Отчёт по лабораторной работе №7 "Метод главных компонент"

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm
```

In [2]:
```
from scipy.io import loadmat

def load_file(filename, keys=None):
    if keys is None:
        keys = ['X', 'y']
    mat = loadmat(filename)
    ret = tuple([mat[k].reshape(mat[k].shape[0]) if k.startswith('y') else mat[k] for k in keys])
    return ret
```

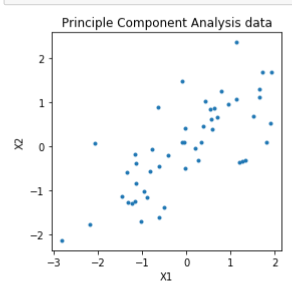
1. Загрузите данные ex7data1.mat из файла.

In [3]:
```
X, = load_file('ex7data1.mat', keys=['X'])
X = X - X.mean(axis=0)  # mean normalization
print(f'X shape: {X.shape}')
```

```
X shape: (50, 2)
```

2. Постройте график загруженного набора данных.

In [4]:
```
ax = plt.subplot()
ax.plot(X[:, 0], X[:, 1], marker='o', linestyle="None", markersize=3)
ax.set_aspect('equal')
ax.set_title("Principle Component Analysis data")
ax.set_xlabel('X1')
ax.set_ylabel('X2')
plt.show()
```



3. Реализуйте функцию вычисления матрицы ковариации данных.

In [5]:
```
def cov(X):
    return np.dot(X.T, X) / X.shape[0]
```

4. Вычислите координаты собственных векторов для набора данных с помощью сингулярного разложения матрицы ковариации (разрешается использовать библиотечные реализации матричных разложений).

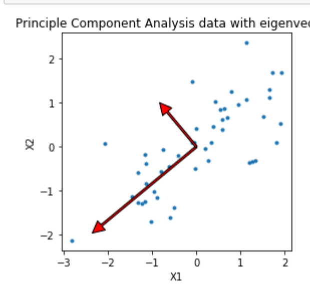
In [6]:
```
def get_eigenvectors(X):
    Sigma = cov(X)
    return np.linalg.svd(Sigma, full_matrices=False)
```

In [7]:
```
U, S, V = get_eigenvectors(X)
print(U)
```

```
[[-0.76908153 -0.63915068]
 [-0.63915068  0.76908153]]
```

5. Постройте на графике из пункта 2 собственные векторы матрицы ковариации.

In [8]:
```
mu = X.mean(axis=0)
projected_data = np.dot(X, U)
sigma = projected_data.std(axis=0).mean()
fig, ax = plt.subplots()
ax.plot(X[:, 0], X[:, 1], marker='o', linestyle="None", markersize=3)
for ind, axis in enumerate(U):
    start, end = mu, mu + (S[ind] + sigma) * axis
    ax.annotate(
        '', xy=end, xycoords='data',
        xytext=start, textcoords='data',
        arrowprops=dict(facecolor='red', width=2.0))
ax.set_aspect('equal')
ax.set_title("Principle Component Analysis data with eigenvectors")
ax.set_xlabel('X1')
ax.set_ylabel('X2')
plt.show()
```



6. Реализуйте функцию проекции из пространства большей размерности в пространство меньшей размерности с помощью метода главных компонент.

In [10]:
```
def transform(X, k=None):
    if k is None:
        k = X.shape[1] - 1
    U, *_ = get_eigenvectors(X)
    U_reduce = U[:, :k]
    return np.dot(X, U_reduce)
```

In [11]:
```
X_reduced = transform(X)
print(f'Reduced X shape: {X_reduced.shape}')
```

```
Reduced X shape: (50, 1)
```

7. Реализуйте функцию вычисления обратного преобразования.

In [12]:
```
def inverse_transform(X, Z, k=None):
    if k is None:
        k = X.shape[1] - 1
    U, *_ = get_eigenvectors(X)
    U_reduce = U[:, :k]
    return np.dot(Z, U_reduce.T)
```

In [13]:
```
X_approx = inverse_transform(X, X_reduced)
print(f'Inversed X shape: {X_approx.shape}')
```

```
Inversed X shape: (50, 2)
```

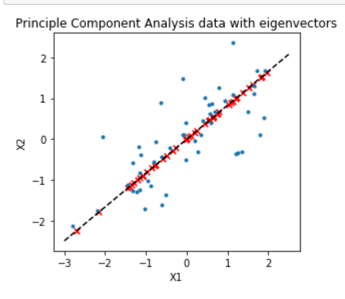
8. Постройте график исходных точек и их проекций на пространство меньшей размерности (с линиями проекций).

In [14]:
```
mu = X.mean(axis=0)
fig, ax = plt.subplots()
ax.plot(X[:, 0], X[:, 1], marker='o', linestyle="None", markersize=3)

U, *_ = get_eigenvectors(X)
m = U[0][1] / U[0][0]
ax.plot(np.linspace(-3, 2.5, 10), m * np.linspace(-3, 2.5, 10),
        color="black", linestyle="--")

ax.scatter(X_approx[:, 0], X_approx[:, 1], c="r", marker="x", s=32)
ax.set_aspect('equal')

ax.set_title("Principle Component Analysis data with eigenvectors")
ax.set_xlabel('X1')
ax.set_ylabel('X2')
plt.show()
```



9. Загрузите данные ex7faces.mat из файла.

In [15]:
```
X_faces, = load_file('ex7faces.mat', keys=['X'])
print(f'X faces shape: {X_faces.shape}')
```

```
X faces shape: (5000, 1024)
```

10. Визуализируйте 100 случайных изображений из набора данных.

In [16]:
```
rand_inds = np.random.choice(np.arange(0, 5000), 100)
fig, axs = plt.subplots(10, 10, sharex=True, sharey=True, figsize=(20,20))
axs = axs.flatten()
graymap = plt.get_cmap("gray")

for i, indx in enumerate(rand_inds):
    im_mat = np.reshape(X_faces[indx, :], (32, 32), order="F")
    axs[i].imshow(im_mat, cmap=graymap, interpolation="None")
    axs[i].xaxis.set_visible(False)
    axs[i].yaxis.set_visible(False)
```

11. С помощью метода главных компонент вычислите собственные векторы.

In [17]:
```
X_faces = X_faces - X_faces.mean(axis=0)
Uf, Sf, Vf = get_eigenvectors(X_faces)
```

12. Визуализируйте 36 главных компонент с наибольшей дисперсией.

In [18]:
```
def plot_components(n_components):
    size = int(np.sqrt(n_components))
    fig, axs = plt.subplots(size, size, sharex=True, sharey=True, figsize=(10,10))
    fig.suptitle(f"{n_components} PCA Eigenfaces", fontsize=18)
    axs = axs.flatten()
    graymap = plt.get_cmap("gray")

    for i, indx in enumerate(range(n_components)):
        im_mat = np.reshape(Vf[indx, :], (32, 32), order="F")
        axs[i].imshow(im_mat, cmap=graymap, interpolation="None")
        axs[i].xaxis.set_visible(False)
        axs[i].yaxis.set_visible(False)
```

In [19]:
```
plot_components(36)
```

36 PCA Eigenfaces

13. Как изменилось качество выбранных изображений?

Качество изображений определенно стало хуже, однако хорошо прослеживаются основные черты лиц. Это значит, что компоненты с наибольшей диспресией отображают наиболее значимые признаки, например, такие как границы носа, глаз, рта, овала лица и т.д.

14. Визуализируйте 100 главных компонент с наибольшей дисперсией.

In [20]:
```
plot_components(100)
```

100 PCA Eigenfaces

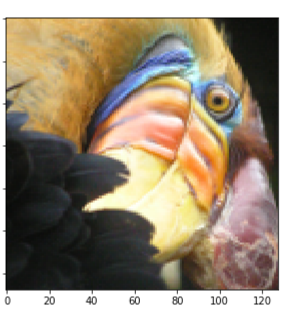
15. Как изменилось качество выбранных изображений?

Исходя из результатов можно заметить, что чем меньше дисперсия компонента, тем хуже прослеживаются черты лица. Это значит, что чем меньше диспресия компонента, тем менее важные признаки она отражает. Менее важные компоненты кодируют мелкие различия между лицами и шум.

16. Используйте изображение, сжатое в лабораторной работе №5.

In [21]:
```
A, = load_file('bird_small.mat', keys=['A'])
print(f'Shape: {A.shape}')
fig, axs = plt.subplots(ncols=1, figsize=[12, 5])
axs.imshow(A)
plt.show()
```

```
Shape: (128, 128, 3)
```



17. С помощью метода главных компонент визуализируйте данное изображение в 3D и 2D.

In [22]:
```
Ax = np.reshape(A, [A.shape[0] * A.shape[1], A.shape[2]])
Ax = Ax - Ax.mean(axis=0)
Ax_reduced = transform(Ax)
Ax_approx = inverse_transform(Ax, Ax_reduced)
```

Сначала цветное изображение 128 на 128 переводится в матрицу размерностью 16384 на 3.

С помощью реализованной в данной ЛР функции метода главных компонент исходная матрица сжимается в матрицу размерностью 16384 на 2.

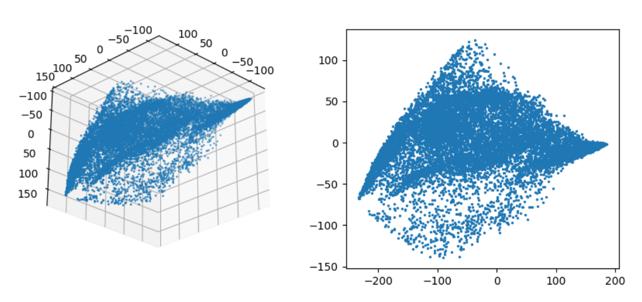
А затем она инверсируется и получается апроксимированная матрица 16384 на 3.

Далее апроксимированная матрица изображается в 3D пространстве, а сжатая матрица в 2D пространстве.

In [26]:
```
from mpl_toolkits.mplot3d import Axes3D
%matplotlib notebook

fig = plt.figure(figsize=(10, 4))
ax = fig.add_subplot(1, 2, 1, projection='3d')
ax.scatter(xs=Ax_approx[:, 0], ys=Ax_approx[:, 1], zs=Ax_approx[:, 2], cmap=cm.coolwarm, s=1)

ax2 = fig.add_subplot(1, 2, 2)
ax2.scatter(Ax_reduced[:, 0], Ax_reduced[:, 1], cmap=cm.coolwarm, s=2)
plt.show()
```



18. Соответствует ли 2D изображение какой-либо из проекций в 3D?

После трансформации признаков изображения получилась плоскость. Она соответствует одной из проекций в 3D изображении. На рисунке слева 3D изображение повёрнуто таким образом, что можно убедиться в совпадении проекции в 3D и изображении в 2D.