TSIRM: A Two-Stage Iteration with least-squares Residual Minimization algorithm to solve large sparse linear and nonlinear systems

Raphaël Couturier, Lilia Ziane Khodja and Christophe Guyeux

Femto-ST Institute, Univ. Bourgogne Franche-Comté (UBFC), France

Abstract

In this paper, a two-stage iterative algorithm is proposed to improve the convergence of Krylov based iterative methods, typically those of GMRES variants. The principle of the proposed approach is to build an external iteration over the Krylov method, and to frequently store its current residual (at each GMRES restart for instance). After a given number of outer iterations, a least-squares minimization step is applied on the matrix composed by the saved residuals, in order to compute a better solution and to make new iterations if required. It is proven that the proposal has the same convergence properties as the inner embedded method itself. Several experiments have been performed using the PETSc toolkit (using default parameters in the absence of detail) to solve linear and nonlinear problems. They show good speedups compared to GMRES with up to 16,394 cores with different preconditioners.

Keywords: Iterative Krylov methods; sparse linear and nonlinear systems; two-stage iteration; least-squares residual minimization; PETSc.

1. Introduction

Iterative methods have recently become more attractive than direct ones to solve very large sparse linear systems [1]. They are more efficient in a parallel context, supporting thousands of cores, and they require less memory and arithmetic operations than direct methods [2]. This is why new iterative methods are frequently proposed or adapted by researchers, and the increasing need to solve

very large sparse linear systems has triggered the development of such efficient iterative techniques suitable for parallel processing.

Most of the successful iterative methods currently available are based on so-called "Krylov subspaces". They consist in forming a basis of successive matrix powers multiplied by an initial vector, which can be for instance the residual. These methods use vectors orthogonality of the Krylov subspace basis in order to solve linear systems. The best known iterative Krylov subspace methods are conjugate gradient and GMRES ones (Generalized Minimal RESidual).

However, iterative methods suffer from scalability problems on parallel computing platforms with many processors, due to their need of reduction operations, and to collective communications to achieve matrix-vector multiplications. The communications on large clusters with thousands of cores and large sizes of messages can significantly affect the performances of these iterative methods. As a consequence, Krylov subspace iteration methods are often used with preconditioners in practice, to increase their convergence and accelerate their performances. However, most of the good preconditioners are not scalable on large clusters.

In this research work, a two-stage algorithm based on two nested iterations called inner-outer iterations is proposed. This algorithm consists in solving the sparse linear system iteratively with a small number of inner iterations, and restarting the outer step with a new solution minimizing some error functions over some previous residuals. Two-stage algorithms are easy to parallelize on large clusters, and the least-squares minimization technique proposed in this paper improves their convergence and performances. For further information on two-stage iteration methods, interested readers are invited to consult [3].

The present article is organized as follows. Related works are presented in Section 2. Section 3 details the two-stage algorithm using a least-squares residual minimization, while Section 4 provides convergence results regarding this method. Section 5 shows some experimental results obtained on large clusters using routines of PETSc toolkit. This research work ends by a conclusion section, in which the proposal is summarized while intended perspectives are

provided.

2. Related works

Krylov subspace iteration methods have increasingly become key techniques for solving linear and nonlinear systems, or eigenvalue problems, especially since the increasing development of preconditioners [1, 4]. One reason for the popularity of these methods is their generality, simplicity, and efficiency to solve systems of equations arising from very large and complex problems.

GMRES is one of the most widely used Krylov iterative method for solving sparse and large linear systems. It has been developed by Saad *et al.* [5] as a generalized method to deal with unsymmetric and non-Hermitian problems, and indefinite symmetric problems too. In its original version called full GMRES, this algorithm minimizes the residual over the current Krylov subspace until convergence in at most n iterations, where n is the size of the sparse matrix. Full GMRES is however too expensive in the case of large matrices, since the required orthogonalization process per iteration grows quadratically with the number of iterations. For that reason, GMRES is restarted in practice after each $m \ll n$ iterations, to avoid the storage of a large orthonormal basis. However, the convergence behavior of the restarted GMRES, called GMRES(m), in many cases depends quite critically on the m value [6]. Therefore in most cases, a preconditioning technique is applied to the restarted GMRES method in order to improve its convergence.

To enhance the robustness of Krylov iterative solvers, some techniques have been proposed allowing the use of different preconditioners, if necessary, within the iteration itself instead of restarting. Those techniques may lead to considerable savings in CPU time and memory requirements. Van der Vorst in [7] has for instance proposed variants of the GMRES algorithm in which a different preconditioner is applied in each iteration, leading to the so-called GMRESR family of nested methods. In fact, the GMRES method is effectively preconditioned with other iterative schemes (or GMRES itself), where the iterations

of the GMRES method are called outer iterations while the iterations of the preconditioning process is referred to as inner iterations. Saad in [8] has proposed Flexible GMRES (FGMRES) which is another variant of the GMRES algorithm using a variable preconditioner. In FGMRES the search directions are preconditioned whereas in GMRES the residuals are preconditioned. However, in practice, good preconditioners are those based on direct methods, as Incomplete LU (ILU) preconditioners [1], which are not easy to parallelize and suffer from the scalability problems on large clusters of thousands of cores.

Recently, communication-avoiding methods have been developed to reduce the communication overheads in Krylov subspace iterative solvers. On modern computer architectures, communications between processors are much slower than floating-point arithmetic operations on a given processor. Communication-avoiding techniques reduce either communications between processors or data movements between levels of the memory hierarchy, by reformulating the communication-bound kernels (more frequently SpMV kernels) and the orthogonalization operations within the Krylov iterative solver. Different works have studied the communication-avoiding techniques for the GMRES method, so-called CA-GMRES, on multicore processors and multi-GPU machines [9, 10, 11].

Compared to all these works and to all the other works on Krylov iterative methods, the originality of our work is to build a second iteration over a Krylov iterative method and to minimize the residuals with a least-squares method after a given number of outer iterations.

3. TSIRM: Two-stage iteration with least-squares residuals minimization algorithm

TSIRM is a two-stage algorithm based on subspace Krylov methods to solve large sparse linear systems. The main idea behind the algorithm is to restart a Krylov method after a few outer iterations with an accurate guess which minimizes the residuals computed with the given Krylov method. The TSIRM algorithm uses a minimization process based on the least-squares method to

compute the solution minimizing the Euclidean norm of the residual. The approach used by the TSIRM method allows to improve the slow convergence of the subspace Krylov methods. In the following, two kinds of iterations are considered: the iterations of the Krylov method and the iterations of the minimization method. In practice, these iterations are classical iterations as in any iterative methods.

Algorithm 1 summarizes the main key points of the TSIRM method. It solves a sparse linear system of n equations of the form $Ax = b$, where A is the sparse square and nonsingular matrix, x is the solution and b is the right-hand side. As explained previously, TSIRM is an inner-outer iteration method. In the inner iteration (line 4), TSIRM solves the linear systems partially by using a few iterations, $mxIt_{kryl}$, of an iterative Krylov method called inner solver. The GMRES method [5], or any of its variants, can potentially be used as inner solver. Moreover, a tolerance threshold tol_{kryl} must be specified for the inner solver. In practice, this threshold must be much smaller than the convergence threshold of the TSIRM algorithm (*i.e.*, tol_{tsirm}). A threshold tolerance is used to stop an iterative method when the current error is lower than this threshold.

Before the minimization step, TSIRM executes s outer iterations and stores the approximations in a dense matrix S. This task consists in copying the solution x_k computed by the inner solver into the column $k \mod s$ of S (line 5), so that s is much smaller than the size of the linear system ($s \ll n$). Then, TSIRM performs a minimization on the residuals ($b - AS$) of the approximations stored inside the matrix S. The objective of the minimization is to compute a new approximation x with a minimal residual. For this, TSIRM solves the linear least-squares problem:

$$\min_{\alpha \in \mathbb{R}^s} ||b - AS\alpha||_2 \tag{1}$$

to find $\alpha \in \mathbb{R}^s$ which minimizes the residuals $||b - AS||_2$. The new solution x is then computed with $x = S\alpha$. In our context, an iterative method such as the Conjugate Gradient method for Least Squares (CGLS) [12] or the Least Squares (LSQR) method [13] is used to solve the least-squares problem (line 9).

Remark that these methods are more appropriate than a single direct method in a parallel context. CGLS has recently been used to improve the performance of multisplitting algorithms [14]. Two parameters are required for these least-squares methods: the maximum number of iterations $mxIt_{ls}$ and the threshold tol_{ls} to stop the method.

The TSIRM algorithm iterates until reaching the convergence with a desired precision tol_{tsirm}. In our case, TSIRM stops the iterations when the residual norm (see lines 2, 6 and 11) is less than the norm of the right-hand side by a predefined factor tol_{tsrim}:

$$||b - Ax_k||_2 < tol_{tsirm} \cdot ||b||_2 \tag{2}$$

where x_k is the solution computed at iteration k. During the resolution, the inner solver is initialized with the last obtained approximation, and the matrix S is reused after each minimization step with new values of the residuals.

Let us summarize the most important parameters of TSIRM:

- tol_{tsirm}: the threshold that stops the TSIRM method;
- $mxIt_{kryl}$: the maximum number of iterations for the Krylov method;
- tol_{kryl}: the threshold used to stop the Krylov method;
- s: the number of outer iterations before applying the minimization step;
- $mxIt_{ls}$: the maximum number of iterations for the iterative least-squares method;
- tol_{ls}: the threshold used to stop the least-squares method.

The parallelization of TSIRM relies on the parallelization of all its parts. More precisely, except the least-squares step, all the other parts are easy to achieve out in parallel. In order to develop a parallel version of our code, we have chosen to use PETSc [15]. Line 8, the matrix-matrix multiplication is implemented and efficient since the matrix A is sparse and the matrix S contains

Algorithm 1 TSIRM

Input: A (sparse matrix), b (right-hand side)

Output: x (solution)

1: Set the initial guess x_0
2: $error = \frac{||b-Ax_0||_2}{||b||_2}$
3: **for** $k = 1, 2, \ldots$ until $error < tol_{tsirm}$ **do**
4: $\quad x_k = Solve_{kryl}(A, x_{k-1}, b, mxIt_{kryl}, tol_{kryl})$
5: $\quad S_{k \mod s} = x_k$
6: $\quad error = \frac{||b-Ax_k||_2}{||b||_2}$
7: $\quad$ **if** $k \mod s = 0$ **and** $error \geqslant tol_{tsirm}$ **then**
8: $\quad\quad R = AS$
9: $\quad\quad \alpha = Solve_{ls}(R, b, mxIt_{ls}, tol_{ls})$
10: $\quad\quad x_k = S\alpha$
11: $\quad\quad error = \frac{||b-Ax_k||_2}{||b||_2}$
12: $\quad$ **end if**
13: **end for**

few columns in practice. In practice, the MatMatMult function is used to do that in PETSc.

As explained previously, at least two methods seem to be interesting to solve the least-squares minimization, the CGLS and the LSQR methods.

The TSIRM code and the CGLS code have been integrated to the PETSc tool. TSIRM has been implemented as any solver for linear systems in PETSc. As it requires to use another solver, we have used a very interesting feature of PETSc that enabled us to use a preconditioner as a linear system with the function `PCKSPGetKSP`. As the LSQR function was already implemented in PETSc, we have used it. CGLS had not been implemented yet, so we implemented it and defined it as a minimization solver in PETSc similarly to LSQR. Both CGLS and LSQR are not complex from a computational point of view. They involve matrix-vector multiplications and some classical operations: dot product, norm, multiplication, and addition on vectors. As presented in Section 5

the minimization step is scalable.

4. Convergence results

We suppose in this section that GMRES(m) is used as solver in the TSIRM algorithm applied on a complex matrix A. Let us denote A^* the conjugate transpose of A, and let $\mathfrak{R}(A) = \frac{1}{2}(A + A^*)$, $\mathfrak{I}(A) = \frac{1}{2i}(A - A^*)$. We now discuss various convergence situations according to the properties of matrix A.

4.1. $\mathfrak{R}(A)$ is positive

Proposition 1. *If $\mathfrak{R}(A)$ is positive, then the TSIRM algorithm is convergent.*

Proof 1. *If $\mathfrak{R}(A)$ is positive, then even if A is complex, it is possible to state that the GMRES algorithm is convergent, see,* e.g., *[6]. In particular, its residual norm decreases to zero.*

At each iteration of the TSIRM algorithm, either a GMRES iteration is realized or a least square resolution (to find the minimum of $||b-Ax||_2$ is achieved on the linear span of the iterated approximation vectors span $(x_{k-s+1}, x_{k-s+2}, \ldots, x_k)$ of the last GMRES stage, where $\mathrm{span}(S) = \left\{ \sum_{i=1}^{k} \lambda_i v_i \middle| k \in \mathbb{N}, v_i \in S, \lambda_i \in \mathbb{R} \right\}$.

Obviously, the minimum of $||b-Ax||_2$ on the set span $(x_{k-s+1}, x_{k-s+2}, \ldots, x_k)$ is lower than or equal to $||b-Ax_k||_2$, which is the last obtained GMRES-residual norm. So we can conclude that the intermediate stage of the least square resolution inserted into the GMRES algorithm does not break the decrease to zero of the GMRES-residual norm.

In other words, the TSIRM algorithm is convergent.

Regarding the convergence speed, we can claim that,

Proposition 2. *If A is a positive matrix, then the convergence of the TSIRM algorithm is linear.*

Furthermore, let r_k be the k-th residue of TSIRM, then we have the following bounds:

$$||r_k|| \leqslant \left(1 - \frac{\alpha}{\beta}\right)^{\frac{km}{2}} ||r_0||, \tag{3}$$

where M is the symmetric part of A, $\alpha = \lambda_{min}(M)^2$ and $\beta = \lambda_{max}(A^T A)$.

Proof 2. *Let us first recall that, when A is a positive real matrix with symmetric part M, then the residual norm provided at the m-th step of GMRES satisfies:*

$$||r_m|| \leqslant \left(1 - \frac{\alpha}{\beta}\right)^{\frac{m}{2}} ||r_0||,$$

where α and β are defined as in Proposition 2. These well-known results can be found, e.g.*, in [5].*

We will now prove by a mathematical induction that, for each $k \in \mathbb{N}^*$, $||r_k|| \leqslant \left(1 - \frac{\alpha}{\beta}\right)^{\frac{mk}{2}} ||r_0||$ when A is positive.

The base case is obvious, as for $k = 1$, the TSIRM algorithm simply consists in applying GMRES(m) once, leading to a new residual r_1 that follows the inductive hypothesis due to the results recalled above.

Suppose now that the claim holds for all $m = 1, 2, \ldots, k-1$, that is, $\forall m \in \{1, 2, \ldots, k-1\}$, $||r_m|| \leqslant \left(1 - \frac{\alpha}{\beta}\right)^{\frac{km}{2}} ||r_0||$. We will show that the statement holds too for r_k. Two situations can occur:

- *If $k \not\equiv 0 \ (mod\ m)$, then the TSIRM algorithm consists in executing GMRES once. In that case, and by using the inductive hypothesis, we obtain $||r_k|| \leqslant \left(1 - \frac{\alpha}{\beta}\right)^{\frac{m}{2}} ||r_{k-1}|| \leqslant \left(1 - \frac{\alpha}{\beta}\right)^{\frac{km}{2}} ||r_0||$.*
- *Else, the TSIRM algorithm consists in two stages: a first GMRES(m) execution leads to a temporary x_k whose residue satisfies:*

$$||r_k|| \leqslant \left(1 - \frac{\alpha}{\beta}\right)^{\frac{m}{2}} ||r_{k-1}|| \leqslant \left(1 - \frac{\alpha}{\beta}\right)^{\frac{km}{2}} ||r_0||$$

and a least squares resolution. Let $\text{span}(S) = \left\{\sum_{i=1}^{k} \lambda_i v_i \middle| k \in \mathbb{N}, v_i \in S, \lambda_i \in \mathbb{R}\right\}$ be the linear span of a set of real vectors S. So,
$\min_{\alpha \in \mathbb{R}^s} ||b - R\alpha||_2 = \min_{\alpha \in \mathbb{R}^s} ||b - AS\alpha||_2$

$$
\begin{aligned}
&= \min_{x \in span(S_{k-s+1}, S_{k-s+2}, \ldots, S_k)} ||b - AS\alpha||_2 \\
&= \min_{x \in span(x_{k-s+1}, x_{k-s+2}, \ldots, x_k)} ||b - AS\alpha||_2 \\
&\leqslant \min_{x \in span(x_k)} ||b - Ax||_2 \\
&\leqslant \min_{\lambda \in \mathbb{R}} ||b - \lambda A x_k||_2 \\
&\leqslant ||b - Ax_k||_2 \\
&= ||r_k||_2 \\
&\leqslant \left(1 - \frac{\alpha}{\beta}\right)^{\frac{km}{2}} ||r_0||,
\end{aligned}
$$

which concludes the induction and the proof.

4.2. $\mathfrak{R}(A)$ is positive definite

Proposition 3. *Convergence of the TSIRM algorithm is at least linear when $\mathfrak{R}(A)$ is positive definite. Furthermore, the rate of convergence is lower than*

$$
\min\left(\left(1 - \frac{{\lambda_{min}^{\mathfrak{R}(A)}}^2}{\lambda_{min}^{\mathfrak{R}(A)}\lambda_{max}^{\mathfrak{R}(A)} + {\lambda_{max}^{\mathfrak{I}(A)}}^2}\right)^{\frac{m}{2}} ; \left(1 - \frac{{\lambda_{min}^{\mathfrak{R}(A)}}^2}{||A||^2}\right)^{\frac{m}{2}}\right),
$$

where λ_{min}^X (resp. λ_{max}^X) is the lowest (resp. largest) eigenvalue of matrix X.

Proof 3. *If $\mathfrak{R}(A)$ is positive definite, then it is positive, and so the TSIRM algorithm is convergent due to Proposition 1.*

Furthermore, as stated in the proof of Proposition 1, the GMRES residue is under control when $\mathfrak{R}(A)$ is positive. More precisely, it has been proven in the literature that the residual norm provided at the m-th step of GMRES satisfies:

1. $||r_m|| \leqslant \left(1 - \frac{{\lambda_{min}^{\mathfrak{R}(A)}}^2}{\lambda_{min}^{\mathfrak{R}(A)}\lambda_{max}^{\mathfrak{R}(A)} + {\lambda_{max}^{\mathfrak{I}(A)}}^2}\right)^{\frac{mk}{2}} ||r_0||$, *see,* e.g., *[16],*
2. $||r_m|| \leqslant \left(1 - \frac{{\lambda_{min}^{\mathfrak{R}(A)}}^2}{||A||^2}\right)^{\frac{mk}{2}} ||r_0||$, *see [17],*

which proves the convergence of GMRES(m) for all m under such assumptions regarding A.

We will now prove by a mathematical induction, and following the same canvas than in the proof of Prop. 1, that: for each $k \in \mathbb{N}^*$, the TSIRM-residual

norm satisfies

$$||r_k|| \leqslant \min\left(\left(1 - \frac{{\lambda_{min}^{\Re(A)}}^2}{\lambda_{min}^{\Re(A)}\lambda_{max}^{\Re(A)} + {\lambda_{max}^{\Im(A)}}^2}\right)^{\frac{m}{2}}; \left(1 - \frac{{\lambda_{min}^{\Re(A)}}^2}{||A||^2}\right)^{\frac{m}{2}}\right)||r_0|| \tag{4}$$

when A is positive definite.

The base case is obvious, as for $k = 1$, the TSIRM algorithm simply consists in applying GMRES(m) once, leading to a new residual r_1 that follows the inductive hypothesis due to the results recalled in the items listed above.

Suppose now that the claim holds for all $u = 1, 2, \ldots, k-1$, that is, $\forall u \in \{1, 2, \ldots, k-1\}$, $||r_u|| \leqslant \left(1 - \frac{{\lambda_{min}^{\Re(A)}}^2}{||A||^2}\right)^{\frac{mu}{2}} ||r_0||$. We will show that the statement holds for r_k too. Two situations can occur:

- *If $k \not\equiv 0 \ (mod\ m)$, then the TSIRM algorithm consists in executing GMRES once. In that case and by using the inductive hypothesis, we obtain $||r_k|| \leqslant \left(1 - \frac{{\lambda_{min}^{\Re(A)}}^2}{\lambda_{min}^{\Re(A)}\lambda_{max}^{\Re(A)} + {\lambda_{max}^{\Im(A)}}^2}\right)^{\frac{m}{2}} \leqslant \left(1 - \frac{{\lambda_{min}^{\Re(A)}}^2}{\lambda_{min}^{\Re(A)}\lambda_{max}^{\Re(A)} + {\lambda_{max}^{\Im(A)}}^2}\right)^{\frac{mk}{2}} ||r_0||$, due to [16]. Furthermore, we have too that: $||r_k|| \leqslant \left(1 - \frac{{\lambda_{min}^{\Re(A)}}^2}{||A||^2}\right)^{\frac{m}{2}} ||r_{k-1}|| \leqslant \left(1 - \frac{{\lambda_{min}^{\Re(A)}}^2}{||A||^2}\right)^{\frac{mk}{2}} ||r_0||$, as proven in [17] and by using the inductive hypothesis. So we can conclude that*

$$||r_k|| \leqslant \min\left(\left(1 - \frac{{\lambda_{min}^{\Re(A)}}^2}{\lambda_{min}^{\Re(A)}\lambda_{max}^{\Re(A)} + {\lambda_{max}^{\Im(A)}}^2}\right)^{\frac{mk}{2}}; \left(1 - \frac{{\lambda_{min}^{\Re(A)}}^2}{||A||^2}\right)^{\frac{mk}{2}}\right) \times ||r_0||.$$

- *Else, the TSIRM algorithm consists in two stages: a first GMRES(m) execution leads to a temporary x_k whose residue satisfies, following the*

previous item:

$$
\begin{aligned}
||r_k|| &\leqslant \min\left(\left(1-\frac{{\lambda_{min}^{\Re(A)}}^2}{\lambda_{min}^{\Re(A)}\lambda_{max}^{\Re(A)}+{\lambda_{max}^{\Im(A)}}^2}\right)^{\frac{m}{2}};\right.\\
&\quad \left.\left(1-\frac{{\lambda_{min}^{\Re(A)}}^2}{||A||^2}\right)^{\frac{m}{2}}\right)\times||r_{k-1}||\\
&\leqslant \min\left(\left(1-\frac{{\lambda_{min}^{\Re(A)}}^2}{\lambda_{min}^{\Re(A)}\lambda_{max}^{\Re(A)}+{\lambda_{max}^{\Im(A)}}^2}\right)^{\frac{mk}{2}};\right.\\
&\quad \left.\left(1-\frac{{\lambda_{min}^{\Re(A)}}^2}{||A||^2}\right)^{\frac{mk}{2}}\right)\times||r_0||
\end{aligned}
$$

and the least squares resolution of $\min_{\alpha\in\mathbb{R}^s}||b-R\alpha||_2$.

Let $\mathrm{span}(S)=\left\{\sum_{i=1}^{k}\lambda_i v_i \middle| k\in\mathbb{N}, v_i\in S, \lambda_i\in\mathbb{R}\right\}$ *be the linear span of a set of real vectors* S*, as defined previously. So,*

$$
\begin{aligned}
&\min_{\alpha\in\mathbb{R}^s}||b-R\alpha||_2=\min_{\alpha\in\mathbb{R}^s}||b-AS\alpha||_2\\
&\quad=\min_{x\in span(S_{k-s+1},S_{k-s+2},\ldots,S_k)}||b-AS\alpha||_2\\
&\quad=\min_{x\in span(x_{k-s+1},x_{k-s+2},\ldots,x_k)}||b-AS\alpha||_2\\
&\quad\leqslant\min_{x\in span(x_k)}||b-Ax||_2\\
&\quad\leqslant\min_{\lambda\in\mathbb{R}}||b-\lambda Ax_k||_2\\
&\quad\leqslant||b-Ax_k||_2\\
&\quad=||r_k||_2\\
&\quad\leqslant\min\left(\left(1-\frac{{\lambda_{min}^{\Re(A)}}^2}{\lambda_{min}^{\Re(A)}\lambda_{max}^{\Re(A)}+{\lambda_{max}^{\Im(A)}}^2}\right)^{\frac{mk}{2}};\right.\\
&\quad\left.\left(1-\frac{{\lambda_{min}^{\Re(A)}}^2}{||A||^2}\right)^{\frac{mk}{2}}\right)\times||r_0||
\end{aligned}
$$

due to the inductive hypothesis. So the statement of Equation (4) *holds for the* k*-th iterate too, which concludes the induction and the proof.*

4.3. A last linear convergence

Proposition 4. *Let us define the field of values of A by*

$$\mathfrak{F}(A) = \left\{ \frac{x^*Ax}{x^*x}, x \in \mathbb{C}^n \setminus \{0\} \right\}.$$

Then if $\mathfrak{F}(A)$ is included into a closed ball of radius r and center c, which does not contain the origin, then the convergence of the TSIRM algorithm is at least linear. More precisely, the rate of convergence is lower than $2\frac{r}{|c|}$.

Proof 4. *This inequality comes from the fact that, in the conditions of the proposition, the GMRES residue satisfies the inequality: $|r_k| \leqslant 2\frac{r}{|c|}^k |r_0|$. An induction inspired by the proofs of Propositions 2 and 3 can transfer this inequality to the TSIRM residue.*

Remark that a similar proposition can be formulated at each time the given solver satisfies an inequality of the form $||r_n|| \leqslant \mu^n ||r_0||$, with $|\mu| < 1$. Furthermore, it is *a priori* possible in some particular cases regarding A, that the proposed TSIRM converges while the GMRES(m) does not.

5. Experiments using PETSc

In this section four kinds of experiments have been performed. First, some experiments on real matrices issued from the Sparse Matrix Collection of the University of Florida, called Davis collection [18], have been achieved out. Second, some experiments in parallel with some linear problems are reported and analyzed. Third, some experiments conducted in parallel with some nonlinear problems are illustrated. Finally some parameters of TSIRM are studied in order to understand their influences.

It should be noticed that preconditioners ILU (Incomplete LU), SOR (Successive Over Relaxation), MG (Multigrid), ASM (Additive Schwarz Method) and BJAC (Block Jacobi)) applied in all the experiments conducted in this paper are used with their default parameters as they are implemented in PETSc.

5.1. Real matrices

In order to see the behavior of our approach when considering only one processor, a first comparison with GMRES or FGMRES and the new algorithm detailed previously has been experimented. Matrices that have been used with their characteristics (names, fields, number of rows, and nonzero coefficients) are detailed in Table 1. These latter, which are real-world applications matrices, have been extracted from the Davis collection [18].

Matrix name	Field	# Rows	# Nonzeros
crashbasis	Optimization	160,000	1,750,416
parabolic_fem	Comput. fluid dynamics	525,825	2,100,225
epb3	Thermal problem	84,617	463,625
atmosmodj	Comput. fluid dynamics	1,270,432	8,814,880
bfwa398	Electromagnetics pb	398	3,678
torso3	2D/3D problem	259,156	4,429,042

Table 1: Main characteristics of the sparse matrices chosen from the Davis collection

Chosen parameters are detailed below. We have stopped the GMRES every 30 iterations (*i.e.*, $mxIt_{kryl} = 30$), which is the default setting of GMRES restart parameter. The parameter s has been set to 8. CGLS minimizes the least-squares problem with parameters $tol_{ls} = 10^{-40}$ and $mxIt_{ls} = 20$. The external precision is set to $tol_{tsirm} = 10^{-10}$. These experiments have been performed on an Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz with the 3.5.1 version of PETSc.

Experiments comparing a GMRES variant with TSIRM in the resolution of linear systems are given in Table 2. The second column describes whether GMRES or FGMRES has been used for linear systems solving. Different preconditioners have been used according to the matrices. With TSIRM, the same solver and the same preconditioner are used. Table 2 shows that TSIRM can drastically reduce the number of iterations needed to reach the convergence, when the number of iterations for the normal GMRES is more or less greater

than 500. In fact this also depends on two parameters: the number of iterations before stopping GMRES and the number of iterations to perform the minimization.

Matrix name	Solver/precond	GMRES		TSIRM CGLS	
		Time	# Iter.	Time	# Iter.
crashbasis	GMRES / none	15.65	518	14.12	450
parabolic_fem	GMRES / ILU	1,009.94	7,573	401.52	2,970
epb3	FGMRES / SOR	8.67	600	8.21	540
atmosmodj	FGMRES / SOR	104.23	451	88.97	366
bfwa398	GMRES / none	1.42	9,612	0.28	1,650
torso3	FGMRES / SOR	37.70	565	34.97	510

Table 2: Comparison between sequential standalone TSIRM using (F)GMRES with (F)GMRES (time in seconds).

5.2. Parallel linear problems

In order to perform larger experiments, we have tested some example applications of PETSc. These applications are available in the *ksp* part [19], which is suited for scalable linear equations solvers:

- *ksp.ex15* on is an example that solves in parallel an operator using a finite difference scheme to discretize a 2D Laplacian operator. This example is used in many physical phenomena, for example, heat and fluid flow, wave propagation, etc.
- *ksp.ex54* is another example that solves the 2D Laplacian operator with quadrilateral finite elements (the standard five point stencil). In this example, the user can define the scaling of material coefficient in an embedded circle called α.

It should be noticed that both examples use positive-defined matrices.

For more technical details on these applications, interested readers are invited to read the codes available in the PETSc sources. These problems have been chosen because they are scalable with many cores.

In the following, larger experiments are described on two large scale architectures: Curie and Juqueen. Both these architectures are supercomputers respectively composed of 80,640 cores for Curie and 458,752 cores for Juqueen. Those machines are respectively hosted by GENCI in France and Jülich Supercomputing Center in Germany. They belong with other similar architectures to the PRACE initiative (Partnership for Advanced Computing in Europe), which aims at proposing high performance supercomputing architecture to enhance research in Europe. The Curie architecture is composed of Intel E5-2680 processors at 2.7 GHz with 2Gb memory by core. The Juqueen architecture, for its part, is composed by IBM PowerPC A2 at 1.6 GHz with 1Gb memory per core. Both those architectures are equipped with a dedicated high speed network.

In many situations, using preconditioners is essential in order to find the solution of a linear system. There are many preconditioners available in PETSc. However, for parallel applications, all the preconditioners based on matrix factorization are not available. In our experiments, we have tested different kinds of preconditioners, but as it is not the subject of this paper, we will not present results with many preconditioners. In practice, we have chosen to use a multigrid (MG) and successive over-relaxation (SOR). For further details on the preconditioners in PETSc, readers are referred to [15].

Table 3 shows the execution times and the number of iterations of example *ksp.ex15* of PETSc on the Juqueen architecture. Different numbers of cores are studied ranging from 2,048 up-to 16,383 with the two preconditioners MG and SOR. In these experiments, the number of components (or unknowns of the problems) per core is fixed at 25,000, also called weak scaling. This number can seem relatively small. In fact, for some applications that need a lot of memory, the number of components per processor requires sometimes to be small. Moreover, with a small number of components per core, the scalability is more difficult to obtain, which is interesting to show the efficiency of our

nb. cores	precond	FGMRES		TSIRM CGLS		TSIRM LSQR		best gain
		Time	# Iter.	Time	# Iter.	Time	# Iter.	
2,048	MG	403.49	18,210	73.89	3,060	77.84	3,270	5.46
2,048	SOR	745.37	57,060	87.31	6,150	104.21	7,230	8.53
4,096	MG	562.25	25,170	97.23	3,990	89.71	3,630	6.27
4,096	SOR	912.12	70,194	145.57	9,750	168.97	10,980	6.26
8,192	MG	917.02	40,290	148.81	5,730	143.03	5,280	6.41
8,192	SOR	1,404.53	106,530	212.55	12,990	180.97	10,470	7.76
16,384	MG	1,430.56	63,930	237.17	8,310	244.26	7,950	6.03
16,384	SOR	2,852.14	216,240	418.46	21,690	505.26	23,970	6.82

Table 3: Comparison of TSIRM using FGMRES with FGMRES for example *ksp.ex15* of PETSc/KSP with two preconditioners (MG and SOR) having 25,000 components per core on Juqueen ($tol_{tsirm} = 10^{-3}$, $mxIt_{kryl} = 30$, $s = 12$, $mxIt_{ls} = 15$, $tol_{ls} = 10^{-40}$), time is expressed in seconds.

method. Other parameters for this application are described in the legend of this table.

In Table 3, we can notice that TSIRM is always faster than FGMRES. The last column shows the ratio between FGMRES and the best version of TSIRM according to the minimization procedure: CGLS or LSQR. Even if we have computed the worst case between CGLS and LSQR, it is clear that TSIRM is always faster than FGMRES. For this example, the multigrid preconditioner (MG) is faster than SOR. The gain between TSIRM and FGMRES is more or less similar for the two preconditioners. Looking at the number of iterations to reach the convergence, it is obvious that TSIRM allows the reduction of the number of iterations. It should be noticed that for TSIRM, in these experiments, only the iterations of the Krylov solver are taken into account. Iterations of CGLS or LSQR were not recorded but they are time-consuming. In general, each $mxIt_{kryl} \times s$ iterations which corresponds to 30×12, there are $mxIt_{ls}$ iterations for the least-squares method which corresponds to 15 in this experiment.

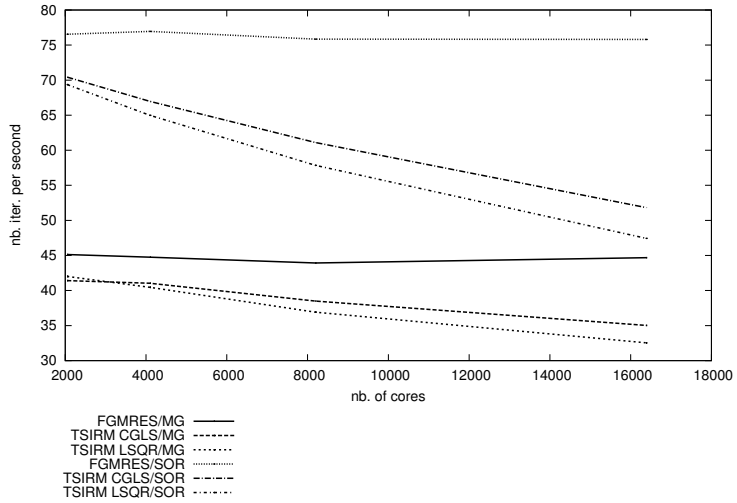


Figure 1: Number of iterations per second with *ksp.ex15* and the same parameters as in Table 3 (weak scaling)

In Figure 1, the number of iterations per second corresponding to Table 3 is displayed. It can be noticed that the number of iterations per second of FGMRES is constant whereas it decreases with TSIRM with both preconditioners. This can be explained by the fact that when the number of cores increases, the time for the least-squares minimization step also increases but, generally, when the number of cores increases, the number of iterations to reach the threshold also increases, and, in that case, TSIRM is more efficient to reduce the number of iterations. So, the overall benefit of using TSIRM is interesting.

In Table 4, some experiments with example *ksp.ex54* on the Curie architecture are reported. For this application, we fixed $\alpha = 0.6$. As it can be seen in that table, the size of the problem has a strong influence on the number of iterations to reach the convergence. That is why we have preferred to change the threshold. If we set it to 10^{-3} as with the previous application, only one iteration is necessary to reach the convergence. So Table 4 shows the results of different executions with different number of cores and different thresholds. As with the previous example, we can observe that TSIRM is faster than FGMRES.

nb. cores	tol_{tsirm}	FGMRES		TSIRM CGLS		TSIRM LSQR		best gain
		Time	# Iter.	Time	# Iter.	Time	# Iter.	
2,048	8×10^{-5}	108.88	16,560	23.06	3,630	22.79	3,630	4.77
2,048	6×10^{-5}	194.01	30,270	35.50	5,430	27.74	4,350	6.99
4,096	7×10^{-5}	160.59	22,530	35.15	5,130	29.21	4,350	5.49
4,096	6×10^{-5}	249.27	35,520	52.13	7,950	39.24	5,790	6.35
8,192	6×10^{-5}	149.54	17,280	28.68	3,810	29.05	3,990	5.21
8,192	5×10^{-5}	785.04	109,590	76.07	10,470	69.42	9,030	11.30
16,384	4×10^{-5}	718.61	86,400	98.98	10,830	131.86	14,790	7.26

Table 4: Comparison of TSIRM using FGMRES with FGMRES for *ksp.ex54* of PETSc/KSP (both with the MG preconditioner) with 25,000 components per core on Curie ($mxIt_{kryl} = 30$, $s = 12$, $mxIt_{ls} = 15$, $tol_{ls} = 10^{-40}$), time is expressed in seconds.

The ratio greatly depends on the number of iterations for FGMRES to reach the threshold: the greater the number of iterations to reach the convergence is, the better the ratio between our algorithm and FGMRES is. This experiment is also a weak scaling with approximately $25,000$ components per core. It can also be observed that the difference between CGLS and LSQR is not significant. Both can be good but it seems not possible to know in advance which one will be the best.

Table 5 shows a strong scaling experiment with example *ksp.ex54* on the Curie architecture. So, in this case, the number of unknowns is fixed at $204,919,225$ and the number of cores ranges from 512 to 8192 with the power of two. The threshold is fixed at 5×10^{-5} and only the MG preconditioner has been tested. Here again we can see that TSIRM is faster than FGMRES. The efficiency of each algorithm is reported (*i.e.* Eff.). It can be noticed that the efficiency of FGMRES is better than the TSIRM one except with $8,192$ cores and that its efficiency is greater than one whereas the efficiency of TSIRM is lower than one. Nevertheless, the ratio of TSIRM with any version of the least-squares method

is always faster. With 8,192 cores when the number of iterations is far more important for FGMRES, we can see that it is only slightly more important for TSIRM.

In Figure 2 we report the number of iterations per second for the experiments reported in Table 5. This figure highlights that the number of iterations per second is more or less the same for FGMRES and TSIRM with a little advantage for FGMRES. It can be explained by the fact that, as we have previously explained, the iterations of the least-squares steps are not taken into account with TSIRM.

nb. cores	FGMRES			TSIRM CGLS			TSIRM LSQR			best gain
	Time	# Iter.	Eff.	Time	# Iter.	Eff.	Time	# Iter.	Eff.	
512	3,969.69	33,120	1	709.57	5,790	1	1622.76	5,070	1	6.37
1024	1,530.06	25,860	1.30	290.95	4,830	1.21	307.71	5,070	1.01	5.25
2048	919.62	31,470	1.08	237.52	8,040	.75	194.22	6,510	.80	4.73
4096	405.60	28,380	1.22	111.67	7,590	.79	91.72	6,510	.84	4.42
8192	785.04	109,590	.32	76.07	10,470	.58	69.42	9,030	.56	11.30

Table 5: Comparison of TSIRM using FGMRES with FGMRES for *ksp.ex54* of PETSc/KSP (both with the MG preconditioner) with 204,919,225 components on Curie with different number of cores ($tol_{tsirm} = 5 \times 10^{-5}$, $mxIt_{kryl} = 30$, $s = 12$, $mxIt_{ls} = 15$, $tol_{ls} = 10^{-40}$), time is expressed in seconds.

5.3. Parallel nonlinear problems

With PETSc, linear solvers are used inside nonlinear solvers. The SNES (Scalable Nonlinear Equations Solvers) module in PETSc implements easy to use methods, like Newton-type, quasi-Newton or full approximation scheme (FAS) multigrid to solve systems of nonlinears equations. As SNES is based on the Krylov methods of PETSc, it is interesting to investigate if the TSIRM method is also efficient and scalable with non linear problems. In PETSc, some examples are provided. An important criteria is the scalability of the initial code

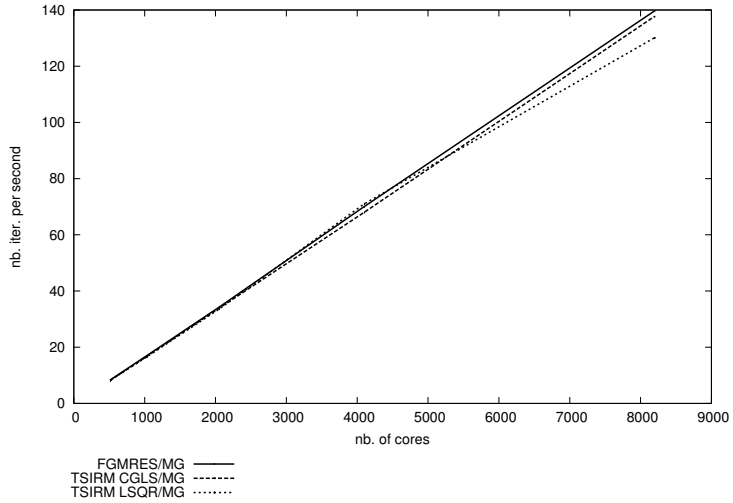


Figure 2: Number of iterations per second with *ksp.ex54* and the same parameters as in Table 5 (strong scaling)

with classical solvers. Consequently, we have chosen two of these examples:

- *snes.ex14* is the code which solves the Bratu (SFI - solid fuel ignition) non-linear partial difference equations in 3 dimension, the matrix is a positive-definite one.
- *snes.ex20* is the code which solves a 3 dimension radiative transport test problem.

For more details on these examples, interested readers are invited to see the code in the PETSc examples [20]. For both these examples, a weak scaling case is chosen where processors have approximately a number of components equals to 100,000.

In Table 6 we report the results of our experiments for the example *snes.ex14* with the block Jacobi preconditioner. For TSIRM the CGLS algorithm is used to solve the minimization step. In this table, we can see that the number of iterations used by the linear solver is smaller with TSIRM compared with FGMRES. Consequently the execution times are smaller with TSIRM. The gain

between TSIRM and FGMRES is around 6 and 7. The parameters of TSIRM are expressed in the caption of the table.

nb. cores	FGMRES/BJAC		TSIRM CGLS/BJAC		gain
	Time	# Iter.	Time	# Iter.	
1,024	159.52	11,584	26.34	1,563	6.06
2,048	226.24	16,459	37.23	2,248	6.08
4,096	391.21	27,794	50.93	2,911	7.69
8,192	543.23	37,770	79.21	4,324	6.86

Table 6: Comparison of TSIRM using FGMRES with FGMRES for *snes.ex14* of PETSc/SNES with a Block Jacobi preconditioner having 100,000 components per core on Curie ($tol_{tsirm} = 10^{-10}$, $mxIt_{kryl} = 30$, $s = 12$, $mxIter_{ls} = 15$, $tol_{ls} = 10^{-40}$), time is expressed in seconds.

In Table 7, the results of the experiments with the example *snes.ex20* are reported. The block Jacobi preconditioner has also been used and CGLS to solve the minimization step for TSIRM. For this example, we can observe that the number of iterations for FGMRES increases drastically when the number of cores increases. With TSIRM, we can see that the number of iterations is initially very small compared to the FGMRES ones and when the number of cores increases, the number of iterations increases slighther with TSIRM than with FGMRES. For this example, the gain between TSIRM and FGMRES ranges between 8 with 1,024 cores to more than 16 with 8,192 cores.

5.4. Influence of parameters for TSIRM

In this section we present the influence of some parameters on the performances of the TSIRM algorithm (the total number of iterations and the execution time). We study the influence of the following parameters: the method to solve the linear least-squares problem in the minimization step, the maximum number of inner iterations $mxIt_{kryl}$, and the size s of matrix AS (*i.e.* matrix S) of the least-squares problem. In the following, we call $nbResMin$ the number of residuals used for the minimization step (*i.e.* the s). We conducted experi-

nb. cores	FGMRES/BJAC		TSIRM CGLS/BJAC		gain
	Time	# Iter.	Time	# Iter.	
1,024	667.92	48,732	81.65	5,087	8.18
2,048	966.87	77,177	90.34	5,716	10.70
4,096	1,742.31	124,411	119.21	6,905	14.61
8,192	2,739.21	187,626	168.9	9,000	16.22

Table 7: Comparison of TSIRM using FGMRES with FGMRES for *snes.ex20* of PETSc/SNES with a Block Jacobi preconditioner having 100,000 components per core on Curie ($tol_{tsirm} = 10^{-10}$, $mxIt_{kryl} = 30$, $s = 12$, $mxIt_{ls} = 15$, $tol_{ls} = 10^{-40}$), time is expressed in seconds.

ments on a processor of 16 cores to solve linear and nonlinear problems of size $200,000$ components per core. We solved problems token from examples *ksp* [19] and *snes* [20] of PETSc. We took two examples of linear problems: *ksp.ex15* and *ksp.34*, and three examples of nonlinear problems: *snes.ex14*, *snes.ex20* and *snes.ex35*. These examples were previously described except *ksp.ex34* and *snes.ex35*. The general descriptions of these latter examples are:

- *ksp.ex34* solves an example based on a 3D Laplacian operator.
- *snes.ex35* solves a Laplacian u = b as a nonlinear problem.

It should be noticed that all these examples use positive-defined matrices.

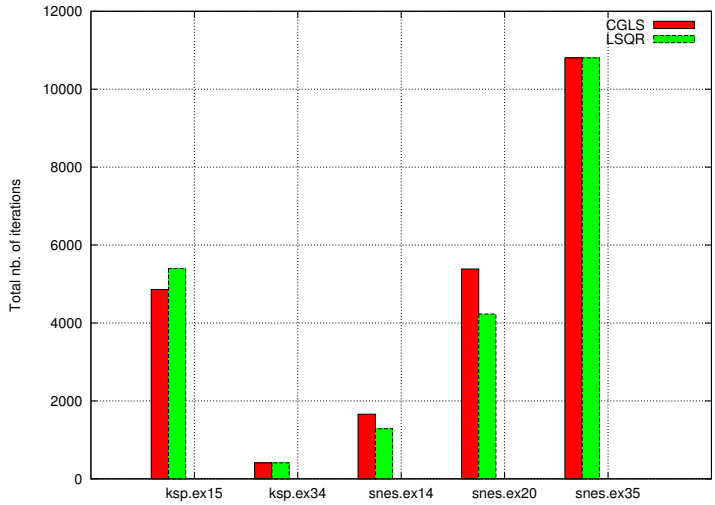
We fixed some parameters in the TSIRM algorithm as follows: the nonlinear systems are solved with a precision of 10^{-8}, the tolerance threshold in the TSIRM algorithm is $tol_{tsirm} = 10^{-10}$, the FGMRES method is used as the inner solver with a tolerance threshold $tol_{kryl} = 10^{-10}$, the additive Schwarz method (ASM) is used as a preconditioner for the FGMRES method, and the least-squares problem in the minimization step is solved with a precision $tol_{ls} = 10^{-40}$ and a maximum number of iterations $mxIt_{ls} = 20$.

Figure 3 shows the total number of iterations and the execution time to reach the convergence by using two different methods CGLS and LSQR in the minimization step. We can see from Figure 3 that both the CGLS and LSQR

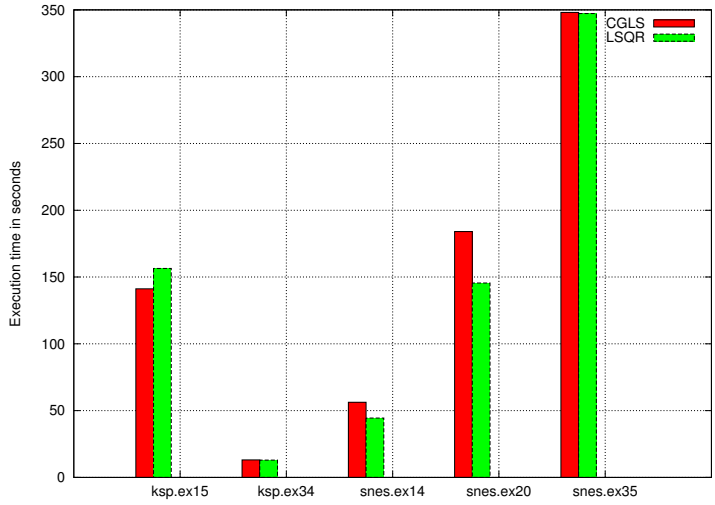
methods might lead to different results depending on the nature of the problem. In these experiments, it is more interesting to use the TSIRM algorithm with CGLS to solve linear systems (about 10%) and the TSIRM algorithm with LSQR to solve nonlinear systems (about 22%). LSQR is known to use less memory storage than CGLS and it is likely to obtain more accurate solutions in fewer iterations for ill-conditioned problems [21]. However, CGLS is more efficient in term of the number of floating-point operations per iteration.

In the following, we show the performances of the TSIRM algorithm, the total number of iterations and the execution time, in 2D figures by varying both parameters: the maximum number of inner iterations $mxIt_{kryl}$ and the number of residuals for the minimization $nbResMin$. In all experiments whose results are shown in Figures 4, 5, 6, 7, and 8, we used the LSQR method in the minimization step. It should be noticed that the total number of iterations and the execution time in Figures 4 and 8 at $mxIt_{kryl} = 10$ and $nbResMin = 2$ are put to the maximum values compared to other configurations, because both examples *ksp.ex15* and *snes.ex35* converge very slowly. One should also remark that values are interpolated in these figures for all values inside computed values (in order to obtain beautiful figures). For all these figures except for Figure 5, the total number of iterations is highly correlated to the execution times. For Figure 5, it is not the case because the number of iterations is quite small.

The best configuration for each figure is given on Table 8. We can see that examples *ksp.ex15* and *snes.ex35* need more inner iterations due to their slow convergence. The best configuration ($mxIt_{kryl}$,$nbResMin$) depends on the nature and the convergence of the application. We can conclude from these experiments and for these sizes of problems that the good convergence might be reached for a small number of $nbResMin$ when the number of inner iterations is enough to compute accurate vectors for matrix S. In fact, in order to improve the convergence, the TSIRM algorithm executes $nbResMin$ (*i.e.* s) times a few iterations (*i.e.* $mxIt_{kryl}$) of inner solver FGMRES, performs the residual minimization on $nbResMin$ solutions, and restarts the resolution with accurate data. In this case of applications having a slow convergence, it is better to



(a) Total number of iterations



(b) Execution time

Figure 3: The total number of iterations and the execution time in seconds using two different methods for the minimization: CGLS and LSQR ($tol_{tsirm} = 10^{-10}$, $tol_{kryl} = 10^{-10}$, $mxIt_{kryl} = 30$, $s = 10$, $tol_{ls} = 10^{-40}$, $mxIt_{ls} = 20$), ASM preconditioner

execute enough inner iterations and build a small matrix S with accurate vectors than to execute a small number of inner iterations and build a large matrix S. The latter situation leads to perform a residual minimization on inaccurate solutions.

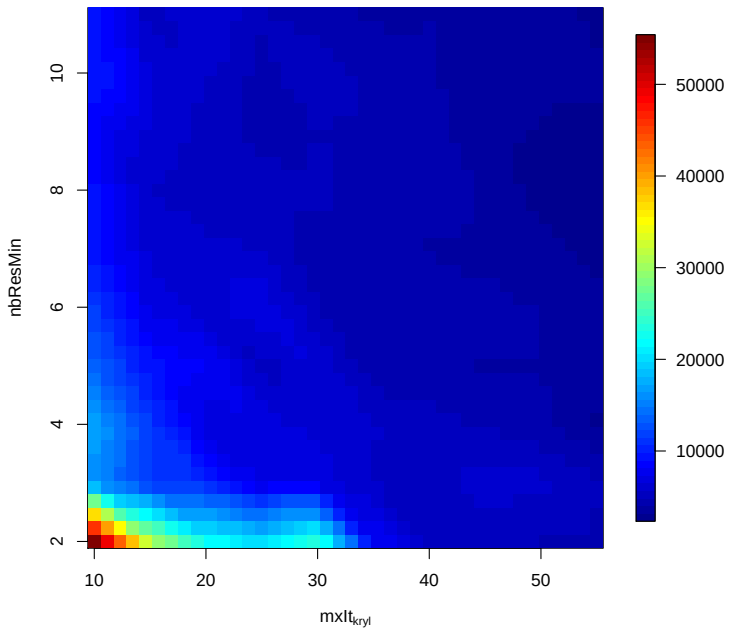
Problem	$mxIt_{kryl}$	$nbResMin$	Total nb. of iter.	Execution time (s)
ksp.ex15	55	7	2730	43.06
ksp.ex34	35	4	300	9.80
snes.ex14	35	4	927	16.31
snes.ex20	35	5	3019	52.34
snes.ex35	55	2	8034	126.13

Table 8: The best configuration for each example: *ksp.ex15*, *ksp.ex34*, *snes.ex14*, *snes.ex20* and *snes.ex35*.

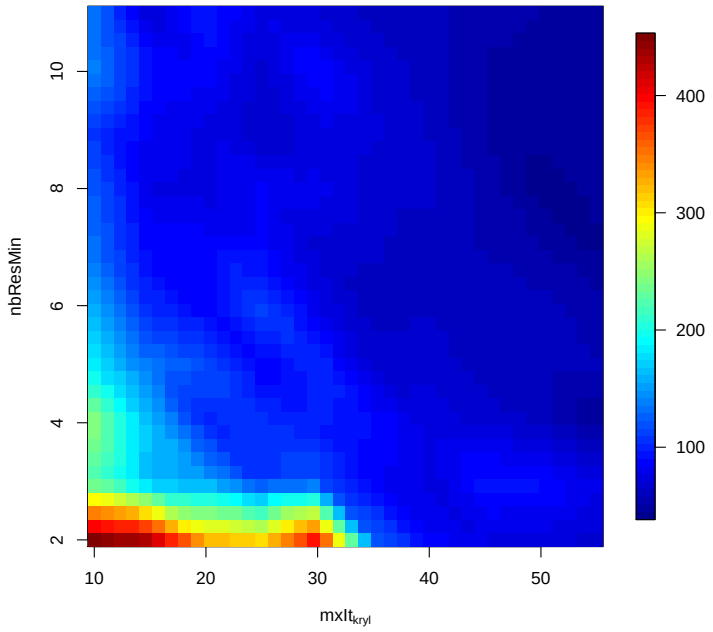
5.5. Influence of the least-square iteration on TSIRM

In this part, we have measured the execution time of the least-square (LS) iteration on some examples and we have compared it with the total execution time of the TSIRM method. We have also compared the number of least-square iterations with the total number of iterations. In fact we consider that the total number of iterations is the sum of all the iterations of the inner solver (i.e. the Krylov solver). As for these experiments we did not have access to the supercomputers used previously, we have performed them on a small machine composed of a bi-processor Xeon(R) CPU E5-2640 v3 @ 2.60GHz. This machine contains 16 real cores. For each tested problem of PETSc, the parameters are given below. It must be noticed that FMGRES has always been used as the inner solver:

- ksp.ex15: size of the problem: $2,000^2$, $tol_{tsirm} = 10^{-6}$, $mxIt_{kryl} = 30$, $mxIt_{ls} = 15$, $tol_{ls} = 10^{-40}$, LS solver: CGLS, preconditioner: mg

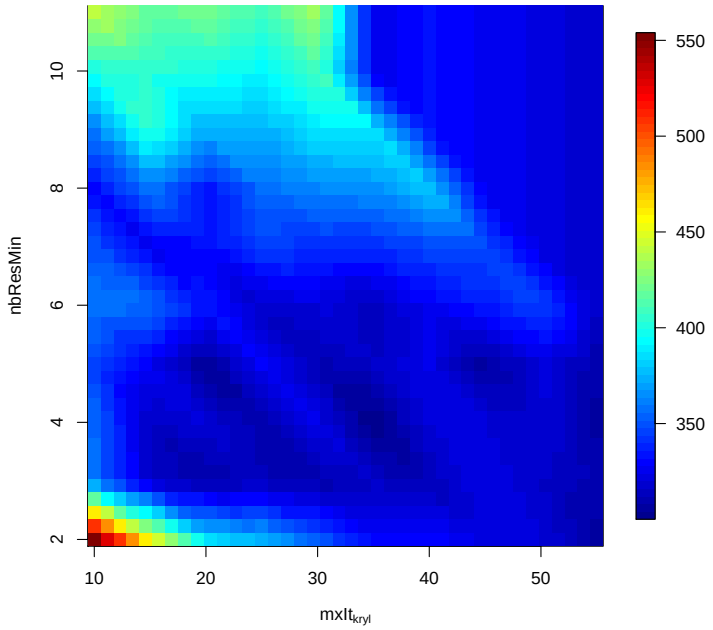


(a) Total number of iterations

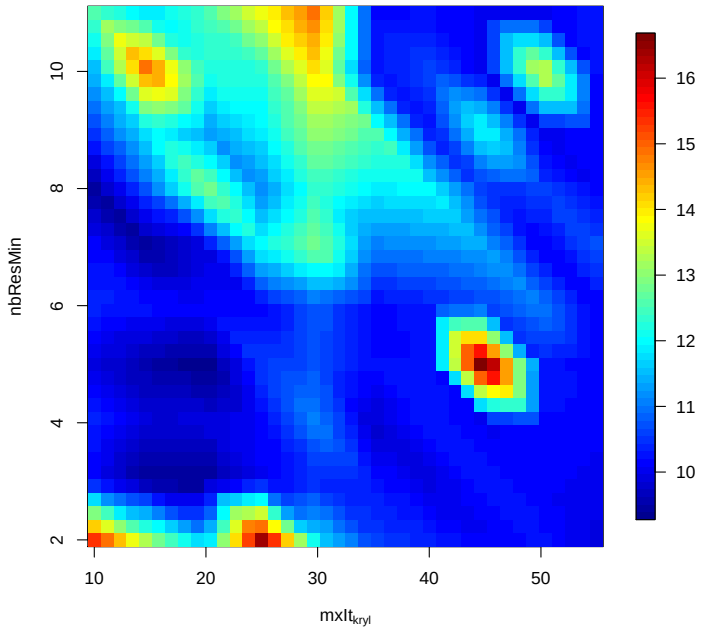


(b) Execution time

Figure 4: The total number of iterations and the execution time in seconds in example *ksp.ex15* of PETSc by varying the number of inner iterations and the size of the least-squares problem.

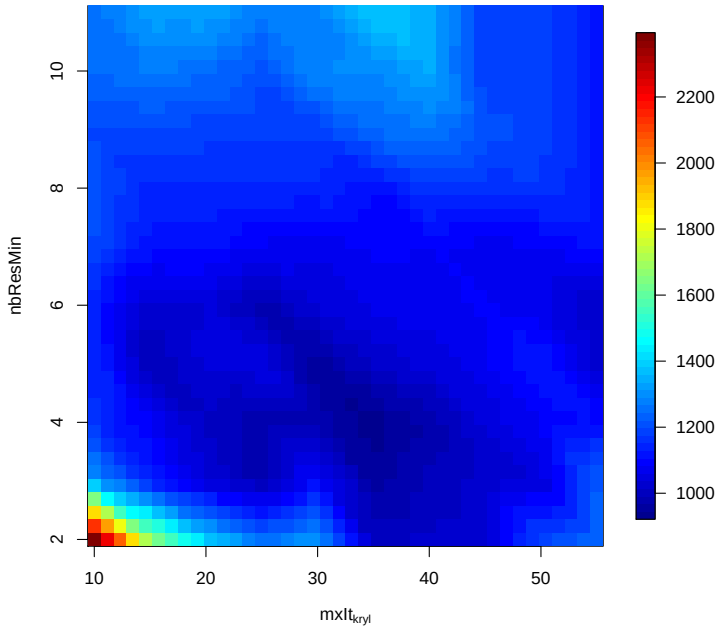


(a) Total number of iterations

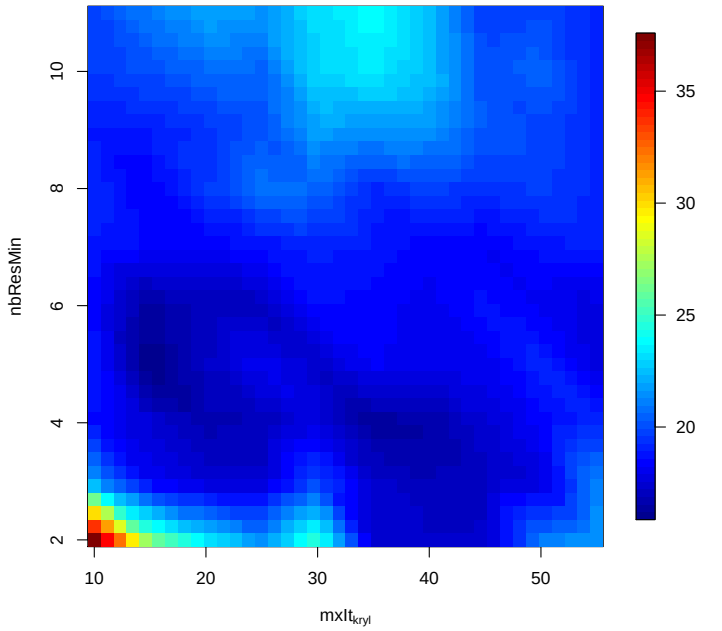


(b) Execution time

Figure 5: The total number of iterations and the execution time in seconds in example *ksp.ex34* of PETSc by varying the number of inner iterations and the size of the least-squares problem.

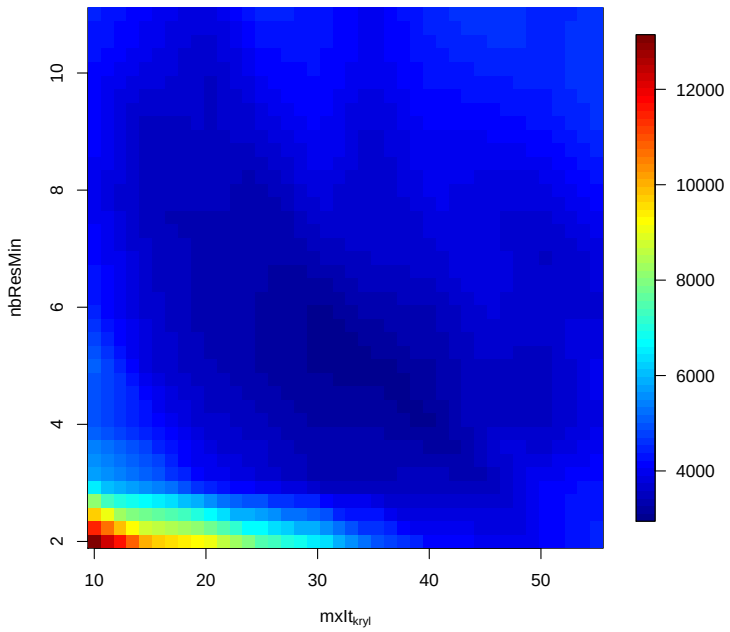


(a) Total number of iterations

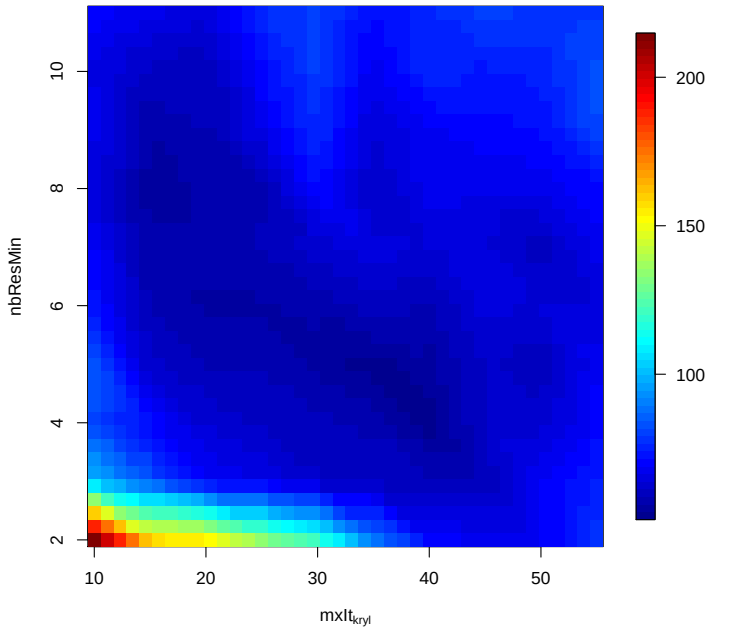


(b) Execution time

Figure 6: The total number of iterations and the execution time in seconds in example *snes.ex14* of PETSc by varying the number of inner iterations and the size of the least-squares problem.

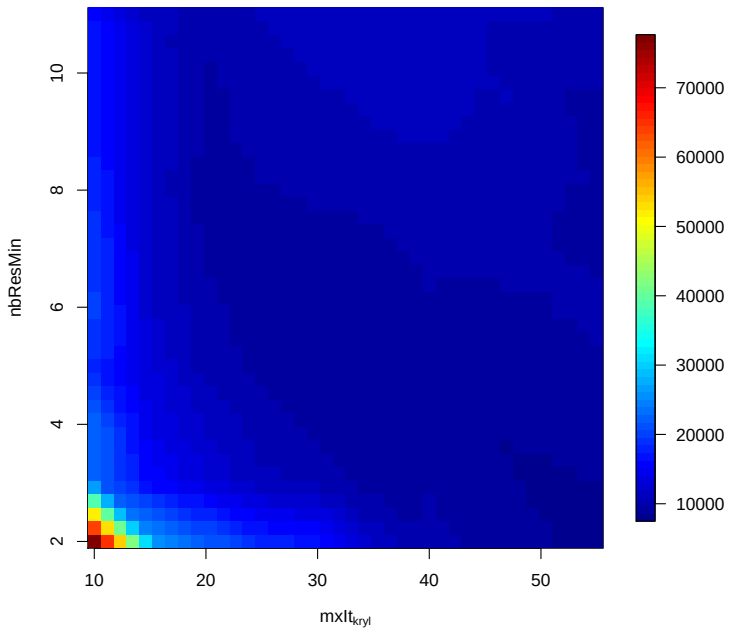


(a) Total number of iterations

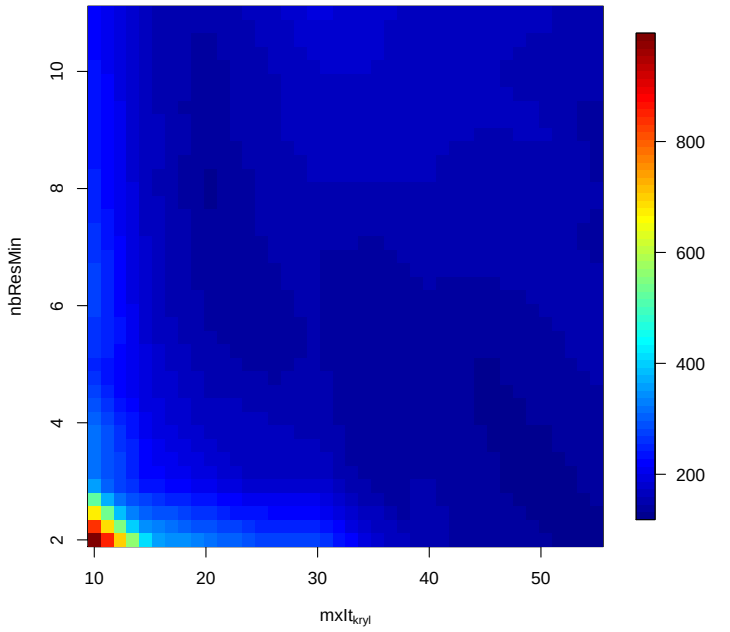


(b) Execution time

Figure 7: The total number of iterations and the execution time in seconds in example *snes.ex20* of PETSc by varying the number of inner iterations and the size of the least-squares problem.



(a) Total number of iterations



(b) Execution time

Figure 8: The total number of iterations and the execution time in seconds in example *snes.ex35* of PETSc by varying the number of inner iterations and the size of the least-squares problem.

- ksp.ex45: size of the problem: 200^3, $tol_{tsirm} = 10^{-10}$, $mxIt_{kryl} = 30$, $mxIt_{ls} = 15$, $tol_{ls} = 10^{-40}$, LS solver: CGLS, preconditioner: asm

- snes.ex14 and snes.ex20: size of the problem: 200^3, $tol_{tsirm} = 10^{-12}$, $mxIt_{kryl} = 30$, $mxIt_{ls} = 15$, $tol_{ls} = 10^{-40}$, LS solver: CGLS, preconditioner: block-jacobi.

In Table 9, experiments run with 16 cores are reported. In this table, the example of PETSc, the total execution time, the execution time of the least-square (LS) method, the total number of iterations, and the number of iterations of LS are provided. We can see that the minimization step based on the least-square method is not a time-consuming task in the TSIRM solver. In these experiments, the minimization step represents about 2% to 3% of the total execution time and the total number of iterations respectively.

Problem	Total exec. time (s)	LS exec. time (s)	Total nb. of iter.	Nb. of iter. of the LS
ksp.ex15	82.65	1.11	1,200	45
ksp.ex45	69.75	0.99	510	15
snes.ex14	95.06	2.39	1,200	45
snes.ex20	358.91	6.91	3,090	105

Table 9: Comparison of the execution times and number of iterations of the TSIRM method and of the Least-Square method for some problems on a 16 cores machine.

6. Conclusion

In this paper a new two-stage algorithm TSIRM has been described. This method allows us to improve the convergence of Krylov iterative methods. It is based on a least-squares minimization step which uses the Krylov residuals.

We have implemented our code in PETSc in order to show that it is efficient and scalable. Some experiments with classical examples of PETSc for linear and nonlinear problems have been performed. We have observed that TSIRM

outperforms GMRES variants when the number of iterations is large. TSIRM is also scalable since we made some experiments with up to 16,394 cores.

We have also observed that TSIRM is efficient with different preconditioners. The influence of some important parameters for TSIRM have been studied (the number of inner iterations, the number of residuals used for the minimization step, ...). It can be noticed that they have a strong influence on the convergence speed.

In future works, we plan to study other problems coming from different research areas. Other efficient Krylov optimisation methods as communication avoiding techniques may be interesting to investigate.

Acknowledgment

This paper is partially funded by the Labex ACTION program (contract ANR-11-LABX-01-01). We acknowledge PRACE for awarding us access to resources Curie and Juqueen respectively based in France and Germany. We also thank the supercomputer facilities of the Mésocentre de calcul de Franche-Comté.

References

References

[1] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.

[2] J. Bahi, S. Contassot-Vivier, R. Couturier, Parallel iterative algorithms: from sequential to grid computing, in: Numerical Analysis and Scientific Computing, Chapman & Hall/CRC, 2008.

[3] N. K. Nichols, On the convergence of two-stage iterative processes for solving linear equations, SIAM Journal on Numerical Analysis 10 (3) (1973) 460–469.

[4] J. Meijerink, H. V. d. Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, Mathematics of Computation 31 (137) (1977) 148–162.

[5] Y. Saad, M. Schultz, GMRES : a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Stat. Comput. 7 (3) (1986) 856–869.

[6] Y. Huang, H. Van der Vorst, Some observations on the convergence behavior of GMRES, Report 89–09, Delft Univ. Technology (1989).

[7] H. A. Van der Vorst, C. Vuik, GMRESR: a family of nested GMRES methods, Numerical Linear Algebra with Applications 1 (4) (1994) 369–386. doi:10.1002/nla.1680010404.
URL http://dx.doi.org/10.1002/nla.1680010404

[8] Y. Saad, A flexible inner-outer preconditioned gmres algorithm, SIAM J. Sci. Comput. 14 (2) (1993) 461–469. doi:10.1137/0914028.
URL http://dx.doi.org/10.1137/0914028

[9] M. Mohiyuddin, M. Hoemmen, J. Demmel, K. Yelick, Minimizing communication in sparse matrix solvers, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09, ACM, New York, NY, USA, 2009, pp. 36:1–36:12. doi:10.1145/1654059.1654096.
URL http://doi.acm.org/10.1145/1654059.1654096

[10] M. Hoemmen, Communication-avoiding krylov subspace methods, Ph.d. thesis, University of California, Berkeley (2010).
URL http://www.cs.berkeley.edu/~mhoemmen/pubs/thesis.pdf

[11] I. Yamazaki, H. Anzt, S. Tomov, M. Hoemmen, J. Dongarra, Improving the performance of ca-gmres on multicores with multiple gpus, in: Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium, IPDPS '14, IEEE Computer Society, Washington, DC, USA,

2014, pp. 382–391. doi:10.1109/IPDPS.2014.48.
URL http://dx.doi.org/10.1109/IPDPS.2014.48

[12] Björck, A., Numerical Methods for Least Squares Problems, SIAM Publications, 1996.

[13] C. C. Paige, M. A. Saunders, LSQR: an algorithm for sparse linear equations and sparse least squares, ACM Trans. Math. Softw. 8 (1) (1982) 43–71. doi:10.1145/355984.355989.
URL http://doi.acm.org/10.1145/355984.355989

[14] R. Couturier, L. Ziane Khodja, A scalable multisplitting algorithm to solve large sparse linear systems, The journal of SupercomputingOnline version, 10.1007/s11227-014-1367-7.

[15] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, H. Zhang, PETSc Web page, http://www.mcs.anl.gov/petsc (2014).
URL http://www.mcs.anl.gov/petsc

[16] Elman, Silvester, Wathen, Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics, 1st Edition, Oxford University Press, 2005.

[17] M. Eiermann, O. G. Ernst, Geometric aspects of the theory of krylov subspace methods, Acta Numerica 10 (2001) 251–312. doi:10.1017/S0962492901000046.

[18] T. Davis, Y. Hu, The University of Florida sparse matrix collection, digest, http://www.cise.ufl.edu/research/sparse/matrices/ (1997).

[19] Krylov Methods - KSP: Examples, http://www.mcs.anl.gov/petsc/petsc-dev/src/ksp/ksp/examples/tutorials/, accessed: 2015-09-23.

[20] Nonlinear solvers - SNES: Examples, http://www.mcs.anl.gov/petsc/petsc-current/src/snes/examples/tutorials/index.html, accessed: 2015-09-23.

[21] C. C. Paige, M. A. Saunders, Algorithm 583: LSQR: Sparse linear equations and least squares problems, ACM Trans. Math. Softw. 8 (2) (1982) 195–209. doi:10.1145/355993.356000.
URL http://doi.acm.org/10.1145/355993.356000