# MA5710 - Assignment 2

Abhinav T K - MA23M002

September 14, 2023

1. Consider the first question [The number of cherries in a can] in Assignment-1. Formulate the model as Black box model.

   **For black box model:**
   Input:
   **r**: Radius of a cherry.
   **R**: Radius of the can.
   **h**: Height of the can.
   Output:
   **n** : The number of cherries that can be packed into the can.

   First we randomly generate data for the radius of can **R**, height of can **h** and radius of cherry **r** in MATLAB.

   Volume of cherry $= \frac{4}{3}\pi r^3$
   Volume of can $= \pi R^2 h$
   Number of cherries that can be fit, **n**:

   $$= \frac{\text{Volume of can}}{\text{Volume of cherry}} = \frac{\pi R^2 h}{\frac{4}{3}\pi r^3} = \frac{3}{4}\frac{R^2 h}{r^3}$$

   We will use this formula to calculate the number of cherries for each input sample randomly generated in MATLAB.

   **For Upper Bound:** From the above equation for no. of cherries, we can take upper bound as $\mathbf{\frac{3}{4}\frac{R^2h}{r^3}}$ assuming there are no gaps or overlaps of cherries while packing.

   **For Lower Bound:** We can take the lower bound to be **0** considering no cherries are packed in the cylinder. This happens if the cherry radius is greater than the height of the can.

   **Code: ma23m002_a2_q1.m**

```
%% Formulate The number of cherries in a can problem as Black box model.

clear;
clc;
rng(321);                          % Set a random seed
n = 1000;                          % No. of samples

lb_r = 10;                         % Lower bound of the cherry radius
ub_r = 20;                         % Upper bound of the cherry radius
lb_R = ub_r + 1;                   % Lower bound of the can radius
ub_R = 100;                        % Upper bound of the can radius
lb_h = 10;                         % Lower bound of the can height
ub_h = 100;                        % Upper bound of the can height

% Generating random numbers for cherry radius from a normal distribution
cherryR = betarnd(3,3,n,1) * (ub_r - lb_r) + lb_r;

 % Generating random numbers for can radius from a normal distribution
```

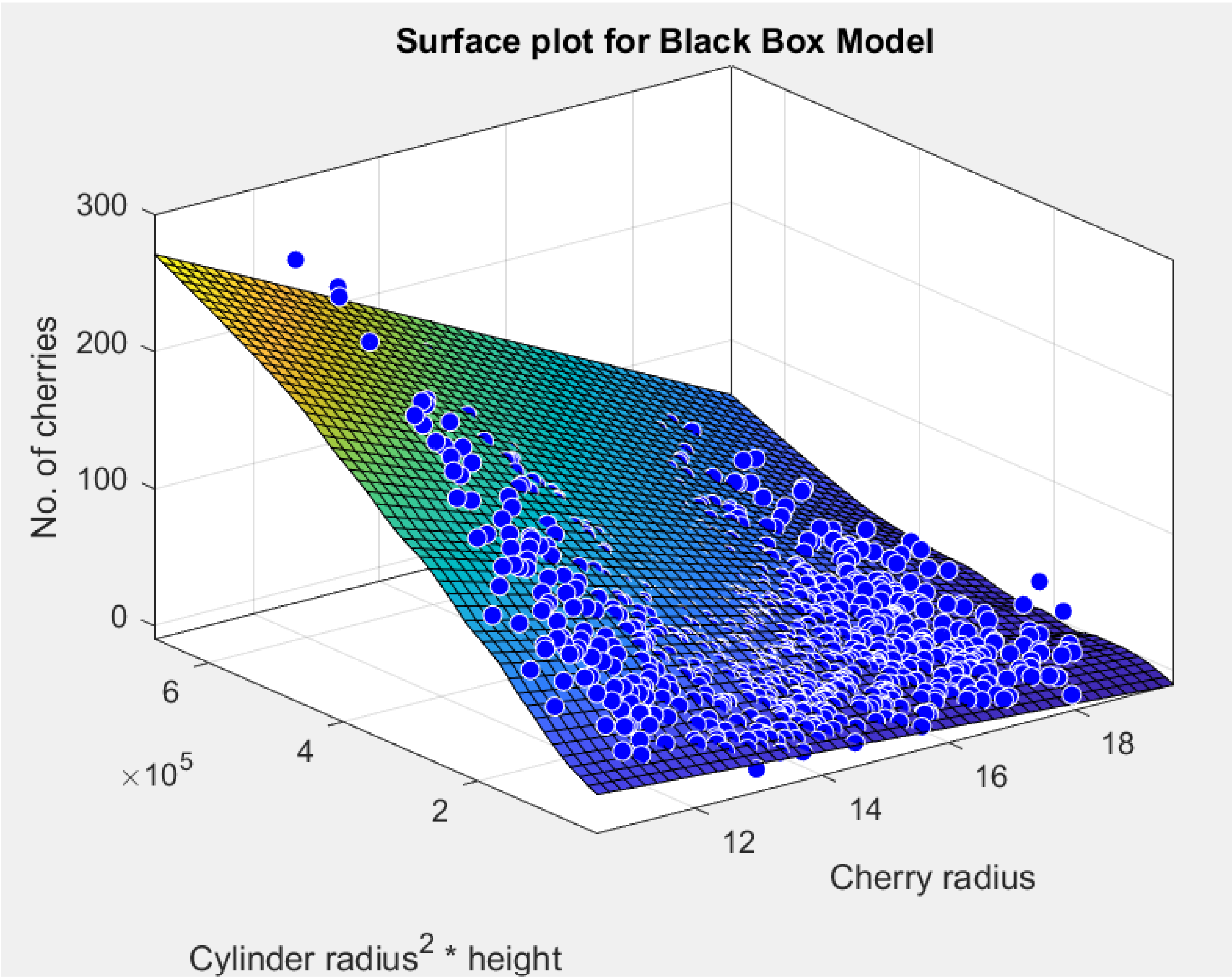
```
canR = betarnd(3,3,n,1) * (ub_R - lb_R) + lb_R;

 % Generating random numbers for can height from a normal distribution
canH = betarnd(3,3,n,1) * (ub_h - lb_h) + lb_h;

cherryV = (4/3)*pi*(cherryR.^3);              % Volume of cherry
canV = pi*(canR.^2).*canH;                    % Volume of can
nCherry = floor(canV./cherryV);               % No. of cherries

% Table with data for surface plot
% Here there are two independent variables, x1 = cherryR ,
% x2 = canR.^2.*canH and a dependent variable n - no. of cherries
data = table(cherryR, canR.^2.*canH, nCherry, 'VariableNames', {'x1', 'x2', 'n'});

% Split the data into training (85%) and testing (15%)
rng(111);  % Set different random seed
partition = cvpartition(n, 'HoldOut', 0.15);     % Splitting using cvpartition
trainData = data(training(partition), :);        % train data
testData = data(test(partition), :);             % test data

% converting train data table to array for fit function
X_train = table2array(trainData(:,1:2));

% converting test data table to array for prediction
X_test = table2array(testData(:,1:2));

% converting train output to array for fit function
y_train = table2array(trainData(:,3));

% converting test output to array for prediction
y_test = table2array(testData(:,3));

% Fitting the model with Lowess smoothing model
model = fit(X_train,y_train,'lowess');

% Surface plot
figure(1);
plot(model,X_train,y_train)
xlabel('Cherry radius');
ylabel('Cylinder radius^2 * height');
zlabel('No. of cherries');
title('Surface plot for Black Box Model');

y_predicted = feval(model, X_test);     % Predict values for test data using our
    fitted model

% Calculate model's performance on the test data
error = sum((y_test - y_predicted).^2);
RMSE = sqrt(error / length(y_predicted));     % Root Mean Squared Error

disp(['Root Mean Squared Error (RMSE) on test data: ', num2str(RMSE)]);
```

**Output:**

For random data generation:
Cherry radius, r: Lower bound = 10, Upper bound = 20
Can radius, R: Lower bound = 21, Upper bound = 100
Can height, h: Lower bound = 10, Upper bound = 100

**Max. number of cherries in train data using fitted model = 159**
**Min. number of cherries in train data using fitted model = 0**
**Root Mean Squared Error (RMSE) on test data: 8.4132**

An excel file containing the random generated data and the output is attached with the mail. **- MA23M002_A2_data.xlsx**



2. Consider the second question [Cascading cups] in Assignment-1. Formulate the model as Black box model. Do not use any result from the White box model.

**For black box model:**

Input:
**N**: The number of cups.
**C**: The capacity of each cup.
**T**: The total time taken to complete the process.

Output:
$\mathbf{W_i}$ : Amount of wine in **i** th cup

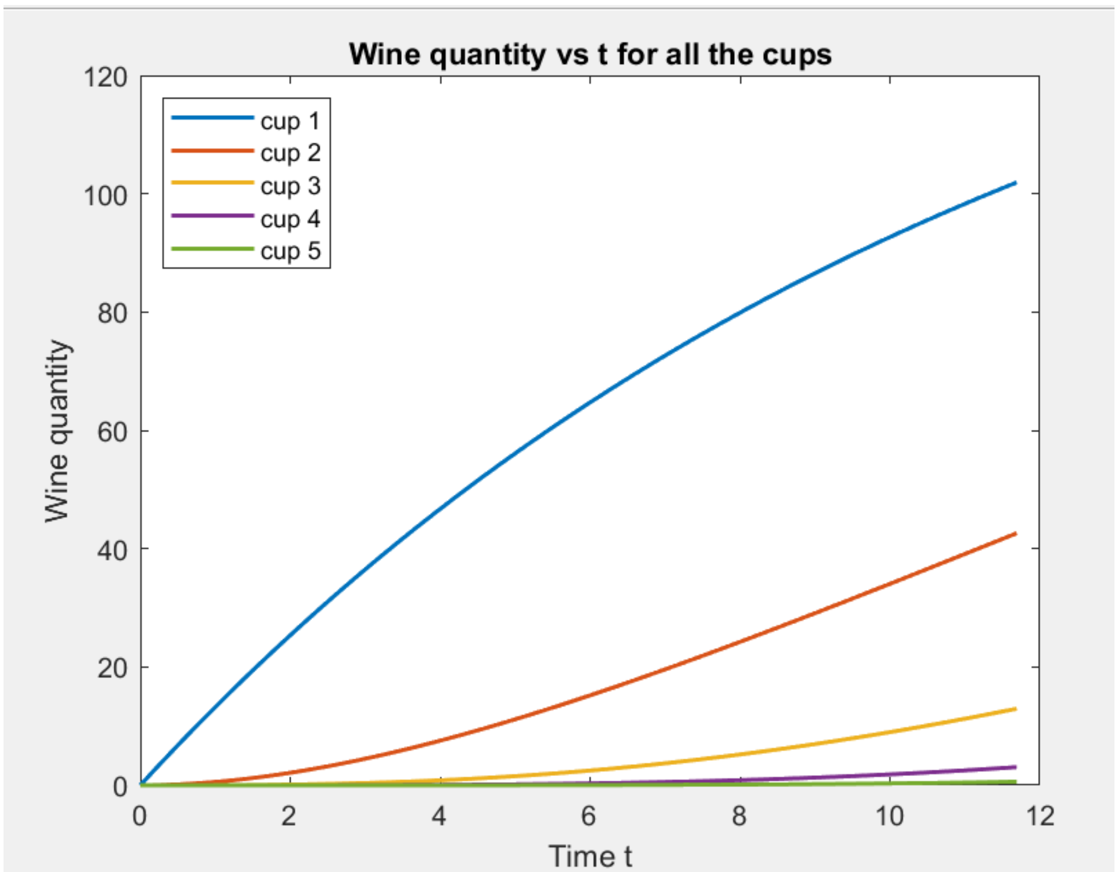
**Code: ma23m002_a2_q2.m**

```
%% Q2. Formulate the Cascading cups model as Black box model.

clear;
clc;
rng(111);
N = 5 ;                     % Number of cups
C = 100 + 100*rand();       % Capacity of each cup generated randomly between 100 to 200
T = 10 + 10*rand();         % Total time taken for the process generated randomly between 10 to 20

timeSpan = [0, T];          % Time interval for consideration
W1_0 = 0.0;                 % W1 value for t = 0
W2_0 = 0.0;                 % W2 value for t = 0
W3_0 = 0.0;                 % W3 value for t = 0
W4_0 = 0.0;                 % W4 value for t = 0
W5_0 = 0.0;                 % W5 value for t = 0

% Using ode45 to solve the system of ODEs for wine quantity in all the cups
[t, W] = ode45(@(t, W) myODESystem(t, W, C, T), timeSpan, [W1_0, W2_0, W3_0, W4_0, W5_0]);

% Display the final quantity of wine in each cup at time T.
disp(['Quantity of wine in Cup 1 at time T: ', num2str(W(end,1))]);
disp(['Quantity of wine in Cup 2 at time T: ', num2str(W(end,2))]);
disp(['Quantity of wine in Cup 3 at time T: ', num2str(W(end,3))]);
disp(['Quantity of wine in Cup 4 at time T: ', num2str(W(end,4))]);
disp(['Quantity of wine in Cup 5 at time T: ', num2str(W(end,5))]);

%Plot the values of W
figure;
plot(t, W,'LineWidth', 1.5);
xlabel('Time t');
ylabel('Wine quantity');
title('Wine quantity vs t for all the cups');
legend('cup 1', 'cup 2', 'cup 3', 'cup 4', 'cup 5','Location','northwest');


% Define the ODE function for series of cups
function dWdt = myODESystem(t, W, C, T)
    W1 = W(1);
    W2 = W(2);
```

```
    W3 = W(3);
    W4 = W(4);
    W5 = W(5);

    % Rate of change of wine in cup i = Inflow from i-1 to i - Outflow from i to i+1
    dW1dt = -W1/T + C/T;          % Rate of change of wine quantity in cup 1
    dW2dt = W1/T - W2/T;          % Rate of change of wine quantity in cup 2
    dW3dt = W2/T - W3/T;          % Rate of change of wine quantity in cup 3
    dW4dt = W3/T - W4/T;          % Rate of change of wine quantity in cup 4
    dW5dt = W4/T - W5/T;          % Rate of change of wine quantity in cup 5
    dWdt = [dW1dt; dW2dt; dW3dt; dW4dt; dW5dt];
end
```

**Output:**

**Quantity of wine in Cup 1 at time T: 101.9086**
**Quantity of wine in Cup 2 at time T: 42.6002**
**Quantity of wine in Cup 3 at time T: 12.946**
**Quantity of wine in Cup 4 at time T: 3.0612**
**Quantity of wine in Cup 5 at time T: 0.59003**

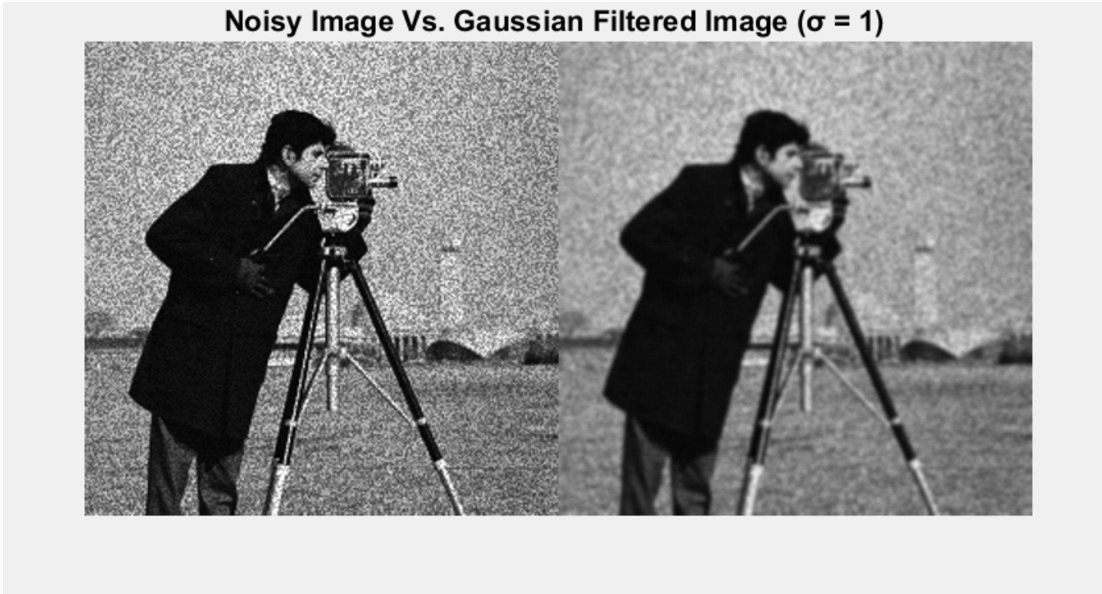


3. **Implement Linear Isotropic Diffusion using inbuilt Gaussian filter function.**

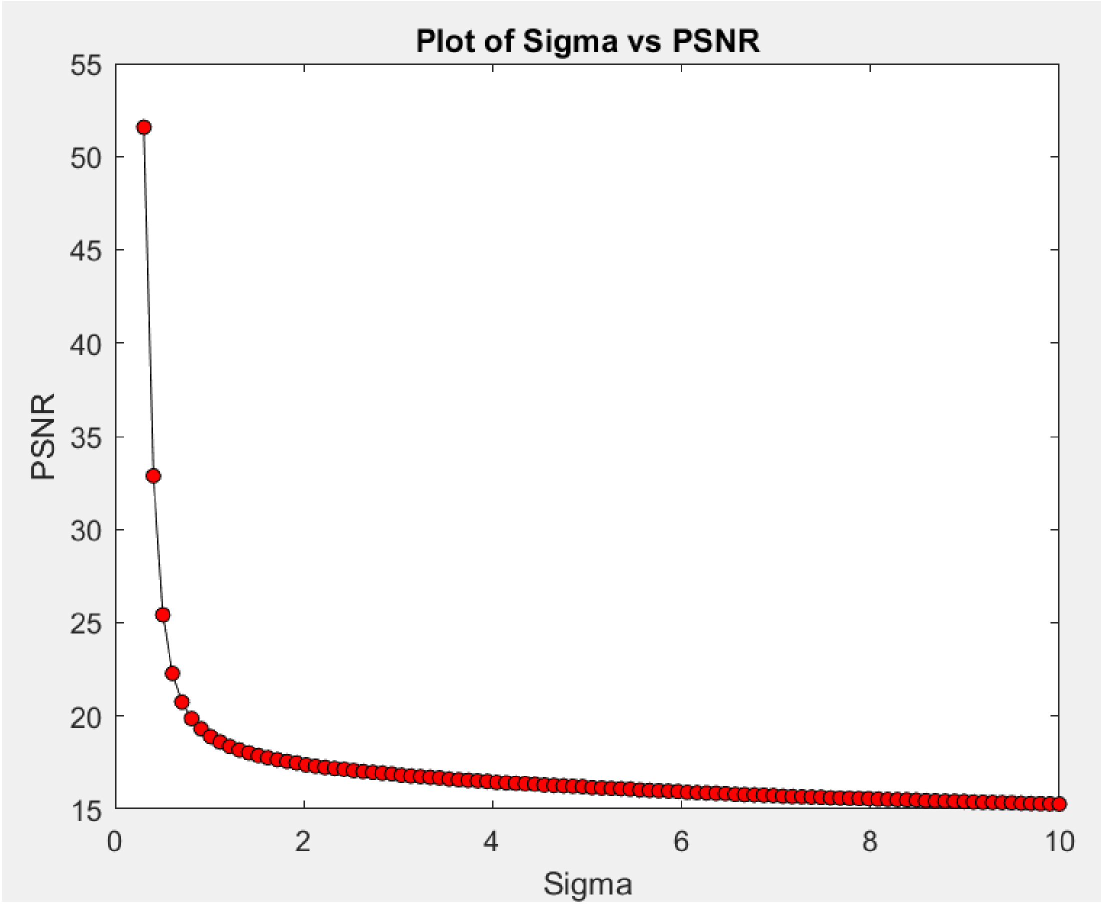
**Code: ma23m002_a2_q3.m**

```
%%Implement Linear Isotropic Diffusion using inbuilt Gaussian filter function.
clear;
clc;

Img = imread('cameraman.tif');          %Take input image
J = imnoise(Img,'speckle');             %add noise to image

n = 100;                               % No. of sigma values
sigma = linspace(0.001,10,n)';         % sigma = Smoothing parameter
peaksnr = zeros(n,1);                  % Initialize PSNR - Metric to understand the quality of cleaning

for i=1:n
    Jfilt = imgaussfilt(J,sigma(i));    %Filter the image with a Gaussian filter with standard deviation sigma(i) to smoothen the noise
    peaksnr(i) = psnr(Jfilt,J);         % PSNR - Metric to understand the quality of cleaning
end

data = [sigma peaksnr];

figure(1);
plot(sigma, peaksnr, 'k-o', "MarkerFaceColor",'r',"MarkerSize",5);
xlabel('Sigma');
ylabel('PSNR');
title('Plot of Sigma vs PSNR');


Jfilt1 = imgaussfilt(J,1);  % Filtered image for sigma = 1
Jfilt5 = imgaussfilt(J,5);  % Filtered image for sigma = 5

figure(2);
montage({J,Jfilt1});    % Compare noise and filtered image
title('Noisy Image Vs. Gaussian Filtered Image (sigma = 1) ');

figure(3);
montage({J,Jfilt5});    % Compare noise and filtered image
title('Noisy Image Vs. Gaussian Filtered Image (sigma = 5)');
```

**Output:**


Noisy Image Vs. Gaussian Filtered Image (σ = 1)

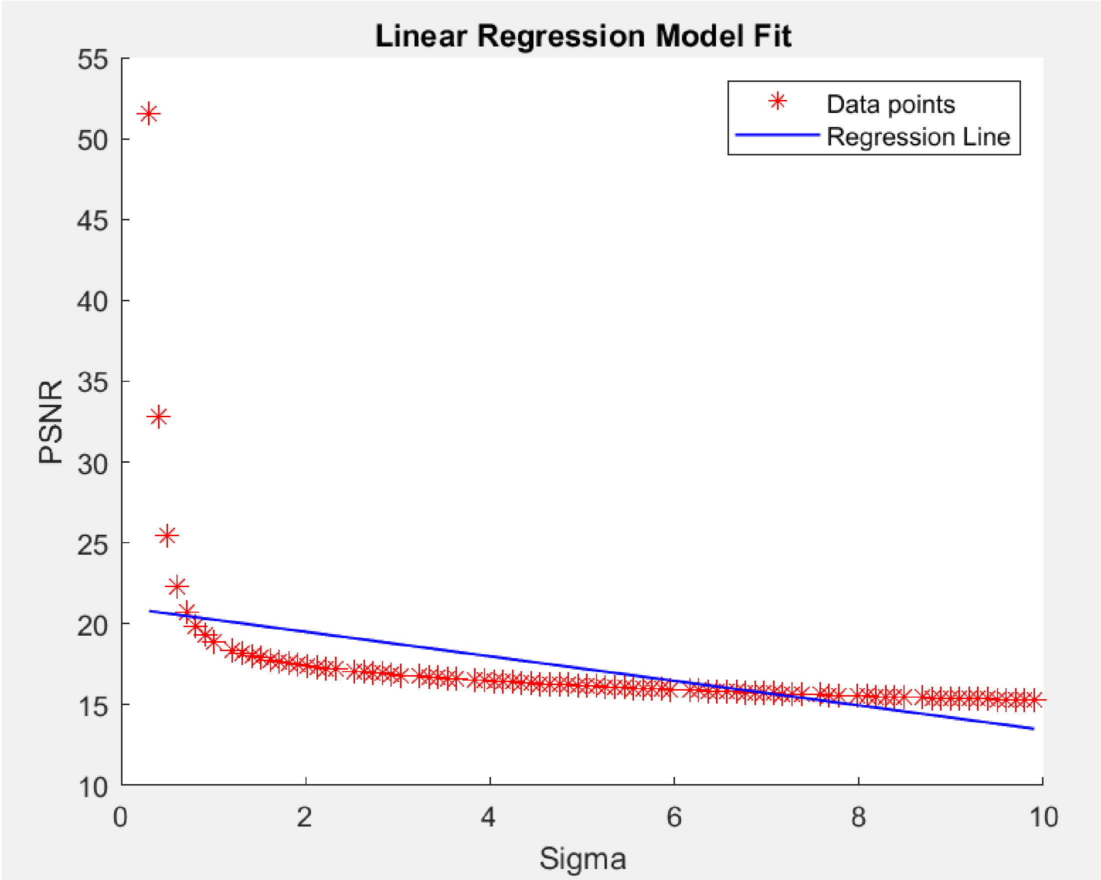

Noisy Image Vs. Gaussian Filtered Image (σ = 5)


Plot of Sigma vs PSNR
55
50
45
40
35
30
25
20
15
PSNR
0
2
4
6
8
10
Sigma

4. Consider the third question in Assignment-2. Formulate the model as Black box-model. Do not use any result from the White box model.

**Code: ma23m002_a2_q4.m**

```
%% Q4. Consider the third question in Assignment-2. Formulate the model as Black box model

clear;
clc;
% Load data of Q3 - Sigma & PSNR.
data_str = load('q3_data.mat','data');
data = data_str.data(4:end,:);
n = height(data);                        %% Size of data

% Split the data into training (90%) and testing (10%)
rng(123);  % Set different random seed

% Partition data for train and test
partition = cvpartition(n, 'HoldOut', 0.10);
trainData = data(training(partition), :);
testData = data(test(partition), :);
X_train =trainData(:,1);                        % Train data for fitting model
X_test = testData(:,1);                         % Test data for prediction
y_train = trainData(:,2);                       % Train data output for fit function
y_test = testData(:,2);                         % Test data output for error calculation
model = fitlm(X_train,y_train);                 % Fitting a linear model using fitlm function
testPredicted = predict(model, X_test);         % Predicting the test data's output using out fitted model

% Calculate model's performance on the test data
errorsq = sum((testPredicted - y_test).^2);                 % Sum of squared errors
RMSE = sqrt(errorsq / length(testPredicted));               % Root Mean Squared Error
disp(['Root Mean Squared Error (RMSE) on test data: ', num2str(RMSE)]);

% Plot the data points
figure(1);
scatter(X_train, y_train,50, 'r','*');  % Red points

hold on;  % Hold the plot for adding the regression line

% Plot the fitted regression line
yPrediction = predict(model, X_train);              % Predicted values using the model for regression line
plot(X_train, yPrediction, 'b', 'LineWidth', 1);    % Regression line

xlabel('Sigma');
ylabel('PSNR');
title('Linear Regression Model Fit');
legend({'Data points', 'Regression Line'}, 'Location', 'Northeast');
hold off;
```




**Output:**

**Root Mean Squared Error (RMSE) on test data: 1.4003**
**The maximum PSNR is when sigma is lowest.**

---

* Please open the excel file attached in the mail to see the data generated for each question. **MA23M002_A2_data**
** The zip folder attached with the mail contains the codes of all questions.