Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по предмету:

«Машинное обучение»

По лабораторной работе №7

«Метод главных компонент»

Выполнил: Сенькович Дмитрий Сергеевич

магистрант кафедры информатики

группы №858642

Проверил: Стержанов Максим Валерьевич

доцент, кандидат технических наук

Минск 2019

**Оглавление**

[1 Введение 2](#_Toc25805581)

[2 Проецирование точек на прямую 5](#_Toc25805582)

[2.1 Постановка задачи 5](#_Toc25805583)

[2.2 Чтение данных 5](#_Toc25805584)

[2.3 График исходных данных 5](#_Toc25805585)

[2.4 Реализация функции вычисления матрицы ковариации 6](#_Toc25805586)

[2.5 Реализация поиска собственных векторов данных с помощью сингулярного разложения матрицы ковариации 6](#_Toc25805587)

[2.6 График данные с собственными векторами матрицы ковариации 6](#_Toc25805588)

[2.7 Реализация функция проецирования данных 7](#_Toc25805589)

[2.8 Реализация обратного преобразования 8](#_Toc25805590)

[2.9 График исходных точек и их проекций 8](#_Toc25805591)

[3 Проецирование точек на прямую 10](#_Toc25805592)

[3.1 Постановка задачи 10](#_Toc25805593)

[3.2 Чтение данных 10](#_Toc25805594)

[3.3 Визуализация 100 случайных лиц 10](#_Toc25805595)

[3.4 Вычисление первых 36 собственных векторов и сжатых изображений 11](#_Toc25805596)

[3.5 Вычисление первых 100 собственных векторов и сжатых изображений 14](#_Toc25805597)

[4 Проецирование точек на прямую 17](#_Toc25805598)

[4.1 Постановка задачи 17](#_Toc25805599)

[4.2 Чтение данных 17](#_Toc25805600)

[4.3 3d и 2d проекция изображения 17](#_Toc25805601)

[4.4 Соответствие изображений в двух пространствах 19](#_Toc25805602)

# 1 Введение

Метод главных компонент - алгоритм, позволяющий снизить размерность пространства. Это актуально в задачах машинного обучения, в которых могут быть десятки тысяч фич, которые далеко не всегда информативны. Также, метод может быть полезен для визуализации многомерных данных.

Суть метода заключается в том, чтобы представить вектор из n-мерного пространства в k-мерного, пониженной размерности, с наименьшими потерями. Алгоритм минимизирует сумму евклидового расстояний до гиперплоскости, на которую проецируется выборка. Задача эта решается сингулярным разложением ковариационной матрицы выборки. Матрица U этого разложения и является матрицей векторов пространства фичей данной выборки, она, а, точнее, часть ее, и используется для проецирования данных.

Для начала, рассмотрим построение ковариационной матрицы:

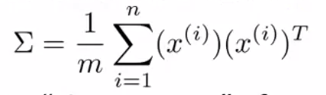


Рисунок 1.1. Формула вычисления ковариационной матрицы

Ковариационная матрица представляет из себя матрицу зависимости каждой фичи. То есть каждый элемент матрицы с индексами (i, j) - значения линейной зависимости фич i и j:



Рисунок 1.2. Формула вычисления ковариации двух случайных величин

Здесь M обозначает мат ожидание случайной величины.

Сингулярное разложение матрицы - singular value decomposition - используется в методе для получения векторов с наибольшей сохранившейся дисперсией после проецирования выборки на пространство размерности ниже оригинального. Это такое разложение матрицы A mxn на три произведение трех матриц U, S, Wt, что U mxm - ортогональная матрица, S mxn - матрица с ненулевыми значениями только на диагонали, V nxn - также ортогональная матрица. Элементы диагонали матрицы S - сингулярные числа матрицы A (корни из собственных векторов матрицы A\*A.T), столбцы матрицы U - собственные вектора матрицы A\*A.T, а столбцы матрицы Wt - собственные вектора матрицы A.T\*A. Собственный вектор матрицы - это такой вектор, при умножении на матрицу который остается “самим собой” с некоторым скалярным множителем - т.е. коллинеарный вектор.

Можно разложить нашу исходную матрицу по столбцам таким образом (множители перед столбцами - сингулярные значения матрицы A\*A.T):

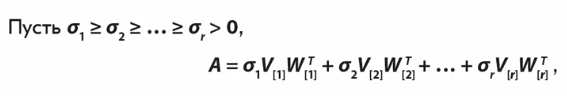


Рисунок 1.4. Разложение матрицы A по столбцам матриц svd разложения

Числа матриц V и Wt <= 1. Существует следующая гипотеза:



Рисунок 1.5. Гипотеза, позволяющая проецировать вектора на пространство меньше плоскости с потерями

Таким образом, можно взять некоторое количество больших сингулярных значений, подсчитать частичную сумму без маленьких сингулярных значений и ожидать, что она будет примерно равна исходной матрице.

Ошибка приближения при такой разложении выглядит следующим образом:

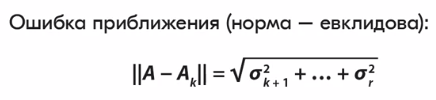


Рисунок 1.6. Ошибка приближения матрицы A

То есть при достаточно маленьких не используемых сингулярных значениях получается хорошее приближение с небольшой ошибкой. Такое приближение является лучшим приближением среди матриц ранга k (ранг матрицы - наивысший из порядков всех ненулевых миноров матрицы).

Далее, как это используется в методе главных компонент. Есть такое наблюдение, что данные собираются в эллипсоид в гиперплоскости, центрированный в мат ожиданиях фич этих данных - эллипсоид рассеяния:

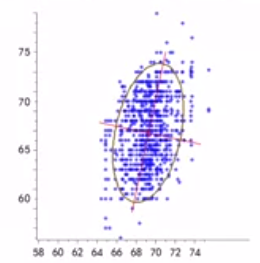


Рисунок 1.7. Пример эллипса рассеяния

Если рассмотреть ковариационную матрицу S этих центрированных данных, то будет иметь место следующее равенство:

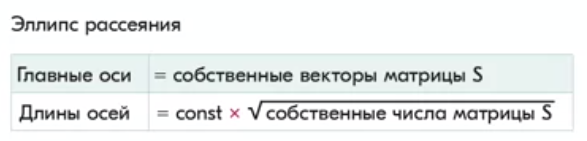


Рисунок 1.8. Взаимосвязь ковариационной матрицы S данных с эллипсом рассеяния этих данных

Если некоторое собственное значение значительно больше другого, то эллипс является “сплюснутым” в сторону собственного вектора, соответствующего большему из собственных значений. То есть можно выдать гипотезу, что на самом деле эти параметры имеют линейный закон связи, а рассеяние вызвано ошибками наблюдения. В этом случае вместо эллипса достаточно рассматривать только его проекцию на его главную большую ось.

Таким образом, можно разложить ковариационную матрицу в сингулярное разложение матриц и рассматривать лишь проекцию данных на некоторое количество векторов с наибольшими длинами - главными компонентами.

Почему можно использовать U в качестве вектора собственных векторов матрицы S? Потому что S - симметричная матрица. SVD - обобщение разложения матрицы на спектральное разложение квадратной матрицы.

# 2 Проецирование точек на прямую

### 2.1 Постановка задачи

Имеется выборка, представляющая из себя набор точек на плоскости. Требуется реализовать метода главных компонент.

### 2.2 Чтение данных

Считаем данные:

x = loadmat("ex7data1.mat")['X']

### 2.3 График исходных данных

Построим данные:

def plot\_data(x, c='b'):

plt.scatter(x[:, 0], x[:, 1], c=c)

plot\_data(x)

plt.show()

Полученный график:

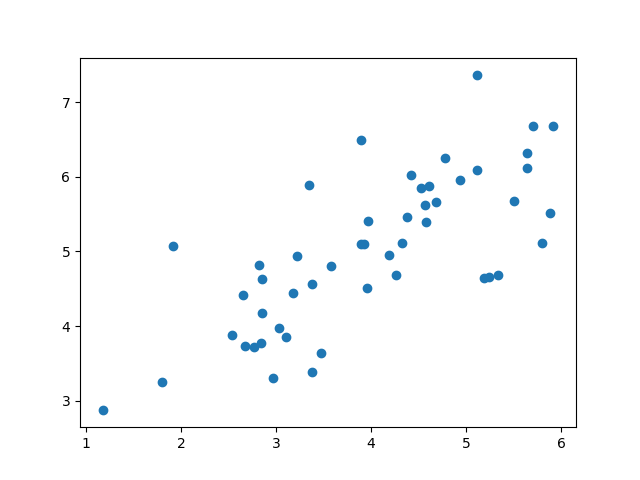


Рисунок 2.3.1. Исходные данные

### 2.4 Реализация функции вычисления матрицы ковариации

Реализуем функцию:

def compute\_covariance\_matrix(x):

m = x.shape[0]

features\_count = x.shape[1]

sum = np.zeros(shape=(features\_count, features\_count))

for xi in x:

col = xi.T.reshape(features\_count, 1)

sum += col.dot(col.T)

return sum / m

### 2.5 Реализация поиска собственных векторов данных с помощью сингулярного разложения матрицы ковариации

Использует сингулярное разложение матрицы из scipy, предварительно нормализовав данные:

def normalize\_and\_scale(x):

features\_count = x.shape[1]

means = np.zeros(shape=(features\_count, 1))

standard\_deviations = np.zeros(shape=(features\_count, 1))

for i in range(features\_count):

means[i] = np.mean(x[:, i])

standard\_deviations[i] = np.std(x[:, i])

normalized\_x = (x - means.T) / standard\_deviations.T

return normalized\_x, means, standard\_deviations

normalized\_x, means, standard\_deviations = normalize\_and\_scale(x)

covariance\_matrix = compute\_covariance\_matrix(normalized\_x)

u, s, c = np.linalg.svd(covariance\_matrix)

Здесь u и будут собственными векторами матрицы нормализованных данных.

### 2.6 График данные с собственными векторами матрицы ковариации

Построим график:

def plot\_eigenvectors(u):

ut = u.T

plt.arrow(0, 0, ut[0][0], ut[0][1], head\_width=0.05, head\_length=0.1, color='m')

plt.arrow(0, 0, ut[1][0], ut[1][1], head\_width=0.05, head\_length=0.1, color='m')

plot\_data(normalized\_x)

plot\_eigenvectors(u)

plt.show()

Получим следующий график (заметим, что мы строим просто собственные вектора, без произведения на собственные значения):

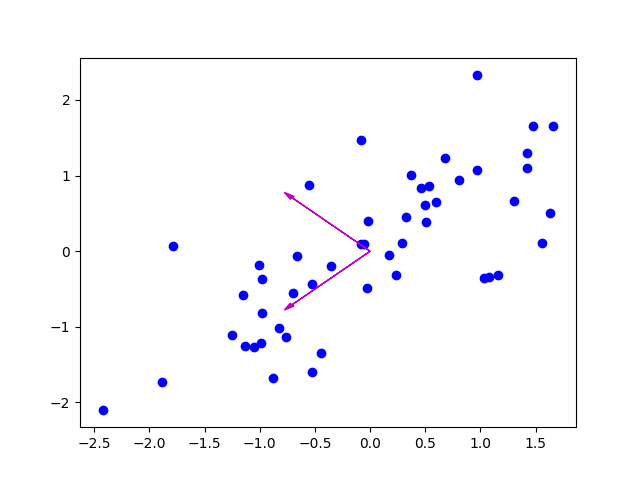


Рисунок 2.6.1. Исходные данные и собственные вектора ковариационной матрицы

### 2.7 Реализация функция проецирования данных

Реализуем функцию:

def project\_data(u\_reduce, x):

return x.dot(u\_reduce)

Здесь u\_reduce - матрицы первых k собственных векторов.

### 2.8 Реализация обратного преобразования

Восстановить данные в полном объеме, конечно, не получится, но можно получить проекцию данных на плоскость:

def recover\_data(z, u\_reduce):

return z.dot(u\_reduce.T)

### 2.9 График исходных точек и их проекций

Построим такой график:

def plot\_projections(normalized\_x, x\_approximated):

for i in range(len(normalized\_x)):

xi = normalized\_x[i]

xai = x\_approximated[i]

plt.plot([xi[0], xai[0]], [xi[1], xai[1]], c='y')

u\_reduce = u[:, :K]

z = project\_data(u\_reduce, normalized\_x)

x\_approximated = recover\_data(z, u\_reduce)

plot\_data(normalized\_x)

plot\_data(x\_approximated, c='r')

plot\_eigenvectors(u)

plot\_projections(normalized\_x, x\_approximated)

plt.show()

Здесь вы выбираем первые k собственных векторов ковариационной матрицы данных (в нашем случае k = 1) и проецируем данные на большую ось эллипса рассеяния. Далее, строим график, указывая исходные данные (синим цветом), проекции этих исходных данных (красные точки), соединяем их для наглядности (желтым цветом) и строим собственные вектора (фиолетовым):

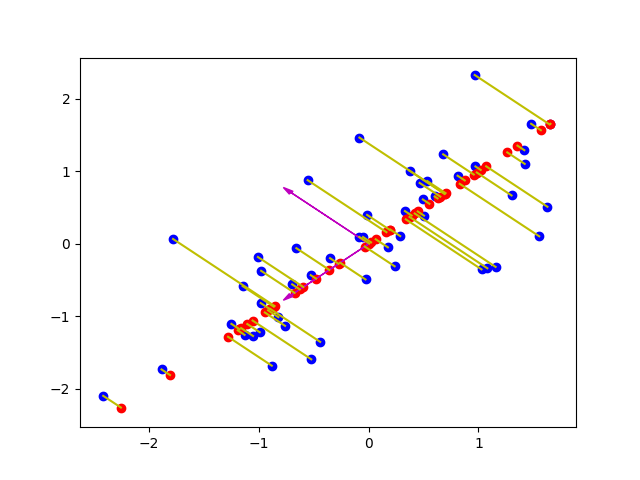


Рисунок 2.9.1. Данные и их проекции

# 3 Проецирование точек на прямую

### 3.1 Постановка задачи

Дан набор 5000 изображений лиц 32x32 в оттенках серого. Исследуем сжатие изображений с помощью svd.

### 3.2 Чтение данных

Считаем данные:

x = loadmat("ex7faces.mat")['X']

### 3.3 Визуализация 100 случайных лиц

Визуализируем 100 случайных лиц:

def pick\_n\_random\_indices(max=5000, n=100):

random\_indices = [0]\*n

for i in range(n):

random\_indices[i] = random.randint(0, max)

return random\_indices

def plot\_n\_random\_faces(x, indices, n=100):

sample\_width = int(math.sqrt(len(x[0])))

plot\_dim = int(math.sqrt(n))

fig, axises = plt.subplots(plot\_dim, plot\_dim, figsize=(10, 10))

axises = axises.ravel()

for i in range(axises.shape[0]):

axis = axises[i]

axis.imshow(x[indices[i]].reshape(sample\_width, sample\_width, order='F'), cmap='gray')

axis.axis('off')

Полученные изображения:



Рисунок 3.3.1. 100 случайных чисел

### 3.4 Вычисление первых 36 собственных векторов и сжатых изображений

Выберем первые 36 собственных векторов (они уже отсортированы по количеству дисперсии по убыванию), спроецируем изображения на них и сравнив с оригиналом:

def plot\_eigenvectors(u):

sample\_width = int(math.sqrt(len(u[0])))

plot\_dim = int(math.sqrt(len(u)))

fig, axises = plt.subplots(plot\_dim, plot\_dim, figsize=(10, 10))

axises = axises.ravel()

for i in range(axises.shape[0]):

axis = axises[i]

axis.imshow(u[i].reshape(sample\_width, sample\_width, order='F'), cmap='gray')

axis.axis('off')

# ------------ first 36 eigenvectors and compressed images ---------

# first k eigenvectors will give max variance as s is already sorted in descending order

K = 36

u\_reduce = u[:, :K]

z = project\_data(u\_reduce, normalized\_x)

x\_approximated = recover\_data(z, u\_reduce)

plot\_eigenvectors(u\_reduce.T)

plt.show()

plot\_n\_random\_faces(x\_approximated, indices, n)

plt.show()

# ------------ first 36 eigenvectors and compressed images ---------

Получим следующие изображения этих собственных векторов:

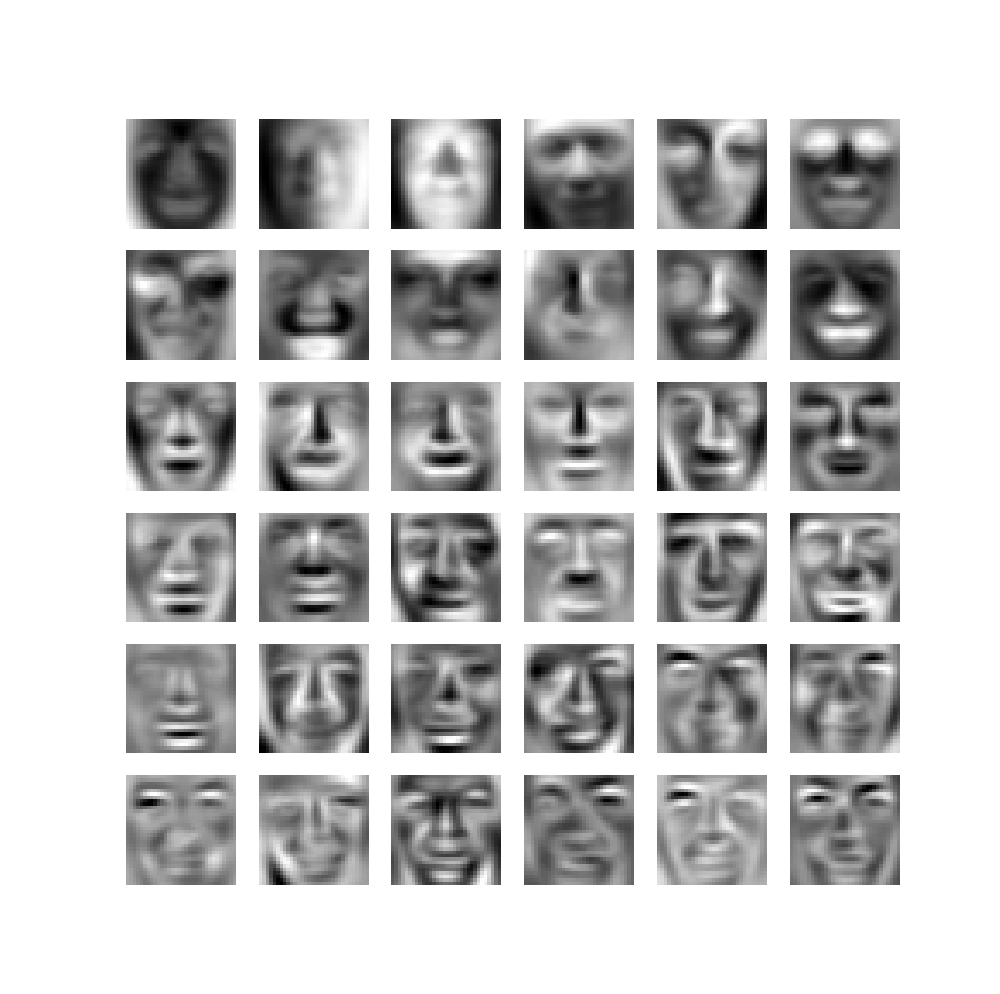


Рисунок 3.4.1. Первые 36 собственных векторов по количеству дисперсии

Проецированные на эти векторы изображения:



Рисунок 3.4.2. 100 изображений из предыдущего пункта, спроецированные на 36 собственных векторов по количеству дисперсии

Видим значительную потерю качества изображений.

### 3.5 Вычисление первых 100 собственных векторов и сжатых изображений

Проведем тот же эксперимент, но уже со 100 первыми собственными векторами по количеству дисперсии. Ниже приведены изображения этих векторов:

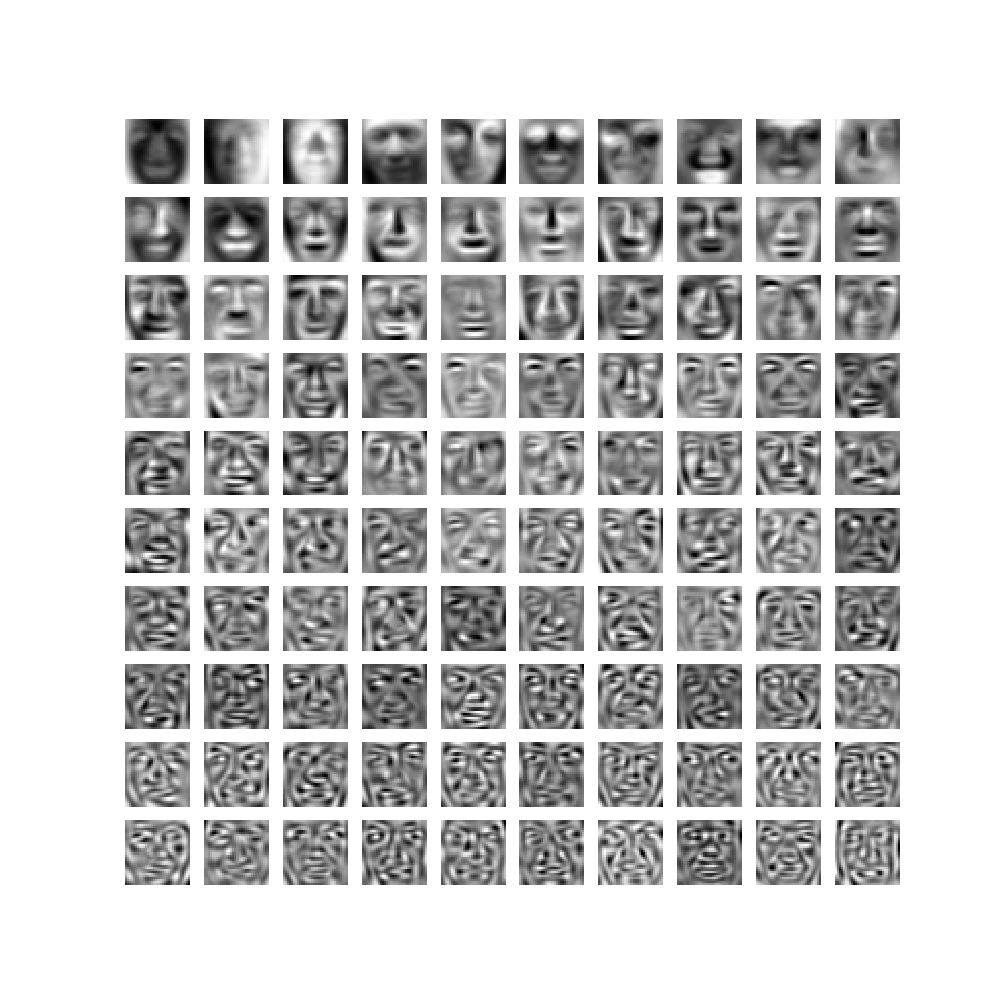


Рисунок 3.5.1. Первые 100 собственных векторов по количеству дисперсии

Приведем 100 сжатых изображений, соответствующих проекциям векторов изображений на эти 100 собственных векторов:



Рисунок 3.5.2. 100 изображений из предыдущего пункта, спроецированные на 100 собственных векторов по количеству дисперсии

Видим существенное изменение качества спроецированных изображений, можно найти довольно сильные сходства с оригинальными. Это объясняется тем, что мы выбрали почти в 3 раза больше собственных векторов, причем с наибольшей дисперсией - т.е. связью между оригинальными фичами.

# 4 Проецирование точек на прямую

### 4.1 Постановка задачи

Дано изображение, сжатое в 5 лабораторной работе 16-ю цветами. Изобразить это изображение в 3d и спроецировать в 2d.

### 4.2 Чтение данных

Считаем данные:

x = loadmat('compressed.mat')['X'].reshape(-1, 3)

### 4.3 3d и 2d проекция изображения

Изобразим компоненты изображения в 3d:

def plot\_image\_3d(x):

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(x[:, 0], x[:, 1], x[:, 2])

# ------------ normalization and svd ---------

normalized\_x, means, standard\_deviations = normalize\_and\_scale(x)

covariance\_matrix = compute\_covariance\_matrix(normalized\_x)

u, s, c = np.linalg.svd(normalized\_x.T)

# ------------ normalization and svd ---------

# ------------ plot image in 3d ---------

K = 3

u\_reduce = u[:, :K]

z = project\_data(u\_reduce, normalized\_x)

x\_approximated = recover\_data(z, u\_reduce)

plot\_image\_3d(x\_approximated)

plt.show()

# ------------ plot image in 3d ---------

Получим следующее изображение:

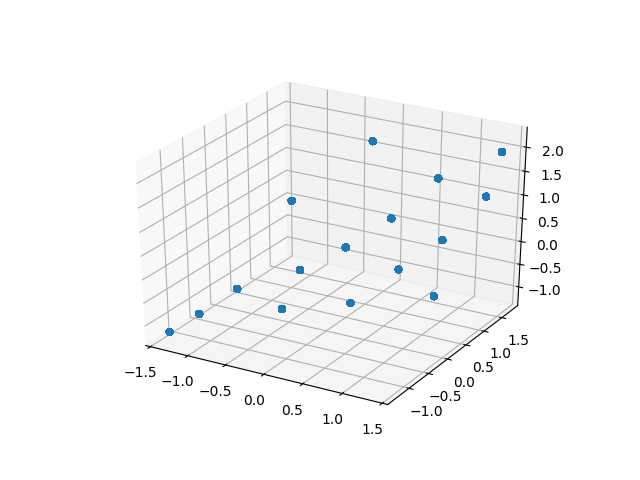


Рисунок 4.3.1. Компоненты изображения в 3d

Теперь получим спроецированное изображение на плоскость с помощью svd:

def plot\_image\_2d(x):

plt.scatter(x[:, 0], x[:, 1])

# ------------ plot image in 2d ---------

K = 2

u\_reduce = u[:, :K]

z = project\_data(u\_reduce, normalized\_x)

x\_approximated = recover\_data(z, u\_reduce)

plot\_image\_2d(x\_approximated)

plt.show()

# ------------ plot image in 2d ---------

Получим следующее изображение:

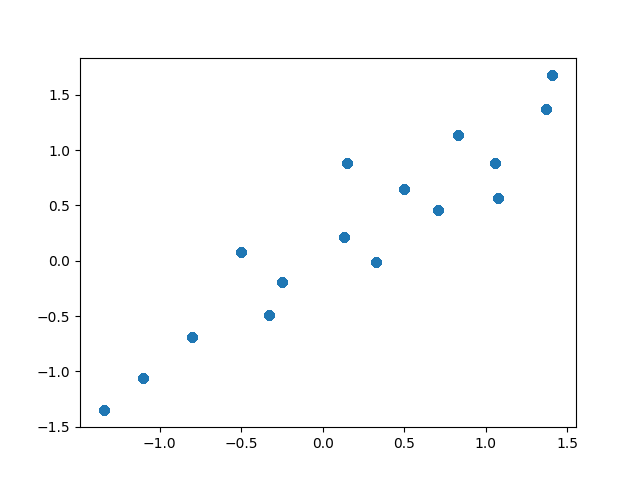


Рисунок 4.3.2. Компоненты изображения, спроецированные на плоскость

### 4.4 Соответствие изображений в двух пространствах

Следует заметить, что изображение проекции, полученное с помощью svd соответствует одной из проекций в 3d:

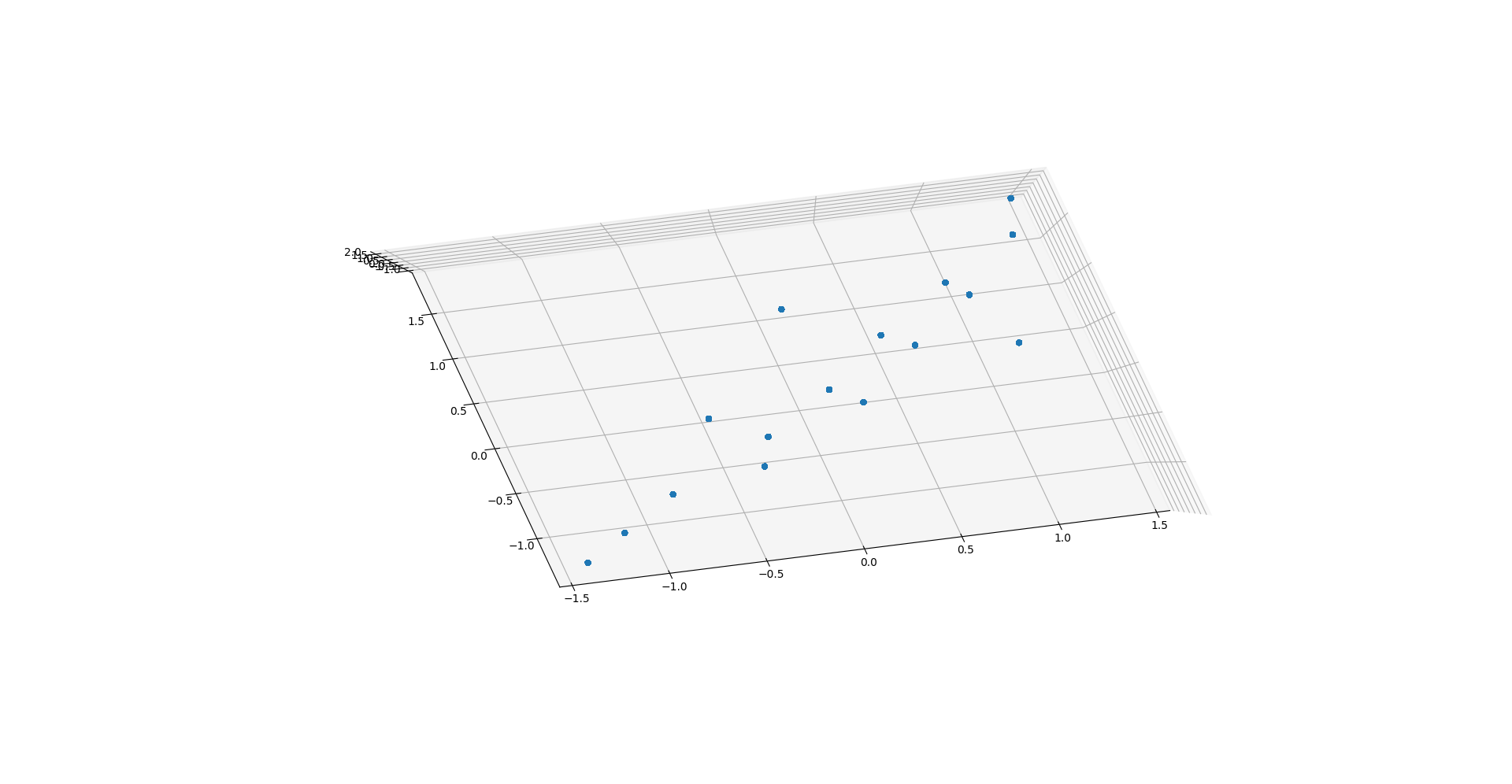


Рисунок 4.4.1. Проекция в 3d пространстве, соответствующая спроецированному на плоскость изображению компонентов