Department of Physics
Umeå University October 15, 2018

Matrix computations and applications

Assignment 3

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 Eigenvalues by hand

$$A = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 5 & 3 \\ 0 & 2 & 4 \end{pmatrix} \quad \text{and} \quad \det(A - \lambda I) = 0$$

$$A - \lambda I = \begin{pmatrix} 2-\lambda & 0 & 0 \\ 0 & 5-\lambda & 3 \\ 0 & 2 & 4-\lambda \end{pmatrix}$$

$$\Rightarrow \quad \det(A - \lambda I) = (2-\lambda)[(5-\lambda)(4-\lambda) - 6] = 0$$

$$\Rightarrow \quad \lambda^2 - 9\lambda + 14 = 0 \quad \Rightarrow \quad \begin{cases} \lambda_1 = 2 \\ \lambda_2 = 7 \\ \lambda_3 = 2 \end{cases}$$

2.2 Eigenvectors

The eigenvectors corresponding to an eigenvalue can be found by computing the basis for $\mathcal{N}(A - \lambda I)$. To find the eigenvectors of A from section 2.1, we start by defining:

$$(A - \lambda_i I)X_i = \begin{pmatrix} 2-\lambda_i & 0 & 0 \\ 0 & 5-\lambda_i & 3 \\ 0 & 2 & 4-\lambda_i \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \mathbf{0}, \quad \begin{cases} \lambda_1 = 2 \\ \lambda_2 = 7 \\ \lambda_3 = 2 \end{cases}$$

We then begin by putting in $\lambda_1 = 2$ and get,

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 3 & 3 \\ 0 & 2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

In this case, we do not get any information about x_1 so we choose to set it equal to zero, $x_1 = 0$. Further we get that

$$y_1 = z_1 \quad \Rightarrow \quad X_1{}' = \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$$

We normalize by dividing with the norm:

$$\|X_1{}'\| = \frac{1}{\sqrt{2}} \quad \Rightarrow \quad X_1 = \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{pmatrix}$$

Using $\lambda_2 = 7$ we get,

$$\begin{pmatrix} -5 & 0 & 0 \\ 0 & -2 & 3 \\ 0 & 2 & -3 \end{pmatrix} \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \quad \Rightarrow \quad x_2 = 0 \text{ and } z_2 = \frac{2}{3} y_2$$

Setting $y_2 = 1$ we get

$$X_2{}' = \begin{pmatrix} 0 \\ 1 \\ \frac{2}{3} \end{pmatrix}, \quad \|X_2{}'\| = \sqrt{\frac{13}{9}} \quad \Rightarrow \quad X_2 = \begin{pmatrix} 0 \\ 0.8321 \\ 0.5547 \end{pmatrix}$$

To get the last eigenvector we use that

$$Ax_3 = \lambda_3 x_3 \quad \Leftrightarrow \quad A \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = 2 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \Rightarrow \quad X_3 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

by inspection.

2.3 Determinant and trace relations

For any triangular matrix, the eigenvalues are equal to its diagonal elements. Further, for any n-by-n matrix it holds that its determinant is equal the product of its eigenvalues. Thus, to get the determinant of C, we onluy need to compute the product of its diagonal elements:

$$det(C) = 1 \cdot 2 \cdot 1 \cdot 4 \cdot 2 = 16.$$

Further, the trace of an n-by-n matrix is equal to the sum of its eigenvalues. Thus, in the case of C we get:

$$trace(C) = 1 + 2 + 1 + 4 + 2 = 10.$$

2.4 Similarity transformation

It holds that

$$B = PAP^{-1} \quad \Leftrightarrow \quad A = P^{-1}BP.$$

We define:

$$Bx = \lambda x \quad \Rightarrow \quad PAP^{-1}x = \lambda x \quad \Rightarrow \quad A(P^{-1}x) = \lambda(P^{-1}x)$$

So, if x is an eigenvector of B with eigenvalue λ, then $P^{-1}x$ is an eigenvector of A with the same eigenvalue. The eigenvector of B is a multiple of the eigenvector of B.

2.5 Diagonalize matrix

Define $\Lambda = S^{-1}AS$ with,

$$A = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 5 & 3 \\ 0 & 2 & 4 \end{pmatrix} \quad \text{and} \quad S = \begin{bmatrix} 0 & 0 & 1 \\ 0.7071 & 0.8321 & 0 \\ -0.7071 & 0.5547 & 0 \end{bmatrix}$$

from section 2.2. We use Matlab to compute $S^{-1}AS$, and get that

$$\Lambda = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

2.6 Matrix exponential

If A is a diagonalizable matrix, we have that

$$e^A = I + A + \frac{1}{2}A^2 + \frac{1}{6}A^3 + \dots$$

$$= I + S\Lambda S^{-1} + \frac{1}{2}(S\Lambda S^{-1})^2 + \frac{1}{6}(S\Lambda S^{-1})^3 + \dots$$

$$= S(I + \Lambda + \frac{1}{2}\Lambda^2 + \frac{1}{6}\Lambda^3 + \dots)S^{-1} = Se^{\Lambda}S^{-1}$$

$$\underline{Q.E.D.}$$

Computing the exponential matrix of A from section 2.1, we get:

$$Se^{\Lambda}S^{-1} = \begin{bmatrix} 7.4 & 0 & 0 \\ 0 & 660.9 & 653.5 \\ 0 & 435.7 & 443.1 \end{bmatrix}.$$

3 Applications

3.1 Rigid body alignment

In this assignment we want to construct a function that performs an optimal alignment of a rigid body P with respect to another rigid body Q. To perform a displacement of a rigid body, we want want to multiply each coordinate-pare (in two dimensions) of P by a rotation matrix R. Also we want to add a a translation vector t to each of these coordinate pairs:

$$Rp_i + t, \quad P = [p_1, \dots, p_n]. \tag{1}$$

In order to align P and Q we want to minimize the diference between the dislocated version of P and the original version of Q. This can be defined as:

$$f(R,t) = \sum_{i=1}^{n} \|(Rp_i + t) - q_i\|_2^2 \tag{2}$$

To optimise this with translation in focus, we compute its derivative with respect to t. After some rewriting, we end up with the expression,

$$f(R) = \sum_{i=1}^{n} \|R(p_i - \bar{p}) - (q_i - \bar{q})\|_2^2, \tag{3}$$

where $\bar{p}$ and $\bar{q}$ are the centres of mass of P and Q respectively. To further simplify, we define the normalised variables $x_i = p_i - \bar{p}$ and $y_i = q_i - \bar{q}$ and get:

$$f(R) = \sum_{i=1}^{n} \|Rx_i - y_i\|_2^2. \tag{4}$$

To find the optimal rotation matrix R we do some manipulation of Equation 4, using properties of trace, the covariance matrix and its singular value decomposition of P and Q etc, and properties of orthogonality. We end up with the expression:

$$R = VU^T, \tag{5}$$

for the R which optimises the alignment of P with respect to Q. However, in order to be sure that this is a *true* rotation and that there is no reflection included, we adjust Equation 5, such that:

$$R = V \begin{pmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & det(VU^T) \end{pmatrix} U^T. \tag{6}$$

This ensures U to be a pure rotation without reflection.

Algorithm:

1. Compute the centre of mass for P and Q:
$$\bar{p} = \frac{1}{n}\sum_{i=1}^{n} p_i \quad \text{and} \quad \bar{q} = \frac{1}{n}\sum_{i=1}^{n} q_i,$$
2. Introduce normalised variables with respect to the centres of mass of P and Q:
$$X = P - \bar{P} \quad \text{and} \quad Y = Q - \bar{Q}$$
3. Set up covariance matrix:
$$S = XY^T$$
4. Perform singular value decomposition on the covariance matrix:
$$U\Sigma V^T = S$$
5. Define an identity matrix of size $n \times n$ and set its last diagonal element to the determinant of VU^T:
$$m = I \in \mathbb{R}^{n\times n}, \quad m_{n,n} = det(VU^T)$$
6. Define R and t:
$$R = VmU^T \quad \& \quad t = \bar{q} - R\bar{p}$$

3.1.1 Test

To test `rigidalign.m`, we run the test-script `rigidtest.m` which initially takes two rigid bodies Q and P, and performs an alignment on Q with respect to P. The result is visualised in Figure 1. The green house shaped body is Q, the blue house shaped body is P and the optimal alignment performed on Q produces the red house shaped body.

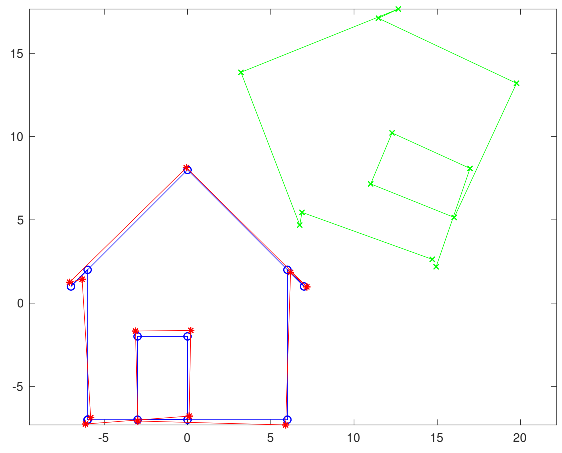

Figure 1: Result from running the test script for `rigidalign.m`. It starts with two house shaped bodies Q (green) and P (blue), and performs an alignment on Q such that it results in PA (red).

3.2 Pagerank

3.2.1 Theory

In this part we want to implement a pagerank algorithm using the "Google matrix" founded in the 90's. It builds on the idea that for each page in the link structure of the web, we can assign a score. This score then represents the importance of each page, and pages that links to important pages are themselves considered important. We let the column vector π denote the PageRank score of the pages, and define $\mathcal{P}_j$ as the set of pages to which page i links to. We set this up as:

$$\pi_i = \sum_{j|i\in\mathcal{P}_j} \frac{\pi_j}{|\mathcal{P}_j|}. \tag{7}$$

In order to express this in a more compressed, linear algebra-friendly way, we define H with $h_{ij} = 1/|\mathcal{P}_i|$, giving us:

$$\pi^T = \pi^T H, \tag{8}$$

This model is based on the concept of a random surfer, clicking around aimlessly page to page on the web. To avoid the case when this random surfer arrives at a page with no out-links, i.e. h_{ij} is all zeros, we define:

$$S = H + ae^T/n, \tag{9}$$

where e is a vector of all ones, and $a_i = 1$ if page i has no outgoing links and zero otherwise. Further, we need to consider that the link graph can have subgraphs, from which the random surfer cannot escape. This problem is solved by introducing:

$$G = \alpha S + (1-\alpha)ee^T/n, \quad 0 < \alpha < 1. \tag{10}$$

This is the so called Google matrix, and by defining it this way we add an element of "boredom" in the random surfer. Simply speaking, there is always a slight chance that the surfer will get bored and re-start its clicking session at a random page of the web. In this way, we add the possibility of "escaping" self-referencing subgraphs. The final expression for performing PageRank is then:

$$\pi^T = \pi^T G. \tag{11}$$

3.2.2 Implementation

To implement this in matlab, we want to create a model that will iteratively update itself such that it converges towards some relevant PageRank vector π. To do this, we start by creating a PageRank vector with randomly generated elements summing to 1, i.e. $\Sigma_i^n \pi_i = 1$. We further employ the power method as

$$\pi^{(k+1)T} = \pi^{(k)T} G, \tag{12}$$

which enables us to update *pi* iteration by iteration. In order to find a decide when the algorithm has converged desirably, we define

$$\|\pi^{(k+1)} - \pi^{(k)}\|_2 < \varepsilon, \tag{13}$$

which denotes the difference of π from the last iteration. This gives us a way of deciding when to terminate the algorithm. In the assignment we receive an H^T on which we want to perform a PageRank using a function `pagerank(HT,alpha,eps)`. To do this, we must first consider that Equation 7 to 11 are made for H, which means we must adjust them for using H^T. We end up with the following expression instead of Equation 12:

$$\begin{aligned}\pi^{(k+1)} &= G^T\pi^{(k)} = \left[\alpha S + (1-\alpha)ee^T/n\right]^T \pi^{(k)} \\ &= [\alpha(H^T + ea^T/n) + (1-\alpha)ee^T/n]\pi^{(k)} \end{aligned} \tag{14}$$

In order to speed up the computation in Matlab it is also important for us to consider in which order we multiply our matrices. For this purpose, we choose to rewrite Equation 14 as:

$$\pi^{(k+1)} = \alpha H^T \pi^{(k)} + \frac{\alpha e}{n}(a^T \pi^{(k)}) + \frac{(1-\alpha)e}{n}(e^T \pi^{(k)}), \tag{15}$$

which is what we use in our algorithm. Finally, to speed up the convergence of our algorithm, we add one last peace,

$$\beta = \|\pi^{(k)}\|_1 - \|\pi^{(k+1)}\|_1 \tag{16}$$

$$\pi^{(k+1)} = \pi^{(k+1)} + \frac{\beta}{n}. \tag{17}$$

Algorithm:

```
Input: [$H^T \in \mathbb{R}^{n \times n}$, $\alpha$, $\epsilon$]

  1. Pre-define initial $\pi$ with randomly generated elements summing to 1:
         $\pi = \text{randomized vector of length n}$
         $\sum_{i=1}^{n} \pi_i = 1.$

  2. Define a vector $a$ based on $H^T$, where each element $a_i$ corresponding to
     an empty column of $H^T$ is set equal to 1, and all other elements are set
     equal to zero:
         $a_i = \begin{cases} 1 \text{ if } h_{i*}^T = 0 \\ 0, \text{ else} \end{cases}$

  3. Define $e$ as a vector of length n with all elements equal to one:
         $e = \text{ones}(n, 1)$

  4. WHILE $\|\pi^{(k+1)} - \pi^{(k)}\|_2 > \varepsilon$

     4.1. Compute $\pi^{(k+1)}$ using Equation 15:
              $\pi^{(k+1)} = \alpha H^T \pi^{(k)} + \frac{\alpha e}{n}(a^T \pi^{(k)}) + \frac{(1-\alpha)e}{n}(e^T \pi^{(k)})$

     4.2. Conduct $\beta$ using Equation 16, and then apply it on $\pi^{(k+1)}$:
              $\beta = \|\pi^{(k)}\|_1 - \|\pi^{(k+1)}\|_1$
              $\pi^{(k+1)} = \pi^{(k+1)} + \frac{\beta}{n}$

     END WHILE
```

To test this algorithm implemented in Matlab, we take the matrix

$$P = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix},$$

and call `[ranks] = pagerank(P',0.9,1e-8)`. The resulting PageRank scores are:

$$\texttt{ranks} = \begin{bmatrix} 0.0372 \\ 0.0540 \\ 0.0415 \\ 0.3751 \\ 0.2060 \\ 0.2862 \end{bmatrix},$$

which is correct, comparing with values in Section 4.5 of Langville and Meyer (2005). To further test if our code is exploiting sparseness enough, we run the following code:

```
rand('seed',28);
n=1e6;
ST=sprand(n,n,1e-8);
tic,pagerank(ST,0.1,1e-8);toc
```

The resulting time spent for our code to perform PageRank on ST, is 0.348547 seconds, which is good enough for this problem.

3.2.3 Scaling function

In order to run more significant tests of our PageRank code, we first need to construct a function which takes a square matrix A with binary elements (zeros or ones) and scales its column. The scaling should be performed in a way that will make the sum of each column equal to 1.

Algorithm:

---

1. Make sure A is a sparse matrix.
2. Divide each column with its own sum in order to normalise them.
3. Return matrix with normalised columns.

3.2.4 Tests and results

Running `testpagerank.m` on the HT received from `http://www.cs.umu.se/kurser/5DA003/HT18/wwwdata.mat`, we can conclude that there are 7 pages with no outgoing links. The page with the most outgoing links was `http://www.teknat.umu.se/om-fakulteten/aktuellt/nyhetsarkiv/index.html` which had 45 outgoing links. The page with most incoming links was `http://www.teknat.umu.se/digitalAssets/2/2902_umu_transparent_16.ico/index.html`, which had 191 incoming links. We perform PageRank on HT using $\alpha = 0.1, 0.2, \ldots, 0.9$ and $\epsilon = a0^{-6}$, and in Table 1 below we present the number of iterations needed for convergence for each value of alpha.

Table 1: Number of iterations needed for convergence, for each value of α.

α:	*iterations*
0.1	5
0.2	6
0.3	7
0.4	7
0.5	9
0.6	10
0.7	11
0.8	13
0.9	15

Further, the top-10 ranked urls based on their PageRank score are present in the list below.

1. `http://www.teknat.umu.se/digitalAssets/2/2902_umu_transparent_16.ico/index.html`,
   rank = 0.0621
2. `http://www.teknat.umu.se/index.html`,
   rank = 0.0382
3. `http://www.teknat.umu.se/utbildning/index.html`,
   rank = 0.0380
4. `http://www.teknat.umu.se/forskning/index.html`,
   rank = 0.0380
5. `http://www.teknat.umu.se/samverkan/index.html`,
   rank = 0.0380
6. `http://www.teknat.umu.se/om-fakulteten/index.html`,
   rank = 0.0380
7. `http://www.teknat.umu.se/student/index.html`,
   rank = 0.0380
8. `http://www.teknat.umu.se/anstalld/index.html`,
   rank = 0.0380

9. `http://www.teknat.umu.se/om-fakulteten/aktuellt/index.html`,
   `rank` = 0.0127

10. `http://www.teknat.umu.se/om-fakulteten/ledning-och-kansli/index.html`,
    `rank` = 0.0127

A Matlab Code

A.1 rigidalign.m

rigidalign.m

```
function [R,t] = rigidalign(P,Q)
%RIGIDALINE - Takes two bodies P and Q. Finds rotation-matrix and
%             translation vector t, for optimal alignment of P with
%             respect to Q.
%
%   MINIMAL WORKING EXAMPLE: We have two 3-point bodies with coordinates
%                           defined by:
%
%           P = [p_x1 p_x2 p_x3; p_y1 p_y2 p_y3]
%           Q = [q_x1 q_x2 q_x3; q_y1 q_y2 q_y3].
%
%           We want to find 2-by-2 matrix R and 2-by-1 matrix t, that
%           optimally align P with respect to Q.
%
%   >> P = [p_x1 p_x2 p_x3; p_y1 p_y2 p_y3]; %define P
%   >> Q = [q_x1 q_x2 q_x3; q_y1 q_y2 q_y3] %define Q
%   >> [R,t] = rigidalign(P,Q) %find R and t by calling rigidalign

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-10-12: Initial version .
%
% Function code starts here...

p_cent = [mean(P(1,:));mean(P(2,:))]; %Compute the center of mass fo body P
q_cent = [mean(Q(1,:));mean(Q(2,:))]; %Compute the center of mass fo body Q

X = P-p_cent; %Introduce new normalized variable X for body P
Y = Q-q_cent; %Introduce new normalized variable X for body Q

S = X*Y'; %Set up covariance matrix
[U, sigma, V] = svd(S); %Perform SVD on the convariance matrix
mid = eye(size(V,1)); %Start by def I-matrix to ensure pure rot matrix R
mid(end,end) = det(V*U');%Set last elem on diag of I-matrix=det(V*U')->pure
R = V*mid*U'; %Define R

t = q_cent - R*p_cent; %Set t
```

A.2 rigidtest.m

rigidtest.m

```
% Load the house coordinates.
P=house;
% Pick an angle between 0..90 degrees
alpha=rand*pi/2
% Create a rotation matrix
R0=[cos(alpha),-sin(alpha);sin(alpha),cos(alpha)];
% Verify that R0 is a rotation matrix.
det(R0) % (should be close to unity)
% Generate a random shift.
m=10+rand(2,1);
% Rotate and shift the house.
PT=R0*P+repmat(m,1,size(P,2));
% Perturb the transformed house with noise.
noiseLevel=0.25;
Q=PT+randn(size(PT))*noiseLevel;
% Plot the original and perturbed house.
plot(P(1,:),P(2,:),'bo-',Q(1,:),Q(2,:),'gx-'); axis equal
% At this point, only P and Q are assumed to be know.
% Now, compute the rigid-body transformation between P and Q...
[R,t]=rigidalign(P,Q);
% Apply the inverse operation to Q to align with P.
PA=R'*(Q-repmat(t,1,size(Q,2)));
% Plot the re-aligned house.
line(PA(1,:),PA(2,:),'marker','*','linestyle','-','color','r');
```

A.3 pagerank.m

pagerank.m

```
function [ranks, i] = pagerank(HT, alpha, eps)
%PAGERANK - Accepts a transposed sparse hyperlink matrix HT and performes
%          PageRank on it. Also takes alpha (in Google matrix) and eps
%          (termination limit).
%
%   MINIMAL WORKING EXAMPLE: We have a sparse hyperlink matrix AT and want
%                            to perform PageRank on it with alpha = 0.8 and
%                            eps = 1e-7.
%
%   >> [ranks, i] = pagerank(AT,0.8,1e-7)
%
%   ==> Score matrix: ranks
%       Number of iterations: i

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-10-12: Initial version .
%
% Function code starts here...

%pre-define initial pi with randomized elements
n = length(HT); %define the length of HT
pi = rand(n,1); %randomly generate a pi
piSum = sum(pi); %compute pi's sum
pi = pi ./ piSum; %all pi's elements by its sum to normalize it-> sum(pi)=1
norm_pi = inf; %set the termination variable to inf

%define a-matrix, holding ones for all-zero columns of HT and zeros else
a = zeros(n,1);
a((sum(HT,1) == 0)) = 1;

%define e-vector of all ones
e = ones(n,1);

%make all ingoing vectors sparse to preserve sparseness of HT
a = sparse(a);
e = sparse(e);
pi = sparse(pi);

% define/set iteration-counter
i = 0;

while norm_pi > eps %check terminating condition

    %save pi^(k)
    pi_prev = pi;

    %calculate pi^(k+1)
    pi = alpha*HT*pi...
        + (alpha/n)*e*(a'*pi)...
        + (1-alpha)/n*e*(e'*pi);

    %calculate beta, made for faster convergence
    beta = norm(pi_prev,1) - norm(pi,1);

    %apply beta on pi^(k+1)
    pi = pi + beta*(1/n);

    %calculate epsilon
    norm_pi = norm(pi - pi_prev);

    %count iteration
    i = i + 1;
end

%set ranks equal to pi for returning final result
ranks = pi;
```

A.4 test_pagerank.m

test_pagerank.m

```
%TEST_PAGERANK - Tests pagerank on example from book and randomised
     ST
%
%   MINIMAL WORKING EXAMPLE: Perform the test
%
%   >> test_pagerank

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-10-12: Initial version .
%
% Function code starts here...
P = [0 1/2 1/2 0 0 0; 0 0 0 0 0 0; 1/3 1/3 0 0 1/3 0;...
     0 0 0 0 1/2 1/2; 0 0 0 1/2 0 1/2; 0 0 0 1 0 0];

ranks = pagerank(P',0.9,1e-8)

rand('seed',28);
n=1e6;
ST=sprand(n,n,1e-8);
tic,pagerank(ST,0.1,1e-8);toc
```

A.5 scalemat.m

scalemat.m

```
function B = scalemat(A)
%SCALEMAT - Takes a matrix A, with binary elements (0 or 1) and re-
    scales
%           it such that each column sum to 1. Returns sparse
    matrix B.
%
%   MINIMAL WORKING EXAMPLE: Rescale matrix,
%
%           A= [1 0 1 1;
%               0 0 1 1;
%               1 0 0 1;
%               0 0 1 0],
%
%           such that each column of the rescaled matrix B sum to
    1.
%
%   >> A = [1 0 1 1; 0 0 1 1; 1 0 0 1; 0 0 1 0]; %define A
%   >> B = scalemat(A); %re-scale A by using "scalemat"
%   >> full(B) %visualize your result by computing the full matrix

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-10-08: Initial version .
%
% Function code starts here...

%Make A sparse
A = sparse(A);

%Define vector of scaling factors for each column. Put them in a
    diagonal-
%sparse matrix.
scale = diag(sparse(1./sum(A,1)));

%Scale A and set this as B.
B = A*scale;
```

A.6 testpagerank.m

testpagerank.m

```
%TESTPAGERANK - Tests pagerank as asked for in "Reporting"
%
%   MINIMAL WORKING EXAMPLE: Perform the test
%
%   >> testpagerank

% Author: Gustav Nystedt , guny0007@ad.umu.se
% 2018-10-12: Initial version .
%
% Function code starts here...

%find number of pages with no outgoing links
nonOut = find(sum(HT,1) == 0);
num_nonOut = length(nonOut);

%save urls for pages with no outgoing links
nonOut_urls = cell(num_nonOut,1);
for i = 1:num_nonOut
    nonOut_urls{i,1} = urls{nonOut(i),1};
end

%find which url that has most num of outgoing links and how many
    they are
[max_out,i_max] = max(sum(HT,1));
max_out_url = urls{i_max,1};
%find which url that has most num of incoming links and how many
    they are
[max_in,i_maxIn] = max(sum(HT,2));
max_in_url = urls{i_maxIn,1};

alpha = 0.1:0.1:0.9; %define vector for all alphas we want to test
n = length(alpha); %save number number of test-alphas
eps = 1e-6; %define epsilon
%construct empty cell-array of length n for storing ranks
ranks = cell(n,1);
%pre-define zero-matrix for storing number of iterations for each
    alpha
iter = zeros(n,1);
%scale HT using scalemat enable calling of pagerank(HT,...)
HT_scale = scalemat(HT);

%iterate over each alpha
for i = 1:length(alpha)
    %perform pagerank on HT for each alpha
    [ranks{i}, iter(i)] = pagerank(HT_scale,alpha(i),eps);
end

%find top-ten pages with highest score for alpha = 0.65
rank_top = pagerank(HT_scale,0.65,eps);
[tTen_val,tTen_ind] = maxk(rank_top,10);

%save urls for top-ten-score pages
tTen_urls = cell(10,1);
for i = 1:10
    tTen_urls{i,1} = urls{tTen_ind(i),1};
end
```