ECE 101 - Lab #1: Introduction to MATLAB

Date Given: September 29, 2025

Date Due: October 10, 11:59 pm PST

Submission: Please export your completed work as a PDF and submit it to Gradescope.

Note: Make sure your MATLAB code is well-commented and outputs are clearly presented, as this will help ensure your work is understood and graded accurately.

Problem 1. MATLAB Onramp Tutorial.

Task 1. Complete the MATLAB Onramp Tutorial. **Be sure to upload your completion certificate to Gradescope.**

In the following set of tasks, enter and execute your code in the "Code" sections, and answer the underlined questions in a "Text " section before or after your code section.

Problem 2: Deciphering an audio message

This problem is about using array/matrix operations to decipher an encrypted audio message.

Task 1. Load the file `Lab_1_F24.mat`. Verify that the variables X and Fs are added to your workspace.

```
load('Lab_1_F25.mat')
```

Task 2. Determine the size of the array X. Determine the value of Fs**.**

Is X a row vector or a column vector?

It is a column vector.

```
size(X)
```

```
ans = 1×2
      124152           1
```

```
Fs
```

```
Fs =
44100
```

The array X is a complex-valued discrete-time signal that represents an encrypted audio signal Y of length N.

(You can try playing the real part, imaginary part, magnitude, and phase of X, and they won't sound like any useful message.)

The encryption process was the following.

1. Using a secret seed, a random reordering or permutation, `perm`, of the numbers from 1 to N was generated.
2. The elements of the signal vector Y were then scrambled according to the permutation, creating a new signal Z=Y(perm).
3. A complex vector W was created, with real part equal to the first $\frac{N}{2}$ elements of Z, and imaginary part equal to the last $\frac{N}{2}$elements.
4. Finally, the vector X was created with real part equal to the magnitude of W, and imaginary part equal to the phase of W.

Task 3. Decipher the signal using array/matrix operations. Do not use loops. You can use the commands `real`, `imag`, `abs`, `angle`, and `exp`, as needed.

More specifically, to decipher X, do the following:

1. Determine from X the value of N.
2. From the real and imaginary parts of X, recover the vector W using Euler's formula.
3. From the real and imaginary parts of W, recover the vector Z using an array concatenation operation.
4. Recreate the permuation `perm`,setting the secret seed to 2023 with the command `rng(2023),` and `perm=randperm(N).`
5. Finally, recover the signal Y. What is the size of your recovered signal?

The operation used here creates a row vector Y of size 1 x 248304 from the permuted column vector Z.

```
% Unscramble the message X
N = 2*length(X);                % determining the length of the audio signal
W = real(X).*(exp(i*imag(X)));  % recover W using array exponentiation and
element by element array multiplication
Z = cat(1,real(W),imag(W));     % recover Z by array concatentation along
the proper dimension
rng(2023)                       % set the seed for the random permutation
perm = randperm(N);             % regenerate the random permutation
Y(perm) = Z;                    % recover Y from Z using perm
```

Task 4. Play the sound file Y. using playback rate `Fs`.

Can you recognize the language being spoken?

```
sound(Y,Fs);
```

Task 5. Create an array M that is the flipped version of Z. You will have to decide which flip command to use, `fliplr` or `flipud`.

Now play the sound file M.

Write the words that you hear.

Do you know the origin of this audio clip?

We marveled at our own magnificence as we gave birth to AI.

Laurence Fishburne as Morpheus in *"The Matrix"* (1999).

```
M = fliplr(Y);
sound(M,Fs);
```

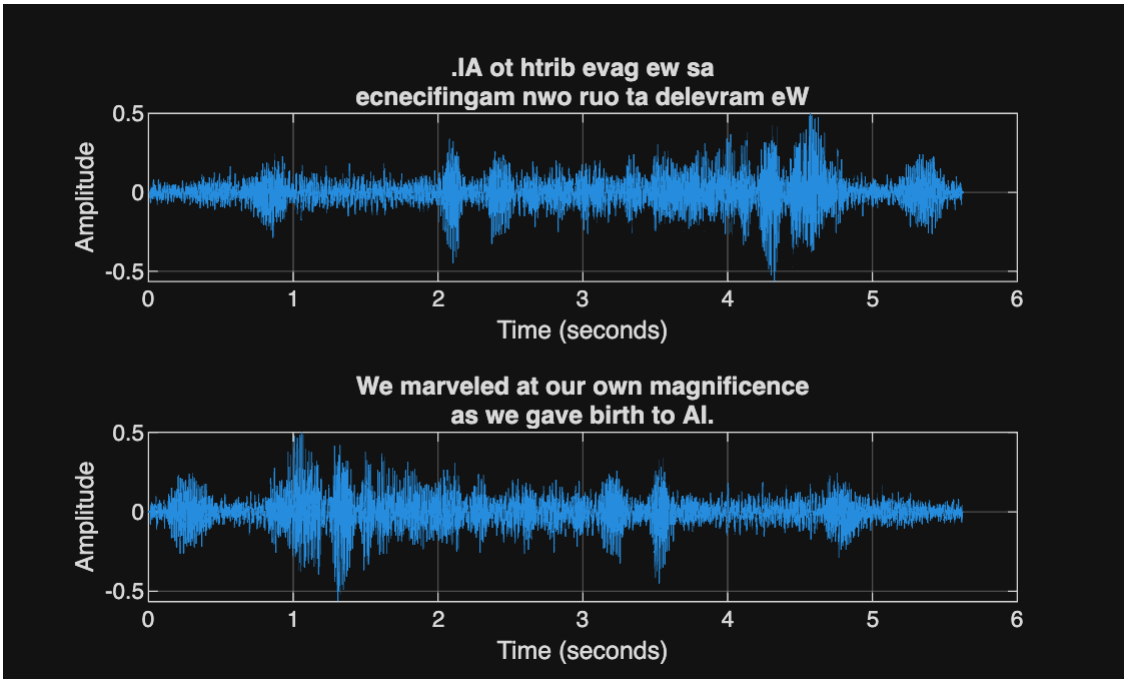
Task 6. Plot both signals Y and M.

Label the x-axis 'Time (seconds)', with numerical scale and tick marks on the x-axis corresponding to actual time (in seconds).

Label the y-axis 'Amplitude'. For both plots, use the title command to write the message contents over the plot.

(You should use two lines for each of the titles.)

```
n = 1:N;
title_text = 'We marveled at our own magnificence' ;
subtitle_text =  'as we gave birth to AI.' ;
reversed_title_text = fliplr(title_text);
reversed_subtitle_text = fliplr(subtitle_text);
figure
subplot(2,1,1)
plot(n/44100,Y)
xlabel('Time (seconds)')
ylabel('Amplitude')
title({reversed_subtitle_text,reversed_title_text})
grid on
subplot(2,1,2)
plot(n/44100,M)
xlabel('Time (seconds)')
ylabel('Amplitude')
title({' ',title_text, subtitle_text})
grid on
```



Task 7. Play M again, with a playback rate of `Fs/2`.

How has the character of the sound changed?

The pitch is lowered by a factor of 2, and the clip is twice as long.

```
sound(M,Fs/2); % The pitch is lowered by a factor of 2, and the duration of
the clip
                    % is twice as long
```

Task 8. Play it once more, now with a playback rate of `2*Fs`.

What does this do to the character of the sound?

The pitch is raised by a factor of 2, and the clip is half as long.

```
sound(M,2*Fs); % The pitch is raised by a factor of 2, and the duration of
the clip
                   %is half as long
```

Problem 3: Operations on vectors

Task 1: Write a MATLAB function 'decimate' that removes every other element from an arbitrary length vector, creating a shorter vector made up of only the elements with odd index numbers in the original vector.

Use only matrix/vector manipulations; do NOT use loops.

Write your function in a code box here but with all lines commented out. You will have to write the actual function in a code box at the very end of your LiveScript for it to be called properly.

```
% function op_vec = decimate(ip_vec)
%   op_vec=ip_vec(1:2:end);
% end
% % output the subsequence formed by elements in the odd positions
```

Task 2. Write a MATLAB function 'interpolate' that creates a longer vector by adding an additional element between neighboring elements in the original vector.

Each new element should equal the average of its neighboring elements.

Again, use only matrix/vector manipulations; do NOT use loops.

Again, write your function in a code box here but with all lines commented out. Write the actual function in the same code box used for the function in Task 1 at the very end of your LiveScript.

```
% function op_vec = interpolate(ip_vec)
%   op_vec(1:2:2*length(ip_vec)-1)= ip_vec;
%   n=1:length(ip_vec)-1;
%   op_vec(2*n) = (ip_vec(n)+ip_vec(n+1))/2;
% end
% % fill the empty space by taking average
```

Task 3.

Test your solution by applying 'decimate' and then 'interpolate' to the vector `x= [7 6 5 4 3 2 1 2 3 4 5 6 7]`.

Then apply them to the vector `y=[8 7 6 5 4 3 2 1 2 3 4 5 6 7 8]`.

Now apply the two operations in succession, in both orders, to each of x and y.

What happens after the execution of 'decimate' followed by 'interpolate', and 'interpolate' followed by 'decimate'?

Does the order of operation affect the result?

Under what general conditions do these two functions commute?

Considering integer "V" signals like x and y, for which lengths will the operations commute?

The order does not affect the result for x, but does affect the result for y. In other words, the operations commute for x but not for y.

interpolate(decimate(v)) = v happens only when each even element in the original vector is the average of its neighbors.

decimate(interpolate(v)) is always equal to v.

Therefore, the operations commute only when each even element in the original vector is the average of its neighbors.

The length of the "V" signal should be a multiple of 4 plus 1, i.e., congruent to 1 modulo 4, in order for the operations to commute.

```
x=[7 6 5 4 3 2 1 2 3 4 5 6 7 ]
```

```
x = 1×13
     7     6     5     4     3     2     1     2     3     4     5     6     7
```

```
decimate(x)
```

```
ans = 1×7
     7     5     3     1     3     5     7
```

```
interpolate(x)
```

```
ans = 1×25
    7.0000    6.5000    6.0000    5.5000    5.0000    4.5000    4.0000    3.5000 ···
```

```
y=[8 7 6 5 4 3 2 1 2 3 4 5 6 7 8]
```

```
y = 1×15
     8     7     6     5     4     3     2     1     2     3     4     5     6 ···
```

```
decimate(y)
```

```
ans = 1×8
     8     6     4     2     2     4     6     8
```

```
interpolate(y)
```

```
ans = 1×29
    8.0000    7.5000    7.0000    6.5000    6.0000    5.5000    5.0000    4.5000 ···
```

```
interpolate(decimate(x))
```

```
ans = 1×13
     7     6     5     4     3     2     1     2     3     4     5     6     7
```

```
decimate(interpolate(x))
```

```
ans = 1×13
     7     6     5     4     3     2     1     2     3     4     5     6     7
```

```
interpolate(decimate(y))
```

```
ans = 1×15
     8     7     6     5     4     3     2     2     2     3     4     5     6 ···
```

```
decimate(interpolate(y))
```

```
ans = 1×15
     8     7     6     5     4     3     2     1     2     3     4     5     6 ···
```

Task 4. Separately apply the functions 'decimate' and 'interpolate' to the sound vector M (reshaped, as necessary) that you created in Problem 1.

Play the resulting vectors with playback rate of fs.

How would you characterize the audible effects of applying the two operations?

How do they compare with the results in Problem 2 from using playback at half and twice the nominal rate of Fs?

Do you think the corresponding reconstructed audio signals in Problem 2 and Problem 3 are truly identical? Explain your reasoning.

(**Note:** you are not expected to answer the last question rigorously at this stage. Once we study sampling theory, you will be able to.)

The vector decimate(M) is obtained from M by decimation (downsampling) by a factor of 2.

It corresponds to samples of the analog signal x(2t) from Problem 1, captured at a sampling rate of Fs=44100 Hz.

When played back at a sample rate of 44100 Hz, it will therefore sound like x(2t).

This is because the sampling frequency will still be at least twice the maximum frequency of x(2t).

So, it should sound the same as the signal in Task 8 of Problem 2.

(We will fully justify this statement when we learn about sampling theory.)

The vector interpolate(M) is sort of obtained from M by interpolation (upsampling) by a factor of 2, **but** with each inserted zero value changed to the average of its neighbors.

The values in M agree with the odd index values obtained by sampling x(t/2) at a frequency of 44100 Hz.

The "interpolated" average values will probably be close to the even index values obtained by sampling x(t/2) at 44100 Hz, but generally not exactly the same.

Therefore, when twice(M) is played back at 44100 Hz, it will sound very similar to, even indistinguishable from, the signal in Task 7 of Problem 2.

But is very likely not exactly the same.

decimate(M) played back at rate Fs

```
Md = decimate(M);
sound(Md,Fs)
```

interpolate(M) played back at rate Fs

```
Mi = interpolate(M);
```

```
sound(Mi,Fs)
```

Problem 4. Complex functions

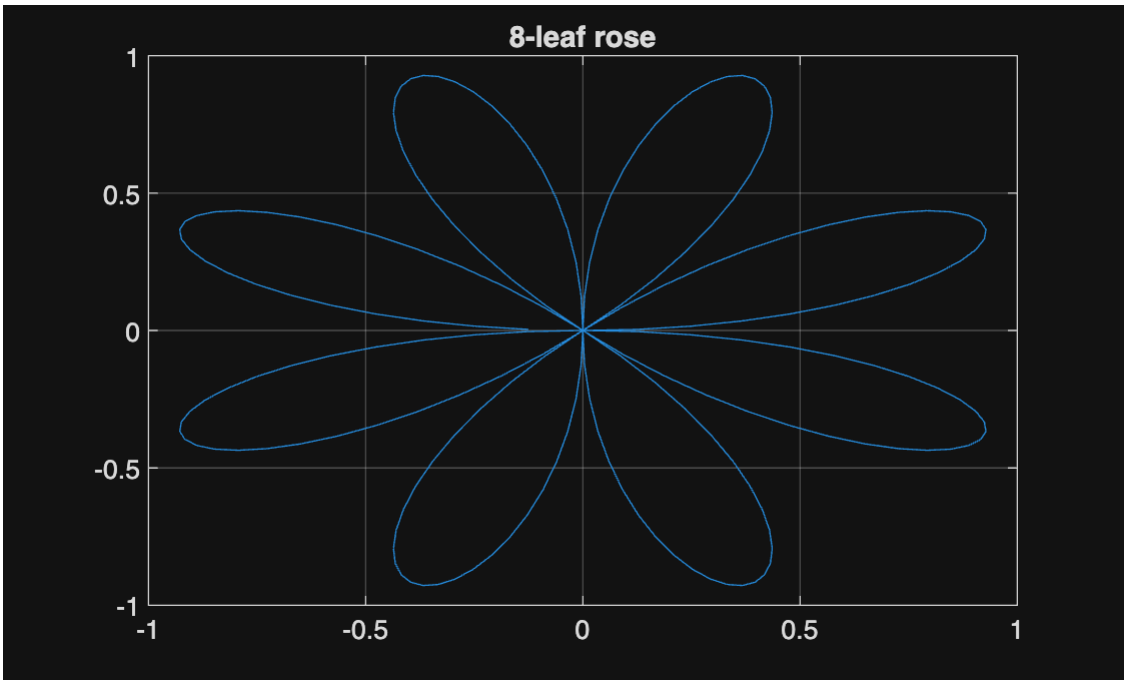
Task 1. Refer to the 5-leaf rose function in Problem 3 of the Lecture 1 Demo.

Give a mathematical formula for a complex function $z(\theta) = r(\theta)e^{j\theta}$ whose plot in the complex plane is a 8-leaf rose.

Set $r(\theta) = \sin(4\theta)$.

Now, set n = 0:199, then multiply n by (2π/200) to get a vector θ containing 200 values from 0 to 2π. Implement and plot your function your function z(θ) in the complex plane.

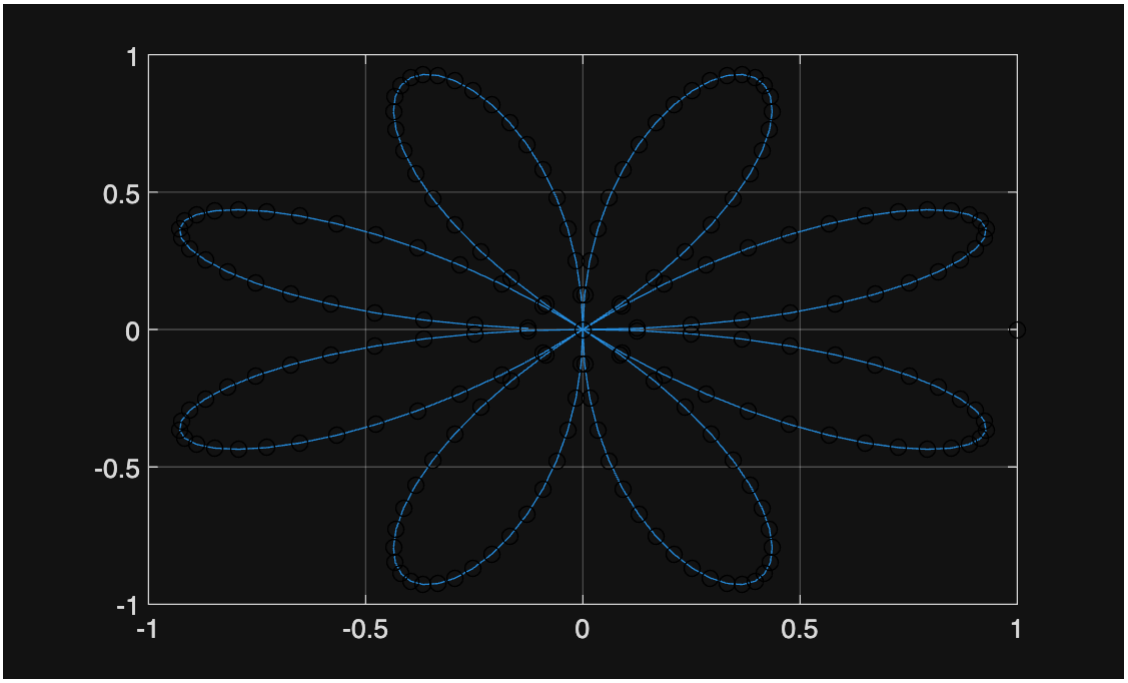
```
figure;
n = 0:199;
theta = n*2*pi/200;
r = sin(4*theta);
z = r .* exp(1i*theta);
plot(z);
title('8-leaf rose');
grid on;
```



Using the technique of Problem 3 of the Lecture 1 Demo, plot the points one at a time, as θ ranges over the 200 values from 0 to 2π. If you number the petals from 1 through 8 in the clockwise direction, starting from the top right, what order do the petals get traced out as θ ranges from 0 to 2π.

The order is 2, 5, 8, 3, 6, 1, 4, 7

```
figure;
plot(z)
hold on
for n=1:200
    plot(z(n),'ok')
    pause(0.05);
end
hold off
grid on;
```



Task 2. Determine mathematically the formulas for the real and imaginary parts of $z(\theta)$, expressing them as sums of sines and cosines.

$$z(\theta) = \sin(4\theta)e^{j\theta} = \sin(4\theta)\cos(\theta) + j\sin(4\theta)\sin(\theta)$$

$$Re\{z(\theta)\} = \sin(4\theta)cos(\theta) = \frac{1}{2}(\sin(5\theta) + \sin(3\theta)).$$

$$Im\{z(\theta)\} = \sin(4\theta)\sin(\theta) = \frac{1}{2}(\cos(3\theta) - \cos(5\theta))$$

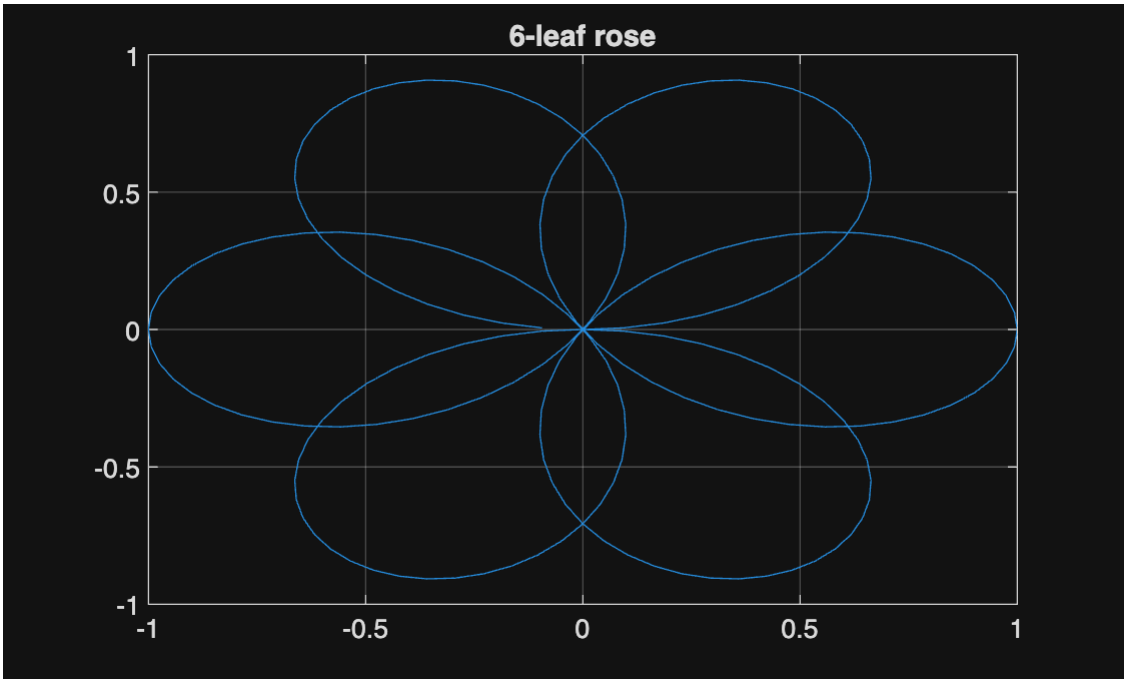
Task 3. In fact, when $r(\theta) = \sin(k\theta)$, the function $z(\theta) = r(\theta)e^{j\theta}, 0 \le \theta < 2\pi$, creates a rose with $k$petals when k is odd, and a rose with $2k$ petals when k is even. So, how can we create a rose with 6 petals? Try setting $k = 3/2$. Can you trace out the whole rose by letting θ run from 0 to 2π? How about if you let θ run from 0 to 4π? Use the technique of Lecture 1 Demo to determine the order in which the petals are traced out, numbering them from 1 to 6, starting at the upper right.

Running from 0 to 2π only traces out 3 petals. Running from 0 to 4π traces out all 6 petals.

The order is 1, 2, 3, 4, 5, 6.

For more on complex rose functions, see **https://en.wikipedia.org/wiki/Rose_(mathematics)**

```
figure;
n = 0:199;
theta = n*4*pi/200;
r = sin((3/2)*theta);
z = r .* exp(1i*theta);
plot(z);
title('6-leaf rose');
grid on;
```




```
figure;
plot(z)
hold on
for n=1:200
    plot(z(n),'ok')
    pause(0.05);
end
```

```
hold off
grid on;
```

Put your functions `decimate` and `interpolate` in the code block after the following dummy code block.

```
0;
```

```
function op_vec = decimate(ip_vec)
% decimate the input vector  by factor of 2 (odd positions)
op_vec=ip_vec(1:2:end);
end


function op_vec = interpolate(ip_vec)
% create expanded output vector; fill odd positions with input values
 op_vec(1:2:2*length(ip_vec)-1)= ip_vec;
 % fill the zero values with average of adjacent values
 n=1:length(ip_vec)-1;
 op_vec(2*n) = (ip_vec(n)+ip_vec(n+1))/2;
end
```