

Iterations emboîtées pour solveurs modaux en mécanique des structures: application au Code_Aster

Contrat EDF-CERFACS No I72/E26316/MTI0077

Lot 1 – Rapport de contrat FR/PA/01/85

F. Chaitin-Chatelin¹ et T. Meškauskas²

Novembre 2001

¹ Université Toulouse I et CERFACS
Email: chatelin@cerfacs.fr

² CERFACS
Email: meska@cerfacs.fr

Centre Européen de Recherche et de Formation Avancée en Calcul
Scientifique, 42 Av. G. Coriolis, 31057 Toulouse, France.

Inner-outer iterations for mode solvers in structural mechanics: application to the Code_Aster

Contract EDF-CERFACS No I72/E26316/MTI0077

Deliverable 1 – Contract Report FR/PA/01/85

F. Chaitin-Chatelin¹ and T. Meškauskas²

November 2001

¹ Université Toulouse I and CERFACS
Email: chatelin@cerfacs.fr

² CERFACS
Email: meska@cerfacs.fr
CERFACS, 42 Av. G. Coriolis, 31057 Toulouse, France.

Contents

1	Improving convergence of iterative inner-outer solvers by a relaxation strategy	2
2	Implicitly restarted Arnoldi method with shifts for a generalized eigenvalue problem	4
2.1	Description of the problem proposed by EDF	4
2.2	Modification of Code_Aster – RLDLGG \rightarrow GCPC	5
2.2.1	Technical details and level of preconditioning	5
2.2.2	Remark on stopping criterion in GCPC	6
2.3	Implementation of a relaxation strategy	7
2.3.1	Control of when to stop the inner loop (subroutine GCPC)	7
2.3.2	Modification of Code_Aster – monitoring of backward error	8
2.4	Summary of changes implemented in Code_Aster	9
3	Numerical results	10
	Test-case ahlv100a	13
	Test-case ahlv100h	14
	Test-case sdls04b	15
	Test-case sdll11b	16
	Test-case tpam3	17
	Acknowledgments	12
	Bibliography	18

1 Improving convergence of iterative inner-outer solvers by a relaxation strategy

In this section we give an introduction to inner-outer Krylov type methods and discuss the possibility to apply the so called relaxation strategy to them.

By inner-outer iterations one means the situation when one iterative solver, referred to as the inner solver, is embedded into another one, referred to as the outer solver. Such an approach naturally arises for generalized eigenproblems approximated by Krylov type methods: in each outer step (which is a step of the eigensolver) we need to solve a linear system to compute a vector of the Krylov basis. If the size of the problem is large, this step requires the use of an iterative linear solver, which is the inner iteration.

Namely, to build the Krylov subspace

$$K_m = \text{span}\{v_0, B^{-1}Av_0, (B^{-1}A)^2v_0, \dots, (B^{-1}A)^{m-1}v_0\}$$

for generalized eigenproblem

$$Au = \lambda Bu, \quad u \neq 0, \quad (1)$$

where both A and B are large sparse matrices, we compute an orthonormal basis of the Krylov subspace

$$K_m = \text{span}\{v_0, v_1, v_2, \dots, v_{m-1}\}.$$

For this, in each outer iteration, we have to solve the linear system

$$Bv_{k+1} = Av_k, \quad k = 0, 1, \dots, m-2, \quad (2)$$

to compute v_{k+1} from v_k .

In many cases, solving of the linear system (2) becomes the most frequent and therefore the most time-consuming part of an algorithm. Note that, as we compute in *finite precision*, error is unavoidable getting v_{k+1} by (2). In practice it becomes very important to find an answer to the following question: *what is the best strategy for stopping the inner iterations for ensuring the convergence of the outer iterations while minimizing the global computational cost?*

This question has been partially addressed by numerical experts since the eighties, in the context of Newton-like methods used as the outer solver [7]. It is generally concluded, as one could expect, that the accuracy of the inner iteration needs to be *increased* when the outer process comes closer to the solution. The proposed strategies for monitoring the inner iterations

have been so far very problem- and method-dependent. It was observed also, that inverse iterations, as Newton-like methods, require inner iterations to be more and more accurate while approaching the solution [12].

In the late nineties, the different behavior of embedded solvers involving a Krylov outer process was first noticed in [8] by Prof. Golub and a co-worker. Indeed for Lanczos or Arnoldi methods, the first Krylov vectors need to be computed with maximum accuracy, and this accuracy can then be *relaxed* as the convergence proceeds.

This seemingly counter-intuitive behavior, obtained by a *relaxation strategy* was thoroughly studied at CERFACS (Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique) in the Qualitative Computing group (see the habilitation manuscript of V. Frayssé [6] and the Ph. D. thesis of A. Bouras [2]). A first investigation was done with industrial support (CNES, [3, 4]). Another CERFACS Technical Report shows experimental evidence that this approach is promising in domain decomposition methods [5].

A workshop with CERFACS industrial partners of 3 days on the subject (see <http://www.cerfacs.fr/algor/iter2000.html>) was organized at CERFACS in September, 2000 with Prof. G. Golub as one of the keynote speakers.

It is easy to understand that relaxation strategy can significantly improve convergence of inner-outer algorithm – in terms of computation time costs. It is obvious that, stopped at a lower accuracy, a linear solver (preconditioned conjugate gradient for example) applied to (2) will require less inner iterations and therefore will be faster. It becomes especially important to avoid the inner iterations, which are not needed, for cases when matrices A and B in (1) are large, since these iterations are very time-consuming.

In this study we will describe details of implementation of the *relaxation strategy* into the `Code_Aster` code for solving the generalized eigenvalue problem (1) by implicitly restarted Arnoldi method with shifts, coupled with preconditioned conjugate gradient linear solver. In the next section we will introduce the reader to the problem proposed by EDF (Electricite de France) and will describe modifications of `Code_Aster`, which were done. Then, to demonstrate that the *relaxation strategy* is efficient, in Section 3 we will present some tests which were performed.

2 Implicitly restarted Arnoldi method with shifts for a generalized eigenvalue problem

2.1 Description of the problem proposed by EDF

We deal with a generalized linear $n \times n$ eigenvalue problem

$$Au = \lambda Bu, \quad u \neq 0, \quad (3)$$

here A and B are symmetric matrices with real coefficients. In (3) we are looking for the eigenpair (λ, u) , where λ is the eigenvalue and u is the corresponding eigenvector. It is often required to find more than one eigenpair.

Besides many other applications, this problem appears in mechanics and it was proposed by EDF (Electricite de France). To solve different mathematical models EDF has developed big software package called `Code_Aster`, which is written in Fortran77. In this study we are interested in implicitly restarted Arnoldi method with shifts (IRAMWS) for the generalized eigenvalue problem (3). In `Code_Aster` this algorithm is called by the operator `MODE_ITER_SIMULT` [11]. The mathematical problem and IRAMWS is described in details in [1]. For reference on IRAMWS itself also see [10].

We briefly remind the reader about the backward errors, associated with the problem (3)

$$\eta_1(\lambda, u) = \frac{\|(A - \lambda B)u\|_2}{(\|A\| + \lambda\|B\|)\|u\|_2}, \quad \eta_2(\lambda, u) = \frac{\|(B^{-1}A - \lambda)u\|_2}{\|B^{-1}A\|\|u\|_2},$$

here $\|\cdot\|$ denotes some matrix norm. The backward error η_1 is associated with the generalized eigenproblem (3) and η_2 with the corresponding standard problem $B^{-1}Au = \lambda u$ equivalent to (3). Of course η_2 is defined only for regular matrices B .

In `Code_Aster`, the quality of a computed eigenpair (λ, u) is measured by the quantity ('norme d'erreur')

$$\eta(\lambda, u) = \frac{\|(A - \lambda B)u\|_2}{\|Au\|_2}, \quad (4)$$

which is somewhat close to the backward error of the problem. For simplicity in the text we will still use the term 'backward error' to name $\eta(\lambda, u)$.

There are a few possible choices, which eigenpairs to compute. User defines his choice by controlling a parameter in `MODE_ITER_SIMULT` (see [11]). Possible options are:

- compute N eigenvalues of smallest magnitude (option 'PLUS_PETITE');
- compute N eigenvalues, closest to a given value (option 'CENTRE');
- compute all eigenvalues, which belong to a given interval (option 'BANDE').

According to the chosen option, the value of the shift σ is computed and the shifted matrix $A^\sigma = A - \sigma B$ is constructed. Then, (3) is replaced by the transformed eigenproblem (see [1])

$$(A - \sigma B)^{-1} B u = \frac{1}{\mu - \sigma} u. \quad (5)$$

2.2 Modification of Code_Aster – RLDLGG → GCPC

As it was described in Section 1, each step of IRAMWS requires the solution of linear system. By (5) and analogy to (1), (2), it follows that matrix of these linear systems is $A - \sigma B$.

In `Code_Aster` this part of IRAMWS, *i. e.* the solver of linear system with matrix $A - \sigma B$ is based on factorization $A - \sigma B = LDL^T$, here L denotes a lower triangular and D – a diagonal matrices. The solution by this direct method is provided by the the Fortran77 subroutine `RLDLGG`.

As the first step in implementing a relaxation strategy in IRAMWS we had to modify the `Code_Aster` code – replacing the factorization $A - \sigma B = LDL^T$ based linear solver by the *preconditioned conjugate gradient* iterative method. Employing an iterative linear solver instead of a direct one already optimizes the whole algorithm, as iterative methods are much more efficient for large problems. Moreover, it creates a possibility to vary a tolerance inside the linear solver, thus opening a door to the next step – relaxation strategy.

2.2.1 Technical details and level of preconditioning

We would like to discuss technical details of implementation of a different linear solver in IRAMWS. In `Code_Aster` this solver – the preconditioned conjugate gradient method is implemented in the Fortran77 subroutine `GCPC`. Schematically all necessary changes can be grouped into the four steps:

- avoid factorization of the shifted matrix $A - \sigma B = LDL^T$, where it is not necessary. For this, in the subroutine `VPFOPR` we replaced calling of `VPSTUR` by calling `VPSHIF` in 3 places;

- define preconditioner in `GCPC`. For this, in the subroutine `VPSORN` we had to create all the objects, needed for preconditioner, in memory first. Then, in the subroutine `OP0045`, define level of incomplete Cholesky preconditioner by setting integer value to the parameter `NIREMP` and define the preconditioner by calling the subroutine `PCLDLT`. Finally, in the subroutine `VPSORN`, we indicated the method of preconditioning (incomplete Cholesky) used inside `GCPC` by setting the parameter `'PREC = 'LDLT''`;
- in the subroutine `VPSORN`, which calls `GCPC`, define the input parameters of the subroutine `GCPC`: `'NITER = NBEQ / 2'` (here `NITER` is a maximum number of iterations allowed for `GCPC`, `NBEQ` is the size of matrices A and B), `'IREP = 0'` (to start iterating in `GCPC` from zero vector). One also needs to assign value to the parameter `EPSI` (which defines stopping criterion of iterations in `GCPC`). Possible ways to define `EPSI` is to set it to the constant (machine precision for example) or to define it by a relaxation strategy;
- everywhere in the subroutine `VPSORN` replace calling of `RLDLGG` by calling `GCPC`.

For a discussion how to tune the level of incomplete Cholesky preconditioning (the variable `NIREMP` in the subroutine `OP0045`) optimally see [9]. Increasing level of preconditioning can speed-up convergence in `GCPC` but also requires more computer time to build the preconditioner. The optimal value is computer platform- and problem-dependent and, according to [9], can be 0, 1 or 2.

Though we have experimented with different values of `NIREMP`, we have chosen `'NIREMP = 1'` as probably the best value. Computations were performed on Sun ultra sparc workstation.

2.2.2 Remark on stopping criterion in GCPC

To find a solution of the linear system

$$A^\sigma x = b$$

by the `Code_Aster` implementation of a preconditioned conjugate gradient method (subroutine `GCPC`) the following stopping criterion is applied: stop iterating if

$$\|r^{(k)}\|_2 < \|r^{(0)}\|_2 \text{EPSI}, \quad (6)$$

here $r^{(k)} = b - A^\sigma x^{(k)}$ is the k -th residual, computed in `GCPC`, and `EPSI` is the input parameter of the subroutine `GCPC`.

If $x^{(0)} = 0$, the stopping criterion (6) is equivalent to

$$\|r^{(k)}\|_2 < \|b\|_2 \text{EPSI}. \quad (7)$$

The subroutine `GCPC` has the input parameter `IREP` to control whether zero initial vector $x^{(0)} = 0$ should be used ('`IREP = 0`'). If the starting vector $x^{(0)}$ is supplied by the user, then '`IREP = 1`' should be assigned.

If '`IREP = 1`', then $r^{(0)}$, and therefore (6) criterion depend on the initial vector $x^{(0)}$. Hence (7) is a preferred stopping condition and it should be implemented in `GCPC`, instead of (6).

Since, in our experiments, we always started the inner iterations from zero '`IREP = 0`', stopping criterion of existing version of `GCPC` was defined correctly.

2.3 Implementation of a relaxation strategy

In each step of IRAMWS (outer iteration) a linear system is solved by the `Code_Aster` subroutine `GCPC` iteratively (inner iterations). Stopping of inner iterations inside the linear solver is controlled by the (6) criterion, which, because the starting vector is set to be zero, is equivalent to the criterion (7).

2.3.1 Control of when to stop the inner loop (subroutine `GCPC`)

To implement a relaxation strategy, we define the tolerance parameter `EPSI` (which is the input parameter of `GCPC`), according to [2, 3, 4, 6], by the formula

$$\text{EPSI}^j = \text{COEFEPSI} \frac{\varepsilon_d}{\max_{(\lambda^j, u^j)} \eta(\lambda^j, u^j)}. \quad (8)$$

Here j denotes the j -th outer step of IRAMWS, backward error η was defined by (4), *maximum* is taken over all computed approximations of eigenpairs (λ^j, u^j) , such that λ^j belongs to the required subset of spectrum (see explanation of options '`PLUS_PETITE`', '`CENTRE`' or '`BANDE`' in subsection 2.1). We also denote double machine precision by $\varepsilon_d \approx 2.22 \times 10^{-16}$ (ε_d is given by the `Code_Aster` constant `R8PREM()`). `COEFEPSI` is a fixed real constant, does not depend on j or any other quantities.

As it can be easily seen by the formula (8), when the convergence of IRAMWS proceeds, backward error η decreases – therefore increasing the tolerance `EPSI` of preconditioned conjugate gradient solver. In other words, the tolerance is *relaxed* with the approach of solution.

In addition to the formula (8) we applied the following restrictions: we required the `EPSI` ^{j} , as the function of j , to be a non-decreasing function; we

introduced the constant 'EPSIMAX = 1.0' to be a maximum allowed value of EPSI^j .

There is a question of further investigations how to choose optimally the value of the coefficient COEFEPSI in (8). The increase of COEFEPSI means more relaxation and vice versa. It should be noted that, if COEFEPSI is too large, already the first Krylov vectors can be computed with insufficient accuracy and this can cause failure of IRAMWS. On the other hand, if COEFEPSI is too small, almost no relaxation will be observed which means that IRAMWS is probably under-optimized.

To tune this parameter, different values of COEFEPSI were tried with the test case ahlv100h. After these experiments we set 'COEFEPSI = 100.0'.

To compare the relaxed and non-relaxed versions of the algorithm, each test was also performed with constant EPSI^j , set to be 'as small as possible', *i. e.* machine precision:

$$\text{EPSI}^j \equiv \varepsilon_d, \quad \forall j \quad (9)$$

2.3.2 Modification of Code_Aster – monitoring of backward error

In the previous implementation of IRAMWS in `Code_Aster`, the approximated eigenvectors u and the backward errors $\eta(\lambda, u)$, by the formula (4), are computed only once – at the end of algorithm, with the converged set of eigenvalues λ . The purpose of computing the final backward error is verification of the results.

However, in order to apply the relaxation strategy (8), the backward errors $\eta(\lambda^j, u^j)$ need to be computed in each step of IRAMWS – whenever the subroutine GCPC is called. For this reason, the set of approximated eigenvectors, corresponding to the computed approximations of eigenvalues, is required also, in each step of the eigensolver.

Therefore, in order to monitor the backward error $\eta(\lambda^j, u^j)$, to the set of `Code_Aster` subroutines we added the new subroutine BCKWER (file bckwer.f). The code of the subroutine BCKWER was written in analogy with that in the subroutine OP0045 (where the final backward errors are computed). While in OP0045 the final $\eta(\lambda, u)$ are obtained by calling the subroutine VPPARA, inside BCKWER we call VPPAR1 instead of VPPARA, for this purpose. The subroutine VPPAR1 (file vppar1.f) is also added by us. The only difference between VPPAR1 and existing `Code_Aster` subroutine VPPARA is that VPPAR1 does not call VPSTOR. Otherwise VPPAR1 is completely analogous to VPPARA.

Each time, before calling BCKWER, we need to compute the approximations of eigenvectors. For this, the subroutine DNEUPD is called, in each step of IRAMWS.

All the modifications, discussed in this and previous subsections (2.3.1 and 2.3.2), were implemented in the `Code_Aster` subroutine `VPSORN`.

2.4 Summary of changes implemented in `Code_Aster`

Inside the code all changes are underlined by the starting and ending indicators, for example

```
'C TM 17/10/01 B'
```

and

```
'C TM 17/10/01 E'.
```

Here 'C' tells the Fortran compiler that it should treat the above line of the code as a comment, 'TM' are initials of Tadas Meškauskas, '17/10/01' denotes the date when this code was added (or last time changed), and 'B' or 'E' signal about the beginning or ending of newly added code.

Next we briefly list all implemented changes.

In the subroutine `VPFOPR` (file `vpfopr.f`):

- '`CALL VPSTUR`' replaced by '`CALL VPSHIF`' in 3 places.

In the subroutine `OP0045` (file `op0045.f`):

- introduced new local variables '`INTEGER NIREMP, LAUX1`';
- '`CALL WKVECT(' &&OP0045.VECT1.AUX', 'V V R', NEQ, LAUX1)`';
- '`NIREMP = 1`', '`CALL PCLDLT(...)`', '`CALL VPSORN(...)`'.

In the subroutine `VPSORN` (file `vpsorn.f`):

- introduced new global parameters:
'`SUBROUTINE VPSORN (..., VAUX1, LRAIDE, MODES, TYPCON, NPIVOT, NBLAGR, OMEMIN, OMEMAX, OPTIOF)`'
such that '`REAL*8 VAUX1(NBEQ), OMEMIN, OMEMAX`',
'`CHARACTER*8 MODES`', '`CHARACTER*16 TYPCON, OPTIOF`',
'`INTEGER LRAIDE, NPIVOT, NBLAGR`' ;
- introduced many new local variables;
- assigned values to newly introduced local variables: new code starts by '`NITER = NBEQ / 2`' and ends by '`UPDATE = .TRUE.`';

- created a few new objects in memory: new code starts by
'MATAS = ZK24(ZI(LDYNFA+1))(1:19)' and ends by
'CRITER = '&&VPSORN_GCPC';
- inserted long piece of new code, implementing a relaxation strategy:
new code starts by 'IF (IDO .EQ. 1) THEN' and ends by
'CRITER = 'ENDIF';
- 'CALL RLDLGG' replaced by 'CALL GCPC' everywhere;
- inserted piece of new code, calculating final backward error and destroying some objects in the memory: new code starts by
'DO 61 J=1,NFREQ' and ends by 'CALL JEDETC('V',KSTOCPREC,1)'

Added new subroutine BCKWER (file bckwer.f), in each step of Arnoldi method computing the obtained backward error. The code inside the subroutine BCKWER is similar to that in the subroutine OP0045.

Added new subroutine VPPAR1 (file vppar1.f), which is called by BCKWER. The only difference between VPPAR1 and existing Code_Aster subroutine VPPARA is that VPPAR1 does not call VPSTOR. Otherwise VPPAR1 is completely analogous to VPPARA.

All changes were implemented on the basis of Code_Aster version 6 code, but it was checked that everything works with the previous version 5.5 too.

3 Numerical results

Results of numerical experiments are provided in the following figures (see Fig. 1, Fig. 2, Fig. 3, Fig. 4 and Fig. 5 further). For each test-case, there are three subfigures a), b) and c). They represent:

- a) the GCPC tolerance parameter EPSI versus the number of IRAMWS iteration. The solid line represents EPSI computed by the (8) relaxation strategy, while the dashed line represents the case (9), with constant EPSI, set to the machine precision;
- b) convergence history – the maximum backward error $\max_{(\lambda^j, u^j)} \eta(\lambda^j, u^j)$ versus the number of IRAMWS iteration j . Maximum is taken over all computed approximations of eigenpairs (λ^j, u^j) , such that λ^j belongs to the required subset of spectrum (see explanation of options

'PLUS_PETITE', 'CENTRE' or 'BANDE' in subsection 2.1). The solid line represents values of maximum backward error obtained by the relaxation strategy (8), while the dashed line represents the case (9), without relaxation;

- c) the final backward errors of converged eigenpairs obtained by the relaxation strategy (8) (black bars), and the ones obtained by (9), without relaxation (white bars).

Also, in Table1 we compare the CPU times, needed by different implementations of IRAMWS (procedure `MODE_ITER_SIMULT`) in `Code_Aster`. All computations were performed on Sun ultra sparc workstation with 128 megabytes of RAM. In Table 1 “Original” stands for original (official) implementation of IRAMWS, in which the obtained linear systems are solved by direct method, based on factorization $A - \sigma B = LDL^T$ (see subsection 2.2). Other two columns in Table 1 represent new implementation of IRAMWS, in which we apply iterative preconditioned conjugate gradient method as a linear solver, instead of direct linear solver. One of these two columns provides CPU time needed by the strategy without relaxation, while another – the CPU time needed by the relaxation strategy.

Test-case	Original	Without relaxation	With relaxation
ahlv100a	0.59 s	9.40 s	7.31 s
ahlv100h	0.47 s	3.45 s	3.16 s
sdls04b	1.47 s	22.24 s	20.11 s
sdll11b	238 s	50444 s	44258 s
tpam3	423 s	29867 s	24748 s

Table 1: CPU time (in seconds) of execution `MODE_ITER_SIMULT`.

The total CPU times measuring execution of the whole tests (the procedure `MODE_ITER_SIMULT` is only a part of the test) are provided in Table 2.

Test-case	Original	Without relaxation	With relaxation
ahlv100a	224 s	238 s	237 s
ahlv100h	206 s	215 s	208 s
sdls04b	212 s	230 s	226 s
sdll11b	1625 s	55882 s	49535 s
tpam3	1185 s	36844 s	31657 s

Table 2: Total CPU time (in seconds).

Note that it was not our goal to implement *efficiently* the preconditioned conjugate gradient linear solver, instead of factorization based one. Therefore, the CPU times obtained using iterative linear solver (preconditioned conjugate gradient method) are much worse than the CPU times obtained by original implementation of IRAMWS, which uses direct linear solver. For large problems iterative linear method always outperforms direct method, if properly implemented. The choose of optimal level of preconditioning (see subsection 2.2.1) can also be a very important issue here. In this report we only emphasize the advantage of *a relaxation strategy*, as a possible improvement of nonrelaxed Krylov-type algorithms containing in them inner-outer iterations.

Acknowledgments

The authors thank O. Boiteau from EDF (Electricite de France) for his time and help with `Code_Aster` software as well as J. P. Gregoire from EDF and L. Giraud from CERFACS (Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique) for helpful discussions. We also wish to acknowledge J. L. Vautescal from EDF for the support granted for this work.

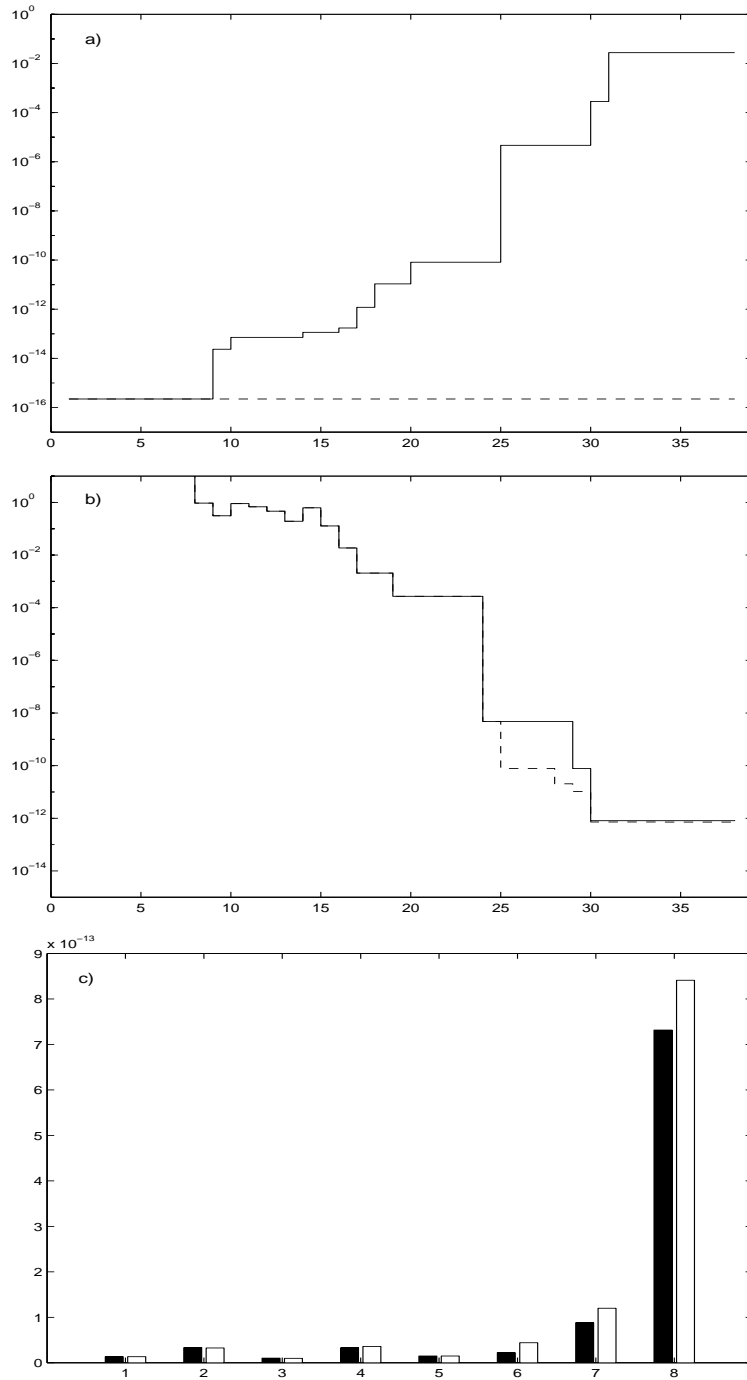


Fig. 1: test-case ahlv100a, $n = 472$, option 'BANDE' - $\lambda \in [1, 1000]$.

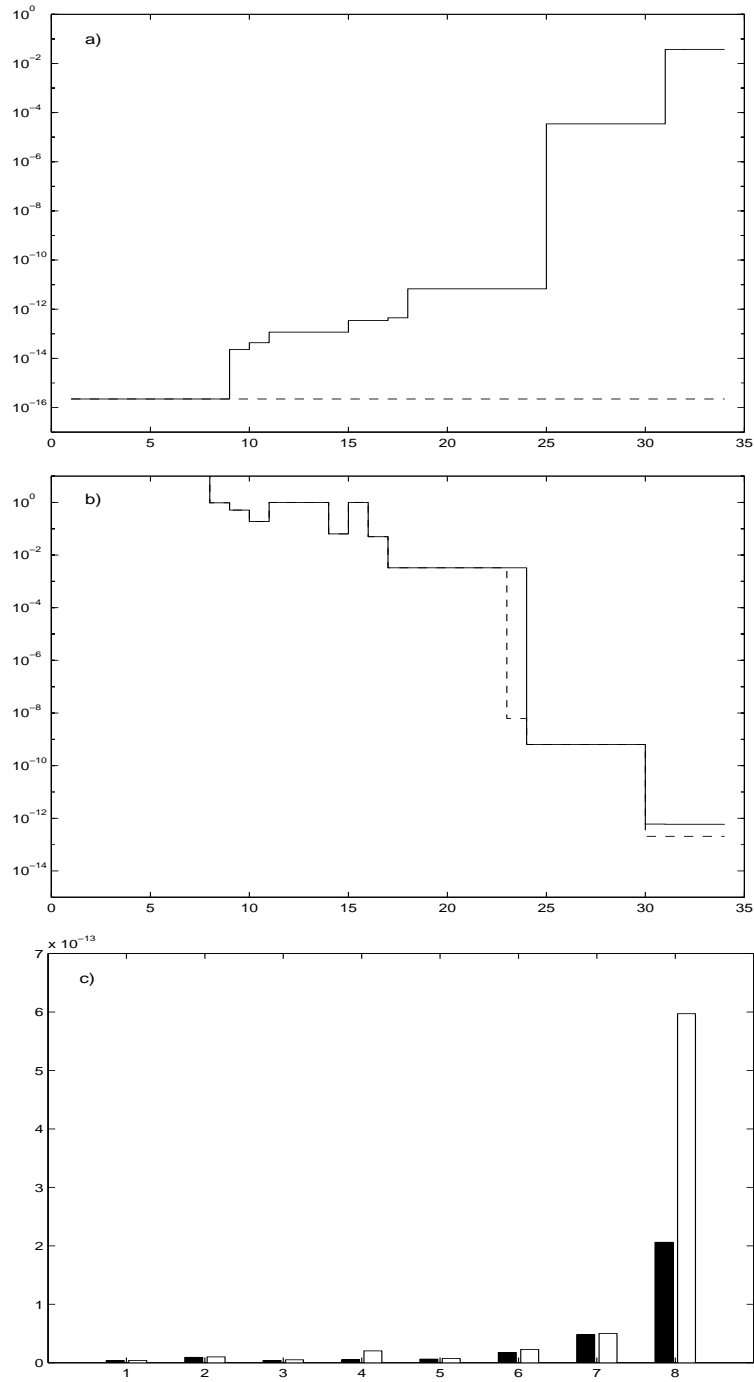


Fig. 2: test-case ahlv100h, $n = 456$, option 'BANDE' - $\lambda \in [1, 1000]$.

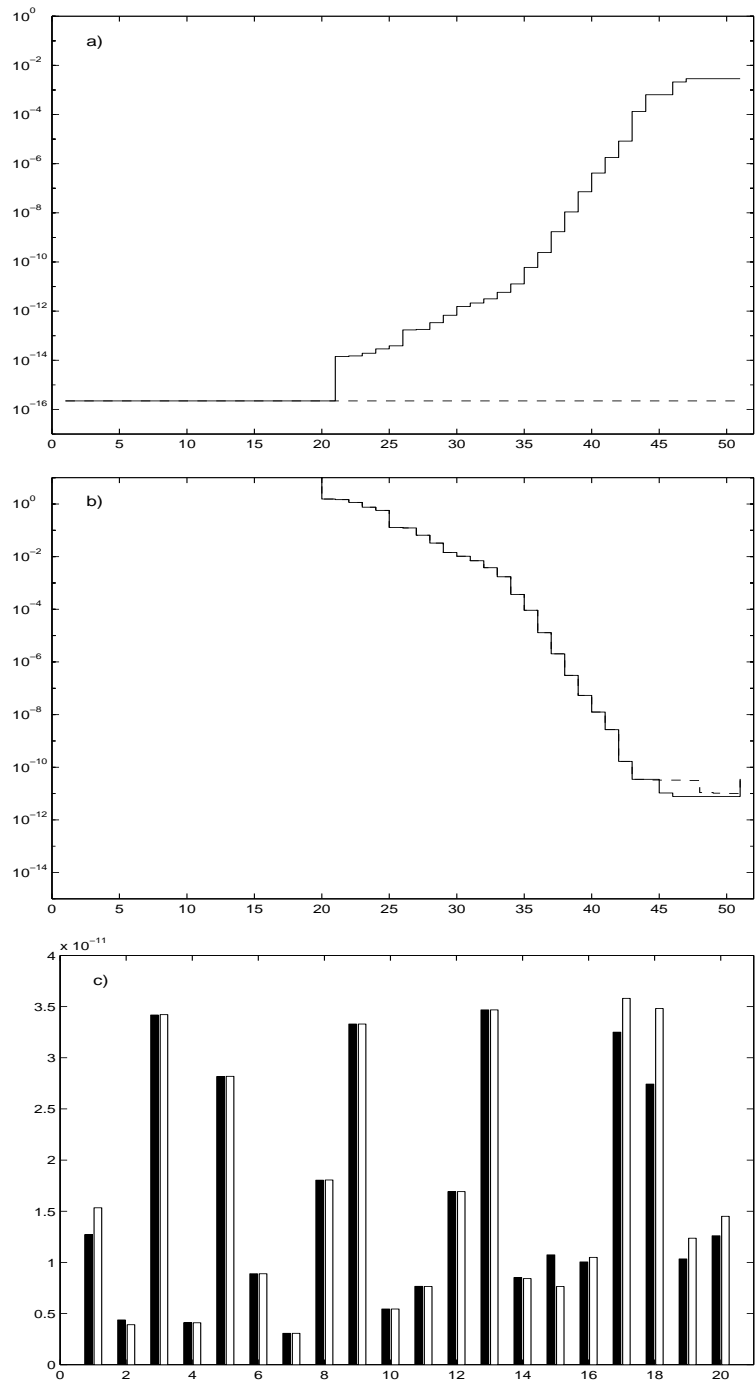


Fig. 3: test-case sds04b, $n = 828$, option 'PLUS_PETITE'.

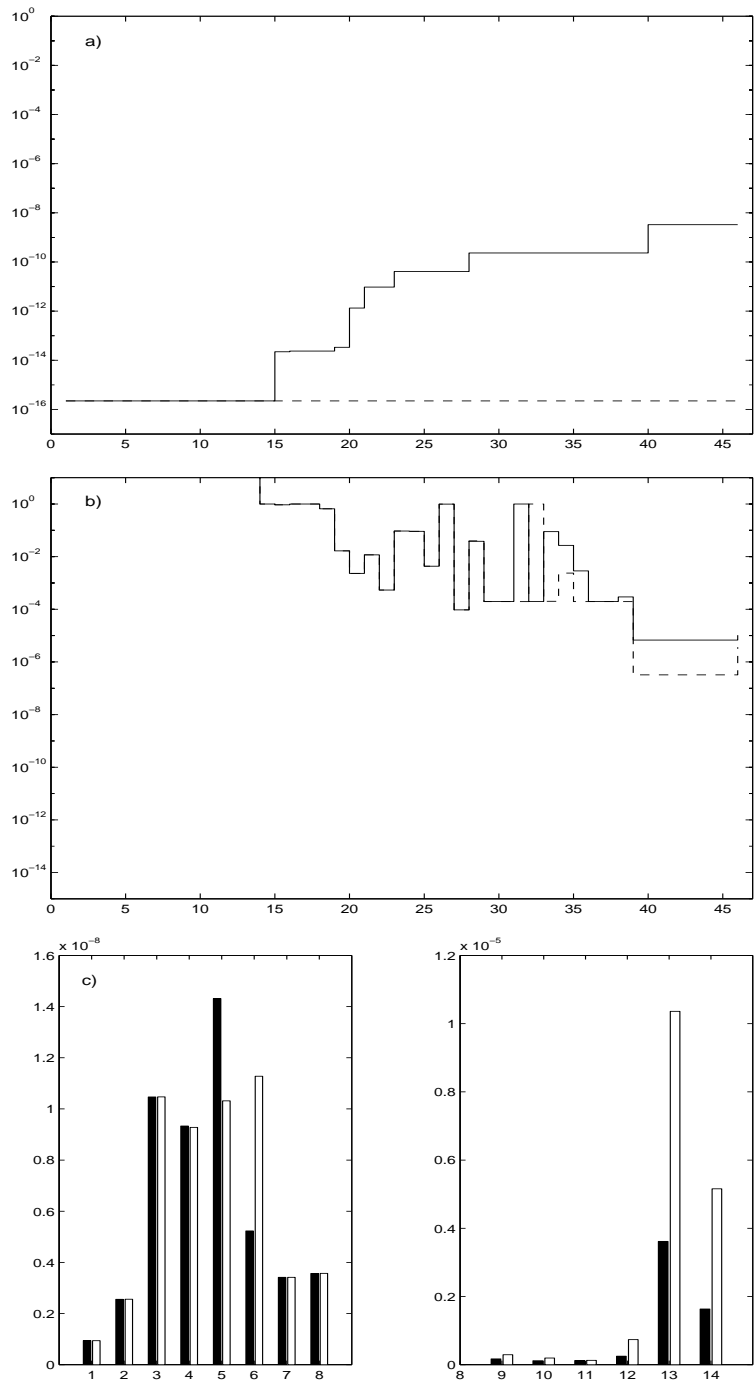


Fig. 4: test-case sdll11b, $n = 38400$, option 'BANDE'.

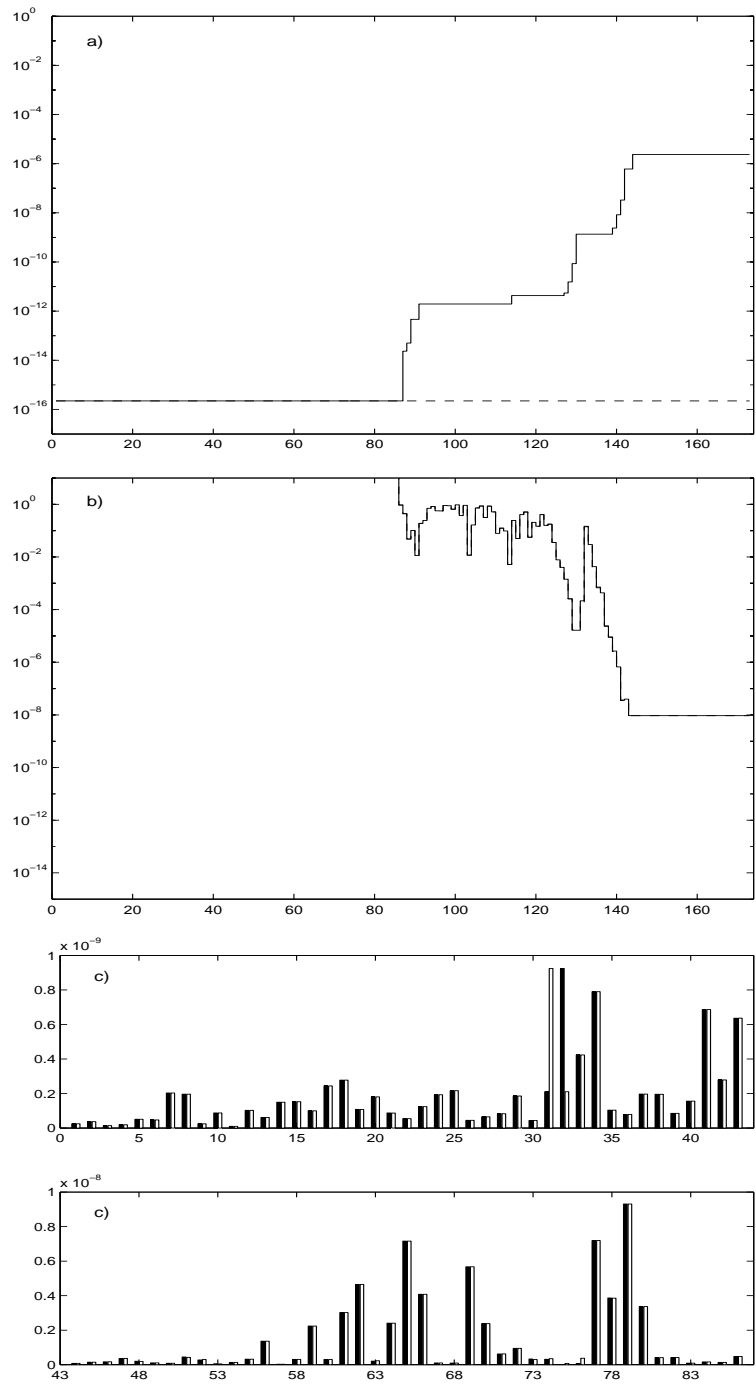


Fig. 5: test-case tpam3, $n = 11880$, option 'BANDE'.

Bibliography

- [1] O. Boiteau, Algorithme de résolution pour le problème généralisé, *Code_Aster, Manuel de Référence*, EDF/MTI/MMN, R5.01.01, 2001.
- [2] A. Bouras, Contrôle de convergence des solveurs emboîtés pour le calcul de valeurs propres avec inversion, *Ph. D. dissertation*, CERFACS, 2000.
- [3] A. Bouras and V. Frayssé, A relaxation strategy for inexact matrix-vector products for Krylov methods, *CERFACS Technical Report TR/PA/00/15*, 2000. Accepted for publication in SIAM Journal in Matrix Analysis and Applications.
- [4] A. Bouras and V. Frayssé, A relaxation strategy for the Arnoldi method in eigenproblems, *CERFACS Technical Report TR/PA/00/16*, 2000.
- [5] A. Bouras, V. Frayssé and L. Giraud, A relaxation strategy for inner-outer linear solvers in domain decomposition methods, *CERFACS Technical Report TR/PA/00/17*, 2000. Submitted to BIT.
- [6] V. Frayssé, The power of backward error analysis, *Habilitation à Diriger des Recherches*, CERFACS, 2000.
- [7] E. Giladi, G. H. Golub and J. B. Keller, Inner and outer iterations for the Chebyshev algorithm, *SIAM J. Numer. Anal.*, 35(1):300–319, 1998.
- [8] G. H. Golub and Q. Ye, Inexact preconditioned conjugate gradient method with inner-outer iteration, Sccm-97-04, Stanford University, 1997.
- [9] J. P. Gregoire, Accélération de la convergence du gradient conjugué préconditionné en utilisant la factorisation incomplète par niveau, EDF, HI-76/00/005/A, 2000.
- [10] R. B. Lehoucq, D. C. Sorensen and C. Yang, ARPACK users guide: Solution of large scale eigenvalue problems with implicitly restarted Arnoldi methods, SIAM, 1998.

- [11] B. Quinnez, Opérateur MODE_ITER_SIMULT, *Code_Aster, Manuel de Référence*, EDF/MTI/MMN, U4.52.03-E, 2000.
- [12] P. Smit and M. H. C. Paardekooper, The effects of inexact solvers in algorithms for symmetric eigenvalue problems, *Lin. Alg. Appl.*, **287**, pp. 337–357, 1999.