Computational Mathematics for learning and data analysis

Non-ML Project 27

Silvio Martinico, Roberto Esposito

December 19, 2021

Abstract

In this report we show the resolution of a problem that arises as the KKT system of the convex quadratic separable Min-Cost Flow Problem. In particular, we approximate the solution of a sparse linear system in two ways: first using the *generalized minimal residual method* (**GMRES**) in the standard way and then using it with the so-called *Schur complement preconditioner*.

Contents

1 The problem

The problem that we are going to analyze is the following: given 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 is a diagonal strictly positive matrix (i.e., $D = diag(D) > 0$) and E is the edge-node adjacency matrix of a given directed graph $G = (N, A)$.
The problem has been solved in two different ways:

A1 GMRES, where the internal problems $\min ||H_n y - ||b||e_1||$ have been solved by updating the QR factorization of H_n at each step;

A2 The same GMRES as A1, but using the so-called Schur complement preconditioner:

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

where S is either $S = -ED^{-1}E^T$ or a sparse approximation of it.

2 Theoretical Aspects

Before we start we give some notation:

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

where $D \in \mathbb{R}^{d \times d}$, $E \in \mathbb{R}^{\tau \times d}$, $x, b \in \mathbb{R}^d$, $y, c \in \mathbb{R}^\tau$ and

$$D = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_d \end{bmatrix}.$$

So, we define $\nu := d + \tau$ and we can write our problem as:

$$A\widetilde{x} = \widetilde{b} \equiv \begin{cases} Dx + E^T y = b \\ Ex = c \end{cases}.$$

Observation 2.1. A is symmetric. In fact, the diagonal blocks (D and 0) are symmetric, while the anti-diagonal blocks (E and E^T) are one the transpose of the other, so the overall matrix A is symmetric.

2.1 Customized GMRES

At first, we are going to solve the problem A1.

We can observe that, taking $e = (1, \dots, 1)^T$ and resembling the fact that each column of E has only two elements different from 0, which are respectively equal to 1 and -1, we have:

$$z = \begin{bmatrix} 0 \\ e \end{bmatrix} \implies z^T A = 0\,.$$

So, since the matrix A is singular, we are not sure that a solution exists. Furthermore A is expected to be very large so we are not searching for the exact vector inside the entire space, but we are looking for an approximation of it inside the Krylov subspace:

$$\mathcal{K}_n(A, \widetilde{b}) := \text{Span}\left(\widetilde{b},\ A\widetilde{b}, ...,\ A^{n-1}\widetilde{b}\right).$$

For this purpose, we exploit the Arnoldi process that finds an orthonormal basis $\{q_1, ..., q_n\}$ of $\mathcal{K}_n(A, \widetilde{b})$.

So, we want to find the $\widetilde{x} \in \mathcal{K}_n$ which minimizes $||A\widetilde{x} - \widetilde{b}||$; this problem is equivalent to find $\widetilde{y} \in \mathbb{R}^n$ which achieves the minimum of $\left|\left|H_n\widetilde{y} - ||\widetilde{b}||e_1\right|\right|$, where $H_n \in \mathbb{R}^{(n+1)\times n}$ arises from the n-th iteration of the Arnoldi process and its components are the $\beta_{i,j}$'s, that are the coefficients in:

$$Aq_j = \beta_{1,j}q_1 + ... + \beta_{j,j}q_j + \beta_{j+1,j}q_{j+1}\ .$$

Let Q_n be the following matrix:

$$Q_n = \left[\begin{array}{c|c|c} & & \\ q_1 & \cdots & q_n \\ & & \end{array}\right].$$

Let $Q = Q_\nu$ and H be the matrix H_ν, but without the last row of zeros, i.e. the matrices given from full Arnoldi iteration. Note that the matrix Q is orthogonal since its columns are orthonormal. We can state the following:

Lemma 2.1. H is symmetric.

Proof:

$$AQ = QH \implies A = QHQ^T \implies$$

$$\implies A^T = (QHQ^T)^T = QH^TQ^T \overset{A=A^T}{\implies}$$

$$\overset{A=A^T}{\implies} QHQ^T = QH^TQ^T \implies H = H^T \implies$$

$$\implies \text{H is symmetric.}$$

□

Proposition 2.1. H is tridiagonal.

Proof:
H is an upper-Hessenberg matrix, i.e. H has the following form:

$$H = \begin{bmatrix} * & * & \cdots & * & * \\ * & * & \cdots & * & * \\ 0 & * & \cdots & * & * \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & * & * \end{bmatrix}$$

From **Lemma (2.1)** H is also symmetric so it must be a tridiagonal matrix, that is:

$$H = \begin{bmatrix} * & * & 0 & \cdots & 0 \\ * & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & * \\ 0 & \cdots & 0 & * & * \end{bmatrix}$$

□

Now, H_n is none other than the submatrix of H obtained by taking the first $n+1$ rows and the first n columns:

$$H_n = \left[\begin{array}{ccccc} * & * & 0 & \cdots & 0 \\ * & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & * \\ 0 & \cdots & 0 & * & * \\ \hline 0 & \cdots & \cdots & 0 & * \end{array}\right]$$

Now that we have explained where H_n comes from, we can rephrase our problem. The original problem was:

$$\min_{\widetilde{x} \in \mathcal{K}_n} ||A\widetilde{x} - \widetilde{b}||\,.$$

Since $\widetilde{x} \in \mathcal{K}_n$, $\widetilde{x} \in \text{Span}(q_1, ..., q_n)$ and $\forall\ i = 1, ..., n$ q_i is the i-th column of Q_n, we can write $\widetilde{x} = Q_n\widetilde{y}$ for some $\widetilde{y} \in \mathbb{R}^n$.

$$\min_{\widetilde{x} \in \mathcal{K}_n} \left|\left|A\widetilde{x} - \widetilde{b}\right|\right| \;=\; \min_{\widetilde{y} \in \mathbb{R}^n} \left|\left|AQ_n\widetilde{y} - \widetilde{b}\right|\right| \overset{(*)}{=} \min_{\widetilde{y} \in \mathbb{R}^n} \left|\left|Q_{n+1}H_n\widetilde{y} - \widetilde{b}\right|\right|$$

where the step marked with "(*)" comes from the fact that $AQ_n = Q_{n+1}H_n$ (because H_n is the matrix of the coefficients of $q_1, \ldots, q_j + 1$ w.r.t. Aq_j, that is the j-th column of AQ_n).

At this point, we recall that the columns of Q_{n+1} form a set of orthonormal vectors, so, we can complete Q_{n+1} to an orthogonal matrix adding some columns and then, multiply $Q_{n+1}H_n\widetilde{y} - \widetilde{b}$ by this matrix because the Euclidean norm is invariant under product by orthogonal matrices.

$$
\begin{aligned}
&= \min_{\widetilde{y}\in\mathbb{R}^n} \left|\left| \left[\, Q_{n+1} \mid \widehat{Q} \,\right]^T \left(Q_{n+1} H_n \widetilde{y} - \widetilde{b}\right) \right|\right| \\
&= \min_{\widetilde{y}\in\mathbb{R}^n} \left|\left| \begin{bmatrix} H_n\widetilde{y} - ||\widetilde{b}||e_1 \\ 0 \end{bmatrix} \right|\right| \qquad (1) \\
&= \min_{\widetilde{y}\in\mathbb{R}^n} \left|\left| H_n\widetilde{y} - ||\widetilde{b}||e_1 \right|\right| .
\end{aligned}
$$

2.1.1 QR-decomposition of H_{n-1}

Recalling that A1 asks us to exploit the QR decomposition of H_n to solve the problem, let's see how we build the factorization of H_{n-1}.
Using Householder reflectors M^i_{n-1} we gradually make H_{n-1} an upper triangular matrix:

STEP 1: Let x be the first column of H_{n-1}.

$$x := (H_{n-1})^1, \quad y := ||x||e_1, \quad v := x - y \quad x, y, v \in \mathbb{R}^n .$$

So, defining $M^1_{n-1} := I - \frac{2}{v^T v} vv^T$, we have:

$$M_{n-1}H_{n-1} = \left[\begin{array}{c|ccc} ||x|| & * & \dots & * \\ \hline 0 & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ 0 & * & \cdots & * \end{array}\right] \qquad (2)$$

STEP 2: We repeat the same procedure taking as the new x the vector highlighted in green in the matrix (2). So we have the new $x, y, v \in \mathbb{R}^{n-1}$ and we define:

$$M^2_{n-1} := \left[\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & I - \frac{2}{v^T v} vv^T & \\ 0 & & & \end{array}\right] .$$

And so on until we reach the step $n-1$.

STEP n − 1 :

$$M^{n-1}_{n-1} := \left[\begin{array}{ccc|cc} & & & 0 & 0 \\ & I & & \vdots & \vdots \\ & & & 0 & 0 \\ \hline 0 & \cdots & 0 & & \\ 0 & \cdots & 0 & V & \end{array}\right] ,$$

where $I \in \mathbb{R}^{(n-2)\times(n-2)}$ and $V := I - \frac{2}{v^T v} vv^T \in \mathbb{R}^{2\times 2}$.

Finally, we get this matrix:

$$\underbrace{M_{n-1}^{n-1} \cdot \ldots \cdot M_{n-1}^{1}}_{Q^T} H_{n-1} =: R\,,$$

where Q is an orthogonal matrix (since each $M_{n-1}^{i}\ \forall\ i = 1, ..., n-1$ is orthogonal) and R is an upper triangular matrix. So, we have the QR decomposition of H_{n-1}:

$$H_{n-1} = QR\,.$$

2.1.2 Updating QR-decomposition

Suppose we have $H_{n-1} = QR$ and we want to build the QR factorization of $H_n = \widetilde{Q}\widetilde{R}$. We want to exploit the relations between H_n and H_{n-1} to update the matrices Q and R so that we obtain $\widetilde{Q}$ and $\widetilde{R}$.

Let's take a look at the structure of H_n:

$$H_n = \left[\begin{array}{ccc|c} & & & 0 \\ & & & \vdots \\ & H_{n-1} & & 0 \\ & & & \beta_{n-1,n} \\ & & & \beta_{n,n} \\ \hline 0 & \cdots\cdots & 0 & \beta_{n+1,n} \end{array}\right].$$

If we take:

$$\underline{Q}^T = \left[\begin{array}{ccc|c} & & & 0 \\ & Q^T & & \vdots \\ & & & 0 \\ \hline 0 & \cdots & 0 & 1 \end{array}\right], \tag{3}$$

$\underline{Q}^T$ is still an orthogonal matrix and it holds:

$$\underline{Q}^T H_n = \left[\begin{array}{ccc|c} & & & \\ & Q^T H_{n-1} & & Q^T\underline{v} \\ & & & \\ \hline 0 & \cdots\cdots\cdots & 0 & \beta_{n+1,n} \end{array}\right], \qquad \underline{v} := \left[\begin{array}{c} 0 \\ \vdots \\ 0 \\ \beta_{n-1,n} \\ \beta_{n,n} \end{array}\right], \tag{4}$$

where $\underline{Q^T H_n}$ is not an upper triangular matrix yet. In order to make it an upper triangular matrix, we have to replace its last row with a row of zeros (because it is an $(n+1) \times n$ matrix).

But first, let's see who is $Q^T\underline{v}$:

$$Q^T = \left[\begin{array}{c|c|c} & & \\ p_1 & \cdots & p_n \\ & & \end{array}\right] \implies c := Q^T\underline{v} = \beta_{n-1,n} \cdot p_{n-1} + \beta_{n,n} \cdot p_n$$

At this point we rewrite $\underline{Q^T} H_n$:

$$\underline{Q^T} H_n = \left[\begin{array}{cccc|c} * & \cdots & \cdots & * & c_1 \\ 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & * & c_{n-1} \\ \hline 0 & \cdots & \cdots & 0 & c_n \\ 0 & \cdots & \cdots & 0 & \beta_{n+1,n} \end{array}\right] \quad .$$

So, we can just apply one Householder reflector to make the last column as we want. Taking:

$$x = \left[\begin{array}{c} c_n \\ \beta_{n+1,n} \end{array}\right], \qquad y = ||x|| \left[\begin{array}{c} 1 \\ 0 \end{array}\right], \qquad v = x - y \ , \tag{5}$$

we define:

$$M_n^n = \left[\begin{array}{ccc|cc} & & & 0 & 0 \\ & I & & \vdots & \vdots \\ & & & 0 & 0 \\ \hline 0 & \cdots & 0 & & \\ 0 & \cdots & 0 & V & \end{array}\right] \tag{6}$$

where $I \in \mathbb{R}^{(n-1)\times(n-1)}$ and $V := I - \frac{2}{v^T v} v v^T \in \mathbb{R}^{2\times 2}$.

So, by multiplying M_n^n and $\underline{Q^T} H_n$, we get:

$$\underbrace{M_n^n \, \underline{Q^T}}_{\widetilde{Q}^T} H_n = \left[\begin{array}{cccc|c} * & \cdots & \cdots & * & c_1 \\ 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & * & c_{n-1} \\ \hline 0 & \cdots & \cdots & 0 & ||x|| \\ 0 & \cdots & \cdots & 0 & 0 \end{array}\right] =: \widetilde{R} \tag{7}$$

So:

$$\widetilde{Q}^T H_n = \left[\begin{array}{ccc|c} & & & c_1 \\ & & & \vdots \\ & Q^T H_{n-1} = R & & \vdots \\ & & & c_{n-1} \\ & & & ||x|| \\ \hline 0 & \cdots\cdots\cdots & 0 & 0 \end{array}\right] .$$

At the end, multiplying on the left by $\widetilde{Q}$, we get: $H_n = \widetilde{Q}\widetilde{R}$.

2.1.3 Exploiting QR in Least Square Problem

Now that we have the QR-decomposition of H_n, let's see how to exploit it in order to improve our algorithm.

Let $H_n = QR$ be the factorization of H_n. R is an upper triangular matrix and, since it is not a square matrix, it must have some rows of zeros (in reality it is just one row, because $R \in \mathbb{R}^{(n+1)\times n}$). So we can write:

$$R = \left[\begin{array}{c} R_1 \\ \hline 0 \quad \cdots \quad 0 \end{array}\right], \qquad Q = \left[\begin{array}{c|c} Q_1 & q \end{array}\right],$$

where q is just a column, $Q_1 \in \mathbb{R}^{(n+1)\times n}$. Let $p \in \mathbb{R}^{n+1}$ be the first row of Q.

Returning to our least square problem:

$$\min_{y\in\mathbb{R}^n} \left|\left| H_n y - ||\widetilde{b}||e_1 \right|\right|,$$

we can rewrite the argument as:

$$\begin{aligned} \left|\left| H_n y - ||\widetilde{b}||e_1 \right|\right| &\overset{Q\in\mathcal{O}(n)}{=} \left|\left| Q^T(H_n y - ||\widetilde{b}||e_1) \right|\right| \\ &\overset{H_n=QR}{=} \left|\left| Q^T Q R y - ||\widetilde{b}|| Q^T e_1 \right|\right| \\ &\overset{Q\in\mathcal{O}(n)}{=} \left|\left| Ry - ||\widetilde{b}||p^T \right|\right| \\ &= \left|\left| \begin{bmatrix} R_1 y - ||\widetilde{b}||(p_1^n)^T \\ ||\widetilde{b}||q_1 \end{bmatrix} \right|\right|, \end{aligned} \tag{8}$$

where p_1^n is the vector of the first n components of p, q_1 is the first component of q and $\mathcal{O}(n)$ is the set of real orthogonal matrices and we applied this property two times: in the first time we used the fact that orthogonal matrices do not change the Euclidean norm and the second time that $Q \in \mathcal{O}(n) \implies Q^TQ = QQ^T = I$.

So, to find the vector which minimizes the norm in (8), we can notice that the bottom block does not depend on y, and so, its square is always in the sum of the norm. Then, for finding a solution, we can consider just the upper block.

At this point, the problem to solve is to find the vector y such that it realizes the minimum:

$$\min_{y\in\mathbb{R}^n} \left|\left| R_1 y - ||\widetilde{b}||(p_1^n)^T \right|\right|. \tag{9}$$

If R_1 is invertible, the minimum of the quantity in (9) is 0.

Let's recall the following proposition we have seen in class:

Proposition 2.2. R_1 is invertible $\iff$ H_n is full column rank.

For exploiting this proposition we now show that H_n is full column rank:

Proposition 2.3. H_n is full column rank.

Proof:
H_n is full column rank $\iff$ the columns of H_n are linearly independent. So, we show this last property.

We recall that H_n is a tridiagonal matrix, so, if in each column the sub-diagonal element is not 0, the columns of H_n are clearly linearly independent.
The sub-diagonal element of H_n in the j-th column is $\beta_{j+1,j}$, that is the coefficient of q_{j+1} in the writing of $A \cdot q_j$ w.r.t. the orthonormal basis $\{q_1, ..., q_{j+1}\}$. This coefficient is equal to 0 if and only if $Aq_j \in \mathcal{K}_{j+1}(A, b) = Span(b, Ab, \ldots, A^j b)$ and this is true if and only if we have a breakdown. But, in case of breakdown, the algorithm stops.

So, until we stop the algorithm, $\beta_{j+1,j} \neq 0$ and this proves the statement.

□

So, the vector we are looking for is:

$$y = R_1^{-1} \cdot ||\widetilde{b}||(p_1^n)^T \,. \tag{10}$$

3 Algorithms stability

In this section we will analyze the stability of our algorithms.

Lemma 3.1. The product by an orthogonal matrix is a backward stable operation.

Proof:
Let Q be an orthogonal matrix and let A be another matrix:

$$Q \odot A = QA + E \,,$$

where E is the forward error matrix.
So, assuming that $f(A) := Q \cdot A$ and that $y = QA$ is the correct output, we have:

$$\begin{aligned} \widetilde{y} &= Q \odot A = QA + E \\ &= Q(A + Q^T E) = f(\widehat{x}) \end{aligned}$$

and, in order to have the backward stability, we want that $||\widehat{x} - x|| = ||x|| \cdot O(u)$, where u is the machine precision.

$$\frac{||\widehat{x} - x||}{||x||} \;=\; \frac{||A + Q^T E - A||}{||A||} \;=\; \frac{||Q^T E||}{||A||}$$

$$\overset{Q \in \mathcal{O}(n)}{=} \frac{||E||}{||A||} \;=\; \frac{||A|| \cdot O(u)}{||A||} \;=\; O(u) \,,$$

where the fact that $||E|| = ||A|| \cdot O(u)$ has been proved in class and it is a consequence of the so-called *Rounding error bound*.

□

Proposition 3.1. The QR-decomposition is backward stable.

Proof:

Each step of the QR-decomposition of a matrix A is given by the product of that matrix by an Householder reflector, which is an orthogonal matrix. So, by the Proposition 3.1, each step of the factorization is backward stable.

If the k-th step of the QR factorization is:

$$Q_k A_{k-1} = A_k \,,$$

then, in machine arithmetic, it becomes:

$$\widetilde{A}_k = Q_k \odot \widetilde{A}_{k-1} = Q_k(\widetilde{A}_{k-1} + F_{k-1}) \,,$$

where $||F_{k-1}|| = ||\widetilde{A}_{k-1}|| \cdot O(u) = ||A|| \cdot O(u)$ (the last equality can easily be shown by induction on k).

So, analyzing each step:

$$\begin{aligned}
\widetilde{A}_1 &= \widetilde{Q}_1\big(A + F_0\big) \\
\widetilde{A}_2 &= \widetilde{Q}_2\big(\widetilde{A}_1 + F_1\big) = \widetilde{Q}_2\Big(\widetilde{Q}_1\big(A + F_0\big) + F_1\Big) = \widetilde{Q}_2\widetilde{Q}_1\Big(A + F_0 + \widetilde{Q}_1^T F_1\Big) \\
&\vdots \\
\widetilde{A}_{n-1} &= \widetilde{Q}_{n-1}\widetilde{Q}_{n-2}\cdots\widetilde{Q}_1 \underbrace{\Big(A + F_0 + \widetilde{Q}_1^T F_1 + \widetilde{Q}_2^T\widetilde{Q}_1^T F_2 + \dots\Big)}_{\widehat{x}} ,
\end{aligned}$$

where $||\widehat{x}||$ is smaller or equal than the sum of the norms of its terms $\widetilde{Q}_k^T \cdots \widetilde{Q}_1^T F_k$ (by triangular inequality) and each of them has norm equal to $||A|| \cdot O(u)$.

So, the norm of $\widehat{x}$ is $||A|| \cdot n \cdot O(u) = ||A|| \cdot O(u)$ and we can conclude like in the Proposition 3.1.

□

Our algorithm is divided into these steps:

1. **QR factorization** (of H_n): it's backward stable by Proposition 3.1;
2. **Upper triangular linear system resolution** ($R_1 y = ||\widetilde{b}||(p_1^n)^T$): we make this using the `MATLAB` backslash operator `mldivide`. In case of upper triangular systems, `mldivide` uses the backward substitution, which is backward stable as proved in [1].
3. **Arnoldi iteration**: since A is symmetric we used the Lanczos method that, due to the loss of orthogonality could have stability problems. As stated in [1], no straightforward theorem is known about the stability of Lanczos iteration. Nonetheless, this iteration is very useful in practice.

So, there are no guarantees on the stability of the overall algorithm, but most of the operations done are backward stable.

4 Algorithm Convergence

In this section we are going to analyze the convergence of the Customized GMRES.
Our main challenge is to solve a linear system $Ax = \widetilde{b}$, but, since A is a singular matrix, we can not know a prior if a solution exists. So, we can consider the associated Least Square Problem:

$$\min_{x\in\mathbb{R}^\nu} \left|\left| Ax - \widetilde{b} \right|\right| .$$

In order to study the convergence of our algorithm we introduce the (relative) *residual*:

$$r := \frac{\left|\left| Ax - \widetilde{b} \right|\right|}{\left|\left| \widetilde{b} \right|\right|} .$$

Using the Arnoldi method we solve the problem inside the Krylov subspace $\mathcal{K}_n(A, \widetilde{b})$. Now we show how to minimize the residual:

$$\min_{x\in\mathcal{K}_n} \frac{\left|\left| Ax - \widetilde{b} \right|\right|}{\left|\left| \widetilde{b} \right|\right|} \overset{(1)}{=} \min_{y\in\mathbb{R}^n} \frac{\left|\left| H_n y - ||\widetilde{b}|| e_1 \right|\right|}{\left|\left| \widetilde{b} \right|\right|}$$

$$\overset{(8)}{=} \min_{y\in\mathbb{R}^n} \frac{1}{\left|\left| \widetilde{b} \right|\right|} \cdot \left|\left| \begin{bmatrix} R_1 y - ||\widetilde{b}|| (p_1^n)^T \\ \\ ||\widetilde{b}|| q_1 \end{bmatrix} \right|\right|$$

$$\overset{(2.2)}{=} \min_{y\in\mathbb{R}^n} \frac{1}{\left|\left| \widetilde{b} \right|\right|} \cdot \left|\left| \begin{bmatrix} 0 \\ ||\widetilde{b}|| q_1 \end{bmatrix} \right|\right|$$

$$= \frac{\cancel{\left|\left| \widetilde{b} \right|\right|} \cdot |q_1|}{\cancel{\left|\left| \widetilde{b} \right|\right|}} = |q_1| .$$

So, the residual is given by $|q_1|$, i.e. the first component of the last column of Q (the orthogonal matrix in the QR-decomposition of H_n). Thus, until we conclude the Arnoldi iteration and we compute Q, we don't have any information about the residual.

4.1 Considerations on the residual

Since the residual at the step n of the Arnoldi process is given by $|q_1^n|$, let's see where it comes from.
Let $H_n = \widetilde{Q}\widetilde{R}$ and $H_{n-1} = QR$ the QR-decompositions of H_n and H_{n-1}.
The value q_1^n is the first component of the last column of $\widetilde{Q}$, so, it is given by the product in (7). Recalling that $\widetilde{Q}^T = M_n^n Q^T$ and the structure of these two matrices ((6) and (3)), we highlight the relation between the residual at the step n and at the step $n-1$:

$$q_1^n = \begin{bmatrix} 0 & \cdots & 0 & v_{2,1} & v_{2,2} \end{bmatrix} \left[\begin{array}{c} p \\ \hline 0 \end{array}\right] = v_{2,1} \cdot q_1^{n-1} ,$$

where $v_{i,j}$s are the elements of the matrix V that comes from (6).
So, we can notice that the residual r does not decrease only when $|v_{2,1}| \geq 1$.
Let v_1 and v_2 be the two components of the vector v; from (5) we have that:

$$|v_{2,1}| \;=\; \frac{|2v_1v_2|}{||v||^2} \;=\; \frac{2|v_1|\cdot|v_2|}{(v_1)^2+(v_2)^2}$$

and:

$$\begin{aligned}
|v_{2,1}| \;\geq\; 1 \;&\iff\; \frac{2|v_1|\cdot|v_2|}{(v_1)^2+(v_2)^2} \;\geq\; 1 \\
&\iff\; (v_1)^2+(v_2)^2-2|v_1|\cdot|v_2| \;\leq\; 0 \\
&\iff\; (|v_1|-|v_2|)^2 \;\leq\; 0 \\
&\iff\; (|v_1|-|v_2|)^2 \;=\; 0 \\
&\iff\; |v_1| = |v_2| \;.
\end{aligned}$$

Let's explore who are v_1 and v_2:

$$v \;=\; \begin{bmatrix} v_1 \\ \\ v_2 \end{bmatrix} \;=\; \begin{bmatrix} c_n - ||x|| \\ \\ \beta_{n+1,n} \end{bmatrix}$$

So:

$$\begin{aligned}
|v_1| \;=\; |v_2| \;&\iff\; \Big| c_n - ||x|| \Big| = \Big|\beta_{n+1,n}\Big| \\
&\iff\; \left| c_n - \sqrt{(c_n)^2+(\beta_{n+1,n})^2} \right| \;=\; \Big|\beta_{n+1,n}\Big| \;,
\end{aligned}$$

and, solving the equation, we get:

$$|v_1| = |v_2| \;\iff\; c_n = 0\,.$$

Since c_n is the n-th component of $c = \beta_{n-1,n}\cdot p_{n-1} + \beta_{n,n}\cdot p_n$, there are no theoretical reasons for it to be equal to 0. So, we expect that, in general, $c_n \neq 0$ and so that the residual decreases at each iteration of the Arnoldi process. We can also look at this result from a more abstract point of view: at each iteration of the Arnoldi process, we enlarge the Krylov subspace; in this way, in the worst case, we just keep the solution of the previous step and the residual does not improve, but surely it can't increase.

We want to observe that, even though a sequence is monotone decreasing, this does not guarantee its convergence to 0 (or to a very small number). However this result gives strength to our algorithm.

5 Algorithm Complexity

5.1 Space Complexity

Since the dimension of the vectors in our algorithm is equal to one of the dimensions of the matrices and we have a constant number of vectors, the space complexity is upper bounded by the dimensions of the matrices we used.

The input matrix A is a block matrix of dimension $\nu \times \nu$, so we should need $O(\nu^2)$ space to save it. However, the left-upper block D is a diagonal matrix, so we just save its diagonal that requires $O(d)$ space; the left-lower block E is a sparse matrix (with $2d$ non-zero elements), so we save it by the `MATLAB` compressed sparse column format, that occupies a space proportional to the number of non-zero elements of E, so, it takes $O(d)$ space. Furthermore, the right-upper block is just the transpose of E, so we don't store it; finally, the right-lower block is a block of zeros and we avoid storing it.
So, we don't save the whole matrix A and thus the required space for A is $O(d)$.

There are other matrices involved in our algorithm: Q_n and H_n, i.e. the matrices that comes from the n-th step of the Arnoldi process, and Q and R that arise from the QR-decomposition of H_n.

Since the dimension of the matrices Q_n and H_n increases at each step of the Arnoldi process and the maximum number of iterations m is fixed a prior, we initialize Q_n and H_n as zero matrices of dimensions, respectively, $\nu \times (m+1)$ and $(m+1) \times m$. So, we need $O(d \cdot m)$ space to save them ($m \leq \nu = d + \tau \leq 2d$ but, in practice, $m \ll \nu$). However, recalling Theorem 2.1, H_n is tridiagonal and symmetric, so we can store just two diagonals, at a space cost of $O(2m) = O(m)$. Even though in this way we optimize the storage space for H_n, the asymptotic cost for Q_n and H_n does not change since it is upper bounded by the space of Q_n.
The overall space complexity for these two matrices is $O(d \cdot m)$.

Finally, $Q \in \mathbb{R}^{(m+1)\times(m+1)}$ and $R \in \mathbb{R}^{(m+1)\times m}$, so, the required space is $O(m^2)$.

Summing up, the total space required by our algorithm is given by the sum of the space for A, Q_n, H_n, Q and R:

$$O(d) + O(d \cdot m) + O(m^2) \;=\; O(d \cdot m)\,.$$

5.2 Time Complexity

The algorithm starts with the Arnoldi process that consists of m iterations. Let's analyze the operations involved in a generic step k of the process.

First of all we generate a new vector obtained by the product between the matrix A and the vector q_k. For doing this we exploit the block structure of A in order to optimize the computation.

$$z := \begin{bmatrix} D & E^T \\ E & 0 \end{bmatrix} \begin{bmatrix} q_k^1 \\ q_k^2 \end{bmatrix} = \begin{bmatrix} d * q_k^1 + E^T q_k^2 \\ E q_k^1 \end{bmatrix}$$

where d is the diagonal vector of D and '*' indicates the component-wise product.
The time complexity of this product is $O(d)$ and it is given d products of real numbers ($O(d)$) plus two matrix-vector products where the matrices are sparse ($O(d)$).

The next step of the Customized GMRES is the computation of the $\beta_{i,k}$s. Since the matrix H_n is tridiagonal, we do not need to compute all the $k+1$ values, but just three of them. Furthermore, for $k > 1$, we can avoid the computation of $\beta_{k-1,k}$ thanks to the symmetry of H_n (Lemma 2.1). The time complexity of the calculation of the two values $\beta_{k+1,k}$ and $\beta_{k,k}$ is $O(d)$.

Now let's analyze the step where we update the QR-decomposition of H_k. The first operation we do in (4) is to compute $Q^T \underline{v}$ which costs $O(k)$ (due to the structure of $\underline{v}$). Then, in (5), we calculate x, y and v in constant time $O(1)$. Finally, we build $\widetilde{Q}^T = M_k^k \underline{Q}^T$; for this product we just need to multiply V by the last two rows of $\underline{Q}^T$ that has a cost of $O(k)$ (we do not save explicitly M_k^k).

So, the overall cost for a generic step is $O(d)$ and it is given by the calculation of z ($O(d)$), the calculation of the $\beta_{i,k}$s ($O(d)$) and the updating of the QR-decomposition of H_k ($O(k)$).
Since we have to repeat m times the step above, the final cost is $O(d \cdot m)$.

To compute the final solution we need to take into account even the operation described in (10), but we do not need to compute explicitly the inverse of R_1. In fact we use the `MATLAB` function `mldivide` which, in case of an upper triangular system, uses the backward substitution.

The backward substitution starts from the bottom ($i = 1$), and, at each step, it does $i - 1$ subtractions and 1 division, thus the number of operations in a generic step is i. Since the number of columns of R_1 is m, it iterates up to m, so we have:

$$\sum_{i=1}^{m} i = \frac{m \cdot (m+1)}{2} \ .$$

Hence the time complexity of the backward substitution is $O(\frac{m^2}{2} + \frac{m}{2}) = O(m^2)$.

Theorem 5.1. The time complexity of the Custom GMRES is $O(d \cdot m)$.

Proof:

The total cost is given by the cost of the Arnoldi process combined with the QR-decomposition ($O(d \cdot m)$) plus the cost to compute the final solution ($O(m^2)$).
The sum of these two costs is:

$$O(d \cdot m) + O(m^2) = O(d \cdot m) \ .$$

□

6 Implementation

In this section we describe our `MATLAB` functions and we explain how they work.

The main function is `main.m`, that takes as input *filename, mode, generate and distribution*, where *filename* is the path of the file containing the instance, *mode* is the execution mode, *generate* is a boolean variable which indicates if the function has to generate or to load the vector D and *distribution* is the probability distribution of data in the vector D.
The possible modes are:

- *precond*: it compares the preconditioned system and the original one;
- *minres*: it compares CustomGMRES and the `MATLAB` function `minres`;
- *residual*: it plots the relative residual (in log scale) w.r.t. the number of iterations;
- *rate*: it plots the sequence $\frac{r_{n+1}}{r_n}$ w.r.t. the number of iterations.

Instead, the (names of the) seven available distributions are:

- *Gamma*: Gamma Distribution with parameters 5 and 1;
- *Beta75*: Beta Distributions with parameters 0.75 and 0.75;
- *Beta44*: Beta Distributions with parameters 4 and 4;
- *Chi*: Chi Square Distribution with parameter 4;
- *MixBin*: A mixture of two Binomial Distribution with probability of success 0.7 and 20 trials;
- *Uniform*: Uniform distribution in the interval $(0, 1)$;
- *Ill*: A distribution which makes D ill-conditioned. It is done starting from a Uniform Distribution in the interval $(0, 1)$ and multiplying $2/5$ of the data for a amplification factor.

This function calls `find_sol()` with different values of the parameter m (the number of iterations of the Arnoldi process). This is done a number of times that depends on the dimension of the matrix A and the value of m is picked as `num_iterations(i)` each time.

At this point, we define `find_sol.m` that, given the parameters passed by `main()`, calls the function `arnoldiqr()` which returns the QR-decomposition of H_m (Q and R), and the matrix Q_{m+1} from the Arnoldi Process. The next step is to compute the solution of the smaller problem, via the backward substitution, in this way:

```
y = R(1:end-1,:) \ (Q(1, 1:end-1)*norm(b))';
```

as done in (10). From this, we compute the final solution: $x = Q_m \cdot y$. Furthermore, `find_sol()` computes the needed residual(s), execution time(s) and lower bound(s).

The function `arnoldiqr()`, the core of our algorithm, implements the Arnoldi Process and computes the QR-decomposition of H_k at each iteration. At the first time, it computes the factorization from scratch and then, in the next iterations, it exploits the decomposition obtained in the previous step to get the updated matrices.
The algorithm proceeds as the classical Arnoldi Process and, for updating the QR-factorization, it works as described in section 2.1.2. The matrix we want to compute is:

$$\widetilde{Q}^T = M_m^m \underline{Q}^T = \left[\begin{array}{c|c} S & \begin{matrix} 0 \\ w \;\; \vdots \\ 0 \end{matrix} \\ \hline V \cdot \underbrace{\left[\begin{array}{ccc|cc} & u & & a & 0 \\ 0 & \cdots & 0 & 0 & 1 \end{array}\right]}_{p} & \end{array}\right],$$

that is the orthogonal matrix of the QR-decomposition of the current step.
Given $\underline{Q}^T$, the only operation we need to do in order to get $\widetilde{Q}^T$ is the product in the bottom block. This is given by the following three lines:

```
p = [u zeros(k-1, 1); a 0; 0 1];
gamma = [v(1)^2*u v(1)*v(2)*u; v(1)^2*a v(1)*v(2)*a; v(1)*v(2) v(2)^2];
Q(1:k+1, k:k+1) = p-(2/(v_norm))*gamma;
```

7 Numerical Experiments

In this section we analyze the performance of our algorithm with different kinds of matrices.

7.1 Data Generation

Since the linear system we try to solve arises from the **KKT** system of the convex quadratic Min-Cost Flow Problem (MCF), the data that we generate must satisfy the conditions of this optimization problem.

To obtain reliable instances of the **MCF** problem, we used netgen, a library for generating network flow graph problems.
This framework takes in consideration several parameters; the most significant for our problem are:

- *seed*: the seed used for the random generation;
- *nodes*: the number of nodes of the graph;
- *sources*: the number of sources of the graph;
- *sinks*: the number of sinks of the graph;
- *density*: the number of edges of the graph;
- *mincost*: the minimum of the vector b;
- *maxcost*: the maximum of the vector b;
- *supply*: the supply of each node (it is negative if the node is a source, zero if the node is a transshipment node and positive if the node is a sink).

The instances we used for the tests come from the *Linear Min-Cost Flow Problems* section of the website CommaLAB UniPi. These instances include all the data we need, except for the diagonal matrix D. In addition, for all the instances we tested, the costs are integers in the range [1, 5000].

In each test we took the values of D according to different probability distributions: uniform, beta, chi square, gamma and other two distributions we have created: the first one is a mixture of binomials distributions, with two (different) local maxima instead of one, while the second one is a combination of two uniform distributions which values have different magnitudes, in order to obtain a more ill-conditioned matrix.

7.2 Convergence

In this section we study the behavior of the residual with respect to the number of iterations; furthermore, we observe how the rate of convergence changes as the number of iterations increases. For doing this, we take in consideration the instance `net12_8_1` and we analyze how the residual or the rate change as the distribution of the data in D changes. For generating the data in D according to different distributions, we used the `MATLAB` function `random()`.

In order to avoid irrelevant steps, which may blow up the computation time without improving enough the residual, we added a further stopping criterion based on the value $\Delta_i := r_{m_i} - r_{m_{i-1}}$.

7.2.1 Residual

We plot, in a semi-log scale, the number of iterations on the x-axis and the residual on the y-axis. In order to identify the trend of the residual, we plot also the line that best fits the points, obtained by the `MATLAB` function `polyfit()`. Instead of using the `MATLAB` log scale, we plotted directly the logarithm of the residual to draw the line.

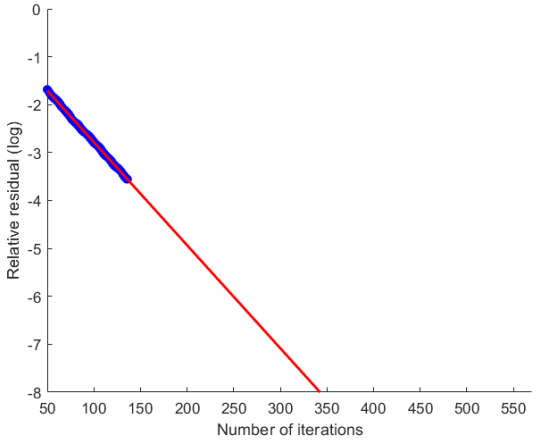

Figure 1: Beta Distribution (4,4)

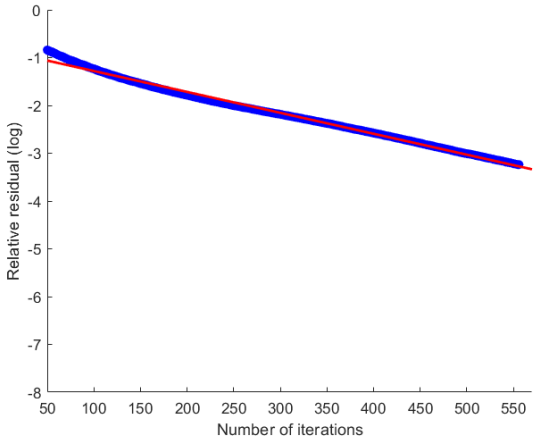

Figure 2: Beta Distribution (0.75, 0.75)

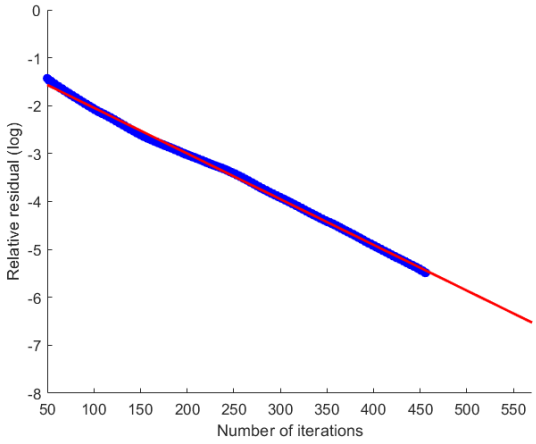

Figure 3: Chi Distribution (4)

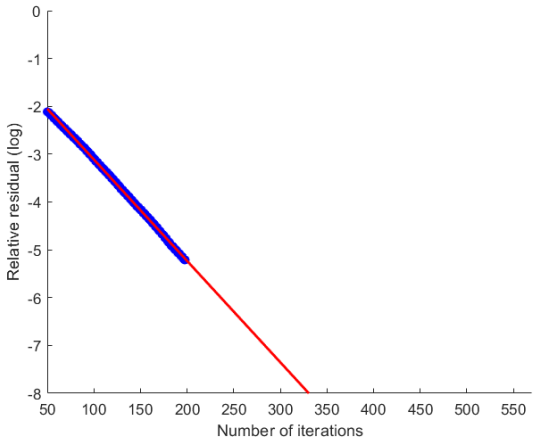

Figure 4: Gamma (5,1)

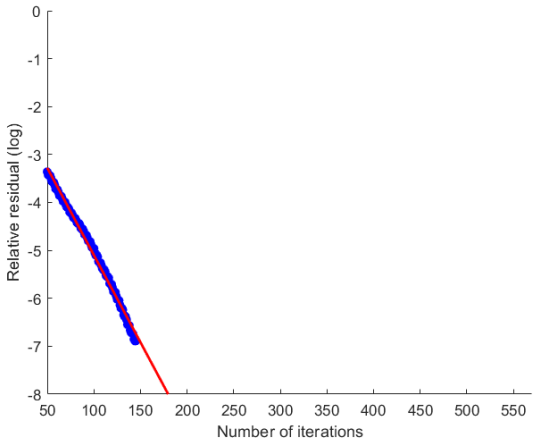

Figure 5: Mixture of Binomials

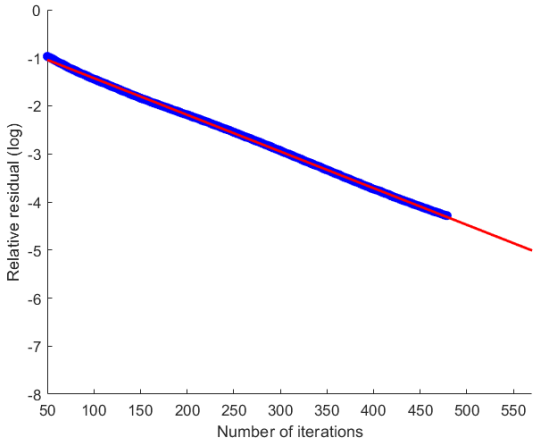

Figure 6: Uniform Distribution

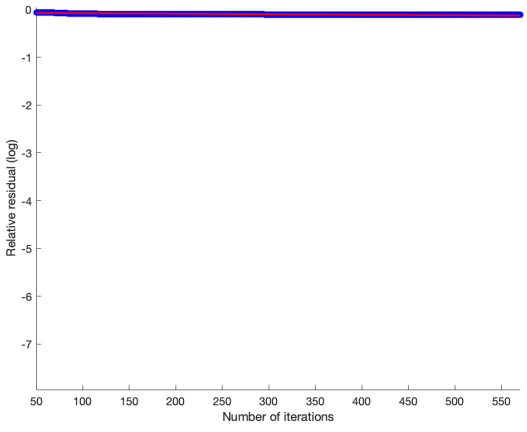

Figure 7: Ill Distribution

In each of these seven tests, the algorithm stopped before reaching the maximum number of iterations we have fixed, thanks to the delta stopping criterion (fixed to $\Delta = 10^{-10}$). As we can see from the figures above, the residual decreases linearly in the semi-log scale, so it is an exponential function of the number of iterations. In fact, its shape is well approximated by a line that sometimes lays slightly above and other times slightly below the points.

Another observation to do is that the distribution of the data in D affects the behavior of the residual. In fact, we can see that, changing the distribution, we have different orders of values for the residual and different slopes for the lines; furthermore, the number of iterations done by CustomGMRES changes. This can depend on the conditioning of the matrix D: ill-conditioned matrices could require more iterations to converge, while well-conditioned matrices could converge quickly.

Before investigating this hypothesis, let's see how the condition number of D influences that of A.

First, we recall this result seen during the course:

Theorem 7.1. Let Q be a symmetric real matrix. Then, it holds:

$$\lambda_{\min}||x||^2 \leq x^T Q x \leq \lambda_{\max}||x||^2 \,,$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimum and the maximum eigenvalues of Q.

In our case, if we consider $Q = A$ and x as a unitary vector, we can deduce that the eigenvalues of D are bounded from that of A. From this, it can be shown that if D is ill-conditioned, then A is ill-conditioned too.

Let's thus analyze the condition number of D in these seven cases:

	MixBin(0.7,20)	$\beta(4,4)$	$\Gamma(5,1)$	$\chi^2(4)$	Uniform	$\beta(0.75,0.75)$	Ill cond.
Cond	5.800	66.725	91.105	2.214e3	3.337e4	3.579e5	1.095e9
Iterations	145	136	198	456	479	556	1204

Table 1: Condition number and number of iterations

From the data in the Table 1 a relationship emerges: higher condition numbers corresponds to higher number of iterations. There is not an explicitly proportionality between *Cond* and *Iterations*, in fact the condition number in the case of the MixBin(0.7,20) distribution is smaller than that of Beta(4,4), while the number of iterations is slightly greater. However, when the condition number related to one distribution is orders of magnitude greater than that related to another distribution, the number of iterations of the first is clearly greater than that of the second. In fact, except for the case just described, this trend is confirmed by the numbers in the table.

7.2.2 Convergence Rate

Now, we look at the residual from the point of view of the convergence rate, which is defined as follows:

$$\mu := \lim_{n\to\infty} \frac{|r_{n+1} - \bar{x}|}{|r_n - \bar{x}|^\alpha} \,,$$

where $\bar{x}$ is the limit of the sequence $\{r_n\}$.
In our case, the sequence $\{r_n\}$ is the sequence of the relative residuals and, for the tests we did with the instance `net12_8_1`, we can assume that $\bar{x} = 0$. Moreover, since we know for sure that the residual does not improve anymore after $n = \nu$ iterations of the Arnoldi process (if not earlier), we can conclude that $\mu = 1$ with $\alpha = 1$.
In reality, we are not interested in what happen with high number of iterations, so we can look at the rate only for useful iterations.

In the plots below, we report on a plane a line which represents, for each distribution we used, a possible early estimate of the rate of convergence (μ).

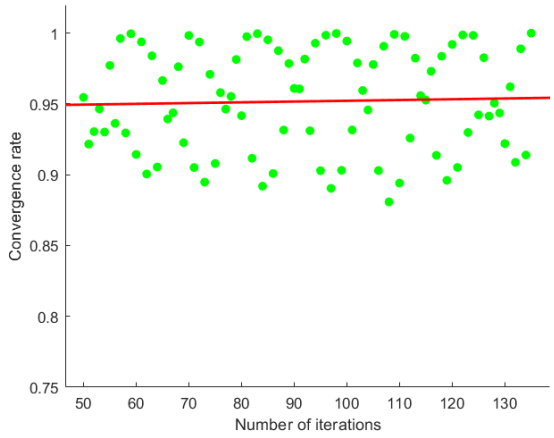

Figure 8: Beta Distribution (4,4)

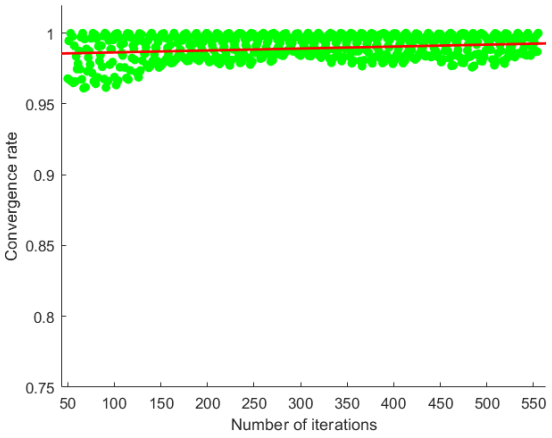

Figure 9: Beta Distribution (0.75, 0.75)

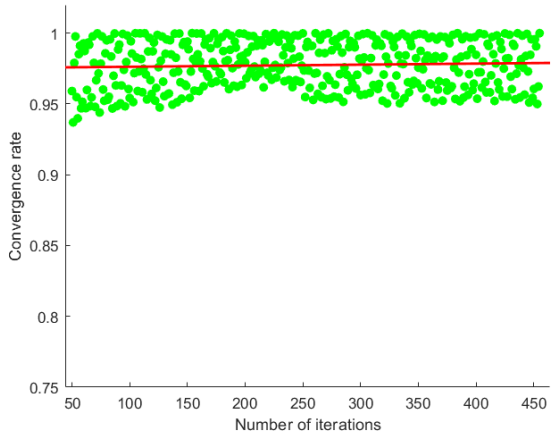

Figure 10: Chi Distribution (4)

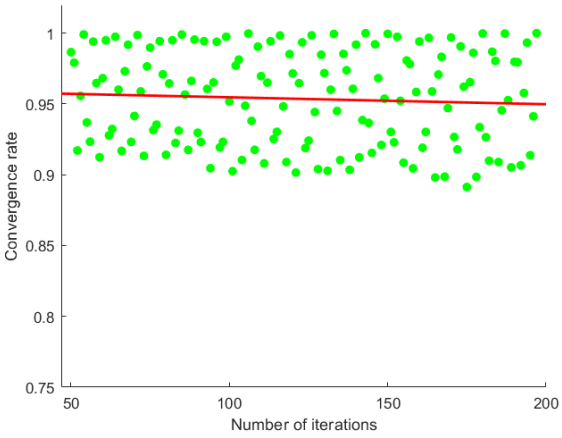

Figure 11: Gamma (5,1)

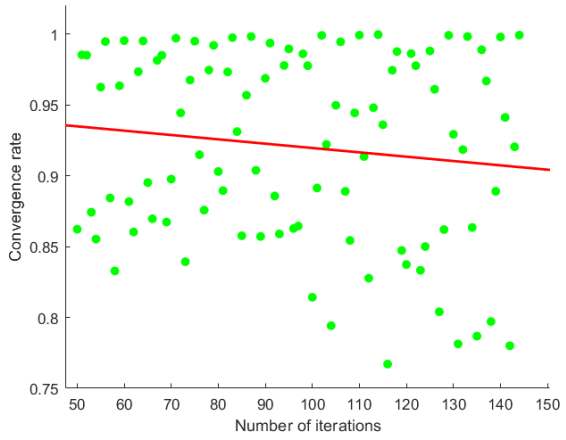

Figure 12: Mixture of Binomials

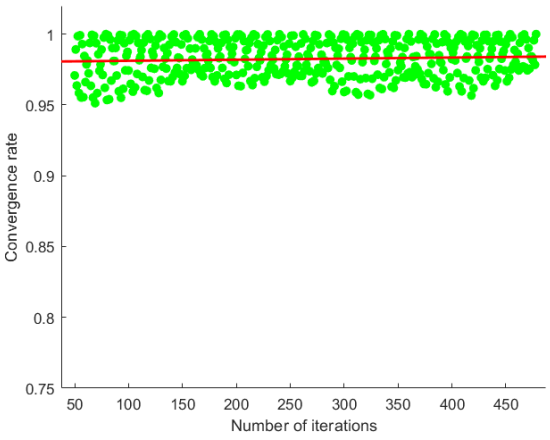

Figure 13: Uniform Distribution

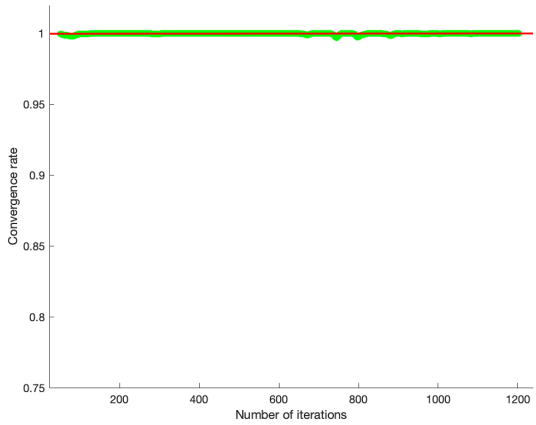

Figure 14: Ill Distribution

From a theoretical point of view, since we can not go to the limit, we can not talk about real convergence. Indeed, from a more practical point of view, from the plots above we can deduce that we have a linear convergence (i.e. $\alpha = 1$ and $\mu < 1$).
For comparing the tests, we plotted on a plane the lines corresponding to the function $y = \mu_i$, where i indicates the different distributions and μ_i comes out from the tests above.

Comparing the Figure 15 below with the Table 1, we can notice that there is a strong relation between μ and the condition number of D: smaller is the condition number, smaller is the convergence rate and so, faster is the convergence.

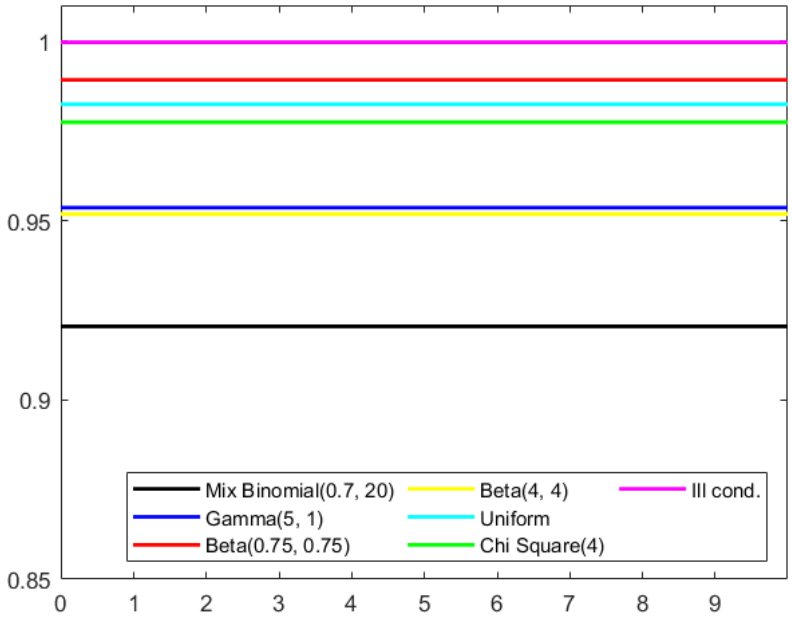

Figure 15: Rates

7.3 Impact of dimensions

Now that we have observed that CustomGMRES converges, we focus on other aspects, in particular we analyze how the dimensions of our problem influence the performance.

We distinguish two cases:

1. We fix the number of nodes and we increase gradually the number of edges without changing the other MCF parameters;
2. We fix the ratio between number of edges and number of nodes and we increase both of them together with some other parameters (sinks, sources, supply).

First of all we start considering the case where the number of nodes is fixed. We analyzed two different test cases:

1. We took as reference the instance `net10_64_2` where we changed only the number of edges, starting from 5000 up to 65000 with jumping intervals of 4000. The number of nodes is 1024, the number of sources and sinks is 102 and the distribution of D is a Gamma with parameters (5,1);
2. For the second test we took as reference the instance `net12_8_1` where the number of edges started from 12000 up to 144000 with jumping intervals of 8000. The number of nodes is 4096, the number of sources and sinks is 409 and the distribution of D is a mixture of Binomials with parameters (0.7,20).

We plotted on a plane the number of edges on the x-axis and the computation time on the y-axis. Then we draw a line that approximate the points.

As we can see from Figure 16, the time complexity seems to increase in a linear way. So, the growth of the number of edges, that is the density of the graph, does not affect negatively the performance of the algorithm.

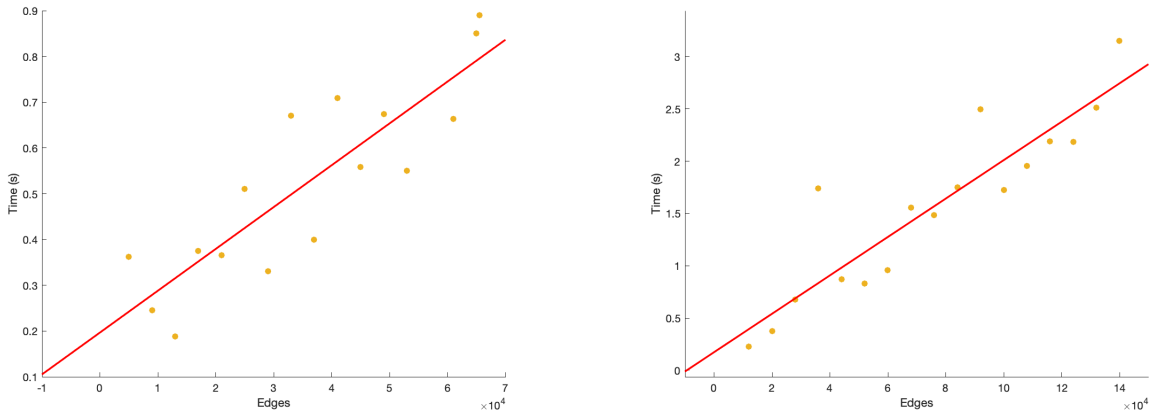

Figure 16: Fixed nodes and increasing edges

Now we move on to the case where we fix the ratio between number of edges and number of nodes:

1. Our reference is the instance `net8_8_1`; we started with 256 nodes, 2048 edges, 26 sources and 26 sinks and we multiplied these numbers by an increasing index at each iteration. The distribution of D is a Gamma with parameters (5,1);

2. The instance we used is `net8_32_3`, with 128 nodes, 4096 edges, 13 sources and 13 sinks and again we multiplied them by a certain index. This time the distribution of D is a mixture of Binomials with parameters (0.7,20).

Plotting the total dimension of the graph (edges + nodes) on the x-axis and the time on the y-axis (17), we can notice, like in the case where the number of nodes is fixed, that the time growth seems to be linear. Furthermore, looking at the two plots we can see how different ratios (edges/nodes) does not have a strong impact on the time behavior of the algorithm; in fact in the first plot the ratio is 8 and in the second plot the ratio is 32.

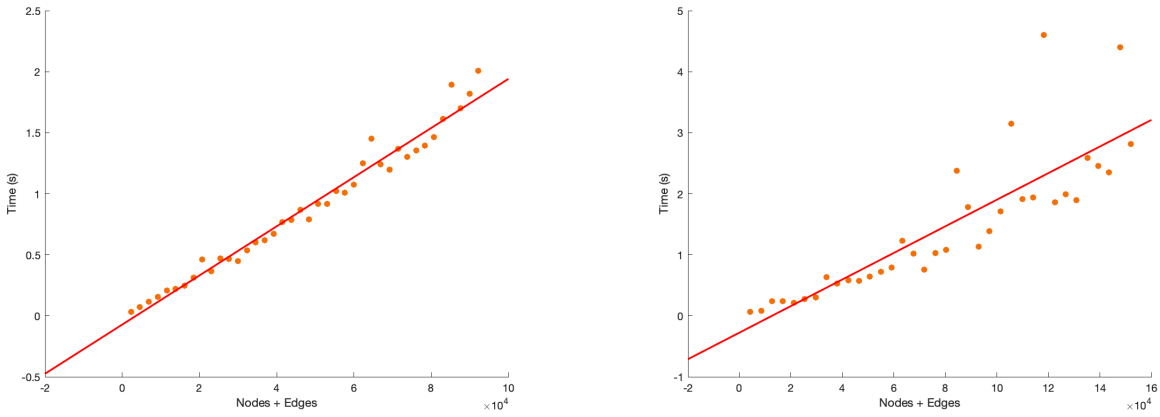

Figure 17: Increasing dimensions proportionally nodes and edges

7.4 Comparison with MINRES

At this point we show on a plane in *log-log* scale the results obtained from the comparison between our algorithm and MINRES, where, on the x-axis there is the *relative residual* and on the y-axis there is the *time* (in seconds).

In the following table we show the data about the matrices used in the tests:

Matrices					
Name	**D**	**E**			
	distribution	**n**	**e**	**sources**	**sinks**
net8_32_3	Uniform(0,1)	256	8192	26	26
net10_32_4	Beta(4,4)	1024	32768	102	102
net8_8_1	Beta(0.75,0.75)	256	2048	26	26
net10_64_2	Gamma(5,1)	1024	65536	102	102
net12_8_1	Chi Square(4)	4096	32768	409	409
net14_8_1	Mix_Binomial(0.7,20)	16384	131072	1638	1638

Table 2: Structure of matrices and vectors tested.

In each figure we show the results of various choices of the number of iterations m for CustomGMRES and MINRES; as this value increases, the colour of the points on the plane shifts from blue to light green (`MATLAB` winter scale). At the same time we plot the values of q_1^m (the lower bound of the residual defined in (4.1)) in red, for comparing it with the actual residual.

Under each figure there is a table which contains explicitly the values of m, q_1^m, r_m and the computation time (in seconds).

For each test we ran our algorithm with different choices of m, starting from a value greater or equal than 50 and with jumping intervals proportional to $\sqrt{\nu}$, up to a variable number depending on ν.

Let's see the results of the tests described in Table 2.

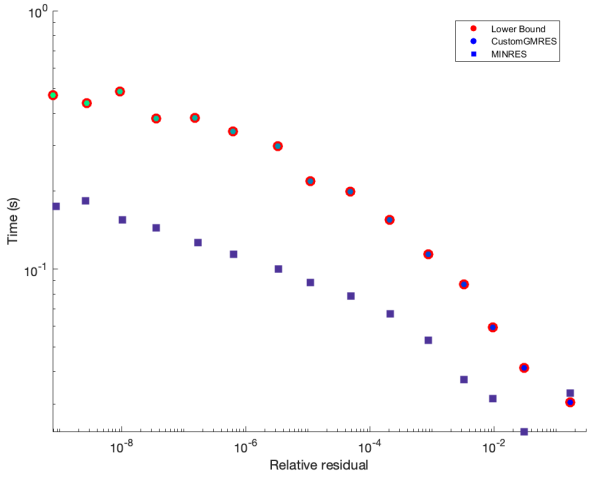

Figure 18: net8_32_3_uniform

net8_32_3 with uniform distribution					
	CustomGMRES			MINRES	
m	q	residual	time	residual	time
142	3.061057e-02	3.061057e-02	4.134117e-02	3.061057e-02	2.349864e-02
326	3.218366e-03	3.218366e-03	8.725560e-02	3.232104e-03	3.739204e-02
510	2.063986e-04	2.063986e-04	1.548432e-01	2.089765e-04	6.705663e-02
694	1.094089e-05	1.094089e-05	2.192536e-01	1.099748e-05	8.858874e-02
970	1.493080e-07	1.493080e-07	3.855935e-01	6.349559e-07	1.143341e-01
1154	9.189447e-09	9.189447e-09	4.858289e-01	3.569937e-08	1.446828e-01
1338	7.656268e-10	7.656268e-10	4.722837e-01	2.581293e-09	1.838020e-01

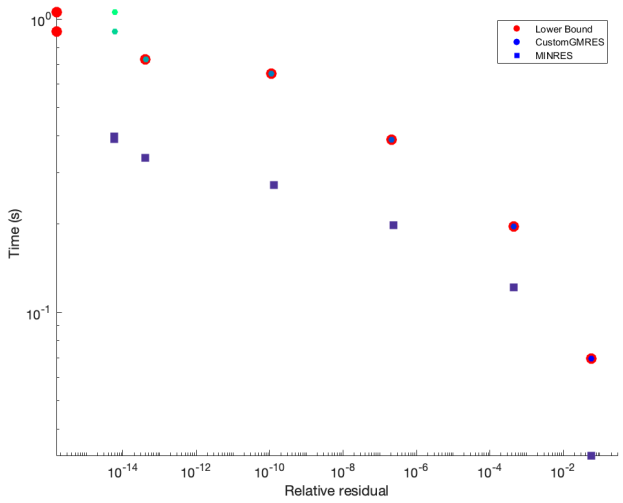

Figure 19: net10_32_4

net10_32_4 with Beta Distribution $\alpha = 4$ and $\beta = 4$					
	CustomGMRES			MINRES	
m	q	residual	time	residual	time
50	5.732876e-02	5.732876e-02	6.960595e-02	5.732876e-02	3.251577e-02
234	4.472121e-04	4.472121e-04	1.967095e-01	4.472115e-04	1.219032e-01
418	2.120734e-07	2.120734e-07	3.883504e-01	2.327472e-07	1.979041e-01
602	1.099668e-10	1.099669e-10	6.523099e-01	1.302482e-10	2.723869e-01
786	4.126885e-14	4.167904e-14	7.305997e-01	4.146247e-14	3.361480e-01
970	1.544678e-16	5.983844e-15	9.091597e-01	5.881132e-15	3.908755e-01
1154	1.538559e-16	6.158553e-15	1.059857e+00	5.881132e-15	3.981198e-01

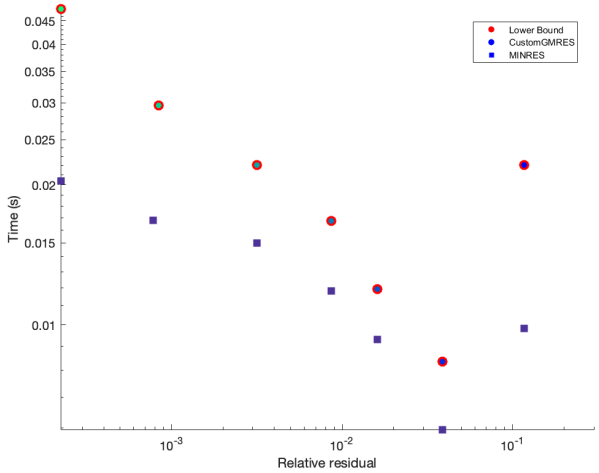

Figure 20: net8_8_1

net8_8_1 with Beta Distribution $\alpha = 0.75$ and $\beta = 0.75$					
	CustomGMRES			**MINRES**	
m	**q**	**residual**	**time**	**residual**	**time**
50	1.166718e-01	1.166718e-01	2.202907e-02	1.166718e-01	9.841343e-03
98	3.882725e-02	3.882725e-02	8.335548e-03	3.882725e-02	5.961468e-03
146	1.606494e-02	1.606494e-02	1.194898e-02	1.605420e-02	9.302450e-03
194	8.633946e-03	8.633946e-03	1.672819e-02	8.640413e-03	1.181755e-02
242	3.177305e-03	3.177305e-03	2.208171e-02	3.172795e-03	1.499534e-02
290	8.410273e-04	8.410273e-04	2.962434e-02	7.799861e-04	1.676901e-02
338	2.240648e-04	2.240648e-04	4.777194e-02	2.236685e-04	2.040416e-02

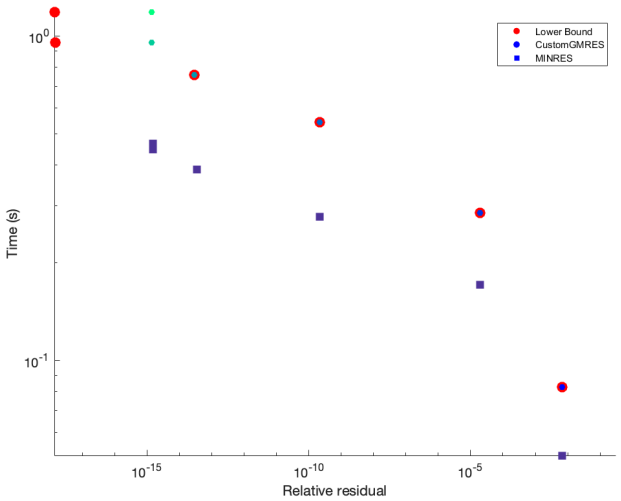

Figure 21: net10_64_2

net10_64_2 with Gamma Distribution $k = 5$ and $\theta = 1$					
	CustomGMRES			MINRES	
m	**q**	**residual**	**time**	**residual**	**time**
50	6.834143e-03	6.834143e-03	8.262636e-02	6.834144e-03	5.090010e-02
179	1.922984e-05	1.922984e-05	2.849449e-01	1.951083e-05	1.706149e-01
308	2.158542e-10	2.158542e-10	5.425952e-01	2.152512e-10	2.768467e-01
437	2.859279e-14	2.864231e-14	7.575895e-01	3.474221e-14	3.868549e-01
566	1.495694e-18	1.368456e-15	9.553049e-01	1.512736e-15	4.470788e-01
695	1.366371e-18	1.374072e-15	1.188061e+00	1.512736e-15	4.664553e-01

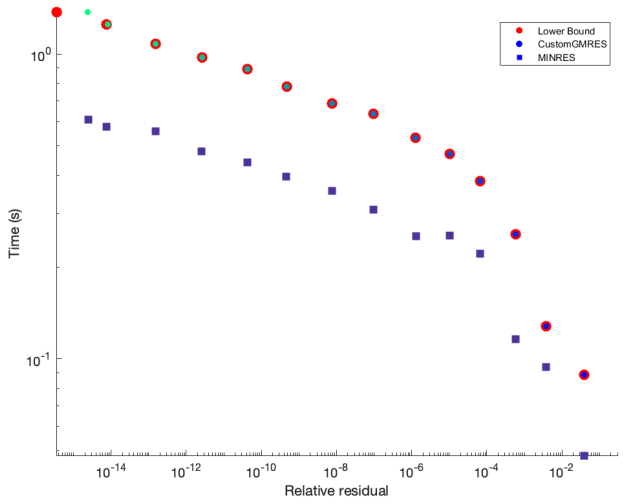

Figure 22: net12_8_1

net12_8_1 with Chi Square Distribution $\chi = 4$					
	CustomGMRES			MINRES	
m	**q**	**residual**	**time**	**residual**	**time**
146	3.821087e-03	3.821087e-03	1.279031e-01	3.820681e-03	9.385775e-02
338	6.602324e-05	6.602324e-05	3.820953e-01	6.580824e-05	2.207108e-01
530	1.280046e-06	1.280046e-06	5.298693e-01	1.288903e-06	2.525937e-01
722	7.597544e-09	7.597544e-09	6.865604e-01	7.614018e-09	3.558649e-01
914	4.178710e-11	4.178710e-11	8.920128e-01	4.172716e-11	4.407030e-01
1106	1.501990e-13	1.502018e-13	1.077373e+00	1.545073e-13	5.565326e-01
1298	3.495998e-16	2.434054e-15	1.374209e+00	2.437351e-15	6.081438e-01

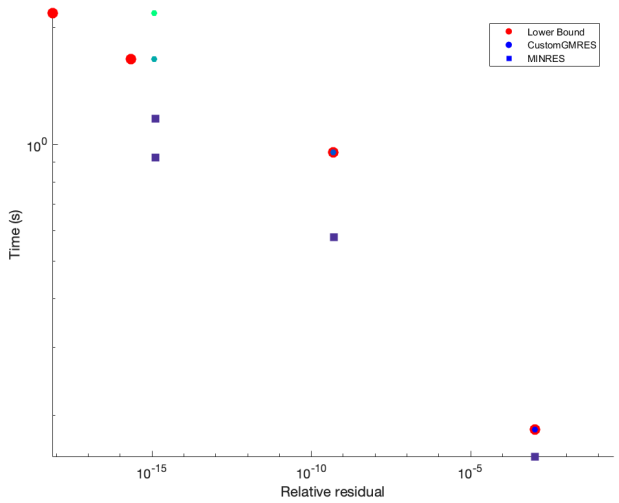

Figure 23: net14_8_1

net14_8_1 with mixture of Binomial $p = 0.7$ and $n = 20$					
	CustomGMRES			**MINRES**	
m	**q**	**residual**	**time**	**residual**	**time**
50	1.008944e-03	1.008944e-03	1.844156e-01	1.008944e-03	1.570028e-01
242	4.766004e-10	4.766004e-10	9.548040e-01	5.114018e-10	5.754294e-01
434	2.219771e-16	1.151815e-15	1.656045e+00	1.289773e-15	9.260074e-01
626	7.511438e-19	1.184620e-15	2.182162e+00	1.289773e-15	1.165151e+00

7.4.1 Considerations on the results

As we can see from the plots and the tables, the actual residual is very precise and it is almost always equal to the lower bound.

All tests stopped before reaching the maximum number of iterations thanks to the check on the delta of the residuals.

Regarding the comparison between CustomGMRES and MINRES, we analyze two aspects:

- **Residual**: The CustomGMRES performs slighlty better than the MINRES, with the same number of iterations in most cases it has a smaller residual than the MINRES (but sometimes it is the opposite);
- **Time**: The MINRES always performs better than the CustomGMRES.

In these experiments we look at different values of m (the number of iterations of the Arnoldi process) to understand the behavior of CustomGMRES. In reality, we could add a check on the lower bound q_1^m in the Arnoldi process and, if it doesn't improve enough from one step to the next, we can stop. Another check that we can do is the computation of the actual residual at fixed steps of the number of iterations (we can't do this at each iteration like in the case of the lower bound because it would be too expensive in terms of computation time).

8 Preconditioning

Now that we have analyzed the various aspects of our algorithm, we can try to improve its performances. As required by the Problem A2, we introduce a preconditioner. The preconditioner we use is the Schur Complement Preconditioner, defined as follows:

$$P := \begin{bmatrix} D & 0 \\ E & S \end{bmatrix},$$

where $S = -ED^{-1}E^T$.

The classical way of using a preconditioner is by left-multiplying its transpose to both the sides of the linear system $A\widetilde{x} = \widetilde{b}$. If P is non-singular, then the solution of the system $P^TA\widetilde{x} = P^T\widetilde{b}$ is equal to that of the original system. In our case, the matrix P is singular. In fact, given e as the vector with all components equal to 1, if we take the vector:

$$z = \begin{bmatrix} 0 \\ e \end{bmatrix},$$

we have that $z^TP = 0$. However, this does not directly imply that the preconditioner does not work. To cover also the case where the singularity of P afflicts the solubility of our system, we can make P non-singular by just adding aI to the matrix S, where I is the identity matrix and a is an opportune real number.

To keep the major advantage of our matrix A, i.e. the symmetry, we have to use a slightly different approach; rewriting the preconditioned system, we have:

$$P^TA\widetilde{x} = P^T\widetilde{b} \;\equiv\; P^TAP\underbrace{P^{-1}\widetilde{x}}_{y} = P^T\widetilde{b} \;\equiv\; P^TAPy = P^T\widetilde{b}.$$

So, we solve the symmetric system $P^TAPy = P^T\widetilde{b}$ and then we multiply y by P obtaining the solution of the original system $\widetilde{x} = Py$.

Before moving on to the experiments, we briefly discuss the costs and the implementation.

Since the matrix P is composed by D and E and we already have stored these matrices, we do not need additional space. So, the space complexity does not change.

With regard to time complexity, we do not explicitly compute the products $S = -ED^{-1}E^T$ and P^TAP, but, when we need to compute a product between one of these matrices and a vector, we just iterate the matrix-vector product from right to left. In this way, also the asymptotic time complexity does not change.

8.1 Preconditioned Experiments

Given the preconditioned linear system $P^TAPy = P^T\widetilde{b}$ we mentioned above, we analyze the performance of our algorithm, using the instances in Table 2. To understand if the preconditioner works, we compare the preconditioned system with the original one.

We tested various instances, but, since it looks they have very similar behavior, we only report the tests with the `net10_32_4` with different values distributions, both well and ill-conditioned. Furthermore, we tested different values of the parameter a (of different magnitude orders) to be sure that the singularity of P does not affect the results. Also in this case, we report only some representative values of a.

In the planes in the plots below, the x-axis represents the residual in *log* scale and the y-axis represents the computation time in seconds. On these planes, we plotted the pair $(residual, time)$ for increasing number of iterations, in three different cases:

- *Non Preconditioned*: it is the result given by the original system $A\widetilde{x} = \widetilde{b}$;
- *Preconditioned Transformed*: it is the residual $\frac{||P^T APy - P^T\widetilde{b}||}{||P^T\widetilde{b}||}$;
- *Preconditioned Original*: it is the residual of the original system $A\widetilde{x} = \widetilde{b}$, but this is computed multiplying by P the solution of the preconditioned system $P^T APy = P^T\widetilde{b}$.

We plotted also the lower bound for the residuals: in red that of the original system and in magenta the one of the system $P^T APy = P^T\widetilde{b}$.

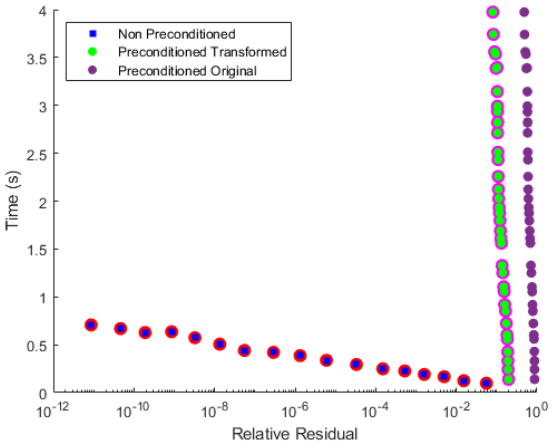

Figure 24: Beta(4,4), $a = 0$

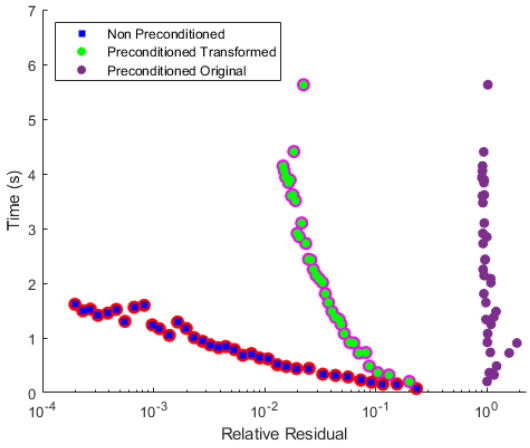

Figure 25: Beta(0.75,0.75), $a = 0$

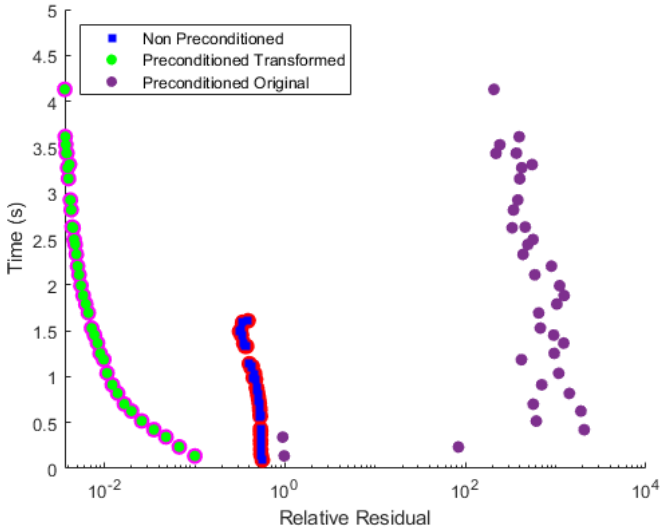

Figure 26: Ill-conditioned, $a = 0$

The simplest case is the non-perturbed one (i.e. $a = 0$). As we can see from the plots above, both the *Non Preconditioned* and the *Preconditioned Transformed* converge, while the *Preconditioned Original* does not converge. In particular, the solution obtained solving the preconditioned system $x = Py$, produces a far worse residual than the original one, even taking much more time.
We observe that, in the ill-conditioned case, the *Preconditioned Transformed* has a better convergence than *Non Preconditioned*. This tells us that the algorithm is able to solve $P^T APy = P^T\widetilde{b}$, but whenever we compute $\widetilde{x}$ and we use it to calculate the residual of $A\widetilde{x} = \widetilde{b}$, we lose the convergence.
Let's see if the convergence is invalidated by the singularity of P and so, if adding the value aI to S improves the convergence. Since we observed that the behavior with the three different distributions tested above is similar, we show the plots only for the distribution Beta(0.75, 0.75).

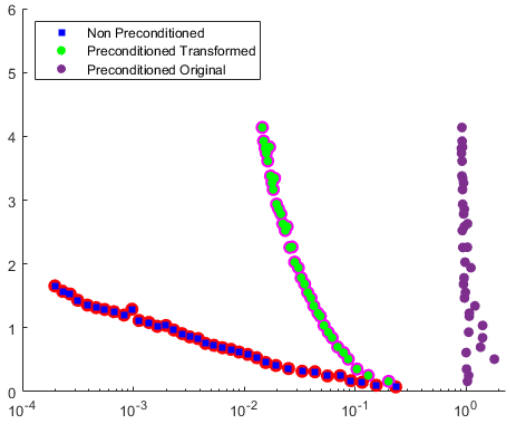

Figure 27: $a = 2$

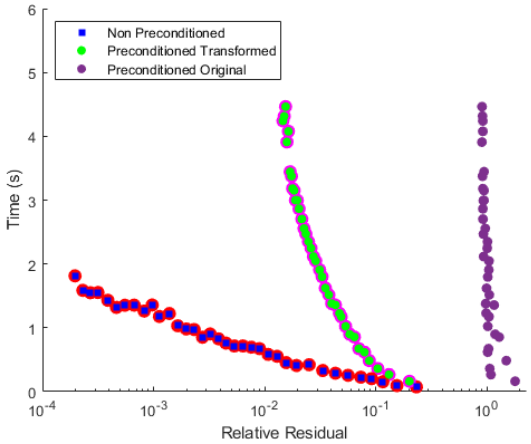

Figure 28: $a = -2$

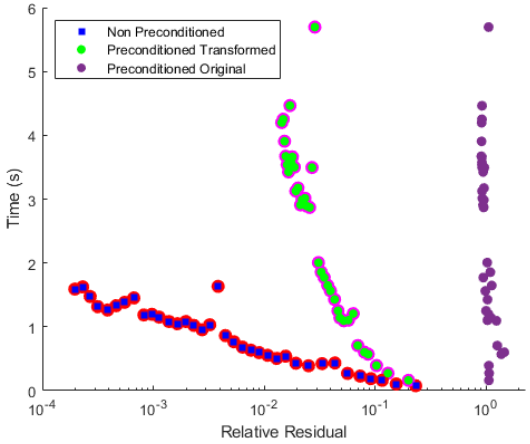

Figure 29: $a = 10$

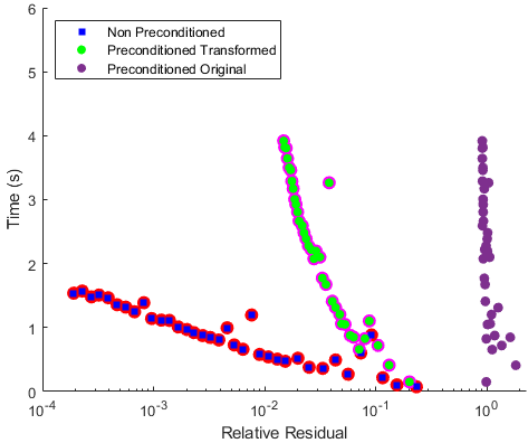

Figure 30: $a = -10$

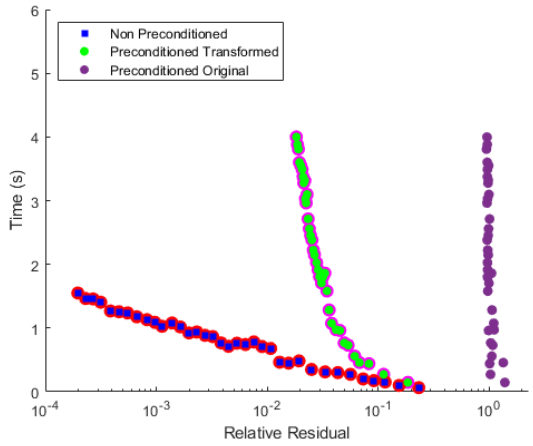

Figure 31: $a = 300$

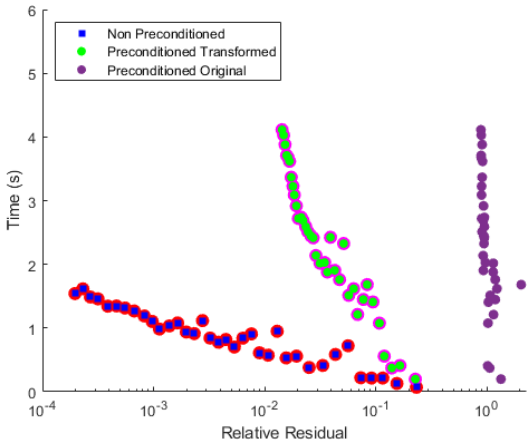

Figure 32: $a = -300$

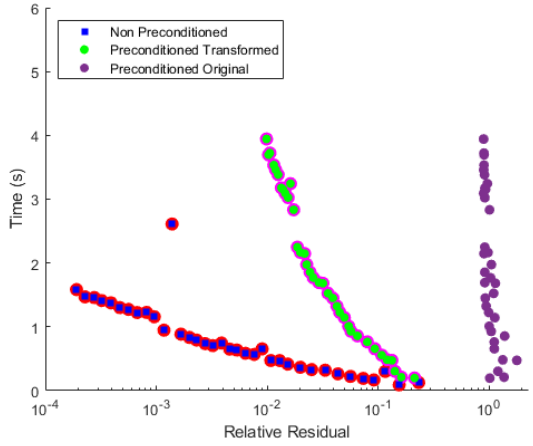

Figure 33: $a = 1000$

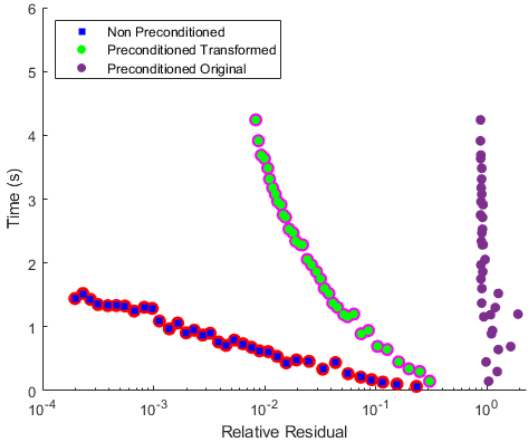

Figure 34: $a = -1000$

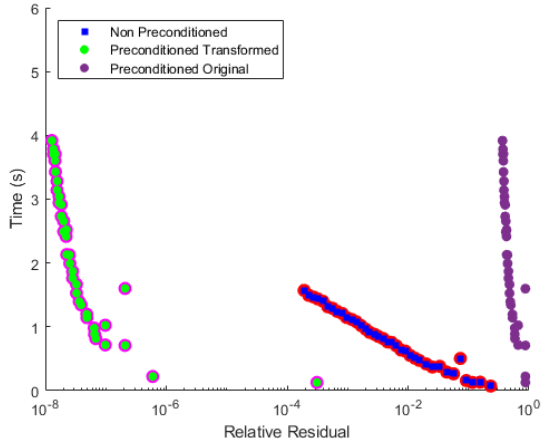

Figure 35: $a = 10^8$

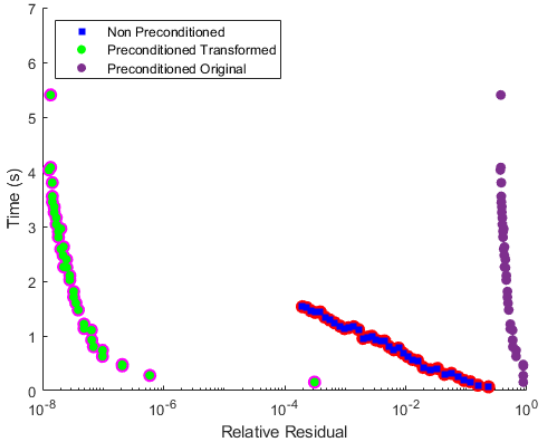

Figure 36: $a = -10^8$

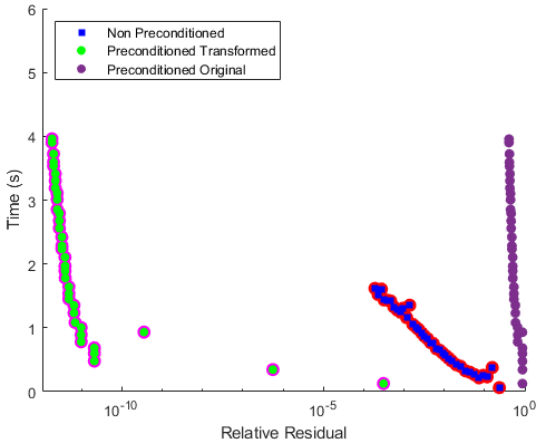

Figure 37: $a = 10^{12}$

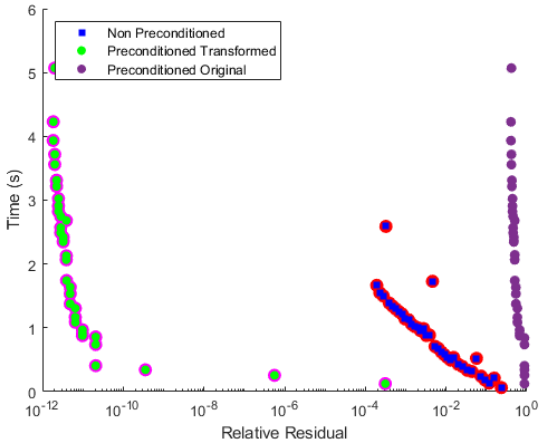

Figure 38: $a = -10^{12}$

Observing the plots above, we can conclude that, either with small or big perturbations, the residual of the *Preconditioned Original* does not improve. The only thing that substantially changes from one case to the other, is the residual of the *Preconditioned Transformed*, that, when a is very big, is small and this is probably due to the fact that the denominator of the residual $||P^T\widetilde{b}||$ increases. However, the good convergence of the *Preconditioned Transformed* is just an indicator of the fact that CustomGMRES works, but the solution of this system has no interests for us except for the computation of $\widetilde{x} = P^T\widetilde{y}$ that, as we can see, is not a good approximation for the solution of $Ax = b$. Thus, we can conclude that, in general, this preconditioner does not improve the relative residual; on the contrary, the residual increases blowing up the computation time.

References

[1] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997. ISBN: 0898713617.