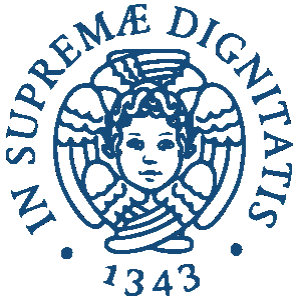


Master Degree in Computer Science
Curriculum Artificial Intelligence

Computational Mathematics for Learning and Data Analysis
A.Y. 2022/2023

Gennaro Daniele **Acciaro** (625009)
Alessandro **Capurso** (638273)

Contents

1 Problem description

(P) is a sparse linear system of the form:

$$\begin{bmatrix} D & E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$$

where $D \in \mathbb{R}^{m \times m}$ is a diagonal positive definite matrix (i.e., $D = diag(D) > 0$) and $E \in \mathbb{R}^{(n-1) \times m}$ is obtained by removing the last row from the node-arc incidence matrix of a given connected directed graph.

- (A1) is GMRES, and you must solve the internal problems min $||H_n y - ||b||e_1||$ by updating the QR factorization of H_n at each step: given the QR factorization of H_{n-1} computed at the previous step, apply one more orthogonal transformation to compute that of H_n.
- (A2) is the same GMRES, but using the so-called Schur complement preconditioner :

  $$P = \begin{bmatrix} D & 0 \\ 0 & -S \end{bmatrix}$$

  where S is either $S = -ED^{-1}E^T$ or a sparse approximation of it (to obtain it, for instance, replace the smallest off-diagonal entries of S with zeros).
  P must be factorized with the Incomplete Cholesky factorization.

1.1 Notation

This section defines the notation that will be used from here on.

- $J = \begin{bmatrix} D & E^T \\ E & 0 \end{bmatrix} \in \mathbb{R}^{(m+n-1) \times (m+n-1)}$
- $\tilde{x} = \begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{R}^{(m+n-1) \times 1}$
- $\tilde{b} = \begin{bmatrix} b \\ c \end{bmatrix} \in \mathbb{R}^{(m+n-1) \times 1}$

2 Methods

2.1 QR Factorization

QR Factorization is a decomposition of a matrix A into a product $A = QR$ of an orthogonal matrix Q and an upper triangular matrix R. There are several implementations of this factorization, in this work, the focus will be on the Givens rotation approach.

A *Givens Rotation* is defined as an orthogonal matrix with the following form:

$$\mathbf{G_i} = \begin{bmatrix} 1 & \cdots & 0 & \dots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \dots & \vdots & & \vdots \\ 0 & \cdots & c_i & \dots & -s_i & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & s_i & \dots & c_i & \cdots & 0 \\ \vdots & & \vdots & \dots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \dots & 0 & \cdots & 1 \end{bmatrix}$$

where given a_i and b_i, c_i and s_i are defined $c_i = \frac{a_i}{\sqrt{a_i^2+b_i^2}}$, $s_i = \frac{b_i}{\sqrt{a_i^2+b_i^2}}$.

Following [4], the method works by iteratively applying Givens rotations to the matrix A in order to eliminate the entries below the diagonal.

These matrices are denoted as 'rotations' because they are all orthogonal and so they preserve the length. Each time a rotation is applied, the b_i element of the matrix is zeroed as follows:

$$\begin{bmatrix} c_i & -s_i \\ s_i & c_i \end{bmatrix} \begin{bmatrix} a_i \\ b_i \end{bmatrix} = \begin{bmatrix} r_i \\ 0 \end{bmatrix}$$

For each column of A, the rotation is applied to the i-th element that is needed to zeros (called b_i) and the $i-1$-th element (called a_i), for each pair of values under the main diagonal element starting from the bottom.

By applying all the rotations to matrix A, an upper triangular matrix R is obtained as result. The concatenation of orthogonal matrices $G_1 \dots G_n$ is called Q. In this way, a QR factorization is obtained as shown in Equation 1.

$$G_n^T G_{n-1}^T \dots G_1^T A = Q^T A = R \tag{1}$$

2.2 Arnoldi

The Arnoldi method relies on the idea of using the Krylov space $K_n(A, \mathbf{b})$ to solve the linear system $Ax = b$. Indeed, let

$$\mathbf{V} = \begin{bmatrix} \mathbf{b} & A\mathbf{b} & A^2\mathbf{b} & \dots & A^{n-1}\mathbf{b} \end{bmatrix}$$

it is possible to solve the system by looking at the best solution inside $\mathbf{V}$. But this can be infeasible due to the alignment of the columns of $\mathbf{V}$, so the Gram-Schmidt algorithm can be used to generate a better basis for $K_n(A, \mathbf{b})$. In particular, the Arnoldi method

generates an orthonormal basis Q that spans $K_{n+1}(A, \mathbf{b})$. The output of the algorithm is an orthonormal basis Q and a Hessemberg matrix $\underline{H_n}$ composed as:

$$Q = \begin{bmatrix} \mathbf{q_1} & \mathbf{q_2} & \dots & \mathbf{q_{n+1}} \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}$$

$$\underline{H_n} = \begin{bmatrix} X & X & \dots & X & X \\ X & X & \dots & X & X \\ 0 & X & \dots & X & X \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & X & X \\ 0 & 0 & \dots & 0 & X \end{bmatrix} \in \mathbb{R}^{(n+1) \times n}$$

In the end, it is possible to state that for some matrix $\underline{H_n} \in \mathbb{R}^{(n+1) \times n}$

$$AQ_n = Q_{n+1}\underline{H_n} \tag{2}$$

where $\underline{H_n}$ is the matrix that contains all the $\beta_{i,j}$ that are the coefficients in:

$$Aq_j = q_1\beta_{1,j} + q_2\beta_{2,j} + \cdots + q_j\beta_{j,j} + q_{j+1}\beta_{j+1,j}$$

Furthermore, in the light of the fact that $\mathbf{v}$ can be written as the following linear combination, because $\mathbf{v}$ always has degree j:

$$b\alpha_0 + Ab\alpha_1 + \cdots + A^j b\alpha_j$$

Hence it is possible to conclude that if $\mathbf{v} = 0$ then $\mathbf{b}, A\mathbf{b}, ..., A^j\mathbf{b}$ are not linearly independent. Here is the pseudocode to describe how the algorithm works [5]:

Algorithm 1 Arnoldi's algorithm

```
b = arbitrary
v = b/||b||_2
for n = 1, 2, 3, ... do
    v = A q_n
    for j = 1, 2, ..., n do
        h_jn = q_j^T v
        v = v − h_jn q_j
    end for
    h_{n+1,n} = ||v||_2
    q_{n+1} = v / h_{n+1,n}
end for
```

2.3 GMRES

The Generalized Minimal RESidual method (GMRES) is an iterative algorithm that looks for the best approximate solution inside $K_n(A, \mathbf{b})$. More in detail, this method aims to minimize:

$$\min_{x \in K_n(A,\mathbf{b})} ||A\mathbf{x} - \mathbf{b}|| \tag{3}$$

where $x = Q_n\mathbf{y}$.
The problem can be written as:

$$\min_{y} ||A(Q_n\mathbf{y}) - \mathbf{b}|| \tag{4}$$

The vector $\mathbf{y}$ is found by solving a least-squares problem.
Actually, it is possible to reduce the dimensions of this problem in the following way:

$$||AQ_n\mathbf{y} - \mathbf{b}|| = ||Q_{n+1}\underline{H}_n\mathbf{y} - \mathbf{b}|| = |||[\ Q_{n+1}\hat{Q}\ ]^T(Q_{n+1}\underline{H}_n\mathbf{y} - \mathbf{b})|| = ... = ||\underline{H}_n\mathbf{y} - \mathbf{e}_1||\mathbf{b}|||| \tag{5}$$

So, in the end, a LS problem of size $(n+1) \times n$ is obtained.

It's worth noting that GMRES has a lucky breakdown, indeed when the Arnoldi method breaks down, the linear system has an exact solution. [5]

2.4 Custom version of GMRES

The proposed solution for the (A1) problem presented in Section 1 is an application of the GMRES where at each Arnoldi's iteration the matrix H_n is factorized using the QR method with the Givens rotations as suggested in [5].

2.4.1 Hn factorization

The first step is to understand how the matrix H_n should be factorized using the QR method. Considering the application of the Givens rotations explained in Section 2.1, taking into account that the matrix is an Hessemberg matrix, the factorization can be computed by applying a rotation per column in order to zeros all the sub-diagonal elements.
In order to understand how this factorization works, suppose to have the following Hessemberg matrix $A \in \mathbb{R}^{4\times4}$:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{pmatrix}$$

R is computed as:

$$\underbrace{\begin{pmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^T}_{G_1} \cdot \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{pmatrix} \rightarrow \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c_2 & -s_2 & 0 \\ 0 & s_2 & c_2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^T}_{G_2} \cdot \begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{pmatrix}$$

$$\rightarrow \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & c_3 & -s_3 \\ 0 & 0 & s_3 & c_3 \end{pmatrix}^T}_{G_3} \cdot \begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a''_{22} & a''_{23} & a''_{24} \\ 0 & 0 & a'_{33} & a'_{34} \\ 0 & 0 & a_{43} & a_{44} \end{pmatrix} \rightarrow \begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & a'_{14} \\ 0 & a''_{22} & a''_{23} & a''_{24} \\ 0 & 0 & a''_{33} & a''_{34} \\ 0 & 0 & 0 & a'_{44} \end{pmatrix} \rightarrow$$

$$\rightarrow R \in \mathbb{R}^{4\times 4} \text{ upper triangular}$$

and Q is computed as:

$$Q = G_1 \cdot G_2 \cdot G_3$$

2.4.2 Factorization in the iterative step

Let n denote the index of iteration, given that the H_n matrix is constructed iteratively, a new vector $h_n \in \mathbb{R}^{n+1}$ is introduced at each iteration. Since the resulting matrix assumes the form of a Hessemberg matrix, the new vector can be factorized with a singular rotation that zeros the last element. Before doing this, the new vector must also be rotated with all previous rotations utilized during earlier iterations, as it is possible to notice in the example provided in Section 2.4.1.
Taking into account the example aforementioned, supposing to be at the third step of the iteration, the rotation matrices G_1 and G_2 are already computed so the new vector is transformed as:

$$h_n = G_3^T * (G_2^T * (G_1^T * h_n))$$

2.4.3 Problem structure study

Considering that our problem has a specific structure, as described in the Introduction, it is worth starting with some observations.

Observation 1. *J is symmetric.*

Proof. Let's recall that J has this form:

$$J = \begin{bmatrix} D & E^T \\ E & 0 \end{bmatrix} \in \mathbb{R}^{(m+n-1)\times(m+n-1)}$$

Analysing each component of this matrix, it is possible to state that it is symmetric. Indeed:

- $D \in \mathbb{R}^{m\times m}$ is a squared and diagonal matrix.
- $0 \in \mathbb{R}^{(n-1)\times(n-1)}$ is a squared and diagonal matrix.
- E^T is the transpose of E.

Considering the sub-matrices composition and how they are arranged into the matrix, it is possible to conclude that J is symmetric. □

Observation 2. *H_n is symmetric.*

Proof. The aim of this proof is to find that $H^T = H$.
From Equation 2, it is possible to state that:

$$JQ = QH \implies JQQ^T = QHQ^T \implies J = QHQ^T$$

Considering Observation 1:

$$(QHQ^T)^T = QHQ^T \implies QH^TQ^T = QHQ^T$$

So it is possible to conclude that:

$$H^T = H$$

□

Observation 3. *H_n is tridiagonal.*

Proof. H_n is an Hessenberg matrix. Considering the Observation 2, if H_n is symmetric it means that it has zeros not only under the sub-diagonal but also above the super-diagonal. So it is tridiagonal. □

2.4.4 Step optimizations

Considering the observations presented in section 2.4.3, it is evident that Arnoldi's algorithm, applied to the (A1) problem, yields tridiagonal matrices (H_n) only. This matrix structure can be exploited to reduce the time complexity of the algorithm.
The proposed algorithm can be divided into three main parts: Arnoldi's iteration, QR factorization and LS solution.

The first part can be optimized by replacing the Arnoldi algorithm with the Lanczos algorithm since the input matrix will be always symmetric, as shown in Observation 1.
The optimization is obtained by modifying the loop at line 5 (referring to the pseudocode in section 2.2).
This loop computes all the values in the new column of H_n. In this case, only two elements will be non-zero, so the loop can be optimized by calculating only the values in the $(n-1)$-th and n-th positions. The calculation of the $(n+1)$-th vector of the Q matrix (performed on line 7) can also be optimized since it utilizes the newly calculated values of H_n. Therefore, the new vector can be computed as follows:

$$v = v - h_{n-1,n} * q_{n-1} - h_{n,n} * q_n$$

Also the second part can be optimized considering the symmetric form of the matrix. After each Arnoldi step, a new column is introduced. This vector must be also rotated w.r.t the previous rotations applied on the previous columns.
In the symmetric case, with $n > 2$, it is not necessary to apply all the previous rotations but it is possible to take into account the rotations that imply the $(n-2)$-th, $(n-1)$-th and n-th row. For what concerns the case where $n = 2$, only the previous rotations that imply the $(n-1)$-th and the n-th row must be taken into account.
After this step, a new Givens rotation matrix is found to zeros the $n+1$-th element of the new vector.

In conclusion, in the LS solution part, no optimizations are needed to be applied because, after the QR factorization, the matrix $H_n = QR$. This means that it is possible to solve the LS problem with the back substitution method.

3 Stability

As we have already mentioned, our algorithm can be divided into three main parts: Lanczos iteration, QR factorization and LS solution. In order to determine the algorithm's resistance to machine precision errors, the stability of each component is checked in this section.

Stability of Lanczos iteration As stated in [1], when the Lanczos algorithm is implemented in a floating point arithmetic machine, the algorithm is not stable due to the round-offs which lead to the loss of orthogonality among Ritz vectors.
This problem can be handled with the **reorthogonalization**: apply the Gram-Schmidt orthogonalization process with respect to all the previously computed vectors.
So, the Arnoldi algorithm described in section 2.2 became:

Algorithm 2 Lanczos algorithm with full reorthogonalization

1: b = *arbitrary*
2: $v = b/||b||_2$
3: **for** $n = 1, 2, 3, \ldots$ **do**
4: $\quad v = Aq_n$
5: $\quad \alpha_j = q_n^T\, v$
6: $\quad v = \sum_{i=1}^{n}(z^T q_i)q_i$ ▷ reorthogonalization step
7: $\quad \beta_j = ||v||_2$
8: $\quad q_{n+1} = \frac{v}{\beta_j}$
9: **end for**

The use of reorthogonalization in the Lanczos algorithm does not guarantee that the algorithm is backward stable, but it reduces the effects of round-off errors.

Stability of QR

Lemma 1. *Multiplying by an orthogonal matrix is a backward stable operation. [3]*

Observation 4. *The QR factorization algorithm with the Givens Rotations method is backward stable.*

Proof. Looking at the k-th step, the operation computed is

$$\tilde{A}_k = \tilde{G}_k \cdot (\tilde{A}_{k-1} + \Delta_{k-1})$$

with

$$||\Delta_{k-1}|| \leq O(u)||\tilde{A}_{k-1}|| = ||A||O(u)$$

Starting from $k = 1$

$$\tilde{A}_1 = \tilde{G}_1 \cdot (A + \Delta_0)$$

$$\tilde{A}_2 = \tilde{G}_2(\tilde{G}z_1 \cdot (A + \Delta_0) + \Delta_1) = \tilde{G}_2\tilde{G}_1((A + \Delta_0) + \tilde{G}_1^T\Delta_1)$$

$$\vdots$$

$$\tilde{A}_k = \tilde{G}_k\tilde{G}_{k-1}\ldots\tilde{G}_2\tilde{G}_1(A + \Delta_0 + \tilde{G}_1^T\Delta_1 + \cdots + \tilde{G}_1^T\tilde{G}_2^T\ldots\tilde{G}_{n-1}^T\Delta_{n-1})$$

Each error term is $||A||O(u)$, so, considering n error terms the bound is

$$||A|| \cdot n \cdot O(u) = ||A|| \cdot O(u)$$

Moreover, all the matrices G_i are orthogonal and the $\tilde{G}_k\tilde{G}_{k-1}\dots\tilde{G}_2\tilde{G}_1$ operation is backward stable as state in the Lemma 1. So, it is possible to conclude the QR factorization with Givens rotations matrices is backward stable. □

Stability of LS Since the Least Squares problem is solved using the QR factorization of the H_n matrix, it is also possible to state that the LS algorithm is backward stable [5].

Conclusions Due to the fact that the Lanczos iteration is not backward stable, there are no assurances regarding the overall algorithm's stability. Even if the stability is not guaranteed, it is possible to compute how accurately the solution found is, through a-posteriori stability test.

Theorem 1. *Let $A \in \mathbb{R}^{m\times m}$, $b \in \mathbb{R}^m$, and x be the solution of $Ax = b$. For a given $\tilde{x}$, define $r = A\tilde{x} - b$. Then,*

$$\frac{||x - \tilde{x}||}{||x||} \leq K(A)\frac{||r||}{||b||}$$

Considering the Least Squares problem, let $A = Q_0R_0$ be a thin QR factorization. Let $r_0 = Q_0^T(A\tilde{x} - b)$. Then, $\tilde{x}$ is the exact solution of the LS problem

$$min||Ax - (b + Q_0r_0)||$$

Thus

$$\frac{||\tilde{x} - x||}{||x||} \leq K_{rel}(LS, b)\frac{Q_0r_0}{||b||} = K_{rel}(LS, b)\frac{||r_0||}{||b||}$$

Obtaining a solution $\tilde{x}$ for which the residual $\frac{||r||}{||b||}$ is of the order of machine precision, then the solution is accurate as possible.[2]

4 Convergence

In order to study the convergence of our algorithm, it is necessary to consider the residual. As state in [5], GMRES converges monotonically:

$$||r_{n+1}|| \leq ||r_n|| \tag{6}$$

The reason is that $||r_n||$ is as small as possible for the Krylov subspace, enlarging K_n to the space K_{n+1}, the residual norm can only decrease or at worst remains unchanged. This means that the error cannot increase, at most can stay equal.

Assuming to operate in the absence of rounding errors, the norm of the residual at the m-th step would be $||r_m|| = 0$.

In order to estimate the convergence rate of the GMRES it is possible to notice that:

$$\min_{x \in K(A,b)} ||Ax - b|| = \min_{p\ of\ degree \leq n-1} ||Ap(A)b - b||$$
$$= \min_{p\ of\ degree \leq n-1} ||(Ap(A) - I) \cdot b|| \leq (\min_{p\ of\ degree \leq n-1} ||Ap(A) - I||) \cdot ||b||$$

where $p(A)$ is a polynomial of degree $\leq n-1$ such that $p(0) = 1$.

The objective is to determine an upper bound for:

$$\min_{p\ of\ degree \leq n-1} ||Ap(A) - I|| \tag{7}$$

Then, assuming that A is diagonalizable it is possible to write A as

$$A = V \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} V^{-1}$$

considering the Formula 7

$$Ap(A) - I = V \begin{pmatrix} \lambda_1 p(\lambda_1) - 1 & & \\ & \ddots & \\ & & \lambda_n p(\lambda_n) - 1 \end{pmatrix} V^{-1}$$

This means that it is possible to rewrite it as:

$$\min_{p\ of\ degree \leq n-1} ||Ap(A) - I|| = \min_{p} ||V \begin{pmatrix} \lambda_1 p(\lambda_1) - 1 & & \\ & \ddots & \\ & & \lambda_n p(\lambda_n) - 1 \end{pmatrix} V^{-1}||$$

$$\leq \min_{p} ||V|| \cdot || \begin{pmatrix} \lambda_1 p(\lambda_1) - 1 & & \\ & \ddots & \\ & & \lambda_n p(\lambda_n) - 1 \end{pmatrix} || \cdot ||V^{-1}||$$

$$\leq K(V) \cdot || \min_{p} \begin{pmatrix} \lambda_1 p(\lambda_1) - 1 & & \\ & \ddots & \\ & & \lambda_n p(\lambda_n) - 1 \end{pmatrix} ||$$

$$= K(V) \left(\min_{p\ of\ degree \leq n-1} \max_{\lambda_i} |\lambda_i p(\lambda_i) - 1| \right)$$

It is possible to conclude that GMRES converges to the exact solution when the input matrix A has very few distinct eigenvalues. Moreover, this happens even when there are few clusters of eigenvalues of A.

The GMRES convergence is not only influenced by this factor but there could be several aspects to take into account.
For example, the conditioning of the input matrix (except for normal matrices) and the machine precision round-off can influence the convergence behaviour of the method.

5 Implementation

Our implementation of GMRES for the problem A1 can be summarized in the following pseudo-algorithm:

Algorithm 3 Our implementation of GMRES

1: **Input:** $D \in \mathbb{R}^m$, $E \in \mathbb{R}^{(n-1)\times(m)}$
2: $dim = m + n - 1$
3: **for** $k = 1, \ldots, dim$ **do**
4: Apply Lanczos method ▷ Algorithm 2
5: Apply old rotations on the new column of H_k
6: Apply current rotation on the new column of H_k
7: Check residual for early stopping
8: **end for**

Our main function is called "**our_gmres.m**" and contains the MATLAB implementation of the pseudo-algorithm, using this signature:

```
our_gmres(D, E, b, starting_point, threshold, reorth_flag)
```

where:

- D: is the input diagonal vector.
- E: is the input matrix.
- b: is the vertical input vector $\tilde{b}$.
- *starting_point*: is the vertical starting point of GMRES.
- *threshold*: is the threshold required.
- *reorth_flag*: if true, apply the double reorthogonalization during lanczos

In addition to implementing the optimizations proposed in 2.4.4, our implementation contains the following optimizations:

1. D is a diagonal matrix, so instead of saving the entire matrix in memory, only a single vector was saved, so all operations in which D appears were adapted to handle this optimization.
2. Since the input matrix is $J = \begin{bmatrix} D & E^T \\ E & 0 \end{bmatrix}$, using block operations it is possible to save iterations, avoid operations with the bottom right block since it is a block consisting of all zero. This optimization is performed twice:
   - **The computation of the residual**
     Since the residual should be calculated as $r = \tilde{b} - Jx$, it is possible to use blocks to optimize this calculation:

$$r = \tilde{b} - Jx = \begin{bmatrix} b \\ c \end{bmatrix} - \begin{bmatrix} D & E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix} - \begin{bmatrix} D \odot x_1 + E^T * x_2 \\ E * x_1 \end{bmatrix} \tag{8}$$

     The element-wise multiplication is needed since D is a vector.

- **Lanczos**
  The same idea can be used to optimize the instruction $v = Jq_n$ inside the Lanczos iteration. So, the vector q_n is split into two subvectors: q_1 and q_2; q_1 contains the first $|D|$ elements, while q_2 contains elements from $|D| + 1$ to the end. Then, v can be calculated using this block operation:

$$v = \begin{bmatrix} D \odot q_1 + E^T * q_2 \\ E * q_1 \end{bmatrix} \tag{9}$$

3. In order to optimize the memory space, instead of saving the rotation matrices (where each one contains four elements), it is possible to keep only c and s for each rotation matrix. Hence, by adopting this approach, 50% of the items are stored in memory.

4. Another included optimization is the implementation of the "*patient*" mechanism, which tracks the number of consecutive iterations where the residual does not significantly change. If the residual has not improved over the last ten iterations, the algorithm terminates and outputs the number of iterations achieved before termination.

Considering that $E \in \mathbb{R}^{(n-1)\times m}$ is a sparse matrix and $D \in \mathbb{R}^{m\times m}$ is handled as a vector, the time complexity of the total algorithm can be studied in the following way:

1. Outer Loop: $O(k)$
2. Lanczos with reorthogonalization: $O(k \times (m + n))$
3. Apply Rotations: $O(1)$
4. Solve Least Square: $O(k^2)$
5. Compute the residual: $O(m \times n)$

Hence, the overall complexity is: $O(k^2 \times (m + n))$; where k is the number of iterations.

Instead, as regards the space complexity, our algorithm needs to keep in memory, in addition to the inputs D and E, the matrices Q and H. The matrix Q has dimension $((m + n), k)$ while H has dimension $(k + 1, k)$, therefore the space complexity is limited by $O(k \times (m + n))$.

6 Preconditioning

As shown in the previous chapters, the convergence rate of the method depends on the properties of the input matrix, in particular, if there are few distinct eigenvalues the method converges faster.
A way to manipulate the eigenvalues of the input matrix is to select a preconditioning matrix P in order to accelerate the convergence of the algorithm.

Considering the original problem

$$Jx = b \tag{10}$$

for any non-singular matrix $P \in \mathbb{R}^{(m+(n-1))\times(m+(n-1))}$, the system

$$P^{-1}Jx = P^{-1}b$$

has the same solution.
If P is well chosen, the second problem achieves convergence in a faster way with respect to the first problem.
In the case where the input matrix is symmetric, to maintain this symmetry, it is necessary to apply the preconditioning matrix P to both sides, resulting in the following equivalent system:

$$PJP^T(P^{-T}x) = Pb$$

The problem A2 requires that the preconditioning matrix must be defined as:

$$P = \begin{bmatrix} D & 0 \\ 0 & -S \end{bmatrix}$$

where S is the Schur complement preconditioner $S = -ED^{-1}E$.

The Incomplete Cholesky factorization is an approximation technique used to efficiently decompose a symmetric positive definite matrix into a product of a lower triangular matrix and its transpose. A2 requires that P must be factorized with this technique such that

$$P \approx R^T R$$

Hence, it is possible to apply split preconditioning to the matrix J in the following way in order to keep the symmetry of the input matrix:

$$R^{-T}JR^{-1}Rx = R^{-T}b$$

This formulation is equivalent to the Formula 10.

Given the structure of matrix P, using Incomplete Cholesky factorization on P is equivalent to performing it on the individual blocks of the matrix, as shown in the Formula 11.

$$ichol(P) = \begin{bmatrix} \sqrt{D} & 0 \\ 0 & ichol(-S) \end{bmatrix} \tag{11}$$

It is worth noting that since D is diagonal, applying factorization on it is equivalent to performing the square root of its elements.

Regarding the implementation, to further optimize the algorithm, the matrix P is not created explicitly but S and D factorized are used in the blocks operations.
Therefore, the new signature of the function is defined as:

```
our_gmres(D, E, S, b, starting_point, threshold, reorth_flag)
```

where S is the Schur complement preconditioner, if it is set to **NaN**, no preconditioning is applied.

In particular, only two operations are affected by the use of preconditioning:

1. **Lanczos**
   The computation of $v = Jq_n$, already optimized through blocks operation as shown in the Formula 9, has been extended considering the preconditioning in the following way:
   $$v = \begin{bmatrix} (D_{chol}^{-1} \odot (D * (D_{chol}^{-1} \odot q_1))) + (E^T * (S_{chol}^{-1} * q_2)) \\ \\ S_{chol}^{-T} * (E * (D_{chol}^{-1} \odot q_1)) \end{bmatrix}$$
   where D_{chol} and S_{chol} are the matrices D and S factorized with the incomplete Cholesky factorization.

2. **The computation of the residual**
   In a similar way, the optimized blocks operation for the computation of the residual (Formula 8) has been extended as:
   $$r = \begin{bmatrix} D_{chol}^{-1} \odot b \\ \\ S_{chol}^{-T} * c \end{bmatrix} - \begin{bmatrix} (D_{chol}^{-1} \odot (D * (D_{chol}^{-1} \odot x_1))) + (E^T * (S_{chol}^{-1} * x_2)) \\ \\ S_{chol}^{-T} * (E * (D_{chol}^{-1} \odot x_1)) \end{bmatrix}$$

In both cases the inverse of the matrices has not been computed explicitly, the backslash MATLAB operator has been used instead.

Considering the complexities of the algorithm:

- The time complexity analysis is similar to the previous one (without taking into account the initialization and factorization of S):

  1. Outer Loop: $O(k)$
  2. Lanczos with reorthogonalization: $O((k \times (m+n)) + (n \times m))$
  3. Apply Rotations: $O(1)$
  4. Solve Least Square: $O(k^2)$
  5. Compute the residual: $O((m \times n) + (n \times m))$

  The new terms in "Lanczos" and "Compute residual" come from the blocks multiplication needed for the computation of v and r. Although these new terms have been introduced, the asymptotic time complexity remains the same: $O(k^2 \times (m+n))$.

- The space complexity remains unchanged since only the $D_{chol} \in \mathbb{R}^{m \times 1}$ vector and the matrix $S \in \mathbb{R}^{m \times n}$ are saved.

7 Experiments

This section describes the experiments that were carried out on the modified algorithm. These experiments examine the performances of the algorithm from various points of views.
To make the experiments replicable, a seed was set for the random generation of values of 42. Furthermore, all the experiments were conducted using a threshold of $1e-10$ as the convergence criterion.
The experiments that have been conducted for problem A1 are:

- Experiment 1: Test about convergence
- Experiment 2: Test on initialization of D
- Experiment 3: Comparisons with MATLAB methods

Instead, with regard to problem A2 the following experiments were conducted:

- Experiment 4: Test about convergence with preconditioning
- Experiment 5: Comparisons with MATLAB Preconditioned methods

All the experiments have been executed on an Intel Core i7-7700HQ 2.8 GHz Quad-Core with 16 GM of RAM.

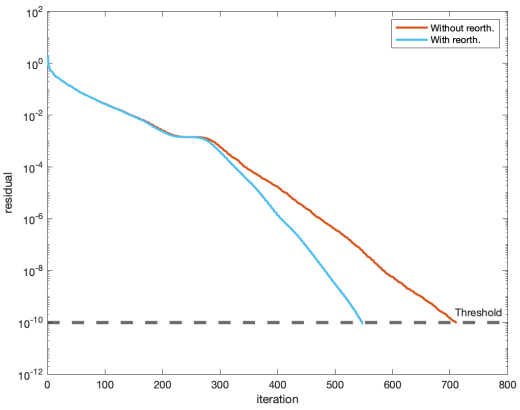
7.1 Experiment 1: Test about convergence

In this test, the correctness of the algorithm has been verified by analyzing its behaviour under varying input problems and examining the properties of convergence with and without reorthogonalization.

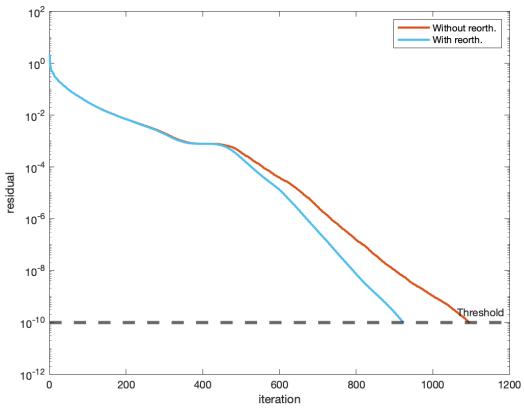
As stated in Chapter 4, the residual from a generic step k to step $k+1$ does not increase but remains the same at most. Therefore, the residuals obtained in the various iterations are analyzed, and it is expected that they exhibit a decreasing trend or, at least, remain constant. Additionally, the impact of applying reorthogonalization to the algorithm has been examined. It is anticipated that the version with reorthogonalization should converge in fewer steps, as it should provide greater stability to the iterative method.
The D sub-matrix has been generated randomly in $[0, 1]$ for all the problems.

# **nodes**	# **edges**	**reorth.**	# **iterations**	**res. rel.**	**time (sec)**
256	2048	No	711	9.36E-11	6.934
256	2048	Yes	548	8.46E-11	3.847
1024	8192	No	1095	9.33E-11	43.614
1024	8192	Yes	923	9.77E-11	37.553
4096	32768	No	1148	9.92E-11	138.734
4096	32768	Yes	1020	9.40E-11	150.573

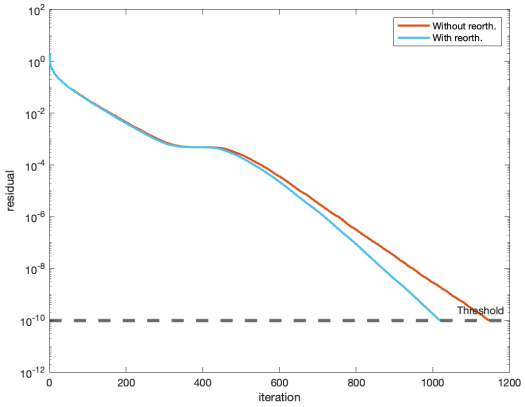
Table 1: Results with different problem sizes



(a) 256 nodes and 2048 edges



(b) 1024 nodes and 8192 edges



(c) 4096 nodes and 32768 edges

Figure 1: Residual through iterations for all the problems

The results are as expected. The reorthogonalization provides a more stable algorithm and leads to faster convergence, as shown in Table 1. For each problem size, the algorithm with reorthogonalization performs better than the one without reorthogonalization. In the Figure 1 the residual has a similar trend for the first iterations, but after that, it is possible to see a greater slope in the plot of the algorithm with reorthogonalization. It can be concluded that reorthogonalization is essential to improve the convergence of the algorithm, although it has a higher computational cost.

7.2 Experiment 2: Test on initialization of D

In this experiment, the impact of the initialization of D on the result has been analyzed. To do so, D is generated using different approaches:

- D generated randomly in [0,1]
- D generated randomly in [4,5]
- D as Identity matrix
- D with two distinct values
- D with five distinct values
- D with all distinct values

The approaches listed above have been selected also basing on the following theorem [6]:

Theorem 2. *Suppose $A \in R^{n\times n}$ is symmetric. Let $B \in R^{m\times m}$ with $m < n$ be a principal submatrix (obtained by deleting both i-th row and i-th column for some values of i). Suppose A has eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$ and B has eigenvalues $\beta_1 \leq \cdots \leq \beta_m$. Then*

$$\lambda_k \leq \beta_k \leq \lambda_{k+n-m}$$

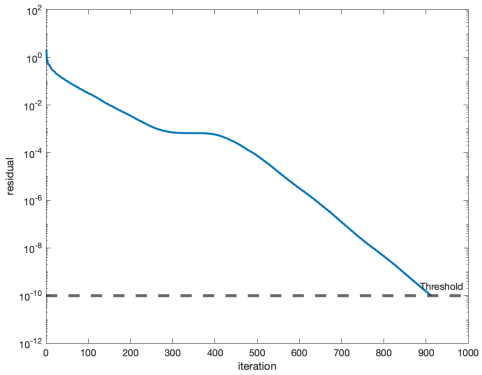
for $k = 1, \ldots, m$. And if $m = n - 1$

$$\lambda_1 \leq \beta_1 \leq \lambda_2 \leq \beta_2 \leq \cdots \leq \beta_{n-1} \leq \lambda_n$$

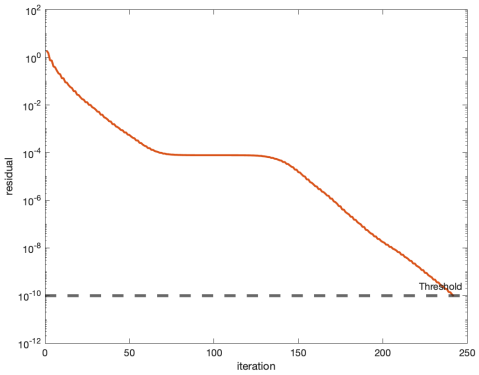
From this statement, it is clear that there is an interconnection between the eigenvalues of J and the ones of D. This property has been exploited to verify the other convergence property stated in the Chapter 4 that state that if there are k few eigenvalues or cluster of eigenvalues the algorithm converges in at most k steps. So it is expected that the problem with D initialized as all ones should converge with the lowest number of iterations. Furthermore, it is expected that the number of iterations should increase as the number of distinct values in D increases.
In order to have comparable results, the different initialization approaches have been tested with the same graph with 1024 nodes and 8192 edges.

init mode	**cond. number (J)**	**det. (J)**	**# iterations**	**res. rel.**
random in [0,1]	3.97E+03	0	911	9.94E-11
random in [4,5]	2.87E+04	-Inf	242	9.43E-11
identity	7.35E+03	-Inf	128	8.68E-11
2 distinct values	1.56E+04	-Inf	185	7.66E-11
5 distinct values	4.88E+04	-Inf	266	8.45E-11
all different values	1.64E+11	-Inf	9214	1.43E-10

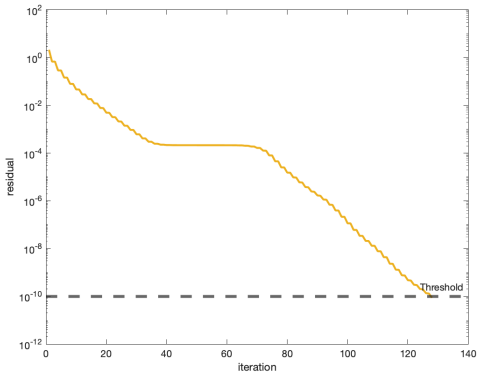
Table 2: Results with different initialization of D



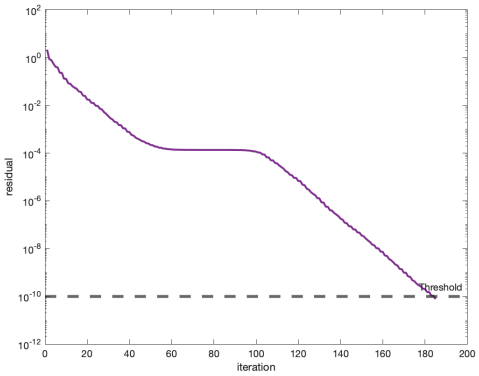
(a) Random in (0,1)



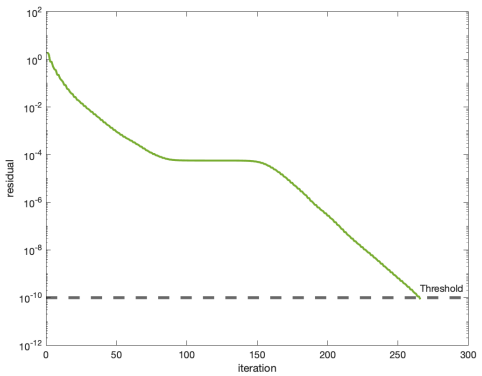
(b) Random in (4,5)



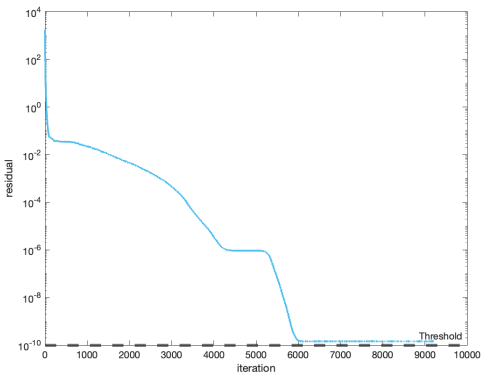
(c) Identity



(d) 2 distinct values



(e) 5 distinct values



(f) All different values

Figure 2: Residual through iterations for all the initialization of D

In general, the results shown in Table 2 and Figure 2 are as expected. Increasing the number of distinct values in D increases the number of iterations required to reach the convergence. It is possible to notice that if D is initialized randomly in $[0, 1]$ the matrix J becomes singular. It happens because the values are more likely to be zero (or near zero) and this could lead to a singular matrix considering that:

$$det(J) = det(D) * det(0 - E^T * D^{-1} * E)$$

If D is initialized with all different values, the condition number of J increases significantly. In conclusion, considering this problem it is worth to initialize D as all ones in order to have a faster convergence.

7.3 Experiment 3: Comparisons with MATLAB methods

Given that the custom version of the GMRES is not only optimized in terms of operations but it also includes the QR factorization of the H_n matrix at each iteration, it could be interesting to compare the performances of the implemented algorithm with respect to the MATLAB native methods for GMRES and MINRES.

MINRES has also been taken into account because the optimizations performed to the implemented algorithm lead to the MINRES implementation.

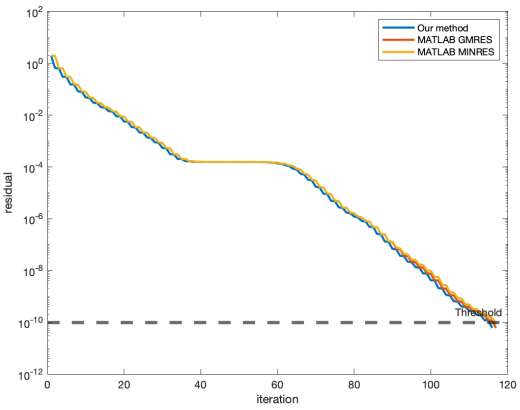
The experiment has been executed using three different problem sizes:

- 256 nodes and 2048 edges
- 1024 nodes and 8192 edges
- 4096 nodes and 32768 edges

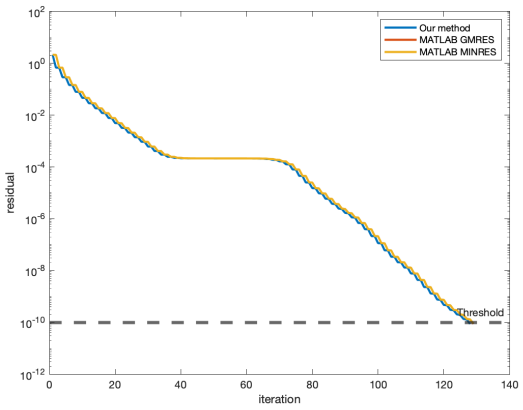
D has been initialized to all ones and each problem has been executed five times averaging the completion time.

	nodes	**edges**	**#iterations**	**res. rel.**	**time (sec)**
Custom GMRES	256	2048	116	6.095E-11	0.093
	1024	8192	128	8.676E-11	0.547
	4096	32768	134	9.463E-11	2.693
Native GMRES	256	2048	116	6.095E-11	1.028
	1024	8192	128	8.676E-11	15.530
	4096	32768	134	9.463E-11	247.120
Native MINRES	256	2048	116	8.752E-11	0.949
	1024	8192	128	9.028E-11	15.191
	4096	32768	138	7.852E-11	248.822

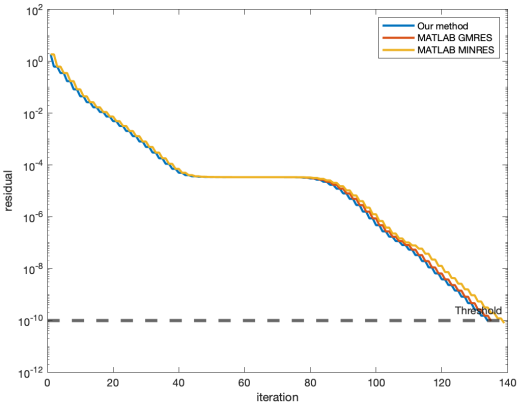
Table 3: Results with different implementations



(a) 256 nodes and 2048 edges



(b) 1024 nodes and 8192 edges



(c) 4096 nodes and 32768 edges

Figure 3: Residual through iterations for all the problem sizes

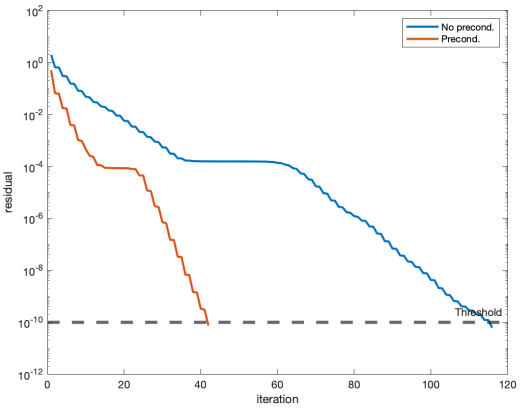
Looking at the Table 3 and Figure 3, it is possible to see that the Custom GMRES behaves the same way compared to the native MATLAB algorithms. It is worth noting that the Custom GMRES is faster than the other approaches, this is due to the optimizations that have been implemented in the Custom GMRES that take into account the particular structure of J.

7.4 Experiment 4: Test about convergence with preconditioning

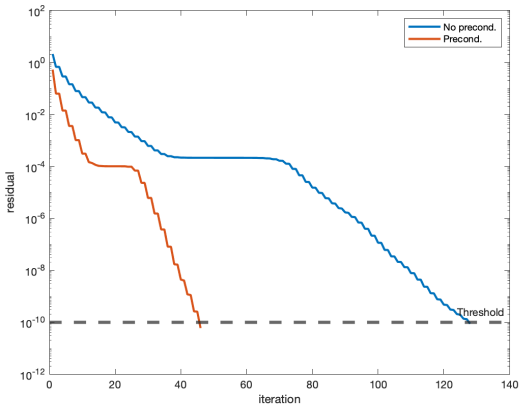
Problem A2 requires the inclusion of a preconditioning matrix in the algorithm. Hence, it might be interesting to compare the performance of the algorithm with and without the use of preconditioning.
It is expected that the preconditioned approach outperforms the classic approach.
The initialization of the D sub-matrix is with all ones and three different problem sizes have been tested.

	precond.	**#iterations**	**res. rel.**
256 nodes/4096 edges	No	116	6.095E-11
	Yes	42	7.428E-11
1024 nodes/8192 edges	No	128	8.676E-11
	Yes	46	5.919E-11
4096 nodes/32768 edges	No	134	9.464E-11
	Yes	48	6.848E-11

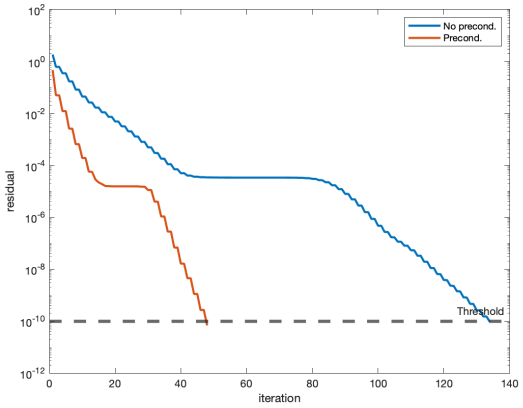
Table 4: Results with different problem sizes



(a) 256 nodes and 2048 edges



(b) 1024 nodes and 8192 edges



(c) 4096 nodes and 32768 edges

Figure 4: Residual through iterations for all the problem sizes

It is evident that the preconditioning helps the convergence, in fact looking at Table 4 and Figure 4 it is possible to see that the execution with the preconditioning has a smaller number of iterations.

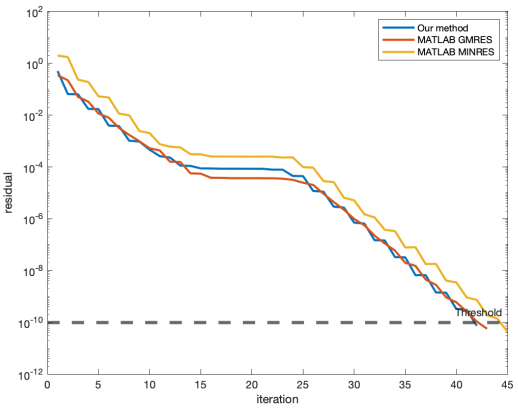
7.5 Experiment 5: Comparisons with MATLAB Preconditioned methods

To get a complete picture of the performances of the algorithm a comparison with the MATLAB native methods on the preconditioned problem should be performed. The experiment has been performed following the methodology applied in Experiment 3.

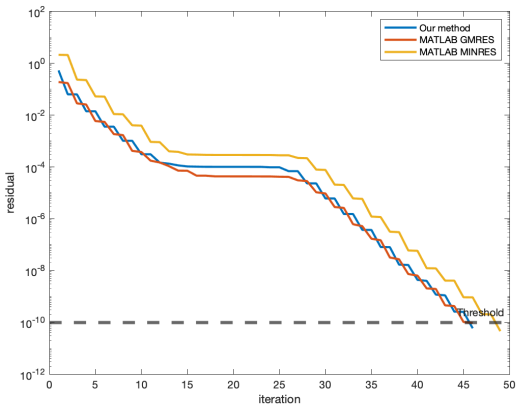
The time columns are expressed in seconds.

	nodes	**edges**	**# it.**	**res. rel.**	**exc. time**	**init. S**	**init. P**
Custom GMRES	256	2048	42	7.428E-11	0.023	0.040	–
	1024	8192	46	5.918E-11	0.244	1.221	–
	4096	32768	48	6.848E-11	3.476	56.475	–
Native GMRES	256	2048	42	5.679E-11	0.036	–	0.094
	1024	8192	45	9.510E-11	0.055	–	2.008
	4096	32768	46	9.733E-11	0.179	–	102.395
Native MINRES	256	2048	44	4.314E-11	0.006	–	0.094
	1024	8192	48	4.570E-11	0.021	–	2.008
	4096	32768	50	5.999E-11	0.078	–	102.395

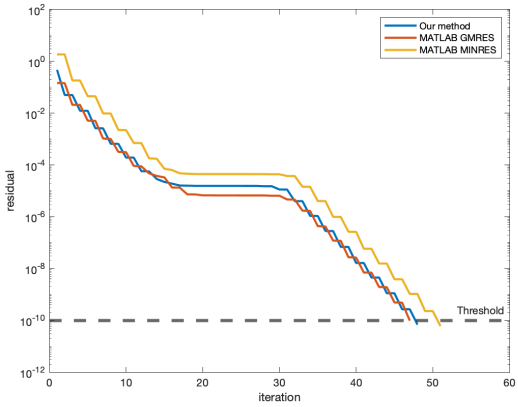
Table 5: Results with different implementations



(a) 256 nodes and 2048 edges



(b) 1024 nodes and 8192 edges



(c) 4096 nodes and 32768 edges

Figure 5: Residual through iterations for all the problem sizes

The number of iterations reported in Table 5 shows that the Custom GMRES achieves similar performance to native methods. As can be seen from Figure 5 the residual trend is similar between Custom GMRES and Native GMRES while that of Native MINRES decreases more slowly. We can see that as the problem size increases, the native methods converge in less time than Custom GMRES. Also taking into account the creation of the matrix preconditioner, our algorithm achieves times that are approximately half of those obtained with the other methods.

8 Conclusions

In this study, a custom variant of the GMRES algorithm has been developed, specifically tailored to problems exhibiting the same structure as the input matrix J.
This work started with an initial phase focused on the theoretical aspects of this iterative approach.
From an initial analysis, it was deduced that the matrix J is symmetric.
The exploitation of this symmetry property has led to substantial enhancements in algorithmic efficiency. Moreover, in accordance with requirements, QR factorization was applied to the matrix H_n during each iteration of the algorithm.

To evaluate the algorithm's performance a series of experiments were conducted.
The outcomes confirm the initial assertions and they have indicated the central role of reorthogonalization in ensuring more stability and, consequently, an accelerated convergence.

Additionally, from the experiments conducted on the initialization of the submatrix D, it becomes evident that the eigenvalues of this submatrix influence the eigenvalues of matrix J. Hence, initializing D with a small set of distinct eigenvalues can facilitate expedited convergence of the algorithm.

Lastly, the implementation of preconditioning has demonstrated its utility in enhancing algorithmic convergence, although at the cost of additional computational effort.
Hence, it is fundamental to assess the trade-off between convergence speed and computational time.

References

[1] James W. Demmel. *Iterative Methods for Eigenvalue Problems.*

[2] Poloni F.G. Slides about stability and residual.

[3] Poloni F.G. Slides about stability least squares.

[4] Jim Lambers. Section 4.2.1: Given rotations, February 2021.

[5] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra.* SIAM, 1997.

[6] David P. Williamson. Spectral graph theory - lecture 4, September 2016.