Department of Physics
Umeå University
September 27, 2018

Matrix computations and applications
Assignment 2

Gustav Nystedt - oi14gnt

Department of computer science
Supervisors: Niclas Börlin & Andrii Dmytryshyn

Contents

1 Introduction

In this assignment we look into how we can transform and decompose matrices in different ways. The first part is focused on theory and concepts that are crucial for what we later do in later parts. The second part is focused on conducting an algorithm for QR-factorisation and in the third/last part we implement an application of singular value decomposition.

2 Theory

2.1 Basis of row- and column space

We want to determine the row- and column space of

$$A = \begin{pmatrix} 1 & 2 & 2 & 1 \\ 4 & 8 & 5 & 5 \\ 2 & 4 & 1 & 3 \end{pmatrix}.$$

To do this, we transform A into reduced row echelon form thhrough the following steps:

$$\begin{pmatrix} 1 & 2 & 2 & 1 \\ 4 & 8 & 5 & 5 \\ 2 & 4 & 1 & 3 \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} 1 & 2 & 2 & 1 \\ 0 & 0 & -3 & 1 \\ 0 & 0 & -3 & 1 \end{pmatrix} \Rightarrow$$

$$\Rightarrow \begin{pmatrix} 1 & 2 & 2 & 1 \\ 0 & 0 & -3 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} 1 & 2 & 0 & \frac{5}{3} \\ 0 & 0 & 1 & -\frac{1}{3} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Looking at the pivot columns of the reduced row echelon form of A given in Equation 2.1, we see that the column space of A is

$$\left\{ \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix}, \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix} \right\}. \tag{1}$$

The rows of the reduced row echelon form of A indicates that the basis of the row space of A is

$$\left\{ \left[1, 2, 0, \tfrac{5}{3}\right], \left[0, 0, 1, -\tfrac{1}{3}\right] \right\}. \tag{2}$$

2.2 Orthogonal matrices

(a) We want to show that the vectors x and Qx have the same length, where $x \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n \times n}$ is orthogonal. This is done by showing that $\|x\|_2 = \|Qx\|_2$, as follows:

$$||Qx||^2 = (Qx)^T Qx = x^T Q^T Qx = x^T x = ||x||^2 \tag{3}$$

$$\Rightarrow \quad ||Qx|| = ||x|| \quad \underline{Q.E.D.} \tag{4}$$

(b) If $\{v_1, \dots v_n\}$ is an orthonormal basis of $\mathbb{R}^n$ and $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix, we know that

$$v_i{}^T v_j = \begin{Bmatrix} 0 \text{ for } i \neq j \\ 1 \text{ for } i = j \end{Bmatrix} \quad \text{and} \quad Q^T Q = I. \tag{5}$$

We want to show that this implies that $\{Qv_1, \dots Qv_n\}$ also is an orthonormal basis of $\mathbb{R}^n$. We check that

$$Qv_i \cdot Qv_j = (Qv_i)^T Qv_j = v_i{}^T Q^T Q v_j = v_i{}^T v_j = \begin{Bmatrix} 0 \text{ for } i \neq j \\ 1 \text{ for } i = j \end{Bmatrix}. \tag{6}$$

This implies that the vectors $Qv_1, \dots Qv_n$ are orthonormal and linearly independent, which is enough to prove what we wanted.

2.3 Singular Value Decomposition

If we have found the singular value decomposition components $A = U\Sigma V^T$ as:

$$\begin{pmatrix} u_1, \dots, u_r, u_{r+1}, \dots, u_m \end{pmatrix} \begin{pmatrix} \sigma_1 & & & \\ & \ddots & & 0_{r\times(m-r)} \\ & & \sigma_r & \\ & 0_{(n-r)\times r} & & 0_{(n-r)\times(m-r)} \end{pmatrix} \begin{pmatrix} v_1 \\ \vdots \\ u_r \\ u_{r+1} \\ \vdots \\ u_m \end{pmatrix}$$

we can specify the rank, r, as the *length* of the square diagonal matrix Σ. Further, the orthonormal basis of the fundamental subspaces of A can be found as:

- $\mathcal{C}(A) = \text{span}(u_1, u_2, \dots, u_r)$
- $\mathcal{N}(A^T) = \text{span}(u_{r+1}, u_{r+2}, \dots, u_m)$
- $\mathcal{C}(A^T) = \text{span}(v_1, v_2, \dots, v_r)$
- $\mathcal{N}(A) = \text{span}(v_{r+1}, v_{r+2}, \dots, v_m)$.

2.4 SVD, part 2

We want to express A^T, $A^T A$, and AA^T in terms of U, Σ, and V. This is done as follows:

$$\begin{aligned} A^T &= (U\Sigma V^T)^T = (V^T)^T (U\Sigma)^T = \underline{V\Sigma^T U^T} \\ A^T A &= (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma^T U^T (U\Sigma V^T) \\ &= V\Sigma^T (U^T U)\Sigma V^T = \underline{V\Sigma^T \Sigma V^T} \\ AA^T &= (U\Sigma V^T)(U\Sigma V^T)^T = (U\Sigma V^T) V\Sigma^T U^T \\ &= U\Sigma(v^T V)\Sigma^T U^T = \underline{U\Sigma\Sigma^T U^T}. \end{aligned}$$

2.5 Normal equations

The normal equations for the linear least squares problem, $\min_x \|Ax - b\|_2$, where $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m\times n}$ has full rank and $m > n$, is:

$$A^T Ax = A^T b. \tag{7}$$

We want to show that the vector x from the solution of the linear system

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

is a solution of the linear least square problem, and thus Equation 7 .
We subtract the first row times A^T from the second row:

$$\begin{pmatrix} I & A \\ A^T - A^T I & 0 - A^T A \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 - A^T b \end{pmatrix}$$

which gives us

$$\begin{pmatrix} I & A \\ 0 & -A^T A \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ -A^T b \end{pmatrix}$$

$$\Rightarrow \quad A^T Ax = A^T b, \quad \underline{Q.E.D.}$$

2.6 QR-factorisation

We want to solve the linear least square problem by using the QR-factorisation of A. We have:

$$A = QR = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix} \begin{pmatrix} 4 & 6 & 8 \\ 0 & 2 & 2 \\ 0 & 0 & 4 \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} 10 \\ -2 \\ 1 \\ -3 \end{pmatrix}.$$

As stated in section 7, the normal equations of the linear least square problem is $A^T Ax = A^T b$. Using the QR-factorisation of A, we can thus rewrite this as:

$$A^T Ax = (QR)^T (QR)x = R^T Q^T QRx = R^T Rx$$

$$A^T b = (QR)^T b = R^T Q^T b$$

Hence,

$$A^T Ax = A^T b \quad \Leftrightarrow \quad R^T Rx = R^T Q^T b$$

$$\Rightarrow \quad x = (R^T R)^{-1} R^T Q^T b = R^{-1} (R^T)^{-1} R^T Q^T b = \underline{R^{-1} Q^T b}$$

Using Matlab, we compute this and get:

$$x = R^{-1} Q^T b = \begin{bmatrix} -4.75 \\ 5 \\ -1 \end{bmatrix}.$$

3 Algorithms

3.1 QR factorisation

We want to construct a function `qr_fac(A)` that performs QR-factorisation of an arbitrary matrix A $\in \mathbb{R}^{m\times n}$, using *householder matrix*.

Algorithm:

Pre-define: $N = min(m,n)$ and $M = max(m,n)$

for k = 1:1:N

1. Define y = $A_{k:N,k}$ and construct the $(N-k+1)$-vector v_k:

$$w = y + sign(y_1)||y||e_1, \quad v_k = \frac{1}{||w||}w, \quad \beta = 2||v_k||^2$$

2. Multiply $A_{k:m,k:n}$ with reflector $H_k = I - \beta v_k {v_k}^T$:

$$A_{k:m,k:n} := H_k A_{k:m,k:n}$$

3. Set $Q_k = I^{M\times M}$, and overwrite $Q_{k_{k:M,k:M}}$ with H_k.

end for

Ultimately: $Q = Q_1 Q_2 \dots Q_M$ and $R = A$

Note: The algorithm will continuously overwrite A such that $R = A$.

In our implentation of this algorithm, we use Matlab. In doing so, we are able to use the built-in function `gallery` to perform step 1, i.e. finding v and β.

3.1.1 Tests

In order to test our algorithm, we use Matlab to construct four different matrices $A_1, \dots, A_4$ of different sizes. We then use our function `qr_fac(A)` to get Q and R for each case. To test the accuracy of our implementation, we then compute the relative error

$$\epsilon = \frac{\|A - QR\|_2}{\|A\|_2} \tag{8}$$

for each case. If our algorithm works well, we expect this to be small.
When performing the test, we receive the following result:

$$\epsilon_1 = 0.25 \cdot 10^{-15}$$
$$\epsilon_2 = 0.18 \cdot 10^{-15}$$
$$\epsilon_3 = 0.79 \cdot 10^{-15}$$
$$\epsilon_4 = 0.45 \cdot 10^{-15}$$

It is clear that these are all of magnitude 10^{-15}, which is a satisfying result.

4 Applications

4.1 Compression of training data

To illustrate the decay of the singular values for the digits, we plot using `semilogy`. In Figure 1 and 2, the singular values are plotted for digits 1 and 5.

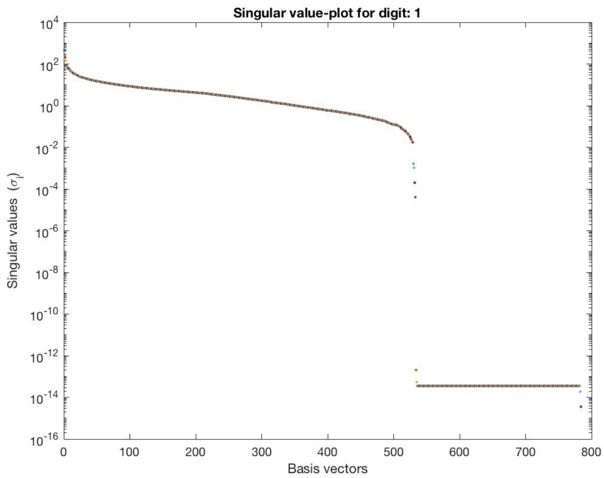

Figure 1: Singular values for digit = 1, plotted using logarithmic scale on y-axis.

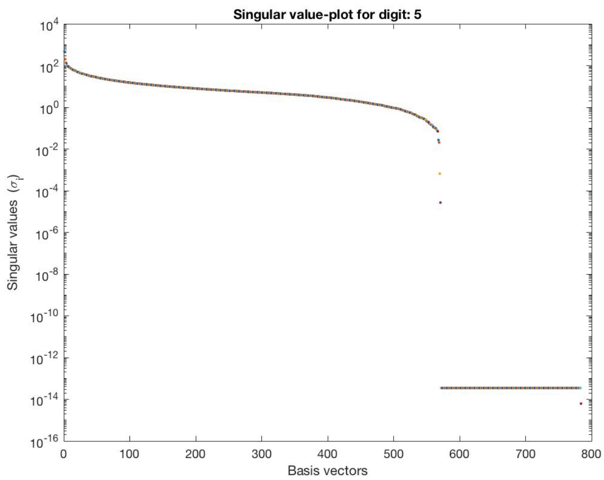

Figure 2: Singular values for digit = 5, plotted using logarithmic scale on y-axis.

It is visible that the decay is rapid in the very beginning and then reaches a plateau, until it ultimately drops ones again.

To speed up the computation and to remove noise in the images, it is desirable to neglect low singular values. To do this, we want to limit our Σ by only using its first $r \times r$-block, leaving us with Σ_r. To choose which value $r = r_k$ that is reasonable to use, we define the error as the spectral norm

$$\epsilon_r = \frac{\|A_k - A_{k,r}\|_2}{\|A_k\|_2} = \frac{\sigma_{r+1}}{\sigma_1}.$$

The r_k that we should use is then the smallest one that still makes $\epsilon_r \leq 0.12$. Hence, we vary r starting at 784 and step-wise decreasing it until the spectral norm is bigger than 0.12. This was done for each digit, and resulted in the values given in Table 1.

Table 1: r_k for each digit 0 to 9.

Digit:	r_k
0	10
1	9
2	15
3	12
4	15
5	16
6	12
7	12
8	13
9	14

From Table 1 it is clear that digit 1 needs the least singular values while digit 5 needs the most.

4.2 Classification

The main goal of this assignment is to learn the program to classify "never seen" digits, from comparison with digits depicted in the training images. We train the model by performing singular value decomposition on the training set for each digit. The $r \times r$ singular value-matrix described in Section 4.1 is then used to choose the number of columns that should be included in U_k (k:th digit). To further classify the previously unseen digit q, we compare it to U_k for each digit by evaluating the approximation error, defined as:

$$\|q - U_k U_k^T q\|_2. \tag{9}$$

The U_k resulting in the smallest approximation error then represents the digit that the program should classify q as.

To quantify the accuracy of our model, we can apply our classification algorithm to all unseen digits in the test set and define:

$$\text{accuracy} = \frac{\text{number of correctly classified digits}}{\text{total number of test digits}}.$$

To further investigate how the accuracy depends on the how we choose r, we perform the classification for each digit sweeping r∈[1, 2, 4, 8, 16, 32, 64, 128]. The results from this test is presented in Table 2.

Table 2: Accuracy for each digit, using different values of r.

Digit: / **r =**	0	1	2	3	4	5	6	7	8	9
1	0.92	0.94	0.75	0.84	0.80	0.65	0.88	0.82	0.76	0.80
2	0.93	0.98	0.85	0.86	0.80	0.79	0.93	0.85	0.83	0.87
4	0.98	0.99	0.88	0.90	0.88	0.90	0.96	0.89	0.89	0.88
8	0.99	0.99	0.92	0.94	0.94	0.91	0.97	0.92	0.92	0.92
16	0.99	1.00	0.94	0.93	0.97	0.93	0.97	0.93	0.93	0.94
32	0.99	0.99	0.94	0.94	0.97	0.94	0.97	0.94	0.94	0.94
64	0.99	0.99	0.94	0.91	0.96	0.90	0.96	0.92	0.94	0.93
128	0.97	0.98	0.92	0.89	0.94	0.84	0.95	0.88	0.90	0.91

Since the data in Table 2 can be perceived a bit messy, we also calculate the average accuracy over the digits for each r. This is visualised in Figure 3 below.

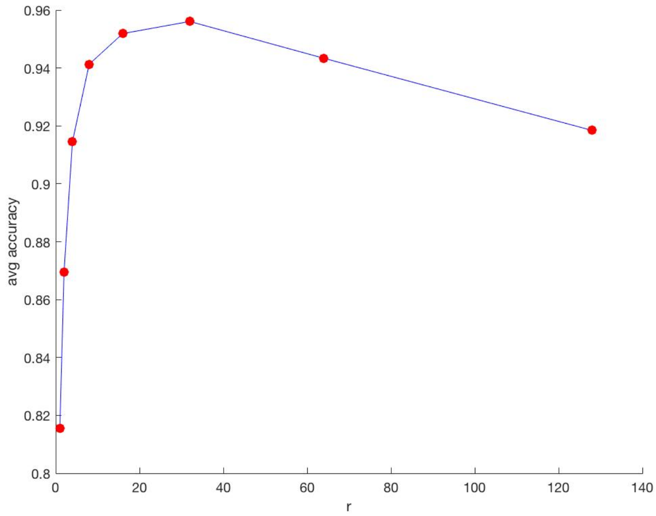

Figure 3: Collision scenario.

As can be seen, the average accuracy over the digits for each r, peaks for r = 32. After this point it stagnates and decays as we use more singular values. This is most likely due to the fact that at a certain point (in this case approximately r = 32), using more data from our U_k will make us include to much "noise" from our training set. This will further lead to the loss of accuracy seen in Figure 3.

To decide which of the digits that is the easiest and hardest to classify, we compute the mean accuracy for each digit over the different r:s. In Figure 4, the mean accuracy is plotted for each digit. It is noticeable that 5 is the digit with lowest mean accuracy, and can thus be considered the hardest to classify, while 1 seems to be the easiest to classify.

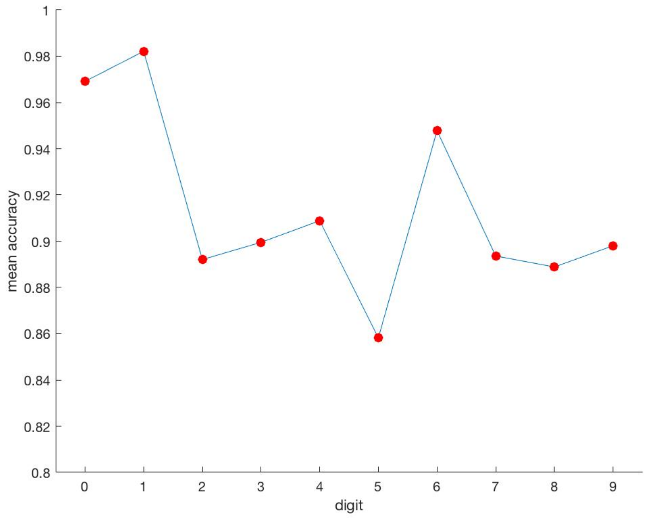

Figure 4: Collision scenario.

We are further interested what digit the model most commonly miss-classify 5 as. To do this, we use r = 32 (since this was the best suited r) and run the classification program ones again, storing all wrongly classified fives to see what they were classified as instead. This showed that the program miss-classified 5 most commonly as 8 (19 misses) followed by 3 (17 misses). Four of the miss-classified digits are illustrated in Figure 5.

Figure 5: The two digits to the left were wrongly classified as eights, and the two digits to the right were wrongly classified as threes.

Looking at Figure 5, it is fair to say that these are examples of digits that could indeed be hard to classify even for a human.

A Matlab Code

A.1 qr_fac.m and test_qr.m

qr_fac.m

```
function [Q,R] = qr_fac(A)
%QR_FAC - Takes matrix A and performs QR-factorisation on it.
    Returns Q & R
%
%   MINIMAL WORKING EXAMPLE: Perform QR-factorisation on a matrix A
    .
%
%   A = [-1 -1 1; 1 3 3; -1 -1 5; 1 3 7]; %define A
%   [Q,R] = qr_fac(A); %perform QR-factorisation on A
%
% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-09-26: Initial version .
%
% Function code starts here...


m = size(A,1); %define the size of A
Q = eye(m); %pre-allocate Q
N = min(size(A)); %define how far the for-loop should loop

for i = 1:N

    %call gallery which returns components for householder matrix
    [v,beta] = gallery('house', A(i:end,i));
    %construct householder matrix from components received above
    H = eye(size(A(i:end,i:end),1))-beta*v*v';
    %overwrite A with householder matrix times A
    A(i:end,i:end) = H*A(i:end,i:end);

    %construct QA as an identity matrix of size m
    QA = eye(m);
    %set elements below current index element equal to householder
    equation
    QA(i:end,i:end) = H;
    %multiply Q with QA. This is to produce Q = Q_n*...*Q_2*Q_1*A.
    Q = Q*QA;

end
%set R equal to the remaining A
R = A;
```

test_qr.m

```
%TEST_QR - Tests qr_fac.m
%
%   MINIMAL WORKING EXAMPLE: ">> test_qr" to see results in
    relevant tests.
%
% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-09-26: Initial version .
%
% Function code starts here...

%Construct 4 different matrices of variable size
A1 = [-1 -1 1; 1 3 3; -1 -1 5; 1 3 7];
A2 = [1 1 1; 1 2 4; 1 3 9; 1 4 16];
A3 = rand(100,120);
A4 = rand(130,100);

%Call qr_fac for each of the matrices
[Q1,R1] = qr_fac(A1);
[Q2,R2] = qr_fac(A2);
[Q3,R3] = qr_fac(A3);
[Q4,R4] = qr_fac(A4);

%Check the relative error of the QR-factorisation for each case
rel_err(1) = norm(A1-Q1*R1)/norm(A1);
rel_err(2) = norm(A2-Q2*R2)/norm(A2);
rel_err(3) = norm(A3-Q3*R3)/norm(A3);
rel_err(4) = norm(A4-Q4*R4)/norm(A4);
rel_err = rel_err';
```

A.2 find_rk.m

find_rk.m

```
function [rk] = find_rk(OCRtrain)
%FIND_RK - Finds smallest r = rk that will still make the spectral norm
%           of OCRtrain-U_r*S_r*V_r' <= 0.12, where where OCRtrain is a set
%           of pictures of hand-drawn digits. OCRtrain = USV' and U_r, S_r
%           and V_r are cropped versions of U,S and V.
%
%   MINIMAL WORKING EXAMPLE: Find smallest possible r =rk that will still
%   make the spectral norm <= 0.12 for training set A.
%
%   rk = find_rk(A);
%
% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-09-26: Initial version .
%
% Function code starts here...

%pre-allocate rk as zero matrix
rk = zeros(length(OCRtrain),1);
%loop over all digits
for j = 1:length(OCRtrain)
    %perform SVD on current digit matrix
    [U,S,V] = svd(OCRtrain{1,j},'econ');
    %store whole A for digit j as USV', for later comparison
    A = U*S*V';
    %store norm of whole A for digit j, for later comparison
    norm_A = norm(A);
    %reset c
    c = 1;
    %reset i
    i = 20;
    %loop while spectral norm is <= 0.12
    while c == 1 && i >= 1
        %decrease number of elements included in U,S and V
        U = U(:,1:i);
        S = S(1:i,1:i);
        V = V(:,1:i);
        %compute spectral norm
        comp = norm(A-U*S*V')/norm_A;
        %check criteria for spectral norm
        if comp > 0.12
            %set rk equal to previous i -> we should take the last i
            %resulting in comp<=0.12
            rk(j) = i+1;
            c = 0;
        end
        i = i-1;
    end
end
```

A.3 train.m

train.m

```
function [U_r] = train(OCRtrain,rk)
%TRAIN Performs Singular value decomposition on training data and crops U
%      at desired/defined rk.
%
%   U_r = train(OCRtrain,rk), where OCRtrain is a set of pictures of
%   hand-drawn digits. Each picture is stored in a column-vector where
%   each element represents a pixel. Train performs Singular value
%   decomposition to OCRtrain and crops the U-matrix from the (rk+1):th
%   element => U_r.
%
%   MINOR WORKING EXAMPLE: Find U_r for training data in A with
%   rk = 15.
%
%   [U_r] = train(A, 15);
%
% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-09-26: Initial version .
%
% Function code starts here...

%pre-allocate U_r = the part of U that should be used when r=rk
U_r = cell(length(OCRtrain),1);

%loop over all digits
for j = 1:length(OCRtrain)
    %get U in SVD for each digit
    [U_r{j},~,~] = svd(OCRtrain{1,j},'econ');
    %limit U by r = rk
    U_r{j} = U_r{j}(:,1:rk(j));

end
```

A.4 classify.m

classify.m

```
function [digit] = classify(q,U_r)
%CLASSIFY - Classify non-seen digit by using information from
    previously
%           seen training data.
%
%   Takes picture of digit q and performs classification with the
%   use of unitary matrix U_r taken from SVD of training set.
    Returns
%   "digit" which is what it classifies q as.
%
%   MINIMAL WORKING EXAMPLE: Classify non-seen picture q with
    information
%   in unitary matrix U_r, gained from using [U_r] = train(OCRtrain
    ,rk).
%
%   digit = classify(q,U_r);
%
% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-09-26: Initial version .
%
% Function code starts here...

%set variable err_min to inf to be sure nothing is initially bigger
     than it
err_min = inf;

%loop over all digits
for i = 1:length(U_r)
    %check approximation error the unseen digit q approximated by
    digit i
    err = norm(q - U_r{i,1}*U_r{i,1}'*q);
    %check if smallest error so far, if true set to new minimum
    error
    if err < err_min
        err_min = err;
        digit = i-1;
    end
end
```

A.5 main_OCR.m

main_OCR.m

```
%MAIN_OCR - Performs tests relevant in Matrix Computations and
%Applications, Assignment 2, by calling numerous functions and
    conducting
%a number of plots.
%
%   MINIMAL WORKING EXAMPEL:
%   ">> main_OCR" to see results in relevant tests.
%
% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-09-26: Initial version .
%
% Function code starts here...

%load train/test-data
load('mnist_all_converted2.mat');

%change data from uint8 to double
for i=1:10
    OCRtrain{i}=im2double(OCRtrain{i});
    OCRtest{i}=im2double(OCRtest{i});
end

%call function for finding r_k for each digit
rk = find_rk(OCRtrain);
%define r sweep values
r_val = [1 2 4 8 16 32 64 128];
%pre-allocate cell array for storing the accuracy
acc_Cell = cell(length(r_val),1);

%loop over all r
for i = 1:8
    %prepare r for calling train.m : must be in vector form
    r = r_val(i)*ones(length(OCRtrain),1);
    %call function train to compress training data
    [U_r] = train(OCRtrain,r);

    %pre-allocate accuracy vector for current r -> will be
    overwritten
    accuracy = zeros(length(OCRtrain),1);
    %loop over all digits
    for num = 0:9
        %reset number of correct classifications
        correct = 0;
        %loop over all images of a digit
        for j = 1:length(OCRtest{1,num+1}(1,:))
            %call classify.m to classify an image of a digit
            digit = classify(OCRtest{1,num+1}(:,j), U_r);

            %check if correctly classified
            if digit == num
                %increase correctly classified digits by one
                correct = correct+1;
            end

        end
        %compute accuracy for the current digit of the current r
        accuracy(num+1) = correct/length(OCRtest{1,num+1}(1,:));
    end
```

```
    %store vector of accuracies for current r
    acc_Cell{i} = accuracy;
end

%calculate average accuracy for each r in r_val over the digits
for i = 1:8
   mean_r(i) = mean(acc_Cell{i,1});
end

%%
%Calculate average accuracy for each digit over r_val
for i = 1:10
    for j = 1:8
        digit(j) = acc_Cell{j,1}(i);
    end
    mean_dig(i) = mean(digit);
end

%visualise the r-dependence of the mean accuracy
figure
hold on
plot(r_val, mean_r)
scatter(r_val, mean_r, 50, 'filled', 'r')
xlabel('r')
ylabel('avg accuracy')

%visualise which digits is easiest/hardest to classify
figure
hold on
plot(0:9,mean_dig)
scatter(0:9,mean_dig,50,'filled','r')
axis([-0.5 9.5 0.8 1])
xlabel('digit')
ylabel('mean accuracy')

%% This part is to look closer at the digit 5 (hardest to classify)
%set r = 32 since this gives highest accuracy
r = 32*ones(length(OCRtrain),1);
%"train" model
[U_r] = train(OCRtrain,r);

%choose digit 5
num = 5;
%reset number of correctly classified digits
correct = 0;
i = 1;
%loop over all images of 5 in the training set
for j = 1:length(OCRtest{1,num+1}(1,:))
    %classify current image
    digit = classify(OCRtest{1,num+1}(:,j), U_r);
    %check if correctly classified
    if digit == num
        %%increase correctly classified
        correct = correct+1;
    else
        %store wrongly classified
        wrong(i) = digit;
        %store index of wrongly classified
        ind_wrong(i) = j;
        i = i + 1;
    end
```

```
end
%visualise random wrongly classified image
im=reshape(OCRtest{num+1}(:,...
              ind_wrong(randsample(length(ind_wrong),1))),[28,28]);
imshow(im,[0,1]);
```