

# EXERCISE CLASS 6 - SPC for iid data

## EXERCISE 1

Data reported in 'ESE06\_ex1.csv' represent the diameters of spheres in a recirculating ball mechanism widely used in machine tool industry. The component is very critical and the tolerance is  $10 \pm 0.5\text{mm}$ .

Verify if the process is in control by using an X-bar and R chart.

```
In [ ]: # Import the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import stats

# Import the dataset
data = pd.read_csv('ESE06_ex1.csv')

# Inspect the dataset
data.head()
```

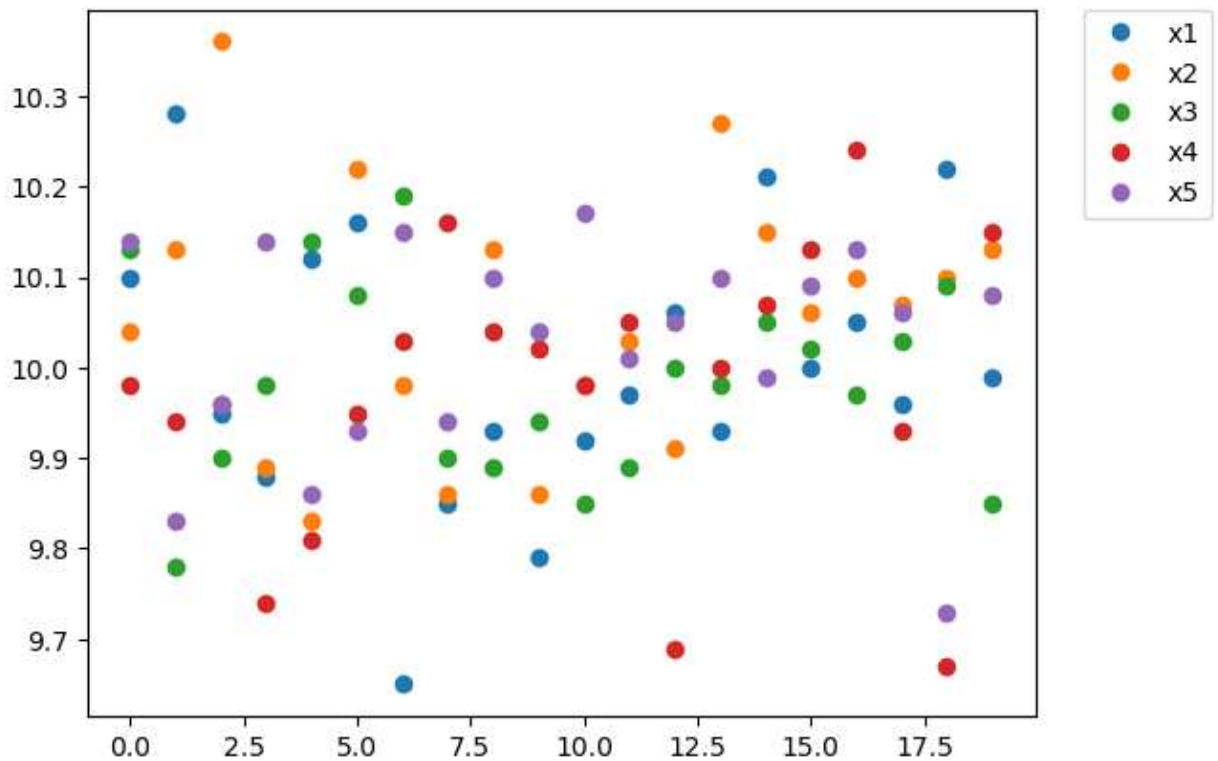
```
Out[ ]:
```

	x1	x2	x3	x4	x5
0	10.10	10.04	10.13	9.98	10.14
1	10.28	10.13	9.78	9.94	9.83
2	9.95	10.36	9.90	9.96	9.96
3	9.88	9.89	9.98	9.74	10.14
4	10.12	9.83	10.14	9.81	9.86

### Solution

Inspect the data by plotting the individual datapoints.

```
In [ ]: # Make a scatter plot of all the columns against the index
plt.plot(data['x1'], linestyle='none', marker='o', label = 'x1')
plt.plot(data['x2'], linestyle='none', marker='o', label = 'x2')
plt.plot(data['x3'], linestyle='none', marker='o', label = 'x3')
plt.plot(data['x4'], linestyle='none', marker='o', label = 'x4')
plt.plot(data['x5'], linestyle='none', marker='o', label = 'x5')
# place the legend outside the plot
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



Doesn't look like strange patterns or outliers are present.

We might also check randomness but we would need to know the within-sample order!

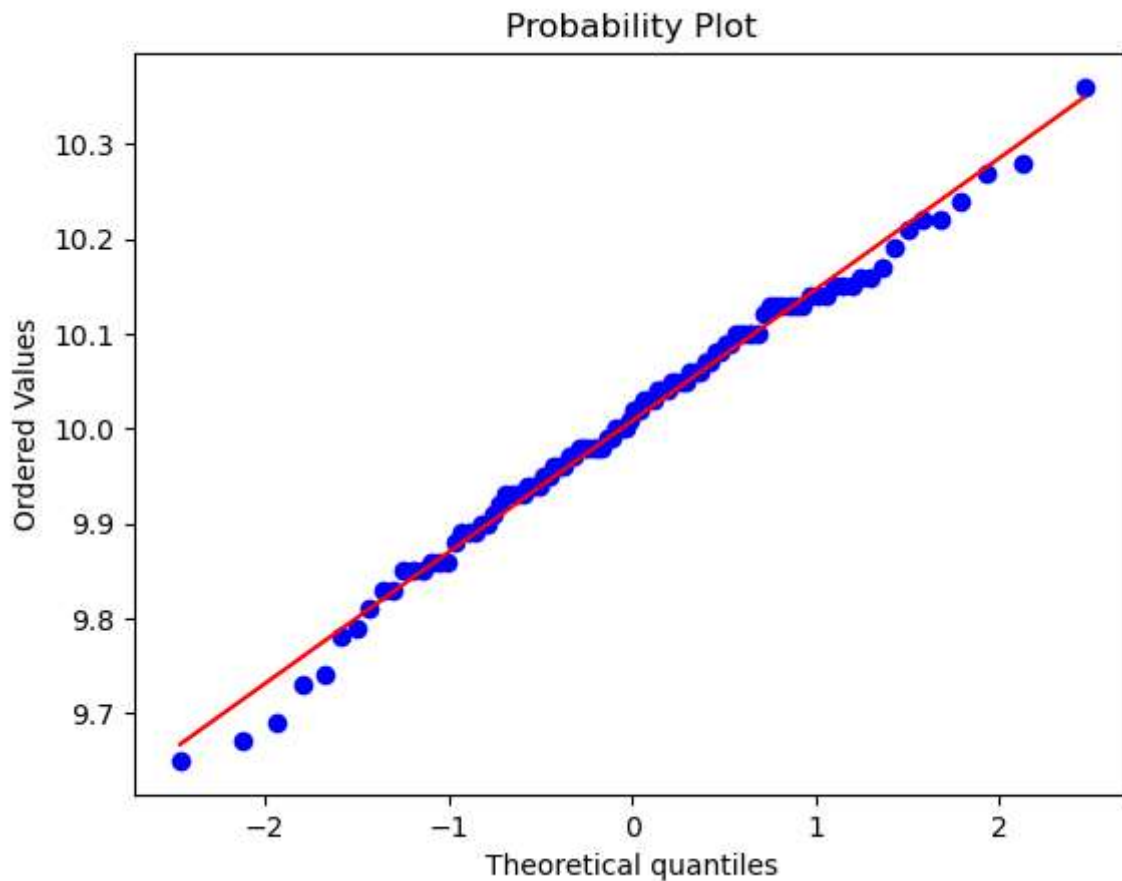
Verify the assumption of normality, assuming all the data are from the same population.

```
In [ ]: # Stack the data into a single column
data_stack = data.stack()

# We can use the Shapiro-Wilk test
_, p_value_SW = stats.shapiro(data_stack)
print('p-value of the Shapiro-Wilk test: %.3f' % p_value_SW)

# QQ-plot
stats.probplot(data_stack, dist="norm", plot=plt)
plt.show()
```

p-value of the Shapiro-Wilk test: 0.753



Let's compute the mean and the range for each sample.

*Note: we need to apply the mean and range functions to each row of the data frame.*

```
In [ ]: # Make a copy of the data
data_XR = data.copy()
# Add a column with the mean of the rows
data_XR['sample_mean'] = data.mean(axis=1)
# Add a column with the range of the rows
data_XR['sample_range'] = data.max(axis=1) - data.min(axis=1)

# Inspect the dataset
data_XR.head()
```

```
Out[ ]:
```

	x1	x2	x3	x4	x5	sample_mean	sample_range
0	10.10	10.04	10.13	9.98	10.14	10.078	0.16
1	10.28	10.13	9.78	9.94	9.83	9.992	0.50
2	9.95	10.36	9.90	9.96	9.96	10.026	0.46
3	9.88	9.89	9.98	9.74	10.14	9.926	0.40
4	10.12	9.83	10.14	9.81	9.86	9.952	0.33

Now compute the grand mean and the mean of the ranges.

```
In [ ]: Xbar_mean = data_XR['sample_mean'].mean()
R_mean = data_XR['sample_range'].mean()

print('Mean of the sample mean: %.3f' % Xbar_mean)
print('Mean of the sample range: %.3f' % R_mean)
```

Mean of the sample mean: 10.008  
Mean of the sample range: 0.314

Since there is no constraint on the choice of Type I error  $\alpha$ , we can set  $K = 3$  ( $\alpha = 0.0027$ )

Remember the formulas for the control limits.

**$\bar{X}$  chart:**

- $UCL = \bar{\bar{X}} + A_2(n)\bar{R}$
- $CL = \bar{\bar{X}}$
- $LCL = \bar{\bar{X}} - A_2(n)\bar{R}$

**$R$  chart:**

- $UCL = D_4(n)\bar{R}$
- $CL = \bar{R}$
- $LCL = D_3(n)\bar{R}$

## Factors for constructing variable control charts

Observations in Sample, n	Chart for Averages			Chart for Standard Deviations								Chart for Ranges							
	Factors for Control Limits			Factors for Center Line		Factors for Control Limits				Factors for Center Line		Factors for Control Limits							
	A	A <sub>2</sub>	A <sub>3</sub>	c <sub>4</sub>	1/c <sub>4</sub>	B <sub>3</sub>	B <sub>4</sub>	B <sub>5</sub>	B <sub>6</sub>	d <sub>2</sub>	1/d <sub>2</sub>	d <sub>3</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>			
2	2.121	1.880	2.659	0.7979	1.2533	0	3.267	0	2.606	1.128	0.8865	0.853	0	3.686	0	3.267			
3	1.732	1.023	1.954	0.8862	1.1284	0	2.568	0	2.276	1.693	0.5907	0.888	0	4.358	0	2.574			
4	1.500	0.729	1.628	0.9213	1.0854	0	2.266	0	2.088	2.059	0.4857	0.880	0	4.698	0	2.282			
5	1.342	0.577	1.427	0.9400	1.0638	0	2.089	0	1.964	2.326	0.4299	0.864	0	4.918	0	2.114			
6	1.225	0.483	1.287	0.9515	1.0510	0.030	1.970	0.029	1.874	2.534	0.3946	0.848	0	5.078	0	2.004			
7	1.134	0.419	1.182	0.9594	1.0423	0.118	1.882	0.113	1.806	2.704	0.3698	0.833	0.204	5.204	0.076	1.924			
8	1.061	0.373	1.099	0.9650	1.0363	0.185	1.815	0.179	1.751	2.847	0.3512	0.820	0.388	5.306	0.136	1.864			
9	1.000	0.337	1.032	0.9693	1.0317	0.239	1.761	0.232	1.707	2.970	0.3367	0.808	0.547	5.393	0.184	1.816			
10	0.949	0.308	0.975	0.9727	1.0281	0.284	1.716	0.276	1.669	3.078	0.3249	0.797	0.687	5.469	0.223	1.777			
11	0.905	0.285	0.927	0.9754	1.0252	0.321	1.679	0.313	1.637	3.173	0.3152	0.787	0.811	5.535	0.256	1.744			
12	0.866	0.266	0.886	0.9776	1.0229	0.354	1.646	0.346	1.610	3.258	0.3069	0.778	0.922	5.594	0.283	1.717			
13	0.832	0.249	0.850	0.9794	1.0210	0.382	1.618	0.374	1.585	3.336	0.2998	0.770	1.025	5.647	0.307	1.693			
14	0.802	0.235	0.817	0.9810	1.0194	0.406	1.594	0.399	1.563	3.407	0.2935	0.763	1.118	5.696	0.328	1.672			
15	0.775	0.223	0.789	0.9823	1.0180	0.428	1.572	0.421	1.544	3.472	0.2880	0.756	1.203	5.741	0.347	1.653			
16	0.750	0.212	0.763	0.9835	1.0168	0.448	1.552	0.440	1.526	3.532	0.2831	0.750	1.282	5.782	0.363	1.637			
17	0.728	0.203	0.739	0.9845	1.0157	0.466	1.534	0.458	1.511	3.588	0.2787	0.744	1.356	5.820	0.378	1.622			
18	0.707	0.194	0.718	0.9854	1.0148	0.482	1.518	0.475	1.496	3.640	0.2747	0.739	1.424	5.856	0.391	1.608			
19	0.688	0.187	0.698	0.9862	1.0140	0.497	1.503	0.490	1.483	3.689	0.2711	0.734	1.487	5.891	0.403	1.597			
20	0.671	0.180	0.680	0.9869	1.0133	0.510	1.490	0.504	1.470	3.735	0.2677	0.729	1.549	5.921	0.415	1.585			
21	0.655	0.173	0.663	0.9876	1.0126	0.523	1.477	0.516	1.459	3.778	0.2647	0.724	1.605	5.951	0.425	1.575			
22	0.640	0.167	0.647	0.9882	1.0119	0.534	1.466	0.528	1.448	3.819	0.2618	0.720	1.659	5.979	0.434	1.566			
23	0.626	0.162	0.633	0.9887	1.0114	0.545	1.455	0.539	1.438	3.858	0.2592	0.716	1.710	6.006	0.443	1.557			
24	0.612	0.157	0.619	0.9892	1.0109	0.555	1.445	0.549	1.429	3.895	0.2567	0.712	1.759	6.031	0.451	1.548			
25	0.600	0.153	0.606	0.9896	1.0105	0.565	1.435	0.559	1.420	3.931	0.2544	0.708	1.806	6.056	0.459	1.541			

For  $n > 25$ .

```
In [ ]: n = 5
A2 = 0.577
```

```

D3 = 0
D4 = 2.114

# Now we can compute the CL, UCL and LCL for Xbar and R
data_XR['Xbar_CL'] = Xbar_mean
data_XR['Xbar_UCL'] = Xbar_mean + A2 * R_mean
data_XR['Xbar_LCL'] = Xbar_mean - A2 * R_mean

data_XR['R_CL'] = R_mean
data_XR['R_UCL'] = D4 * R_mean
data_XR['R_LCL'] = D3 * R_mean

# Inspect the dataset
data_XR.head()

```

```

Out[ ]:

```

	x1	x2	x3	x4	x5	sample_mean	sample_range	Xbar_CL	Xbar_UCL	Xbar_LCL	R_CL
0	10.10	10.04	10.13	9.98	10.14	10.078	0.16	10.0082	10.189378	9.827022	0.314
1	10.28	10.13	9.78	9.94	9.83	9.992	0.50	10.0082	10.189378	9.827022	0.314
2	9.95	10.36	9.90	9.96	9.96	10.026	0.46	10.0082	10.189378	9.827022	0.314
3	9.88	9.89	9.98	9.74	10.14	9.926	0.40	10.0082	10.189378	9.827022	0.314
4	10.12	9.83	10.14	9.81	9.86	9.952	0.33	10.0082	10.189378	9.827022	0.314

Add two columns to store the violations of the control limits.

```

In [ ]: data_XR['Xbar_TEST1'] = np.where((data_XR['sample_mean'] > data_XR['Xbar_UCL']) |
                                         (data_XR['sample_mean'] < data_XR['Xbar_LCL']), data_XR['sample_mean'],
                                         data_XR['sample_mean'])
data_XR['R_TEST1'] = np.where((data_XR['sample_range'] > data_XR['R_UCL']) |
                               (data_XR['sample_range'] < data_XR['R_LCL']), data_XR['sample_range'],
                               data_XR['sample_range'])

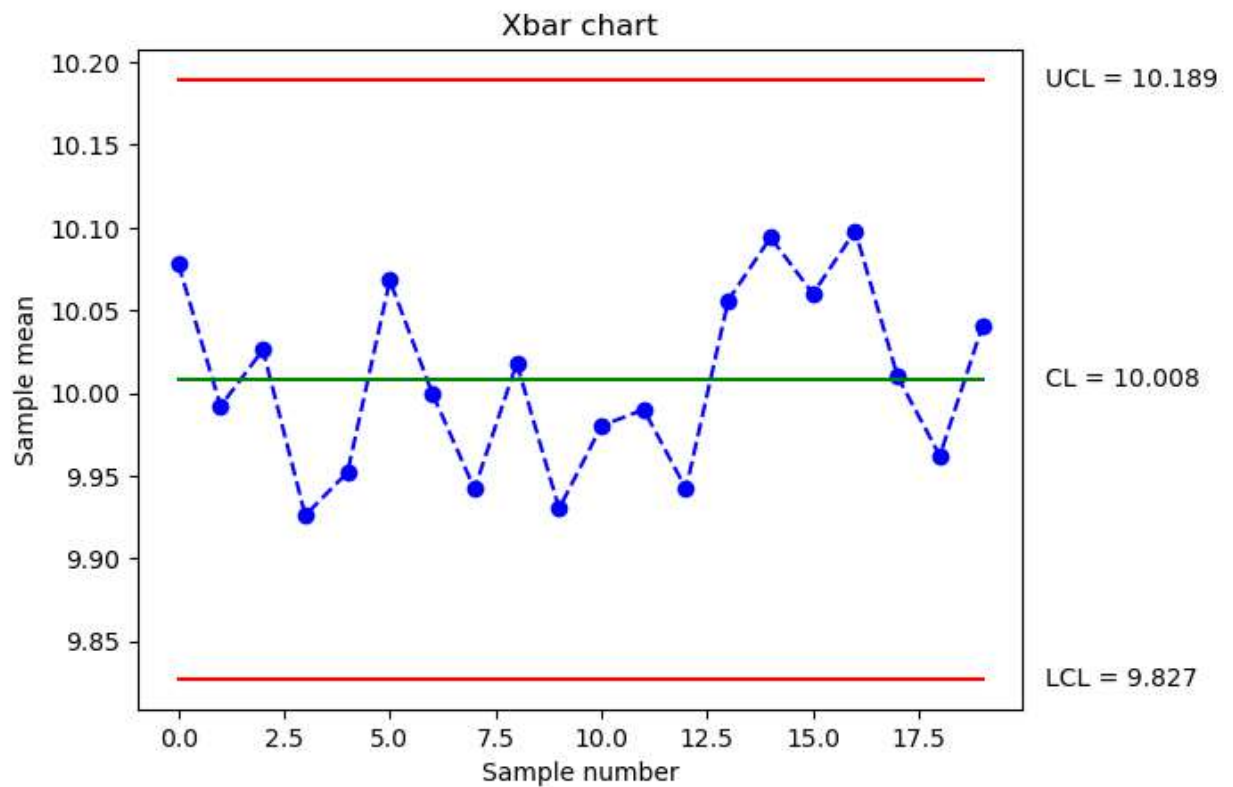
```

Now plot the limits and the data in the charts.

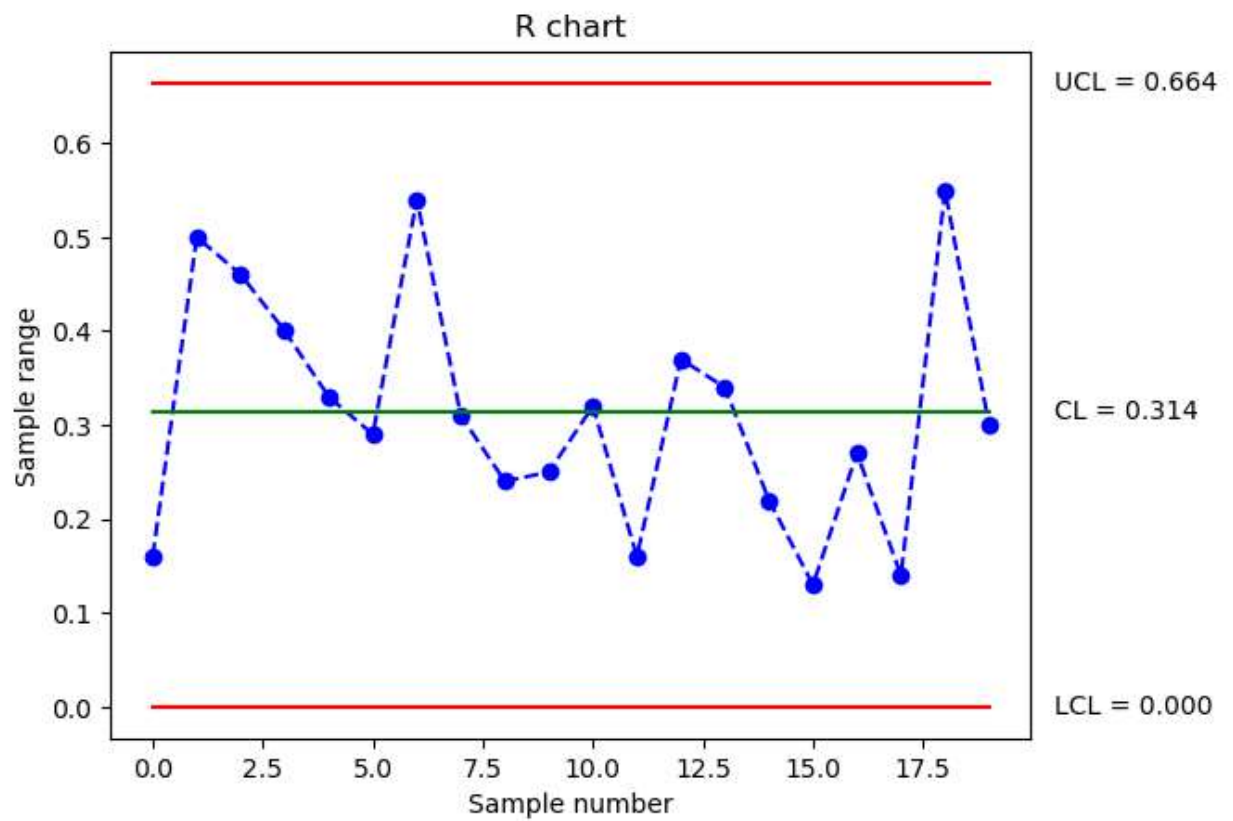
```

In [ ]: # Plot the Xbar chart
plt.title('Xbar chart')
plt.plot(data_XR['sample_mean'], color='b', linestyle='--', marker='o')
plt.plot(data_XR['Xbar_UCL'], color='r')
plt.plot(data_XR['Xbar_CL'], color='g')
plt.plot(data_XR['Xbar_LCL'], color='r')
plt.ylabel('Sample mean')
plt.xlabel('Sample number')
# add the values of the control limits on the right side of the plot
plt.text(len(data_XR)+.5, data_XR['Xbar_UCL'].iloc[0], 'UCL = {:.3f}'.format(data_XR['Xbar_UCL'].iloc[0]))
plt.text(len(data_XR)+.5, data_XR['Xbar_CL'].iloc[0], 'CL = {:.3f}'.format(data_XR['Xbar_CL'].iloc[0]))
plt.text(len(data_XR)+.5, data_XR['Xbar_LCL'].iloc[0], 'LCL = {:.3f}'.format(data_XR['Xbar_LCL'].iloc[0]))
# highlight the points that violate the alarm rules
plt.plot(data_XR['Xbar_TEST1'], linestyle='none', marker='s', color='r', markersize=10)
plt.show()

```



```
In [ ]: # Plot the R chart
plt.title('R chart')
plt.plot(data_XR['sample_range'], color='b', linestyle='--', marker='o')
plt.plot(data_XR['R_UCL'], color='r')
plt.plot(data_XR['R_CL'], color='g')
plt.plot(data_XR['R_LCL'], color='r')
plt.ylabel('Sample range')
plt.xlabel('Sample number')
# add the values of the control limits on the right side of the plot
plt.text(len(data_XR)+.5, data_XR['R_UCL'].iloc[0], 'UCL = {:.3f}'.format(data_XR['R_UCL'].iloc[0]))
plt.text(len(data_XR)+.5, data_XR['R_CL'].iloc[0], 'CL = {:.3f}'.format(data_XR['R_CL'].iloc[0]))
plt.text(len(data_XR)+.5, data_XR['R_LCL'].iloc[0], 'LCL = {:.3f}'.format(data_XR['R_LCL'].iloc[0]))
# highlight the points that violate the alarm rules
plt.plot(data_XR['R_TEST1'], linestyle='none', marker='s', color='r', markersize=10)
plt.show()
```



The process is in control.

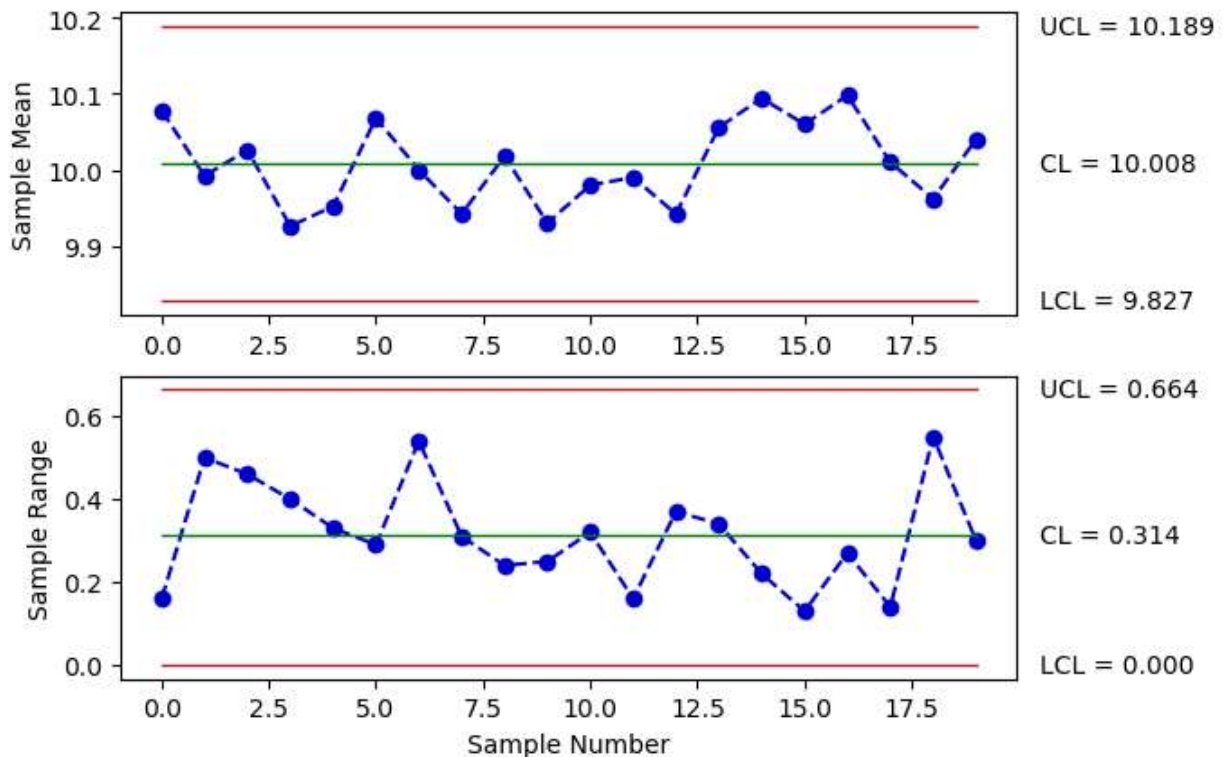
Alternatively, we can use the `XbarR` function from the `qcc.ControlCharts` package.

```
In [ ]: import qcc

# Create the control chart with the data
data_XR_qcc = qcc.ControlCharts.XbarR(data)
```



Xbar-R charts



## EXERCISE 1 (continued)

Given the previous dataset:

1. Redesign the X-bar and R chart in order to achieve in both the charts a Type I error equal to 0.002 (assuming that the normal approximation applies for both of them).
2. Determine the operating characteristic curve (OC) for the X-bar chart (by using  $K=3$  and expressing the shift of the mean in standard deviation units)
3. Determine the corresponding ARL curve.
4. Estimate the standard deviation through the statistic  $R$ .
5. Design the confidence interval on the process mean that corresponds to the control limits computed in point 1.

### Point 1

Redesign the X-bar and R chart in order to achieve in both the charts a Type I error equal to 0.002 (assuming that the normal approximation applies for both of them).

#### Solution

Assuming that the normal approximation applies for both of them, we need to find the value of  $K$  such that  $\alpha = 0.002$ :



$$K = z_{\alpha/2}$$

```
In [ ]: # Compute the new K_alpha value
alpha = 0.002
K_alpha = stats.norm.ppf(1-alpha/2)

print('K = %.3f' % K_alpha)

K = 3.090
```

Now let's design the control charts with the new value of K.

Remember the formulas for the control limits for  $K \neq 3$ .

$\bar{X}$  chart:

- $UCL = \bar{\bar{X}} + z_{\alpha/2} \frac{1}{d_2\sqrt{n}} \bar{R}$
- $CL = \bar{\bar{X}}$
- $LCL = \bar{\bar{X}} - z_{\alpha/2} \frac{1}{d_2\sqrt{n}} \bar{R}$

R chart:

- $UCL = \bar{R} + z_{\alpha/2} \frac{d_3}{d_2} \bar{R}$
- $CL = \bar{R}$
- $LCL = \max(0; \bar{R} - z_{\alpha/2} \frac{d_3}{d_2} \bar{R})$

## Factors for constructing variable control charts

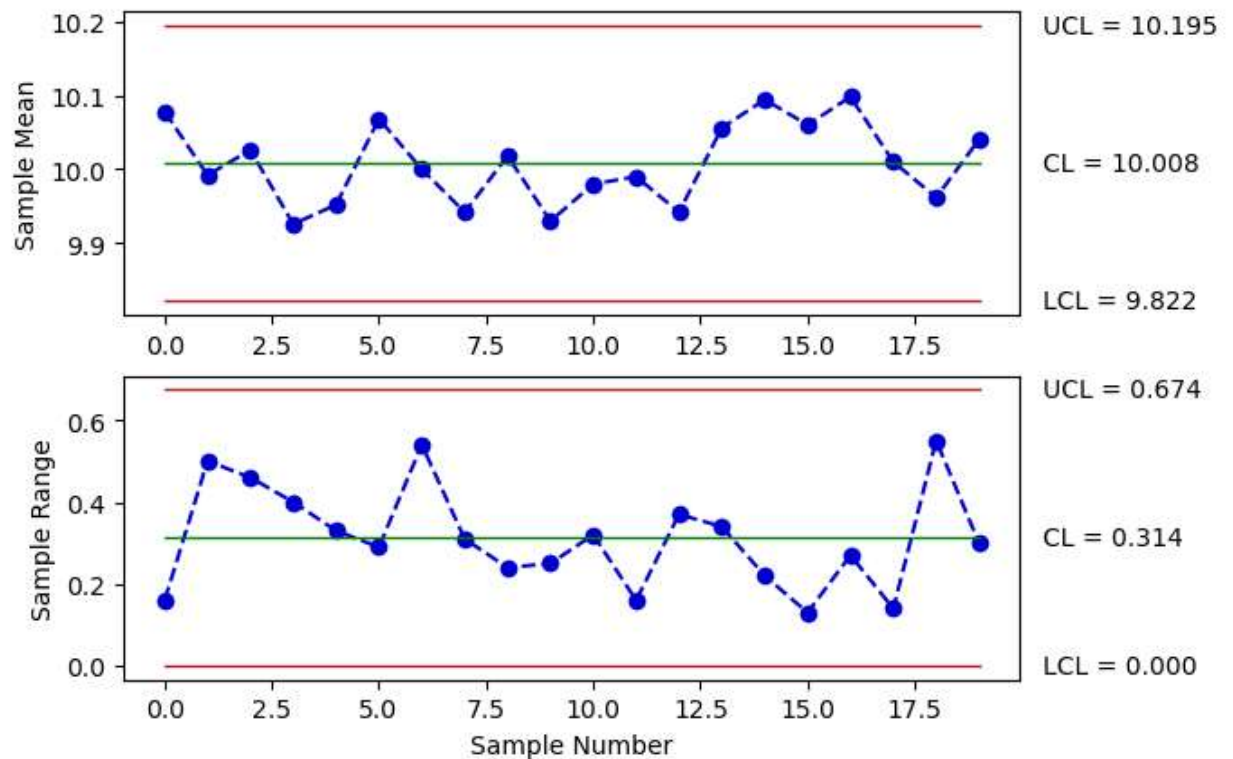
Observations in Sample, n	Chart for Averages					Chart for Standard Deviations					Chart for Ranges							
	Factors for Control Limits					Factors for Center Line					Factors for Control Limits							
	A	A <sub>2</sub>	A <sub>3</sub>	c <sub>4</sub>	1/c <sub>4</sub>	B <sub>3</sub>	B <sub>4</sub>	B <sub>5</sub>	B <sub>6</sub>	d <sub>2</sub>	1/d <sub>2</sub>	d <sub>3</sub>	D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>		
2	2.121	1.880	2.659	0.7979	1.2533	0	3.267	0	2.606	1.128	0.8865	0.853	0	3.686	0	3.267		
3	1.732	1.023	1.954	0.8862	1.1284	0	2.568	0	2.276	1.693	0.5907	0.888	0	4.358	0	2.574		
4	1.500	0.729	1.628	0.9213	1.0854	0	2.266	0	2.088	2.059	0.4857	0.880	0	4.698	0	2.282		
5	1.342	0.577	1.427	0.9400	1.0638	0	2.089	0	1.964	2.326	0.4299	0.864	0	4.918	0	2.114		
6	1.225	0.483	1.287	0.9515	1.0510	0.030	1.970	0.029	1.874	2.534	0.3948	0.848	0	5.078	0	2.004		
7	1.134	0.419	1.182	0.9594	1.0423	0.118	1.882	0.113	1.806	2.704	0.3698	0.833	0.204	5.204	0.076	1.924		
8	1.061	0.373	1.099	0.9650	1.0363	0.185	1.815	0.179	1.751	2.847	0.3512	0.820	0.388	5.306	0.136	1.864		
9	1.000	0.337	1.032	0.9693	1.0317	0.239	1.761	0.232	1.707	2.970	0.3367	0.808	0.547	5.393	0.184	1.816		
10	0.949	0.308	0.975	0.9727	1.0281	0.284	1.716	0.276	1.669	3.078	0.3249	0.797	0.687	5.469	0.223	1.777		
11	0.905	0.285	0.927	0.9754	1.0252	0.321	1.679	0.313	1.637	3.173	0.3152	0.787	0.811	5.535	0.256	1.744		
12	0.866	0.266	0.886	0.9776	1.0229	0.354	1.646	0.346	1.610	3.258	0.3069	0.778	0.922	5.594	0.283	1.717		
13	0.832	0.249	0.850	0.9794	1.0210	0.382	1.618	0.374	1.585	3.336	0.2998	0.770	1.025	5.647	0.307	1.693		
14	0.802	0.235	0.817	0.9810	1.0194	0.406	1.594	0.399	1.563	3.407	0.2935	0.763	1.118	5.696	0.328	1.672		
15	0.775	0.223	0.789	0.9823	1.0180	0.428	1.572	0.421	1.544	3.472	0.2880	0.756	1.203	5.741	0.347	1.653		
16	0.750	0.212	0.763	0.9835	1.0168	0.448	1.552	0.440	1.526	3.532	0.2831	0.750	1.282	5.782	0.363	1.637		
17	0.728	0.203	0.739	0.9845	1.0157	0.466	1.534	0.458	1.511	3.588	0.2787	0.744	1.356	5.820	0.378	1.622		
18	0.707	0.194	0.718	0.9854	1.0148	0.482	1.518	0.475	1.496	3.640	0.2747	0.739	1.424	5.856	0.391	1.608		
19	0.688	0.187	0.698	0.9862	1.0140	0.497	1.503	0.490	1.483	3.689	0.2711	0.734	1.487	5.891	0.403	1.597		
20	0.671	0.180	0.680	0.9869	1.0133	0.510	1.490	0.504	1.470	3.735	0.2677	0.729	1.549	5.921	0.415	1.585		
21	0.655	0.173	0.663	0.9876	1.0126	0.523	1.477	0.516	1.459	3.778	0.2647	0.724	1.605	5.951	0.425	1.575		
22	0.640	0.167	0.647	0.9882	1.0119	0.534	1.466	0.528	1.448	3.819	0.2618	0.720	1.659	5.979	0.434	1.566		
23	0.626	0.162	0.633	0.9887	1.0114	0.545	1.455	0.539	1.438	3.858	0.2592	0.716	1.710	6.006	0.443	1.557		
24	0.612	0.157	0.619	0.9892	1.0109	0.555	1.445	0.549	1.429	3.895	0.2567	0.712	1.759	6.031	0.451	1.548		
25	0.600	0.153	0.606	0.9896	1.0105	0.565	1.435	0.559	1.420	3.931	0.2544	0.708	1.806	6.056	0.459	1.541		

For n > 25.

```
In [ ]: # We can use the same function again. This time we need to specify the new K_alpha val
```

```
data_XR_alpha = qcc.ControlCharts.XbarR(data, K = K_alpha)
```

Xbar-R charts



## Point 2

Determine the operating characteristic curve (OC) for the X-bar chart (by using  $K=3$  and expressing the shift of the mean in standard deviation units).

### Solution

To determine the OC curve, we need to compute the probability of  $\beta$  for each value of the shift  $\mu$ .

We are testing the null hypothesis  $H_0$  that the sample mean  $\bar{X}$  is normally distributed with mean  $\mu_0$  and variance  $\sigma^2/n$ .

$$H_0 : \bar{X} \sim N(\mu_0, \sigma^2/n)$$

The alternative hypothesis is that the sample mean is normally distributed with mean  $\mu_1$  and variance  $\sigma^2/n$ .

$$H_1 : \bar{X} \sim N(\mu_1, \sigma^2/n)$$

So  $\beta$  is the probability of rejecting  $H_0$  when  $H_1$  is true.

$$\beta = P(LCL \leq \bar{X} \leq UCL | H_1)$$

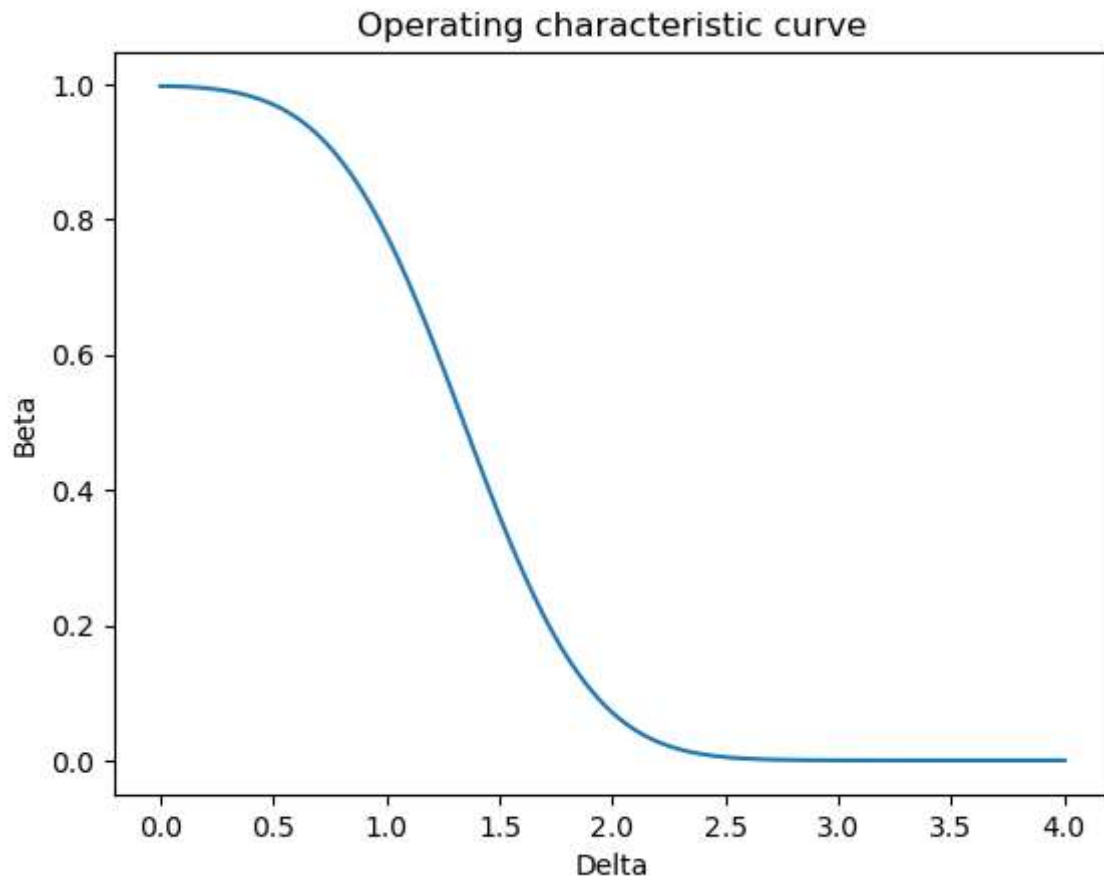
$$\beta = P(Z \leq \frac{UCL - \mu_1}{\sigma/\sqrt{n}}) - P(Z \leq \frac{LCL - \mu_1}{\sigma/\sqrt{n}})$$

If we define  $\delta = (\mu_1 - \mu_0)/\sigma$ , we can write:

$$\beta = P(Z \leq 3 - \delta\sqrt{n}) - P(Z \leq -3 - \delta\sqrt{n})$$

```
In [ ]: # Define a range of values for beta
delta = np.linspace(0, 4, 100)
# Compute the corresponding beta values
beta = stats.norm.cdf(3 - delta*np.sqrt(n)) - stats.norm.cdf(-3 - delta*np.sqrt(n))

# Plot the beta values
plt.plot(delta, beta)
plt.xlabel('Delta')
plt.ylabel('Beta')
plt.title('Operating characteristic curve')
plt.show()
```



## Point 3

Determine the corresponding ARL curve.

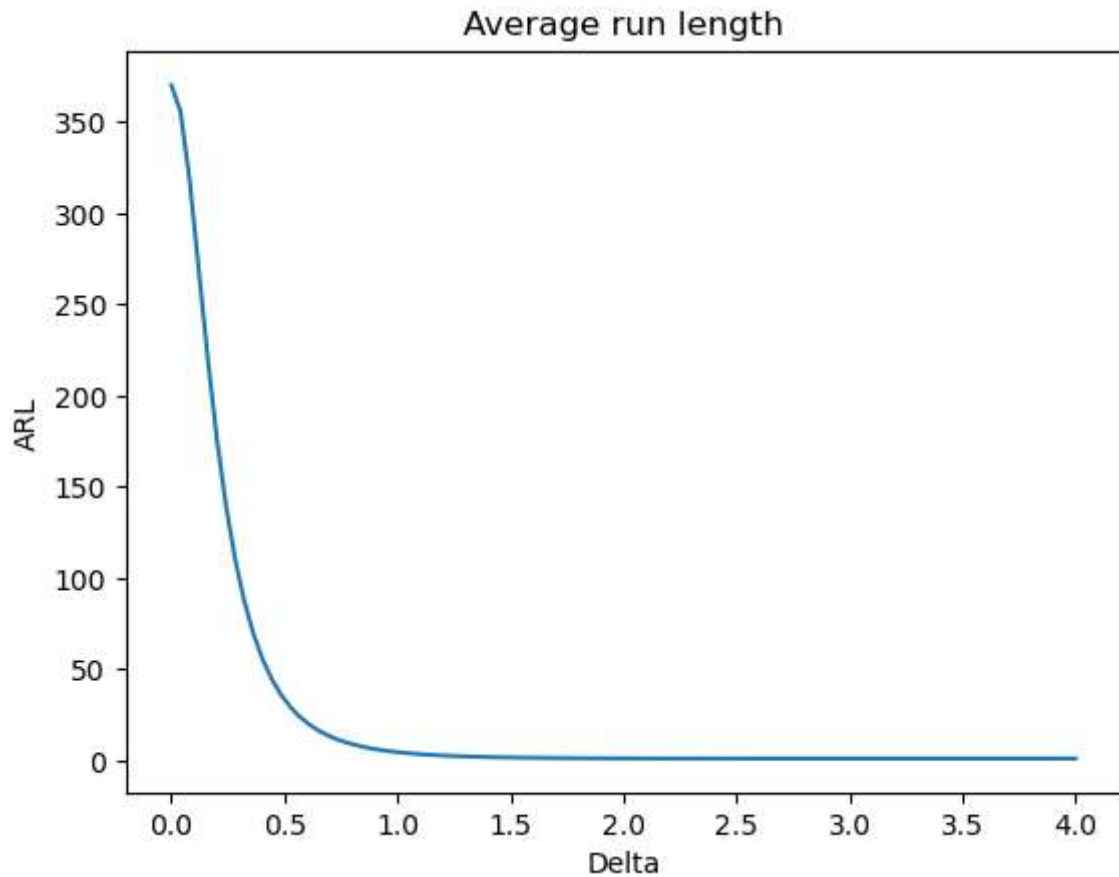
## Solution

We know that the ARL curve is defined as:

$$ARL = \frac{1}{1 - \beta}$$

```
In [ ]: # Compute ARL using the previous values of beta
        ARL = 1/(1-beta)

        # Plot the ARL values
        plt.plot(delta, ARL)
        plt.xlabel('Delta')
        plt.ylabel('ARL')
        plt.title('Average run length')
        plt.show()
```



## Point 4

Estimate the standard deviation through the statistic  $R$ .

### Solution

The standard deviation is estimated through the statistic  $R$  as:

$$\hat{\sigma} = \frac{\bar{R}}{d_2(n)}$$

You can use the function `getd2` from `qcc.constants` to get the value of  $d_2(n)$ .

```
In [ ]: d2 = qcc.constants.getd2(n)
sigma_hat = R_mean / d2
print('Sigma_hat = %.3f' % sigma_hat)
```

Sigma\_hat = 0.135

## Point 5

Design the confidence interval on the process mean that corresponds to the control limits computed in point 1.

### Solution

The confidence interval corresponding to the control limits computed in point 1 uses:

- $n = 5$
- $\alpha = 0.002$
- $\hat{\sigma} = 0.135$  (computed from the data)
- $\bar{X} = 10.008$  (computed from the data)

Remember the formula of the confidence interval (assume that  $\hat{\sigma}$  is the real population variance):

$$\bar{X} - z_{\alpha/2} \frac{\hat{\sigma}}{\sqrt{n}} \leq \mu \leq \bar{X} + z_{\alpha/2} \frac{\hat{\sigma}}{\sqrt{n}}$$

You can compute the CI using the formula or using the `interval` function from the `stats.norm` package.

```
In [ ]: CI = stats.norm.interval(1-alpha, loc=Xbar_mean, scale=sigma_hat/np.sqrt(n))
print('CI = (%.3f, %.3f)' % CI)
```

CI = (9.822, 10.195)

The CI limits correspond to the LCL and UCL of the control chart. Indeed, the X-bar control chart can be interpreted as a recursive application of the confidence interval on the mean along time  $t$ , assuming that the estimated sample variance is the real (population) variance.