```python
#
# svd_image.py
#

import matplotlib.pyplot as plt

import numpy as np
import numpy.linalg as LA
import argparse
import  scipy.misc   as misc
from sklearn import decomposition
import time



def reconstr_matrix(U,D,V,k):
    rec_mat=np.dot(U[:,:k],np.dot(D[:k,:k],V[:k,:]))
    return rec_mat


def rgb2gray(rgb):

    if(len(rgb.shape)>2):
        r, g, b = rgb[:,:,0], rgb[:,:,1], rgb[:,:,2]
        L = 0.2989 * r + 0.5870 * g + 0.1140 * b
    else:
        L = rgb
    return L

def ParseArguments():
    parser = argparse.ArgumentParser(description="Project ")
    parser.add_argument('--image', default="", required=True, help='Color image  (default: %(default)s)')
    parser.add_argument('--alg', default="pca", required=False, help='pca, nmf  (default: %(default)s)')
    args = parser.parse_args()
    return args.image,args.alg


image_file, alg  =  ParseArguments()

#image = misc.imread(image_file ,mode="RGB")

image = plt.imread(image_file)

M=rgb2gray(image)



#glowna czesc programu: wykonaj svd


print("Calculating SVD ...", end="", flush=True)
start_time = time.time()
U,d,V=LA.svd(M)
print("\t\t took %s seconds " % round((time.time() - start_time),5))




h=M.shape[0]
w=M.shape[1]

#d jest wektorem, zamieniamy w macierz diagonalna
D=np.diag(d)

ile=4;
nr=1;

how_much_rec=1;

rr=reconstr_matrix(U,D,V,how_much_rec)

error_x = []
error_y = []
eigen_sum = []
```

```
for i in range(0,ile):
    for j in range(0, ile):
        fig1=plt.figure(1);
        plt.subplot(ile,ile,nr);

        #zrekonstruuj macierz uzywajac how_much_rec wektorow wlasnych
        rr=reconstr_matrix(U,D,V,how_much_rec)
        #liczymy bledy
        error_x.append(how_much_rec)
        err = ((M-rr)**2).sum();
        error_y.append(err)
        eigen_sum.append((d[how_much_rec:len(d)]**2).sum())
        #rysuj rr
        plt.imshow(rr, cmap=plt.get_cmap('gray'))
        plt.title("ile = " + str(how_much_rec))
        plt.axis('off')
        how_much_rec+=2;

        nr=nr+1
fig1.suptitle("PCA reconstructed")


nr=1;
ile=4
how_much_rec=0
for i in range(0,ile):
    for j in range(0, ile):
        how_much_rec+=1;
        fig2=plt.figure(2);
        plt.subplot(ile,ile,nr);
        k=how_much_rec
        rr_eigenimage = np.dot(U[:,k:(k+1)],V[k:(k+1),:])
        plt.imshow(rr_eigenimage, cmap=plt.get_cmap('gray'))
        plt.title("i = " + str(how_much_rec))
        plt.axis('off')
        nr=nr+1

fig2.suptitle("PCA eigenimages")




#osobny rys. z bledami
plt.figure(3)
plt.plot(error_x,np.zeros(len(error_x)), color='red');
plt.plot(error_x,error_y, color='blue')

plt.plot(error_x,np.zeros(len(error_x)), color='red');
plt.plot(error_x,eigen_sum, color='green')


plt.figure(4)
plt.imshow(M, cmap=plt.get_cmap('gray'))
plt.title("Original (grayscale)")




#NMF:

# ~ plt.figure(5)
# ~ plt.title("NMF")
# ~ M_recons=np.dot(W,H)
# ~ plt.imshow(M_recons, cmap=plt.get_cmap('gray'))


if(alg=="nmf" or alg=="NMF"):

    #p_red=min(int(h/2),int(w/2))
    p_red=min(int(h),int(w))

    print("Calculating NMF...", end="", flush=True)
    start_time = time.time()
    model = decomposition.NMF(n_components=p_red, init='random', random_state=0)
    W=model.fit_transform(M)
    H=model.components_
    print("\t\t took %s seconds " % round((time.time() - start_time),5))


    nr=1;
```

```
    ile=4
    how_much_rec=0
    for i in range(0,ile):
        for j in range(0, ile):
            how_much_rec+=8;
            fig5=plt.figure(5);
            plt.subplot(ile,ile,nr);
            k=how_much_rec
            rr_eigenimage = np.dot(U[:,k:(k+1)],V[k:(k+1),:])
            M_recons=np.dot(W[:,0:k],H[0:k,:])
            plt.imshow(M_recons, cmap=plt.get_cmap('gray'))
            plt.title("i = " + str(how_much_rec))
            plt.axis('off')
            nr=nr+1

    fig5.suptitle("NMF reconstructed")


    nr=1;
    ile=4
    how_much_rec=0
    for i in range(0,ile):
        for j in range(0, ile):
            how_much_rec+=1;
            fig6=plt.figure(6);
            plt.subplot(ile,ile,nr);
            k=how_much_rec
            rr_eigenimage = np.dot(U[:,k:(k+1)],V[k:(k+1),:])
            M_recons=np.dot(W[:,(k-1):k],H[(k-1):k,:])
            plt.imshow(M_recons, cmap=plt.get_cmap('gray'))
            plt.title("i = " + str(how_much_rec))
            plt.axis('off')
            nr=nr+1

    fig6.suptitle("NMF eigenimages")




plt.show()

```