

## **A Jacobi-Davidson type method with a correction equation tailored for integral operators**

**Paulo B. Vasconcelos · Filomena D. d'Almeida ·  
Jose E. Roman**

Received: date / Accepted: date

**Abstract** We propose two iterative numerical methods for eigenvalue computations on large dimensional problems issued from finite approximations of integral operators, and describe their parallel implementation. A matrix representation of the problem on a moderate dimensional space, defined from an infinite dimensional one, is computed along with its eigenpairs. These are taken as initial approximations and iteratively refined, by means of a correction equation based on the reduced resolvent operator and performed on the moderate size space, to enhance their quality. Each refinement step requires the prolongation of the solution of the correction equation back to a higher dimensional space, defined from the infinite dimensional one. This approach is particularly adapted for the computation of eigenpair approximations of integral operators, where prolongation and restriction matrices can be easily built making a bridge between coarser and finer discretizations. We propose two methods that apply a Jacobi-Davidson like correction: Multipower Defect-Correction (MPDC), which uses a single-vector scheme, if the eigenvalues to refine are simple, and Rayleigh-Ritz Defect-Correction (RRDC), which is based on a projection onto an expanding subspace. Their main advantage lies in the fact that the correction equation is performed on a smaller space while for general solvers it is done on the higher dimensional one. We discuss implementation and parallelization details, using the PETSc and SLEPc packages. Also, numerical results on an astrophysics application, whose mathematical model involves a weakly singular integral operator, are presented.

---

This work was partially supported by European Regional Development Fund through COMPETE, FCT - Fundação para a Ciência e a Tecnologia through CMUP - Centro de Matemática da Universidade do Porto and Spanish Ministerio de Ciencia e Innovación under projects TIN2009-07519 and AIC10-D-000600.

Paulo B. Vasconcelos  
Faculdade de Economia da Universidade do Porto, rua Dr. Roberto Frias s/n, 4200-464 Porto, Portugal  
E-mail: pjv@fep.up.pt

Filomena D. d'Almeida  
Faculdade de Engenharia da Universidade do Porto, rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal  
E-mail: falmeida@fe.up.pt

Jose E. Roman  
D. Sistemes Informàtics i Computació, Universitat Politècnica de València, Camí de Vera s/n, 46022 València, Spain  
E-mail: jroman@dsic.upv.es

**Keywords** Eigenvalue problems · integral operators · Krylov subspace methods · Jacobi-Davidson approximation · parallel computing

## 1 Introduction

The computation of eigenpairs of a compact Fredholm integral operator can be achieved by discretizing it on a subspace, yet accurate solutions may require projections on a large dimensional space. To reduce the computational costs, the projection can be performed on a subspace of moderate dimension and the eigenlements of its representing matrix can be computed using state-of-the-art methods. These can then be iteratively refined, by a defect correction type procedure, still of moderate size, to yield a better approximation to the spectral elements of the operator. This defect correction procedure is similar to the expansion phase in Jacobi-Davidson type methods, except that the former requires the solution of a moderate size problem instead of the large dimensional one on the latter. This can be accomplished by introducing information from the left eigenvectors of the moderate size problem and from the finer discretization through matrix-vector operations only.

Let us consider the computation of the eigenpairs of the problem

$$T\varphi = \theta\varphi, \quad \varphi \neq 0, \quad (1)$$

where the integral operator  $T : \mathcal{X} \rightarrow \mathcal{X}$  is defined on a Banach space with kernel  $g$ ,

$$(T\varphi)(\tau) = \int_0^{\tau^*} g(|\tau - \tau'|)\varphi(\tau')d\tau', \quad \tau \in [0, \tau^*]. \quad (2)$$

We are interested in the case where  $\theta$  is a non-zero simple isolated eigenvalue. A generalization for the computation of a cluster of eigenvalues can be found in [4].

Discretization mechanisms, for instance (Kantorovich) projection onto a finite dimensional subspace, are usually used to solve the integral problem. Operator  $T$  is approximated by  $T_n$ ,  $T_n x = \pi_n T x$ ,  $x \in \mathcal{X}$ , its projection onto a finite dimensional subspace  $\mathcal{X}_n$  of  $\mathcal{X}$  spanned by an ordered basis  $e_n = [e_{n,1}, \dots, e_{n,n}]$ . The projection is characterized by

$$\pi_n x = \sum_{j=1}^n \langle x, e_{n,j}^* \rangle e_{n,j}, \quad (3)$$

where  $e_n^* = [e_{n,1}^*, \dots, e_{n,n}^*]$ ,  $e_{n,j}^* \in \mathcal{X}^*$ ,  $j = 1, \dots, n$ , is the adjoint basis of  $e_n$  ( $\mathcal{X}^*$  is the adjoint space of  $\mathcal{X}$ ).

The spectral problem

$$T_n \varphi_n = \theta_n \varphi_n, \quad (4)$$

where  $\varphi_n \in \mathcal{X}$  and the eigenvalue  $\theta_n$  is an approximation to  $\theta$ , is to be solved. The solution  $\varphi_n$  is normalized so that  $\langle \varphi_n, \psi_n \rangle = 1$ , where  $\psi_n \in \mathcal{X}^*$  is chosen as an eigenfunction of  $T_n^*$  corresponding to  $\bar{\theta}_n$ ,

$$T_n^* \psi_n = \bar{\theta}_n \psi_n. \quad (5)$$

For  $n$  large enough there exists a  $\theta_n$  of  $T_n$  in a given neighborhood of  $\theta$  such that  $|\theta_n - \theta| \leq c\|(T_n - T)\|$  (convergence in norm) or, for  $\theta \neq 0$ ,  $|\theta_n - \theta| \leq c\|(T_n - T)T\|$  ( $v$ -convergence),  $c$  constant independent of  $n$ . These convergence results can be found in [2, ch. 2]. It is worth to note that this approach can be extended to deal with multiple eigenvalues or finite clusters of eigenvalues.

The above formulation, for a finite and fixed  $n$ , leads to matrix eigenvalue problems of dimension  $n$

$$A_n u_n = \theta_n u_n \quad \text{and} \quad A_n^* v_n = \bar{\theta}_n v_n, \quad (6)$$

where  $A_n = \langle T e_n, e_n^* \rangle$  represents the restriction of  $T_n$  to  $\mathcal{X}_n$ , and  $\varphi_n = e_n u_n$ , with  $u_n = \frac{1}{\theta_n} \langle \varphi_n, e_n^* \rangle$  and  $v_n = \langle e_n, \psi_n \rangle$  (further details can be seen in [4]). These spectral approximations will be used as initial approximations for the refinement process.

The rest of the paper is organized as follows. Section 2 introduces the Multipower Defect-Correction (MPDC) and the Rayleigh-Ritz Defect-Correction (RRDC) methods. Implementation details for both algorithms are given in section 3. The implementation is carried out by using PETSc and SLEPc, as explained in subsection 3.1. The computation of initial approximations, a prerequisite for both methods, is done with SLEPc as discussed in subsection 3.2. An important step of the methods is the correction equation, in the present case a resolution of a linear system of equations. This is treated in detail in subsection 3.3. Also, the issue of matrix permutations for reduction of parallel overhead is discussed in subsection 3.4. Section 4 shows the performance of the methods in solving an astrophysics problem, in terms of parallel efficiency. Also, the influence of some parameters of the algorithms are studied. Finally, some conclusions are given in section 5.

## 2 The MPDC and RRDC methods

For a required precision we may need a large dimensional matrix, say  $A_m$ , or, alternatively, as mentioned in the previous section, we can use an iterative refinement method on the solution of an eigenvalue problem of dimension  $n \ll m$ , achieving the same precision as the one obtained with  $A_m$ . The methods proposed next share such a refinement procedure, aiming at maintaining their computational costs moderate, differing only on the accelerating mechanisms used.

Let  $F : \mathcal{X} \rightarrow \mathcal{X}$  be defined by

$$F(\varphi) = T\varphi - \langle T\varphi, \psi_n \rangle \varphi, \quad \varphi \in \mathcal{X}, \quad (7)$$

where  $\psi_n$  is the solution of (5). As for Davidson-type methods, the eigenvalue problem is tackled as a Newton-type method by approximately solving  $F(\varphi) = 0$ . It is worth mentioning that being  $\varphi$  an eigenvector of  $T$  associated to the eigenvalue  $\theta \neq 0$ , normalized by  $\langle \varphi, \psi_n \rangle = 1$ , then  $\langle T\varphi, \psi_n \rangle = \theta \langle \varphi, \psi_n \rangle = \theta$ . The zeros of (7) are approximated by a fixed slope Newton method starting with  $\varphi_n^{(0)} = \varphi_n$ ,  $\varphi_n^{(k+1)} = \varphi_n^{(k)} - (D_0 F)^{-1} F(\varphi_n^{(k)})$ ,  $k = 0, 1, \dots$ , where  $(D_0 F)\eta = T\eta - \langle T\eta, \psi_n \rangle \varphi_n - \langle T\varphi_n, \psi_n \rangle \eta$ ,  $\eta \in \mathcal{X}$ , is the approximation of the Fréchet derivative of  $F$  at  $\varphi_n$ . Furthermore, a perturbed fixed slope Newton method is obtained by replacing  $T$  by  $T_n$ ,  $(\tilde{D}_0 F)\eta = T_n \eta - P_n(T_n \eta) - \theta_n \eta$ , where  $P_n$  is the spectral projection of  $T_n$  associated to  $\theta_n$

$$P_n \eta = \langle \eta, \psi_n \rangle \varphi_n, \quad \eta \in \mathcal{X}. \quad (8)$$

Since  $T_n$  and  $P_n$  commute,  $(\tilde{D}_0 F)\eta = (T_n - \theta_n I)\eta$ , imposing  $P_n \eta = 0$ . The perturbed fixed slope Newton method gives rise to

$$\varphi_n^{(k+1)} = \varphi_n^{(k)} - S_n \xi, \quad \xi = F(\varphi_n^{(k)}) \quad (9)$$

and the computation of  $\eta = S_n \xi$  amounts to solve the following system (correction equation)

$$\begin{cases} (T_n - \theta_n I)\eta = (I - P_n)\xi \\ P_n \eta = 0 \end{cases}. \quad (10)$$

The operator  $S_n$  is indeed the reduced resolvent of  $T_n$  associated with  $\theta_n$ , that is, the inverse of the restriction of  $T_n - \theta_n I$  to the range of  $I - P_n$ .  $S_n$  is defined in [7, p. 107] as  $\lim_{z \rightarrow \theta_n} (T_n - zI)^{-1} (I - P_n)$  and it verifies  $(T_n - zI)^{-1} S_n = (I - P_n)$ . Moreover, since we do not have access to operator  $T$ , it is replaced with  $T_m$  corresponding to a much finer grid. Iteration (9) corresponds also to a defect correction method applied to (7) using  $S_n + \varphi_n$  as approximate inverse (see [4]).

Under these assumptions the functions involved in (10) may be represented by vectors of their  $m$  components in terms of the basis  $e_m$  of the functional space  $\mathcal{X}_m$ . When these functions are restricted to  $\mathcal{X}_n$  we will use vectors of their  $n$  components in terms of the basis  $e_n$ , in this case the names of the vectors will have the subscript  $n$ . The operators  $T_m$  restricted to  $\mathcal{X}_m$  and  $T_n$  restricted to  $\mathcal{X}_n$  will be represented by  $A_m = \langle T e_m, e_m^* \rangle$  and  $A_n$ . We will denote by  $R = \langle e_m, e_n^* \rangle$  and  $E = \langle e_n, e_m^* \rangle$ , respectively, the restriction and extension matrices required to map the representation of functions from an  $m$ -dimensional space to an  $n$ -dimensional one, and vice-versa.

The MPDC method (Algorithm 1) interlaces the defect-correction stage (10) with a certain number of power iterations as a cheap single-vector mechanism, whereas the acceleration of the RRDC method (Algorithm 2) relies on a Rayleigh-Ritz procedure applied to a search subspace of increasing dimension.

As an initialization to both algorithms, we have to solve a moderate size eigenproblem (6) to get  $\theta_n$ ,  $u_n$  and the left eigenvector  $v_n$ . The eigenvectors are normalized so that  $v_n^* u_n = 1$ , and in the  $m$ -dimensional space, the approximate eigenvectors must be normalized in the same way. The algorithms compute increasingly good approximations of the right eigenvector  $u_m^{(k)}$  but not for the left eigenvector, which is fixed and computed as the prolongation of  $v_n$  to the larger space:  $\hat{v}_m = A_m^* R^* v_n$ .

---

**Algorithm 1:** Multipower Defect-Correction method

normalize  $u_n$  and  $v_n$  so that  $v_n^* u_n = 1$

$u_m^{(0)} = E u_n$  and normalize so that  $\|u_m^{(0)}\| = 1$

$\hat{v}_m = A_m^* R^* v_n$  and normalize so that  $\hat{v}_m^* u_m^{(0)} = 1$

for  $k = 0, 1, 2, \dots$

$y_m^{(0)} = u_m^{(k)}$

[Multipower stage:]

for  $j = 1, 2, \dots, \ell$

$$\mu^{(j)} = \hat{v}_m^* A_m y_m^{(j-1)}, \quad y_m^{(j)} = \frac{1}{\mu^{(j)}} A_m y_m^{(j-1)}, \quad \text{with } \mu^{(j)} \neq 0$$

[Defect-Correction stage:]

compute the residual  $r_m = A_m y_m^{(\ell)} - \mu^{(\ell)} y_m^{(\ell)}$

compute the correction  $t_m = S_n r_m$ , according to Eqs. (11)–(12)

update next iterate  $u_m^{(k+1)} = y_m^{(\ell)} - t_m$  and normalize so that  $\hat{v}_m^* u_m^{(k+1)} = 1$

---

Algorithm 1 performs  $\ell$  number of power iterations before the defect correction phase to improve the accuracy of the approximation while Algorithm 2 performs an orthogonal Rayleigh-Ritz projection of  $A_m$  onto the subspace whose orthonormal basis is expanded with the corrections obtained from the restricted space. Note that, for efficiency reasons, matrix  $G$  can be computed incrementally, that is, one row and column in each iteration.

**Algorithm 2:** Rayleigh-Ritz Defect-Correction method

normalize  $u_n$  and  $v_n$  so that  $v_n^* u_n = 1$

initialize the basis  $Q = [u]$  with  $u = (Eu_n)/\|Eu_n\|_2$ , with  $\|Eu_n\|_2 \neq 0$

$\hat{v}_m = A_m^* R^* v_n$  and normalize so that  $\hat{v}_m^* u = 1$

for  $k = 0, 1, 2, \dots$

[Extraction stage:]

compute the projected matrix  $G = Q^T A_m Q$

compute the most wanted eigenvalue  $\mu_i$  and eigenvector  $z_i$  of  $G$

compute the Ritz vector  $u_m = Qz_i$  and normalize it so that  $\hat{v}_m^* u_m = 1$

[Defect-Correction stage:]

compute the residual  $r_m = A_m u_m - \mu_i u_m$

compute the correction  $t_m = S_n r_m$ , according to Eqs. (11)–(12)

orthonormalize  $t_m$  against the basis  $Q$  and expand the search subspace  $Q = [Q, t_m]$

Given  $g = A_m Q_{:,k}$ , where  $Q_{:,k}$  denotes the last column of the expanding basis  $Q$ , the new column to be appended to  $G$  can be computed as  $Q^T g$ , and similarly for the elements of the new row.

In both algorithms, the matrix version of the computation of  $t_m = S_n r_m \in \mathbb{R}^m$  for a given residual  $r_m \in \mathbb{R}^m$ , corresponding to (10), is done by solving the following system of linear equations of dimension  $(n+1) \times n$  to get  $t_n$

$$\begin{cases} (A_n - \theta_n I)t_n = Rr_m - v_n^*(Rr_m)u_n \\ v_n^* t_n = 0 \end{cases}. \quad (11)$$

To extend  $t_n$  to  $\mathbb{R}^m$  one must obtain a projection for  $\eta$  (see equation (10)) onto  $\mathcal{X}_m$ . Plugging the second equation onto the first it results  $T_n \eta - \theta_n \eta = (I - P_n)\xi$ . From the definitions of  $T_n$  (3) and  $P_n$  (8),

$$\sum_{j=1}^n \langle T\eta, e_{n,j}^* \rangle e_{n,j} - \theta_n \eta = \xi - \langle \xi, \psi_n \rangle \varphi_n.$$

Applying  $\langle \cdot, e_{m,i}^* \rangle$ ,  $i = 1, \dots, m$ , it results

$$\begin{aligned} \sum_{j=1}^n \langle T e_{n,j}, e_{m,i}^* \rangle \langle \eta, e_{n,j}^* \rangle - \theta_n \langle \eta, e_{m,i}^* \rangle &= \langle \xi, e_{m,i}^* \rangle - \langle \xi, \psi_n \rangle \langle \varphi_n, e_{m,i}^* \rangle \\ &= \langle \xi, e_{m,i}^* \rangle - \langle \xi, \frac{1}{\theta_n} T_n^* \psi_n \rangle \langle \varphi_n, e_{m,i}^* \rangle \text{ from (5)} \\ &= \langle \xi, e_{m,i}^* \rangle - \frac{1}{\theta_n} \langle T_n \xi, \psi_n \rangle \langle \varphi_n, e_{m,i}^* \rangle \\ &= \langle \xi, e_{m,i}^* \rangle - \frac{1}{\theta_n} \sum_{j=1}^n \langle e_{n,j}, \psi_n \rangle \langle T \xi, e_{n,j}^* \rangle \langle \varphi_n, e_{m,i}^* \rangle. \end{aligned}$$

Using matrix notation we obtain

$$EA_n t_n - \theta_n t_m = r_m - \frac{1}{\theta_n} v_n^* (A_n r_n) u_m,$$

bearing in mind that  $\langle \eta, e_n^* \rangle = t_n$ ,  $\langle \eta, e_m^* \rangle = t_m$ ,  $\langle \varphi_n, e_m^* \rangle = u_m$ ,  $\langle \xi, e_n^* \rangle = r_n$  and  $\langle \xi, e_m^* \rangle = r_m$ . That is

$$t_m := \frac{1}{\theta_n} \left( EA_n t_n + \frac{1}{\theta_n} v_n^* (A_n r_n) u_m - r_m \right). \quad (12)$$

Thus, the expansion vector  $t_m$  is computed through the resolution of a linear system of equations of moderate size  $n$  (11) to obtain  $t_n$ , followed by a matrix-vector product and a prolongation, according to (12). Algorithm 2 is very similar to the standard Jacobi-Davidson method [15], see Algorithm 3. The main difference is the computation of the correction equation (penultimate line in both algorithms). In both cases, an iterative linear solver is applied, the generic Jacobi-Davidson operating on a large matrix ( $A_m$ ) whereas our approach computes the correction more cheaply (on  $A_n$ ). The correspondence of the two algorithms makes it possible to incorporate advanced features in RRDC as well, such as restarting (to keep the size of the search subspace limited), locking (to search for further eigenpairs once the first one is converged) and harmonic extraction (to enhance approximations in the case of interior eigenvalues).

---

**Algorithm 3:** Jacobi-Davidson method

initialize the basis  $Q = [u]$  with  $\|u\|_2 = 1$

for  $k = 0, 1, 2, \dots$

[Extraction stage:]

compute the projected matrix  $G = Q^T A_m Q$

compute the most wanted eigenvalue  $\mu_i$  and eigenvector  $z_i$  of  $G$

compute the Ritz vector  $u_m = Q z_i$

[Subspace expansion stage:]

compute the residual  $r_m = A_m u_m - \mu_i u_m$

solve approximately  $(I - u_m u_m^*)(A_m - \mu_i I)(I - u_m u_m^*) t_m = -r_m, \quad t_m \perp u_m$

orthonormalize  $t_m$  against the basis  $Q$  and expand the search subspace  $Q = [Q, t_m]$

---

It is also possible to relate Algorithm 1 with standard methods for eigenvalue problems such as the Rayleigh quotient iteration (RQI). In [14] it is shown that inexact RQI is equivalent, under some conditions, to the Newton-Grassmann method. This latter method can be seen as a single-vector version of Jacobi-Davidson, that is, the current eigenvector approximation  $u_m^{(k)}$  is replaced by  $u_m^{(k)} + t_m$ , where  $t_m$  is computed with the same correction equation as in Algorithm 3. This approach avoids the problem of iteratively solving a nearly singular linear system in RQI. Algorithm 1 does this as well, but removing the singularity in the smaller space. Despite the viability of extending Algorithm 1 for computing invariant subspaces, in a similar way as has been done for RQI [1], this is beyond the scope of this paper.

As for the convergence of these methods, we prove (in [8]) the following theorem in the context of functional analysis, for multiple or clustered eigenvalues of  $T$ . This theorem guarantees the convergence of an operator counterpart of Algorithm 1.

**Theorem 1** *Let  $(T_n)_n$  be a sequence of compact integral operators  $v$ -convergent to  $T$  ( $v$ -convergence:  $\|T_n\|$  is bounded,  $\|(T_n - T)T\| \rightarrow 0$  and  $\|(T_n - T)T_n\| \rightarrow 0$ ), as in the case of Kantorovich approximations. For a fixed integer  $\ell$  (the number of power iterations), for all  $k$  (the number of iterative refinement steps) and for  $n$  large enough, there exist constants  $\alpha$  and  $\beta$  independent of  $n$ , such that,*

$$\|\phi_n^{(k)} - \phi\| \leq \alpha \|(T_n - T)T^\ell\| (\beta \|(T_n - T)T^\ell\|)^k. \quad (13)$$

This result is an extension of the one stated in [2, p. 144] where only a single power iteration is used. The proof in [8] is, however, more difficult and not straightforward, due to

the interlacing of the  $\ell$  power iterations with the defect correction iterative process. As part of the proof we show that  $|\mu^{(j)}| \geq |\theta/2|$  for  $j = 1, \dots, \ell$ , which guarantees that the method has no breakdown.

For Algorithm 2, RRDC method, a proof is not yet developed, but a defect correction outer iteration is still used, completed with a richer search subspace (instead of a single vector, compared with MPDC). Hence we can expect that its proof would follow the previous ones. The RRDC method exploits the complete subspace constructed so far, while MPDC only takes a simple linear combination of the last eigenvector approximation  $y_m^{(\ell)}$  and the computed correction  $t_m$ . The computed eigenvalue approximations are different in both methods, so the correction  $t_m$  in RRDC is not the same as in MPDC. However, we expect a convergence behaviour similar to Jacobi-Davidson, since the Rayleigh-Ritz procedure is able to extract the best eigenvector approximation available in the current subspace, which is not worse than in the case of MPDC.

### 3 Implementation Details

In this section, we provide details on how the defect correction algorithms described in section 2 have been implemented, making use of the PETSc and SLEPc libraries.

#### 3.1 The PETSc/SLEPc Framework

PETSc, the Portable Extensible Toolkit for Scientific Computation [6], is a software framework for the parallel solution of numerical problems. Its design follows an object-oriented approach in order to be able to manage the complexity of numerical methods for very large and sparse problems on parallel computers. It is designed to provide enough flexibility to make software reuse feasible in many different contexts, but also with other goals in mind such as numerical robustness, computational efficiency, portability to different computing platforms, interoperability with other software, etc.

For solving linear systems of equations, PETSc provides a variety of iterative methods such as GMRES, that can be combined with different preconditioners such as Jacobi or block Jacobi [13]. PETSc allows the use of external libraries that are seamlessly integrated in the framework, thus complementing the offered functionalities. An example of such libraries is Hypre [9], which provides different preconditioners such as the algebraic multigrid preconditioner (AMG) [10] and a parallel incomplete LU with thresholding (PILUT).

SLEPc, the Scalable Library for Eigenvalue Problem Computations [11, 12], is a software library for the solution of large, sparse eigenvalue problems on parallel computers. It can be used for the solution of eigenproblems formulated in either standard or generalized form ( $Ax = \lambda x$  or  $Ax = \lambda Bx$ ), both Hermitian and non-Hermitian, with either real or complex arithmetic, as well as other related problems such as the singular value decomposition.

SLEPc is built on top of PETSc, and extends it with all the functionalities necessary for the solution of eigenvalue problems. It provides uniform and efficient access to a growing number of eigensolvers. Most of these solvers belong to the class of projection methods, see [5], either Krylov methods or Davidson methods. In particular, SLEPc implements several variants of the Arnoldi method, including the Krylov-Schur method [17] that incorporates a very efficient restarting mechanism. An implementation of the Jacobi-Davidson eigensolver [15] is also available. The user is able to easily switch among different eigensolvers by simply specifying the method at run time. Apart from the eigensolver, many other options

can be specified such as the number of eigenvalues to compute, the requested tolerance, or the portion of the spectrum of interest.

When implementing the proposed algorithms with PETSc and SLEPc, we can distinguish the following types of operations:

- Computation of initial approximations, that is, vectors  $u_n$  and  $v_n$  satisfying equations (6). This is done by computing eigenvalues and the corresponding right and left eigenvectors of matrix  $A_n$ . This computation is carried out only once with SLEPc, for instance with the Krylov-Schur eigensolver.
- Solution of linear systems of the form (11). This is carried out with PETSc.
- Matrix-vector multiplications. On one hand, it will be necessary to compute matrix-vector products with matrices  $A_n$  and  $A_m$ . On the other hand, restriction and prolongation from one grid to the other can also be performed by matrix-vector products with matrices  $R$  and  $E$ , or by direct algebraic implementation with simple MatmulR and MatmulE routines. In PETSc, all these matrices are represented as sparse matrices, and the MatMult operation can be invoked for carrying out the matrix-vector product in parallel.
- Vector operations, including inner product, norm, addition and scaling, will be performed with the VecNorm, VecAXPY and VecScale routines from PETSc, respectively.

### 3.2 Computation of Initial Approximations

In our implementation, the user is able to select, at run time, the number of eigenpairs to refine,  $nv$ . At the beginning,  $nv$  eigenvalues of matrix  $A_n$ , and the corresponding left and right eigenvectors, are computed with SLEPc. The method used, the requested precision, and other parameters such as the maximum subspace dimension, can also be set by the user. Among the available methods, Krylov-Schur was the best compromise between performance and robustness.

The Krylov-Schur method [17] is a Krylov projection eigensolver for non-Hermitian matrices, that incorporates an effective restarting mechanism. It uses the Arnoldi iteration for building an orthonormal basis of the  $k$ -dimensional Krylov subspace defined by the matrix,  $A_n$ , and a given initial vector. This mechanism is based on two stages. First, the space of approximants is reduced to a smaller dimension, taking care that the relevant spectral information is retained (the wanted eigenvalues). Then, the space is extended again to dimension  $k$ . This mechanism is equivalent to implicit restart [16], but it is simpler to implement because it is not necessary to keep a Hessenberg form throughout the computation. Additional details can be found in [17].

### 3.3 Defect-Correction Stage

We will now proceed to discuss how to solve system (11), that is,

$$(A_n - \theta_n I)t_n = \tilde{y}, \quad (14)$$

with  $\tilde{y} = Rr_m - v_n^* Rr_m u_n$  and  $t_n$  satisfying

$$v_n^* t_n = 0. \quad (15)$$

This system could be solved by an LU decomposition with partial pivoting, which would safely compute a solution vector  $t_n$  enforcing the constraint.

An alternative for solving Eq. (14) is to use an iterative solver. We first consider unpreconditioned GMRES. Recall that without a preconditioner, after  $k$  steps GMRES computes an approximate solution  $t_n^{(k)}$  as  $t_n^{(k)} = t_n^{(0)} + Q_k z$ . For simplicity, we assume that the initial guess  $t_n^{(0)}$  is zero, thus

$$t_n^{(k)} = Q_k z, \quad (16)$$

where  $Q_k$  is an orthogonal basis of a certain Krylov subspace, and the components of  $z$  are the coordinates of  $t_n^{(k)}$  with respect to that basis. In our case, the Krylov subspace is

$$\mathcal{K}_k(A_n - \theta_n I, r^{(0)}) = \text{span} \left\{ r^{(0)}, (A_n - \theta_n I)r^{(0)}, \dots, (A_n - \theta_n I)^{k-1} r^{(0)} \right\}, \quad (17)$$

with the initial residual being  $r^{(0)} = \tilde{y}$ . In order to get a solution vector that satisfies the constraint, Eq. (15), we must have

$$t_n^{(k)} = (I - u_n v_n^*) Q_k z, \quad (18)$$

and this can be achieved by applying the projector  $I - u_n v_n^*$  to all vectors  $q_i$  subsequently added to the basis. In our case,  $r^{(0)}$  already verifies the constraint (15).

An implementation of the above scheme requires a modification of GMRES, in such a way that  $(A_n - \theta_n I)q_j$ , the candidate vector to be added to the basis, is pre-multiplied by  $I - u_n v_n^*$ . The new vector is thus computed as  $(I - u_n v_n^*)(A_n - \theta_n I)q_j$ . For practical reasons, instead of  $I - u_n v_n^*$  we consider the projector  $I - \tilde{v}_n \tilde{v}_n^*$ , the orthogonal projector onto  $\mathcal{R}(v_n)^\perp$ , where  $\tilde{v}_n = v_n / \|v_n\|$ . This projector can be implemented very easily in PETSc and it also guarantees Eq. (15).

It remains to analyze what happens when preconditioning is used. In the case of GMRES preconditioned on the left, the analogue of Eq. (14) is

$$M^{-1}(A_n - \theta_n I)t_n = M^{-1}\tilde{y}, \quad (19)$$

where  $M$  is a certain approximation of  $A_n - \theta_n I$ . If at the beginning of the method the initial residual is preconditioned and then projected with  $I - u_n v_n^*$ , and the new basis vectors are computed by GMRES as  $(I - u_n v_n^*)M^{-1}(A_n - \theta_n I)q_j$ , then we end up with an approximate solution

$$t_n^{(k)} = (I - u_n v_n^*) \hat{Q}_k z, \quad (20)$$

where now the basis  $\hat{Q}_k$  corresponds to the preconditioned Krylov subspace  $\mathcal{K}_k(M^{-1}(A_n - \theta_n I), M^{-1}r^{(0)})$ . This solution will satisfy the constraint (15) as well.

From a more practical perspective, the implementation in PETSc has been done as follows. A linear solver object (KSP) is created outside the defect-correction loop. This is possible because the coefficient matrix of the linear system,  $A_n - \theta_n I$ , is constant throughout the computation. Therefore, the solver object is created and set-up only once, and this is very important because some parts of this preparation can be computationally very expensive, for instance the construction of an incomplete LU preconditioner. Once the solver object is prepared, the function `KSPSetNullSpace` is invoked in order to set vector  $\tilde{v}_n$  as the one that will constitute the projector used in all linear system solves, as discussed above.

As mentioned at the beginning of this section, PETSc provides many iterative linear solvers. We have used GMRES, although other solvers could have been used as well. Regarding the preconditioners, algebraic multigrid (AMG) showed a very good behaviour in this application, but others can be considered also (see section 4 for a comparison in terms of performance). A final comment in relation to the solution of the linear systems is that we

have requested a similar accuracy (tolerance) to both the linear solver and the outer defect correction iteration. A more relaxed tolerance is possible for the linear solver, but in that case the overall computation is less robust.

All parameters concerning the solution of linear systems, such as tolerances or preconditioners to be used, can be specified by the user in the command line.

### 3.4 Matrix Permutations

When developing parallel codes for distributed memory computer architectures, it is important to distribute the data among processors in such a way that computational work is equally balanced while communications are reduced as much as possible. For our implementation, we use PETSc's convention of distributing both matrices and vectors in a block row-wise scheme, that is, each processor owns a contiguous sequence of rows. In this subsection, we analyze the possibility of reducing the application's communications by a suitable permutation of the matrices.

We propose an algorithm to explore the relation between the patterns of the small and large matrices. For simplicity of exposition, it is assumed that  $m$  is a multiple of  $n$ . Then, we can view the fine computational grid as the result of interleaving  $m/n$  shifted copies of the coarse grid. If we permute the rows and columns of  $A_m$  in such a way that the indices of an embedded coarse grid are made consecutive, additionally consider the parallel distribution of the small matrix  $A_n$ , and renumber the unknowns so that the local pieces of all the copies corresponding to the same processor are placed contiguously, a block structured pattern results.

We have implemented the possibility of performing this permutation in both methods. If this option is selected, then the program works with matrices  $A_m(p, p)$ ,  $R(:, p)$ , and  $E(p, :)$ , where we have used the colon notation in the Matlab sense. The permutation vector  $p$  is computed as  $p = \text{perm}(i, n, m, nloc)$  with Algorithm 4, where  $i$  is the index corresponding to the natural ordering,  $n$  and  $m$  are the dimensions of the matrices involved, and  $nloc$  is the number of rows of  $A_n$  locally owned by each processor.

---

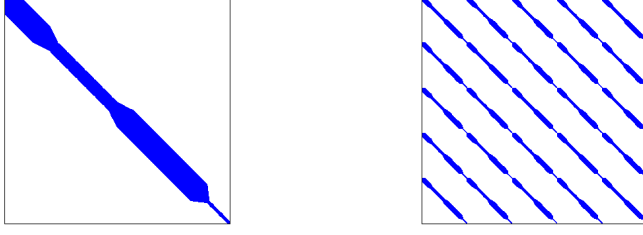
**Algorithm 4:**  $p = \text{perm}(i, n, m, nloc)$   
 $ratio = \frac{m}{n}; \quad pos = \text{floor}(\frac{i}{ratio}) + \text{rem}(i, ratio) * n$   
 $ii = pos - \text{floor}(\frac{pos}{n}) * n$   
 $p = ii + [\text{floor}(\frac{ii}{nloc}) * (ratio - 1) + \text{floor}(\frac{pos}{n})] * nloc$

---

## 4 Numerical Results and Performance Analysis

For the numerical tests we consider an integral formulation of a radiative transfer problem in stellar atmospheres with integral operator

$$(T\varphi)(\tau) = \frac{\varpi}{2} \int_0^{\tau^*} E_1(|\tau - \tau'|) \varphi(\tau') d\tau', \quad 0 \leq \tau < \tau^*, \quad (21)$$



**Fig. 1** Original sparsity pattern of matrix  $A_m$  (left) and pattern resulting from the interleave permutation (right).

defined on  $\mathcal{X} = L^1([0, \tau^*])$ , where  $\tau^*$  is the optical depth of the stellar atmosphere,  $\varpi \in ]0, 1[$  is the albedo and  $E_1$  is the first exponential integral function

$$E_1(\tau) = \int_1^\infty \frac{\exp(-\tau\mu)}{\mu} d\mu, \quad \tau > 0. \quad (22)$$

This problem has a logarithmic singularity at  $\tau = 0$  (a detailed description of this problem can be found in [3]). Piecewise constant functions for  $e_n$  were considered:  $e_{n,j}(\tau) = 1$  if  $\tau \in ]\tau_{n,j-1}, \tau_{n,j}[$  and 0 otherwise, determined by a grid of  $n + 1$  points (not necessarily uniform):  $0 = \tau_{n,0} < \tau_{n,1} < \dots < \tau_{n,n} = \tau^*$ . With this choice of basis, which is intended to absorb the singularity of the kernel, we have  $\langle e_{n,j}, e_{n,i}^* \rangle = \delta_{i,j}$  and

$$\langle x, e_{n,j}^* \rangle = \frac{1}{\tau_{n,j} - \tau_{n,j-1}} \int_{\tau_{n,j-1}}^{\tau_{n,j}} x(\tau) d\tau, \quad x \in \mathcal{X}.$$

For this example, the entries of matrix  $A_n$  can be obtained by analytical integration (see [3, 4]),

$$A_n(i, j) = \langle T e_{n,j}, e_{n,i}^* \rangle = \frac{\varpi}{2h_{n,i}} \int_{\tau_{n,i-1}}^{\tau_{n,i}} \int_0^{\tau} E_1(|\tau - \tau'|) e_{n,j}(\tau') d\tau' d\tau, \quad (23)$$

where  $h_{n,i} = \tau_{n,i} - \tau_{n,i-1}$ ,  $i = 1, \dots, n$ .

In our tests, we have always used a tolerance of  $10^{-11}$  for the refinement of eigenpairs, to show the performance of the methods, and consider a constant albedo of  $\varpi = 0.75$ .

We begin by illustrating the impact of the use of the permutation of Algorithm 4 on the sparsity pattern of matrix  $A_m$ , see Figure 1 (left) (the pattern of  $A_n$  is similar). Figure 1 (right) shows the first permutation done by the algorithm, where one can easily see  $m/n$  copies of the sparse pattern of  $A_n$ , repeated both horizontally and vertically. Of course, the resulting pattern is not yet appropriate for parallel computation with the straightforward decomposition by blocks of rows. Finally, Figure 2 illustrates the final result produced by the algorithm for the case of 4 and 8 processors, where a block tridiagonal structure is obtained.

Before analyzing the behaviour and performance of both methods we will consider the generation of the matrices  $A_m$ ,  $A_n$ ,  $R$  and  $E$ . These computations are embarrassingly parallel. The first two matrices are computed, for the example to be treated, by formula (23) (for the finer grid just replace  $n$  by  $m$ ). One could also have used other discretization methods for integral operators that compute these matrices by numerical quadrature formulae, or, for other examples the formulae for the elements of  $A_m$  and  $A_n$  may be simpler. Here, these

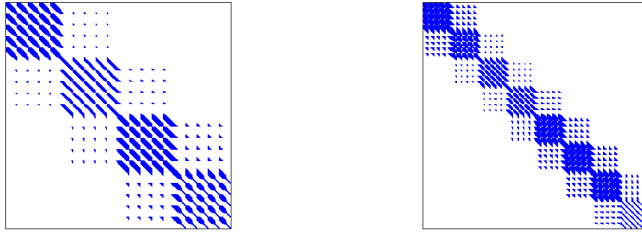


Fig. 2 Sparsity pattern of matrix  $A_m$  after permutation with 4 processors (left) and 8 processors (right).

computations are very time consuming, particularly in the case of  $A_m$ . The other two, and for the basis choice made, are simply  $E(k, j) = 1$  if  $k \in [(j-1)m/n + 1, jm/n]$  and 0 otherwise and  $R(i, k) = h_{m,k}/h_{n,i}$  if  $k \in [(i-1)m/n + 1, im/n]$  and 0 otherwise. More details can be found in [3, 4].

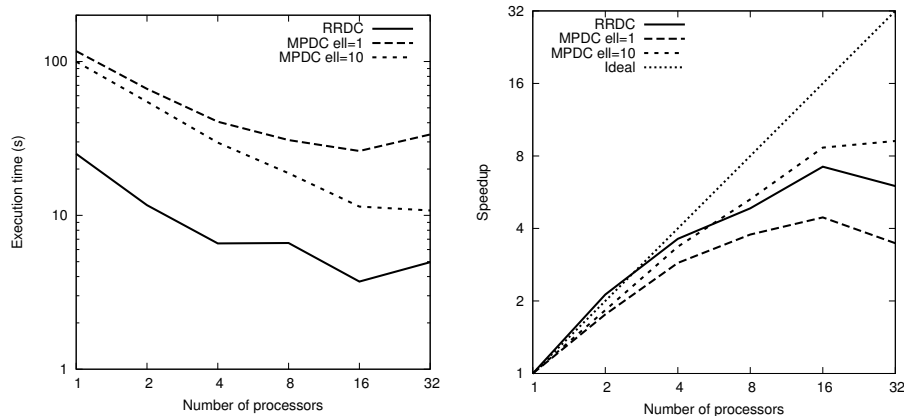
In order to optimize the generation of these matrices, our implementation makes use of a small cache to store the most recently computed exponential-integral evaluations. In this way, the computation of some entries of  $A_n$  and  $A_m$  can reuse one of the values stored in the cache, thus reducing the computational cost significantly. It is difficult to determine a priori which integral values were already computed, but, nevertheless, there is a locality effect that makes reuse more likely in close matrix elements. In our experience, a percentage of 25% cache hits is obtained in a typical run. In spite of this optimization, matrix generation is still very expensive computationally, with times of about two orders of magnitude larger than the computational times taken by the eigensolution stage.

In the following, we are mainly concerned with the time required by the different methods to compute one eigenpair, with a subspace dimension equal to 32 (when applicable).

The machine used for the parallel computations was the Odin cluster, located at Universitat Politècnica de València. It consists of 50 dual-processor nodes, with 2.8 GHz Pentium Xeon processors and 1 GB of memory per node. The nodes are interconnected with a high-speed SCI network with 2-D torus topology. Only one processor per node was used in the tests. The software configuration includes PETSc 3.1, SLEPc 3.1, and Hypr 2.6.0b.

Our first tests aimed at determining whether the matrix permutation described in subsection 3.4 achieves a performance gain or not with respect to the computation with the original ordering. The conclusion is that for this application there seems to be no significant benefit in using the permutation. The reason for this is that original pattern of the matrices is already narrowly banded. When we apply this code to other integral operators this preprocessing phase may be more beneficial.

Figure 3 shows a comparison of the time spent in the refinement stage of both the RRDC method and MPDC with  $\ell = 1$  and  $\ell = 10$ , together with the corresponding parallel speedup achieved, for a non-symmetric problem of order  $m = 64000$  with initial approximations obtained from problem size  $n = 6400$ . It is clear from the time plot that RRDC is better in terms of response time (almost an order of magnitude with respect to MPDC), although it scales a bit worse. In the MPDC method we can appreciate the impact of the  $\ell$  parameter: a few inner power iterations are beneficial to accelerate convergence with a moderate added cost. Larger values for  $\ell$  are possible, but there is a trade-off that depends on the application. At some point one can expect a degradation in the MPDC method: the cost becomes expensive without any further improvement. In terms of parallel speedup (right plot), performing some

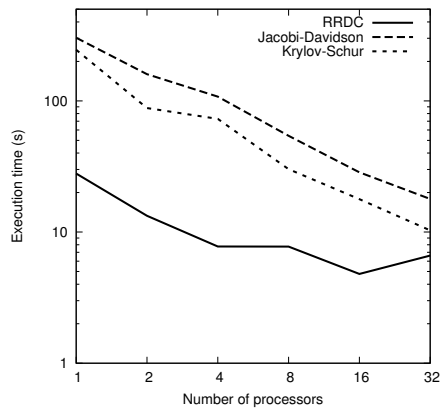


**Fig. 3** Execution time (left) and speedup (right) for the refinement step for  $m = 64000, n = 6400$  on up to 32 processors for RRDC and MPDC (with  $\ell = 1$  and  $\ell = 10$ ). The speedup is referred to time per iteration.

power iterations is beneficial from the parallel point of view because these iterations have less communication overhead compared to the rest of the algorithm. An additional comment is that in all cases the speedup for 32 processors is clearly decaying, since the problem size is not large enough.

Now we want to compare the proposed approach with state-of-the-art methods that operate only on the larger problem size ( $m = 64000$  in this case), to demonstrate that exploiting the reduced size approximations is really competitive. For this comparison, it is necessary to take into consideration also the time required to compute the initial approximations (left and right eigenvectors of  $A_n$ ), apart from the refinement iterations. In our case, we compute the eigentriplets of  $A_n$  with inexact shift-and-invert Krylov-Schur, that is, Krylov-Schur on matrix  $(A - \sigma I)^{-1}$  where the inverse is handled implicitly with an iterative linear solver (GMRES with AMG preconditioning, see below for a discussion on preconditioners). The cost of the initial approximations is directly related to the value of  $n$ , that is, the dimension of  $A_n$ . A good initial approximation (with larger  $n$ ) will result in less computational effort for the MPDC and RRDC methods, yet at a higher initial cost. On the other hand, a cheap coarse initial approximation implies a greater effort for the iterative refinement to achieve the required precision. Along with the understanding of this trade-off, it is also interesting to investigate whether the behavior on one processor persists or not when using several processing elements.

Figure 4 shows the total execution time of RRDC (including the initial approximations) as well as Jacobi-Davidson and shift-and-invert Krylov-Schur on the large problem ( $m = 64000$ ). From the figure, two conclusions can be drawn. First, the RRDC method is much faster, thus confirming our hypothesis that solving the correction equation in the smaller space is competitive. We have to point out that in these tests the tolerance used for the Jacobi-Davidson correction equation (solved with GMRES) can be relaxed to  $10^{-3}$  without compromising robustness, whereas for the refinement in RRDC it is necessary to use a tolerance of  $10^{-12}$ . Nevertheless, RRDC is still faster. Second, in terms of parallel speedup, the behaviour of RRDC is worse than the methods that operate exclusively on the larger space. This can be attributed to the loss of efficiency in the smaller space (where most of



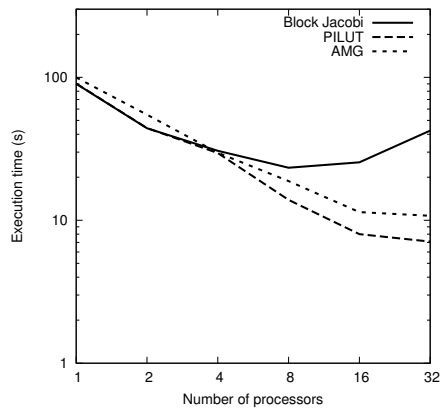
**Fig. 4** Total execution time for  $m = 64000$  and  $n = 6400$  on up to 32 processors for RRDC compared to Jacobi-Davidson and Krylov-Schur.

the computations are done in RRDC), since  $n = 6400$  is too small for such large number of processors. A more detailed scalability analysis is included at the end of this section.

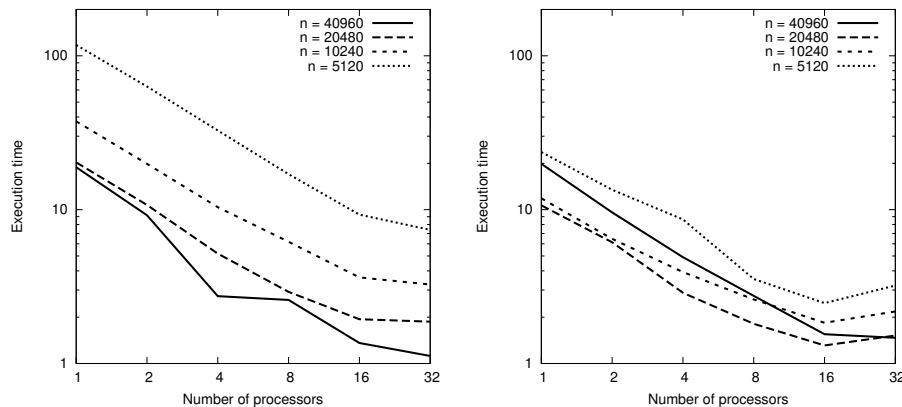
The results mentioned in the previous paragraph correspond to the computation of just one eigenpair. The situation changes when more eigenpairs are required, being more favorable for Krylov methods since they improve several approximations simultaneously whereas Davidson methods improve one at a time. For instance, the computation of three eigenpairs with shift-and-invert Krylov-Schur takes about the same time as one (243 seconds), but Jacobi-Davidson will require approximately three times more. We do not report times for this setting because our implementation of RRDC does not include locking, which is required for computing several eigenvalues, but we anticipate a behaviour similar to Jacobi-Davidson. Regarding MPDC, it takes 364 seconds with one processor (about 99, 120, and 145 seconds for the first, second and third eigenvalue, respectively), so it is slower than Krylov-Schur. However, we must mention that the refinement of each eigenvalue is totally independent, and therefore their computation can be trivially distributed to different processes, or in a multi-communicator scheme (two-level parallelism). In this case, the required time would be the maximum of the three (145 seconds).

Several preconditioners were tested for the solution of (11) to accelerate GMRES: Jacobi, block Jacobi, PILUT, and AMG, see Figure 5. From the tested preconditioners, all except Jacobi perform well. For increasing number of processors, both PILUT and AMG show a steady decrease in computing time. Thus, their speedup seems to be very good, at least up to 32 processors. In the case of block Jacobi, the number of iterations required by the linear solver grows with the number of processors, as expected since the effectiveness of this preconditioner decays with an increase in the number of subdomains. Therefore, speedup stagnates starting after 8 processors. Both PILUT and AMG have a very good performance, and PILUT is slightly faster in this application. Jacobi is not shown in the plot due to excessively long computation times. Some of the above mentioned preconditioners have a high set-up cost, but one should notice that the computation of the preconditioner matrix is carried out only once, at the beginning of the Defect-Correction iteration.

In order to better assess the scalability of the proposed methods, we need to use a larger problem size. At the same time, we are going to analyze the impact of the ratio between the large ( $m$ ) and small ( $n$ ) dimensions. Figure 6 shows the times to compute a refined solution for a symmetric problem of dimension  $m = 204800$ , from initial approximations obtained



**Fig. 5** Execution time for  $m = 64000$  and  $n = 6400$  on up to 32 processors for several preconditioners to accelerate GMRES. The method used is MPDC with  $\ell = 10$ .



**Fig. 6** Total execution time for  $m = 204800$  and for several values of  $n$  on up to 32 processors, for MPDC (left) and RRDC (right). The value of  $\ell$  is 10 in this case.

for several values of  $n$ . Times include the computation of initial approximations and the refinement iterations. In the case of MPDC (left plot) we can see that the smaller the ratio the better the performance, and this indicates that MPDC has more difficulties when refining eigenvalues from coarse initial approximations. Nevertheless, the scalability is good up to 32 processors. In contrast, the RRDC method (right plot) seems to be more robust with respect to variations of the ratio  $m/n$ . With a small number of processors, an intermediate value of the ratio is best. But, as discussed previously, for large number of processors it is worthwhile to set a sufficiently large value of  $n$ . Overall, MPDC scales better than RRDC.

## 5 Conclusions

With this work we provide two parallel codes for the computation of eigenpairs of integral operators. They are based on the defect correction principle and were implemented on top

of the software libraries PETSc and SLEPc. A brief description of the methods to introduce the implementation details was presented. Several numerical tests were done, using an astrophysics application as test problem, which requires the solution of large dimensional eigenproblems.

Both the Multipower Defect-Correction and the Rayleigh-Ritz Defect-Correction methods are able to refine eigenpairs from initial rough approximations. The fact that they start with small dimensional eigenproblems is certainly an important feature when approximations for large dimensional problems or for fine discretizations are required. The methods are well suited and effective for the computation of a small set of eigenvalues. Although these methods are not general-purpose such as Jacobi-Davidson or Krylov-Schur, they can be very effective for computing eigenvalues of integral operators. It is also worth mentioning that the proposed methods can be used to compute eigenvalues in any part of the spectrum, not only in the rightmost extreme.

From the tests performed with the parallel implementation of both methods, we can conclude that they showed good behavior on a distributed-memory parallel environment. The methods showed good scalability and robustness. The test problem used is numerically more difficult to solve as the problem size grows, since the eigenvalues tend to be closer to each other, that is why Krylov methods need to be used in combination with a shift-and-invert technique to achieve convergence.

The subspace variant, RRDC, could be enriched with standard techniques that have been well studied in iterative eigensolvers, such as restarting mechanisms, or harmonic extraction.

## Acknowledgements

We thank the two anonymous referees for their valuable comments and suggestions, which greatly helped us improve this document.

## References

1. Absil PA, Mahony R, Sepulchre R, Dooren PV (2002) A Grassmann-Rayleigh quotient iteration for computing invariant subspaces. *SIAM Review* 44(1):57–73
2. Ahues M, Largillier A, Limaye BV (2001) *Spectral Computations with Bounded Operators*. Chapman and Hall, Boca Raton, USA
3. Ahues M, d’Almeida FD, Largillier A, Titaud O, Vasconcelos P (2002) An  $L^1$  refined projection approximate solution of the radiation transfer equation in stellar atmospheres. *Journal of Computational and Applied Mathematics* 140(1-2):13–26
4. Ahues M, d’Almeida FD, Largillier A, Vasconcelos PB (2006) Defect correction for spectral computations for a singular integral operator. *Communications on Pure and Applied Analysis* 5(2):241–250
5. Bai Z, Demmel J, Dongarra J, Ruhe A, van der Vorst H (eds) (2000) *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA
6. Balay S, Buschelman K, Eijkhout V, Gropp WD, Kaushik D, Knepley M, McInnes LC, Smith BF, Zhang H (2010) PETSc users manual. Tech. Rep. ANL-95/11 - Revision 3.1, Argonne National Laboratory
7. Chatelin F (2011) *Spectral Approximation of Linear Operators*. SIAM, Philadelphia

- 
8. d'Almeida FD, Vasconcelos PB (2012) Convergence of multipower defect correction for spectral computations of integral operators. submitted
  9. Falgout RD, Yang UM (2002) hypre: A library of high performance preconditioners. In: Sloot PMA, Tan CJK, Dongarra J, Hoekstra AG (eds) Computational Science - ICCS 2002, International Conference, Amsterdam, The Netherlands, April 21-24, 2002. Proceedings, Part III, Springer, Lecture Notes in Computer Science, vol 2331, pp 632–641
  10. Henson VE, Yang UM (2002) BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics: Transactions of IMACS* 41(1):155–177
  11. Hernandez V, Roman JE, Vidal V (2005) SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software* 31(3):351–362
  12. Hernandez V, Roman JE, Tomas A, Vidal V (2010) SLEPc users manual. Tech. Rep. DSIC-II/24/02 - Revision 3.1, D. Sistemes Informàtics i Computació, Universitat Politècnica de València
  13. Saad Y (2003) *Iterative methods for sparse linear systems*, 2nd edn. Society for Industrial and Applied Mathematics, Philadelphia, PA
  14. Simoncini V, Eldén L (2002) Inexact Rayleigh quotient-type methods for eigenvalue computations. *BIT Numerical Mathematics* 42(1):159–182
  15. Sleijpen GLG, van der Vorst HA (2000) A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM Review* 42(2):267–293
  16. Sorensen DC (1992) Implicit application of polynomial filters in a  $k$ -step Arnoldi method. *SIAM Journal on Matrix Analysis and Applications* 13:357–385
  17. Stewart GW (2001) A Krylov–Schur algorithm for large eigenproblems. *SIAM Journal on Matrix Analysis and Applications* 23(3):601–614