Università della Svizzera italiana

Institute of Computing CI

Numerical Computing **2023**

Student: Hun Rim **Discussed with:** Georgy Batyrev

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?**
Eigenvector : Eigenvector(i.e. v) of a square matrix(i.e. A) is a nonzero vector which linearly transforms in scale while staying in the same span when applied to the square matrix.

Eigenvalue: Often expressed with λ, eigenvalue is a scalar which represents how much the eigenvector gets stretched when multiplied by a square matrix(A). Although the direction doesn't change, the direction of the vector can be inverted from the transformation, hence, eigenvalue can be negative.
To conclude, for a square matrix A when applied to its eigenvector v, the linear transformation can be represented as combination of the eigenvalue and the eigenvector.

$$Av = \lambda v \tag{1}$$

Eigenbasis: An eigenbasis of a square matrix(A) is a **set of linearly independent eigenvectors** that spans the entire vector space of A. Basically it is set of eigenvectors found from all eigenvalues corresponding to the square matrix.

(b) **What assumptions should be made to guarantee convergence of the power method?**
For power method to guarantee a convergence, we need to assume that our power method is converging towards our dominant eigenvalue. Also we need to assume that the randomly chosen x_0, the initial vector, is is in the same direction as our eigenvector.

(c) **What is the shift and invert approach?**
When λ_2, λ_1 are close, because the convergence is relative to asymptotic error constant $\mid \frac{\lambda_2}{\lambda_1} \mid$, the convergence can be very slow when it approaches 1. Hence, we use shift and invert (Inverse Iteration) approach to fasten the convergence for smaller problems as it has higher complexity ($\mathrm{O}(n^3)$) for computation but faster approach to convergence in comparison to power iteration($\mathrm{O}(n^2)$).

For example, λ_i are eigenvalue of A, and $\lambda_i - \alpha$ are eigenvalue for A - αI. Then, eigenvalue for $B = (A - \alpha I)^{-1}$ is $\mu_i = \frac{1}{\lambda_i - \alpha}$, while the eigenvectors remain unchanged. If we apply power method to B, the new convergence rate will be $\mid \frac{\mu_2}{\mu_1} \mid = \mid \frac{\lambda_1 - \alpha}{\lambda_2 - \alpha} \mid$. Hence, choosing $\alpha \approx \lambda_1$ is very beneficial and common as it brings rapid convergence.

(d) **What is the difference in cost of a single iteration of the power method, compared to the inverse iteration?**
When solving a power method, we need to carry out matrix-vector multiplication, which has a time complexity of $O(n)^2$ like I mentioned in the previous answer. When solving an inverse iteration, we need to solve a linear system which has a time complexity of $O(n)^3$. Obviously, inverse iteration has far more severe time complexity in comparison to power method even though it has more rapid convergence. Hence, inverse method is recommended for smaller problems.

(e) **What is a Rayleigh quotient and how can it be used for eigenvalue computations?**
The Rayleigh quotient is a possible improvement on inverse iteration method when computing the eigenvalues. It guarantees upto cubic function when approaching the convergence. They achieve this quick approaching speed via replacing α with Rayleigh quotient in each iteration. As Rayleigh quotient has to refactor the matrix every iteration, it has higher time complexity but the benefits are worthwhile.

$$R(v) = \frac{v^T A v}{v^T v} \tag{2}$$

2. Connectivity matrix and subcliques [5 points]

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

3. Connectivity matrix and disjoint subgraphs [10 points]

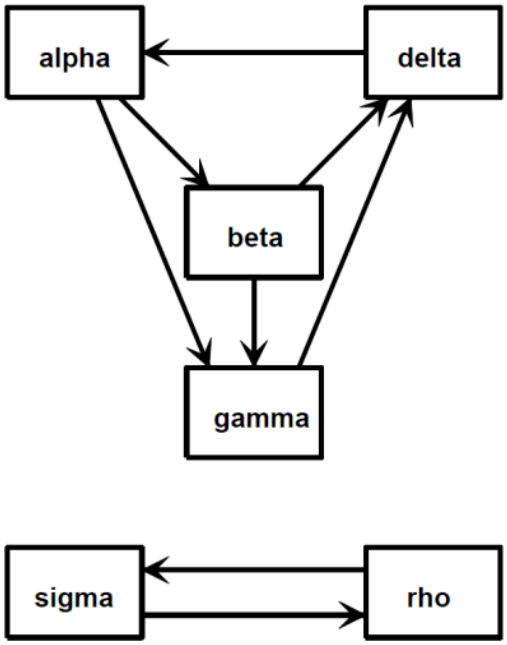

Figure 1: The tiny web

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

   Following the example given in the Project description page:

   U = ['alpha', 'beta', 'gamma', 'delta', 'rho', 'sigma']
   i = [2, 3, 3, 4, 4, 1, 6, 5]
   j = [1, 1, 2, 2, 3, 4, 5, 6]
   n = 6
   G = sparse(i, j, 1, n, n)

   To briefly describe the variables, U is the array of shortened URLs, i is the array of destination URLs in-terms of its index in the URL array, and j is the array of starting point URLs in same

context as the destination URLs. Naturally, n is the dimension of connectivity matrix (G) as the tiny web consists of 6 websites. Finally, the connectivity matrix is formed using a sparse matrix depicting the relation between 2 vectors *i, and j* in a $n \times n$ matrix format. When we call the *full(G)* to see the $n \times n$ format of connectivity matrix G, we get the following result: (Assuming we follow the row and column order according to the elements of the vector U)

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

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

   Knowing our transition probability $p = 0.85$, we can use the given function, **pagerank**, from *pagerank.m* MATLAB script and call the following on MATLAB command window to get the PageRanks of our imaginary **tiny web**:

   $f_x >> p = 0.85$
   $f_x >> pagerank(U, G, p)$

   Using arrays U, G, which are reutilized, the result will be the following:

Figure 2: Barchart for pagerank of nodes in the tiny web (figure 1)

If we summarize the bar graph data, and connectivity matrix data into a table, we get the following:

Index	U	PageRank	Incoming	Outgoing
1	alpha	0.19814	1	2
2	beta	0.109209	1	2
3	delta	0.155623	2	1
4	gamma	0.203694	2	1
5	rho	0.166667	1	1
6	sigma	0.166667	1	1

Figure 3: Summarized table for pagerank of nodes in the tiny web

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

Index	U	PR(0.85)	PR(0.00)	PR(0.999999)	In	Out
1	alpha	0.19814	0.166667	0.205128	1	2
2	beta	0.109209	0.166667	0.102564	1	2
3	delta	0.155623	0.166667	0.153846	2	1
4	gamma	0.203694	0.166667	0.205128	2	1
5	rho	0.166667	0.166667	0.166667	1	1
6	sigma	0.166667	0.166667	0.166667	1	1

Figure 4: Summarized table for pagerank(s) of nodes in the tiny web (PR: PageRank)

p is the probability that the random walk follows a link(edge) and (1 - p) is the probability that it leads to an arbitrary page.
Hence, if $p \rightarrow 0$ all the pages have the same pagerank as it always leads to an arbitrary page, and all the pages have the same pagerank as it always leads to a random website in the web. When $p \rightarrow 1$, because the possibility of leading to an arbitrary page is negligible, pagerank directly correlates to incoming link's pagerank and their number of outgoing links. I.e. alpha has two outgoing hyperlinks so beta's pagerank is exactly half of alpha unlike when $p \rightarrow 0.85$. Furthermore, usually delta and alpha's pagerank are very close but never same as there are constant factors added to their pagerank due to the possibility of going to a random page.
However, When p converges to 1, because there are negligible chance of being directed to an arbitrary page, it simplifies the computation of pageranks.

Problem with p being 0, is that in this sort of web (disjoint graph), it would become impossible to traverse some parts of the graph.

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?

In the default section, I implemented the following code:

```
z = ((1-p) * (c~=0) + (c==0)) / n;
A = p * G * D + e * z;
x = e / n;
oldx = zeros(n, 1);
limit = 0.00001;
while norm(x - oldx) > limit
    oldx = x;
    x = A * x;
end
x = x / sum(x);
```

As it can be seen from the code implementation, the while loop continues the iteration while:

```
norm(x - oldx) > limit
```

Meaning, the iteration gets terminated when the normal between old and new x converges to the given limit and goes below it. (which in this case was 0.01)

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 / sum(x, 1);
end
```

Use your functions pagerank1.m and pagerank2.m (set $\alpha = 0.99$) 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.

Index	U	PR(figure 1)	PR	PR1	PR2	In	Out
1	alpha	0.3210	0.3210	0.3210	0.3210	2	2
2	beta	0.1705	0.1705	0.1705	0.1705	1	2
3	delta	0.1066	0.1066	0.1066	0.1066	1	3
4	gamma	0.1368	0.1368	0.1368	0.1368	2	1
5	rho	0.0643	0.0643	0.0643	0.0643	1	0
6	sigma	0.2007	0.2007	0.2007	0.2007	2	1

Figure 5: Result of computing PR for figure 1 (in project file) data using all 3 pageranking scripts.

implementation for *pagerank2.m*, Inverse iteration method is as following:

```
z = ((1-p)*(c~=0) + (c==0))/n;
disp('Using Inverse Iteration Implementation')
x = e / n;
A = p * G * D + (e * z);
oldx = zeros(n, 1);
limit = 0.00001;
alpha = 0.99;
while norm(x - oldx) >= limit
    oldx = x;
    x = (alpha * I - A) \ x;
    x = x/sum(x);
end
```

3. We now want to analyse the impact of α on the inverse iteration. Using the ETH500 example, set α 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.

	$\alpha = 0.99$	$\alpha = 0.95$	$\alpha = 0.92$	$\alpha = 0.915$	$\alpha = 0.9$, 0.8
Iteration Count	5	11	28	38	NaN generated

Figure 6: Iteration count for different values of α when limit is 10^{-5}

From limited data observed through inverse iteration algorithm ($\alpha = 0.92, 0.915$ are added for better analysis), we can observe that the α is inversely proportional to iteration count, because as α decreases, iteration count till convergence increases as it is moving away from λ. Also the iteration count exponentially increases as α gets smaller. Furthermore, after certain value of α (between 0.9 to 0.915), it becomes incomputable.

4. Use your functions *pagerank1.m* and *pagerank2.m* (set $\alpha = 0.99$) 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.*

   After running power method algorithm and inverse iteration algorithm on 3 different data sets (*web1.mat, web2.mat, web3.mat*) it became clear inverse iteration goes through immensely less number of iterations to reach the converging point. However, to measure the two algorithm's performance more accurately, their average execution time is also measured.

Dataset	PR1 Iteration	PR1 Run Time(avg)	PR2 Iteration	PR2 Run Time(avg)
web1.mat	145	0.0518	14	0.0682
web2.mat	145	0.0532	14	0.0668
web3.mat	212	0.0432	15	0.0675

Figure 7: Table showing iteration count and average execution time of reaching convergence using power method and inverse iteration

   The average execution time was measured using $timeit(func())$ function built into the MATLAB, which takes a pointer to a function and outputs the function's average execution time after running it numerous times.

   To clearly see the execution time difference, the convergence limit was set to 10^{-17} and as it can be seen from the table above, even though power method clearly exceeds inverse iteration in terms of iteration count (lower is better), the average execution time of inverse iteration method is $30^{+}\%$ higher (lower is better) than power method. To conclude, power method provides faster results as advantages although it requires more iterations.

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

Visualization of Matrices:
The sparsity pattern of the original and permutted $A\_SymPosDef$ matrices are shown below:

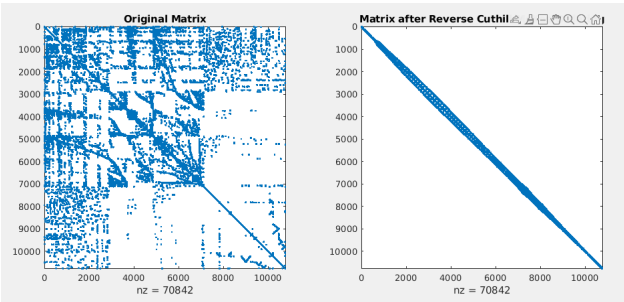

Figure 8: Spy graphs of original matrix (left) and matrix after Reverse Cuthill McKee permutation(right)

From the figure above we can observe the way sparse values, covering large area have been minimized to a denser, smaller area.

Visualization of Cholesky Factors

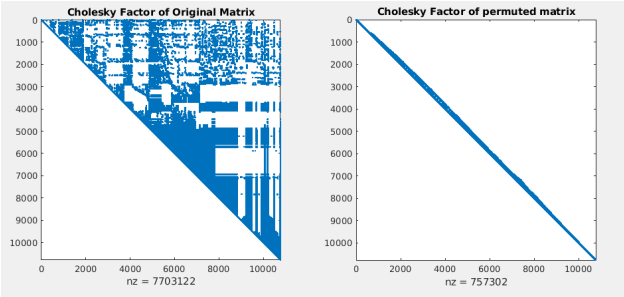

Figure 9: Original Matrix after Cholesky decomposition,(left) and Permutted matrix after Cholesky decomposition(right)

From figure 9, we may observe the similar pattern of behavior as in graph above. Interestingly, it is important to note that if the matrix must be positive definite using eigenvalues, else we can't compute Cholesky Factorization. (To recreate the graphs, please run *reverse.m* script)

6. Sparse Matrix Factorization [10 points]

Let $A \in R^{n \times n}$ be symmetric and positive definite, with entries A_ij defined as follows:

$$A_{ij} = \begin{cases} 1 & if \quad i = 1 \quad or \quad i = n \quad or \quad j = 1 \quad or \quad j = n \quad and \quad i \neq j \\ n + i - 1 & if \quad i = j \\ 0 & otherwise \end{cases} \tag{3}$$

Please note that the increasingly larger values on the diagonal are necessary to ensure the positive definiteness of matrix A.

1. Construct matrix A for the case n = 10 and explicitly write down its entries. How many non-zero elements does it have?
   According to the given A_{ij} equation, when n = 10, the matrix A is as following:

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

   The square matrix A has 56 entries which are zero and 44 entries which are non-zero values.

2. We now want to derive a general formula to compute the number of non-zero entries. Show that, for a given matrix $A \in R^{n \times n}$ with this structure, the number of non-zero elements are $5n - 6$.

   Fundamently, number of non-zero elements equal to:

   elements on the border of matrix + element on left-to-right diagonal - overlapping elements

Their are n elements on each borders and the diagonal so:

$$total - non - zero - entries = 4n + n - \alpha = 5n - \alpha \tag{4}$$

Where α equals to number of overlapping elements.
If we count all the borders of a square matrix, each of the edges will be counted twice so we need to minus four from our $5n$, now when we count the non-zero entries on the diagonal on top of our calculation made on entries on the border, two edges where $i = j, i = 1, n$ will overlap once each. Hence, we need to minus another two from our $5n - 4$. Finally we get:

$$total - non - zero - entries = 5n - 4 - 2 = 5n - 6 \tag{5}$$

3. Write a function *A_construct()*, which takes as input n and returns, as output, the matrix A defined in Eq. 3 and its number of non-zero elements nz. Test your function in a script *ex2c.m for n = 10* and compare your results with those you obtained in point (a). Furthermore, within the same script, visualise the non-zero structure of matrix A by using the command *spy()*.

```
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
```

Figure 10: Result matrix of *A_construct(n = 10)* function

As it can be seen, resultant matrix of the *A_construct(10)* has identical entries as the answer to question 6.1, and the following is the spy graph of the function:

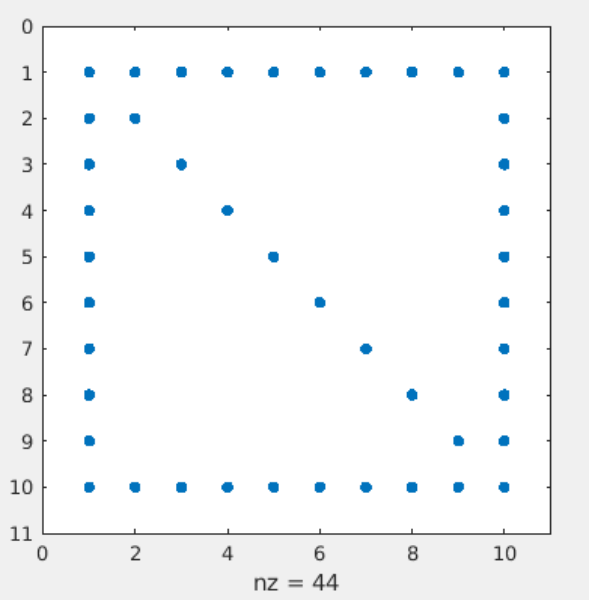

Figure 11: Result spy graph of *A_construct(n = 10)* function

The function *A_construct()* has time complexity of $O(n^2)$. In the code below, to express inequality an exclamation mark(! =) is used for readability, but in MATLAB tilde(∼=) symbol is used.

```
function nmm = A_construct(n)
    A = sparse(zeros(n,n));
    for i = 1:n
        for j = 1:n
            if (i == 1 || i == n || j == 1 || j == n) && i != j
                A(i, j) = 1;
            elseif i == j
                A(i, j) = n + i - 1;
            end
        end
    end
    nmm = A;
end
```

4. Using again the *spy()* command, visualize side by side the original matrix A and the result of the *Cholesky factorization (chol() in Matlab).*

```
original = A_construct(10);
cholesky = chol(original);
tiledlayout(1, 2);
nexttile
spy(original);
title('Original Matrix Spy Graph');
nexttile
spy(cholesky);
title('Cholesky Factorization Spy Graph');
```

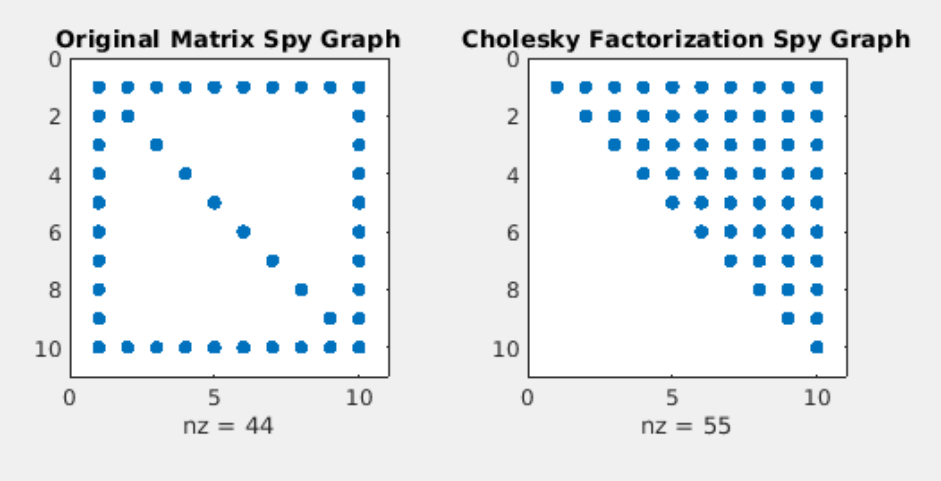

Figure 12: Original Matrix Spy Graph[left], Result Of Cholesky Factorization [Right]

The graphs above is generated by a MATLAB script stored in *original_cholesky.m* file using the code given in the previous page.

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?

   There are serious problem behind solving Cholesky factorization using $Ax = b$ for a given right-hand-side vector b.

a) With this method, you need to store both original matrix (let's call it A) and the lower triangle (L). Considering square matrix A has size $n \times n$ where $n = 100,000$ and it has to store L of the same size, it may exceed the given memory limit.

b) Also it has a very high time complexity. The computation has a cost of $O(n^3)$. For a square matrix with row and column size of $n = 100,000$, it is a very costly and inefficient operation.

To mitigate the issue highlighted above, there are few solutions.

a) If we use sparse matrix and algorithm fit for it, for matrices containing a lot of zeros, it will significantly reduce both space and time complexity.

b) Secondly, it is also possible to divide the problem into smaller pieces and use parallelization to solve the problem using multiple threads, fastening the computation speed.

c) Thirdly, it is possible to take a more iterative approach more suited for larger sparse systems such as *Conjugate Gradient(CG) Method*, instead of solving the system directly using Cholesky decomposition.

7. Degree Centrality [5 points]

```
load('householder/housegraph.mat');
degree = sum(A, 2);
[degreeSorted, indexNames] = sort(degree);
count = length(degreeSorted);
maxDegree = degreeSorted(count);
coauthors = strings(5, maxDegree)
authors = strings(5, 1);
k = 1;
for i = count : -1 : count - 4
    ind = indexNames(i);
    j = 1;
    while ~isspace(name(ind, j))
        authors(k) = strcat(authors(k), name(ind, j));
        j = j + 1;
    end
    k = k + 1;
end

l = 1;
for i = count: -1 : count -4
    nameCount = 1;
    ind = indexNames(i);
    for j = 1 : count
        if (A(ind, j) == 1)
            k = 1
            while ~isspace(name(j, k))
                coauthors(l, nameCount) = strcat
                    (coauthors(l, nameCount), name(j, k));
                k = k + 1;
            end
            nameCount = nameCount + 1;
        end
    end
l = l + 1;
end
```

By executing the code above through *degreeOfCentrality.m* script, it is possible to see the top 5 authors from the *authors variable* and their coauthros from the *coauthors variable*. The table below shows a summary of gathered information:

Degree Of Centrality	Index	Author	Co-authors
32	1	Golub	Golub, Wilkinson, TChan, Varah, Overton etc.
16	104	Demmel	Edelmen, VanLoan, Bai, Schreiber, Kahan etc.
14	86	Plemmons	Golub, Nagy, Harrod, Pan, Funderlic etc.
13	81	Heath	Golub, TChan, Funderlic, George, Gilbert etc.
13	44	Schreiber	TChan, VanLoan, Moler, Gilber, Pothen etc.

Figure 13: Table of authors and their coauthors sorted according to author's degree of centrality.

8. The Connectivity of the Coauthors [5 points]

How many coauthors have the authors in common? Think about a general procedure that allows you to compute the list of common coauthors of two authors and express it in matrix notation. Use the formula you derived to compute the common coauthors of the pairs (Golub, Moler), (Golub, Saunders), and (TChan, Demmel). Who are these common coauthors? Report their names.

The solution is very straight forward. From the connectivity matrix A which contains all the connection between authors and coauthors, if two author's columns have 1(have connection) on the same row, it means they have a common coauthor. From this logical process we can define the common coauthors between two authors as following:

$$commonCoauthorIndex = A(col : author_1) \cap A(col : author_2) \tag{6}$$

If this logic is implemented as a script in MATLAB(view *ex8.m* script for the code), we get the following results:

Author1	Author2	Common Coauthors
Golub	Moler	Wilkinson, VanLoan
Golub	Saunders	Golub, Saunders, Gill
TChan	Demmel	Schreiber, Arioli, Duff, Heath

Figure 14: Table of authors and their common co-authors

9. PageRank of the Coauthor Graph [5 points]

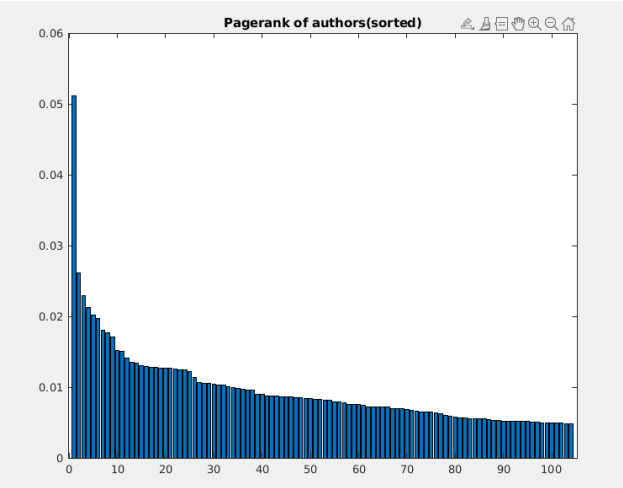

Figure 15: Pagerank of authors sorted(descending)

The graph has been generated from the following script below using the given pagerank script as part of the project package.

```
load('householder/housegraph.mat');
U = cellstr(name);
authorRanks = pagerank(U, A, 0.85);
sortAuthorRank = sort(authorRanks);
sortAuthorRank = flip(sortAuthorRank);
bar(sortAuthorRank);
title('Pagerank of authors(sorted)');
```

The script simply uses A as the connectivity matrix and generates the pagerank as authorRanks variable. First it is sorted in ascending order and it is flipped using a flip() function and displayed as a bar graph.

All the MATLAB script used during this project are stored in the src folder.