Homework 2

Meta information	
Name	Savan Kiran
Program	Masters in Computer Science
Questions skipped	N/A
Questions substituted	N/A
Extra credit questions	N/A

PART A

1. 1600 random light spectra

Generated 1600 random light spectra as 101 element row vectors using `rand()` function. Computed the response for the random light spectra against the given sensor values. The response values had to be scaled to the range [0,255] to be represent as RGB values. The linear RGB responses had to be reshaped into 40x40 matrix composed of 10x10 pixel blocks. I used `reshape` and `repelem` functions in matlab to achieve it.

```
%Q1
%   Display an image that visualizes all 1600 RGB values derived from
the
%   given sensor data and random light spectrum generated
num_questions=num_questions+1;
S=importdata('rgb_sensors.txt');
rng(477);
L=rand([1600,101]);
C=L*S;
maxC=max(C(:));
C_RGB=C.*(255/maxC);
imdata=reshape(C_RGB(1:1600,:),[40,40,3]);
imdata=repelem(imdata,10,10);
imdata=uint8(imdata);
figure('Name','Q1 - Image','NumberTitle','off');
imshow(imdata);
```

Please find the image of 40x40 grid in Figure 1.

2. Sensor error calculation

Using the responses calculated earlier, we estimate the sensor values again (as though we know nothing about the actual sensor values). We plot the actual and estimated sensor values as line plots depicted in Figure 2. Both estimated and actual sensor values have 3 channels, namely Red, Green and Blue(RGB) represented by their respective colors. The estimated sensors are plotted as dotted lines whereas actual sensors are plotted as dashed lines. Since both values are very similar, it is difficult to notice the difference.

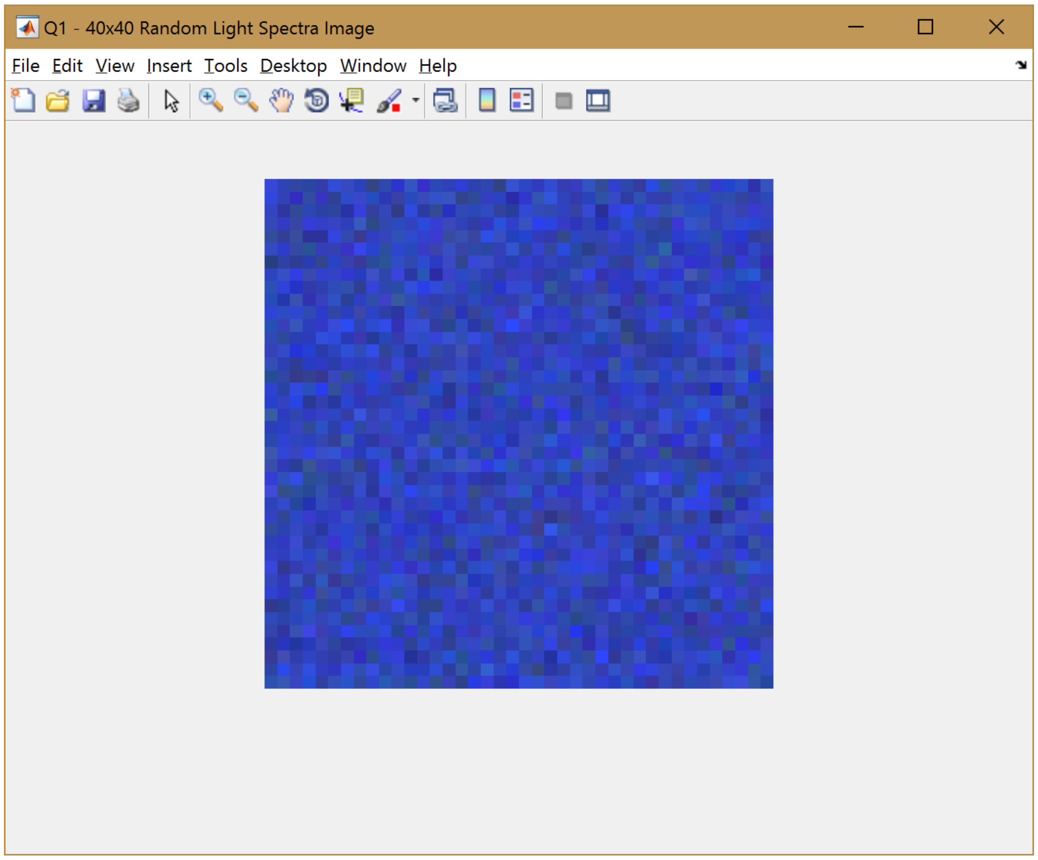
Q1 - 40x40 Random Light Spectra Image
File Edit View Insert Tools Desktop Window Help

Figure 1. 40x40 Random Light Spectra Visualization with given RGB sensor values

We now calculate the RMS error of the 3 (RGB) sensor values and the 3 (RGB) response components to better understand the difference between estimated and actual sensors. Below is a summary of the errors and we can see that it is nearly 0 which explains the almost perfect overlap of the two curves.

```
RMS error between real and estimated sensor -
Red
   6.9287e-11
Green
   9.8093e-11
Blue
   2.6927e-10
RMS error between RGB values calcuated using real and estimated sensor
-
Red
   2.2018e-13
Green
   3.1921e-13
Blue
   8.8425e-13
```

Below is the code snippet for Q2.

```
%Q2
%   1. Comparing real sensor values with estimated values.
%       a. dotted line - est values
%       b. dash line - real values
%   2. RMS error between est. sensor and real sensor values for RGB
%   channels.
%   3. RMS error between R,G,B channels calculated using est. sensor
and
%   real sensor values.
num_questions=num_questions+1;
S_est=(inv(L'*L)*L')*C;
display_plot(S,S_est,'Q2 - Sensors - real v/s estimated
plot',0,2.5*(10.^4));

[rmsEr_S_R,rmsEr_S_G,rmsEr_S_B]=rms_error(S,S_est);
text='RMS error between real and estimated sensor - ';
display_rms_error(text,rmsEr_S_R,rmsEr_S_G,rmsEr_S_B);

C_est=L*S_est;
maxC_est=max(C_est(:));
C_RGB_est=C_est.*(255/maxC_est);
[rmsEr_C_R,rmsEr_C_G,rmsEr_C_B]=rms_error(C_RGB,C_RGB_est);
text='RMS error between RGB values calcuated using real and estimated
sensor -';
display_rms_error(text,rmsEr_C_R,rmsEr_C_G,rmsEr_C_B);
```

Below code shows how RMS error is calculated.

```
function [rms_r,rms_g,rms_b] = rms_error(A, A_est)
A_diff=(A-A_est).^2;
rms_r=sqrt(mean(A_diff(:,1)));
rms_g=sqrt(mean(A_diff(:,2)));
rms_b=sqrt(mean(A_diff(:,3)));
end
```

Below code displays the sensors as line plots.

```
function display_plot(A, A_est, text, y_ll, y_ul)
wavelengths=380:4:780;
figure('Name',text,'NumberTitle','off');
plot(wavelengths,A_est(:,1),':r');
hold on
plot(wavelengths,A_est(:,2),':g');
plot(wavelengths,A_est(:,3),':b');
plot(wavelengths,A(:,1),'--r');
plot(wavelengths,A(:,2),'--g');
plot(wavelengths,A(:,3),'--b');
axis([350 780 y_ll y_ul]);
legend('Est. sensor(R)','Est. sensor(G)','Est. sensor(B)','Real
sensor(R)','Real sensor(G)','Real sensor(B)');
end
```

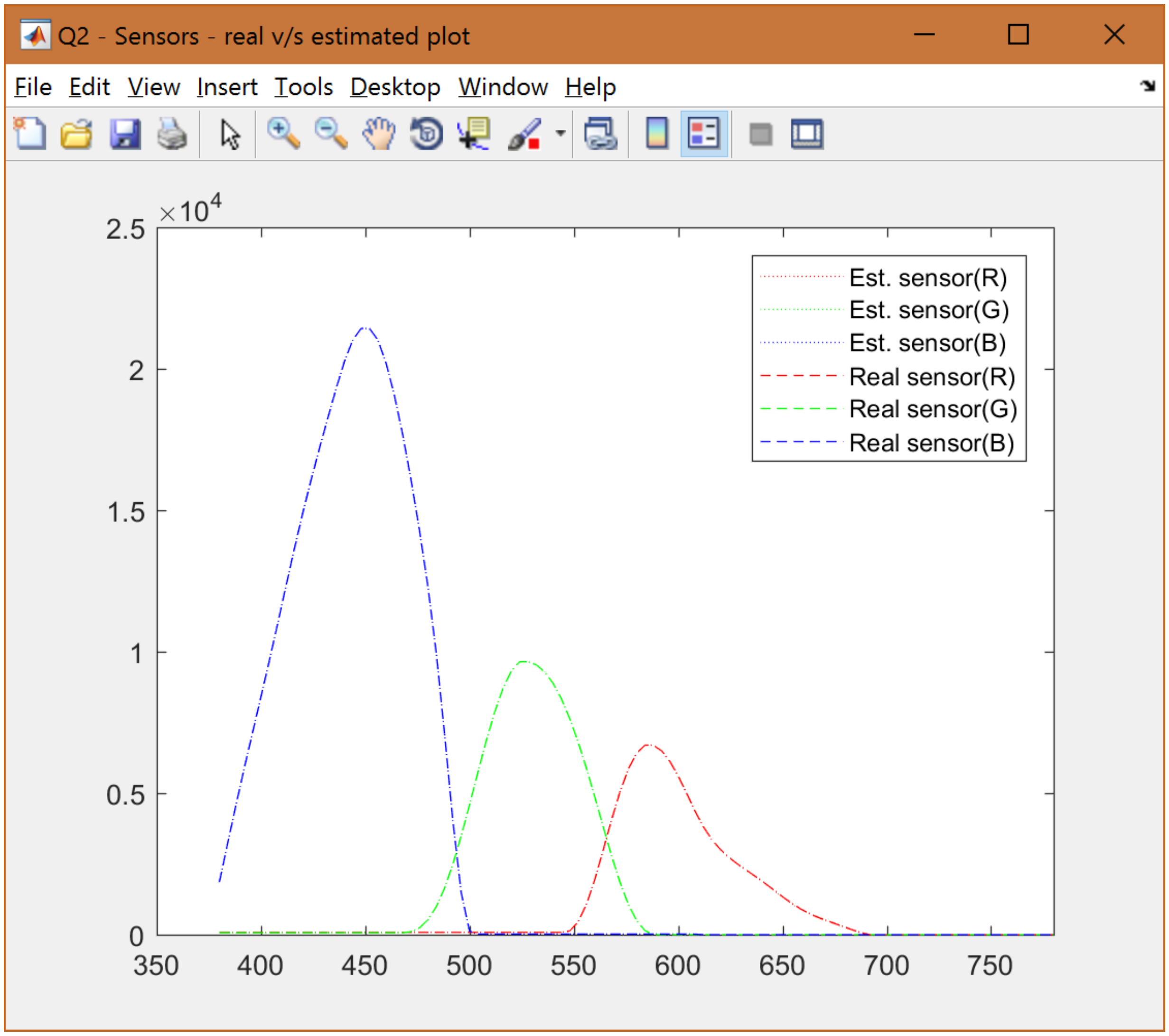

Figure 2. Sensors estimated from responses v/s actual sensors. Estimated in dotted line and actual in dashed line

3. Sensor error calculation – with Gaussian noise.

We will now add Gaussian noise that is normally distributed with a standard deviation of 10 to each of the R,G,B responses for 1600 colors. Using these new responses (w/ noise), we estimate the sensors again and plot them (refer to Figure 3). Introduction of noise created responses that is no longer in [0,255] range. We therefore, clip the RGB responses to fit in [0,255] range and eliminate negative values and values >255. Now re-calculate the sensors and plot them (refer to Figure 4).

As before, we calculate the RMS error between the actual and estimated sensors for the sensor values and RGB responses. The values are shown in Table 1.

RGB error w/ noise - Red 25.6288 Green 26.0089 Blue 25.1175	RGB error w/ noise w/ clipping- Red 24.9295 Green 25.8337 Blue 24.5670
Sensor error w/ noise - Red 2.6473e+03 Green 2.5405e+03 Blue 2.2288e+03	Sensor error w/ noise w/ clipping - Red 2.5945e+03 Green 2.5108e+03 Blue 2.1950e+03

Table 1. RMS error between estimated and actual sensors for sensor values and RGB responses with noise (sd=10)

Below is the code snippet for all the error calculations and plotting.

```
%Q3
%   1. Comparing real sensor values with estimated values w/ Gaussian
Noise.
%       a. dotted line - est values
%       b. dash line - real values
num_questions=num_questions+1;
sig=10;
t1=['Q3 - Sensors w/ Noise at Noise sd:', num2str(sig)];
t2=['Q3 - Sensors w/ Noise clipped at Noise sd:', num2str(sig)];
[C_n,S_n,C_n_c,S_n_c]=compute_w_noise(1,sig,true,L,S,C,maxC,t1,t2);
[er_Cn_R,er_Cn_G,er_Cn_B]=rms_error(C_RGB,C_n);
[er_Sn_R,er_Sn_G,er_Sn_B]=rms_error(S,S_n);
[er_Cnc_R,er_Cnc_G,er_Cnc_B]=rms_error(C_RGB,C_n_c);
[er_Snc_R,er_Snc_G,er_Snc_B]=rms_error(S,S_n_c);

text='RGB error w/ noise -';
display_rms_error(text,er_Cn_R,er_Cn_G,er_Cn_B);
text='Sensor error w/ noise -';
display_rms_error(text,er_Sn_R,er_Sn_G,er_Sn_B);

text='RGB error w/ noise w/ clipping-';
display_rms_error(text,er_Cnc_R,er_Cnc_G,er_Cnc_B);
text='Sensor error w/ noise w/ clipping -';
display_rms_error(text,er_Snc_R,er_Snc_G,er_Snc_B);
```

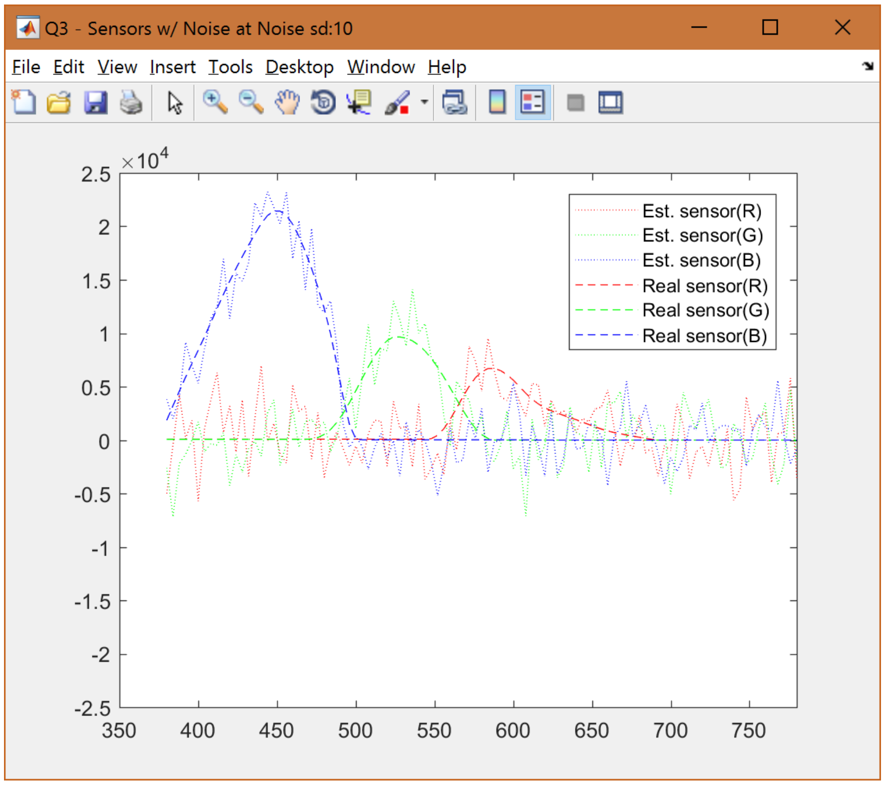

Figure 3. Sensors w/ noise at sd=10

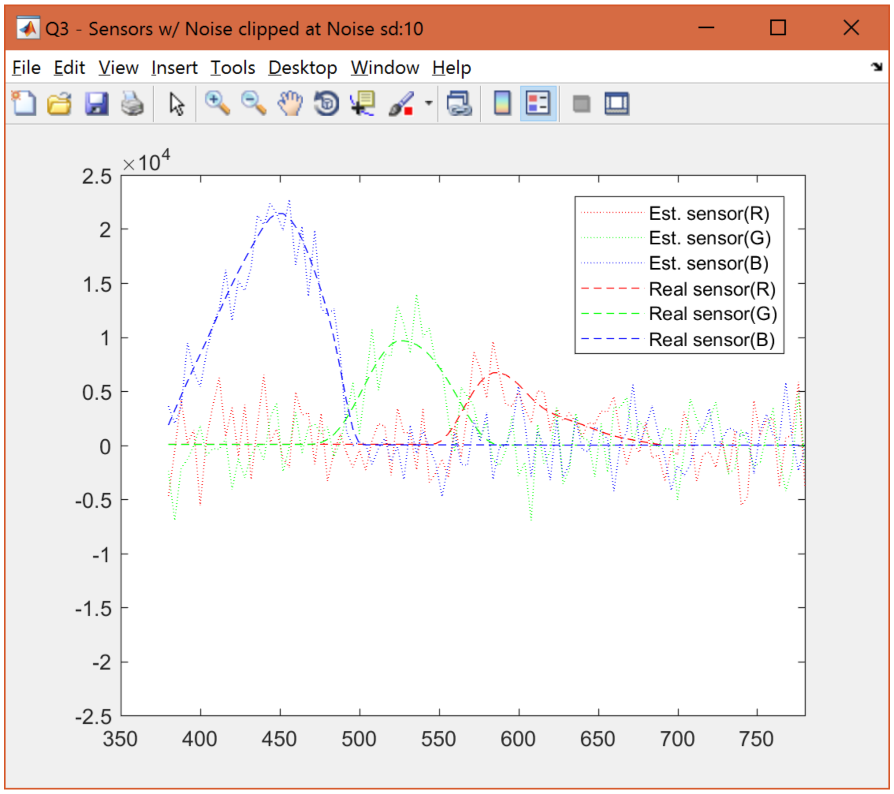

Figure 4. Sensors w/ noise at sd=10 and clipped responses

4. Sensor error calculation – multiple levels of noise

In this part, we calculate the RMS error for standard deviations (sd) of noise from 0->100 at intervals of 10. Unlike last part, we calculate 1 RMS error value for all 3 R,G,B parts combined. We provide the plots for sd=50 (Figure 5 & 6) and sd=100 (Figure 7 & 8) and RMS error values for all the sd values (Table 2).

Standard deviation	w/ Noise, w/o Clipping		w/ Noise, w/ Clipping	
	RGB error	Sensor error	RGB error	Sensor error
0	0.0009433	0.020435	0.0009432	0.020477
10	25.529	2418.4372	25.0726	2400.6761
20	51.1442	4812.7217	45.7015	4266.2202
30	75.2094	7473.2314	62.4575	6223.3966
40	101.1194	9171.3149	77.7263	7088.3161
50	127.0021	11947.0075	87.5567	8075.9398
60	153.2086	13956.9503	96.3153	8710.3243
70	180.9506	17158.0497	104.287	9308.863
80	202.4461	19294.576	107.0567	9997.5018
90	231.1524	20208.7896	113.7735	9593.0951
100	254.0916	22433.0709	115.9068	10534.8501

Table 2. RGB and Sensor RMS error values for w/ and w/o clipping for varying sd values

As expected, with increase in sd of noise, RMS error increase for both w/ and w/o clipping. The RMS error is generally lower for w/ clipping when compared to its w/o clipping counterpart because the max error is bounded because of clipping.

Below is the code snippet for the same.

```
for i=0:10
    sig=i*10;
    t1=['Q4 - Sensors w/ Noise at Noise sd:', num2str(sig)];
    t2=['Q4 - Sensors w/ Noise clipped at Noise sd:', num2str(sig)];

[C_n,S_n,C_n_c,S_n_c]=compute_w_noise(1,sig,mod(i,5)==0&&i>0,L,S,C,max
C,t1,t2);
    [er_Cn_R,er_Cn_G,er_Cn_B]=rms_error(C_RGB,C_n);
    [er_Sn_R,er_Sn_G,er_Sn_B]=rms_error(S,S_n);
    [er_Cnc_R,er_Cnc_G,er_Cnc_B]=rms_error(C_RGB,C_n_c);
    [er_Snc_R,er_Snc_G,er_Snc_B]=rms_error(S,S_n_c);
    er_Cn=sqrt(mean([er_Cn_R,er_Cn_G,er_Cn_B].^2));
    er_Sn=sqrt(mean([er_Sn_R,er_Sn_G,er_Sn_B].^2));
    er_Cnc=sqrt(mean([er_Cnc_R,er_Cnc_G,er_Cnc_B].^2));
    er_Snc=sqrt(mean([er_Snc_R,er_Snc_G,er_Snc_B].^2));
    disp(['w/ noise sd:', num2str(sig), ' RGB err:', num2str(er_Cn), '
Sensor err:', num2str(er_Sn)]);
    disp(['w/ noise w/ clipping sd:', num2str(sig), ' RGB err:',
num2str(er_Cnc), ' Sensor err:', num2str(er_Snc)]);
end
```

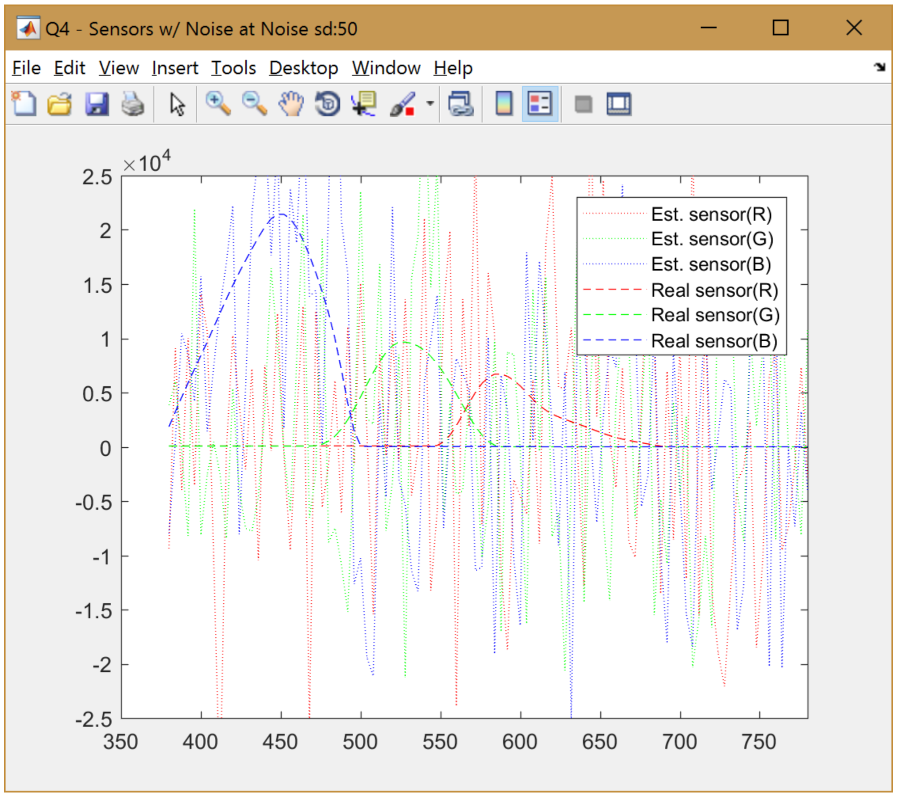

Figure 5. Sensors w/ noise at sd=50

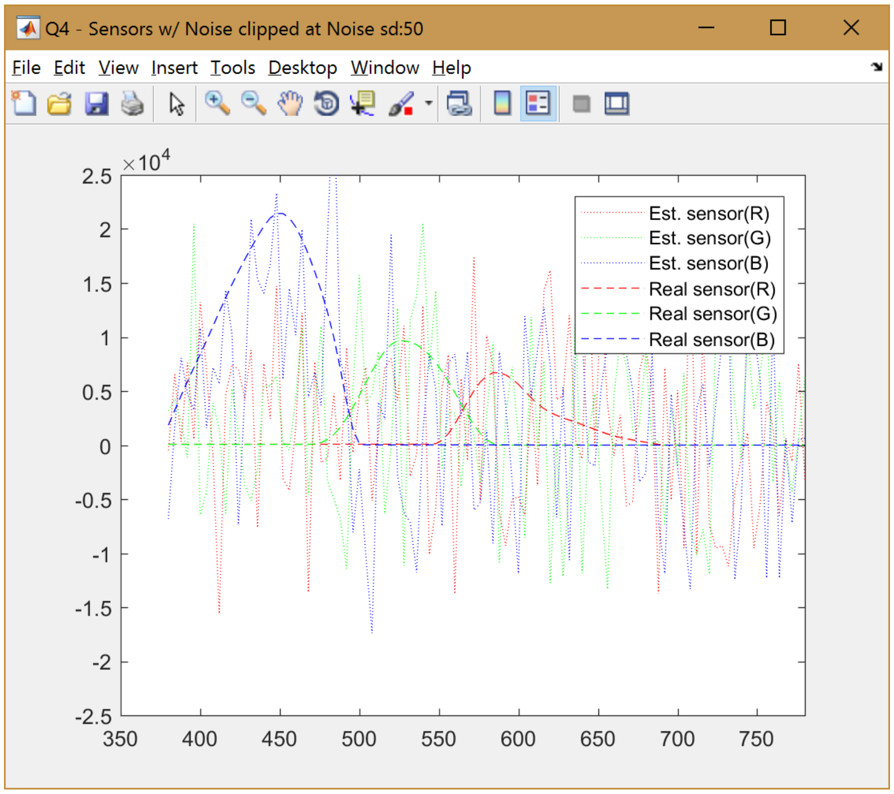

Figure 6. Sensors w/ noise at sd=50 and clipped responses

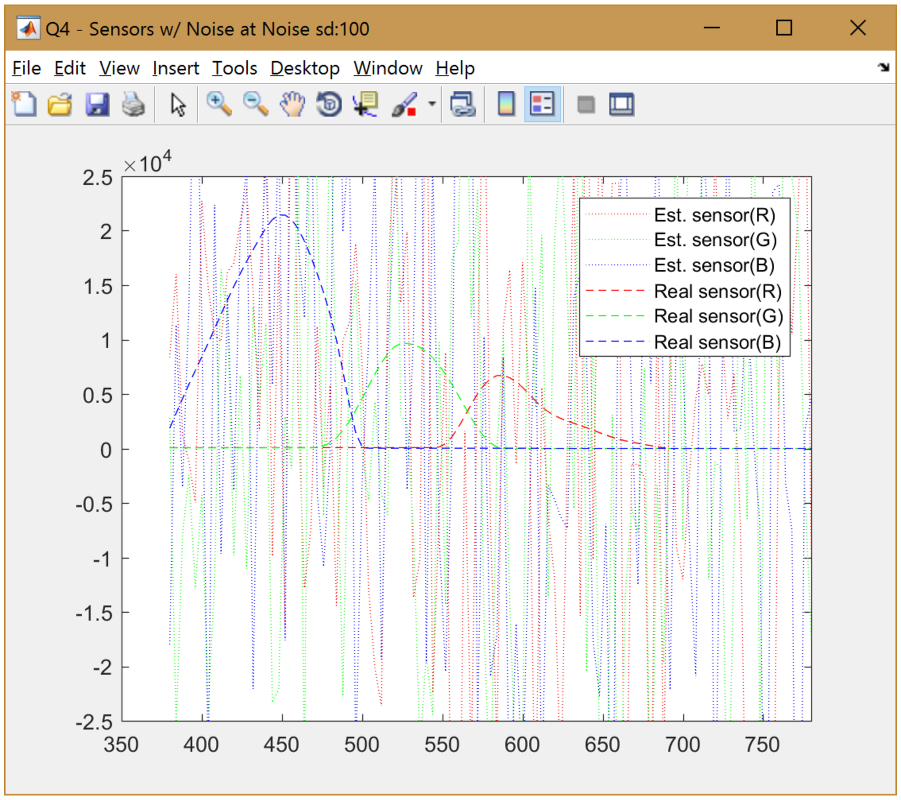

Figure 7. Sensors w/ noise at sd=100

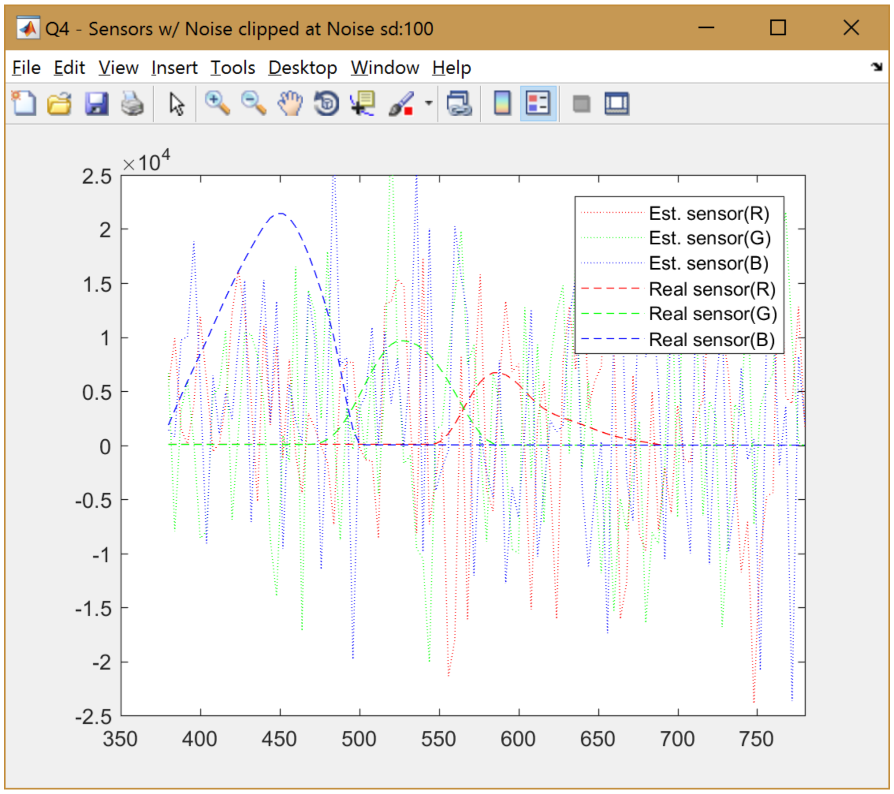

Figure 8. Sensors w/ noise at sd=100 and clipped responses

Part B

5. Gamma correction

When the gray apple was matched with stripes viewed at a distance by the user, the value was 80 on [0,255] scale. We've Gamma correction transformation given by,

$$F(x) = 255 * (\frac{x}{255})^{\frac{1}{2.2}}$$

$$Substituting\ x = 80, we\ get$$

$$F(80) = 255 * \left(\frac{80}{255}\right)^{\frac{1}{2.2}}$$

$$F(80) = 150.556$$

We have calculated the Gamma correction function that helps us represent the non-linear images on a linear modern display by adding the gamma correction. This helps us see the pictures in the color space that we're used to see rather than the linear space. On the other hand, when the image is captured, we do the reverse, i.e., inverse Gamma correction, to save the real-world image in non-linear space as linear image that can be understood by the displays/monitors.

PART C

6. Sensitivity estimation using real light spectra and responses

Using the (new) real light spectra and the corresponding R,G,B responses, we estimate the sensors. We plot the sensors for estimated and actual in terms of R,G,B channels as before in Figure 9. We also calculate the RMS errors (values below) for it and find that it is extremely large.

```
Sensor RMS error (Real light spectra)-
Red
   1.2573e+06
Green
   9.0844e+05
Blue
   8.2289e+05
RGB RMS error (Real light spectra)-
Red
    7.4110
Green
    5.3863
Blue
    3.8726
```

We can conclude that the sensor values derived from real-world data are terrible by looking at the large RMS errors. Exploring further with condition number on the real light spectra and condition number on random light spectra, we can observe an enormous difference. The real light spectra has condition number in the order of 10^4, meaning an error in least significant bit of LHS can result in error up to lower 4 digits on RHS. Whereas, for the random light spectra, the condition number was only 29. I used the `cond` function to find the condition number.

```
Condition number
   Random light spectra: 29.3655
   Real light spectra: 4.2955e+04
```

This is probably because of a lot of noise present in the real light spectra because of hitting and reflecting from many surfaces which makes a limited dimensionality light appear to have all 101 dimensionalities when in reality, it doesn't.

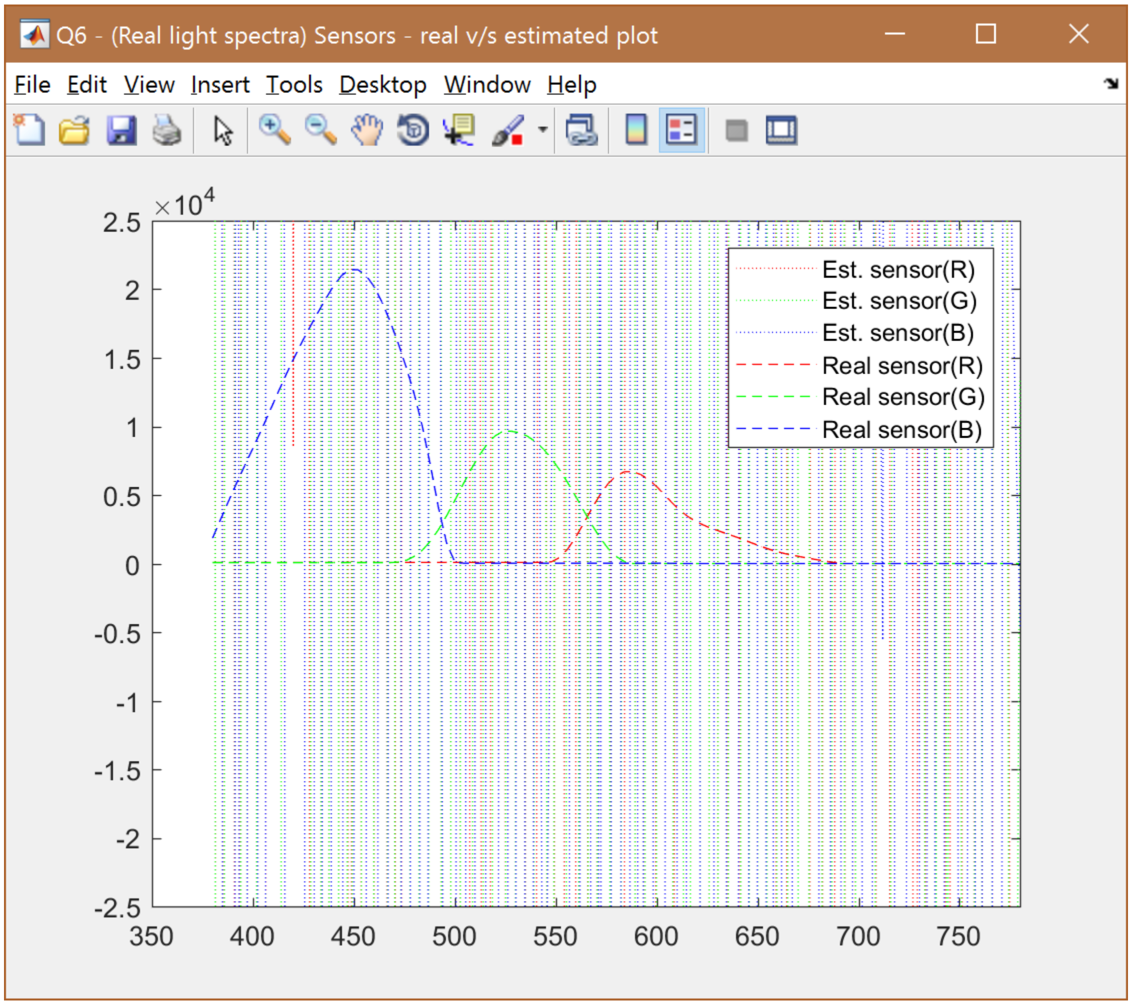

Figure 9. Sensor (estimated v/s real) derived from real light spectra

The code snippet for Q6 is in the following page.

7. Sensitivity estimation using constrained least squares

We could improvise the sensitivity estimation by limiting the sensitivity vector to be non-negative through constrained least squares. We can use `quadprog` function in matlab to do the same. The results are plotted on Figure 10.

```
%Q6
%   1. Estimate sensitivity for real light spectra and RGB responses
%   2. Plot the sensors and calculate RMS errors
L2=importdata('light_spectra.txt');
C2=importdata('responses.txt');
S_est2=(inv(L2'*L2)*L2')*C2;
display_plot(S,S_est2,'Q6 - (Real light spectra) Sensors - real v/s 
estimated plot',-2.5*(10.^4),2.5*(10.^4));

[rmsEr_S2_R,rmsEr_S2_G,rmsEr_S2_B]=rms_error(S,S_est2);
text='Sensor RMS error (Real light spectra)- ';
display_rms_error(text,rmsEr_S2_R,rmsEr_S2_G,rmsEr_S2_B);

C_est2=L2*S_est2;
[rmsEr_C2_R,rmsEr_C2_G,rmsEr_C2_B]=rms_error(C2,C_est2);
text='RGB RMS error (Real light spectra)- ';
display_rms_error(text,rmsEr_C2_R,rmsEr_C2_G,rmsEr_C2_B);

rng(477);
L_temp=rand([598,101]);
disp('Condition number, random light spectra');
disp(cond(L_temp));
disp('Condition number, real light spectra');
disp(cond(L2));
```

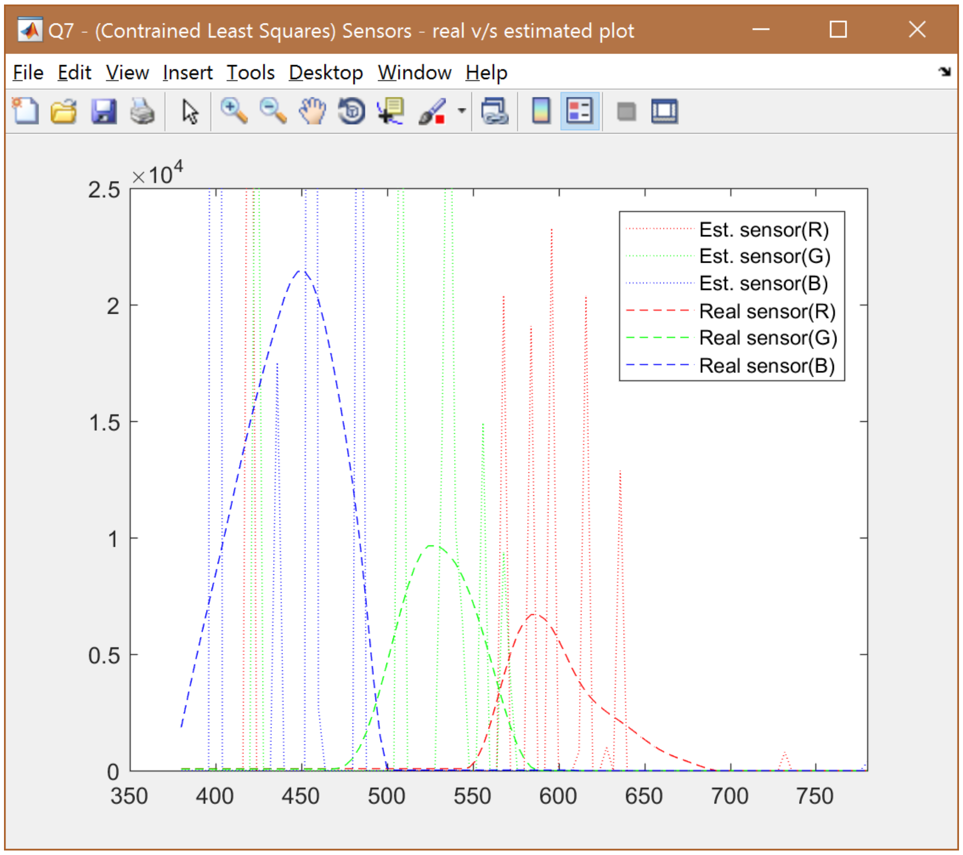

Figure 10. Sensor (estimated v/s real) estimated using constrained least squares

Below is the code snippet for constrained least squares calculations using `quadprog`.

```
%Q7
%   Use constrained least squares to improve the sensor estimation
H=L2'*L2;
f=-L2'*C2;
r_f=f(:,1);
g_f=f(:,2);
b_f=f(:,3);
A=-eye(101);
b=-zeros(101,1);

r=quadprog(H,r_f,A,b);
g=quadprog(H,g_f,A,b);
b=quadprog(H,b_f,A,b);

S_cls=[r g b];
display_plot(S,S_cls,'Q7 - (Contrained Least Squares) Sensors - real 
v/s estimated plot',0,2.5*(10.^4));
```

8. Smoothening the estimated sensor

We now have a non-negative sensor but it still varies a lot and is not smooth. One way to smoothen is to push the derivative towards zero. Computing successive differences of rows in the light spectra is how we calculate the derivative. The matrix `M` is one which has 1, -1 in each row with a circular shift of 1 for every next row. We use this in its transposed form to calculate the vector `D` of successive differences.

We then solve $(\lambda M) * R = 0$ in a least squares setup and use λ for controlling the smoothness. We need to augment the 100 rows from vector D to the light spectra and corresponding RGB responses as well (which will be 0 as the response to the derivative is to be pulled to 0).

Figures 11 through 15 show estimated sensor v/s real sensor plots for following values of lambda [90,100,120,140,150]. Lambda values 90 and 100 are too low whereas lambda values 140 and 150 are too high. Lambda values 120 is just about right where we see the 3 peaks of estimated RGB sensors almost the same actual/real sensors.

Find the figures in the following pages. Find the code snippet below.

```
function Q8(L2,C2,S,lambda)
[n_rows,n_cols]=size(L2);
M=zeros(n_cols-1,n_cols);
for r=1:n_cols-1
    M(r,r)=1;
    M(r,r+1)=-1;
end
M=M.*lambda;

%D2=diff(L2(1:101,:));
D2=L2(1:101,:)*M';
L2_aug=[L2; D2'];
C2_aug=[C2; zeros(100,3)];
```

```
H=L2_aug'*L2_aug;
f=-L2_aug'*C2_aug;
r_f=f(:,1);
g_f=f(:,2);
b_f=f(:,3);
A=-eye(101);
b=-zeros(101,1);

rSensor=quadprog(H,r_f,A,b,[],[]);
gSensor=quadprog(H,g_f,A,b,[],[]);
bSensor=quadprog(H,b_f,A,b,[],[]);

S_cls_2=[rSensor gSensor bSensor];
display_plot(S,S_cls_2,['Q8 - Lambda='
num2str(lambda)],0,2.5*(10.^4));
end
```

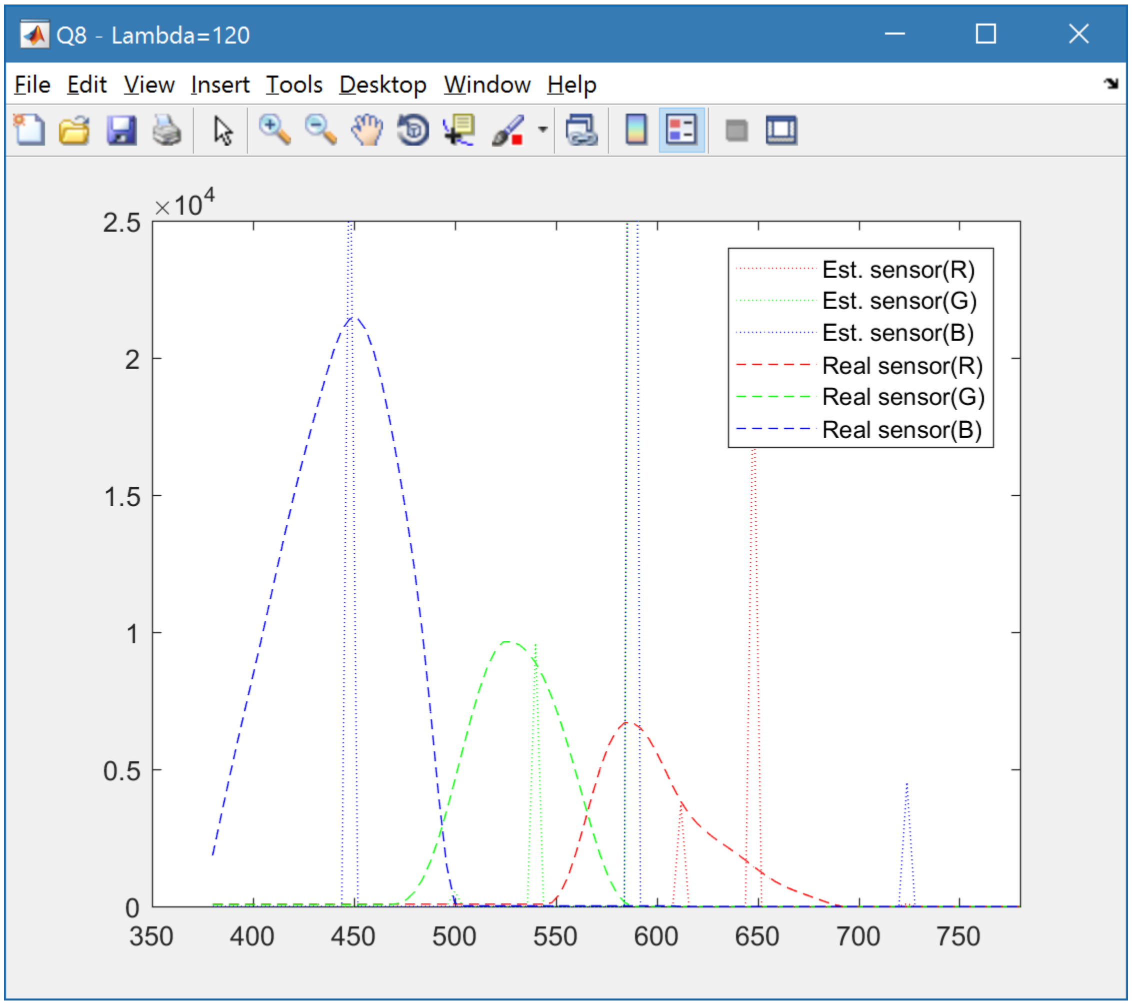

Figure 11. Sensor values for lambda=120 (right lambda value)

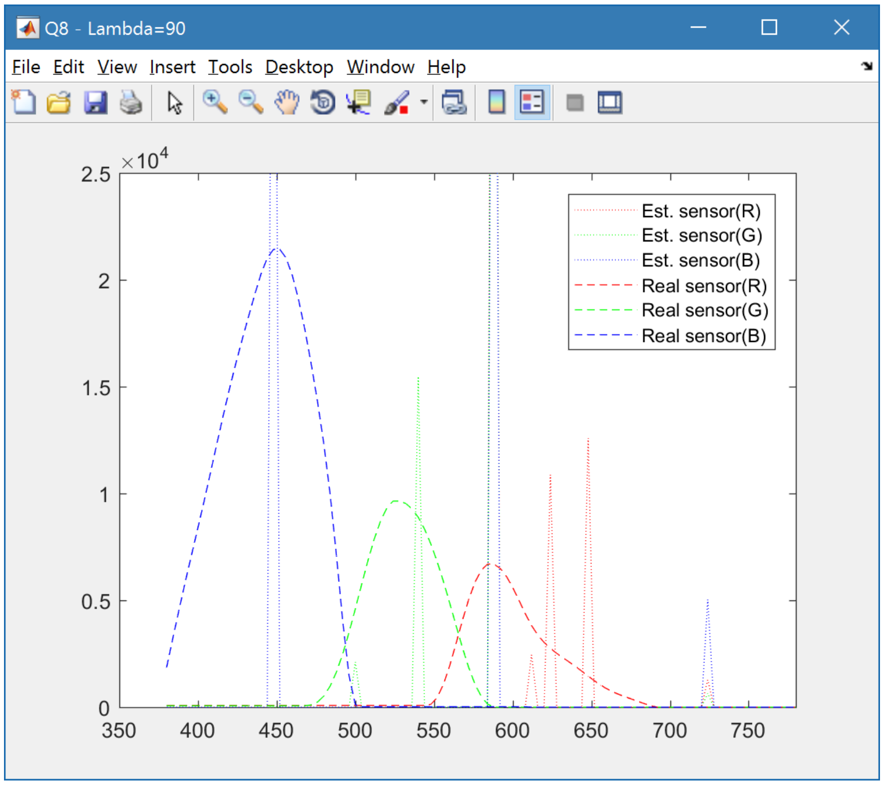

Figure 12. Sensor values for lambda=100 (too less)

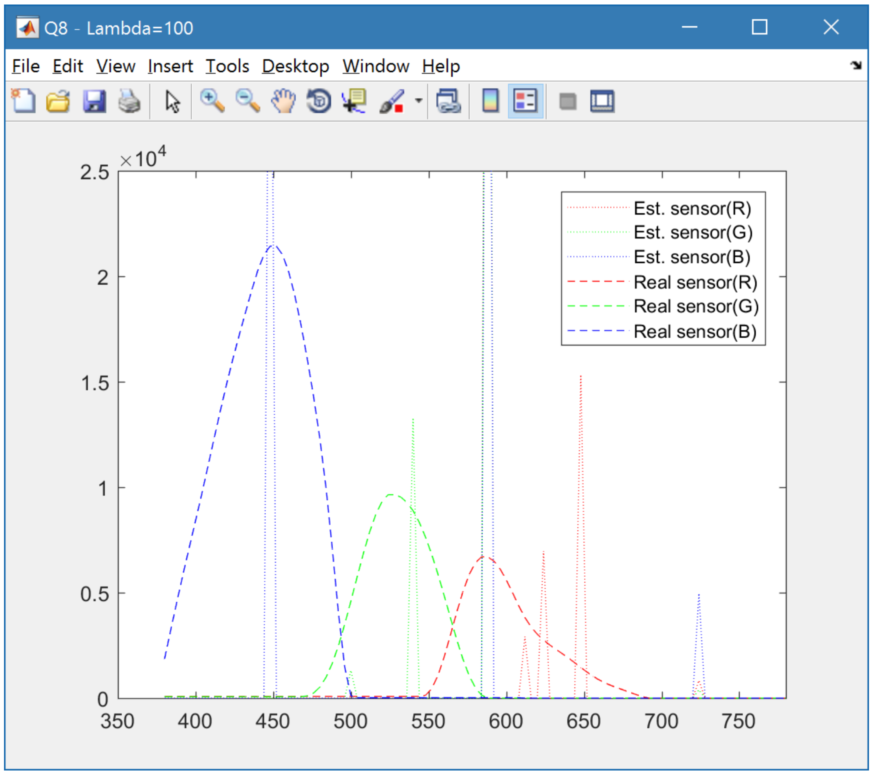

Figure 13. Sensor values for lambda=100 (too less)

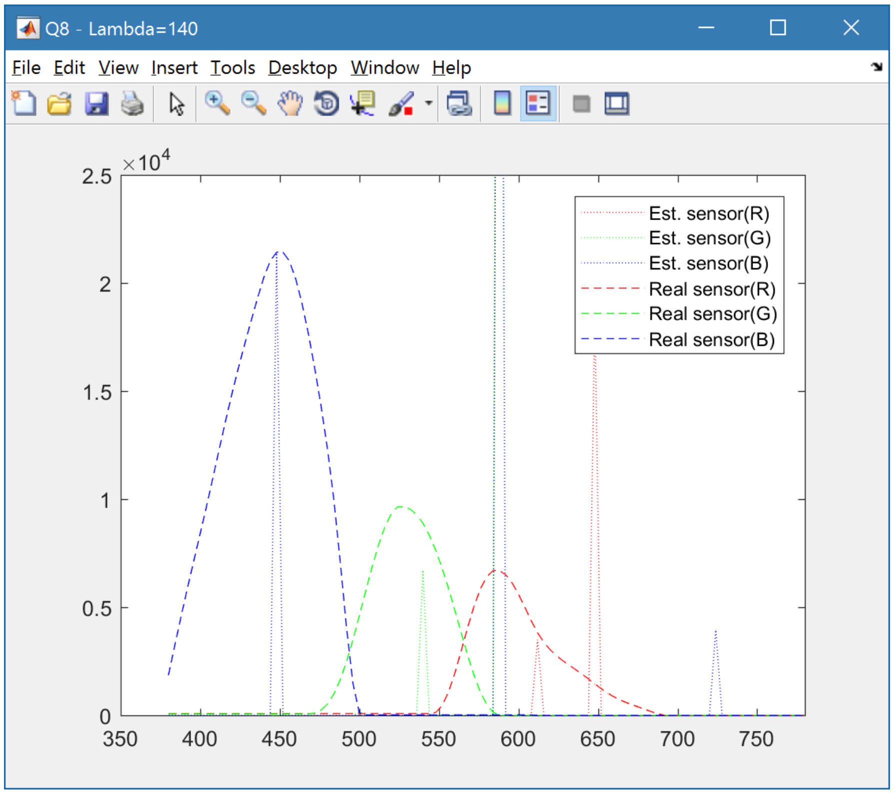

Figure 14. Sensor values for lambda=140 (too high)

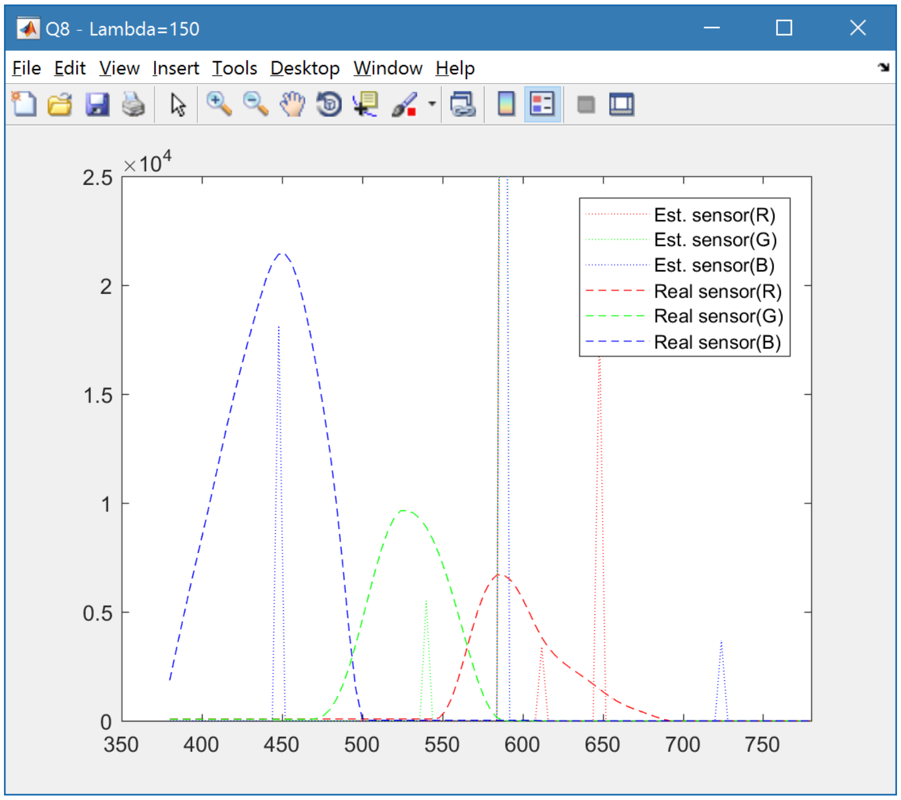

Figure 15. Sensor values for lambda=150 (too high)