MathWorks® | *Training Services*

Course Completion Certificate

Andrew Onozuka

has successfully completed **100%** of the self-paced training course

MATLAB Onramp

DIRECTOR, TRAINING SERVICES

10 January 2025

Lab #1: Introduction to MATLAB

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 `Lab1_F25.mat`. Verify that the variables X and Fs are added to your workspace.

```
load("Lab1_S25.mat")
```

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

Is X a row vector or a column vector?

A: The size of array X is 37265x1, the value of Fs is 11025. X is a column vector.

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 2025 with the command `rng(2025)`, and `perm=randperm(N)`.
5. Finally, recover the signal Y. What is the size of your recovered signal?

A: 74530, 1

```
% Step 1: Determine N
N = 2 * length(X);

% Step 2: Recover W from magnitude and phase using Euler's formula
W = real(X) .* exp(1j * imag(X));

% Step 3: Recover Z from real and imaginary parts of W
Z = [real(W); imag(W)];

% Step 4: Recreate the permutation
rng(2025);
perm = randperm(N);

% Step 5: Recover Y by undoing the permutation
Y = zeros(N, 1);
Y(perm) = Z;

% Check size
size(Y)
```

```
ans = 1×2
       74530           1
```

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

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

Write the words that you hear.

A: This task was appointed to you. And if you do not find a way, no one will.

Do you know the origin of this audio clip?

A: No

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

Now play the sound file M.

Which flip command did you have to use?

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

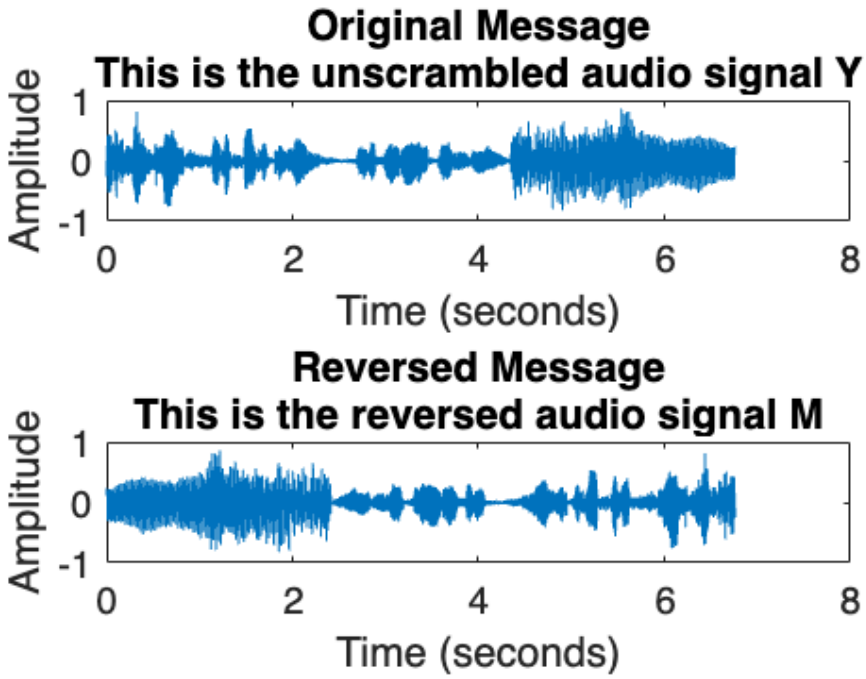
Task 6. Plot both signals Y and M.

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.)

```
% Time vector
t = (0:N-1) / Fs;  % Converts sample index to time in seconds

% Plot original signal Y
figure;
subplot(2,1,1);
plot(t, Y);
xlabel('Time (seconds)');
ylabel('Amplitude');
title({'Original Message', 'This is the unscrambled audio signal Y'});

% Plot reversed signal M
subplot(2,1,2);
plot(t, M);
xlabel('Time (seconds)');
ylabel('Amplitude');
title({'Reversed Message', 'This is the reversed audio signal M'});
```



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

How has the character of the sound changed?

A: The pitch drops by one octave, and sounds deeper and drawn out.

```
sound(Y, Fs/2);
```

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

What does this do to the character of the sound?

A: The pitch increases by one octave, and sounds higher pitched and sped up.

```
sound(Y, 2*Fs);
```

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 y = decimate(x)
% %DECIMATE Remove every other element from the input vector.
% %   y = DECIMATE(x) returns a vector y containing only the elements
% %   from x that have odd index numbers: x(1), x(3), x(5), ...
%
% %   This function uses only vector operations, no loops.

% % Keep elements at odd indices
% y = x(1:2:end);
```

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 y = interpolate(x)
% %INTERPOLATE Insert averaged elements between each pair of elements in x.
% %   y = INTERPOLATE(x) returns a new vector y such that for each pair
% %   of neighboring elements in x, their average is inserted between them.
%
% %   For example:
% %   x = [1 3 5] → y = [1 2 3 4 5]

% % Compute midpoints (averages between each pair of elements)
% midpoints = (x(1:end-1) + x(2:end)) / 2;

% % Interleave original elements and midpoints
% y = zeros(1, 2*length(x) - 1);      % Preallocate result
% y(1:2:end) = x;                     % Original elements at odd indices
% y(2:2:end) = midpoints;            % Midpoints at even indices
```

Task 3.

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

Now apply the two operations in succession, in both orders, to x.

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

Now apply the two operations in succession, in both orders, to y.

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

A: Decimate discards every other element from the original keeping the odd indices, interpolate inserts average values between elements.

Does the order of operation affect the result?

A: Yes, if we call decimate first, the details in the even indices are permanently lost, whereas in the other configuration we can fully reconstruct the signal.

Under what general conditions do these two functions commute?

A: The functions commute when the signal is a perfect linear ramp (constant step) and has odd length.

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

A: When the signal is a linear ramp and has odd length.

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

x_d_i = interpolate(decimate(x));
y_d_i = interpolate(decimate(y));

x_i_d = decimate(interpolate(x));
y_i_d = decimate(interpolate(y));

isequal(x_d_i, x_i_d)  % likely false
```

```
ans = logical
   1
```

```
isequal(y_d_i, y_i_d)  % likely false
```

```
ans = logical
   0
```

```
x
```

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

```
x_d_i
```

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

```
x_i_d
```

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

```
y
```

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

```
y_d_i
```

```
y_d_i = 1×15
     1     2     3     4     5     6     7     7     7     6     5     4     3     2     1
```

```
y_i_d
```

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

Task 4. Separately apply the functions 'decimate' and 'interpolate' to the sound vector Y (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?

A: After decimate, it sounds less detailed with some frequencies missing, after interpolate, it sounds similar but slightly blurred.

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

A: In Problem 2, changing the playback rate changes how fast the sound plays and shifts the pitch. In Problem 3, decimate and interpolate change the actual data, so the sound quality is affected, but the speed stays the same.

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

A: The reconstructed signals are **not truly identical** — playback rate adjustments preserve all original samples, while decimation permanently discards data and interpolation only estimates it. As a result, the structure and fidelity of the audio differ between the two approaches.

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

decimate(Y) played back at rate Fs

```
Y_decimated = decimate(Y);
sound(Y_decimated, Fs);
```

interpolate(Y) played back at rate Fs

```
Y_interpolated = interpolate(Y);
sound(Y_interpolated, Fs);
```

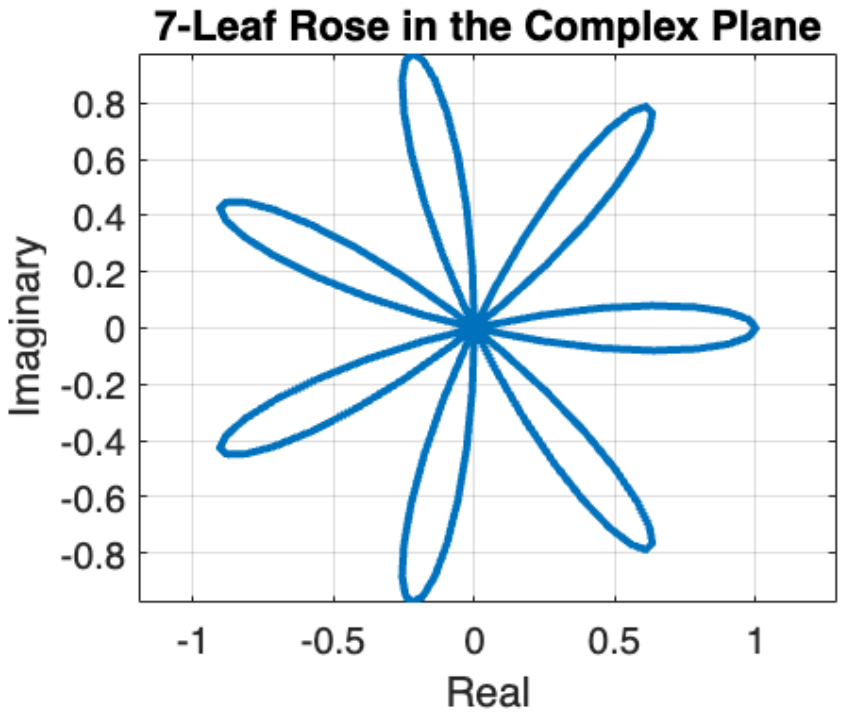
Problem 4. Complex functions

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 7-leaf rose.

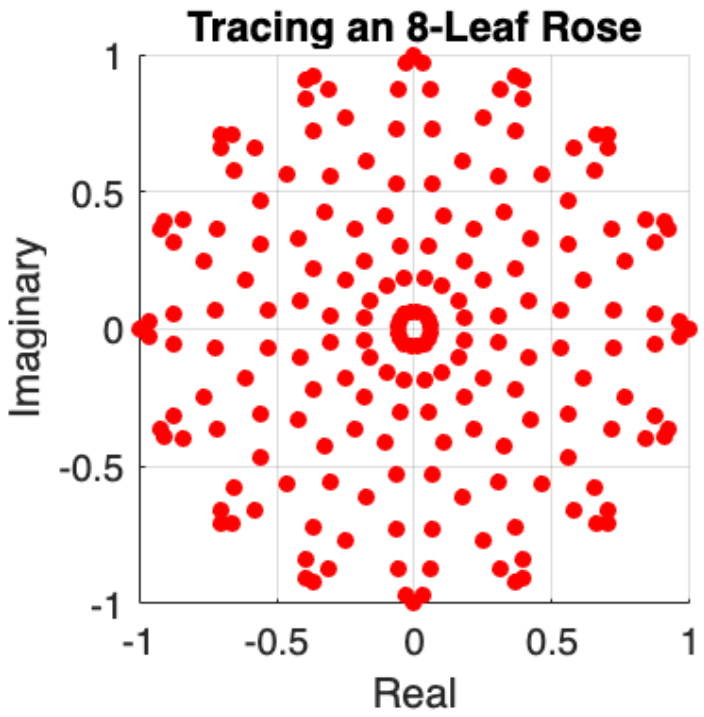
Now, set n = 0:199, then multiply n by (2π/200) to get a vector θ containing 200 values from 0 to 2π. Implement and plot your function your function z(θ) in the complex plane.

```
% Task 1: Plot a 7-leaf rose in the complex plane

% Step 1: Create theta values
n = 0:199;
theta = (2*pi/200) * n;

% Step 2: Define the 7-leaf rose complex function (petal-shaped)
% Use amplitude modulation: z(θ) = cos(kθ) * e^(jθ)
k = 7;
z = cos(k * theta) .* exp(1j * theta);  % 7-leaf rose

% Step 3: Plot in the complex plane
figure;
plot(real(z), imag(z), 'LineWidth', 2);
axis equal;
grid on;
xlabel('Real');
ylabel('Imaginary');
title('7-Leaf Rose in the Complex Plane');
```



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.

```
% Parameters
k = 8;                          % number of petals
n = 0:199;
theta = (2*pi/200) * n;         % theta from 0 to 2π

% Rose function (petal shape)
z = cos(k * theta) .* exp(1j * theta);

% Set up figure
figure;
axis equal;
axis([-1 1 -1 1]);              % fixed plot limits
grid on;
xlabel('Real');
ylabel('Imaginary');
title('Tracing an 8-Leaf Rose');
hold on;

% Animation: plot points one at a time
for i = 1:length(z)
    plot(real(z(i)), imag(z(i)), 'ro', 'MarkerSize', 4, 'MarkerFaceColor', 'r');
    pause(0.01);  % adjust speed here
end
```



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

A:

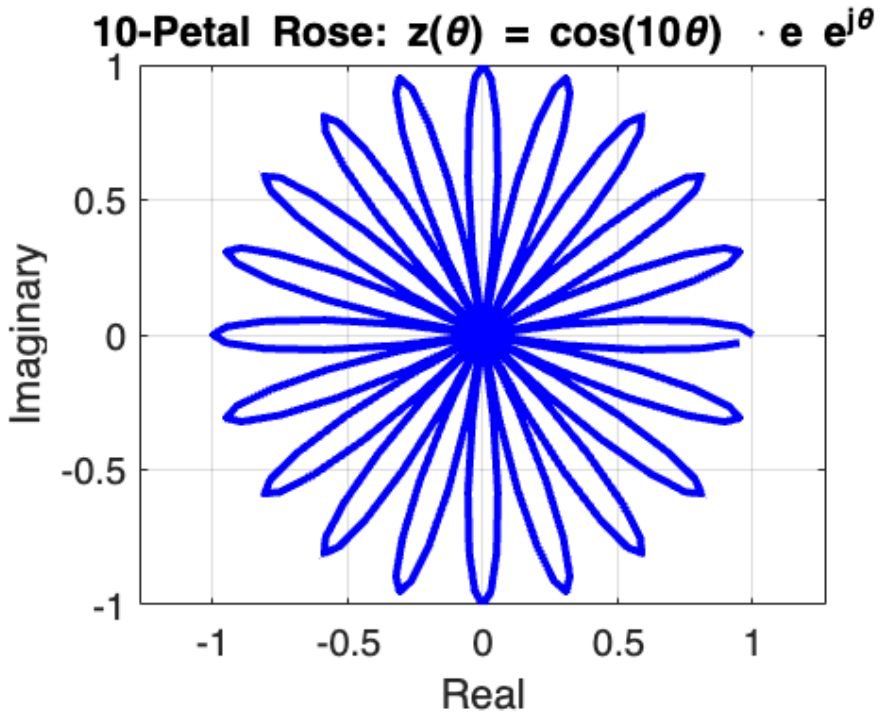
Re(z(theta)) = 0.5 * (cos(6*theta) + cos(8*theta))

Im(z(theta)) = 0.5 * (sin(8*theta) + sin(6*theta))

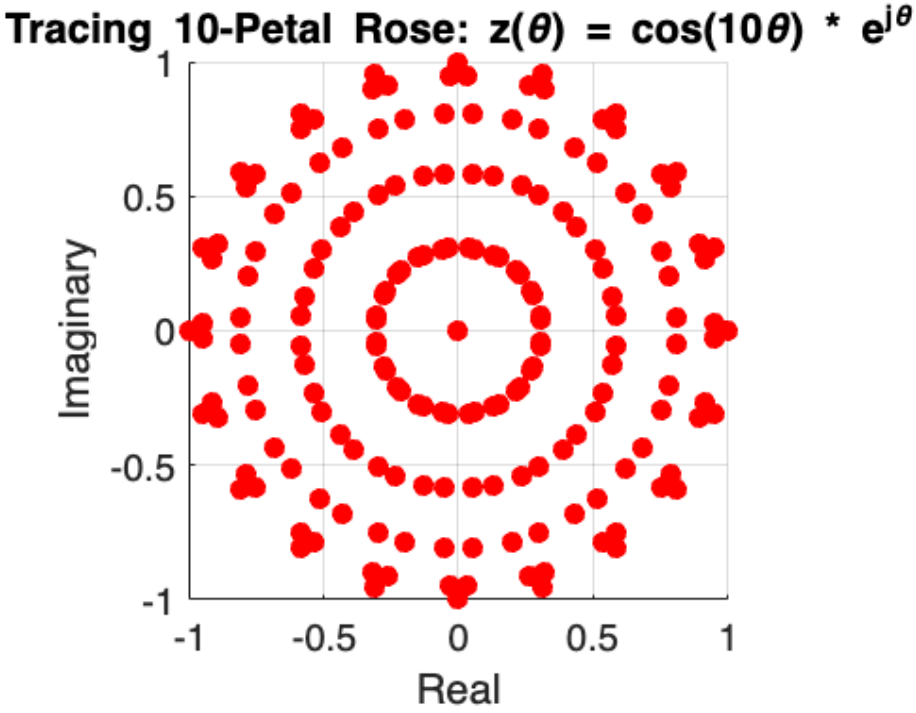
Task 3. In fact, when $r(\theta) = \sin(k\theta)$, the function $z(\theta) = r(\theta)e^{j\theta}, 0 \leq \theta < 2\pi$, creates a rose with $k$petals when k is odd, and a rose with $2k$ petals when k is even. So, how can we create a rose with 10 petals? Try setting

$k = 5/2$. 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 10, starting at the upper right.

```
% Task 3 - Plotting a 10-petal rose with z(theta) = cos(5*theta) * exp(j*theta)

% Step 1: Define theta from 0 to 2*pi
n = 0:199;
theta = (2*pi/200) * n;

% Step 2: Define the rose function with k = 10 (creates 10 petals when k is odd)
k = 10;
z = cos(k * theta) .* exp(1j * theta);  % 10-petal rose

% Step 3: Plot the complete rose
figure;
plot(real(z), imag(z), 'b', 'LineWidth', 2);
axis equal;
grid on;
xlabel('Real');
ylabel('Imaginary');
title('10-Petal Rose: z(\theta) = cos(10\theta) \cdote e^{j\theta}');
```



```
% Animated tracing to determine petal order (0 to 2*pi)

figure;
axis equal;
axis([-1 1 -1 1]);
grid on;
xlabel('Real');
ylabel('Imaginary');
title('Tracing 10-Petal Rose: z(\theta) = cos(10\theta) * e^{j\theta}');
hold on;

% Loop to trace one point at a time
for i = 1:length(z)
    plot(real(z(i)), imag(z(i)), 'ro', 'MarkerSize', 5, 'MarkerFaceColor', 'r');
    pause(0.01);  % adjust speed as needed
end
```



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
  midpoints = (ip_vec(1:end-1) + ip_vec(2:end)) / 2;
op_vec = zeros(1, 2*length(ip_vec) - 1);  % preallocate output
op_vec(1:2:end) = ip_vec;                % assign input values to odd indices
op_vec(2:2:end) = midpoints;             % assign midpoints to even indices
end
```