In [1]:

```
import os
import numpy as np
from sklearn.decomposition import PCA
import scipy as sp

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

sns.set(font_scale=1.3)
```

Снижение размерности фотографий лиц

Загрузим датасет человеческих лиц. Выборка состоит из 280 изображений лиц, каждое имеет размер 32x32. Каждый пиксель принимает значения от 0 до 255. Для того, чтобы работать с изображением как с вектором, матрица пикселей растянута в вектор размерности $32 * 32 = 1024$.

In [2]:

```
X = np.loadtxt('./train_faces.npy')
print(X)
print(X.shape)
```

```
[[ 86. 115. 145. ... 114.  19.  28.]
 [112. 136. 154. ...  47.  48.  49.]
 [ 54.  51.  55. ...  88. 101. 115.]
 ...
 [136. 147. 146. ...  71.  72.  73.]
 [101. 133. 145. ...  73.  72.  70.]
 [164. 168. 171. ...  73.  73.  75.]]
(280, 1024)
```

Посмотрим, как вообще выглядят эти лица.

In [3]:

```
plt.figure(figsize=(12, 5))
for i in range(40):
    plt.subplot(4, 10, i + 1)
    plt.imshow(X[i].reshape((32, 32)), cmap='gray')
    plt.axis('off')
```

Выборку лиц можно визуализировать с помощью PCA.

In [4]:

```
pca = PCA(n_components=2)
transform_X = pca.fit_transform(X)

plt.figure(figsize=(10, 6))
plt.scatter(transform_X[:, 0], transform_X[:, 1],
            alpha=0.7, color='#0099CC')
plt.xlabel('Проекция на первую главную компоненту')
plt.ylabel('Проекция на вторую главную компоненту')
plt.show()
```

Посчитаем сингулярные числа по лицам

In [5]:

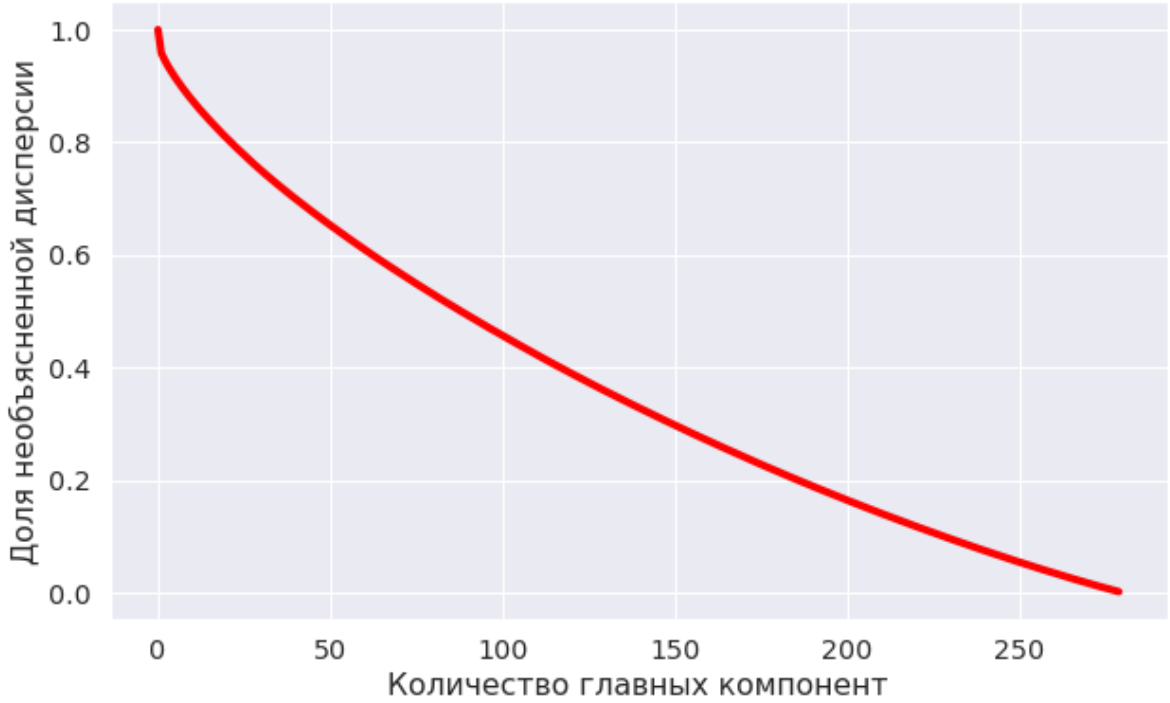
```
singular_values = np.sqrt(sp.linalg.svdvals(X))
print(singular_values)
```

```
[270.02107555  81.76404121  71.92180336  63.85645457  60.47495063
  56.97681205  54.38612376  52.63991889  51.29444879  48.9306597
  48.05430541  46.27031825  44.57307096  44.23762041  43.12249147
  42.43873139  42.1382968   41.79402284  39.90162559  39.59301688
  39.38842827  38.59158176  37.78522122  37.44580732  37.33453722
  36.83974898  36.51562856  35.87708042  35.53168822  34.6576802
  34.19263315  33.77840437  33.20729287  33.05394887  32.67148983
  32.41788508  31.93664462  31.67958577  31.46609998  31.01900346
  30.69365023  30.66897852  30.46077067  30.20090206  30.13833702
  29.79712323  29.61987132  29.27109293  29.17493082  28.9992181
  28.7627508   28.62864934  28.38232255  28.25700151  28.02453193
  27.94107503  27.73590765  27.59999604  27.52359993  27.32994225
  27.13916253  26.95289744  26.72215879  26.62105914  26.46766421
  26.2913381   26.17518663  26.00199557  25.84704212  25.76791375
  25.70812539  25.45996892  25.31691508  25.24912485  25.19646767
  25.00534229  24.85640855  24.83896682  24.64249614  24.56414164
  24.47280728  24.41962978  24.34821291  24.15778084  24.03080457
  23.84928782  23.69808541  23.6293131   23.59098986  23.4852478
  23.4408856   23.35349248  23.20045468  23.05306964  22.94416266
  22.90375817  22.81158718  22.76956188  22.69289214  22.60769024
  22.54380341  22.48752386  22.32848943  22.2861731   22.15789197
  22.14893996  21.98355145  21.90128055  21.84544926  21.76420497
  21.66003015  21.61604209  21.5210829   21.41314642  21.28241697
  21.2054897   21.11028013  21.06774313  20.97992188  20.87534205
  20.85976762  20.80140669  20.6956908   20.6618699   20.48056267
  20.37674585  20.28823904  20.25896648  20.15973205  20.06071432
  19.97629038  19.92172137  19.83896061  19.79382286  19.7311388
  19.6886491   19.6605601   19.50894729  19.50588726  19.38750488
  19.36804597  19.34720124  19.20023059  19.14834498  19.00735163
  18.97833084  18.93115596  18.88889882  18.85983805  18.7563858
  18.69271255  18.67089849  18.53545864  18.41923509  18.38942134
  18.3488957   18.30516501  18.26677291  18.20629175  18.14573928
  18.09957528  17.9574528   17.9101938   17.86082216  17.82776219
  17.7385843   17.72969643  17.71942635  17.55668775  17.54136675
  17.49404124  17.41291085  17.36867276  17.30462454  17.28258828
  17.17000638  17.0565429   16.99419551  16.99241342  16.92401049
  16.88959328  16.82142117  16.72606701  16.67258113  16.63759309
  16.53890437  16.40219282  16.32458001  16.2728625   16.24970158
  16.23309236  16.1456619   16.102075    16.06392053  15.95248308
  15.92526803  15.85163714  15.78171669  15.67209933  15.65540497
  15.58369626  15.55290412  15.44187549  15.42826616  15.36529682
  15.29902288  15.24978733  15.18313931  15.16320356  15.09031943
  14.99448913  14.95556488  14.92056057  14.85448092  14.82609873
  14.78792167  14.70628931  14.6099832   14.59102608  14.48292565
  14.42867953  14.41468338  14.34769791  14.28004293  14.19162765
  14.15123176  14.12999956  14.0775557   14.00213125  13.94084716
  13.90444518  13.87230485  13.79066534  13.7430407   13.67003859
  13.63096137  13.55317049  13.48662465  13.46760255  13.3609163
  13.34543324  13.26104245  13.20773782  13.14658699  13.10405195
  13.02798595  12.98950446  12.89935626  12.83812194  12.82984375
  12.69807944  12.65951031  12.5521212   12.43909152  12.40902352
  12.35539372  12.34738151  12.26745813  12.22799299  12.19081654
  12.10908738  12.00557753  11.90748493  11.86265895  11.80937298
  11.73196238  11.69583804  11.56875701  11.52345956  11.36234589
  11.27862312  11.23608669  11.17752869  11.11712979  10.98775459
  10.71058556  10.62431901  10.52388164  10.5030034   10.39199224]
```

По сингулярным числам можно построить график доли необясненной дисперсии.

In [6]:

```
error = singular_values[::-1].cumsum() / singular_values.sum()
error = error[::-1]

plt.figure(figsize=(10, 6))
plt.plot(range(len(error)), error, c='red', linewidth=4)
plt.xlabel('Количество главных компонент')
plt.ylabel('Доля необъясненной дисперсии')
plt.show()
```



Теперь посмотрим на то, что такое главные компоненты

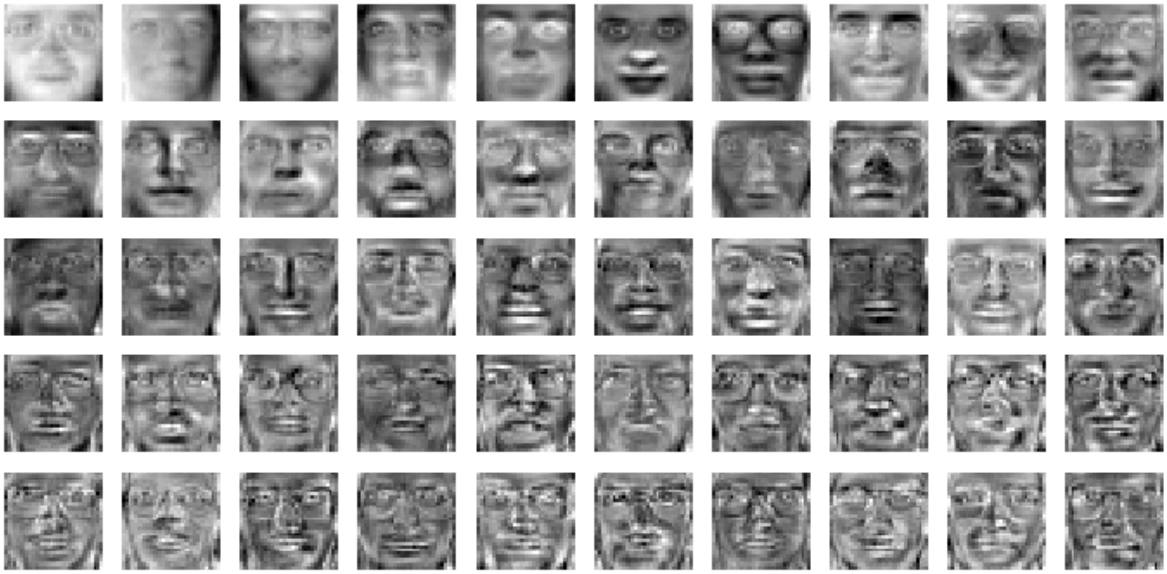
In [7]:

```
pca = PCA(n_components=50)
transform_X = pca.fit_transform(X)
```

Ниже на первой строке изображено среднее лицо, которое получается путем усреднения всех лиц по выборке. Далее в строчках изображены главные компоненты в порядке убывания их степени важности.

In [8]:

```
plt.figure(figsize=(1, 1))
plt.imshow(pca.mean_.reshape((32, 32)), cmap='gray')
plt.axis('off')
plt.show()

plt.figure(figsize=(12, 6))
for i in range(50):
    plt.subplot(5, 10, i + 1)
    plt.imshow(pca.components_[i].reshape((32, 32)), cmap='gray')
    plt.axis('off')
```



Посмотрим на то, как восстанавливаются лица по главным компонентам. Для наглядности возьмем первые 10 компонент.

In [9]:

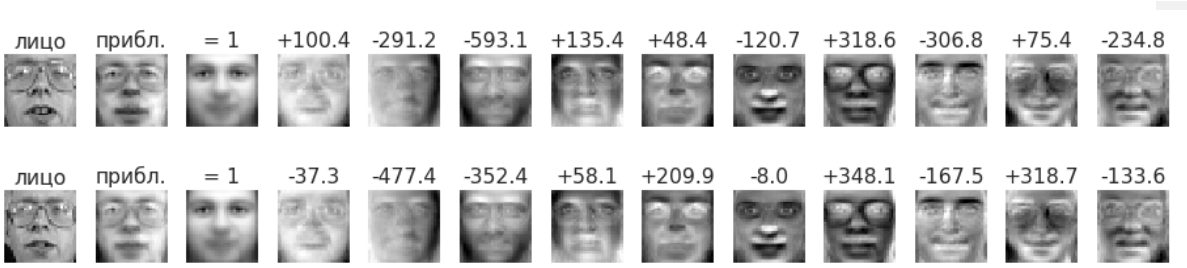
```
pca = PCA(n_components=10)
transform_X = pca.fit_transform(X)
```

Ниже в первой колонке изображено исходное лицо, во второй его приближение (проекция на первые 10 главных компонент). В третьей колонке среднее лицо, и в следующих 10 колонках первые 10 главных компонент. Над компонентами указаны коэффициенты, с которыми они входят в приближаемое лицо.

Стоит отметить, что все компоненты нормированны, однако при отрисовке они растягиваются до диаппазона [0, 255]. Поэтому коэффициенты относятся не к изображениям компонент, а к самим нормированным компонентам.

In [10]:

```
for i in range(10):
    X_hat = pca.inverse_transform(transform_X[i])
    plt.figure(figsize=(16, 1))

    plt.subplot(1, 13, 1)
    plt.imshow(X[i].reshape((32, 32)), cmap='gray')
    plt.axis('off')
    plt.title('лицо')

    plt.subplot(1, 13, 2)
    plt.imshow(X_hat.reshape((32, 32)), cmap='gray')
    plt.axis('off')
    plt.title('прибл.')

    plt.subplot(1, 13, 3)
    plt.imshow(pca.mean_.reshape((32, 32)), cmap='gray')
    plt.axis('off')
    plt.title('= 1')

    for j in range(10):
        plt.subplot(1, 13, 4 + j)
        plt.imshow(pca.components_[j].reshape((32, 32)), cmap='gray')
        plt.axis('off')
        plt.title('{}{:.1f}'.format('+' if transform_X[i, j] >= 0 else '',
                                    transform_X[i, j]))

    plt.show()
```



Прикладная статистика и анализ данных, 2019

Никита Волков

https://mipt-stats.gitlab.io/ (https://mipt-stats.gitlab.io/)