Feature Engineering for Data Science

PCA (Principal Component Analysis)

- Image compression
- Removal of uncorrelated phenomena (irregularities) in the image
- Methods for anomaly detection in the image
    - Localization of uncorrelated phenomena by filtering correlated phenomena using PCA (X-Xpca => occurrence of anomalies in the image)
    - Localization of uncorrelated phenomena by filtering correlated phenomena using PCA (Xpca => occurrence of anomalies in the image)

Note (SVD ... Singular Value Decomposition)

The SVD method (decomposition) is analogous and is used for the same purposes as SVD in signal and image processing. SVD is recommended for sparse matrices (images), while PCA is recommended in the opposite cases.

PCA

Compression - Reducing the number of columns in matrix/image $\mathbf{X}$

Image

$$\mathbf{X} = [\quad \mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_M \,]$$

$$= \begin{bmatrix} x_1(k=1) & x_2(k=1) & \cdots & x_M(k=1) \\ x_1(k=2) & x_2(k=2) & \cdots & x_M(k=2) \\ \vdots & \vdots & \cdots & \vdots \\ x_1(k=N) & x_2(k=N) & \cdots & x_M(k=N) \end{bmatrix} \ldots N \times M$$

Column Centering (typically Z-Score here)

$$\mathbf{x}_i \leftarrow \frac{\mathbf{x}_i - \overline{\mathbf{x}_i}}{\sigma_i}; i = 1 \ldots M$$

Covariance Matrix (matrix of correlation coefficients between columns of $\mathbf{X}$)

$$\mathbf{covX} = \frac{1}{N}\mathbf{X}^T \cdot \mathbf{X}.$$

Eigenvalues and Eigenvectors

$$d, \mathbf{V} = \text{eig}(\mathbf{covX}),$$

where

$$\mathbf{V} = [\,\mathbf{v}_1 \quad \mathbf{v}_2 \quad \ldots \quad \mathbf{v}_M \,] \text{ where } \mathbf{v}_i \ldots M \times 1.$$

In the columns of $\mathbf{V}_{PCA}$ are the eigenvectors of the covariance matrix.

We choose the number of most significant eigenvectors $m << M$.

Column Compression:

$$\mathbf{X}_{compressed} = \mathbf{X} \cdot \mathbf{V}_{[:,1:m]} \ldots N \times m.$$

Decompression

$$\mathbf{X}_{PCA} = (\mathbf{X} \cdot \mathbf{V}_{[:,1:m]}) \cdot \mathbf{V}^T_{[:,1:m]} \ldots N \times M,$$

where $\mathbf{V}^T_{[:,1:m]}$ is the pseudoinverse.

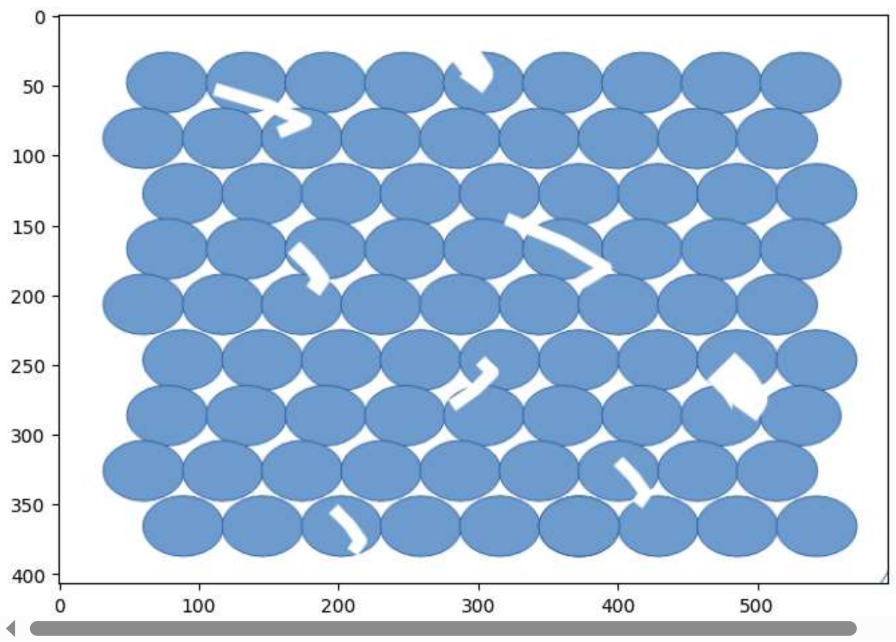
Then, transpose the matrix $\mathbf{X}_{PCA}$, and similarly compress the original image through rows...

PCA pro odfiltrování nekorelovaných anomálií pomocí n a m nejvýznamnějších vlastních vektorů kovarianční matice (zachovávají se lineárně korelované=opakující se jevy).

Or vice versa: PCA for filtering correlated phenomena (for compression and decompression, we do not use n and m most significant eigenvectors of the covariance matrix).

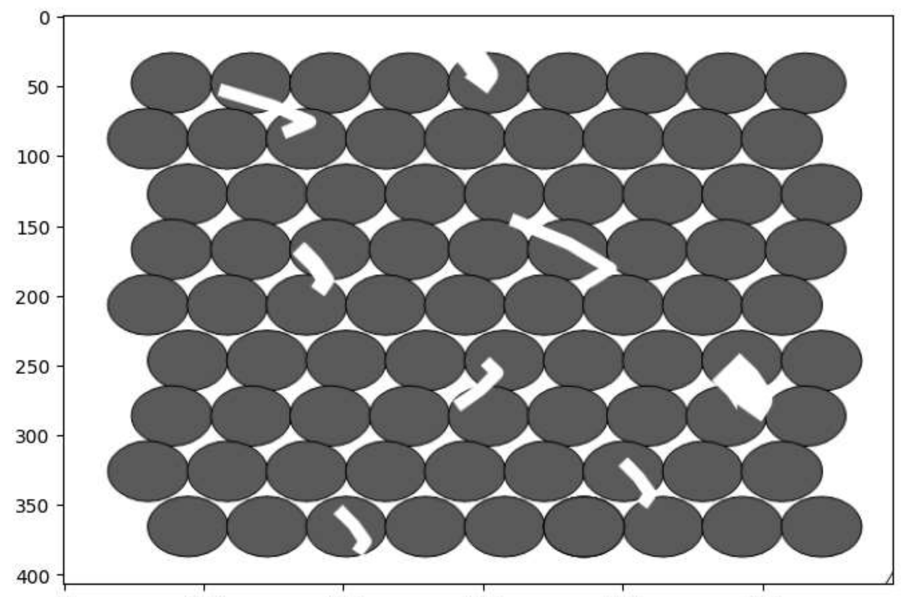
Similarly:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import PIL as pil
img1="./img/regular_patttern_with_perturb.png"


im1 = pil.Image.open(img1)


plt.figure(figsize=(8,8))
plt.imshow(im1)
plt.show()
```



```
from scipy import signal
im1=np.mean(im1,2)  # gray img, 2-D matrix,...X
```

```
plt.figure(figsize=(8,8))
plt.imshow(im1,cmap='gray')
plt.show()

print("im1 is now matrix, i.e.", type(im1), im1.shape)

```

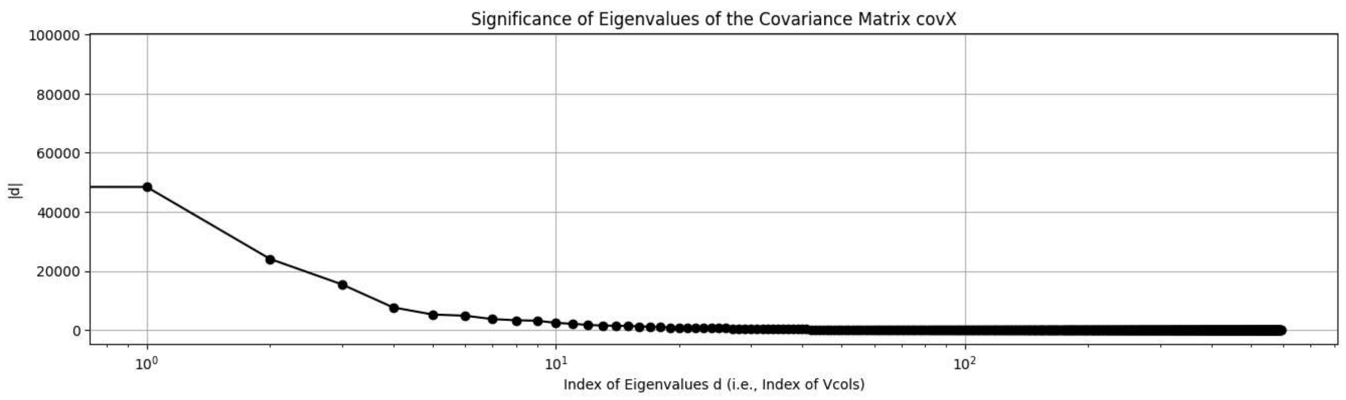


```
im1 is now matrix, i.e. <class 'numpy.ndarray'> (407, 594)
```

Localization of uncorrelated events by filtering correlated events using PCA (X-Xpca => occurrence of anomalies in the image)

i.e. filtering correlated events => occurrence of anomalies

```python
X = im1.copy()
X = (X - np.mean(X)) / np.std(X)

covX = np.dot(X.T, X)  # Covariance matrix (M x M)

d, Vcols = np.linalg.eig(covX)  # Eigenvalues and eigenvectors

plt.figure(figsize=(16, 4))
plt.title("Significance of Eigenvalues of the Covariance Matrix covX")
plt.semilogx(abs(d), '-ok')
plt.ylabel('|d|')
plt.xlabel('Index of Eigenvalues d (i.e., Index of Vcols)')
plt.grid()
plt.show()


```



Compression using PCA

(And its use for the localization of anomalous events by filtering uncorrelated events using PCA)

i.e., residuals = original - image without uncorrelated events => occurrence of anomalies

Compression through Columns and Subsequently through Rows

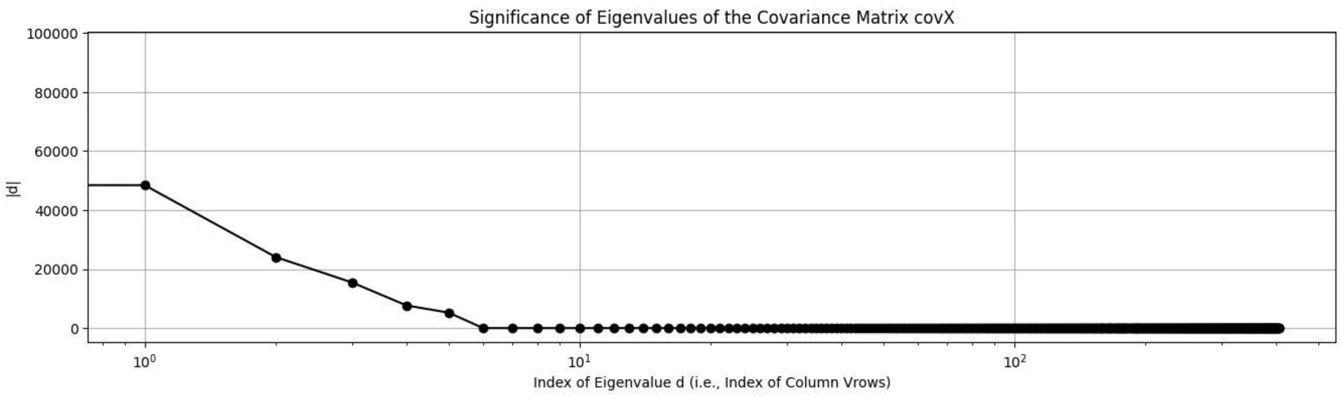
Through Columns: I choose $m << M$ most significant eigenvectors, i.e., n columns $\mathbf{V}_{cols}$ for which the eigenvalues d are the largest (see the previous graph) (the compressed image will have N rows and m columns).

```
print("X ... N x M =",X.shape)
m=6
print("choosing m=",m)

Xcols=np.dot(X,Vcols[:,:m])  # compressed na m sloupcu

print("Xcols ... N x m =",Xcols.shape)

```

```
X ... N x M = (407, 594)
choosing m= 6
Xcols ... N x m = (407, 6)
```

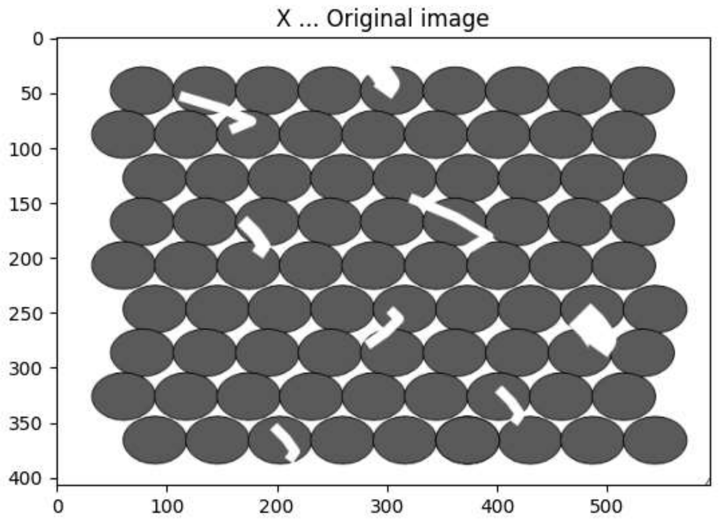
Along the rows: I select $n << N$ most significant eigenvectors, i.e., n columns $\mathbf{V}_{rows}$ for which the eigenvalues d are the largest (see the following graph).

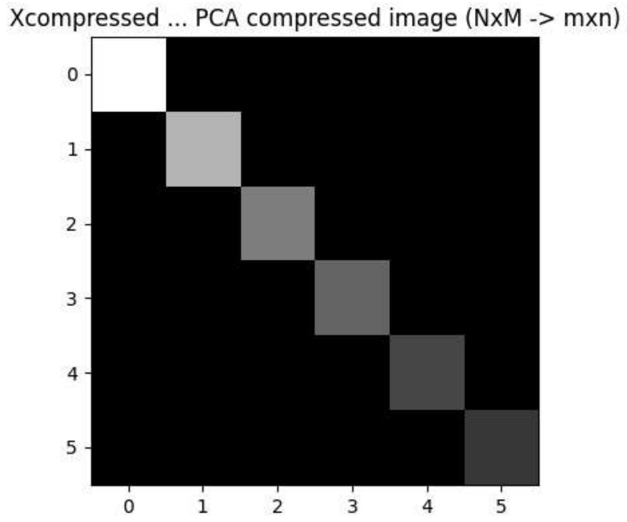
```
# Compression along the rows
Xrows = Xcols.T
covX = np.dot(Xrows.T, Xrows)  # Covariance matrix (N x N)
d, Vrows = np.linalg.eig(covX)

plt.figure(figsize=(16, 4))
plt.title("Significance of Eigenvalues of the Covariance Matrix covX")
plt.semilogx(abs(d), '-ok')
plt.ylabel('|d|')
plt.xlabel('Index of Eigenvalue d (i.e., Index of Column Vrows)')
plt.grid()
plt.show()

```

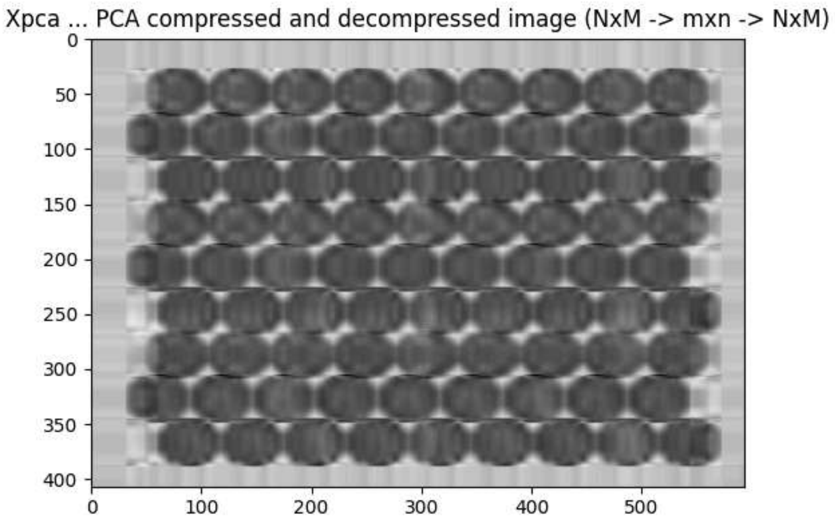


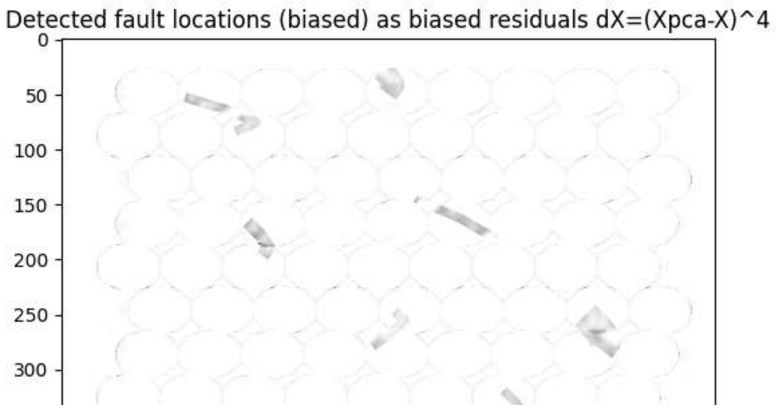
```
print("Xrows ... n x M =", Xrows.shape)
n = 6
print("Choice of n =", n)
Xcompressed = np.dot(Xrows, Vrows[:, :n])  # Compressed image with n rows and m columns
print("Xcompressed ... n x m =", Xcompressed.shape)

### Decompression - retaining only correlated events (the degree of correlation depends on the choice of m and n)

Xpca_rows = np.dot(Xcompressed, Vrows[:, :n].T)
Xpca_cols = Xpca_rows.T
Xpca = np.dot(Xpca_cols, Vcols[:, :m].T)

Xpca = np.real(Xpca)

plt.figure(figsize=(16, 20))
plt.subplot(411)
plt.imshow(X, cmap='gray')
plt.title("X ... Original image")
plt.subplot(412)
plt.imshow(abs(Xcompressed), cmap='gray')
plt.title("Xcompressed ... PCA compressed image (NxM -> mxn)")
plt.subplot(413)
plt.imshow(Xpca, cmap='gray')
plt.title("Xpca ... PCA compressed and decompressed image (NxM -> mxn -> NxM)")
plt.subplot(414)
dX = (Xpca - X) ** 4  # Thresholding (enhancing the difference between white and black), using powers of 2, 4, 6, 8
plt.imshow(-dX, cmap='gray')
plt.title("Detected fault locations (biased) as biased residuals dX=(Xpca-X)^4")
plt.show()

```

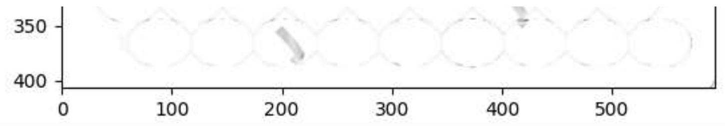
```
Xrows ... n x M = (6, 407)
Choice of n = 6
Xcompressed ... n x m = (6, 6)
```











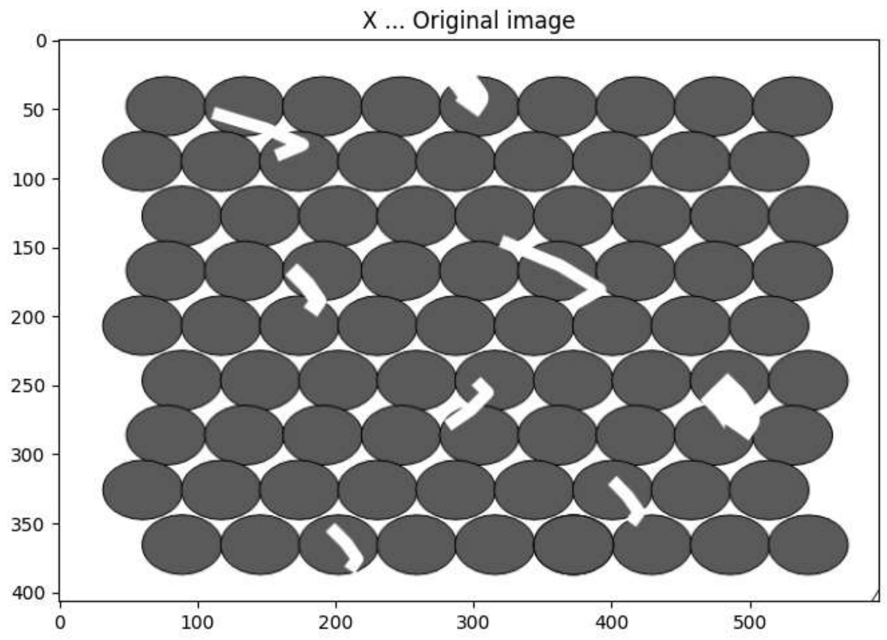
```
Xpca.shape
```

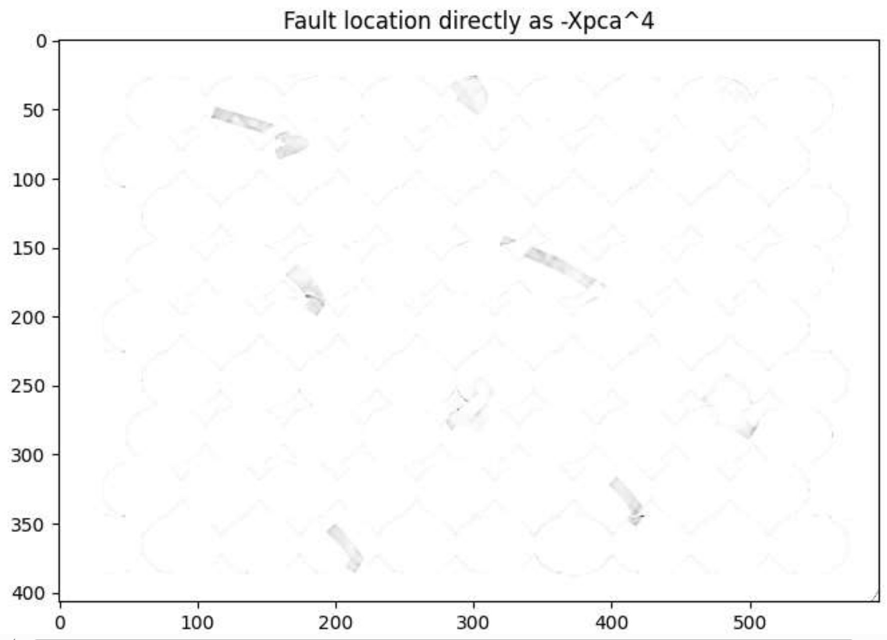
```
(407, 594)
```

Localization of uncorrelated events by filtering correlated events using PCA (Xpca => occurrence of anomalies in the image)

i.e. filtering correlated events => occurrence of anomalies

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import PIL as pil
from scipy import signal

# Path to the image
img1="./img/regular_patttern_with_perturb.png"
im1 = pil.Image.open(img1)
im1 = np.mean(im1, 2)  # Image as a single matrix

# Copy and normalize the image
X = im1.copy()
X = (X - np.mean(X)) / np.std(X)

# Calculate the covariance matrix (M x M)
covX = np.dot(X.T, X)

# Eigenvalues and eigenvectors
d, Vcols = np.linalg.eig(covX)

# Define the value 'm'
m = 6

# Project the data onto the first 'm' columns of Vcols
Xcols = np.dot(X, Vcols[:, m:])

# Project the data along the columns to get 'Xrows'
Xrows = Xcols.T

# Calculate the covariance matrix (N x N) for 'Xrows'
covX = np.dot(Xrows.T, Xrows)

# Eigenvalues and eigenvectors for 'Xrows'
d, Vrows = np.linalg.eig(covX)

# Define the value 'n'
n = 6

# Compress the image to 'n' rows and 'm' columns
Xcompressed = np.dot(Xrows, Vrows[:, n:])

# Perform the decompression
Xpca_rows = np.dot(Xcompressed, Vrows[:, n:].T)
Xpca_cols = Xpca_rows.T
Xpca = np.dot(Xpca_cols, Vcols[:, m:].T)
Xpca = np.real(Xpca)

# Create a figure for visualization
plt.figure(figsize=(10, 12))
plt.subplot(211)
plt.title("X ... Original image")
plt.imshow(X, cmap='gray')
plt.subplot(212)
plt.imshow(-Xpca**4, cmap='gray')
plt.title("Fault location directly as -Xpca^4")
plt.show()

```





Tasks:

T3.1 Try PCA on your own image for the detection of uncorrelated anomalies in the image (or noise).**[0.3 Points]**

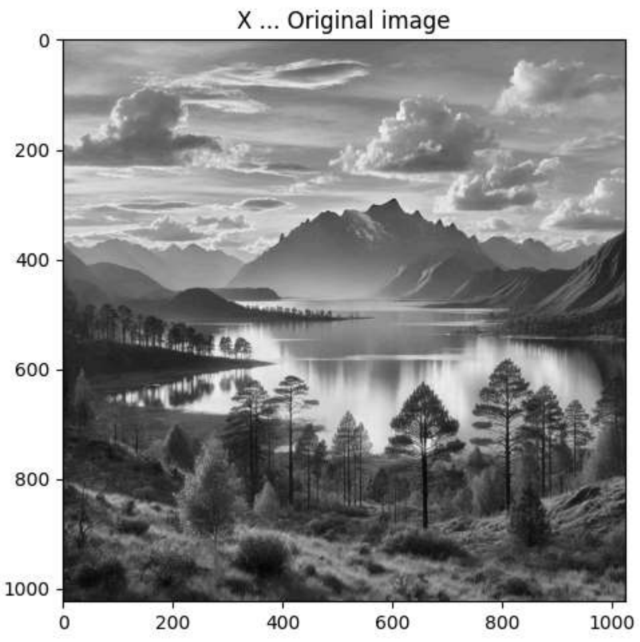
T3.2 Explore the influence of the choice of the number of eigenvectors on the quality of detection.**[0.3 Points]**

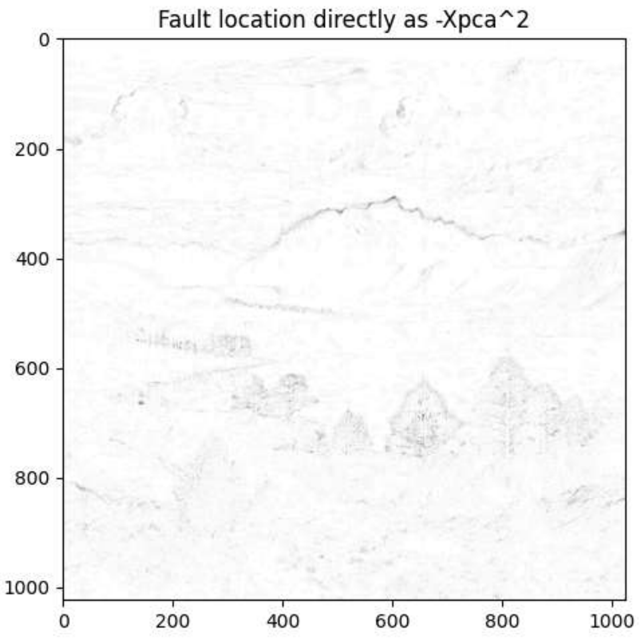
T3.3 Create concise documentation for the solution, code, and the achieved results (3-5 pages of A4). Properly cite the used sources!**[0.6 Points]**

```
# 3.1) Try PCA on your own image for the detection of uncorrelated anomalies in the image (or noise)
# 3.2) Explore the influence of the choice of the number of eigenvectors on the quality of detection.

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import PIL as pil
img1="/content/image1.webp"


im1 = pil.Image.open(img1)

from scipy import signal
im1=np.mean(im1,2)  # gray img, 2-D matrix,...X

X = im1.copy()
```

```python
X = (X - np.mean(X)) / np.std(X)

# Calculate the covariance matrix (M x M)
covX = np.dot(X.T, X)

# Eigenvalues and eigenvectors
d, Vcols = np.linalg.eig(covX)

print("X ... N x M =",X.shape)

m = 8

# Project the data onto the first 'm' columns of Vcols
Xcols = np.dot(X, Vcols[:, m:])

# Project the data along the columns to get 'Xrows'
Xrows = Xcols.T

# Calculate the covariance matrix (N x N) for 'Xrows'
covX = np.dot(Xrows.T, Xrows)

# Eigenvalues and eigenvectors for 'Xrows'
d, Vrows = np.linalg.eig(covX)

# Define the value 'n'
n = 8

# Compress the image to 'n' rows and 'm' columns
Xcompressed = np.dot(Xrows, Vrows[:, n:])

# Perform the decompression
Xpca_rows = np.dot(Xcompressed, Vrows[:, n:].T)
Xpca_cols = Xpca_rows.T
Xpca = np.dot(Xpca_cols, Vcols[:, m:].T)
Xpca = np.real(Xpca)

# Create a figure for visualization
plt.figure(figsize=(10, 12))
plt.subplot(211)
plt.title("X ... Original image")
plt.imshow(X, cmap='gray')
plt.subplot(212)
plt.imshow(-Xpca**2, cmap='gray')
plt.title("Fault location directly as -Xpca^2")
plt.show()
```

```
X ... N x M = (1024, 1024)
```

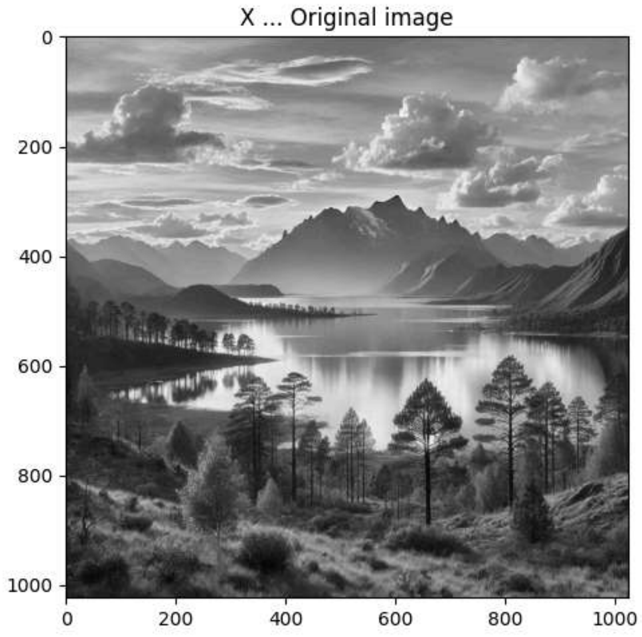


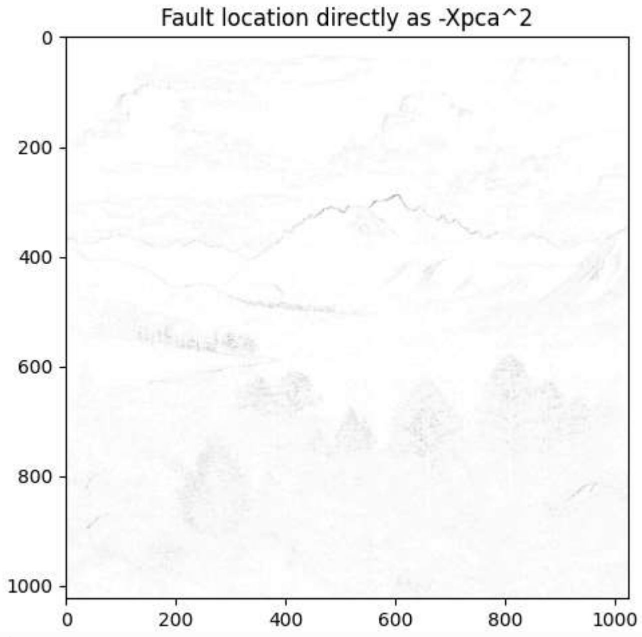


```python

X = im1.copy()
X = (X - np.mean(X)) / np.std(X)

# Calculate the covariance matrix (M x M)
covX = np.dot(X.T, X)

# Eigenvalues and eigenvectors
d, Vcols = np.linalg.eig(covX)

print("X ... N x M =",X.shape)

m = 50

# Project the data onto the first 'm' columns of Vcols
Xcols = np.dot(X, Vcols[:, m:])

# Project the data along the columns to get 'Xrows'
Xrows = Xcols.T

# Calculate the covariance matrix (N x N) for 'Xrows'
covX = np.dot(Xrows.T, Xrows)

# Eigenvalues and eigenvectors for 'Xrows'
d, Vrows = np.linalg.eig(covX)

# Define the value 'n'
n = 50
```

```python

# Compress the image to 'n' rows and 'm' columns
Xcompressed = np.dot(Xrows, Vrows[:, n:])

# Perform the decompression
Xpca_rows = np.dot(Xcompressed, Vrows[:, n:].T)
Xpca_cols = Xpca_rows.T
Xpca = np.dot(Xpca_cols, Vcols[:, m:].T)
Xpca = np.real(Xpca)

# Create a figure for visualization
plt.figure(figsize=(10, 12))
plt.subplot(211)
plt.title("X ... Original image")
plt.imshow(X, cmap='gray')
plt.subplot(212)
plt.imshow(-Xpca**2, cmap='gray')
plt.title("Fault location directly as -Xpca^2")
plt.show()
```

```
X ... N x M = (1024, 1024)
```





```python
X = im1.copy()
X = (X - np.mean(X)) / np.std(X)

# Calculate the covariance matrix (M x M)
covX = np.dot(X.T, X)

# Eigenvalues and eigenvectors
```

```
d, Vcols = np.linalg.eig(covX)

print("X ... N x M =",X.shape)

m = 1
```