) = i;
    logX(i) = log(X(i));
    logY(i) = log(Y(i));
end

%plotting
plot(X, Y);
%*
xlim([0 17]);
%plot(logX, logY);
title("Decay of Truncation Error of the Matrix Exponential");
xlabel("Number of steps");
ylabel("Truncation error");
```

2.

a) $\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix}$

Begin

$L_{11} U_{11} = A_{11}$

recursive call on (A_{11})

$L_{11} U_{12} = A_{12}$

$\Rightarrow U_{12} = L_{11}^{-1} A_{12}$

$L_{21} U_{11} = A_{21}$

$\Rightarrow L_{21} = A_{21} U_{11}^{-1}$

$L_{22} U_{\cancel{11}} = A_{21}$

$\Rightarrow L_{21} = A_{21} U_{11}^{-1}$

$L_{22} U_{22} = \underbrace{A_{22} - L_{21} U_{12}}$ Recursive call on this

Base case, when matrix is 1×1, $[a]_x = \underset{L}{[1]}\underset{U}{[a]}$

b) Implementing this on:

$$A = \begin{pmatrix} 1 & 3 & 0 & 0 \\ 2 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 2 & 0 & 0 & 2 \end{pmatrix}$$

$$A_{11} = \begin{pmatrix} 1 & 3 \\ 2 & 1 \end{pmatrix}$$

PTO

Recursive call on (A_{11})

returns: $\underset{L}{\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}} \underset{U}{\begin{pmatrix} 1 & 3 \\ 0 & -5 \end{pmatrix}}$

$$A = \begin{pmatrix} 1 & 0 & \underline{0} \\ 2 & 1 & \\ L_{21} & & L_{22} \end{pmatrix} \begin{pmatrix} 1 & 3 & U_{12} \\ 0 & -5 & \\ \underline{0} & & U_{22} \end{pmatrix}$$

$$U_{12} = L_{11}^{-1} A_{12} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\Rightarrow U_{12} = \begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

$$L_{21} = A_{21} U_{11}^{-1} = \begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix} \begin{pmatrix} 1 & 3 \\ 0 & -5 \end{pmatrix}^{-1}$$

$$= \begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix} \cdot \frac{1}{5} \begin{pmatrix} 5 & -3 \\ 0 & 1 \end{pmatrix}$$

$$\Rightarrow L_{21} = \begin{pmatrix} 0 & \frac{-1}{5} \\ 2 & 6/5 \end{pmatrix}$$

PTO

$$A_{22} - L_{21}U_{12} = \begin{pmatrix}1 & 0\\ 0 & 2\end{pmatrix} - \begin{pmatrix}0 & -\frac{1}{5}\\ 2 & \frac{6}{5}\end{pmatrix}\begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix}$$

$$= \begin{pmatrix}1 & 0\\ 0 & 2\end{pmatrix} - \begin{pmatrix}0 & -\frac{1}{5}\\ 0 & \frac{6}{5}\end{pmatrix} = \begin{pmatrix}1 & \frac{1}{5}\\ 0 & \frac{4}{5}\end{pmatrix} (*)$$

$$\left(N.B. \quad * = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}1 & \frac{1}{5}\\ 0 & \frac{4}{5}\end{pmatrix} \right)$$

$$A = \underbrace{\begin{pmatrix}1 & 0 & 0 & 0\\ 2 & 1 & 0 & 0\\ 0 & -\frac{1}{5} & 1 & 0\\ 2 & \frac{6}{5} & 0 & 1\end{pmatrix}}_{= L}\underbrace{\begin{pmatrix}1 & 3 & 0 & 0\\ 0 & -5 & 0 & 1\\ 0 & 0 & 1 & \frac{1}{5}\\ 0 & 0 & 0 & \frac{4}{5}\end{pmatrix}}_{= U}$$

---

c) $A \in \mathbb{C}^{n \times n}$ but n is not an int. power of 2, so n could be any number ($\mathbb{N}$)

The approach I propose would be to go row, column by row, column and pass in the remaining $(n-1)\times(n-1)$ matrix, altered so we should only need one recursive call

PTO

We can write $A = LU$ similarly to 2(a) but with a new approach like this

$$\underline{A} = \begin{pmatrix} a_{11} & \underline{a}_{12} \\ \underline{a}_{21} & \underline{A}_{22} \end{pmatrix} \overset{(*)}{=} \begin{pmatrix} 1 & \underline{0} \\ \underline{l}_{21} & \underline{L}_{22} \end{pmatrix} \begin{pmatrix} u_{11} & \underline{u}_{12} \\ \underline{0} & \underline{U}_{22} \end{pmatrix}$$

$$= \begin{pmatrix} u_{11} & \underline{u}_{12} \\ u_{11}\underline{l}_{21} & \underline{l}_{21} \otimes \underline{u}_{12} + \underline{L}_{22}\underline{U}_{22} \end{pmatrix} \quad (**)$$

Here I have represented vectors and matrices as underlined and scalars are not underlined

So, we have that:

a_{11} - scalar
$\underline{a}_{12}$ - $1 \times (n-1)$ row vector
$\underline{a}_{21}$ - $(n-1) \times 1$ column vector
$\underline{A}_{22}$ - $(n-1) \times (n-1)$ matrix

Comparing $(*)$ and $(**)$ gives us:

$a_{11} = u_{11}$
$\underline{a}_{12} = \underline{u}_{12}$
$\underline{a}_{21} = u_{11}\underline{l}_{21}$
$\underline{A}_{22} = \underbrace{\underline{l}_{21} \otimes \underline{u}_{12}}_{\text{N.B. outer product, not mat. mult.}} + \underline{L}_{22}\underline{U}_{22}$

PTO

Rearranging these gives us:

$u_{11} = a_{11}$ ①
$\underline{u}_{12} = \underline{a}_{12}$ ②
$\underline{l}_{21} = u_{11}^{-1} \underline{a}_{21}$ ③
$$\underline{L}_{22} \underline{U}_{22} = \underline{A}_{22} - \underbrace{\underline{a}_{21} a_{11}^{-1}}_{= \underline{l}_{21}} \underbrace{\underline{a}_{12}}_{= \underline{u}_{12}} \quad ④$$

"Calculating" ①, ②, ③ gives us the first row and column of L and U

Let's call $\underline{L}_{22} \underline{U}_{22} := S$

S is of size $(n-1) \times (n-1)$

Perform a recursive call on S

Our end condition is when S is 1×1, in which case $L = 1$, $U = A$ (where A is recursively passed in value of S)

~~This algorithm~~

d) MA398_Assignment1_Exercise2d.m

Please note, I ~~have~~ have included "recursive_Lu.m" for my part (e) to use, they are identical.

```matlab
function [L,U] = recursive_lu(A)

    %some notes about my code:
    %This doesn't return the same L,U as matlab's lu() method, but it still
    %provides an L,U that satisfy the relationship A=LU. I think this is
    %probably because lu() uses a much more sophisticated algorithm.
    %Additionally, I think that it would have been possible to create a
    %more efficient algorithm, which incurs two recursive calls and splits
    %the matrix differently, rather than row,col by row,col per call.

    %m is useless, but size() returns both dimensions
    [m,n] = size(A);

    %base case
    if n == 1
        L = 1;
        U = A;
        return;
    end

    %setting up ALU form
    A11 = A(1,1);
    A12 = A(1,2:n);
    A21 = A(2:n,1);
    A22 = A(2:n,2:n);

    L11 = 1;
    U11 = A11;
    U12 = A12;
    L21 = A21 / U11;
    %for loop to fill op array as the outer product
    for j = 1:n-1
            for k = 1:n-1
            %op(k,j) =  U12(j) * L21(k) / A11; wrong way around
            op(k,j) =  A21(k) * A12(j) / A11;
            %calculating outer product and dividing by scalar A11
            end
    end

    %debugging
    %disp(op);
    %disp(A22);
```

```
    S = A22 - op;

    %recursive call
    [L22, U22] = recursive_lu(S);

    L12(1:n-1) = 0;
    U21(1:n-1) = 0;
    U21 = transpose(U21);
    %without transposing, U21 is wrong way around

    %setting up current L,U to pass up
    L = [L11, L12; L21, L22];
    U = [U11, U12; U21, U22];
    return;
end
```

e) MA398_Assignment1 - Exercise2e.m

I have made some comments on the code in ~~the~~ the file.

I will add plots of values of k and time (both normal time and ln(time))

As for the complexity, the time taken for k seems to be similar to a cubic time complexity

It is also interesting to note, the times for $k = 3, 4, 5, 6 <$ for $k = 2$.

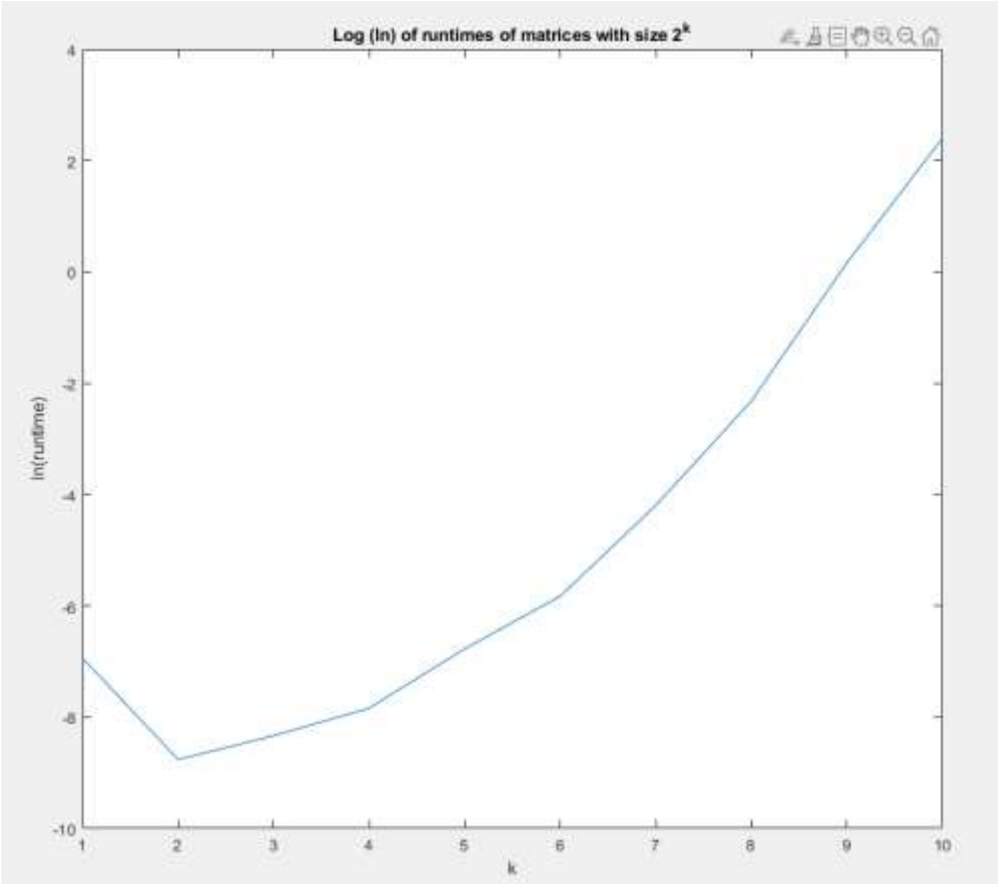
This is probably because in the cases $k = 1, 2$, alot of unnecessary operations are happening and ~~these~~ I would guess these may take longer to check than for higher values.

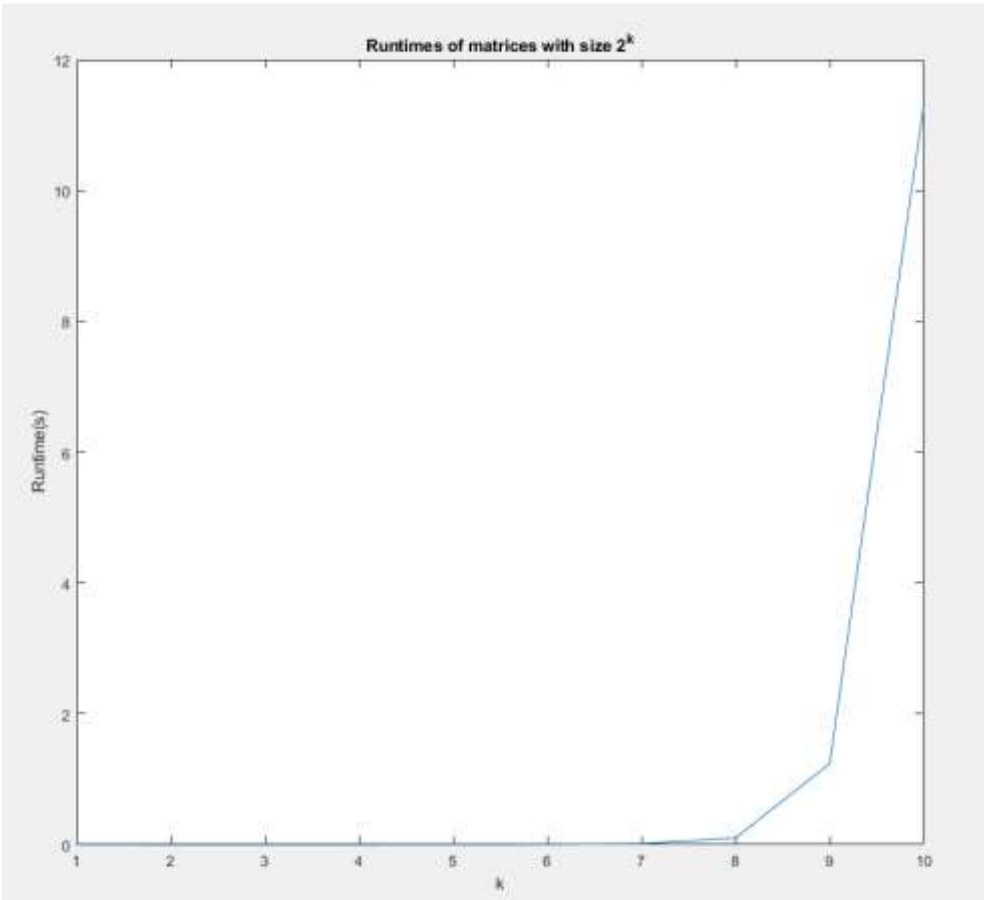
Unsurprisingly, the time increase is massive past ~~k~~ $k=6$.

Past $k=10$ seems to be too much for matlab unfortunately. It would be interesting to see how long it would take if $k = 100$, which would be a matrix with 2^{200} elements.

f) A 'typical' matrix isn't likely to not have an LU factorisation because you can just reorder the rows to have non zero determinants

```matlab
%more than 10 uses lots of ram and matlab does not like it, I
%assume that is normal for over 4 million matrix entries
for k = 1:10
    mat = rand(2.^k);
    tic;
    recursive_lu(mat);
    time(k) = toc;
end

for i = 1:k
    X(i) = i;
end

%time graph
figure(1)
plot(X, time);
title("Runtimes of matrices with size 2^{k}");
xlabel("k");
ylabel("Runtime(s)");

%seems like cubic complexity, hard to tell exactly just off of time graph,
%but it looks very similar to O(n^3) complexity

%log graph
figure(2)
plot(X, log(time));
title("Log (ln) of runtimes of matrices with size 2^{k}");
xlabel("k");
ylabel("ln(runtime)");
```


Log (ln) of runtimes of matrices with size 2^k
ln(runtime)
k
4
2
0
-2
-4
-6
-8
-10
1
2
3
4
5
6
7
8
9
10


Runtimes of matrices with size 2^k
Runtime(s)
k
12
10
8
6
4
2
0
1
2
3
4
5
6
7
8
9
10