CSC 477/577

Set 577

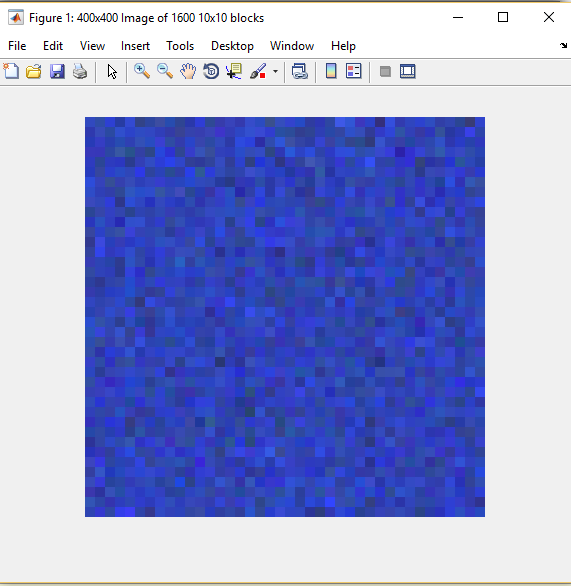
Submitted by: **Sridhar Mocherla**

**Introduction/Abstract:**

The assignment has three parts. In Part A, we are provided a set of camera sensitivities for a set of wavelengths. We try to produce an image with a randomly generated light spectra values which is scaled so that resulting RGB lies in [0,255]. We use the C=L\*R formula to generate the resulting RGB and display it in a 400x400 image. We generate noise of order 10 and observe RMS for R, G and B channels and overall.

In Part B, we observe high RMS values for sensitivities with a vector of simulated light spectra values and try to reduce the variation by quadratic optimization. In Part C, we try to find vanishing points for two different images by drawing parallel lines.

1. **400x400 image**



**Figure 1:** 400x400 image made of 1600 10x10 blocks of uniform RGB scaled to [0,255]

**Context:**

The image in Figure 1 is bluish because the spectral sensitivity values of Blue is much higher than the Red and Green counterparts. Below is the code block which illustrates how the 400x400 image is created by repeating 10x10 blocks.

mod\_resp\_mat = sens\_mat\*mod\_light\_mat;

new\_max = max(mod\_resp\_mat(:));

red\_vec = mod\_resp\_mat(1,:);

green\_vec = mod\_resp\_mat(2,:);

blue\_vec = mod\_resp\_mat(3,:);

red\_block=[];

green\_block=[];

blue\_block=[];

%repeat 10x10 blocks and have 40 per row

for i=1:40:1561

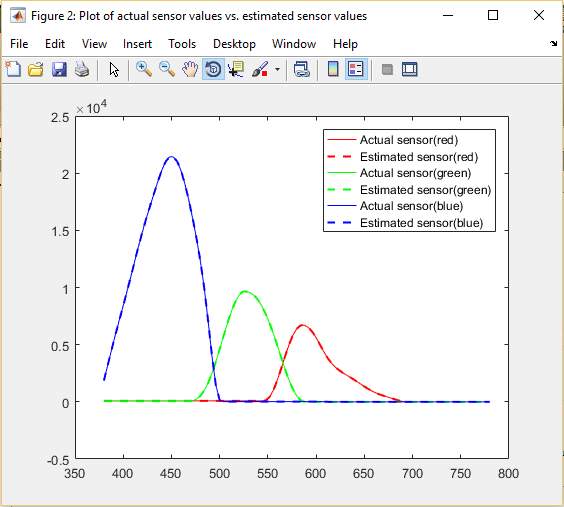
red\_block = [red\_block;repelem(red\_vec(i:i+39),10,10)];

green\_block = [green\_block;repelem(green\_vec(i:i+39),10,10)];

blue\_block = [blue\_block;repelem(blue\_vec(i:i+39),10,10)];

end

1. **Sensitivity curves vs Wavelengths and RMS**



**Figure 2:** Sensitivity vs Wavelengths curve – Actual sensitivities and estimated sensitivities from least squares method almost overlap

**Context:**

Figure 2 shows sensitivity values vs wavelengths of light used. There are two sensitivity curves for R, G and B. The continuous curve indicates the actual sensitivities provided and the dotted line curve indicates the sensitivities estimated using the least squares method. In this case, they almost overlap as seen. Hence, the estimated sensitivity values are very accurate.

We also compute the RMS errors for **sensor values for R, G and B**. The values are as below

red\_sens\_rms (RMS value for sensor for red) = 9.2164e-12

green\_sens\_rms (RMS value for sensor for green) = 1.4121e-11

blue\_sens\_rms (RMS value for sensor for blue) = 3.3335e-11

We also compute the RMS errors for **generated R, G and B values**. The values are as below

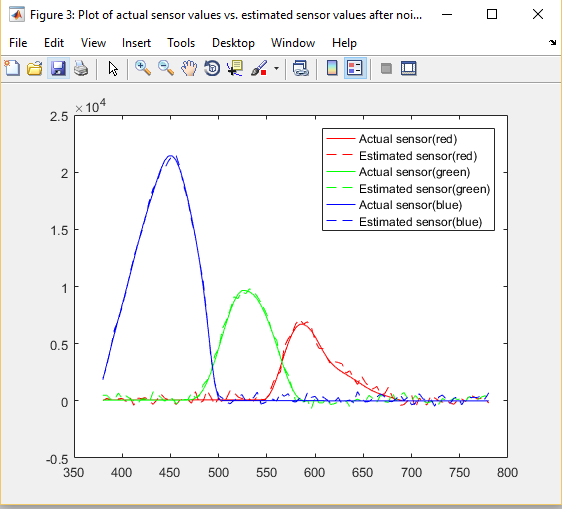
red\_rms = 6.5589e-14

green\_rms = 1.1241e-13

blue\_rms = 1.9986e-13

As we the values are very low and almost near and this was also indicative in the plot.

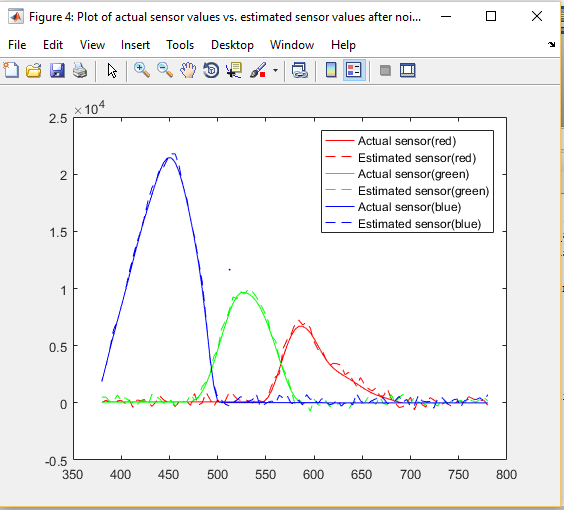
1. **Noise simulation and clipping**



**Figure 3:** Sensitivity curves estimates vs actual after addition of noise of order 10

**Context:**

A normally distributed noise of order 10 is added to the response RGB values. The spectral sensitivities estimated as a result exhibits deviation from the actual sensitivities. We can clearly see in Figure 3 that dotted curves (estimated) and continuous curves (actual) are apart by a certain distance.



**Figure 4:** Sensitivities (estimated and actual) after addition of noise of order 10 and clipping by scale factor.

The curves in Figure 3 and Figure 4 do not differ significantly. The clipping factor is 0.98 (maximum value is 259.3) and hence clipping does not alter estimated sensitivities significantly.

Now we compute the RMS errors for the sensor values for R, G and B separately. The values are as below:

red\_clipped\_sens\_rms = 353.0348 (RMS error for Sensor values for red)

green\_clipped\_sens\_rms = 318.4662 (RMS error for Sensor values for green)

blue\_clipped\_sens\_rms = 332.6003 (RMS error for Sensor values for blue)

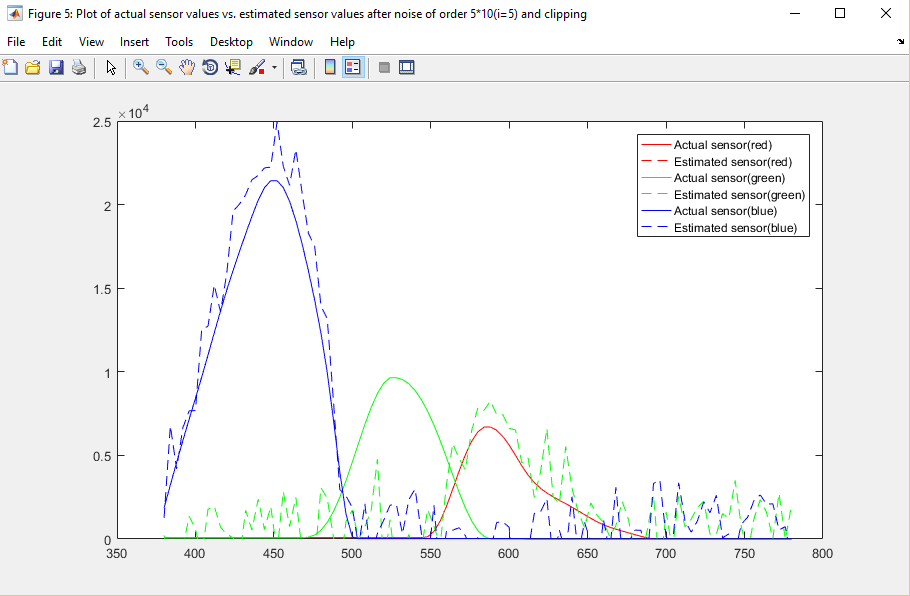
We also compute the RMS errors for the generated R, G and B as in the previous problem. The values are as below:

red\_clipped\_rms = 5.1567 (RMS error for Red response value)

green\_clipped\_rms = 4.9980 (RMS error for Green response value)

blue\_clipped\_rms = 4.9938 (RMS error for Blue response value)

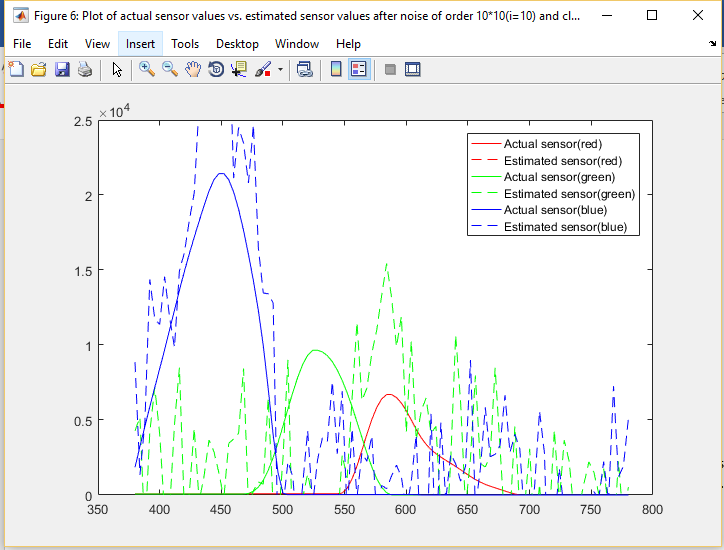
1. **Increase in noise by order of i**



**Figure 5:** Sensitivity curves (estimated vs actual) with added noise (order 5\*10) and clipping to the responses (i=5)

**Context:**

The noise of order 50 is added to the responses and the sensitivities is estimated from linear least squares method. In figure 5, we see that there is greater difference between the estimated and actual sensitivities. We now see the plot for i=10 below in Figure 6. There is a clear pattern here – increase in the noise added to the response leads to an increase in the estimated sensitivities and hence a deviation from actual values measured.



**Figure 6:** Sensitivities (actual vs estimates) with responses augmented with noise of order 10\*10 (i=10)

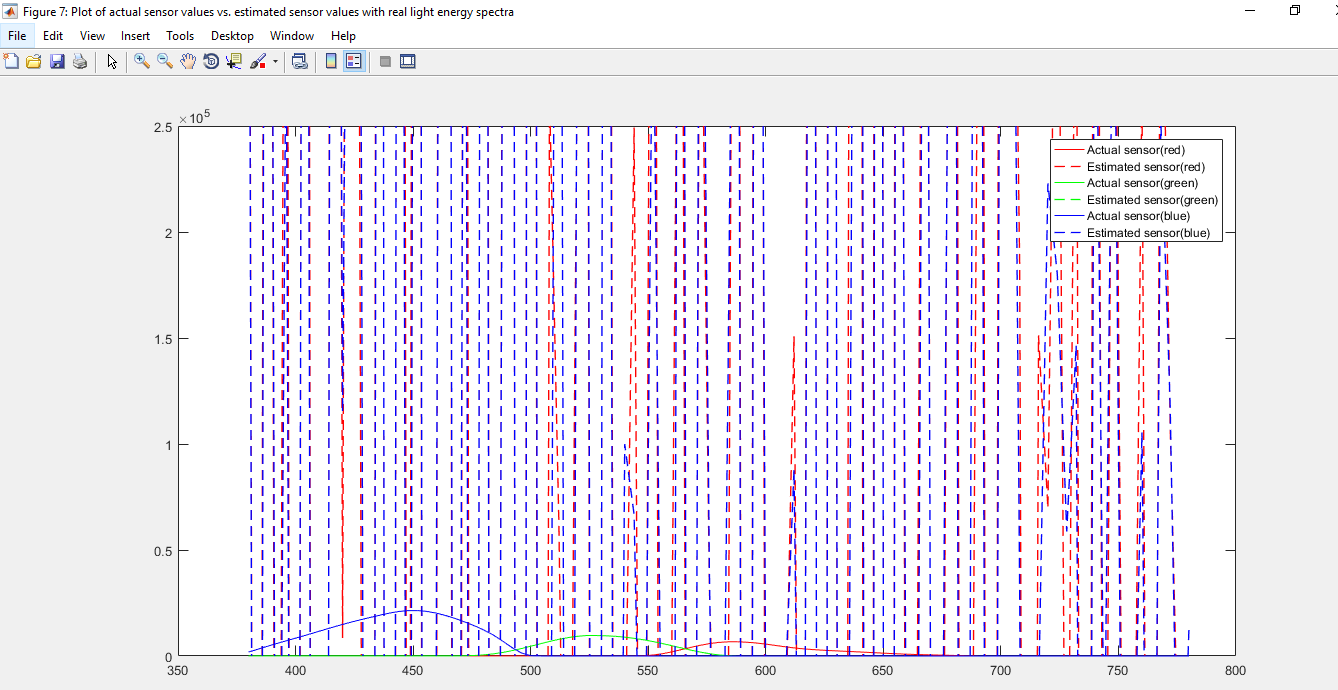
We provide the 11x2 matrix which shows the RMS (overall) errors for the estimated sensor values for R,G and B without clipping and with clipping as below

|  |  |
| --- | --- |
| Without clipping (starts from i=0) | With clipping (starts from i=0) |
| 0.000000000021568 | 0.000000000030155 |
| 316.263721434636 | 351.157598560686 |
| 596.541951517369 | 673.795793203383 |
| 841.215305072618 | 1079.54707933075 |
| 1151.81483390584 | 1400.15358512518 |
| 1415.58487956890 | 1666.24448872726 |
| 1788.25709587196 | 2187.36623507505 |
| 2040.35468119647 | 2675.76158824365 |
| 2584.91298740651 | 3213.55089219913 |
| 2720.43853360913 | 3838.79980570427 |
| 2912.60609686543 | 4104.64327272446 |

**Table 1:** 11x2 matrix with overall RMS values for i=0 to 10 without clipping and with clipping

In Table 1, The RMS error increases with increase in order of noise, which is already noticeable in the sensitivity curves. This is true for both clipping and without clipping. The magnitude in increase in RMS error seems to be higher in case of clipping. This might be due to a scalar constant introduced by clipping to restrict RGB range to [0,255].

1. **Sensitivities using simulated light spectra data**



**Figure 7:** Sensitivities (actual vs estimated) based on values in light\_spectra.txt and responses.txt

**Context:**

As suggested in the question, there is an incredible difference in the estimated sensor values and the actual sensor values. We investigate the light spectra matrix to find out the possible reason behind this.

We use the condition number (cond() function in MATLAB) to compare the condition numbers for the light spectra matrix provided and a randomly generated matrix of the same. A high condition number indicates that the matrix is ill-conditioned. A condition number of a matrix is the ratio of the largest singular value in the matrix to the smallest singular value in the matrix. A high condition number means a huge variation in the distribution of values in the matrix and there is a high chance of accuracy being lost in computation.

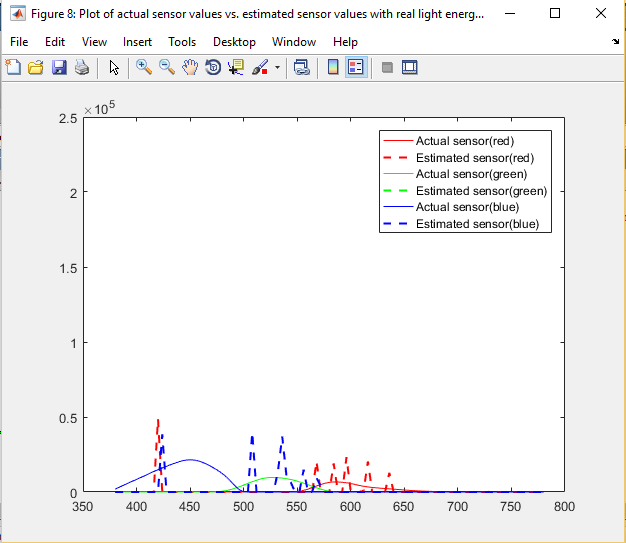
Condition numbers of the light spectra matrix and a randomly generated matrix is as below

Cond(spectra) = 42,955

Cond(rand(598,101)) = 28.9448

We see that a randomly generated matrix has much lower condition number than the spectral values.

1. **Constrained least squares**



**Figure 8**: Sensitivities (Actual vs estimated) after quadratic programming is used to constrain the spectral values

**Context:**

We model the formula L\*R = C in the form of a quadratic equation and use the constraint (solution must be positive) and then compute the parameters H,f,A and b required to compute the red, green and blue sensor values. The code snippet for this is as below

H = spectra'\*spectra;

f = -spectra' \* resp\_mat;

red\_f = f(:,1);

green\_f = f(:,2);

blue\_f = f(:,3);

A = -eye(101);

b = -zeros(101,1);

red\_sensor = quadprog(H,red\_f,A,b);

green\_sensor = quadprog(H,green\_f,A,b);

blue\_sensor = quadprog(H,blue\_f,A,b)

Figure shows the sensitivity values obtained using quadprog(). The values are constrained and many of the values for each of the estimated plots is zero.

1. **Derivative**

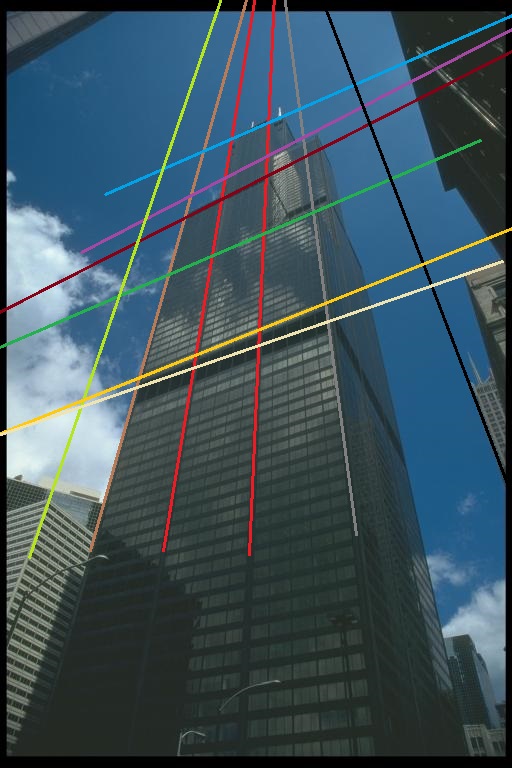
The matrix D is computed by subtracting consecutive rows and storing them as rows in a new matrix. In MATLAB, we can compute D by using diff() function.

sens\_positive\_mat = [red\_sensor,green\_sensor,blue\_sensor];

D = diff(sens\_positive\_mat);

**Part - C**

1. **Images and vanishing points**:



**Figure 9:** Building image with parallel lines drawn along the plane

**Context:**

In Figure 9, we draw several lines (many pairs of them parallel to each other) along the plane of the building. We observe a couple of pairs of parallel lines would eventually intersect based on their trajectory. The yellow, two red and gray lines are likely to meet if extended further at a specific point. Another set of lines (horizontal, yellow and white) which are parallel are also likely to meet at a further point. Both these points can be called *vanishing points*. Traditional objects have 1 to 3 vanishing points so the building is real and in perspective.

C:\Users\sridh\AppData\Local\Microsoft\Windows\INetCacheContent.Word\chandelier.tiff

**Figure 10:** Chandelier image with lines drawn among possible points to find if vanishing points exist

**Context:**

In Figure 10, we draw several lines through the small circles and few of these are parallel. However, none of the parallel when extended seem to any point. Hence, there won’t be any vanishing points for this image and a result the image is not in perspective. The pair of parallel lines colored orange and purple will not meet when extended and remain parallel. Another pair of parallel lines colored red and sky blue also when extended do not meet at any further point. Hence, it can be believed that the image is not in perspective.