Università della Svizzera italiana

Institute of Computing CI

Numerical Computing **2023**

Student: Kyla Kaplan

Solution for Project 1 **Due date:** Wednesday, 11 October 2023, 23:59 AM

Numerical Computing 2023 — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to iCorsi (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, MATLAB). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

1. Theoretical questions [15 points]

(a) **What are an eigenvector, an eigenvalue and an eigenbasis?**

An eigenvalue is a scalar, and characteristic property of a matrix. By definition, it may be zero, and it can also be a complex number. The set of existing eigenvalues of a matrix is also known as the spectrum of a matrix. If λ is an eigenvalue, and A is an nxn matrix, then **non-zero** vector x is an eigenvector, if it satisfies the following equation:

$$Ax = \lambda x$$

Together, eigenvalues and eigenvectors exist in pairs (λ, x), also known as eigenpairs. An eigenbasis is therefore a matrix basis, in which every vector is a linearly independent eigenvector, which span the entire vector space.

(b) **What assumptions should be made to guarantee convergence of the power method?**

The power method is a simple technique for computing the dominant eigenvalue and eigenvector of a matrix. It is based on the idea that repeated multiplication of a random vector (almost any vector) by the given matrix yields vectors that eventually tend towards the direction of the dominant eigenvector.

The rate of convergence of the power method depends on the ratio between the dominant eigenvalue, and the next dominant eigenvalue (e.g. λ_1 and λ_2). If λ_1 and λ_2 are too close, then the asymptotic error constant $|\frac{\lambda_2}{\lambda_1}|$ (which convergence is relative is to), can be slow. Therefore, choosing eigenvalues that are further apart is better for speed.

In order to guarantee convergence, we assume that any inputted matrix in the method is diagonalizable, i.e. has eigenvalues, and that there exists n linearly independent eigenvectors making up a basis.

Also, importantly, we assume that the method converges towards the most real dominant eigenvalue. And, we assume that the randomly chosen initial vector is in the same direction as the final eigenvector.

(c) **What is the shift and invert approach?**

When using the power method, the convergence rate is dependent on the asymptotic error constant, $|\frac{\lambda_2}{\lambda_1}|$, which can be relatively slow. Alternatively, the shift and invert approach/inverse iteration speeds up the convergence. While it has a higher computational complexity $O(n^3)$, than the power iteration $O(n^2)$; it approaches convergence in a faster manner.

As an example, λ_i is an eigenvalue of A, and $\lambda_i - a$ is an eigenvalue for $A - aI$. Then, for B (inverted) will be: $B = (A - aI)^{-}1$, for B, the eigenvalue will be $\mu_i = \frac{1}{\lambda_i - a}$. In this case, applying the power method will result in:

$$|\frac{\mu_2}{\mu_1}| = |\frac{\frac{1}{\lambda_2 - a}}{\frac{1}{\lambda_1 - a}}| = |\frac{\lambda_1 - a}{\lambda_2 - a}|$$

Therefore, it would be advantageous to choose an a that is closer to the eigenvalues λ_1 and λ_2 . In the shift and invert approach, it is also advantageous to choose an a, such that the rate is as small as possible.

(d) **What is the difference in cost of a single iteration of the power method, compared to the inverse iteration?**

The power method has a complexity of $O(n^2)$, since it involves solving matrix-vector multiplication operations on each round. While the shift and invert approach has a complexity of has a complexity of $O(n^3)$, since it implies solving systems of linear equations. Although the complexity cost of the shift and invert approach is more costly than the power method, is results in a rapid convergence, and the workaround is to therefore use it for smaller problems for faster results. Realistically, using the inverse iteration for problems like in internet searching would not be preferable, and there we employ the power method.

(e) **What is a Rayleigh quotient and how can it be used for eigenvalue computations?**

The Rayleigh quotient, $R(v) = \frac{v^T Av}{v^T v}$, is an addition to the shift and invert approach for finding and computing eigenvalues. Thanks to the Rayleigh quotient, convergence is sped up cubically, by replacing a. Although it has a higher time complexity due to refactoring the matrix at every iteration, the plus is the convergence speed.

2. Connectivity matrix and subcliques [5 points]

Range (between)	Dominant Organization
73 - 100	baug.ethz.ch
113 - 130	math.ethz.ch
164 - 182	mavt.ethz.ch
198 - 220	biol.ethz.ch
221 - 263	chab.ethz.ch
264 - 315	math.ethz.ch
319 - 348	erdw.ethz.ch
350 - 356	hest.ethz.ch
359 - 373	usys.ethz.ch
385 - 393	usys.ethz.ch
396 - 416	mtec.ethz.ch
426 - 431	mtec.ethz.ch
436 - 461	gess.ethz.ch
488 - 500	bilanz.ch

3. Connectivity matrix and disjoint subgraphs [10 points]

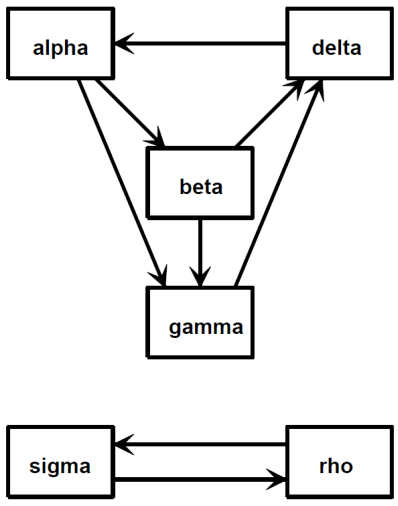


Figure 1: Six-node tiny web

1. What is the connectivity matrix G? Which are its entries?

   Generally, a connectivity matrix is an nxn sparse logical matrix where n is the number of web pages. The position (a, b) in the matrix is marked as 1 if there is an outgoing link from b to a and 0 otherwise. For the connectivity matrix of graph G, the columns and the rows belong to the following respective nodes: *"alpha", "beta", "gamma", "delta", "rho", "sigma"*. The

connectivity matrix G for the six-node webgraph is:

$$G = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

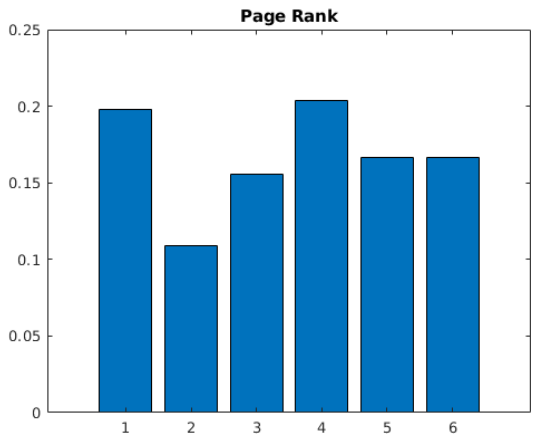
2. What are the PageRanks if the hyperlink transition probability p assumes the default value of 0.85?

   We utilize the `pagerank.m` MATLAB script, with the following inputs/information of our graph:

```
% U is the array of "URLs"
>> U = ["alpha", "beta", "gamma", "delta", "rho", "sigma"]
% array i is 'destination' of URL's relative to
% its position in U
>> i = [2, 3, 3, 4, 4, 1, 6, 5]
% j is the starting point of the URL's relative to
% their destination
>> j = [1, 1, 2, 2, 3, 4, 5, 6]
% n is number of nodes
>> n = 6
>> G = sparse(i, j, 1, n, n)
% we call a sparse matrix to show the relation between i and j
% with 1, in an nxn format

%then, p = 0.85
>> p = 0.85
>> pagerank(U, G, p)
```

   Resulting in the following barchart:



   And now, we summarize the further information into a table:

URL (U)	Index (i)	PageRank	Incoming	Outcoming
alpha	1	0.1981	1	2
beta	2	0.1092	1	2
delta	3	0.1556	2	1
gamma	4	0.2037	2	1
rho	5	0.1667	1	1
sigma	6	0.1667	1	1

We can notice how *gamma* is the highest-ranking "URL".

3. Describe what happens with this example to both the definition of PageRank and the computation done by pagerank in the limit p $\to$ 1.

URL (U)	Index (i)	$p = 0$	$p = 0.85$	$p = 0.99$
alpha	1	0.1667	0.1981	0.2051
beta	2	0.1667	0.1092	0.1026
delta	3	0.1667	0.1556	0.1538
gamma	4	0.1667	0.2037	0.2051
rho	5	0.1667	0.1667	0.1667
sigma	6	0.1667	0.1667	0.1667

In the above table, we can notice how pagerank changes according to p, which is the probability that a walk from one edge to another randomly follows a link, with $(1 - p)$ that it leads to another. If $p = 0$, then the pagerank remains the same for all URL's, since it will always lead equally randomly to an arbitrary page. If $p \approx 1$, then the chance that it will lead arbitrarily to a page becomes irrelevant, as the pagerank begins to correlate to the pagerank of an incoming link, and the number of outgoing links.

We can see further that for *rho* and *sigma*, their pagerank does not change, as they are disjoint from the full graph, and in the case with $p = 0$, they are never reached, and thus their pagerank score is unaffected.

4. PageRanks by solving a sparse linear system [25 points]

1. Create `pagerank1.m` by modifying `pagerank.m` to use the power method instead of solving the sparse linear system. The key statements are:

```
G = p*G*D
z = ((1-p)*(c~=0) + (c==0))/n;
while termination_test
    x = G*x + e*(z*x)
end
```

What is an appropriate test for terminating the power iteration?

At every iteration we compare the distances between elements of the newly-computed x and the previously-computed x to see if any of them are greater than the limit (which we set at a precision of 10^5). This way we check if they did not converge yet.

```
% Solve (I - p*G*D)*x = e
disp('Using power method\n');
% count = 0;
A = p * G * D;
z = ((1-p) * (c~=0) + (c==0)) / n ;
x = e / n ;

prevx = zeros (n , 1) ;
% we set the limit at 10^-5
limit = 0.00001;

while norm(x-prevx) > limit
    prevx = x ;
    x = A * x + e * (z * x);
    % count = count + 1
end
% disp("count: " + count);
```

2. Create pagerank2.m by modifying pagerank.m to use the inverse iteration. The key statements are:

```
while termination_test
    x = (alpha*I - A)\x
    x = x/norm(x,1)
end
```

Use your functions pagerank1.m and pagerank2.m (set $a = 0.99$) to compute the PageRanks of the six-node example presented in Figure 1. Make sure you get the same result from each of your three functions.

```
% Solve (I - p*G*D)*x = e
disp ('Using Inverse Iteration Implementation\n');

% count = 0;
z = ((1-p) * (c~=0) + (c==0))/n ;
A = p * G * D + (e * z) ;
x = e / n;

prevx = zeros(n, 1) ;

% we set the limit at 10^-5
limit = 0.00001;
a = 0.8;

while norm(x-prevx) >= limit
    prevx = x;
    x = ((a * I) - A) \ x ;
    % normalizing here
    x = x/sum(x);
    % count = count + 1;
end
% disp("count: " + count);
% disp("x: " + x);
```

For all three scripts (with $p = 0.85$):

URL (U)	Index (i)	PR default	PR1 Power	PR2 Inverse
alpha	1	0.1981	0.1981	0.1981
beta	2	0.1092	0.1092	0.1092
delta	3	0.1556	0.1556	0.1556
gamma	4	0.2037	0.2037	0.2037
rho	5	0.1667	0.1667	0.1667
sigma	6	0.1667	0.1667	0.1667

3. We now want to analyse the impact of a on the inverse iteration. Using the ETH500 example, set a equal to 0.8, 0.9, 0.95 and 1. Comment on the different number of iterations the four cases take until convergence. Analyse your results and explain what you observe. Hint: Check your solution x for all 4 cases. Are they always the same?

	$a = 0.8$	$a = 0.9$	$a = 0.95$	$a = 1$
Iteration Count	n/a	n/a	11	2

We can see that a should be inversely proportional to the iteration count, especially as the convergence increases.

4. Use your functions `pagerank1.m` and `pagerank2.m` (set $a = 0.99$) to compute the PageRanks of three selected graphs (`web1.mat, web2.mat and web3.mat`). Report on the convergence of the two methods for these subgraphs and summarize the advantages and disadvantages of the power method implemented in `pagerank1.m` against the inverse iteration in `pagerank2.m`.

File	PR1 count	PR2 count
web1.mat	35	4
web2.mat	31	4
web3.mat	34	4

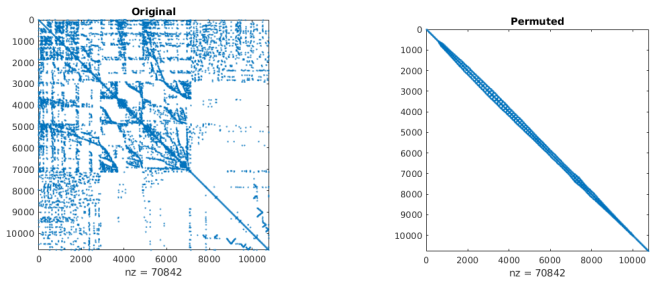
The power method has less iterations than the inverse method, however (theoretically), it is faster than the inverse method, as it requires more time complexity to compute.

5. The Reverse Cuthill–McKee Ordering [5 points]

After running the following in the MATLAB command window:

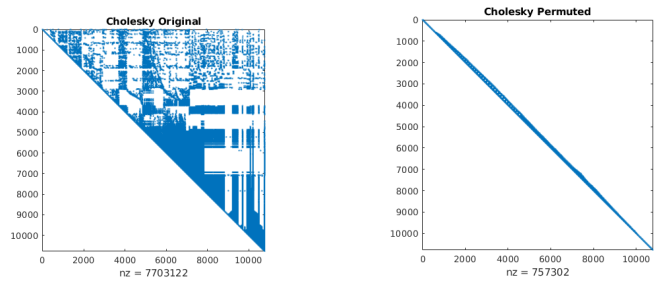
```
>> S = load('A_SymPosDef.mat');
>> p = symrcm(S.A_SymPosDef);
>> R = S.A_SymPosDef(p,p);
>> spy(S.A_SymPosDef), title('Original');
>> spy(R), title('Permuted');
>> CS = chol(S.A_SymPosDef);
>> CR = chol(R);
>> spy(CS), title('Cholesky Original');
>> spy(CR), title('Cholesky Permuted');
```

We can observe the following:



(a) (b)

Figure 2: Spy Graph of Original vs. Permuted



(a) (b)

Figure 3: Cholesky version of spy Graph of Original vs. Permuted

6. Sparse Matrix Factorization [10 points]

1.
$$A = \begin{pmatrix} 10 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 11 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 12 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 13 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 14 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 15 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 16 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 17 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 19 \end{pmatrix}$$

A has 44 non-zero entries.

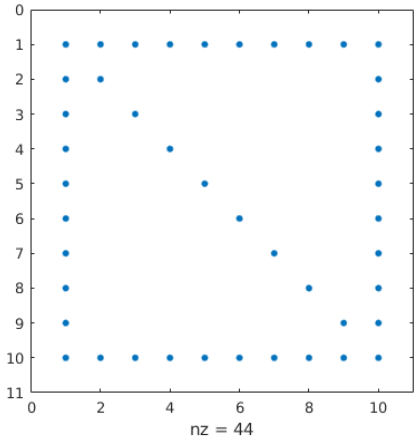
2. The non-zero elements appear on the diagonal and on the sides/upper/lower sides of the matrix. Therefore, total element = 5n, minus non-zero elements = 4, minus overlapping 1's on the edges = 2. Thus, 5n - 4 - 2 = 5n - 6.

3.
```
function [A] = A_construct(n)
A = zeros(n);
for i = 1: n
    for j = 1: n
        if (i ~= j && (i == 1 || i == n || j == 1 || j == n))
            %perimeters
            A(i, j) = 1;
        elseif (i == j)
            %diagonals
            A(i, j) = (n + i) - 1;
        end
    end
end
nz = 5*n-4;
disp(nz);
spy(A);
end
```

This results in:

```
ans =
      10     1     1     1     1     1     1     1     1     1
       1    11     0     0     0     0     0     0     0     1
       1     0    12     0     0     0     0     0     0     1
       1     0     0    13     0     0     0     0     0     1
       1     0     0     0    14     0     0     0     0     1
       1     0     0     0     0    15     0     0     0     1
       1     0     0     0     0     0    16     0     0     1
       1     0     0     0     0     0     0    17     0     1
       1     0     0     0     0     0     0     0    18     1
       1     1     1     1     1     1     1     1     1    19
```

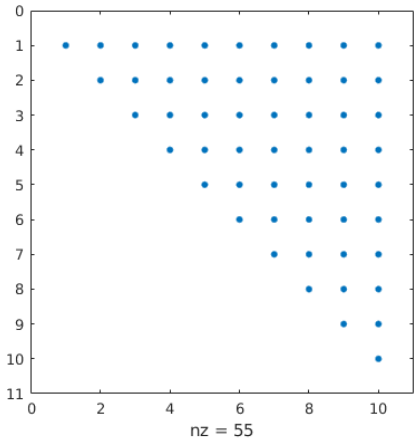
With the corresponding spy plot being:



4. Executing the following in the Command Window:

```
original = A_construct(10);
chol = chol(original);
spy(chol);
```

Results in:



5. Explain why, for n = 100, 000, using chol() to solve Ax = b for a given right-hand-side vector b would be problematic. Are there ways to mitigate this issue?

   Firstly, storage would be an issue. Secondly, time complexity. Thirdly, the sparsity of the matrix. Mitigating it via RCM ordering is one way to mainly deal with the matrix sparsity.

7. Degree Centrality [5 points]

8. The Connectivity of the Coauthors [5 points]

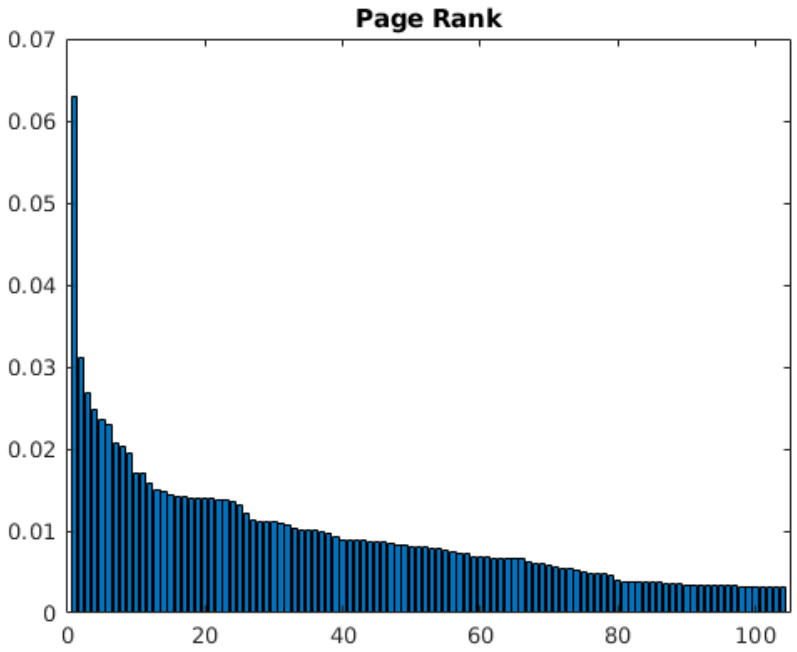
9. PageRank of the Coauthor Graph [5 points]

The script (`pagerank.m` remains unmodified, except for making sure G isn't self-referential), and the following is executed in the Command Window:

```
load('housegraph . mat');
U = name;
T = pagerank(U, A, 0.85);
sorted = sort(T);
flipped = flip(sorted);
```

```
bar(flipped);
```

Simply sorting and flipping the ranking, and then creating a bar chart results in:



For any missing MATLAB scripts, they are therefore included in this PDF, or have been executed using the MATLAB Command Window.