ECE462 Multimedia Systems

Laboratory Assignment 1: Colour Image Processing

Winter 2024

TA: Behrad TaghiBeyglou

Procedure - Week 2

1 Objective quality measurement

Operations on images, such as colour transformations and compression, introduce errors due to nonlinear operations, quantization, and finite precision arithmetic. Thus, standard metrics to measure the quality of reconstructed images are needed. The closer the reconstructed image resembles the original one, the better the values provided by the metric. The following measures are discussed in this lab:

- Mean Squared Error

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} ||\overrightarrow{P_i} - \overrightarrow{Q_i}||^2, \tag{1}$$

  where:
  $\overrightarrow{P_i}$: pixel i of the original image,
  $\overrightarrow{Q_i}$: pixel i of the reconstructed image,
  n: total number of pixels in image,
  and,

$$||\overrightarrow{P_i} - \overrightarrow{Q_i}||^2 = \frac{(R_{P_i} - R_{Q_i})^2 + (G_{P_i} - G_{Q_i})^2 + (B_{P_i} - B_{Q_i})^2}{3}.$$

- Peak Signal-to-Noise Ratio

$$\text{PSNR} = 10\log_{10}\frac{255^2}{MSE}, \tag{2}$$

  Note, this assumes an 8-bit unsigned integer representation (i.e., peak signal is 255). More generally, for a b-bit unsigned integer representation, the peak signal used in Equation 2 is $2^b - 1$.

- Signal-to-Noise Ratio

$$\text{SNR} = 10\log_{10}\frac{\frac{1}{n}\sum_{i=1}^{n} ||\overrightarrow{P_i}||^2}{MSE}, \tag{3}$$

  where, $||\overrightarrow{P_i}||^2 = \frac{R_{P_i}^2 + G_{P_i}^2 + B_{P_i}^2}{3}$.

1. Write three separate functions to calculate MSE, PSNR, and SNR of an image and its corrupted version. Each function should have two inputs: the original image ($\vec{P}$) and the corrupted image ($\vec{Q}$). You should be able to write these functions with only a few lines of code, and without any loops (refer back to the lab preparation for hints). Since your inputs may be in data types such as `uint8`, it is prudent to convert them to `double` (allowing full precision calculations on them) at the beginning of the function (if they are already in double, the conversion will have no effect).

2. Write a script that loads the images `jbeans_corrupted_sample.ppm` and `jbeans.ppm` (available on the course website), and then call your functions to compute the MSE, PSNR and SNR. (The `jbeans_corrupted_sample.ppm` image is simply the `jbeans.ppm` image corrupted with Gaussian noise in each of the R, G, and B channels).

   5 Marks **Demonstrate your functions to the TA – i.e., run the `jbeans_corrupt` script and show the three results**

3. Load the `jbeans.ppm` image and convert it into the YC_bC_r colour space (as in week 1 procedure). Name the newly generated image `jbeans_ycc`. Perform the following operation for each of the values $\sigma^2 = [10^2, 20^2, 40^2, 80^2, 100^2]$ (i.e., write a `for` loop where σ^2 is assigned one of these values for each iteration — see end of handout for a hint on how to write such `for` loops). For each iteration:

   (a) Generate a noise image of size 256×256 with each pixel value following a Gaussian distribution of mean $\mu = 0$ and variance σ^2. Name this image `noise`.
   Hint: The command `randn()` generates a matrix of random numbers drawn from a Gaussian distribution with zero mean and unit variance. Before you use this function during a particular MATLAB session, you should first use `randn('state',sum(100*clock))` to randomize the state of the number generator. You can then use `randn(N)` to make a matrix of size $N \times N$. You must scale the values by σ (not σ^2 !) to achieve the correct variance.

   (b) Add the noise image `noise` to the Y component of the transformed image `jbeans_ycc`. Name this image `ycc_corrupted_y`.

   (c) Repeat the previous steps by adding noise to the C_b, and C_r components respectively. Name the generated images `ycc_corrupted_Cb` and `ycc_corrupted_Cr` respectively. Note: Each of the above corrupted images is only corrupted in ONE particular channel.

   (d) Using your YC_bC_r to RGB function to transform the three corrupted images back to the RGB domain to obtain `rgb_corrupted_y`, `rgb_corrupted_Cb`, `rgb_corrupted_Cr`. If your transform doesn't do so already, make sure you round and convert the output RGB values to `uint8`.

   (e) Calculate the MSE, PSNR and SNR of the three corrupted images comparing with the original one. Store the resulting values for each iteration of the loop

(i.e., for each variance) in vectors or a matrix. (Note that when you use a matrix to store all the results, you should be clear what each row/colomn represents).

(f) After execution of the loop, plot the MSE values of all the three corrupted images as functions of the variance in one figure.

Hint: To include more than one plots in the same figure use MATLAB's command `hold`. For example:

```
figure; hold on;
plot(vars,mse_vals_corrupted_y);
plot(vars,mse_vals_corrupted_cb,'r'); % Plot in red colour
hold off;
title('This is the plot title');
xlabel('This is the x-axis label');
ylabel('This is the y-axis label');
```

Hint: To distinguish different curves on the same plot, you might want to make use of different line styles and markers. Type "help plot" to check how to do that.

(g) Repeat previous step for PSNR and SNR plots.

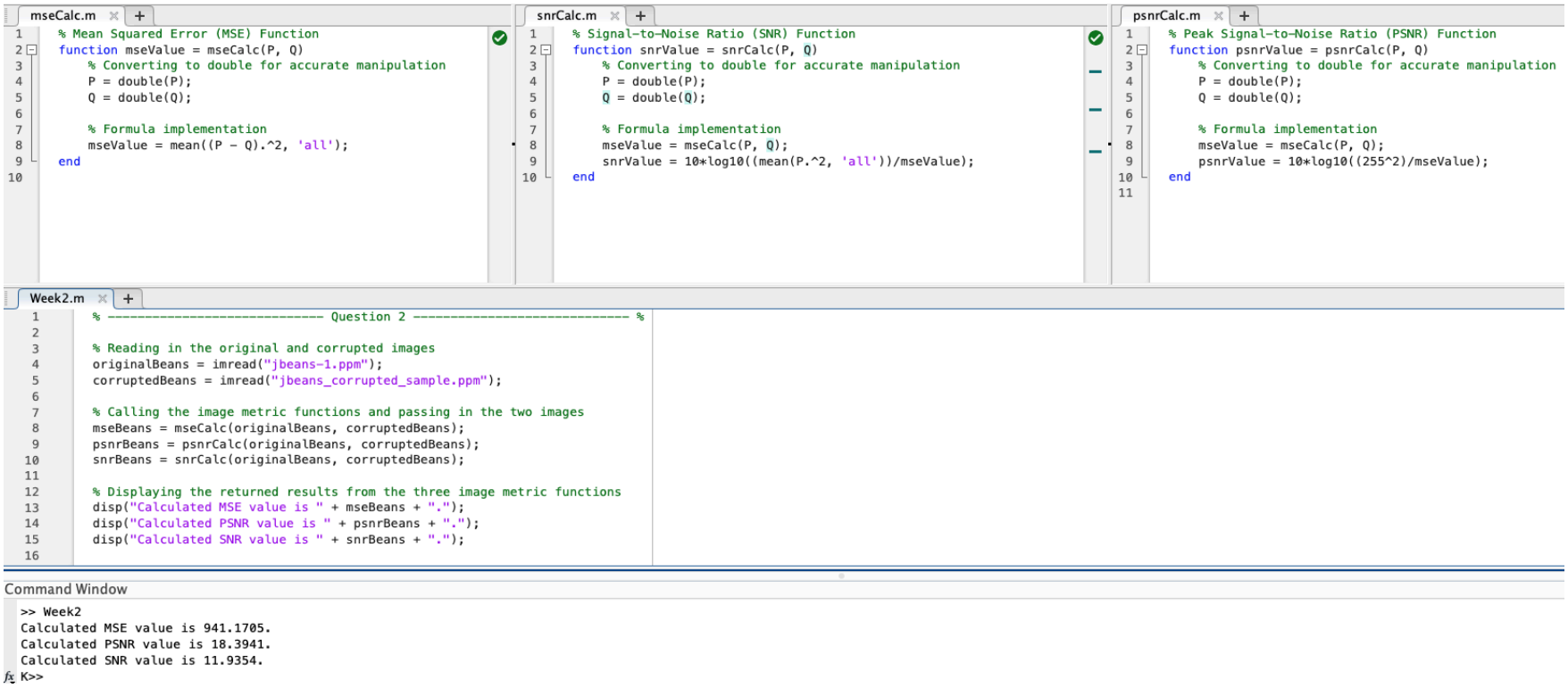
4 Marks **Show the TA the three reconstructed, corrupted images for σ^2=6400 on screen**

6 Marks **Show the TA the three plots**

2 `for` Loop Hint

To write a `for` loop in a scenario such as the one in this lab, you can assign the values you loop through to a vector, say `vars` in this case. Then you can write your loop like this:

```
vars = [10^2,20^2,40^2,80^2,100^2];

for i = 1:length(vars)
   ...
   (some operation using vars(i))
   ...
end
```



mseCalc.m

```
% Mean Squared Error (MSE) Function
function mseValue = mseCalc(P, Q)
    % Converting to double for accurate manipulation
    P = double(P);
    Q = double(Q);

    % Formula implementation
    mseValue = mean((P - Q).^2, 'all');
end
```

snrCalc.m

```
% Signal-to-Noise Ratio (SNR) Function
function snrValue = snrCalc(P, Q)
    % Converting to double for accurate manipulation
    P = double(P);
    Q = double(Q);

    % Formula implementation
    mseValue = mseCalc(P, Q);
    snrValue = 10*log10((mean(P.^2, 'all'))/mseValue);
end
```

psnrCalc.m

```
% Peak Signal-to-Noise Ratio (PSNR) Function
function psnrValue = psnrCalc(P, Q)
    % Converting to double for accurate manipulation
    P = double(P);
    Q = double(Q);

    % Formula implementation
    mseValue = mseCalc(P, Q);
    psnrValue = 10*log10((255^2)/mseValue);
end
```

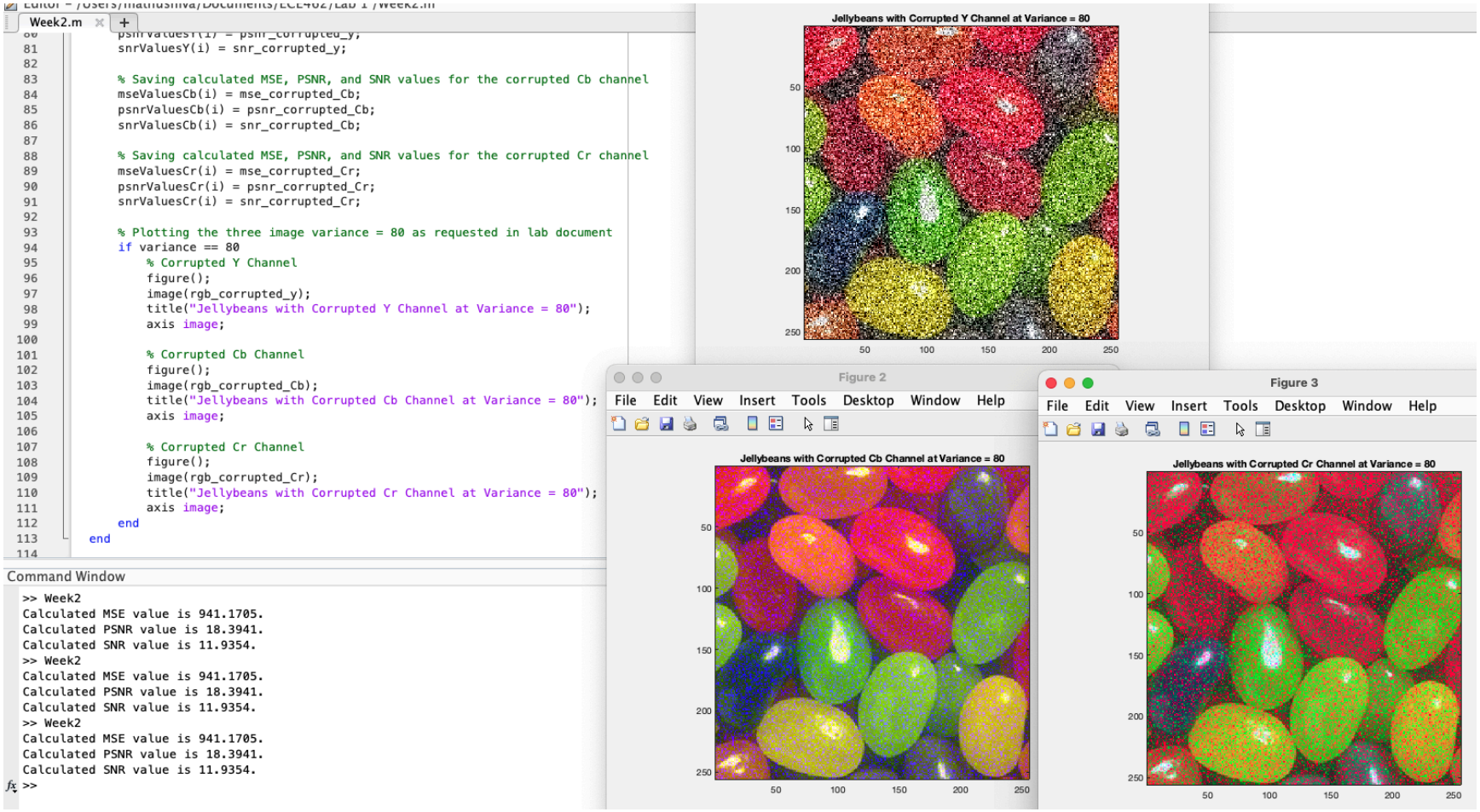
Week2.m

```
% ----------------------------- Question 2 ------------------------------ %

% Reading in the original and corrupted images
originalBeans = imread("jbeans-1.ppm");
corruptedBeans = imread("jbeans_corrupted_sample.ppm");

% Calling the image metric functions and passing in the two images
mseBeans = mseCalc(originalBeans, corruptedBeans);
psnrBeans = psnrCalc(originalBeans, corruptedBeans);
snrBeans = snrCalc(originalBeans, corruptedBeans);

% Displaying the returned results from the three image metric functions
disp("Calculated MSE value is " + mseBeans + ".");
disp("Calculated PSNR value is " + psnrBeans + ".");
disp("Calculated SNR value is " + snrBeans + ".");

```

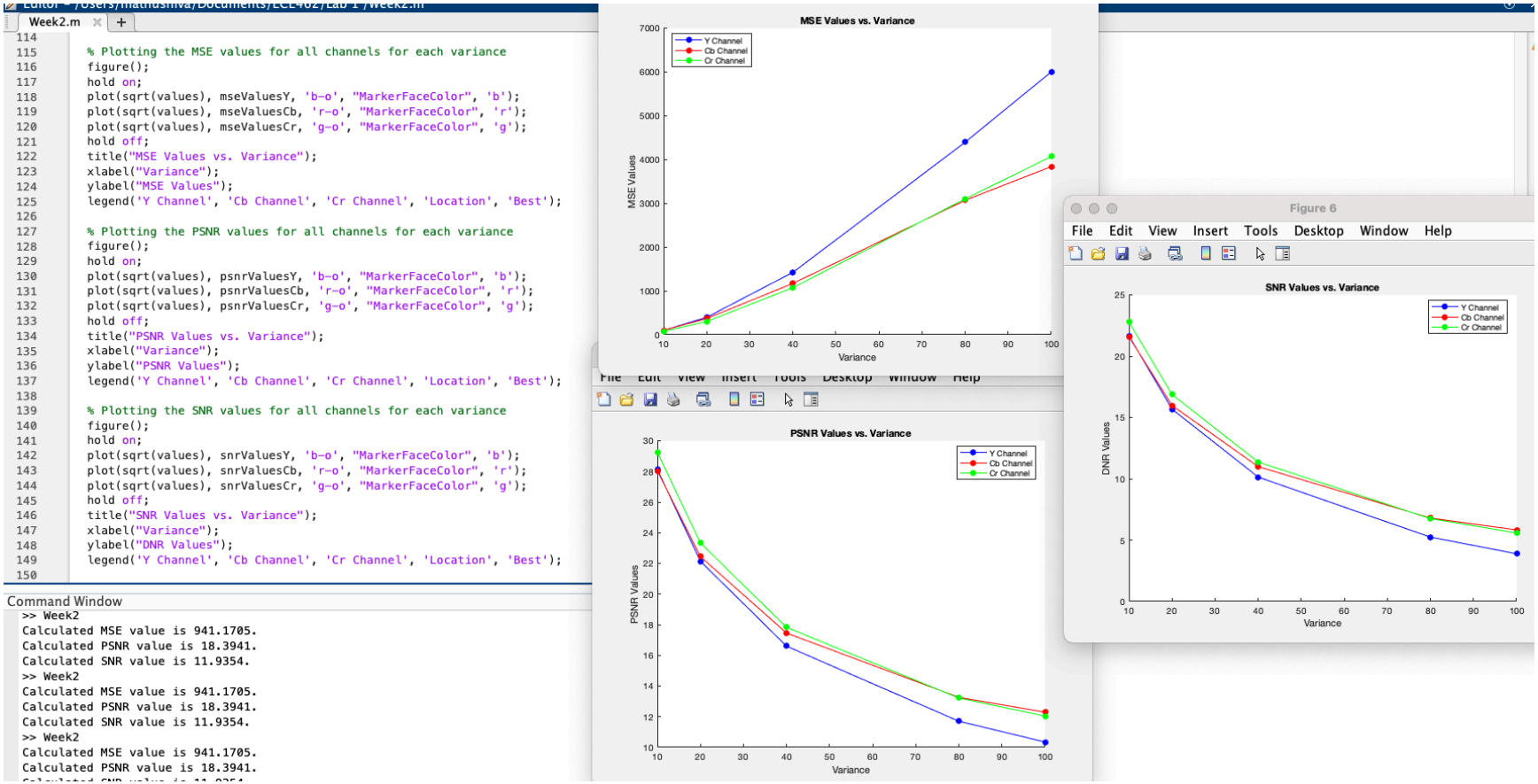
Command Window

```
>> Week2
Calculated MSE value is 941.1705.
Calculated PSNR value is 18.3941.
Calculated SNR value is 11.9354.
fx K>>
```

In *mseCalc.m*, *snrCalc.m*, and *psnrCalc.m*, I have implemented the respective functions according to the formulas provided in the lab document. All three functions return the final calculated value in variables *mseValue*, *snrValue*, and *psnrValue*. In *Week2.m*, I called this function, passing in the two required inputs as the normal original image and then the provided corrupted image. I then display these values to the command terminal, which can be seen at the bottom of the screenshot.



Through an *if statement*, if the variance was currently 80 (or σ^2 = 6400), then we would print the three corrupted images. The first image is with the original Cb and Cr components, and the corrupted Y component (noise + original Y component). The second image is with the original Y and Cr components, and the corrupted Cb component (noise + original Cb component). The third image is with the original Y and Cb components, and the corrupted Cr component (noise + original Cr component). We used *randn* to create a 256x256 noise image. We then added this to individual components to create the corrupted components, and then concatenated it with the other components to create a full YCbCr image. Full codebase is in the zip. file.



After calculating the MSE, PSNR, and SNR values for each of the three corrupted components across all provided variances, this allowed for us to create three graphs. Each graph was for MSE, PSNR, and SNR. Each graph then further contained three plots. Each of these plots represented the individual corrupted Y, Cb, or Cr component and their calculated values for the specific image metric across all the variances. These values were plotted accordingly in different colours. The values were calculated and saved in nine different vectors. These vectors included the date for MSE - Y Channel, MSE - Cb Channel, MSE - Cr Channel for each variance, PSNR - Y Channel, PSNR - Cb Channel, PSNR - Cr Channel for each variance, and SNR - Y Channel, SNR - Cb Channel, SNR - Cr Channel for each variance. These vectors were used as the data for these plots. Full codebase is in the zip. file.