Journal of Computational Physics 227 (2008) 9477–9497

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Computing several eigenpairs of Hermitian problems by conjugate gradient iterations

E.E. Ovtchinnikov

Harrow School of Computer Science, University of Westminster, Watford Road, Northwick Park, London HA1 3TP, UK

ARTICLE INFO

Article history:
Received 12 April 2007
Received in revised form 11 December 2007
Accepted 28 June 2008
Available online 30 July 2008

Keywords:
Eigenvalue computation
Conjugate gradient method

ABSTRACT

The paper is concerned with algorithms for computing several extreme eigenpairs of Hermitian problems based on the conjugate gradient method. We analyse computational strategies employed by various algorithms of this kind reported in the literature and identify their limitations. Our criticism is illustrated by numerical tests on a set of problems from electronic structure calculations and acoustics.

1. Introduction

The conjugate gradient (CG) method was originally introduced as a method for solving linear systems by [11]. Later it was re-interpreted as a functional minimization method and has since been widely used as such in the smooth nonlinear optimization, see e.g. [9,29,5,23]. CG is also used in the Hermitian eigenvalue computation, via the minimization of the Rayleigh quotient functional, see [3,22,8,6], and is popular as an eigenvalue computation tool with computational physicists, see e.g. [30]. One of the attractive features of CG method is that it can be applied to the generalized eigenvalue problem $Lx = \lambda Mx$, where L and M are Hermitian operators, the latter assumed positive definite, without the need to solve linear systems with either L or M or a linear combination thereof. Another useful feature is the ability to directly employ the so-called *preconditioning*, a convergence acceleration technique that opens way to the efficient solution of large-scale problems and is nowadays widely used for solving large linear systems. We note that most other methods for solving large-scale eigenvalue problems, such as Lanzcos method and its variants, do not enjoy the first feature, and can only use preconditioning indirectly, for solving certain auxiliary linear systems.

The application of the conjugate gradient method to the minimization of the Rayleigh quotient functional produces the leftmost eigenvalue and corresponding eigenvector. (For simplicity of discussion, here we refrain from considering the case where iterations stop at a saddle point, i.e. at an eigenvector corresponding to an internal eigenvalue.) Further eigenpairs can be computed by conjugate gradient iterations in the subspace orthogonal to the computed eigenvector, or some other *deflation* technique, see e.g. [21]. However, the convergence of iterations can be very slow in the case of a problem with clustered eigenvalues. By analogy with the power method (see e.g. [21]), one expects 'cluster robust' convergence if several vectors are iterated simultaneously, and indeed, the convergence results of [18,20] suggest that this is the case for conjugate gradients as well.

A number of conjugate gradient algorithms for simultaneous computation of eigenpairs can be found in the literature; our discussion of these exclude various algorithms that use CG for solving certain auxiliary linear systems. In [16] CG is applied simultaneously to several vectors with orthogonalization of new vector iterates after each CG step ensuring that they do not converge to the same eigenvector. In this paper we refer to this approach to the use of CG in eigenvalue computation as

E-mail address: e_ovtchinnikov@hotmail.com

0021-9991/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2008.06.038

simultaneous. In addition, they considered, and discarded, two alternative approaches. The first one, which later became known as *trace minimization*, essentially employed CG for the minimization of the sum of the Rayleigh quotients on m orthogonal vectors, m being the number of eigenpairs to be computed. The second employed the Rayleigh–Ritz procedure in the trial subspace spanning current vector iterates and the respective gradients of the Rayleigh quotient conjugated to previous search directions; here we refer to this approach as the *block* one. All three approaches have since been used (in some cases being apparently re-invented) by other authors. The simultaneous CG algorithm of [16], enhanced with preconditioning, is employed by [26]. The use of CG for the trace minimization is thoroughly studied by [6].[1] The block CG algorithm described by [16] was re-invented by [1]. An algorithm based on a conceptually close block CG approach was suggested by [15]. The present paper introduces a variation of the block CG algorithm of [16] and a new block CG algorithm designed in [20] based on the local convergence analysis of a generic block CG scheme instantiated by the block algorithms just mentioned.

In this paper we analyse, both theoretically and numerically, the computational strategies implemented by the mentioned CG algorithms for simultaneous eigenvalue computation and comment on their merits and limitations. Such a discussion does not appear to be found elsewhere in the literature, and some authors seem to be unaware that the approaches to eigenvalue computation they employ can be found in [16]. As a result, the arguments *pro et contra* certain ways to use CG in eigenvalue computation offered by [16] remain ignored. The same fate appears to befall the ideas of other authors. The CG algorithms of [6] are mentioned in neither of the recent publications [1,2], and although referenced, this paper has not attracted any comments of [15]. The results of the latter paper, in turn, are not mentioned by [2], who have chosen to implement the algorithm of [26] clearly inferior to that of [15], as can be seen from the numerical tests of the present paper.

Our discussion of various approaches to the use of CG in eigenvalue computation is illustrated by numerical tests on several problems from the electronic structure computation and acoustics, two areas where CG is especially competitive due to the availability of very efficient AMG preconditioners and the high computational cost of matrix factorizations hindering the performance of shift-and-invert eigensolvers. An extensive range of numerical tests with CG with AMG preconditioning and algorithms based on shift-and-invert techniques demonstrating the competitiveness of the former can be found in [1]. To a certain extent, the numerical tests of the present paper complement those of the cited paper, which contains tests of two arbitrarily chosen block CG algorithms but does not analyse comparatively their observed performance. It should be stressed, however, that the methodologies employed by the two papers are quite different. While the conclusions of [1] are entirely based on experimental results, the criticism of the present paper is brought about by theoretical arguments, and the tests are there mostly for illustration purposes.

The plan of the paper is as follows: In Section 2, we discuss the available conjugate gradient algorithms for eigenvalue computation, starting with algorithms for computing the leftmost eigenvalue and proceeding to a more detailed description and discussion of algorithms for simultaneous eigenvalue computation mentioned above. Based on this discussion, we select a group of algorithms the convergence properties of which we illustrate in Section 4. In Section 3, we give formal description of the selected algorithms and calculate their computational costs. In Section 4, we present and discuss the results of numerical tests with these algorithms.

2. Conjugate gradient schemes for eigenvalue computation

2.1. Schemes for computing the leftmost eigenpair

Let us briefly discuss the application of the conjugate gradient method to the computation of the leftmost eigenvalue of a Hermitian operator before addressing the computation of several eigenpairs simultaneously.

Let L be a Hermitian operator. Denote by λ_j and x_j the eigenvalues and eigenvectors of L enumerated in the ascending order of λ_j, and denote by $\lambda(u)$ the corresponding Rayleigh quotient functional, i.e.

$$\lambda(u) = \frac{(Lu,u)}{\|u\|^2}. \tag{1}$$

The minimum value of $\lambda(u)$ is λ_1; hence λ_1 can be computed by the conjugate gradient method for the minimization of the Rayleigh quotient functional $\lambda(u)$. We remind that this method produces a sequence of vector iterates x^i by the following two-term recurrent formulae:

$$x^{i+1} = x^i - \alpha_i y^i, \tag{2}$$

$$y^{i+1} = g^{i+1} + \beta_i y^i, \tag{3}$$

where x^0 is an arbitrary nonzero vector (initial guess), $y^0 = g^0$, g^i is the gradient of $\lambda(u)$ at $u = x^i$, and α_i minimizes $\lambda(x^i - \alpha y^i)$. We observe that step (2) employs linear search in the direction indicated by y^i, hence here we refer to y^i as the search direction vector, or simply search direction. Specific properties of the Rayleigh quotient functional allow certain modifications of the above general CG scheme: the new vector iterate x^{i+1} can be normalized, the gradient g^i can be replaced by the residual vector $r^i = r(x^i) = Lx^i - \lambda(x^i)x^i$, which is collinear to g^i, and the linear search can be replaced with the minimization in the two-dimensional subspace $\mathrm{span}\{x^i,\ y^i\}$, for which the Rayleigh–Ritz procedure can be employed. Yet another approach is to minimize $\lambda(v)$ along a geodesics on the sphere $\|v\| = 1$ (see [6]).

[1] The trace minimization approach is also employed in the two-level algorithm of [25] with inner CG iterations; as pointed out earlier, in the present paper we do not discuss this and other algorithms that use CG in such subsidiary way.

Step (3) is referred to as the conjugation of g^{i+1} to y^i, or else the conjugation of the search directions y^{i+1} and y^i. Various conjugation schemes, i.e. ways to compute β_i, have been suggested in the literature, notably:

$$\beta_i = \frac{\|g^{i+1}\|^2}{\|g^i\|^2} \quad \text{(Bradbury–Fletcher scheme)}; \tag{4}$$

$$\beta_i = \frac{(g^{i+1},\ g^{i+1} - g^i)}{\|g^i\|^2} \quad \text{(Polak–Ribiere scheme)}; \tag{5}$$

$$\beta_i = -\frac{(H(x^{i+1})g^{i+1},\ y^i)}{(H(x^{i+1})y^i,\ y^i)} \quad \text{('exact conjugation' scheme)}; \tag{6}$$

$$\beta_i = -\frac{(g^{i+1},\ y^i)_L}{\|y^i\|_L^2} \quad \text{(Perdon–Gambolati scheme)} \tag{7}$$

(cf. respectively [3,23,5,22]). In (6), $H(x^{i+1})$ is the Hessian of $\lambda(u)$ at $u = x^{i+1}$, or a multiple thereof, e.g.

$$H(u) = (1 - 2\,P_u)(L - \lambda(u))(1 - 2\,P_u), \tag{8}$$

where P_u is the orthogonal projector onto u (the term 'exact conjugation' for (6) is borrowed from [6]). In (7) L is assumed to be positive definite.

We observe that the above expressions are borrowed verbatim from the case of the quadratic functional $\psi(u) = (Lu,\ u) - (f,\ u) - (u,\ f)$ with Hermitian positive definite L, and they are actually used for the minimization of a general smooth nonlinear functional $\psi(u)$ (in which case (4) is usually referred to as Fletcher–Reeves scheme, since it was adapted to a general $\psi(u)$ by [9]). Yet another way to compute β_i was suggested by Takahashi [29]: β_i can be chosen to minimize the functional at hand at $x^{i+2} = x^{i+1} - \alpha_{i+1}(g^{i+1} + \beta_i y^i)$. It is not difficult to see that in this case the new vector iterate x^{i+2} minimizes the functional in question in the two-dimensional manifold $x^{i+1} + \mathrm{span}\{y^i,\ y^{i+1}\} = x^{i+1} + \mathrm{span}\{y^i,\ g^{i+1}\}$, and, in the case of the Rayleigh quotient functional, in the three-dimensional subspace $\mathrm{span}\{x^{i+1},\ g^{i+1},\ y^i\} = \mathrm{span}\{x^{i+1},\ g^{i+1},\ x^i\}$; hence this scheme is known as *locally optimal* [24,14].

Yet another conjugation scheme is suggested by [19] based on the so-called Jacobi orthogonal complement correction equation of [27]:

$$J(\lambda_1, x^{i+1})(x^{i+1} - x_1) = r^{i+1}, \tag{9}$$

where x_1 is normalized in such way that $x^{i+1} - x_1$ is orthogonal to x^{i+1}, and

$$J(\mu, u) = (1 - P_u)(L - \mu)(1 - P_u). \tag{10}$$

Since the residual r^{i+1} is collinear to the gradient of $\lambda(u)$ at $u = x^{i+1}$, and x^{i+1} is the minimum point of $\lambda(u)$ on the line $u(\tau) = x^i - \tau y^i$, the residual r^{i+1} is orthogonal to y^i, and hence the correction is orthogonal to y^i in the sense of the semi-scalar product $(J(\lambda_1, x^{i+1})\cdot,\cdot)$ (it is not difficult to see that $J(\lambda_1, u)$ is positive semi-definite for any u). In view of this, [19] suggests making y^{i+1} orthogonal to y^i in $J(\lambda_1, x^{i+1})$-scalar product, which minimizes the $J(\lambda_1, x^{i+1})$-angle between y^{i+1} and the correction vector. The resulting conjugation scheme, with λ_1 replaced by any available approximation thereof, e.g. λ^{i+1}, is referred to as Jacobi scheme by [19], and is shown to be asymptotically (as $x^i \to x_1$) equivalent to Takahashi (locally optimal), Polak–Ribiére and exact conjugation schemes there.

It is important to note that in the framework of the conjugate gradient method any scalar product can be used for computing the gradients g^i (the same scalar product and corresponding norm must then be used in the above expressions for β_i), which opens way to the use of *preconditioning*. In practical terms, if the energy scalar product (Nu, v) for a certain Hermitian positive definite operator N is used, then the gradient g^i is proportional to $N^{-1}r^i$, and the step (3) transforms into

$$y^{i+1} = Kr^{i+1} + \beta_i y^i, \tag{11}$$

where $K = N^{-1}$. Accordingly, the conjugation schemes become:

$$\beta_i = \frac{\|r^{i+1}\|_K^2}{\|r^i\|_K^2} \quad \text{(Bradbury–Fletcher scheme)}; \tag{12}$$

$$\beta_i = \frac{(r^{i+1},\ r^{i+1} - r^i)_K^2}{\|r^i\|_K^2} \quad \text{(Polak–Ribiere scheme)}; \tag{13}$$

$$\beta_i = -\frac{(H(x^{i+1})Kr^{i+1},\ y^i)}{(H(x^{i+1})y^i,\ y^i)} \quad \text{('exact conjugation' scheme)}; \tag{14}$$

$$\beta_i = -\frac{(Kr^{i+1},\ y^i)_L}{\|y^i\|_L^2} \quad \text{(Perdon–Gambolati scheme)}; \tag{15}$$

$$\beta_i = -\frac{(J(\lambda^{i+1}, x^{i+1})Kr^{i+1},\ y^i)}{(J(\lambda^{i+1}, x^{i+1})y^i,\ y^i)} \quad \text{(Jacobi scheme)}. \tag{16}$$

It remains to note that the conjugate gradient method can also be applied to the generalized eigenvalue problem $Lx = \lambda Mx$, in which case the Rayleigh quotient and residual vector become

$$\lambda(u) = \frac{(Lu, u)}{(Mu, u)}, \quad r(u) = Lu - \lambda(u)Mu, \tag{17}$$

while the (scaled) Hessian $H(u)$ and the Jacobi correction operator $J(u, \mu)$ become

$$H(u) = (1 - 2P^*_{u,M})(L - \lambda(u)M)(1 - 2P_{u,M}), \tag{18}$$

$$J(\mu, u) = (1 - P^*_{u,M})(L - \mu M)(1 - P_{u,M}), \tag{19}$$

where $P_{u,M}$ is the orthogonal projector onto span$\{u\}$ in the sense of the scalar product $(\cdot, \cdot)_M = (M\cdot, \cdot)$ and $P^*_{u,M}$ is its adjoint. We stress that M^{-1} is not used by any of the CG schemes discussed here.

2.2. Schemes for computing several leftmost eigenpairs

As mentioned in the Introduction, one way to compute several eigenpairs, and the most commonly used one, is to compute them successively by single-vector iterations using this or that *deflation* technique. For instance, having computed the first eigenpair by any of the conjugate gradient schemes discussed in the previous section, we proceed to minimizing the Rayleigh quotient in the subspace orthogonal to the computed eigenvector, which produces the second eigenpair, and so on. Here we discuss various alternative computational schemes suggested in the literature whereby eigenpairs of interest are computed simultaneously.

Below the superscript i refers to the iteration number, and subscript j enumerates approximate eigenpairs in the ascending order of approximate eigenvalues.

The `SIRQIT-CG` algorithm of [16] computes the search direction y^i_j by conjugating the gradient g^i_j of $\lambda(u)$ at $u = x^i_j$ to the respective previous search direction y^{i-1}_j (Bradbury–Fletcher scheme is suggested), and orthogonalizes the new CG iterate $x^{i+1}_j = x^i_j - \alpha_{ij} y^i_j$ to $x^{i+1}_1, \ldots, x^{i+1}_{j-1}$. After certain number of iterations (three are recommended), the Rayleigh–Ritz procedure is applied in the trial subspace spanning x^i_j and the iterations are restarted. Sartoretto et al. [26] use the same iterative scheme with preconditioning and orthogonalization of y^i_j to $x^{i+1}_1, \ldots, x^{i+1}_{j-1}$, and recommend 10–20 inner iterations. Fu and Dowling [10] use a similar scheme but with conjugation by Perdon–Gambolati scheme and no Rayleigh–Ritz procedure.

The computational strategy implemented by the algorithms just described is to stick to single-vector CG iterations keeping the interaction between individual vector iterates to a minimum between the restarts. The rationale for this strategy given by [16] is to avoid 'destruction of the underlying motivation for conjugacy', which is is considered as undermining the method. The intention to use a familiar method, single-vector CG, proved to be successful in practical computation and supported by plausible theoretical arguments (that any smooth functional is 'almost quadratic' in the vicinity of a minimum point), is fairly understandable. However, avoiding the interaction between individual vector iterates is, actually, not such a good idea. Indeed, focusing for a moment on the vectors x^i_1 produced by the algorithms in question between restarts, we observe that they coincide with those produced by a single-vector CG, hence, any improvement in performance over the single-vector iterations with deflation (indeed observed in numerical tests) may only come from the Rayleigh–Ritz procedure. Thus, on the one hand, one needs to do restarts frequently enough, in order to benefit from this procedure, and, on the other hand, not too frequently, in order to fully benefit from the CG convergence acceleration compared to much less efficient steepest descent, and any choice for the inner iterations number (between 1 and infinity) inevitably compromises both acceleration mechanisms.

Alongside with the approach just described, [16] considered, and discarded, two alternative approaches to the use of CG in eigenvalue computation. The first one, which has since become known as the *trace minimization*, utilises the fact that the normalized eigenvectors corresponding to m leftmost eigenvalues minimize the quadratic functional

$$\psi(u_1, \ldots, u_m) = \sum_{j=1}^{m} (Lu_j, u_j) = \mathrm{Tr}\ U^* L U, \quad U = [u_1, \ldots, u_m], \tag{20}$$

over all sets of m orthonormal vectors $u_1, \ldots, u_m$. Hence, the eigenvalue problem for the Hermitian operator L acting in n-dimensional Euclidean space can be re-formulated as the quadratic minimization problem in nm-dimensional space subject to constraint $U^*U = I$ (or $U^*MU = I$, in the case of the generalized problem). where I is the m-by-m identity matrix, or else the unconstrained problem for the non-quadratic functional

$$\psi(u_1, \ldots, u_m) = \mathrm{Tr}((U^*U)^{-\frac{1}{2}} U^* L U (U^*U)^{-\frac{1}{2}}). \tag{21}$$

[16] considered solving this problem by CG iterations

$$X^{i+1} = X^i - \alpha_i Y^i, \quad Y^{i+1} = G^{i+1} + \beta_i Y^i, \tag{22}$$

where $X^i = [x^i_1, \ldots, x^i_m]$, $Y^i = [y^i_1, \ldots, y^i_m]$ and $G^i = [g^i_1, \ldots, g^i_m]$. Observing that a CG iteration is controlled by just two scalar quantities, α_i and β_i, rather than $2m$ employed by the simultaneous CG scheme they favoured, [16] dismissed this approach as not delivering 'a commensurate increase in power'.

The second alternative approach considered by [16] can be described as follows: Denote by $\mathscr{RR}_m(\mathscr{V})$ the set of m Ritz vectors in the trial subspace $\mathscr{V}$ corresponding to m leftmost Ritz values (see Section A.1 for the summary of the Rayleigh–Ritz procedure). As mentioned above, the linear search step (2) can be replaced by the Rayleigh–Ritz procedure in $\mathrm{span}\{x^i, y^i\}$, i.e.

$$x^{i+1} = \mathscr{RR}_1(\mathrm{span}\{x^i,\ y^i\}). \tag{23}$$

Accordingly, [16] suggested the following 'block' generalization of (23) and (3):

$$X^{i+1} = \mathscr{RR}_m(\mathrm{span}\{X^i,\ Y^i\}) \tag{24}$$

$$Y^{i+1} = G^{i+1} + Y^i B_i, \tag{25}$$

where B_i is a m-by-m diagonal matrix and X^i, Y^i and G^i are same as above. In this paper, we refer to the conjugate gradient schemes of such form (but with a general m-by-m matrix B_i) as 'block' ones. It is observed in [16] that the iterative scheme (24) and (25) 'appears to have the necessary power'; nevertheless, it is discarded as 'improper extension of Rayleigh quotient minimization by CG', since conjugacy of n-by-m matrices 'is not a well-defined concept'. Both alternatives are claimed to be proved inferior in performance in numerical tests, the results of which, however, are not presented in the cited paper.

Both of the two approaches suggested and discarded by [16] appear to come largely unnoticed and similar iterative schemes have been suggested later by several authors, often without a reference to [16]. A trace minimization CG algorithm based on rather sophisticated geometrical calculations was suggested in [6]. Johnson and Joannopoulos [13] observed that this algorithm did not seem to work well when preconditioning is employed and suggested using iterations (22) for the unconstrained functional (21) instead. It is interesting to note that in Section 4.4 of [6] yet another trace minimization scheme is mentioned that is essentially a block scheme (24) and (25) with $B_i = \beta_i I_m$, where I_m is m-by-m identity and β_i is a scalar given by

$$\beta_i = \frac{\langle G^{i+1}, G^{i+1} - G^i\rangle}{\langle G^i, G^i\rangle}, \quad \langle U, V\rangle = \mathrm{Tr}\ V^*U. \tag{26}$$

Numerical test of the present paper show that this approach to the trace minimization works reasonably well with preconditioning. A block conjugate gradient scheme with a diagonal B_i based on Bradbury–Fletcher scheme, i.e. with $\|g_j^{i+1}\|^2/\|g_j^i\|^2$ on the diagonal, was re-invented and shown to be actually working, albeit in the framework of a somewhat different iterative scheme, by [1]; in this paper we test a similar block Polak–Ribiére scheme.

The trace minimization approach is another example of a computational strategy that aims at employing existing CG schemes that have been used in nonlinear optimization. The criticism of this approach offered by [16] is rather vague and appeals to reader's intuition: it looks highly unlikely that an algorithm employing a conjugation scheme with just one scalar parameter β_i would be competitive to that employing several conjugation parameters. However, in our tests, an algorithm based on the trace minimization approach actually often performed better than the algorithm favoured by [16], owing largely to the aforementioned internal conflict ailing the latter. Still, treating all eigenpairs equally, an inherent feature of the trace minimization approach, has its limitations. Available convergence results for CG (see e.g. [5]) estimate the convergence rate in terms of the condition number of the Hessian of the minimized functional. The calculations of [6] (Section 4.4) imply that this condition number is asymptotically (as $\mathrm{span}\{X^i\}$ approaches the invariant subspace corresponding to $\lambda_1, \ldots, \lambda_m$) equal to that of the block diagonal operator matrix $\widehat{L}$ with operators $\widehat{L}_j = (1-P)(L-\lambda_j)(1-P)$ on the diagonal, where P is the orthogonal projector onto the invariant subspace corresponding to m leftmost eigenvalues. Simple calculation shows that the latter condition number is $(\lambda_n - \lambda_m)/(\lambda_{m+1} - \lambda_m)$ and hence, the convergence may be slow if the eigenvalues of interest are poorly separated from the rest of the spectrum. To make things worse, the convergence of *all* m eigenpairs, rather than just the mth one, may be slow because the algorithm minimizes the sum of Rayleigh quotients. A conceptually similar case is that of the system $\widehat{L}\hat{u} = 0$. It is not difficult to see that in the case where the gap $\lambda_{m+1} - \lambda_m$ is small, the poor conditioning of the last block $\widehat{L}_m$ would hinder the convergence of all parts of the vector iterate $\hat{u}^i$ to the solution $\hat{u}$ (a vector in the null-space of $\widehat{L}$), whereas the convergence of the part $\hat{u}_1^i$ corresponding to the diagonal block $\widehat{L}_1$ could be faster (assuming that λ_1 is further from λ_{m+1} than λ_m) if the system $\widehat{L}_1\hat{u} = 0$ was solved independently. In the case of the eigenvalue problem, such an 'indiscriminate' convergence behaviour of the trace minimization CG is rather unfortunate from the practical point of view because it denies the algorithm the benefits of deflation, i.e. removing the convergent eigenpairs from further calculations. The use of the Rayleigh–Ritz procedure introduces a certain degree of discrimination between individual eigenpairs, so that leftmost ones converge earlier than the rest, but still not early enough compared to other algorithms, as we will see in Section 4.

The block CG approach is radically different to the previous two in its attempt to introduce an iterative scheme that is genuinely designed for simultaneous computation of eigenvalues, rather than borrowed from the nonlinear optimization. Longsine and McCormick [16] did not consider their attempt quite successful in view of the uncertainty regarding the conjugation matrix B_i, and the diagonal B_i they contemplated, apparently for want of a better choice, is merely the one they used in `SIRQIT-CG`. Our experience is that it fails sometimes, but with Bradbury–Fletcher scheme replaced by Polak–Ribiére one, (24 and 25) proved to be a formidable competitor to other CG schemes discussed here.

An approach conceptually close to the block CG scheme of [16] is employed by [15], where the following iterative scheme is suggested:

$$X^{i+1} = \mathscr{R}\mathscr{R}_m(\mathrm{span}\{X^i,\ G^i,\ X^{i-1}\}). \tag{27}$$

The above scheme is obviously related to the locally optimal (Takahashi) scheme, and the algorithm of [15] based on this scheme is known as the *locally optimal block (preconditioned) conjugate gradient* (LOBPCG) method; the term 'preconditioned' indicates explicitly that gradients can be computed in any scalar product – cf. Section 2.1. The new approximate eigenvalues produced by (27) are, by the minimax principle, not greater than those produced by (24 and 25) for any conjugation matrix B_i, and the LOBPCG invariably converged significantly faster in terms of the number of iterations than (24 and 25) with diagonal B_i in our tests. However, the computational cost of a LOBPCG iteration is considerably higher than that for all previously mentioned CG algorithms, and so is the risk of the Rayleigh–Ritz procedure failure caused by poorly conditioned basis of the trial subspace.

It is theoretically possible to reformulate LOBPCG as a block CG scheme 24 and 25 with B_i satisfying the local optimality condition in the sense of the smallest possible new approximate eigenvalues. It is difficult though, if possible at all, to obtain an explicit formula for such local optimal B_i. The investigation of the convergence properties of 24 and 25 by [20] resulted in a suggestion for B_i that is optimal in a different sense, notably: each new search direction y_j^{i+1} is asymptotically (as $\mathrm{span}\{X^{i+1}\}$ approaches the invariant subspace corresponding to $\lambda_1, \ldots, \lambda_m$) close to the best possible one for the line search $x_j^{i+2}(\tau) = x_j^{i+1} - \tau(1 - P_{i+1})y_j^{i+1}$, where P_{i+1} is the orthogonal projector onto X^{i+1} (again, the orthogonalization ensures convergence to different eigenpairs). Ovtchinnikov [20] shows that these 'individually locally optimal' search directions satisfy the orthogonality condition $(J(\lambda^{i+1},\ X_j^{i+1})y_j^{i+1}, y_j^i) = 0$, where $J(\mu, U)$ is a generalization of the Jacobi orthogonal complement correction operator $J(u, \mu)$ of the previous section given by

$$J(\mu, U) = (1 - P_U)(L - \mu)(1 - P_U), \tag{28}$$

and P_U is the orthogonal projector onto $\mathrm{span}\{U\}$. In order to obtain compact expressions for the entries $\beta_{kj}^{(i)}$ of B_i, [20] rewrites (24 and 25) equivalently as

$$[X^{i+1},\ Z^{i+1}] = \mathscr{R}\mathscr{R}_{m_i}(\mathrm{span}\{X^i,\ Y^i\}) \tag{29}$$

$$Y^{i+1} = G^{i+1} + Z^{i+1}B_i, \tag{30}$$

where m_i is the dimension of the trial subspace $\mathrm{span}\{X^i,\ Y^i\}$ and the columns z_k^{i+1} of Z^{i+1} are Ritz vectors corresponding to $l_i = m_i - m$ right-most Ritz values in this subspace. The entries of the conjugation matrix B_i are then given by

$$\beta_{kj}^{(i)} = -\frac{((L - \lambda(x_j^{i+1}))g_j^{i+1},\ z_k^{i+1})}{\lambda(z_k^{i+1}) - \lambda(x_j^{i+1})}. \tag{31}$$

The optimality enjoyed by the search directions of block Jacobi-conjugation scheme just described is obviously of a weaker kind than that enjoyed by LOBPCG. However, this does not automatically imply that the convergence of the Jacobi scheme has to be slower; in fact, in some of our tests it was even faster than that of LOBPCG. Apparently, the situation we face here is similar to that with LOBPCG compared to the block generalized Davidson method. Based on numerical tests with certain globally optimal algorithm, [15] conjectures that although the approximate eigenvalues computed on each step of the block generalized Davidson method are not greater than those computed by a step of LOBPCG, the convergence of the former cannot be significantly faster than that of the latter. The numerical tests of the present paper, performed on a wider range of problems, suggest that the same might be said about Jacobi scheme versus LOBPCG.

In what follows we illustrate numerically the performance the iterative scheme described above with the exception of Fu–Dowling scheme, which uses Perdon–Gambolati conjugation scheme that is shown to be inferior to other schemes in performance by [8]. For successive eigenvalue computation we opt for Polak–Ribiére scheme, which proved to be remarkably efficient and reliable in our tests, and for the trace minimization, we opt for ((24)–(26)).

3. Algorithms

In this subsection, we present formal descriptions of the algorithms that have been tested numerically. Below n_e stands for the number of eigenpairs needed and m for the number of iterated vectors: in algorithms for simultaneous computation of eigenpairs it is advisable to use $m > n_e$ in order to avoid convergence problems with the rightmost eigenpairs due to the poor separation of the respective eigenvalues from the rest of the spectrum. We assume that a set of m linearly independent vectors $x_1, \ldots, x_m$ is available at the start of the iterations. For simplicity of notation, we drop the superscript indicating the iteration number, i.e. current approximate eigenvectors are denoted simply by x_j. The maximal number of iterations is denoted by $i_{\max}$.

For the sake of generality, the algorithms are formulated for the generalized eigenvalue problem $Lx = \lambda Mx$; modifications for the case of the standard eigenvalue problem are fairly obvious. Below we use the terms M-orthogonalization and M-normalization for the orthogonalization and normalization in the scalar product $(\cdot,\cdot)_M = (M\cdot,\cdot)$ and norm $\|\cdot\|_M = \sqrt{(\cdot,\cdot)_M}$ (note that this notation is extended to arbitrary Hermitian positive definite M).

For each algorithm we give a summary of main computational expenses; the rest are negligible for the values of m and n used in the tests.

3.1. Successive Polak–Ribiére algorithm

One can compute eigenpairs by single-vector CG with deflation using any conjugation scheme – in our tests we opt for that by Polak–Ribiére.

Algorithm 3.1. [Successive computation of eigenpairs by Polak–Ribiére scheme with deflation]

For $k = 1, \ldots, m = n_e$, do:

(1) If $k > 1$, then M-orthogonalize x_k to $x_1, \ldots, x_{k-1}$ and M-normalize it.
(2) For $i = 1, 2, \ldots, i_{max}$, do:
  (a) Compute the Rayleigh quotient $\lambda_k = \lambda(x_k)$ and the residual $r = Lx_k - \lambda_k M x_k$.
  (b) If the norm of the residual is less than the residual tolerance, go to next k.
  (c) Compute $g = Kr$.
  (d) M-orthogonalize g to $x_1, \ldots, x_k$.
  (e) If $i > 1$, then
    (i) compute $\gamma = (r, g)$ and $\beta = (\gamma - (r_p, g))/\gamma_p$, where r_p and γ_p are the residual vector and γ from the previous iteration;
    (ii) if $\beta < 0$ set $\beta = 0$;
    (iii) compute the new search direction $y = g + \beta y_p$, where y_p is the previous search direction.
  (f) Set $r_p = r$, $y_p = y$ and $\gamma_p = \gamma$.
  (g) M-orthogonalize y to $x_1, \ldots, x_{k-1}$.
  (h) Compute the Gram matrices

$$L_k = \begin{bmatrix} \lambda_k & (Ly, x_k) \\ (x_k, Ly) & (Ly, y) \end{bmatrix}, \quad M_k = \begin{bmatrix} 1 & (My, x_k) \\ (x_k, My) & (My, y) \end{bmatrix},$$

  and compute the M_k-normalized eigenvector $[\sigma, \tau]^T$ of the matrix pencil $L_k - \lambda M_k$ corresponding to the leftmost eigenvalue; make sure that $\sigma > 0$.
    (i) Update $x_k = \sigma x_k + \tau y$.

(3) Terminate with error flag indicating that not all eigenpairs converged.

We note that 2e.ii follows a recommendation of [17].

Each iteration of the above algorithm requires one application of K, one application of L, for computing Ly at step 2h, and two applications of M, one at step 2g and one for computing My at step 2h. The new residual at step 2a is computed by updating recursively Lx_k and Mx_k rather than by applying L and M.

Main memory storage requirements of the above algorithm are m vectors x_j of dimension n; the number of multiplications by M can be halved by storing extra m vectors Mx_j.

Main computational cost on top of the application of K, L and M is the computation of $(m-1)m$ scalar products and axpy's (linear combinations $ax + y$ of two vectors x and y of dimension n) for the orthogonalization of g and y to $x_1, \ldots, x_{k-1}$.

3.2. Sartoretto–Pini–Gambolati algorithm

The next algorithm is a modification of the `SRQMCG2` algorithm of [26], a preconditioned version of `SIRQIT-CG` algorithm of [16], with restart every $n_{inner} > 1$ iteration. The modification introduced here consists in the use of a different stopping criterion that allows deflation.

Algorithm 3.2. [`SRQMCG2` by [26] with deflation]

- Set the total number of iterations i_{total} and the number of convergent eigenpairs l to 0.
- While $i_{total} < i_{max}$, do repeatedly:

(1) Compute two $(m-l)$-by-$(m-l)$ Gram matrices $L_{l,m}$ and $M_{l,m}$ with entries $\lambda_{ij} = (Lx_j, x_i)$ and $\mu_{ij} = (Mx_j, x_i)$, $i, j = l+1, \ldots, m$, and compute eigenvectors $s_j = [\sigma_{ij}]_{i=l+1}^{m}$, $j = 1, \ldots, m-l$ of the matrix pencil $L_{l,m} - \lambda M_{l,m}$. Update

$$x_{l+j} := \sum_{i=l+1}^{m} \sigma_{ij} x_i, \quad j = 1, \ldots, m-l.$$

(2) Set the number of non-convergent eigenpairs n_{nc} to zero.
(3) For $i = 1, \ldots, n_{inner}$ do:

(a) For $k = l+1, \ldots, m$ do:
  (i) Compute the Rayleigh quotient $\lambda_k = \lambda(x_k)$ and the residual $r = Lx_k - \lambda_k M x_k$.
  (ii) If the norm of the residual is less than the residual tolerance, then
    (A) if $i = 1$ and $k = l+1$, then increment l;
    (B) if $n_{\mathrm{nc}} = 0$, then go to next k; otherwise increment n_{nc}.
  (iii) Compute $g = Kr$.
  (iv) If $k > 1$, M-orthogonalize g to $x_1, \ldots, x_{k-1}$.
  (v) If $i > 1$, then
    ·compute $\gamma_k = (r, g)$ and $\beta = \gamma_k/\gamma_{k,\mathrm{p}}$, where $\gamma_{k,\mathrm{p}}$ is taken from the previous iteration (see step 3a.vi below);
    ·compute the new search direction $y_k = g + \beta y_{k,\mathrm{p}}$, where $y_{k,\mathrm{p}}$ is the respective previous search direction; otherwise set $y_k = g$.
  (vi) Set $y_{k,\mathrm{p}} = y_k$ and $\gamma_{k,\mathrm{p}} = \gamma_k$.
  (vii) Compute the Gram matrices

$$L_k = \begin{bmatrix} (Lx_k, x_k) & (Ly_k, x_k) \\ (x_k, Ly_k) & (Ly_k, y_k) \end{bmatrix}, \quad M_k = \begin{bmatrix} (Mx_k, x_k) & (My_k, x_k) \\ (x_k, My_k) & (My_k, y_k) \end{bmatrix},$$

  and compute the M_k-normalized eigenvector $[\sigma, \tau]^{\mathrm{T}}$ of the matrix pencil $L_k - \lambda M_k$ corresponding to the leftmost eigenvalue; make sure that $\sigma > 0$.
  (viii) Update $x_k = \sigma x_k + \tau y_k$.
  (ix) If $k > 1$, then M-orthogonalize x_k to $x_1, \ldots, x_{k-1}$.

(b) Increment i_{total}.
(c) If $l \geqslant n_{\mathrm{e}}$, then set l to zero, execute step 1 and stop.
(d) If $n_{\mathrm{nc}} = 0$ go to step 1.

Each inner iteration of the above algorithm requires one application of K, two applications of L (at steps 3a.i and 3a.vii') and four applications of M (at steps 3a.i, 3a.iv, 3a.vii, and 3a.ix) per each non-convergent eigenpair.

Main memory storage requirements are $2m$ vectors x_k and y_k. The number of L-applications can be reduced to one per non-convergent eigenpair by storing extra $2m$ vectors Lx_k and Ly_k (cf. Section 3.1); the same applies to M.

Main computational cost on top of the application of K, L and M is the computation of $(m-1)m$ scalar products and axpy's for the orthogonalization of x_k and y_k to $x_1, \ldots, x_{k-1}$.

The impact of the Rayleigh–Ritz procedure is, for $n_{\mathrm{inner}} = 10$, as used in the tests, negligible compared to other costs.

The difference between the above algorithm and the algorithm `SRQMCG2` by [26] is that the cited paper suggests stopping the iterations when the average residual norm defined as

$$r_{\mathrm{a}} = \sqrt{\frac{1}{m}\sum_{i=1}^{m} \|r(x_j)\|^2}$$

falls below the specified tolerance. One drawback of such a stopping criterion is that one generally has to iterate vectors which have already converged to the required accuracy (potentially, one may even end up with a zero residual, which might lead to overflow when computing β). Another drawback is that individual residuals are not guaranteed to be smaller than the tolerance multiplied by $\sqrt{m}$, which might be a problem for large m.

The stopping criteria implemented by the modification of Sartoretto–Pini–Gambolati algorithm used here aims at removing convergent eigenpairs from the computation as soon as reasonable in order to reduce the amount of computation on each inner iteration and the size of the trial subspace of the Rayleigh–Ritz procedure on step 1. An eigenpair is deemed to be convergent if the following three conditions are met: (i) the residual norm is smaller than the required residual tolerance (cf. step 3a.ii), (ii) the approximate eigenvector in question is a Ritz vector (cf. the condition that $i = 1$ in 3a.ii.A), and (iii) the eigenpair in question is the leftmost non-convergent (cf. the condition that $k = l+1$ in 3a.ii.A). In order to reduce the number of applications of the preconditioner, the conjugate gradient step is not applied if the first of the above three conditions is met by the approximate eigenpair in question and all those to the left of it (cf. the condition $n_{\mathrm{nc}} = 0$ in 3a.ii.B).

3.3. Block Polak–Ribiére algorithm

The next algorithm uses an instance of the block CG scheme (24) and (25) with a diagonal B_i computed by Polak–Ribiére scheme.

Algorithm 3.3. Algorithm 3.3. [Block Polak–Ribiére algorithm]

(1) Compute two m-by-m Gram matrices L_X and M_X with entries $\lambda_{ij} = (Lx_j, x_i)$ and $\mu_{ij} = (Mx_j, x_i)$, $i, j = 1, \ldots, m$, and compute M_X-normalized eigenvectors $s_j = [\sigma_{ij}]_{i=1}^{m}$, $j = 1, \ldots, m$ of the matrix pencil $L_X - \lambda M_X$. Compute initial approximate eigenvectors x_j, $j = 1, \ldots, m$, by updating

$$x_j := \sum_{i=1}^{m} \sigma_{ij} x_i.$$

(2) Set the number of converged eigenpairs l to zero.
(3) For $i = 1, 2, \ldots, i_{\max}$, do repeatedly:
  (a) Compute the Rayleigh quotients $\lambda_j = \lambda(x_j)$ and the residuals $r_j = Lx_j - \lambda_j Mx_j$, $j = l+1, \ldots, m$.
  (b) While the norm of r_{l+1} is less than the residual tolerance, increment l; if $l \geqslant n_e$ execute step 1 and terminate.
  (c) Compute $W = KR$, where $R = [r_{l+1}, \ldots, r_m]$ and $W = [w_{l+1}, \ldots, w_m]$.
  (d) Compute $\gamma_j = (w_j, r_j)$. If $i > 1$, then
    (i) compute $\beta_j = \max\{0, (\gamma_j - (w_j, r_{j,p}))/\gamma_{j,p}\}$, $j = l+1, \ldots, m$, where $r_{j,p}$ and $\gamma_{j,p}$ are the residual r_j and scalar product γ_j taken from the previous iteration (see step 3e);
    (ii) compute the new search direction matrix $Y = [y_{l+1}, \ldots, y_m]$, where $y_j = w_j + \beta_j z_j$ and z_j are taken from the previous iteration (see step 3e); else set Y = W.
  (e) Set $z_j = y_j$, $r_{j,p} = r_j$ and $\gamma_{j,p} = \gamma_j$, $j = l+1, \ldots, m$.
  (f) M-orthogonalize Y to $X_{all} = [x_1, \ldots, x_m]$.
  (g) Compute $W = MY$ and $M_Y = Y^*W$, and compute the eigenvalues μ_j and eigenvectors q_j, $j = 1, \ldots, m-l$, of the auxiliary eigenvalue problem

$$M_Y q = \mu D_Y q, \tag{32}$$

  where D_Y is a diagonal matrix with the same diagonal as M_Y. Select μ_j that are not less than ϵm, where ϵ is significantly above the machine accuracy, and update $Y := YQ$ and $W := WQ$, where the columns of the matrix Q are the corresponding eigenvectors q_j.
  (h) Compute matrices $L_{X,Y}$ and $M_{X,Y}$ given by

$$L_{X,Y} = \begin{bmatrix} \Lambda & X^*LY \\ (LY)^*X & Y^*LY \end{bmatrix}, \quad M_{X,Y} = \begin{bmatrix} I & X^*W \\ W^*X & Y^*W \end{bmatrix}, \tag{33}$$

  where $\Lambda = \mathrm{Diag}\{\lambda_{l+1}, \ldots, \lambda_m\}$, and I is the $m-l$-by-$m-l$ identity matrix, and compute the $M_{X,Y}$-normalized eigenvectors s_j of the problem

$$L_{X,Y} s = \lambda M_{X,Y} s. \tag{34}$$

  Form the matrix S whose columns are the eigenvectors s_j corresponding to $m-l$ leftmost eigenvalues of (34) and update $X := [X, Y]S$.
(4) Terminate with error flag indicating that not all eigenpairs converged.

Each iteration of the above algorithm requires $m-l$ applications of K, $2(m-l)$ applications of L (steps 3a and 3h) and $3(m-l)$ applications of M (steps 3a, 3f and 3g).

Main memory requirements are the storage of X, Y, R, R_p and W ($5m$ vectors). The number of L-applications can be reduced to $m-l$ per iteration by storing LX and LY (extra $2m$ vectors); the same holds for M.

Main computational cost per iteration on top of the application of K, L and M is the computation of matrices $V = X_{all}^* MY$ on step 3h, X^*MY on step 3f, Y^*MY on steps 3g and 3h X^*LY on step 3h and Y^*LY on step 3h, $(m(m-l) + 5(m-l)^2)n$ multiplications in total, and the computation of products $X_{all}V$ on step 3f, YQ and WQ on step 3g and $[X, Y]S$ on step 3h, $(m(m-l) + 4(m-l)^2)n$ multiplications in total, assuming that no search directions are discarded on step 3g. For the standard problem WQ is not computed, hence $(m-l)^2 n$ less multiplications are needed. Reducing the number of L-multiplications by storing L-images requires additional $2(m-l)^2 n$ multiplications for the update $LX := [LX, LY]S$; the same additional cost is incurred by using this technique for M.

The role of step 3g is to ensure that Cholesky factorization of $M_{X,Y}$, which is used by DSYGV eigensolver routine from LAPACK employed in the implementation of this and other algorithms, does not fail because of the loss of positive definiteness caused by round-off errors. In exact arithmetic, $M_{X,Y}$ is a diagonal matrix with μ_j on the diagonal. Hence, μ_j that are close to the round-off error level are too sensitive to round-off errors, and Cholesky factorization becomes too inaccurate or may fail altogether. Dropping μ_j's that are below the cut-off level used on step 3g ensures that this does not happen. At the same time, the closeness of the cut-off level to the round-off error level keeps angles between the discarded directions and accepted ones close to minimal possible.

3.4. Trace minimization algorithm

The trace minimization algorithm tested in this paper uses another instance of the generic block CG scheme (24 and 25), the one with $B_i = \beta_i I_m$, where β_i is given by (26). Unlike with other schemes of [6], the use of preconditioning is fairly straightforward and the scheme applies verbatim to the generalized eigenvalue problem without the need to compute $M^{-1}v$. The only difference from Algorithm 3.3 is that all β_j are equal to

$$\beta = \frac{\mathrm{Tr}\ (R - R_\mathrm{p})^* KR}{\mathrm{Tr}\ R_\mathrm{p}^* K R_\mathrm{p}}, \tag{35}$$

where R and R_p are matrices with columns r_j and $r_{j,\mathrm{p}}$, $j = l+1, \ldots, m$, respectively.

3.5. Block Jacobi-conjugate gradients algorithm

The next algorithm again uses an instance of the block CG scheme (24 and 25); this time with a full matrix B_i that makes new search directions 'individually optimal' (see Section 2.2).

Algorithm 3.4. Algorithm 3.5. [Block Jacobi-conjugate gradients.]

(1) Perform step 1 of Algorithm 3.3.
(2) Set the number of converged eigenpairs l to zero.
(3) For $i = 1, 2, \ldots, i_{\max}$, do repeatedly:
  (a) Compute the Rayleigh quotients $\lambda_j = \lambda(x_j)$ and the residuals $r_j = Lx_j - \lambda_j M x_j$, $j = l+1, \ldots, m$.
  (b) While the norm of r_{l+1} is less than the residual tolerance, increment l; if $l \geqslant n_\mathrm{e}$ execute step 1 and terminate.
  (c) Compute $W = KR$, where $R = [r_{l+1}, \ldots, r_m]$ and $W = [w_{l+1}, \ldots, w_m]$.
  (d) If $i > 1$, then
    (i) compute matrices $\Gamma = (LZ)^* W$ and $\Delta = (MZ)^* W$, where Z is the previous search directions matrix taken from the previous iteration (see step 3g);
    (ii) compute matrix B_i with entries β_{pq} given by

$$\beta_{pq} = -\frac{\gamma_{pq} - \lambda_{l+q}\delta_{pq}}{\mu_\mathrm{p} - \lambda_{l+q}}, \quad p = 1, \ldots, n_Z, \quad q = 1, \ldots, m-l, \tag{36}$$

    where γ_{pq} and δ_{pq} are entries of Γ and Δ, n_Z is the width of Z, and μ_p are n_Z rightmost eigenvalues of the problem (34) solved at the previous iteration (see step 3g);
    (iii) compute the new search direction matrix $Y = W + ZB_i$.
      else set $Y = W$.
  (e) M-orthogonalize Y to $X_\mathrm{all} = [x_1, \ldots, x_m]$.
  (f) Perform step 3g of Algorithm 3.3.
  (f) Compute matrices $L_{X,Y}$ and $M_{X,Y}$ given by (33), and compute the $M_{X,Y}$-normalized eigenvectors s_j of the problem (34). Form the matrix S whose columns are the eigenvectors s_j corresponding to $m-l$ leftmost eigenvalues of (34) and the matrix V whose columns are the remaining eigenvectors of (34), compute $Z = [X, Y]V$ and update $X := [X, Y]S$.
(4) Terminate with error flag indicating that not all eigenpairs converged.

Each iteration of the above algorithm requires $m-l$ applications of K, $3(m-l)$ applications of L (steps 3a, 3d and 3g) and $4(m-l)$ applications of M (steps 3a, 3d, 3e and 3f).

Main memory requirements are the storage of X, Y, Z and W ($4m$ vectors). The number of L-applications can be reduced to $m-l$ per iteration by storing LX and LY (extra $2m$ vectors); the same holds for M.

Main computational cost per iteration on top of the application of K, L and M is the computation of matrices $(LZ)^* W$ and $(MZ)^* W$ on step 3d, $V = X_\mathrm{all}^* MY$ on step 3e, $X^* MY$ on step 3g, $Y^* MY$ on steps 3f and 3g, $X^* LY$ on step 3g and $Y^* LY$ on step 3d, $(m(m-l) + 7(m-l)^2)n$ multiplications in total, and the computation of products ZB_i on step 3d, $X_\mathrm{all} V$ on step 3e, YQ and WQ on step 3f and $[X, Y]S$ and $[X, Y]V$ on step 3g, $(m(m-l) + 7(m-l)^2)n$ multiplications in total, assuming that no search directions are discarded on step 3f. For the standard problem WQ is not computed, hence $(m-l)^2 n$ less multiplications are needed. Reducing the number of L-multiplications by storing L-images requires additional $4(m-l)^2 n$ multiplications for the update $LX := [LX, LY]S$ and $LZ = [LX, LY]V$; the same additional cost is incurred by using this technique for M.

3.6. Stabilized LOBPCG

The LOBPCG algorithm as described by [15] suffers from instabilities due to a poorly conditioned basis of the trial subspace. In our tests we use a version of LOBPCG that is stabilized by discarding 'almost' linearly dependent search directions in the same manner as in Algorithm 3.3.

Algorithm 3.5. Algorithm 3.6. [Stabilized LOBPCG.]

(1) Perform step 1 of Algorithm 3.3.
(2) Set the number of converged eigenpairs l to zero.
(3) For $i = 1, 2, \ldots, i_{\max}$, do repeatedly:
  (a) Compute the Rayleigh quotients $\lambda_j = \lambda(x_j)$ and the residuals $r_j = Lx_j - \lambda_j M x_j$, $j = l+1, \ldots, m$.

(b) While the norm of r_{l+1} is less than the residual tolerance, increment l; if $l \geqslant n_e$ execute step 1 and terminate.
(c) Compute $Y = KR$, where $R = [r_{l+1}, \ldots, r_m]$.
(d) M-orthogonalize Y to $X_{\text{all}} = [x_1, \ldots, x_m]$.
(e) Perform step 3g of Algorithm 3.3, using R in place of W.
(f) Compute matrices given by (33) using R in place of W. If $i = 1$, then go to step 3g, else go to step 3h.
(g) Compute the $M_{X,Y}$-normalized eigenvectors s_j of the problem (34). Form the matrix $[S_X, S_Y]^{\mathrm{T}}$, where S_X is $(m-l)$-by-$(m-l)$, whose columns are the eigenvectors s_j corresponding to $m-l$ leftmost eigenvalues of (34). Compute $Z = YS_Y$, update $X := XS_X + Z$ and go to next i.
(h) M-orthogonalize Z to $X_{\text{all}} = [x_1, \ldots, x_m]$ and Y.
(i) Perform step 3e with Y replaced with Z.
(j) Compute matrices

$$L_{X,Y,Z} = \begin{bmatrix} L_{X,Y} & [X,Y]^* LZ \\ (LZ)^* [X,Y] & Z^* LZ \end{bmatrix}, \quad M_{X,Y,Z} = \begin{bmatrix} M_{X,Y} & [X,Y]^* R \\ R^* [X,Y] & Z^* R \end{bmatrix},$$

where $X = [x_{l+1}, \ldots, x_m]$, $R = MY$ and $L_{X,Y}$ and $M_{X,Y}$ are given by (33), and compute the $M_{X,Y,Z}$-normalized eigenvectors s_j of the problem

$$L_{X,Y,Z} s = \lambda M_{X,Y,Z} s. \tag{37}$$

Form the matrix S whose columns are the eigenvectors s_j corresponding to $m-l$ leftmost eigenvalues of (37). Split S vertically into blocks S_X, S_Y and S_Z corresponding to X, Y and Z and update $Z = YS_Y + ZS_Z$ and $X := XS_X + Z$.

(4) Terminate with error flag indicating that not all eigenpairs converged.

Each iteration of the above algorithm requires $m-l$ applications of K, $3(m-l)$ applications of L (steps 3a,3f and 3j) and $5(m-l)$ applications of M (steps 3a,3d,3e,3h and 3i).

Main memory requirements are the storage of X, Y, Z, and R ($4m$ vectors). The number of L-applications can be reduced to $m-l$ per iteration by storing LX, LY and LZ (extra $3m$ vectors); the same holds for M.

Main computational cost per iteration on top of the application of K, L and M is the computation of matrices $Q_Y = X_{\text{all}}^* MY$ on step 3d, $X^*R = X^*MY$ on step 3f, $Y^*R = Y^*MY$ on steps 3e and 3f, X^*LY and Y^*LY on step 3f, $Q_Z = [X_{\text{all}}, Y]^* MZ$ on step 3h, $X^*R = X^*MZ$ and $Y^*R = Y^*MZ$ on step 3j, $Z^*R = Z^*MZ$ on steps 3i and 3j, X^*LZ, Y^*LZ and Z^*LZ on step 3j, $(2m(m-l) + 13(m-l)^2)n$ multiplications in total, and the computation of products $X_{\text{all}}Q_Y$ on step 3d, YQ and RQ on step 3e $[X_{\text{all}}, Y]Q_Z$ on step 3h, ZQ and RQ on step 3i and XS_X, YS_Y and ZS_Z on step 3j, $(2m(m-l) + 8(m-l)^2)n$ multiplications in total, assuming that no search directions are discarded on step 3e. For the standard problem RQ is not computed on steps 3e and 3i, hence $2(m-l)^2 n$ less multiplications are needed. Reducing the number of L-multiplications by storing L-images requires additional $3(m-l)^2 n$ multiplications for the update of LX and LZ on step 3j and $(m(m-l) + (m-l)^2)n$ multiplications for the update of LZ on step 3h; the same additional cost is incurred by using this technique for M.

3.7. The main linear algebra cost summary for block schemes

For reader's convenience, the number of matrix multiplications for the four block CG schemes is summarized in Tables 1 (standard problem) and 2 (generalized problem). We note that these multiplications represent the main computational expenses per iteration outside the application of K, L and M.

Table 1
Matrix multiplications per iteration: the standard problem

Algorithm	$V_m^* W_k$	$V_k^* W_k$	$V_m Q_m (+L)$	$V_k Q_k (+L)$
Polak–Ribiére/TM	1	5	1	3(+2)
Jacobi	1	7	1	6(+4)
LOBPCG	2	13	2(+1)	6(+4)

Table 2
Matrix multiplications per iteration: the generalized problem

Algorithm	$V_m^* W_k$	$V_k^* W_k$	$V_m Q_m (+L)(+M)$	$V_k Q_k (+L)(+M)$
Polak–Ribiére/TM	1	5	1	4(+2)(+2)
Jacobi	1	7	1	7(+4)(+4)
LOBPCG	2	13	2(+1)(+1)	8(+4)(+4)

In the tables, V_p and W_p stand for n-by-p matrices, and Q_p for a p-by-k matrix, where $k = m - l$. The items in brackets represent the number of additional multiplications for the case where L-images and/or M-images are stored.

4. Numerical illustration

This section presents the results of numerical experiments with algorithms described in the previous section applied to several standard and generalized eigenvalue problems.

In our tests we use an algebraic multigrid (AMG) preconditioner very similar to that used by [1]; the differences between the two are technical and totally irrelevant to the purposes of the present paper. A brief description of the AMG preconditioning can be found in the cited paper, further details and references can be found in the review by [28].

Numerical experiments were performed on Dell Precision 490 workstation with Intel® Xeon® 5130 dual core processor at 2 GHz and 3 GB RAM. The tested algorithms were implemented in C++ using Microsoft®.NET compiler. For basic linear algebra operations (dense matrix multiplications featuring in Tables 1 and 2 and the like), highly optimized BLAS and LAPACK routines included in Intel® Math Kernel Library (version 6.1) were employed. In all tests, (pseudo-)randomly generated initial vectors were used.

4.1. Preliminary tests

We begin with the standard Laplacian-in-a-brick problem discretized by 7-point finite differences, i.e. we compute eigenvalues of the matrix

$$L = L_{n_x,a_x} \otimes I_{n_y} \otimes I_{n_z} + I_{n_x} \otimes L_{n_y,a_y} \otimes I_{n_z} + I_{n_x} \otimes I_{n_y} \otimes L_{n_z,a_z},$$

where $L_{n,a}$ is an n-by-n three-diagonal matrix with $2h^{-2}$ and $-h^{-2}$ on the main diagonal and the two adjacent diagonals respectively, $h = a/(n+1)$, I_n is the n-by-n identity and $\otimes$ stands for the tensorial product of matrices.

In out first series of tests we do not use preconditioning, so that the convergence is determined by the spectrum of L only. The purpose of these tests is to try to verify that the algorithms at hand enjoy the kind of convergence one would expect from CG. We remind that in the case of a linear system $Lx = f$, the number of CG iterations needed to reduce the error by a given factor is bounded from above by a value approximately proportional to the square root of the condition number of L. A comparison of available convergence results for linear systems and eigenvalue problems suggests that in eigenvalue computation the role of the said condition number is played by the ratio $(\lambda_n - \lambda_j)/(\lambda_{m+1} - \lambda_j)$, where m is as above, i.e. the number of simultaneously iterated vectors (see e.g. [16]). Assuming $n_x = n_y = n_z = n_{xyz}$, the latter quantity is of the order n_{xyz}^2. Hence, a linear growth of the iteration number in n_{xyz} suggests that the algorithm in question enjoys a 'proper' CG convergence.

The two plots in Fig. 1 correspond to $n_{xyz} = 10$ and 40, respectively. In the legends, 'successive PR' stands for 'successive Polak–Ribiére' (Algorithm 3.1), 'SPG' for 'Sartoretto–Pini–Gambolati' (Algorithm 3.2), 'PR' for '(block) Polak–Ribiére' (Algorithm 3.3), 'tm' for 'trace minimization employing Rayleigh–Ritz procedure' (the algorithm from Section 3.4), 'Jacobi' for 'Jacobi-conjugated gradients' (Algorithm 3.4), and 'LOBPCG' for Algorithm 3.5. Each curve represents the sum of the errors in the $n_{e=11}$ leftmost eigenvalues, computed by iterating $m = 14$ vectors, plotted against the iteration number. (Here and in most other plots we have to resort to this 'integral' convergence indicator in order to portray convergence behaviour of 6 algorithms using one plot per test, which hopefully makes it easier for the reader to appreciate differences and similarities and reduces the number of plots sixfold.) We observe that the slope of all curves, save for the one that corresponds to Algorithm 3.2, remains virtually unchanged. Since the range of the x-axis is proportional to n_{xyz}, we conclude that the asymptotic convergence of all algorithms except Algorithm 3.2 is a 'proper' one.

All the subsequent tests employ AMG preconditioning.

The next test exposes the dangers of using a poorly conditioned basis in the trial subspace. Fig. 2 (left) plots the eigenvalue error histories for the Laplacian on $20 \times 20 \times 20$ grid in a unit cube, with eigenvalues computed by LOBPCG with non-orthogonalized basis in $\mathrm{span}\{X, Y, Z\}$; the residual tolerance is set to 10^{-4}. The orthogonalization of the basis used in Algorithm 3.5 helped to prevent instabilities in all tests reported here but has not made this algorithm fully reliable, as can be seen from the right-hand side plot in Fig. 2 (this time the residual tolerance is 10^{-8} and basis orthogonalization *is* used); apparently, more thorough orthogonalization is needed (cf. [12]). We stress that the choice of a particular orthogonalization scheme used in Algorithm 3.5 is motivated primarily by its relatively low computational expenses (compared to e.g. the Gram–Schmidt procedure with re-orthogonalization), and the fact that it proved to be sufficient to prevent instabilities in all tests presented here save for the one with excessive accuracy mentioned above. We note also that no instabilities have ever been observed in other block algorithms tested here, even when the orthogonalization of the basis was disabled. Still, it is not difficult to fail any such algorithm by supplying a Krylov subspace for L as the initial one (gradients of the Rayleigh quotient at Ritz vectors in such trial subspace are all collinear). For this reason, all algorithms tested here do use the orthogonalization as per their descriptions in Section 3.

In the next series of tests we study effects of the eigenvalue clustering. Due to the xyz-symmetry, the spectrum of the discretized Laplacian in a cube contains triple eigenvalues. Hence, small variations of the three sizes produce three-eigenvalue clusters. In Fig. 3 we plot the sum of the errors in the $n_{e=20}$ leftmost eigenvalues, computed by iterating $m = 25$ vectors, against the iteration number for each algorithm applied to the discretized Laplacian in the brick

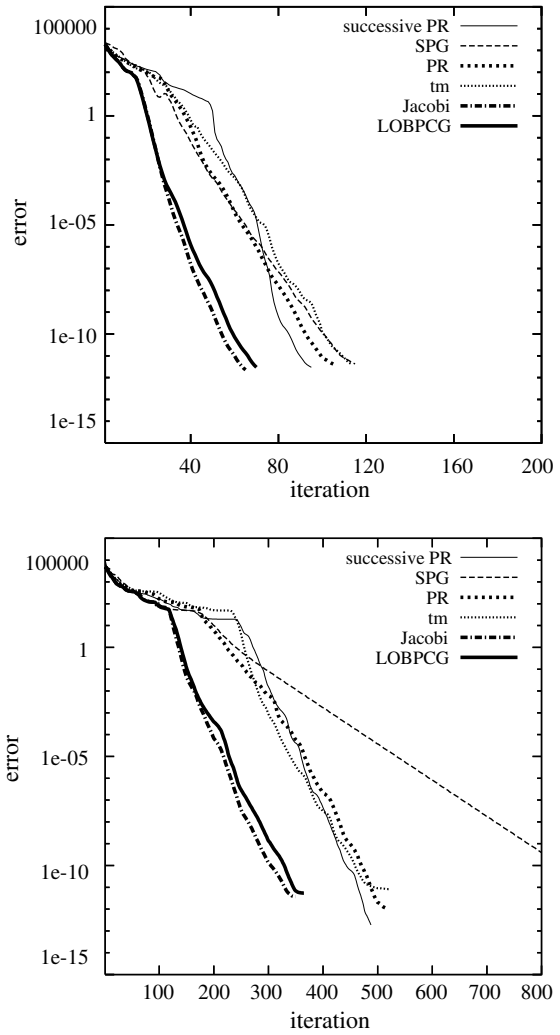

Fig. 1. 'CG-like' behaviour test: $n \times n \times n$ Laplacian, $n = 10,$ and 40.

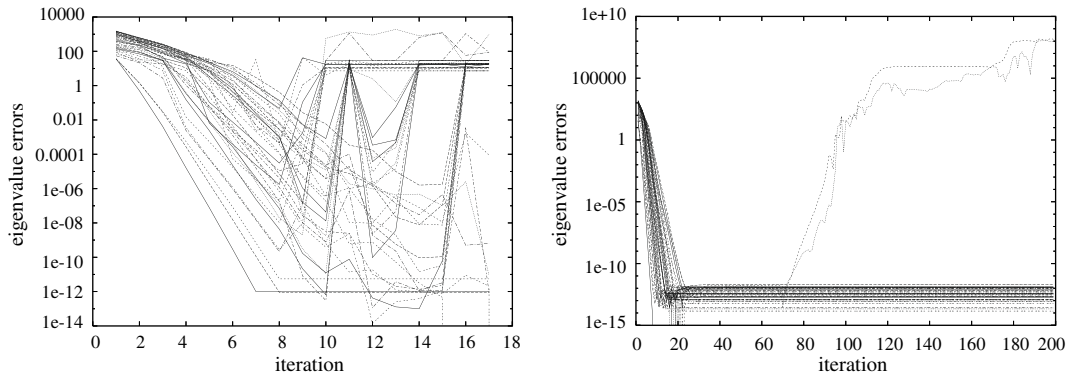

Fig. 2. Instability caused by an 'almost linearly dependent' basis in the trial subspace.

$(0, 1+\delta) \times (0, 1+2\delta) \times (0, 1+3\delta)$. The top left, top right and bottom plots correspond to $\delta = 0.1$, 0.01 and 0.001, respectively. We observe that the convergence of the successive iterations deteriorates in the presence of clustered eigenvalues, which is not surprising and is actually the prime motivation for going for simultaneous iterations: the convergence of the latter is either unaffected at all (cf. Algorithms 3.4 and 3.5) or affected only slightly.

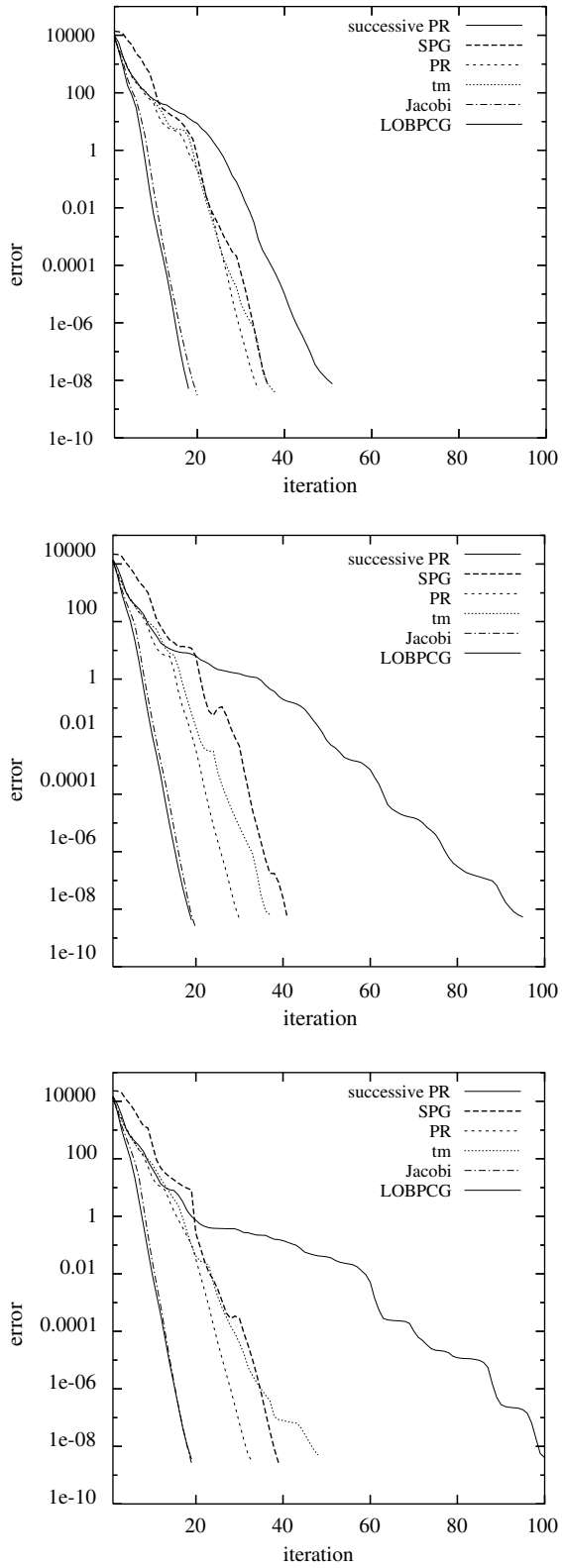

Fig. 3. Cluster robustness test: Laplacian in $(0,\ 1+\delta)\times(0,\ 1+2\delta)\times(0,\ 1+3\delta)$, $\delta = 10^{-1},\ 10^{-2},$ and 10^{-3}.

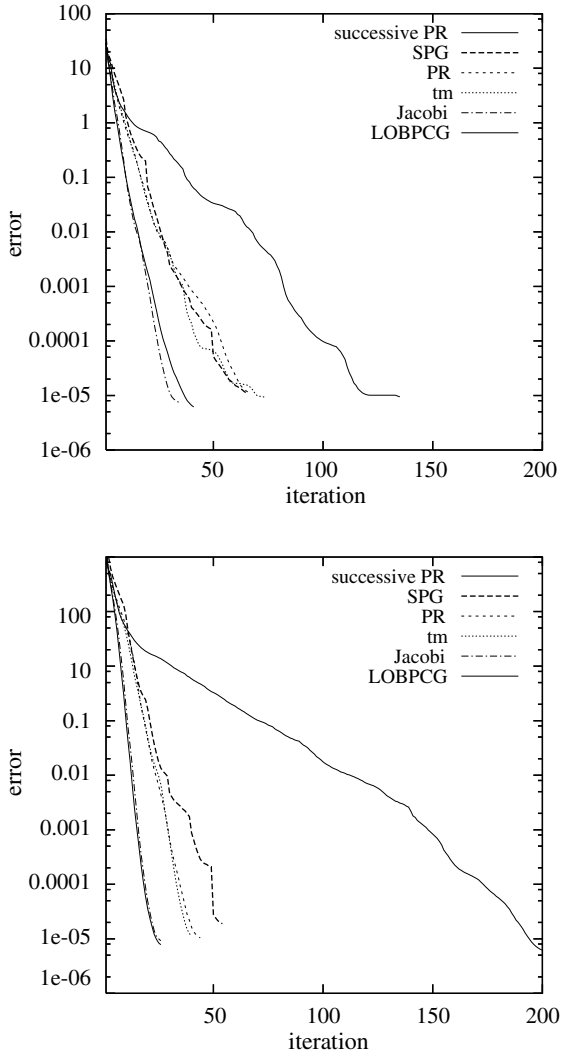

Fig. 4. Si34H36: total eigenvalue error for $n_{e=20}$ (upper) and 80 (lower).

4.2. Standard problem tests

The tests of this section were performed on matrices from PARSEC group of the University of Florida matrix collection maintained by Tim Davis (see http://www.cise.ufl.edu/research/sparse/matrices/). These matrices originate from electronic structure calculations and are among the collection's most formidable ones (sparse Cholesky flop counts reach above 10^{13} for some). At the same time, electronic structure calculations is an area where CG algorithms with AMG preconditioning are especially efficient (to the extent that it was possible to perform the tests of this section on a PC rather than a supercomputer), and are likely to be the best ever.

The problem from which the matrices in question have originated is described in [4] and further details on the PARSEC matrices can be found in [31] (for a wider outlook on electronic structure calculations see [7]). Here we only mention that the problem in question relates to the energy levels of electrons in a molecule, and that each matrix can be split into the sum of two matrices, the first of which represents the discretization of the Laplacian by a high-order finite differences. Owing to this, an AMG preconditioner for the 7-point discretization of the Laplacian on the same grid proved to be fairly efficient in our tests.

In the tests, the residual tolerance was set to 10^{-3}. The exact eigenvalues were not available; hence, in order to estimate the eigenvalue error, we used eigenvalues computed with the residual tolerance 10^{-6} in place of the exact ones. Figs. 4–6 show the behaviour of the total eigenvalue error for matrices `Si34H36` (problem size: 97,569; nonzeros: 5,156,379), `Si41Ge41H72` (problem size: 185,639; nonzeros: 15,011,265) and `Si87H76` (problem size: 240,369; nonzeros: 10,661,631),

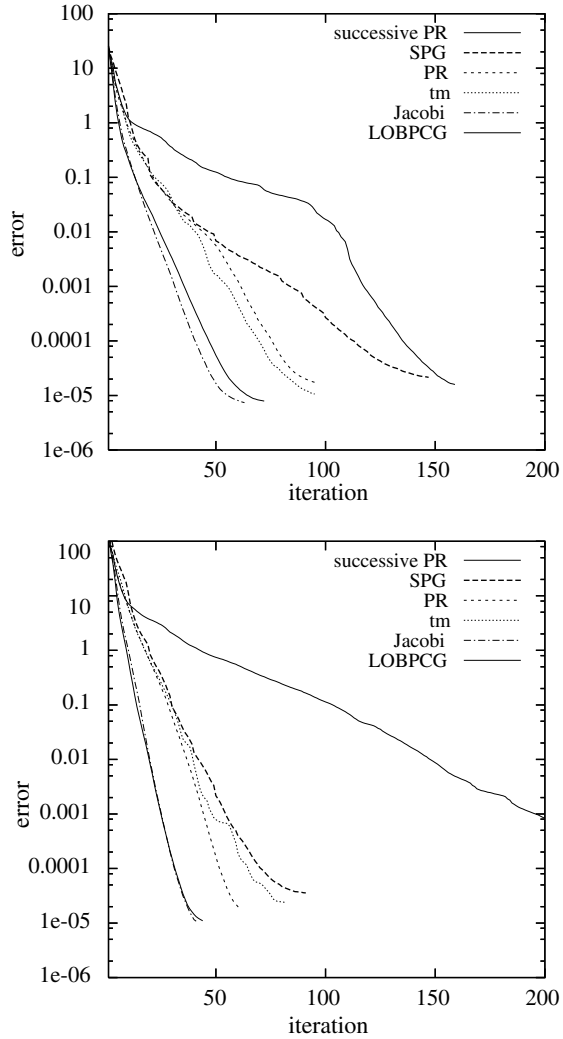

Fig. 5. Si41Ge41H72: total eigenvalue error for $n_{e=20}$ (upper) and 80 (lower).

respectively, and Tables 3–5 show CPU times in seconds.[2] In all tests $m = n_e + 5$, and in all algorithms we opt for storing L-images in order to reduce the number of multiplications by L. We observe that the successive computation of eigenvalues by Polak–Ribiére algorithm is clearly non-competitive. The rest do reasonably well, with LOBPCG and Jacobi algorithms demonstrating superior (and virtually identical) convergence. In terms of the CPU time, the latter algorithm is clearly more efficient due to the smaller size of the trial subspace of the Rayleigh–Ritz procedure. As the number of iterated vectors increase, the block Polak–Ribiére algorithm becomes more competitive due to computationally cheaper conjugation scheme (cf. Table 1).

4.3. Generalized problem tests

In this section we present the results of the tests with the generalized eigenvalue problem for two finite element matrices `qa8fk` and `qa8fm` from Andrew Cunningham group in Tim Davis' matrix collection. The problem in question relates to acoustics – another area where CG with AMG preconditioning is likely to be the best tool for computing eigenvalues.

In the tests, the residual tolerance was set to 10^{-4}, and eigenvalues computed with the residual tolerance 10^{-8} were used in the error estimation as the exact ones.

[2] It should be noted that the number of eigenpairs computed in most tests is less than the number of eigenpairs of practical interest. Nevertheless, the reported results are practically meaningful in the following sense. For problems of a very large size, the block size m may be limited by the available memory, and one may like to resort to a hybrid 'successive-simultaneous' approach whereby eigenpairs are computed in small portions by using a block CG scheme applied in the subspace orthogonal to computed eigenvectors (cf. [1]).

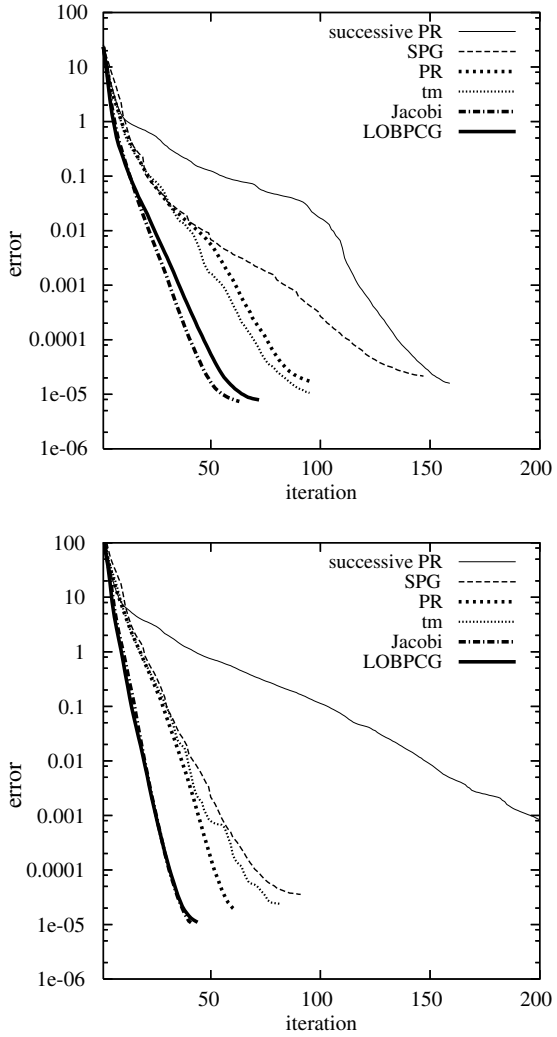

Fig. 6. Si87H76: total eigenvalue error for $n_{e=20}$ (upper) and 80 (lower).

Table 3
CPU times for Si34H36

n_e	Successive PR	PR	SPG	TM	Jacobi	LOBPCG
20	157	97	102	139	77	98
40	395	211	252	275	172	207
60	710	302	398	426	259	325
80	1020	367	476	489	342	436
100	1280	501	684	753	461	603

Table 4
CPU times for Si41Ge41H72

n_e	Successive PR	PR	SPG	TM	Jacobi	LOBPCG
20	502	289	359	329	221	267
40	995	426	470	513	371	496
60	1640	809	907	956	657	850
80	2350	1040	1190	1290	933	1150

Table 5
CPU times for Si87H76

n_e	Successive PR	PR	SPG	TM	Jacobi	LOBPCG
20	420	373	489	418	307	408
40	1080	610	853	786	541	746
60	1830	1080	1610	1290	913	1210
80	2690	1270	1690	1720	1430	1720

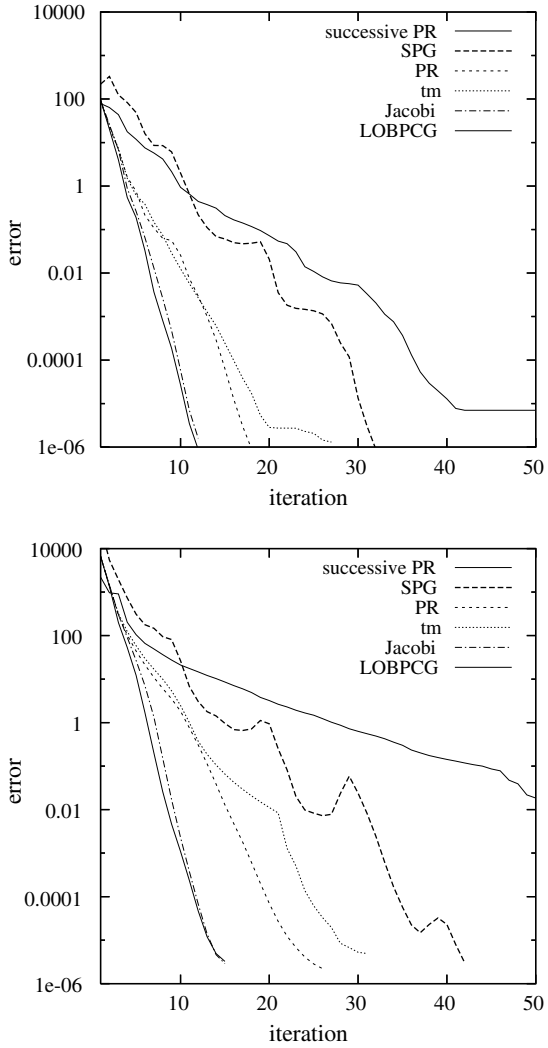

Fig. 7. Acoustics problem: total eigenvalue error for $n_{e=20}$ (upper) and 80 (lower).

Table 6
CPU times for the acoustics problem

n_e	Successive PR	PR	SPG	TM	Jacobi	LOBPCG
20	43.3	31.4	57.5	40.5	26.9	30.6
40	100	64.3	126	82.5	62.6	75.5
60	208	123	231	151	118	147
80	318	183	355	245	181	228

The two plots in Fig. 7 show the total eigenvalue error for $n_{e=20}$ and 80, respectively, and Table 6 shows the CPU times in seconds for $n_{e=20}$, 40, 60, and 80. In all tests $m = n_e + 5$, and in all algorithms we opt for storing L and M-images in order to reduce the number of multiplications by L and M. We observe pretty much the same convergence behaviour as with PARSEC matrices, save for Sartoretto–Pini–Gambolati algorithm, the convergence behaviour of which is rather erratic.

4.4. Conclusions

(1) A conclusion one can safely draw from the numerical results presented here is that the simultaneous computation of eigenpairs is an approach that is surely worth the effort needed to overcome theoretical and practical difficulties involved, as opposed to the successive computation, which is considerably simpler in both respects but is much less efficient and reliable.
(2) The block CG scheme (24 and 25) is remarkably robust: it works reasonably well for a range of matrices B_i.
(3) Idiosyncrasies of the approaches to using CG for the simultaneous eigenvalue computation discussed in Section 2.2 have visible effects on the performance of the algorithms in focus.A rather erratic convergence behaviour of Algorithm 3.2, in particular, its 'non-CG-like' performance in the first series of Laplacian tests, is likely to be the result of an internal conflict between its two major components, the Rayleigh–Ritz procedure and CG, discussed in Section 2.2.Convergence behaviour of the remaining four algorithms illuminates the importance of 'proper' conjugation of search directions in a block CG algorithm. The trace minimization, block Polak–Ribiére and Jacobi algorithms use the generic block scheme (24) and (25) with the conjugation matrix B_i that is a multiple of identity, a diagonal matrix and a full rectangular matrix, respectively. As we argued in Section 2.2, a trace minimization algorithm is, by its nature, likely to produce approximate eigenpairs that converge at a similar rate that may be adversely affected by poor separation of the eigenvalues of interest from the rest of the spectrum. A comparison with the block Polak–Ribiére algorithm (see Fig. 8) shows that this is indeed the case, as the slopes of all eigenvalue error curves for the Rayleigh–Ritz trace minimization algorithm are close to each other and to the last curves for the block Polak–Ribiére algorithm. As a result, it takes much longer for the leftmost eigenpairs to converge and be removed from the computation by deflation, which affects the CPU times.The performance of the block Polak–Ribiére algorithm is remarkably good in terms of the CPU time. However, this is largely due to a computationally cheap conjugation scheme (cf. Section 3.7). In terms of the convergence rate, the block Polak–Ribiére algorithm is clearly inferior to that of the Jacobi algorithm and LOBPCG; hence the former may become more expensive than the latter two in terms of the CPU time, if the computational cost of the operator(s) and/or the preconditioning is higher than in our tests. (One should bear in mind though that the use of a

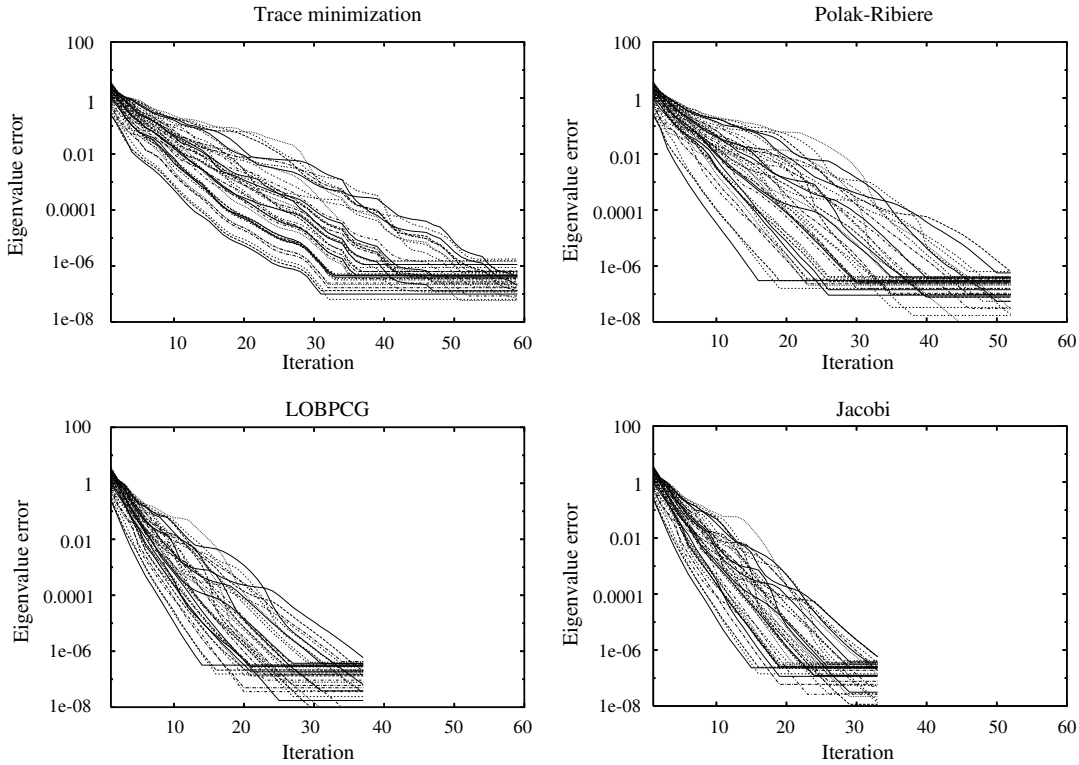

Fig. 8. Si41Ge41H72: eigenvalue error histories for $n_{e=40}$.

more thorough stabilization technique in LOBPCG, such as the one suggested in [12], is likely to increase the computational expenses per iteration, which would compromise the performance of this algorithm.) Finally, we observe that a remarkably similar convergence behaviour of the latter two algorithms suggest that the individual local optimality enjoyed by the search directions of Jacobi scheme is a sufficient one: extra m vectors employed by LOBPCG do not appear to bring about any tangible improvement in convergence (cf. the discussion in Section 2.2).

Appendix A

A.1. The Rayleigh–Ritz procedure

The Rayleigh–Ritz procedure is a procedure for finding approximations to eigenvectors of a given Hermitian operator L in a given subspace $\mathcal{V}$, referred to as the *trial subspace*.

Let m be the dimension of $\mathcal{V}$ and let $v_1, \ldots, v_m$ be a set of linear independent vectors in $\mathcal{V}$. Assuming for a moment that $\mathcal{V}$ contains an eigenvector x_j of L, one can compute x_j is as follows: Since x_j belongs to $\mathcal{V}$, we have $x_j = Vx_j^V$, where V is the matrix with columns $v_1, \ldots, v_m$, and x_j^V is a column vector of height m. In order to find x_j^V, we observe that $Lx_j = LVx_j^V = \lambda_j x_j = \lambda_j Vx_j^V$, where λ_j is the corresponding eigenvalue of L. Multiplying the equation $LVx_j^V = \lambda_j Vx_j^V$ by the adjoint matrix V^* (in order to obtain an m-by-m system) we find that $V^*LVx_j^V = \lambda_j V^*Vx_j^V$, i.e. x_j^V is an eigenvector of the generalized eigenvalue problem

$$L_V x^V = \lambda^V M_V x^V, \quad L_V = V^*LV, \quad M_V = V^*V. \tag{A.1}$$

Now let us consider the case where $\mathcal{V}$ contains an approximation $\tilde{x}_j = V\tilde{x}_j^V$ to x_j rather than x_j. Denote $\tilde{\lambda}_j = \lambda(\tilde{x}_j)$. Since $\tilde{x}_j$ is close to x_j, the norm of the residual $r_j = L\tilde{x}_j - \tilde{\lambda}_j\tilde{x}_j$ is small, and hence so is the norm of $V^*r_j = V^*LV\tilde{x}_j^V - \tilde{\lambda}_j V^*V\tilde{x}_j^V = L_V\tilde{x}_j^V - \tilde{\lambda}_j M_V\tilde{x}_j^V = r_j^V$. The latter vector is the residual vector for $\tilde{\lambda}_j$ and $\tilde{x}_j^V$ in the context of the problem (A.1) and, by well-known results from the eigenvalue approximation theory (see e.g. [21]), the smallness of r_j^V implies that $\tilde{\lambda}_j$ and $\tilde{x}_j^V$ are close to an eigenpair of (A.1). Hence, in order to extract approximations to eigenvectors of L from $\mathcal{V}$, one can compute eigenvalues λ_j^V of the m-by-m eigenvalue problem (A.1), called Ritz values, and corresponding eigenvectors x_j^V, form vectors $\tilde{x}_j = Vx_j^V$, called Ritz vectors, and treat those that have small residuals $L\tilde{x}_j - \lambda_j^V\tilde{x}_j$ as approximate eigenvectors.

It remains to note that the computational procedure just described, known as the Rayleigh–Ritz procedure, can also be applied to the generalized problem $Lx = \lambda Mx$, the only difference being the definition of the Gram matrix M_V as V^*MV.

For further details see e.g. [21].

References

[1] P. Arbenz, U.L. Hetmaniuk, R.B. Lehoucq, R.S. Tuminaro, A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods, Int. J. Numer. Meth. Eng. 64 (2005) 204–236.
[2] L. Bergamaschi, Á. Martínez, G. Pini, Parallel preconditioned conjugate gradient optimization of the Rayleigh quotient for the solution of sparse eigenproblems, Appl. Math. Comput. 175 (2006) 1694–1715.
[3] W.W. Bradbury, R. Fletcher, New iterative methods for solution of the eigenproblem, Numer. Math. 9 (1966) 259–267.
[4] J.R. Chelikowsky, The pseudopotential–density functional method applied to nanostructures, J. Phys. D: Appl. Phys. 33 (2000) R33–R50.
[5] J.W. Daniel, The conjugate gradient method for linear and nonlinear operators, SIAM J. Numer. Anal. 4 (1967) 10–26.
[6] A. Edelman, T.A. Arias, S.T. Smith, The geometry of algorithms with orthogonality constraints, SIAM J. Matrix Anal. Appl. 20 (1998) 303–353.
[7] J.-L. Fattebert, M. Buongiorno Nardelli, Finite difference methods for ab initio electronic structure and quantum transport calculations of nanostructures, in: Handbook of Numerical Analysis, vol. X, Special Volume: Computational Chemistry, Elsevier Science, 2003.
[8] Y.T. Feng, D.R. Owen, Conjugate gradient methods for solving the smallest eigenpair of large symmetric eigenvalue problems, Int. J. Numer. Meth. Eng. 39 (1996) 2209–2229.
[9] R. Fletcher, C.M. Reeves, Function minimization by conjugate gradients, Comput. J. 7 (1964) 149–154.
[10] Z. Fu, E.M. Dowling, Conjugate gradient eigenstructure tracking for adaptive spectral estimation, IEEE Trans. Signal Process. 43 (1995) 1151–1160.
[11] M.R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Nat. Bureau Stand. 49 (1952) 409–436.
[12] U. Hetmaniuk, R. Lehoucq, Basis selection in LOBPCG, J. Comput. Phys. 218 (2006) 324–332.
[13] S.G. Johnson, J.D. Joannopoulos, Block-iterative frequency-domain methods for Maxwell's equations in a planewave basis, Opt. Express 8 (2000) 173–190.
[14] A.V. Knyazev, A preconditioned conjugate gradient method for eigenvalue problems and its implementation in a subspace, Int. Ser. Numer. Math. 96 (1991) 143–154.
[15] A.V. Knyazev, Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method, SIAM J. Sci. Comp. 2 (2001) 517–541.
[16] D.E. Longsine, S.F. McCormick, Simultaneous Rayleigh-quotient minimization methods for $Ax = \lambda Bx$, Linear Algeb. Appl. 34 (1980) 195–234.
[17] J. Nocedal, Conjugate gradient methods and nonlinear optimization, in: L. Adams, J.L. Nazareth (Eds.), Linear and Nonlinear Conjugate Gradient Related Methods, SIAM, 1996.
[18] E.E. Ovtchinnikov, Cluster robustness of preconditioned gradient subspace iteration eigensolvers, Linear Algeb. Appl. 415 (2006) 140–166.
[19] E.E. Ovtchinnikov, Jacobi correction equation, line search and conjugate gradients in Hermitian eigenvalue computation I: computing an extreme eigenvalue, SIAM J. Numer. Anal., in press.
[20] E.E. Ovtchinnikov, Jacobi correction equation, line search and conjugate gradients in Hermitian eigenvalue computation II: computing several extreme eigenvalues, SIAM J. Numer. Anal., in press.
[21] B.N. Parlett, The Symmetric Eigenvalue Problem, Prentice Hall, 1980.
[22] A. Perdon, G. Gambolati, Extreme eigenvalues of large sparse matrices by Rayleigh quotient and modified conjugate gradients, Comp. Meth. Appl. Mech. Eng. 56 (1986) 251–264.
[23] E. Polak, Computational Methods in Optimization: A Unified Approach, Academic Press, 1971.
[24] B.T. Polyak, Introduction to optimization, Optimization Software (1987).

[25] A.H. Sameh, J.A. Wisniewski, A trace minimization algorithm for the generalized eigenvalue problem, SIAM J. Numer. Anal. 19 (1982) 1243–1259.
[26] F. Sartoretto, G. Pini, G. Gambolati, Accelerated simultaneous iterations, J. Comput. Phys. 81 (1989) 53–69.
[27] G.L.G. Sleijpen, H.A. van der Vorst, A Jacobi–Davidson iteration method for linear eigenvalue problems, SIAM J. Matrix Anal. Appl. 17 (1996) 401–425.
[28] K. Stüben, A review of algebraic multigrid, J. Comput. Appl. Math. 128 (2001) 281–309.
[29] I. Takahashi, A note on the conjugate gradient method, Inform. Process. Jpn. 5 (1965) 45–49.
[30] X. Yang, T.K. Sarkar, E. Arvas, A survey of conjugate gradient algorithms for solution of extreme eigenproblems of a symmetric matrix, IEEE Trans. Acoust. Speech Signal Process. 37 (1989) 1550–1556.
[31] Y. Zhou, Y. Saad, A Chebyshev–Davidson, Algorithm for Large Symmetric Eigenvalue Problems, Technical Report, Minnesota Supercomputing Institute, University of Minnesota, 2005.