Assignment: 2

Mathematical Modelling in Industry

Name: **Lokendra Kumar**
Roll No: MA23M008
Submitted to: Dr. Sundar S
Date of Sub: 14/09/2023

1 Assignment

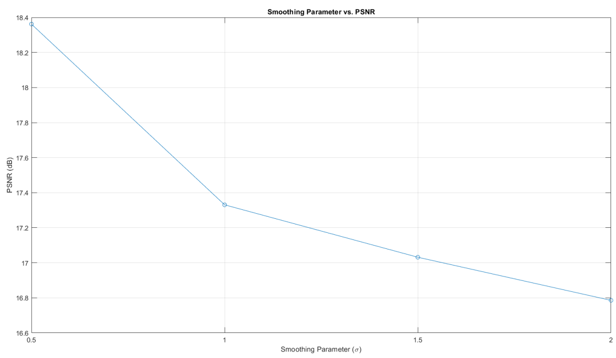
Q.3 Implement Linear Isotropic Diffusion using inbuilt Gaussian filter function.

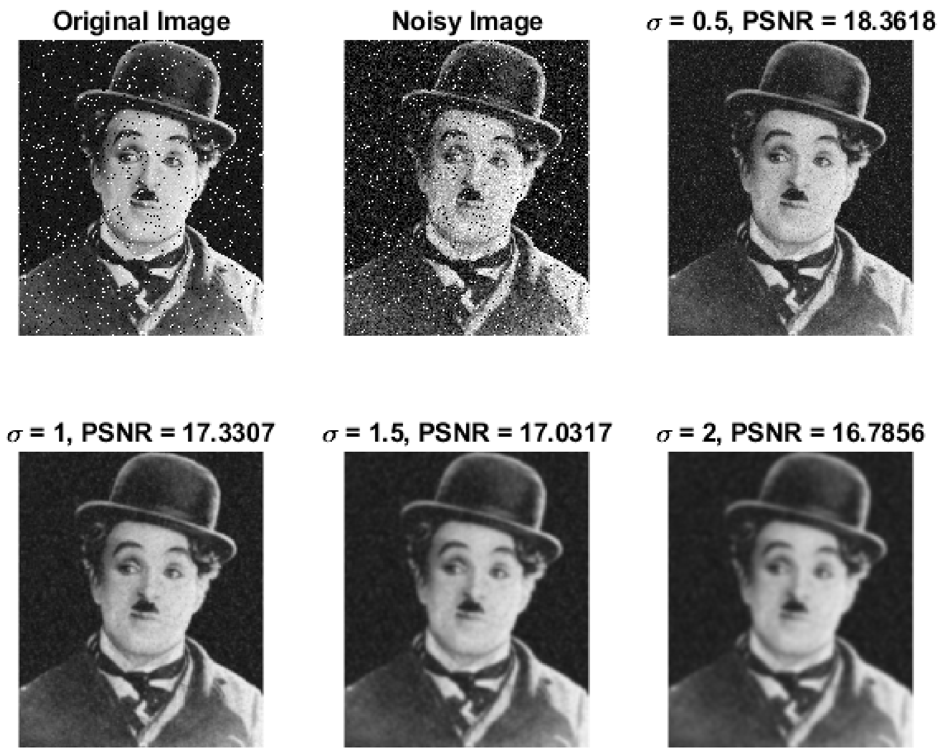
Sol. For Linear Isotropic Diffusion using inbuilt Gaussian filter function, MATLAB code is following.

```
% Load the image
image = imread('logo1.png');

% Add Gaussian noise to the image
sigma = 25;
noisy_image = imnoise(image, 'gaussian', 0, (sigma/255)^2);

% Display the original and noisy images
subplot(2, 3, 1);
imshow(image);
title('Original Image');

subplot(2, 3, 2);
imshow(noisy_image);
title('Noisy Image');

% Define different values of sigma (smoothing parameter)
sigmas = [0.5, 1, 1.5, 2];

% Number of diffusion iterations
num_iterations = 5;

% Initialize a table to store PSNR values
psnr_table = zeros(length(sigmas), 1);

% Clean the noisy image for different sigma values
for i = 1:length(sigmas)
sigma = sigmas(i);

% Apply Gaussian filter for smoothing
cleaned_image = noisy_image;  % Initialize cleaned_image with
    noisy_image
for j = 1:num_iterations
cleaned_image = imgaussfilt(cleaned_image, sigma);
end

% Calculate PSNR
mse = mean((double(image(:)) - double(cleaned_image(:))).^2);
max_pixel_value = double(max(image(:)));
psnr = 10 * log10((max_pixel_value^2) / mse);

% Store PSNR in the table
psnr_table(i) = psnr;

% Display the cleaned image
subplot(2, 3, i + 2);
imshow(uint8(cleaned_image));
title(['\sigma = ', num2str(sigma), ', PSNR = ', num2str(psnr)]);
end
```

```matlab

    % Create a table of Smoothing Parameter vs. PSNR
    table_data = table(sigmas', psnr_table, 'VariableNames',
       {'SmoothingParameter', 'PSNR'});
    disp('PSNR Table:');
    disp(table_data);

    % Plot Smoothing Parameter vs. PSNR
    figure;
    plot(sigmas, psnr_table, '-o');
    xlabel('Smoothing Parameter (\sigma)');
    ylabel('PSNR (dB)');
    title('Smoothing Parameter vs. PSNR');
    grid on;

    % Ensure subplots are properly displayed
    set(gcf, 'Position', get(0,'Screensize'));
```

Listing 1: Your MATLAB code caption here





Output:PSNR Table to understand the quality of cleaning.

```
                    SmoothingParameter      PSNR
                    ------------------     ------

                              0.5           18.362
                              1             17.331
                              1.5           17.032
                              2             16.786
```

Listing 2: PSNR Table

Q.2 A series of cups of equal capacity have been filled with water and arranged one below another. Pour into the first cup a quantity of wine equal to the capacity of the cup at a constant rate and let the overflow in each cup, go into the cup just below. Assuming that complete mixing of wine and water takes place instantaneously. Find the amount of wine in each cup at any time t and at the end of the process at time T. For this question formulate the model as Black box model.

Sol. First we fix $T = 10$ and generate a the values randomly considering a normal distribution and generate 50 iid samples of q, and t (where $0 < t <= T, q > 0$) from it. Let x_n be the amount of wine in the nth cup where

$$x_n = q\left(1 - \frac{\left(1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots + \frac{1}{(n-1)!}\right)\frac{t}{T}}{e^{\frac{t}{T}}}\right).$$

Now using regression(machine learning method) we are going to solve it.

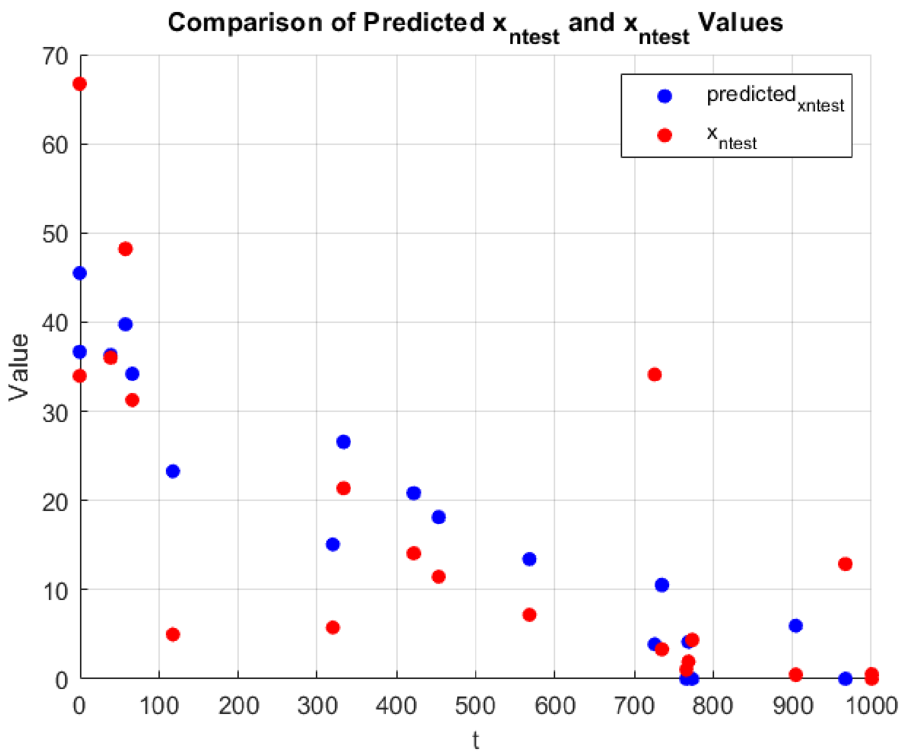
```
T = 100; % Maximum value of t
num_samples = 100; % Number of samples

% Generate random values for t and q
mu_t = T / 2; % Mean of t
sigma_t = T / 3; % Standard deviation of t

mu_q = 50; % Mean of q
sigma_q = 20; % Standard deviation of q

% Generate random samples
t_samples = max(0, min(T, t_samples));% To get only +ve values
q_samples = normrnd(mu_q, sigma_q, 1, num_samples);

% Ensure that generated values are within the specified range
t_samples = max(0, min(T, t_samples));
q_samples = max(0, q_samples);

% Display the generated samples
disp("Generated t samples:");
disp(t_samples);

disp("Generated q samples:");
disp(q_samples);
% Split the generated dataset into train and test datasets
train_ratio = 0.8; % 80% of data for training, 20% for testing
num_train_samples = round(num_samples * train_ratio);
```

```
num_test_samples = num_samples - num_train_samples;

% Split the t_samples and q_samples into train and test sets
t_train = t_samples(1:num_train_samples);
q_train = q_samples(1:num_train_samples);

t_test = t_samples(num_train_samples+1:end);
q_test = q_samples(num_train_samples+1:end);

% Initialize an array to store the predicted values x_n
x_n_train = zeros(size(t_train));

% Calculate the predicted variable x_n for the train dataset
for i = 1:num_train_samples
x_n_train(i) = q_train(i) * (1 - sum(1 ./ factorial(0:i-1)) * (t_train(i)
   / T) / exp(t_train(i) / T));
end

% Display the predicted x_n for the train dataset
disp("Predicted x_n for the train dataset:");
disp(x_n_train);


% Define a custom equation for curve fitting
eqn = @(a, t) a(1) * (1 - sum(1 ./ factorial(0:length(a)-2)) * (t / T) ./
   exp(t / T));

% Initial guess for fitting parameters
a0 = [1];

% Fit the curve to the training data using lsqcurvefit
fit_params = lsqcurvefit(eqn, a0, t_train, x_n_train);

% Calculate the fitted values for the training dataset
fitted_values_train = eqn(fit_params, t_train);

% Display the fitted parameters
disp("Fitted parameters:");
disp(fit_params);

% Use a linear regression model to predict on the test dataset (same as
   before)
X_train = [t_train', q_train']; % Predictors
X_test = [t_test', q_test']; % Test predictors

% Add a constant term to the predictors for regression
X_train = [ones(num_train_samples, 1), X_train];
X_test = [ones(num_test_samples, 1), X_test];

% Fit a linear regression model to the training data
regression_model = fitlm(X_train, x_n_train);

% Make predictions on the test dataset
predicted_x_n_test = max(0,predict(regression_model, X_test));
```

```
% Display the predicted values on the test dataset
disp("Predicted x_n for the test dataset:");
disp(predicted_x_n_test);
% Initialize an array to store the predicted values x_n for the test dataset
x_n_test_formula = zeros(size(t_test));

% Calculate the predicted variable x_n for the test dataset using the formula
for i = 1:num_test_samples
sum_term = 0;
for j = 0:(i-1)
sum_term = sum_term + 1 / factorial(j);
end
x_n_test_formula(i) = q_test(i) * (1 - sum_term * (t_test(i) / T) / exp(t_test(i) / T));
end
% Determine the minimum number of rows among the variables
rows = 20;

data = cell(rows, 5);  % 5 columns: t, T, q, x_n_test, predicted_x_n_test

% Fill in the cell array with the available data
for i = 1:rows
data{i, 1} = t_test(i);
data{i, 2} = T;
data{i, 3} = q_test(i);
data{i, 4} = x_n_test_formula(i);
data{i, 5} = predicted_x_n_test(i);
end

% Create the table using cell2table
test_data_table = cell2table(data, 'VariableNames', {'t', 'T', 'q', 'x_n_test', 'predicted_x_n_test'});

% Display the table
disp(test_data_table);

% Create a figure
figure;

% Scatter plot for predicted_x_n_test values in blue
scatter(t_test, predicted_x_n_test, 'b', 'filled');

hold on;

% Scatter plot for x_n_test values in red
scatter(t_test, x_n_test_formula, 'r', 'filled');

% Add labels and a legend
xlabel('t');
ylabel('Value');
title('Comparison of Predicted x_n_{test} and x_n_{test} Values');
```

```
legend('predicted_x_n_{test}', 'x_n_{test}');

% Display the grid
grid on;

% Show the plot
hold off;
```

Listing 3: Your MATLAB code caption here



Output table:

t	q	x_n_test	predicted_x_n_test
90.153	13.462	8.5353	0
3.1961	60.718	56.959	39.206
56.852	63.857	12.454	14.233
0	36.249	36.249	34.631
50.947	56.352	9.6355	15.194
100	53.257	0.031644	0
81.371	73.182	1.4451	4.7911
49.769	52.568	9.3338	14.814
38.725	70.789	20.198	24.665
67.609	47.665	3.1121	5.0268
85.23	37.034	0.44602	0
50.491	49.24	8.4506	13.637
100	46.014	2.9264e−09	0

22.805	67.633	34.257	31.519
0	48.877	48.877	37.784
25.06	33.175	15.586	21.834
43.357	46.903	11.072	16.478
37.575	73.772	22.024	25.962
21.754	41.702	21.863	25.55
94.575	36.828	0.056202	0

Q.1 How many cherries each of radius r can be packed in a can of radius R and height h? Obtain upper and lower bounds. For this question formulate the model as Black box model.

Sol. first we are generating the 100 iid sample values of r, R and h randomly, considering a normal distribution. Now using regression(machine learning method) we are going to solve it.

```

% Number of samples
n = 100;

% Generate random values for r, R, and h following a normal distribution
r = abs(normrnd(5, 1, [1, n])); % r > 0
R = abs(normrnd(10, 2, [1, n])); % r < R
h = max(2*r, abs(normrnd(15, 3, [1, n]))); % h >= 2*r

% Create a dataset matrix
data = [r; R; h]';

% Split the dataset into train and test sets (e.g., 80% train, 20% test)
train_ratio = 0.8;
n_train = floor(n * train_ratio);
n_test = n - n_train;

% Shuffle the dataset
data = data(randperm(n), :);

% Split into train and test sets
train_data = data(1:n_train, :);
test_data = data(n_train+1:end, :);

% Calculate min and max cherries packed using the formula for the train dataset
minCherriesPacked_train = floor(train_data(:, 2).^2 .* train_data(:, 3) ./ (2 .* train_data(:, 1).^3));
maxCherriesPacked_train = floor(0.74 * 3 * train_data(:, 2).^2 .* train_data(:, 3) ./ (4 .* train_data(:, 1).^3));

% Display the min and max cherries packed for the train dataset
fprintf('Min Cherries Packed (Train): %s\n', mat2str(minCherriesPacked_train));
fprintf('Max Cherries Packed (Train): %s\n', mat2str(maxCherriesPacked_train));

% Fit a polynomial curve to the train data for minCherriesPacked
x = train_data(:, 1); % r values
y = minCherriesPacked_train;
```

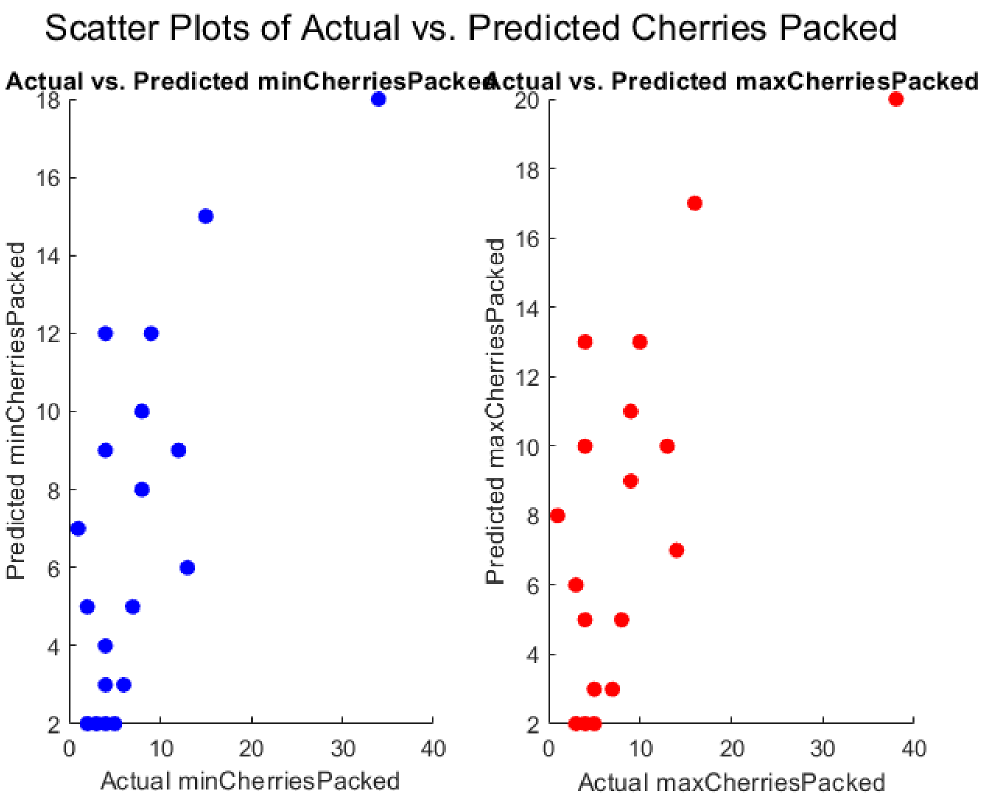
```
degree = 2; % Choose the degree of the polynomial curve

% Fit the polynomial curve
p_min = polyfit(x, y, degree);

% Predict minCherriesPacked on the test dataset
x_test = test_data(:, 1); % r values
minCherriesPacked_pred = floor(polyval(p_min, x_test));

% Fit a polynomial curve to the train data for maxCherriesPacked
y = maxCherriesPacked_train;

% Fit the polynomial curve
p_max = polyfit(x, y, degree);

% Predict maxCherriesPacked on the test dataset
maxCherriesPacked_pred = floor(polyval(p_max, x_test));

% Display predictions
fprintf('Predicted minCherriesPacked on Test Data: %s\n',
    mat2str(minCherriesPacked_pred));
fprintf('Predicted maxCherriesPacked on Test Data: %s\n',
    mat2str(maxCherriesPacked_pred));

% Calculate min and max cherries packed for the test dataset using the
    formulas
minCherriesPacked_test = floor(test_data(:, 2).^2 .* test_data(:, 3) ./
    (2 .* test_data(:, 1).^3));
maxCherriesPacked_test = floor(0.74 * 3 * test_data(:, 2).^2 .*
    test_data(:, 3) ./ (4 .* test_data(:, 1).^3));

% Predict minCherriesPacked and maxCherriesPacked using the fitted curves
minCherriesPacked_pred = floor(polyval(p_min, test_data(:, 1))); %
    Predict minCherriesPacked
maxCherriesPacked_pred = floor(polyval(p_max, test_data(:, 1))); %
    Predict maxCherriesPacked

% Create a table of values with the additional prediction columns
dataTable = table(test_data(:, 1), test_data(:, 2), test_data(:, 3), ...
minCherriesPacked_test, maxCherriesPacked_test, ...
minCherriesPacked_pred, maxCherriesPacked_pred, ...
'VariableNames', {'r', 'R', 'h', 'minCherriesPacked',
    'maxCherriesPacked', 'minCherriesPacked_pred',
    'maxCherriesPacked_pred'});

% Display the table
disp(dataTable);
```

Listing 4: Your MATLAB code caption here

Output Table:

r	R	h	minCherriesPacked	maxCherriesPacked	minCherriesPacked_pred	maxCherriesPacked_pred
5.9105	11.366	16.511	5	5	2	2

5.5894	9.4599	13.866	3	3	3	3
4.71	15.575	17.76	20	22	7	7
5.6125	9.4046	20.215	5	5	3	3
5.5712	10.311	22.526	6	7	3	3
5.9421	12.832	17.331	6	7	2	2
5.2296	7.6889	12.142	2	2	4	4
5.5954	6.7227	13.33	1	1	3	3
6.2616	7.6667	15.449	1	2	2	2
5.2495	8.9555	16.129	4	4	4	4
4.8682	10.288	16.665	7	8	6	6
5.5265	10.492	15.194	4	5	3	3
3.559	7.9356	14.887	10	11	17	19
4.9451	12.287	13.709	8	9	5	6
4.4592	11.502	15.639	11	12	8	9
4.507	7.4335	17.772	5	5	8	9
5.44	9.981	15.952	4	5	3	3
5.2696	9.8402	16.428	5	6	4	4
3.9484	12.059	9.8237	11	12	13	14
3.9797	10.582	14.856	13	14	13	14



Q.4 Formulate the Linear Isotropic Diffusion using inbuilt Gaussian filter function as Black box model.

Sol. First we are Generating random data for linearly independent variables d (diffusivity), t (diffusion time), and sigma (Gaussian standard deviation). Then we are creating a black-box model using linear regression.

```
% Generate synthetic data for linearly independent variables
% Variables: d (diffusivity), t (diffusion time), and sigma
    (Gaussian standard deviation)
n_samples = 1000; % Number of data samples

% Generate random values for d, t, and sigma within specified
    ranges
d_min = 0.1;
```

```
    d_max = 10;
    t_min = 0.1;
    t_max = 100;
    sigma_min = 0.1;
    sigma_max = 10;

    d = d_min + (d_max - d_min) * rand(n_samples, 1);
    t = t_min + (t_max - t_min) * rand(n_samples, 1);
    sigma = sigma_min + (sigma_max - sigma_min) * rand(n_samples, 1);

    % Calculate the corresponding output (result of Gaussian smoothing)
    output = sqrt(2 * t) .* sigma;

    % Create a black-box model using linear regression
    X = [d, t]; % Independent variables
    Y = output; % Dependent variable

    % Fit a linear regression model
    mdl = fitlm(X, Y);

    % Display the model summary
    disp(mdl);

    % Predict the output for new data
    % Example: Predict the output for d = 5 and t = 50
    new_d = 5;
    new_t = 50;
    predicted_output = predict(mdl, [new_d, new_t]);
    disp(['Predicted Output for d = ', num2str(new_d), ' and t = ', num2str(new_t), ' is  ', num2str(predicted_output)]);


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Load the image
    image = imread('logo1.png');

    % Generate synthetic data for d (diffusivity) and t (diffusion time)
    % Example values:
    d = 5;   % Adjust as needed
    t = 50;  % Adjust as needed

    % Apply Gaussian smoothing to the image using specified d and t
    sigma = sqrt(2 * t);
    smoothed_image = imgaussfilt(image, sigma, 'FilterSize', 5); % You can adjust the filter size as needed

    % Display the original and smoothed images
    figure;
    subplot(1, 2, 1);
    imshow(image);
    title('Original Image');

```

```
        subplot(1, 2, 2);
        imshow(smoothed_image);
        title(['Smoothed Image (t=', num2str(t), ', d=', num2str(d),
           ')']);
```

Listing 5: Your MATLAB code caption here