Facilities Council

A preconditioned block conjugate gradient algorithm for computing extreme eigenpairs of symmetric and Hermitian problems

E. E. Ovtchinnkov and J. K. Reid

May 2010

RAL-TR-2010-019

Library and Information Services
SFTC Rutherford Appleton Laboratory
Harwell Science and Innovation Campus
Didcot
OX11 0QX
UK
Tel: +44 (0)1235 445384
Fax: +44(0)1235 446403
Email: library@rl.ac.uk

ISSN 1358-6254

RAL-TR-2010-019

A preconditioned block conjugate gradient algorithm for computing extreme eigenpairs of symmetric and Hermitian problems

E. E. Ovtchinnikov[1] and J. K. Reid[2]

Abstract

This report describes an algorithm for the efficient computation of several extreme eigenvalues and corresponding eigenvectors of a large-scale standard or generalized real symmetric or complex Hermitian eigenvalue problem. The main features are: (i) a new conjugate gradient scheme specifically designed for eigenvalue computation; (ii) the use of the preconditioning as a cheaper alternative to matrix factorization for large discretized differential problems; (iii) simultaneous computation of several eigenpairs by subspace iteration; and (iv) the use of efficient stopping criteria based on error estimation rather than the residual tolerance.

Keywords: Hermitian eigenvalue problems, preconditioned conjugate gradient method, Fortran 95.

[1]School of Electronics and Computer Science,
University of Westminster,
Watford Road, Northwick Park,
London HA1 3TP.

[2]Computational Science and Engineering Department,
Atlas Centre,
Rutherford Appleton Laboratory,
Oxon OX11 0QX

May 2010.

1 Introduction

This report describes an algorithm for computing several leftmost eigenvalues and corresponding eigenvectors of a real symmetric matrix L of order n or of the generalized problem

$$Lx = \lambda Mx, \tag{1.1}$$

where M is a real symmetric positive-definite matrix of order n. We do not discuss the case where the rightmost eigenvalues are wanted since this is trivially available by working with $-L$. The algorithm in question is implemented by the authors as a code called `HSL_EA19` that is available within the HSL mathematical software library (HSL 2007). There is also a version of the code for Hermitian matrices, but we consider only the real case here, since the changes for the Hermitian case are very straightforward.

Because `HSL_EA19` is designed primarily for the case where n is very large, it does not require that the matrices L and M are available explicitly. It returns to the user from time to time with a request that a given $n \times k$ matrix Y be multiplied by L or M. This reverse communication interface provides great flexibility over the way that the multiplications are performed.

In this report, we enumerate the eigenvalues λ_j and corresponding eigenvectors x_j in ascending order of λ_j, each counted as many times as its multiplicity, and assume that they are M-orthogonalized, $(Mx_j, x_j) = 1$. Here and throughout the report (u, v) stands for $\sum_i u_i v_i$, and $\|u\|$ for $\sqrt{(u,u)}$. Occasionally, we also make use of the notation $(u, v)_M \equiv (Mu, v)$ and $\|u\|_M = \sqrt{(u,u)_M}$. For matrices we use the following two norms: $\|H\|$ is the maximal singular value and $\|H\|_F$ the Frobenius norm (the square root of the trace of $H^T H$).

Our algorithm is based on considering the Rayleigh quotient,

$$\lambda(u) = \frac{(Lu, u)}{(Mu, u)}, \tag{1.2}$$

whose minimum is attained at $u = x_1$ with the value λ_1. To find x_1 and λ_1, we use the method of preconditioned conjugate gradients (PCG), which differs from the standard conjugate gradient method in that the gradient is multiplied by an $n \times n$ matrix K called the *preconditioner*. Available bounds for the convergence factor of PCG iterations are monotonically increasing functions of the ratio of the rightmost to the leftmost positive eigenvalue of $K(L - \lambda_1 M)$ (see Ovtchinnikov, 2006d). If L is positive definite,[1] this is bounded by $cond(KL)\lambda_{(2)}/(\lambda_{(2)} - \lambda_1)$, where $\lambda_{(2)}$ is the second distinct eigenvalue, and $cond(KL)$ denotes the condition number of KL. Thus, if λ_1 and $\lambda_{(2)}$ are well separated, the use of a preconditioner for which $cond(KL)$ is close to 1 ensures good convergence.

To accommodate the computation of several eigensolutions and the case where the eigenvalues are not well separated, we bring into play the so-called *Rayleigh-Ritz procedure*, intimately related to the *minimax principle*

$$\lambda_j = \min_{U_j} \left[\max_{u \in U_j} \lambda(u) \right], \tag{1.3}$$

where the minimum is taken over all j-dimensional subspaces. The minimum is attained when U_j is the space spanned by the first j eigenvectors. If we restrict the choice of U_j to vectors in a subspace of dimension l, called in this context the *trial subspace*, we will obtain over-estimates of the first l eigenvalues, called *Ritz values*. The corresponding approximate eigenvectors are called *Ritz vectors*.

If the columns v_j of the $n \times l$ matrix V span the trial subspace, any vector u in the subspace can be written as Vy, where y is an l-vector. For the Rayleigh quotient, we find

$$\lambda(u) = \frac{(LVy, Vy)}{(MVy, Vy)} = \frac{(V^T LVy, y)}{(V^T MVy, y)} \tag{1.4}$$

which is the Rayleigh quotient for the l-dimensional problem

$$V^T LVy = \lambda V^T MVy. \tag{1.5}$$

[1] If it is not, one just needs to consider an equivalent shifted problem $(L - \sigma M)x = (\lambda - \sigma)M$ with some $\sigma < \lambda_1$.

We use this to obtain approximate eigenvalues of the given problem.

We replace the line search for the new approximate eigenvector x_1^{i+1} employed on each step of PCG by a search for the subspace spanning several approximate eigenvectors x_j^{i+1}, $j = 1, 2, \ldots, m$. We require the user to choose m to be large enough to include all the wanted eigenvalues. It is desirable that it also be such that $\lambda_{m+1} - \lambda_j$ is not small for a wanted λ_j. The subspace search procedure is described in §2.2; for now we just mention that each new approximation x_j^{i+1} is defined as a linear combination of all approximate eigenvectors from the previous iteration and respective search directions satisfying certain optimality conditions. Convergence results of Ovtchinnikov (2006a) show that, for a positive-definite L, the convergence rate for the j-th eigenvalue can be estimated in terms of $cond(KL)\lambda_{m+1}/(\lambda_{m+1} - \lambda_j)$. Hence, the convergence is not adversely affected by the distance between λ_j and any of the other m leftmost eigenvalues: we may say that the subspace iterations are *cluster robust* because the convergence is not adversely affected by a wanted eigenvalue λ_j being in a tight cluster.

The choice of the preconditioner K is the user's responsibility. Again, the matrix itself is not required; instead `HSL_EA19` returns to the user with a request that a given matrix Y be multiplied by K.

The Rayleigh-Ritz procedure employed at each iteration of our algorithm is rather expensive. To reduce the cost, we use a process known as deflation. Any approximate eigenvectors that we deem to be sufficiently accurate are removed from V and the remaining vectors are M-orthogonalized with respect to them. If there are k such vectors, this reduces the size of the eigenproblem by k. Thus it is important that the code recognises convergent eigenpairs early, i.e. that the error estimates used by the code are accurate. In Section 3.2, we briefly discuss error estimates commonly used in practical eigenvalue computation, comment on their limitations, and make the case for the new error estimates and corresponding stopping criteria employed in `HSL_EA19`.

We use the order notation $\mathcal{O}(\ldots)$ for quantities that become small in late iterations and are assumed to be negligible by our code. Formally, if i is the iteration index, $y^i = \mathcal{O}\left(z^i\right)$ means that there is a constant c that depends on the problem but not on i such that $|y^i| \leq c|z^i|$.

The rest of this report is organized as follows. Section 2 concerns the PCG iteration, starting with a subsection on the case with $m = 1$, following with one on the general case, and finally providing a summary of the whole algorithm. Section 3 concerns implementation details, with subsections on the implementation of the Rayleigh-Ritz procedure, the convergence criteria, and deflation (freezing of converged eigenvectors). Finally, in Section 4, we report on numerical experiments.

2 Jacobi-conjugated preconditioned gradient algorithm

2.1 PCG for the Rayleigh quotient minimization

PCG is a minimization algorithm that computes a sequence of approximations x^i to the minimum point of a given functional $\psi(u)$ by the following update scheme:

$$x^{i+1} = x^i - \alpha_i y^i \tag{2.1}$$

$$y^{i+1} = Kg^{i+1} + \beta_i y^i, \tag{2.2}$$

where g^{i+1} is the gradient of $\psi(u)$ at $u = x^{i+1}$ and K is the preconditioner, starting from a given x^0 with $y^0 = Kg^0$. In (2.1), α_i is chosen to minimize $\psi(x^{i+1})$ for this x^i and y^i, so we refer to this step as the *line search*. The role of β_i in (2.2), referred to as the conjugation of search directions, is to improve the new search direction y^{i+1} and thereby achieve convergence acceleration compared to the preconditioned steepest descent (the case $\beta_i = 0$). In the case of a quadratic functional with positive-definite Hessian, it is possible to choose β_i in such a way that each search direction is globally optimal, i.e. each x^{i+1} is not merely the point of local minimum on a line, but also the point of minimum in the hyperplane

$$\{x^0 + y \ : \ y \in \text{span}\{y^0, \ldots, y^i\}\} = \{x^0 + y \ : \ y \in \text{span}\{Kg^0, \ldots, Kg^i\}\}. \tag{2.3}$$

Beyond the quadratic case, the global optimality is no longer possible, and various formulae used for β_i are supported by asymptotic considerations based on the fact that any smooth functional is 'almost quadratic' in the vicinity of the global minimum. The corresponding variations of (2.1)-(2.2), which are mathematically equivalent in the quadratic case, are called CG schemes.

The application of PCG to the Rayleigh quotient minimization meets with additional difficulties as the Hessian $H(u)$ of $\lambda(u)$ is not positive definite. Apart from placing this case beyond the remit of available convergence results, the indefiniteness hinders practical implementation of one of the only two CG schemes for non-quadratic optimization for which the convergence results have been obtained, notably, the one where β_i is such that $(H(x^{i+1})y^{i+1}, y^i) = 0$ ('exact conjugation', or Daniel scheme), i.e. $\beta_i = -(H(x^{i+1})Kg^{i+1}, y^i)/(H(x^{i+1})y^i, y^i)$.

The scale-invariance of the Rayleigh quotient allows a number of modifications[1] in the standard PCG update scheme (2.1)-(2.2): the vector iterate x^i can be normalized, the gradient replaced with the residual vector $r^i = Lx^i - \lambda(x^i)Mx^i$, to which it is collinear, and, most importantly, the line search step (2.1) can be replaced with the Rayleigh-Ritz procedure:

$$x^{i+1} \quad = \quad \mathcal{RR}_1(\,\mathrm{span}\{x^i, y^i\}), \tag{2.4}$$

where we denote by $\mathcal{RR}_k(\mathcal{V})$ the set of M-normalized Ritz vectors of (1.1) in the trial subspace $\mathcal{V}$ corresponding to k leftmost Ritz values. This facilitates the extension of (2.1)-(2.2) to simultaneous computation of eigenpairs, as we show in the next section.

The Rayleigh-Ritz procedure may be employed to automate the choice of β_i as well as α_i by noting that the relationship

$$x^{i+1} = x^i - \alpha_i Kg^i - \alpha_i\beta_{i-1}y^{i-1} = x^i - \alpha_i Kg^i - \alpha_i\beta_{i-1}(x^i - x^{i-1})/\alpha_{i-1}$$

allows us to write, in the quadratic case,

$$x^{i+1} = \arg \min_{v\in\,\mathrm{span}\{Kg^i, x^i - x^{i-1}\}} \psi(x^i + v), \tag{2.5}$$

since the plane $\{x^i + v \,:\, v \in \mathrm{span}\{Kg^i, x^i - x^{i-1}\}\}$ lies in the hyperplane (2.3), and the global optimality implies the local optimality. This approach to computing β_i was suggested by Takahashi (1965), and the resulting update scheme is known as locally optimal. In the case of the Rayleigh quotient minimization, (2.5) is equivalent to[2]

$$x^{i+1} = \mathcal{RR}_1(\,\mathrm{span}\{x^i, Kg^i, x^{i-1}\}). \tag{2.6}$$

This, too, is useful in the next section.

2.2 Block PCG for computing several eigenpairs simultaneously

A straightforward extension of (2.2) and (2.4) to simultaneous computation of m eigenpairs is

$$X^{i+1} \quad = \quad \mathcal{RR}_m(\,\mathrm{span}\{X^i, Y^i\}), \tag{2.7}$$

$$Y^{i+1} \quad = \quad KR^{i+1} + Y^iB_i, \tag{2.8}$$

where $X^i = [x_1^i, \ldots, x_m^i]$, $Y^i = [y_1^i, \ldots, y_m^i]$, $R^i = [r_1^i, \ldots, r_m^i]$, $r_j^i = Lx_j^i - \lambda(x_j^i)Mx_j^i$, and B_i is an $m \times m$ matrix. We note that the conjugation scheme (2.8) was first suggested by Longsine & McCormick (1980), who considered two options: $B_i = \beta_i I_m$, where β_i is a scalar,[3] and I_m is the $m \times m$ identity (i.e. essentially,

[1] With some schemes, these modifications produce equivalent x^{i+1}. In the general case, they are asymptotically equivalent – see Ovtchinnikov (2008a).

[2] We note though that unlike (2.5) for the quadratic $\psi(u)$, (2.6) is not equivalent to other PCG schemes, although some produce asymptotically close x^{i+1} (Ovtchinnikov 2008a).

[3] No explicit formula for β_i was given in the cited paper; various suggestions can be found in Edelman et al. (1998).

a scalar), and $B_i = \mathrm{diag}\{\beta^i_{jj}\}_{j=1}^m$, where β^i_{jj} is computed as in (2.2) for respective r^{i+1}_j and y^i_j or r^i_j, e.g. $\beta^i_{jj} = \|r^{i+1}_j\|^2/\|r^i_j\|^2$ (preconditioning was not used). Both schemes were discarded: the former for not delivering 'a commensurate increase in power' (compared to a hybrid successive-simultaneous scheme they favoured), and the latter for being 'improper extension of Rayleigh quotient minimization by CG', since conjugacy of n-by-m matrices 'is not a well-defined concept'. Both conjugation schemes were subsequently used by other authors (being, apparently, re-invented, since references to Longsine & McCormick (1980) were absent): $B_i = \beta_i I_m$ with various scalars β_i was used by Edelman et al. (1998) (with a different generalization of the line search step in place of (2.7), although (2.7) is also mentioned as an option), and the mentioned diagonal B_i by Arbenz et al. (2005).

The numerical tests of Ovtchinnikov (2008c) demonstrate that the block PCG scheme (2.7)-(2.8) works remarkably well for a range of conjugation matrices B_i, including the two mentioned in the previous paragraph. They demonstrate also that the choice of B_i does affect the convergence, and thus the worries of Longsine & McCormick (1980) regarding the lack of well-defined concept of block conjugation are not unfounded. It is observed by Ovtchinnikov (2008c) that the trial subspace in (2.7) is contained in the trial subspace of the locally optimal block PCG (LOBPCG) method of Knyazev (2001) – yet another block PCG method whereby

$$X^{i+1} = \mathcal{RR}_m(\,\mathrm{span}\{X^i, KR^i, X^{i-1}\}), \tag{2.9}$$

which is the generalization of (2.6). Consequently, the new approximations computed by (2.7) cannot be better than those computed by (2.9). In fact, the numerical tests of Ovtchinnikov (2008c) demonstrate that the convergence of (2.7)-(2.8) with the scalar or diagonal B_i is considerably worse than that of LOBPCG, although the scheme with a diagonal B_i outperformed LOBPCG in terms of CPU time, owing to a less computationally expensive update. At the same time, the tests of Ovtchinnikov (2008c) demonstrate that it is possible for a block scheme (2.7)-(2.8) to match LOBPCG in terms of the convergence rate, and therefore outperform the latter in terms of the CPU time (owing to the use of a smaller trial subspace) by a proper choice of B_i. Also, it is more economical with storage.

Based on the theoretical convergence analysis of (2.7)-(2.8), Ovtchinnikov (2008b) has made a suggestion for B_i that enjoys the optimality property that B_i be such that each search direction y^i_j is 'individually' asymptotically optimal in the sense that no other choice of B_i can substantially reduce $\tilde\lambda^{i+1}_j = \lambda(\tilde x^{i+1}_j) = \min_\tau \lambda(x^i_j - \tau y^i_j)$. It is shown by Ovtchinnikov (2008b) that the sum of $\tilde\lambda^{i+1}_j$ is asymptotically equal to the sum of Ritz values in the subspace spanning $\tilde x^{i+1}_j$. The latter subspace, called 'core' subspace in the cited paper is contained both in the trial subspace of (2.7) and that of (2.9), and a remarkable similarity of convergence of these two block PCG schemes suggests that the individual line searches implicitly present in these schemes play major role in the error reduction, which motivates the described 'individual' optimality approach.

In order to obtain a compact formulae for the entries of B_i, Ovtchinnikov (2008b) observes that the update scheme (2.7)-(2.8) can be equivalently[4] reformulated as follows:

$$[X^{i+1}, Z^{i+1}] = \mathcal{RR}_{m_i}(\,\mathrm{span}\{X^i, Y^i\}), \quad m_i = \dim\,\mathrm{span}\{X^i, Y^i\}, \tag{2.10}$$

$$Y^{i+1} = KR^{i+1} + Z^{i+1}B_i. \tag{2.11}$$

The dimension m_i of $\mathrm{span}\{X^i, Y^i\}$ may be less than $2m$ because of the possible linear dependence of vectors $x^i_1, \ldots, y^i_m$. If it is $m + m'$, then Z^{i+1} is an $m \times m'$ matrix. The entries $\beta^{(i)}_{kj}$ of B_i are shown to be given by an explicit formula

$$\beta^{(i)}_{kj} = \frac{((\lambda(x^{i+1}_j)M - L)Kr^{i+1}_j, z^{i+1}_k)}{\lambda(z^i_k) - \lambda(x^i_j)} \tag{2.12}$$

(if $\lambda(z^i_k) = \lambda(x^i_j)$, which we have not observed in practice, we set $\beta^{(i)}_{kj} = 0$).

[4]Excluding the case where the angle between $\mathrm{span}\{X^i\}$ and $\mathrm{span}\{X^{i+1}\}$ is $\pi/2$.

The main theoretical tool used by Ovtchinnikov (2008b) in the derivation of this new block PCG update scheme is the so-called Jacobi orthogonal complement correction equation of Sleijpen & van der Vorst (1996) and its generalization by Brandts (2003); hence, (2.10)-(2.12) is called Jacobi-conjugate preconditioned gradient (JCPG) method by Ovtchinnikov (2008a), Ovtchinnikov (2008b), and Ovtchinnikov (2008c).

2.3 Algorithm summary

Setting aside, for the time being, various implementation issues, such as the stopping criteria and deflation and detecting almost linearly dependent vectors, the JCPG algoritm can be summarized as follows.

- **Input:** linearly independent vectors $v_1, \ldots, v_m$.
- **Output:** approximate eigenvalues $\lambda_1, \ldots, \lambda_m$ and eigenvectors $x_1, \ldots, x_m$.

1. **Initialization**
   (a) Compute $\widehat{L} = V^T L V$ and $\widehat{M} = V^T M V$, where $V = [v_1, \ldots, v_m]$.
   (b) Compute eigenvectors $\widehat{x}_j$ of the generalized problem $\widehat{L}\widehat{x} = \widehat{\lambda}\widehat{M}\widehat{x}$ normalized by $\|\widehat{x}_j\|_M = 1$.
   (c) Compute $x_j = V\widehat{x}_j$. Set $X = [x_1, \ldots, x_m]$.
2. **Main CG loop (to be repeated until convergence)**
   (a) Compute $\lambda_j = \lambda(x_j)$ and $r_j = Lx_j - \lambda_j M x_j$, $j = 1, \ldots, m$.
   (b) Compute $y_j = Kr_j$, $j = 1, \ldots, m$. Set $Y = [y_1, \ldots, y_m]$.
   (c) If not the first iteration, then
      i. Compute (using (2.12)) the $m' \times m$ matrix B with entries β_{kj} given by

$$\beta_{kj} = \begin{cases} \dfrac{\lambda_j (My_j, z_k) - (Ly_j, z_k)}{\lambda_{m+k} - \lambda_j}, & \lambda_{m+k} > \lambda_j, \\ 0, & \lambda_{m+k} = \lambda_j, \end{cases} \qquad k = 1, \ldots, m', \quad j = 1, \ldots, m,$$

      where m', $Z = [z_1, \ldots, z_{m'}]$ and λ_j ($j = 1, \ldots m + m'$) are computed at steps (2e), (2h) and (2g), respectively, on the previous iteration.
      ii. Update $Y = Y + ZB$.
   (d) M-orthogonalize Y to X, i.e. ensure $(MY, X) = 0$, discarding any y_j that form too small an angle with $\mathrm{span}\{X\}$.
   (e) M-orthogonalize Y, i.e. ensure for $j \neq k$ $(My_j, y_k) = 0$, discarding any almost linearly dependent y_j. Set m' to be the number of remaining columns in Y.
   (f) Compute the matrices

$$\widehat{L} = \begin{bmatrix} X^T L X & X^T L Y \\ Y^T L X & Y^T L Y \end{bmatrix}, \quad \widehat{M} = \begin{bmatrix} X^T M X & X^T M Y \\ Y^T M X & Y^T M Y \end{bmatrix}. \tag{2.13}$$

   (g) Compute eigenvectors $\widehat{q}_j$ of the eigenproblem $\widehat{L}\widehat{q}_j = \lambda_j \widehat{M}\widehat{q}_j$, $j = 1 \ldots, m + m'$, normalized by $\|\widehat{q}_j\|_{\widehat{M}} = 1$. Set $\widehat{Q} = [\widehat{q}_1, \ldots, \widehat{q}_{m+m'}]$.
   (h) Update

$$[X, Z] = [X, Y]\widehat{Q}. \tag{2.14}$$

In the next section, we give a more detailed description of steps 2d and 2e of this algorithm.

We note that it is a good idea to set m to a slightly greater value than the number of wanted eigenpairs because this improves the convergence to rightmost wanted eigenpairs (cf. the convergence estimate of Ovtchinnikov (2006a) mentioned in the Introduction) and facilitates the error estimation (cf. §3.2.1).

3 The implementation of JCPG

3.1 The implementation of the Rayleigh-Ritz procedure

The straightforward implementation of the Rayleigh–Ritz procedure outlined in §1 may fail if the columns of V are 'almost' linearly dependent, as can be seen from the following simple example.

Let $M = I$ and let L be the matrix obtained by the standard 5-point finite difference discretization of Dirichlet boundary value problem for the two-dimensional Laplace operator in a rectangle. If the sizes of the rectangle are a_x and a_y, and we use (n_x+1)-by-(n_y+1) mesh, then L is $n_y \times n_y$ block tridiagonal matrix with $n_x \times n_x$ blocks. The diagonal blocks are tridiagonal matrices with $2(h_x^{-2} + h_y^{-2})$ on the main diagonal, where $h_x = a_x/(n_x + 1)$ and $h_y = a_y/(n_y + 1)$, and $-h_x^{-2}$ on the two adjacent diagonals, and the non-zero off-diagonal blocs are $n_x \times n_x$ identity matrices multiplied by h_y^{-2}. Now consider the Rayleigh–Ritz procedure in the trial subspace spanned by $v_j = L^{j-1}v_1/\|L^{j-1}v_1\|$, $j = 2, 3, \ldots$, where $v_1 = [1, \ldots, 1]^T$. As j increases, v_j approach the eigenvector x_1 of L, and hence $v_1, \ldots, v_l$ become 'almost' linearly dependent for l large enough. To see how this affects the Rayleigh–Ritz procedure in practice, let as set $a_x = a_y = 1$ and $n_x = n_y = 11$. The Ritz values for $l = 13$ and 14 produced by the LAPACK eigensolver subroutine `DSYGV` applied to (1.5) are given in Table 3.1.

Table 3.1: Instability of the Rayleigh-Ritz procedure with ill-conditioned basis

$l = 13$	$l = 14$
	-655.97607
19.62671	19.62673
94.15728	94.17159
162.95198	171.62060
219.74592	225.10133
273.91863	333.86539
372.57971	390.01678
486.08180	502.69981
569.65790	589.44838
712.04108	721.51902
851.42629	856.97597
968.34324	973.67487
1051.49771	1054.87070
1130.45281	1131.68478

Clearly, the first Ritz value for $l = 14$ is a spurious one that is there due to the round-off errors: in exact arithmetic all Ritz values would be positive due to the positive definiteness of L. We observe also that all remaining computed Ritz values for $l = 14$ are greater than those for $l = 13$. Hence, not only a spurious eigenvalue has appeared, but also the accuracy in the remaining eigenvalues has deteriorated (exact eigenvalues are smaller than the Ritz values, in particular, those for $l = 13$, due to the minimax principle).

To see what exactly went wrong, let us have a look at the way `DSYGV` handles the problem at hand. First, Cholesky factorization is applied to $M_V = V^T MV$ i.e. an upper-triangular matrix U is computed such that $M_V = U^T U$. The variable change $y = Ux$ transforms the generalized eigenvalue problem $L_V x = \lambda M_V x$, where $L_V = V^T LV$, into the standard one for $L_{VU^{-1}} = U^{-T} L_V U^{-1}$, which is then solved by the subroutine `DSYEV`. The computation of $L_{VU^{-1}}$ involves the solution of systems $Uv = u$. In the case at hand, U is poorly conditioned: near-linear-dependence of $v_1, v_2, \ldots, v_{14}$ translates into near-singularity of M_V and further into near-singularity of U. Consequently, the solution of $Uv = u$ is inaccurate, and the computed $L_{VU^{-1}}$ deviates significantly from the exact one.

It should be stressed that the appearance of a spurious eigenvalue is not `DSYGV`'s fault. $\|Vx\|$ may be

very small compared to $\|x\|$ because the columns of V are 'almost' linearly dependent. In this case, (x, λ) will be an approximate eigenpair for any reasonable value of λ because $||L_V x - \lambda M_V x||$ will be small. As a result, one may end up with spurious eigenvalues.

An obvious way to avoid problems caused by near linear dependence is to orthogonalize the columns of V using the Gram-Schmidt procedure. In `HSL_EA19` we opt for a different approach that allows our code to benefit from highly optimized BLAS matrix multiplication subroutines rather than use considerably less efficient matrix-vector multiplication subroutines on which the Gram-Schmidt procedure is based.

Our approach is based on the observation that the difference between the eigenvalue $\tilde{\lambda}_i$ computed by LAPACK subroutines for solving the generalized symmetric eigenvalue problem $\widehat{L}\widehat{q} = \widehat{\lambda}\widehat{M}\widehat{q}$ and the exact one $\widehat{\lambda}_i$ is (see LAPACK User Guide) approximately

$$\texttt{EERRBD}(i) = \left(\|\widehat{M}^{-1}\|\|\widehat{L}\| + |\tilde{\lambda}_i|\kappa\right)\epsilon,$$

where κ is the (spectral) condition number of $\widehat{M}$ (the ratio of the largest to smallest eigenvalue) and ϵ is the machine accuracy (the value returned by the Fortran 95 intrinsic function `EPSILON`). The first term inside the brackets is at most $\kappa \max_i |\widehat{\lambda}_i|$.[5] Hence,

$$\max_i \texttt{EERRBD}(i) \leq \epsilon\kappa \max_i \left(|\widehat{\lambda}_i| + |\tilde{\lambda}_i|\right) \approx 2\epsilon\kappa \max_i |\widehat{\lambda}_i|,$$

and in order to keep the effect of the round-off errors under control, we need to make sure that the value κ is not too large. In `HSL_EA19` we employ the following procedure that ensures that $\kappa \leq \kappa_0$, where $\kappa_0 = 10^3$ in the single precision package and $\kappa_0 = 10^6$ in the double precision one. In what follows, we use the notation of §2.3.

1. We compute the matrix $\widehat{M}$ given in (2.13) and compute its eigenvalues. If the ratio of the largest to smallest eigenvalue is not greater than κ_0, the remaining steps are not performed.

2. We M-orthogonalize the columns of Y to those of X, i.e. update $Y := Y - XM_{X,Y}$, where $M_{X,Y} = X^T MY$ was computed on the previous step, and compute $M_{X,Y}$ again.

3. We discard a column y_j of the new Y if the angle it forms with $\mathrm{span}(X)$ deviates significantly from a right angle, which it would have been in exact arithmetic. The precise criterion for discarding is as follows: we calculate the norm of $X^T My_j$, which is equal to the M-norm of the projection of y_j onto $\mathrm{span}(X)$, and if $\|X^T My_j\| \geq \theta\|y_j\|_M$ where $0 < \theta < 1$ ($\theta = 0.5$ in the current version), this search direction is discarded.

4. We re-orthogonalize y_j to X if $\|X^T My_j\| \geq \frac{\gamma_0}{\sqrt{m}}\|y_j\|_M$, where $\gamma_0 = \frac{\kappa_0 - 1}{\kappa_0 + 1}$.

5. We M-normalize Y, compute the matrix $M_Y = Y^T MY$ and its eigenpairs $\{\nu_j, q_j\}$ and update $Y := YQ$, where Q is the matrix whose columns are the normalized eigenvectors q_j enumerated in descending order of corresponding eigenvalues. We discard any new y_j with M-norm too close to ϵ (less than 10ϵ in the current version), and M-normalize the rest.

6. We compute the matrix $\widehat{M}$ given by (2.13) and, if necessary, discard some of the last rows and columns to arrive at a sub-matrix with the condition number not exceeding κ_0. This is done as follows (below m' stands for the number of remaining search direction: note that this number may decrease while this step is performed).

   `do while (`m' `> 1)`

   Apply the estimates of Lemma 1 of §A (with $A = X^T MX$, $B = Y^T MY$ and $C = Y^T MX$) to compute the upper bound for the condition number of current $\widehat{M}$. If this bound is less or equal to κ_0, then exit this loop.

[5] We have $\widehat{L} = \widehat{M}^{1/2}\widehat{M}^{-1/2}\widehat{L}\widehat{M}^{-1/2}\widehat{M}^{1/2}$ and $\|\widehat{M}^{1/2}\| = \|\widehat{M}\|^{1/2}$. The norm of $\widehat{M}^{-1/2}\widehat{L}\widehat{M}^{-1/2}$ is the absolute maximum of eigenvalues of $\widehat{L}\widehat{x} = \widehat{\lambda}\widehat{M}\widehat{x}$. The condition number of $\widehat{M}$ is equal to $\|\widehat{M}^{-1}\|\|\widehat{M}\|$.

```
    do i = 2, m'
```
Use Lemma 1 to compute a lower bound for the i-th largest eigenvalue of $\widehat{M}$ and an upper bound for the i-th smallest eigenvalue of $\widehat{M}$, in order to find a lower bound $\tilde{\kappa}_i$ for the condition number κ_i that $\widehat{M}$ would have if its last $i-1$ rows and columns are removed. If $\tilde{\kappa}_i \leq \kappa_0$, remove the last $i-1$ rows and columns from $\widehat{M}$, decrement m' by $i-1$ and exit this loop.

```
    end do
end do
```

The rationale for this procedure is as follows.

Assume for a moment that we use exact arithmetic. Since after step 5 the updated Y satisfy $Y^T M Y = Q^T M_Y Q$, and the latter matrix is a diagonal one with ν_j on the diagonal, the columns of the new Y are orthogonal, and their M-norms are $\sqrt{\nu_j}$. Since we have also orthogonalized them to X at step 2, the matrix $\widehat{M}$ is a diagonal one. Hence, after M-normalization of non-zero y_j, and discarding zero ones, we end up with $\widehat{M}$ that has the condition number $\kappa = 1$.

In inexact arithmetic, step 2 generally does not make the columns of Y M-orthogonal to those of X. If a given y_j happens to be very close to $\mathrm{span}(X)$, then the corresponding column of $XM_{X,Y}$ is very close to y_j, and subtracting one from the other is strongly affected by round-off errors. In fact, if the norm of the result is too close to ϵ, the updated y_j may be linearly dependent on the columns of X despite the orthogonalization, and the re-orthogonalization may fail. For this reason, in step 3 we discard the new y_j if the angle it forms with $\mathrm{span}(X)$ deviates substantially from a right angle. We note that the criterion for discarding is rather soft because we want to keep as many vectors as possible; discarding vectors generally affects the convergence, and actually did so in some tests.

Skipping step 4 for the moment, let us discuss the remaining two steps. If we denote $Y' = YQ$, then $Y = Y'Q^T$, and hence each y_j before update can be represented as the sum $y_j^k + y_j^d$, where y_j^k is the linear combination of the columns of Y' that we keep and y_j^d of those we discard. The square of the M-norm of y_j^d is equal to the sum of $\nu_i q_{ji}^2$, where q_{ij} are the entries of Q, the sum being taken over i's corresponding to discarded columns of Y'. Since q_j are normalized, $\|y_j^d\|_M^2$ is not greater than the maximum of ν_i taken over discarded columns of Y'. Each ν_i is equal to the square of the norm of the respective column of Y', and hence the M-norm of y_j^d is not greater than the maximal M-norm of a discarded direction. Thus, the criterion for discarding used by step 4 ensures that the M-norm of y_j^d is close to the round-off error level, and hence discarding columns of Y after update in step 4 is equivalent to introducing perturbations $\mathcal{O}(\epsilon)$ into the original search directions.

In inexact arithmetic, step 5 does not produce M-orthogonal Y, and the condition number of $\widehat{M}$ may deviate significantly from 1. In order to ensure that $\kappa \leq \kappa_0$, in step 6 we discard some directions corresponding to smallest ν_i (as we have just shown, this is equivalent to introducing perturbations $\mathcal{O}\left(\sqrt{\nu_j}\right)$ into original directions until either we are left with just one or the condition number of the block of $\widehat{M}$ corresponding to remaining directions is guaranteed to be not greater than κ_0. The meaning of the outer loop of step 6 is fairly obvious, and the rationale for the inner loop is as follows. The i-th smallest eigenvalue of $\widehat{M}$ is not less than the smallest eigenvalue of the matrix $\widehat{M}_i$ obtained by removing the last $i-1$ rows and columns from $\widehat{M}$ by virtue of the fact that the eigenvalues of $\widehat{M}_i$ are Ritz values of $\widehat{M}$ in the subspace of column vectors with last $i-1$ components equal to zero. For the same reason, the i-th largest eigenvalue of $\widehat{M}$ is not greater than the largest eigenvalue of $\widehat{M}_i$. Therefore, the ratio $\tilde{\kappa}_i$ computed in the inner loop is a lower bound for the condition number κ_i of $\widehat{M}_i$. If the former exceeds κ_0, then so does the latter, and we need to remove at least $i-1$ last rows and columns from $\widehat{M}$. The case $i = 1$ is excluded because if $\tilde{\kappa}_i < \kappa_0$, then we do not now whether κ_i is less or greater than κ_0. By starting with $i = 2$ rather than $i = 1$ we make sure that after we exit from it to continue the outer loop the dimension of $\widehat{M}$ has decreased, and thus the procedure terminates after a finite number of steps.

Let us show now that thanks to step 4 we are bound to have $\kappa \leq \kappa_0$, modulo a term $\mathcal{O}\left(\|X^T M X - I\|\right)$, in the case where the procedure used in step 6 terminates because there is only one direction left, in which

case it does not guarantee the bound $\kappa \leq \kappa_0$.

Step 3 ensures that the vectors y_j re-orthogonalized in step 4 form a substantial angle with $\mathrm{span}(X)$, which implies that after the re-orthogonalization the new y_j will be orthogonal to X to machine accuracy, and hence the inequality we check in step 3 cannot hold for them.[6] Thus, after the re-orthogonalization, we have $\|X^T M y_j\| < (\gamma_0/\sqrt{m})\|y_j\|_M$ for all remaining y_j.

Let us denote by q_{j1} the components of the eigenvector q_1 of M_Y corresponding to the largest eigenvalue and normalized so that $q_1^T M_Y q_1 = 1$. To avoid confusion, in this paragraph we denote by y_j the columns of Y before step 5; thus the direction we are left with after step 6 is Yq_1. We have:

$$\|X^T MY q_1\|^2 \leq \left(\sum_j \|X^T M y_j\||q_{j1}|\right)^2 \leq \frac{\gamma_0^2}{m}\left(\sum_j \|y_j\|_M |q_{j1}|\right)^2 \leq \gamma_0^2 \sum_j q_{j1}^2 = \gamma_0^2 q_1^T q_1 = \frac{\gamma_0^2}{\nu_1}$$

Since the diagonal entries of M_Y are units, the largest eigenvalue ν_1 of M_Y is not less than 1. Hence, $\|X^T MY q_1\| \leq \gamma_0$.

Assuming for a moment $X^T MX = I$, it is not difficult to see (e.g. via (1.33), which become equations in the case at hand) that the smallest eigenvalue of $\widehat{M}$ is $1-\gamma$ and the largest is $1+\gamma$, where $\gamma = \|X^T MY q_1\| \leq \gamma_0$. Thus, the condition number of $\widehat{M}$ is not greater than $(1+\gamma_0)/(1-\gamma_0) = \kappa_0$. In practice, the entries of $X^T MX$ deviate from those of the identity matrix, and the estimates of Lemma 1 adjusted accordingly imply that κ may exceed κ_0 by a quantity of the order $\mathcal{O}\left(\|X^T MX - I\|\right)$, which is why in step 6 we enforce the condition $m' > 0$ rather than rely on this step's terminating with $m' > 0$.

It remains to note that the main computational expenses of the described six-step procedure, assuming no directions are discarded in step 5, and no re-orthogonalization needed, are dense matrix-matrix multiplications involving $9nm^2$ scalar multiplications in total: $3nm^2$ in step 1, $2nm^2$ in step 2, $2nm^2$ in step 5 and $2nm^2$ in step 6. In comparison, the Gram-Schmidt procedure without re-orthogonalization would have to use matrix-by-vector multiplications (except for step 1), which are considerably less efficient than matrix-by-matrix multiplications.[7] We note also that the re-orthogonalization in step 4 is an extremely rare occasion[8] – the angle between the search directions and $\mathrm{span}\{X\}$ is usually substantial (if no preconditioning is used, then search directions are simply orthogonal to $\mathrm{span}\{X\}$).

3.2 Convergence criteria

In `HSL_EA19`, an eigenpair is considered to have converged if one or more of the following conditions, as selected by the user, are satisfied:

1. The Euclidean norm of the corresponding residual vector is not greater than $\max\{\epsilon_{r,abs}, \epsilon_{r,rel}\|Lx\|\}$, where $\epsilon_{r,abs}$ and $\epsilon_{r,rel}$ are the prescribed absolute and relative tolerance respectively.

2. The estimated eigenvalue error is not greater than $\max\{\epsilon_{\lambda,abs}, \epsilon_{\lambda,rel}\delta_\lambda\}$, where $\epsilon_{\lambda,abs}$ and $\epsilon_{\lambda,rel}$ are the prescribed absolute and relative tolerance respectively, and δ_λ is an estimated average distance between eigenvalues.[9]

3. The estimated eigenvector error is not greater than than the prescribed relative error tolerance.

[6] Unless, of course, κ_0 is too large, which is not what we have, or m is too large. Regarding the latter case, we note that m would have to approach ϵ^{-2} for our assumption about re-orthogonalized y_j to cease to be reasonable. Problems of such size are obviously beyond the capabilities of eithr `HSL_EA19` or any other known algorithm.

[7] Computing the product $X^T Y$ for two 10^5-by-10^2 dense matrices X and Y using a matrix-by-vector multiplication subroutine `DGEMV` from BLAS in a loop is about 3 times slower than a single call to a matrix-by-matrix subroutine `DGEMM` on a Dell Precision 490 Workstation with Intel Math Kernel Library 6.1. With Intel Math Kernel Library 10.1 it is about 9 times slower.

[8] It was only observed in our tests when $m > n/2$, which is unlikely to occur in normal `HSL_EA19` use since this is intended for problems with modest m and large n.

[9] In the current version, for $m = 1$ the average distance is estimated as twice the Rayleigh quotient on a pseudo-random vector divided by the problem size; for $m > 1$, we take the minimal of the two values: the one just mentioned and the average distance between computed approximate eigenvalues.

The first condition is fairly straightforward and common in eigenvalue computation. The other two require *a posteriori* error estimation, which is rather more complicated, and stopping criteria of such kind do not appear to be used by available eigensolvers. In this section we describe the two approaches to error estimation employed by `HSL_EA19`.

3.2.1 Residual error estimation

Let us start with the eigenvalue error estimation for the standard eigenvalue problem ($M = I$). In this section we assume all approximate eigenvectors to be normalized in the standard Euclidean norm.

By the well-known Krylov-Weinstein estimate, for any approximate eigenpair $\{\tilde{\lambda}, \tilde{x}\}$ there exists an eigenvalue λ of A such that

$$|\tilde{\lambda} - \lambda| \leq \|A\tilde{x} - \tilde{\lambda}\tilde{x}\| \tag{3.15}$$

(see e.g. Parlett (1980)). The expression in the right-hand side is the norm of the residual vector for $\{\tilde{\lambda}, \tilde{x}\}$, which motivates the use of the first stopping criteria listed at the beginning of this section for a normalized approximate eigenvector $\tilde{x}$ of the standard eigenproblem.

An alternative estimate by Kato and Temple shows that (3.15) is often far too pessimistic: for i such that $\lambda_i \leq \tilde{\lambda} = \lambda(\tilde{x}) < \lambda_{i+1}$ one has

$$\tilde{\lambda} - \lambda_i \leq \frac{\|A\tilde{x} - \tilde{\lambda}\tilde{x}\|^2}{\lambda_{i+1} - \tilde{\lambda}}. \tag{3.16}$$

We observe that the upper bound for the relative error $(\tilde{\lambda} - \lambda_i)/(\lambda_{i+1} - \tilde{\lambda})$ provided by (3.16) is exactly the square of that provided by (3.15) for the same $\tilde{\lambda}$ and $\tilde{x}$. Hence, if an eigensolver relies on (3.15) for the eigenvalue error estimation, it is likely to perform at least twice as many iterations as actually needed in the case where the user is interested in eigenvalue accuracy only.

In the case where more that one approximate eigenvector is available, one can benefit from still tighter estimates produced by Lehmann's method (Lehmann 1963). The latter computes, for a given value σ (to which we refer here as Lehmann pole) that is not a Ritz value, a set of values $\cdots \leq \tau_{-2} \leq \tau_{-1} < \sigma < \tau_1 \leq \tau_2 \leq \cdots$ such that each semi-closed interval $[\tau_{-k}, \sigma)$ or $(\sigma, \tau_k]$ contains not less than k exact eigenvalues. Lehmann intervals can be used for the eigenvalue error estimation in the following manner. Let $\tilde{\lambda}_j$ and $\tilde{x}_j$, $j = 1, \ldots, m$, be the Ritz values and normalized Ritz vectors, and $r_j = L\tilde{x}_j - \tilde{\lambda}_j\tilde{x}_j$ the corresponding residual vectors. Assuming

$$\tilde{\lambda}_{k-1} < \lambda_k, \tag{3.17}$$

for some $k \leq m+1$, one places the Lehmann pole in between $\tilde{\lambda}_{k-1}$ and λ_k, strictly right from the former, in which case the eigenvalue inclusion property of Lehmann intervals implies $\tau_{j-k} \leq \lambda_j$, $j = 1, \ldots, k-1$. Since $\lambda_j \leq \tilde{\lambda}_j$ by the minimax principle, the difference $\tilde{\lambda}_j - \tau_{j-k}$ is the upper bound for the error $\tilde{\lambda}_j - \lambda_j \geq 0$. It remains to note that by Ovtchinnikov (2009), $\tau_{j-k} \geq \nu_j^\sigma$, where ν_j^σ are the eigenvalues of the matrix

$$L_\sigma = \tilde{\Lambda} - S_\sigma^T S_\sigma, \tag{3.18}$$

where $\tilde{\Lambda}$ is the $k \times k$ diagonal matrix with $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_k$ on the diagonal, and the columns of S_σ are $(\sigma - \tilde{\lambda}_j)^{-1/2} r_j$. Thus, $\tilde{\lambda}_j - \lambda_j \leq \tilde{\lambda}_j - \nu_j^\sigma$.

A major practical problem with the Lehmann error estimation scheme is that λ_k is generally not available and hence it is not possible to find a proper place for the Lehmann pole, as we may be unable to ensure that $\sigma \leq \lambda_k$. However, in the framework of a subspace iteration algorithm such as the one implemented by `HSL_EA19`, one usually deals with approximate eigenvalues $\tilde{\lambda}_j$ that are close to respective[10]

[10] It is theoretically possible that some exact eigenvalues 'go missing', i.e. are not approximated by any $\tilde{\lambda}_j$. In practical eigenvalue computation by subspace iterations this is all but impossible because the convergence to non-consecutive eigenvalues is unstable and is invariably destroyed either by round-off errors or preconditioning or both – cf. more detailed arguments of Ovtchinnikov (2009).

exact eigenvalues λ_j, at least for some $j = 1, \ldots, k$. Hence, one can employ a lower bound for some λ_k as the Lehmann pole, provided that it is not too close to $\tilde{\lambda}_{k-1}$. The analysis of the properties of Lehmann bounds by Ovtchinnikov (2009) produced the following algorithm, which is employed, with slight modifications, by `HSL_EA19` when the user opts for the residual eigenvalue error estimation.

1. The largest k is identified for which $\tilde{\lambda}_k - \tilde{\lambda}_{k-1} \geq \|R_k\| + \epsilon_\lambda$, where $R_k = [r_1, \ldots, r_{k-1}]$ and ϵ_λ is the absolute eigenvalue error tolerance; if none found, k is set to 1.

2. If $k > 1$, then $\sigma = \tilde{\lambda}_k - \epsilon_\lambda$ is used as the Lehmann pole for estimating the eigenvalue errors in $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_{k-1}$ via eigenvalues of (3.18).

3. For $\tilde{\lambda}_k, \ldots, \tilde{\lambda}_{n_w}$, Kahan's inequality $|\tilde{\lambda}_j - \lambda_j| \leq \|R^k\|$ is used, where $R^k = [r_k, \ldots, r_{n_w}]$ and $n_w \leq m$ is the number of wanted eigenpairs.

4. Once all wanted eigenpairs have converged, the Lehmann bounds are updated using all the latest approximate eigenvectors and residual vectors. If all eigenpairs satisfy the convergence criteria, the iterations are terminated; otherwise, they are restarted using current approximate eigenvectors as initial vectors.

The use of ϵ_λ in place of $\|R^k\|$, which is, strictly speaking, required on step 1, makes it possible to have 'rough' estimates for leftmost eigenvalues before those further to the right have small enough residuals. The last step of the algorithm replaces 'rough' estimates with 'true' ones (since now $\|R^k\| \leq \epsilon_\lambda$), and the iterations may be restarted if the former are too rough. We note that since Kahan's inequality generally overestimates the actual errors considerably, the convergence to last few eigenpairs may take large number of iterations. Finally, we observe that $\|R_k\| \leq \|R_k\|_F \equiv \sqrt{\mathrm{Tr} R_k^T R_k}$ and $\|R^k\| \leq \sqrt{m-k+1}\max_{k \leq j \leq m} \|r_j\|$. In order to simplify the calculation, in `HSL_EA19`, we use Frobenius norm of R_k on step 1 and the norm of r_j as the bound for $\tilde{\lambda}_j - \lambda_j$ on step 3.

It is shown by Ovtchinnikov (2009) that for a well-separated simple Ritz value $\tilde{\lambda}_j$, the bound $\tilde{\lambda}_j - \nu_j^\sigma$ for the error $\tilde{\lambda}_j - \lambda_j$ has the following asymptotics:

$$\tilde{\lambda}_j - \nu_j^\sigma = \frac{\|r_j\|^2}{\sigma - \tilde{\lambda}_j} + \mathcal{O}\left(\|R_k\|^4\right). \tag{3.19}$$

This asymptotic relationship allows one to reach beyond the machine accuracy in the following sense. The relative accuracy in eigenvalues is roughly the square of that for eigenvectors owing to the fact that the Rayleigh quotient is stationary on eigenvectors. Hence, one can compute eigenvectors to single precision and then compute Rayleigh quotients for them in double precision, thus achieving nearly double the accuracy in respective eigenvalues (cf. the second test example in the specification for the package `HSL_EA19` that employs this technique). If one takes advantage of such an approach, then Lehmann bounds are only helpful until the differences between them and corresponding Ritz values are sufficiently above the single machine accuracy. In `HSL_EA19`, the main term in the right-hand side of (3.19) is used in place of Lehmann bounds when this is not the case.

Turning to the eigenvector errors, we observe that the results of Ovtchinnikov (2006b) imply that for any indices $p \leq q < k$ such that $\lambda_q < \lambda_{q+1}$ and either $p = 1$ or $\lambda_{p-1} < \lambda_p$ the following estimate for the angle between $\tilde{x}_j$, $p \leq j \leq q$, and the invariant subspace $\mathcal{X}_{p,q} = \mathrm{span}\{x_p, \ldots, x_q\}$ is valid:

$$\sin^2\{\tilde{x}_j; \mathcal{X}_{p,q}\} \leq \left(1 + \mathcal{O}\left(\|R_k\|^2\right)\right) \frac{\|r_j\|^2}{(\sigma - \tilde{\lambda}_j)^2}, \quad j = p, \ldots, q \tag{3.20}$$

(see the proof of theorem 4 in the cited paper). We stress that the asymptotically small term in (3.20) does not depend on the distances between $\lambda_p, \ldots, \lambda_q$. In `HSL_EA19`, the square root of the main term in the right-hand side of (3.20) is used as the error estimate for the sine of the angle between $\tilde{x}_j$ and $\mathcal{X}_{p,q}$. No estimates are computed for $\tilde{x}_k, \ldots, \tilde{x}_m$.

3.2.2 Kinematic error estimation

Numerical tests with the error estimation scheme described in the previous section have amply demonstrated that Lehmann bounds are usually too pessimistic: in fact, they usually overestimate actual eigenvalue errors by 2-3 orders of magnitude. Another drawback of the residual error estimation is the fact that in order to apply it to the generalized problem $Lx = \lambda Mx$, one needs to compute the norms $\|r_j\|_{M^{-1}} = \sqrt{(M^{-1}r_j, r_j)}$ rather than the standard Euclidean norms. To avoid solving auxiliary systems with the matrix M, one could, in principle, compute the minimal eigenvalue η_1 of this matrix and use the estimate $\|r_j\|_{M^{-1}} \le \eta_1^{-1/2}\|r_j\|$. However, the computation of η_1 is generally not an easy task, and the mentioned estimate may be too inaccurate in the case of a finite-element problem on a highly non-uniform mesh.

Extensive numerical testing of the algorithm implemented by `HSL_EA19` has revealed that one can obtain much more accurate error estimates by opting for a novel approach described in this section, which we tentatively call *kinematic* error estimation, and which employs reasoning similar to that of Reid (1964), page 23. Consider a trivial observation: if one could have q_{ij} such that

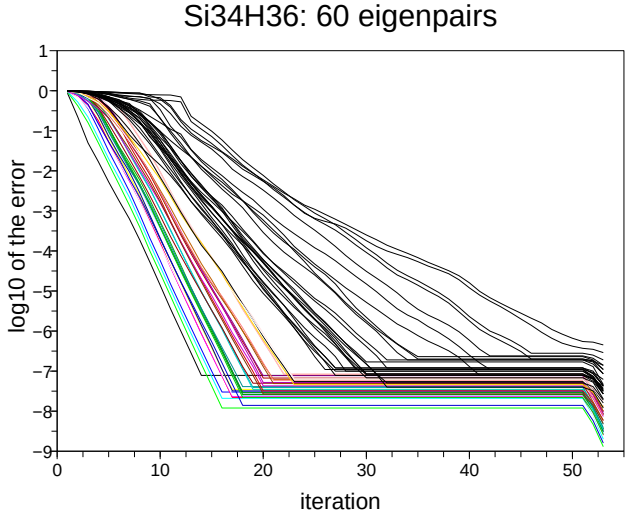
$$\lambda_j^i - \lambda_j \le q_{ij}(\lambda_j^{i-1} - \lambda_j)$$

(we use the notation of §2.2; i stands for the iteration number), then

$$\lambda_j^i - \lambda_j \le \frac{q_{ij}}{1 - q_{ij}}(\lambda_j^{i-1} - \lambda_j^i). \tag{3.21}$$

Now, for q_{ij} one could turn to the convergence estimates of Ovtchinnikov (2008b); however, they involve quantities that cannot be easily computed. The key observation that unlocks the potential of the trivial estimate (3.21) is that in practical computation with the block CG schemes described in §2.2, the eigenvalue errors usually behave as shown in Fig. 3.1, which plots the errors in 60 leftmost eigenvalues of the matrix `si34h36` from Tim Davis' collection computed by JCPG. Quite remarkably, every single convergence curve becomes very close to an oblique straight line[11] (in semi-logarithmic scale used in the figure) in the course of iterations, which means that all actual relative eigenvalue reduction ratios $q_{ij}^a = (\lambda_j^i - \lambda_j)/(\lambda_j^{i-1} - \lambda_j)$ tend to constants q_j that do not depend on the iteration number.

Figure 3.1: Error histories for eigenvalues of `si34h36`.



[11] The algorithm does not change approximate eigenvalues deemed to have converged, hence the horizontal lines.

To exploit the noted feature of the eigenvalue errors, it remains to figure out how to compute the constants q_j based on the approximate eigenvalue's history. One might start by noticing that if $q_{ij}^a \to q_j$ as $i \to \infty$, then q_j is the limit of the geometrical average $q_j^{(l)}$ of $q_{ij}, i = i_0+1, \ldots, l$, the so-called asymptotic convergence factor:

$$q_j = \lim_{l\to\infty} q_j^{(l)}, \quad q_j^{(l)} = \left(\prod_{i=i_0+1}^{l} q_{ij}^a\right)^{\frac{1}{l-i_0}} = \left(\prod_{i=i_0+1}^{l} \frac{\lambda_j^i - \lambda_j}{\lambda_j^{i-1} - \lambda_j}\right)^{\frac{1}{l-i_0}} = \left(\frac{\lambda_j^l - \lambda_j}{\lambda_j^{i_0} - \lambda_j}\right)^{\frac{1}{l-i_0}}, \quad i_0 \geq 0. \tag{3.22}$$

Next, let us show that that λ_j can be replaced with its approximation λ_j^l. If we denote

$$\tilde{q}_j^{(l)} = \left(\frac{\lambda_j^l - \lambda_j^{l+1}}{\lambda_j^{i_0} - \lambda_j^{l+1}}\right)^{\frac{1}{l-i_0}}, \tag{3.23}$$

then

$$q_j^{(l)} \geq \tilde{q}_j^{(l)} \geq (1 - q_{lj}^a)^{\frac{1}{l-i_0}} q_j^{(l)}.$$

Hence, as l increases, $\tilde{q}_j^{(l)} \to q_j^{(l)} \to q_j$.

Thus, one arrives at the following simple error estimate:

$$\lambda_j^i - \lambda_j \approx \frac{q_{ij}}{1 - q_{ij}}(\lambda_j^{i-1} - \lambda_j^i), \quad q_{ij} = \left(\frac{\lambda_j^{i-1} - \lambda_j^i}{\lambda_j^{i_0} - \lambda_j^i}\right)^{\frac{1}{i-i_0-1}}, \quad i = i_0+2, i_0+3, \ldots \tag{3.24}$$

It should be emphasized that (3.24) *does not provide the upper bound for the eigenvalue error*, only an approximation to the latter, which can be larger or smaller than the actual error. In order to reduce the risk of underestimation, in `HSL_EA19` the value q is set to the maximum of the two values: the one given in (3.24) and

$$q_{ij}^* = \max_{i_- \leq l \leq i_+} \frac{\lambda_j^l - \lambda_j^i}{\lambda_j^{l-1} - \lambda_j^i}, \tag{3.25}$$

where (i_-, i_+) is a 'trust' region. We observe that the ratio in the right-hand side of (3.25) is an approximation to the actual error ratio $(\lambda_j^l - \lambda_j)/(\lambda_j^{l-1} - \lambda_j)$. The trust region is chosen to satisfy the following requirements:

1. The last iteration index i_+ in the trust region is such that the approximate error $\lambda_j^l - \lambda_j^i$ for $l = i_+$ is sufficiently larger than $\lambda_j^{i-1} - \lambda_j^i$ (10 times larger in the current version), so that the ratio in (3.25) is close to the ratio of the actual eigenvalue errors before and after l-th iteration.
2. The first iteration index i_- in the trust region is such that $\lambda_j^{l-1} - \lambda_j^l > \lambda_j^l - \lambda_j^{l+1}$ for $l = i_- + 1, \ldots, i_+ - 1$, whereas $\lambda_j^{i_- -1} - \lambda_j^{i_-} \leq \lambda_j^{i_-} - \lambda_j^{i_-+1}$. This ensures that the convergence curve is not too 'wobbly' in the trust region, i.e. the semi-logarithmic plot of the error always slopes downwards.

For added reliability, the error estimation scheme just described is applied only if $\lambda_j^{i-1} - \lambda_j^i$ is less than the required absolute eigenvalue error; otherwise the j-th approximate eigenpair is not considered to be close enough to the exact one.

Once the eigenvalue errors have been estimated, one can obtain estimates for the eigenvector errors using the following estimate that is proved in §A.2: assuming that $\lambda_l^i < \lambda_{l+1}$ for some $l < k \leq m$, one has

$$\sin^2\{\mathcal{X}_l^i; \mathcal{X}_l\} \leq \frac{1 + \epsilon_{i,k}}{\lambda_{k+1} - \lambda_l} \sum_{j=1}^{l} (\lambda_j^i - \lambda_j), \tag{3.26}$$

where $\mathcal{X}_l = \text{span}\{x_1, \ldots, x_l\}$, $\mathcal{X}_l^i = \text{span}\{x_1^i, \ldots, x_l^i\}$ (if $l = k$, then $\epsilon_{i,k} = 0$) and

$$\epsilon_{i,k} = 4\frac{\lambda_{k+1} + \lambda_{l+1} - 2\lambda_1}{(\lambda_{l+1} - \lambda_l^i)^2} \sum_{j=l+1}^{k} (\lambda_j^i - \lambda_j). \tag{3.27}$$

We remind the reader that the sine of the angle (or the gap) between two subspaces is defined as

$$\sin\{\mathcal{U};\mathcal{V}\} = \max\left\{\max_{u\in\mathcal{U}}\sin\{u;\mathcal{V}\},\ \max_{v\in\mathcal{V}}\sin\{v;\mathcal{U}\}\right\}$$

(if $\dim\mathcal{U} = \dim\mathcal{V}$ then the two maximums inside the curly brackets coincide; otherwise $\sin\{\mathcal{U};\mathcal{V}\} = 1$). We emphasize that (3.26) is formulated for a group of eigenvalues rather than for a single one in order to be efficiently applicable to clustered eigenvalues: for $l = 1$ the term $\epsilon_{i,k}$ is not small if λ_1 is in a cluster unless the error in the first k eigenvalues is small compared to $(\lambda_2 - \lambda_1^i)^2/(\lambda_k - \lambda_1)$. In `HSL_EA19`, we essentially apply (3.26) to a group of eigenvalues separated from the rest of the spectrum by at least the estimated average distance δ_λ between eigenvalues. This allows us to replace the denominator in (3.27) with δ_λ^2. The numerator is estimated by $2(\lambda_{k+1} - \lambda_1)$, and for k we select the largest index such that error estimates for $\lambda_1, \ldots, \lambda_k$ are already available, and $\epsilon_{i,k}$ computed based on these estimates does not exceed 0.8.

The kinematic error estimation relies on the accurate computation of eigenvalue decrements $\lambda_j^{i-1} - \lambda_j^i$ that are based on the Rayleigh quotients of the corresponding eigenvectors. If this difference is small, the directly computed value will have poor relative accuracy (or none). The following approach is employed in `HSL_EA19` for estimating decrements that are smaller than the LAPACK eigensolver's error.

Let Q be the matrix whose columns are the eigenvectors of the problem $Y^TLYq = \nu Y^TMYq$. If we are left with $m' \le m$ search directions after the orthogonalization procedure described in §3.1, then $Q^TY^TMYQ = I_{m'}$ (the m'-by-m' identity matrix) and $N = Q^TY^TLYQ$ is diagonal with $\nu_1, \ldots, \nu_{m'}$ on the diagonal. Since we can equivalently use the columns of $[X, YQ]$ as the basis for the Rayleigh-Ritz procedure on i-th iteration instead of those of $[X, Y]$, the new approximate eigenvalues λ_j^i are the m leftmost eigenvalues of the problem

$$\widehat{L}\widehat{x}^i = \lambda^i\widehat{M}\widehat{x}^i, \quad \widehat{L} = \begin{bmatrix} \Lambda & H^T \\ H & N \end{bmatrix}, \quad \widehat{M} = \begin{bmatrix} I_m & G^T \\ G & I_{m'} \end{bmatrix}, \tag{3.28}$$

where $\Lambda = \operatorname{diag}\{\lambda_1^{i-1}, \ldots, \lambda_m^{i-1}\}$, $H = [h_1, \ldots, h_m] = Q^TY^TLX$, and $G = [g_1, \ldots, g_m] = Q^TY^TMX$. We observe that the m leftmost columns of the $(m + m')$-by-$(m + m')$ identity matrix are the Ritz vectors of this problem (in the trial subspace of column vectors with zero last m' components) corresponding to the Ritz values $\lambda_1^{i-1}, \ldots, \lambda_m^{i-1}$, and the corresponding residual vectors form $(m + m')$-by-m' matrix whose upper m-by-m block is zero and lower m'-by-m block is formed by vectors $s_j = h_j - \lambda_j^{i-1}g_j$. An asymptotic equation of Ovtchinnikov (2008b) relating the differences between eigenvalues and Ritz values to the respective residuals implies that

$$\lambda_j^{i-1} - \lambda_j^i \approx s_j^T(N - \lambda_j^{i-1}I_{m'})^{-1}s_j = \sum_{l=1}^{m'}\frac{|s_{lj}|^2}{\nu_l - \lambda_j^{i-1}}, \quad s_j = [s_{1j}, \ldots, s_{m'j}]^T,$$

and `HSL_EA19` uses these approximations to the decrements $\lambda_j^{i-1} - \lambda_j^i$ when the latter approach the round-off error level.

Let us now have a look at how the kinematic error estimation works in practice compared to the residual error estimation. Table 3.2 compares the kinematically estimated errors for ten leftmost eigenvalues of the matrix `Si87H76` from Tim Davis' collection with 'actual errors' obtained by computing eigenvalues to much smaller error tolerance (all results are computed using double precision). We observe that most of the estimated errors are quite close to the 'actual' ones: the ratio of estimated to 'actual' error is less than 4 (less than 2 for all but the last two eigenvalues). At the same time, the error in the first eigenvalue is slightly underestimated.

For comparison, Table 3.3 presents estimated and 'actual' errors for ten leftmost eigenvalues of the same matrix but computed by `HSL_EA19` with the error estimation option switched to the residual estimates based on Lehmann bounds (in both cases the absolute eigenvalue error tolerance was set to 0.001). We observe considerable overestimation of the eigenvalue errors; furthermore, some estimates are confusing.

Table 3.2: Kinematic error estimates for eigenvalues of $Si_{87}H_{76}$ vs. actual errors.

eigenvalue	estimated error	actual error
-1.196375	$7.29 \cdot 10^{-6}$	$7.69 \cdot 10^{-6}$
-1.162052	$1.06 \cdot 10^{-5}$	$7.91 \cdot 10^{-6}$
-1.162048	$1.66 \cdot 10^{-5}$	$1.19 \cdot 10^{-5}$
-1.162042	$2.52 \cdot 10^{-5}$	$1.82 \cdot 10^{-5}$
-1.123360	$2.26 \cdot 10^{-5}$	$1.71 \cdot 10^{-5}$
-1.123329	$9.39 \cdot 10^{-5}$	$4.75 \cdot 10^{-5}$
-1.123267	$2.08 \cdot 10^{-4}$	$1.10 \cdot 10^{-4}$
-1.121457	$5.87 \cdot 10^{-6}$	$2.93 \cdot 10^{-6}$
-1.121381	$1.79 \cdot 10^{-4}$	$7.97 \cdot 10^{-5}$
-1.108059	$5.33 \cdot 10^{-4}$	$1.28 \cdot 10^{-4}$

Table 3.3: Lehmann error estimates for eigenvalues of $Si_{87}H_{76}$ vs. actual errors.

eigenvalue	estimated error	actual error
-1.196380	$6.79 \cdot 10^{-5}$	$2.30 \cdot 10^{-6}$
-1.162060	$3.45 \cdot 10^{-4}$	$4.64 \cdot 10^{-8}$
-1.162057	$1.29 \cdot 10^{-4}$	$3.21 \cdot 10^{-6}$
-1.162051	$1.09 \cdot 10^{-5}$	$9.24 \cdot 10^{-6}$
-1.123377	$5.18 \cdot 10^{-4}$	$9.89 \cdot 10^{-8}$
-1.123376	$1.70 \cdot 10^{-5}$	$2.70 \cdot 10^{-7}$
-1.123370	$1.40 \cdot 10^{-5}$	$6.59 \cdot 10^{-6}$
-1.121460	$1.18 \cdot 10^{-4}$	$2.50 \cdot 10^{-9}$
-1.121458	$2.14 \cdot 10^{-6}$	$1.97 \cdot 10^{-6}$
-1.108188	$7.41 \cdot 10^{-6}$	$8.73 \cdot 10^{-8}$

Consider, for example, the estimates for the three-eigenvalue cluster formed by second, third and fourth leftmost eigenvalues. The estimated error for the leftmost eigenvalue of this cluster exceeds the actual error by four orders of magnitude, whereas the one for the rightmost eigenvalue is remarkably close to the actual error, which appears to contradict what the first sentence of §3.2.2 says. However, one should bear in mind that the pairing of Lehmann bounds and Ritz values based simply on their indices can be misleading. In the case at hand, the Lehmann bound $-1.162051 - 1.09 \cdot 10^{-5} \approx -1.162061$ apparently corresponds to the second leftmost Ritz value -1.162060 rather than the fourth one. Thus, the estimated error for the second leftmost eigenvalue is actually about 10^{-6} and those for the third and fourth are of the order 10^{-4}. The same reasoning applies to the two-eigenvalue cluster formed by eighth and ninth eigenvalues, with the 'correct' estimate of about 10^{-7} for the eighth eigenvalue and about 10^{-4} for the ninth. Even after such a correction, most estimated errors are about two orders of magnitude above the actual ones.

In the two tests, the superior accuracy of the kinematic error estimation resulted in about 15% shorter computation time compared to the run with the Lehmann-based estimation, which represents a tangible but hardly impressive saving. However, when `HSL_EA19` was applied to `shipsec1` matrix from the same collection, the Lehmann-based run failed, as the residuals stopped decreasing after few iterations because of the round-off errors and the large norm of the matrix. It should be stressed that this was not the JCPG fault, as `HSL_EA19` actually computed requested eigenvalues to high accuracy after exceeding the imposed 200 iterations limit, but the employed error estimation scheme completely failed to notice that the eigenpairs have already converged long ago. With the error estimation switched to the kinematic one, just 12 iterations produced ten leftmost eigenvalues to about 10^{-5} absolute accuracy (about 10^{-4} relative one).

3.3 Deflation

An iterative scheme for computing the leftmost eigenpair, such as PCG scheme (2.1)-(2.2), can be easily adapted for the computation of several leftmost eigenpairs with the help of technique known as *deflation*. Having computed the leftmost eigenpair, one modifies the problem in such a way that the second leftmost eigenpair of the original problem becomes the leftmost eigenpair of the modified, or *deflated* problem (cf. a similar technique for the computation of the roots of a polynomial). The most commonly used deflation technique is to minimize the Rayleigh quotient functional $\lambda(u)$ in the subspace M-orthogonal to the computed eigenvectors; if the computed eigenvectors are exact, the remaining ones are in this subspace. If we denote by X_k the matrix whose columns are the k computed approximate eigenvectors, then new vector iterates x^i_{k+1} satisfy $P_k x^i_{k+1} = 0$, where $P_k = X_k X_k^T M$ is the M-orthogonal projector onto $\mathcal{X}_k = \mathrm{span}\{X_k\}$ (we note that the columns of X_k are M-orthonormal, i.e. $X_k^T M X_k$ is k-by-k identity, owing to the way the approximate eigenvectors are computed). Hence, the Rayleigh quotient restricted to the M-orthogonal complement $\mathcal{X}_k^\perp$ to $\mathcal{X}_k$ can be written down as

$$\lambda(u) = \frac{(Lu,u)}{(Mu,u)} = \frac{(L(I-P_k)u,(I-P_k)u)}{(M(I-P_k)u,(I-P_k)u)} = \frac{((I-P_k)^T L(I-P_k)u,u)}{((I-P_k)^T M(I-P_k)u,u)} \equiv \lambda_\perp(u) \text{ for } u \in \mathcal{X}_k^\perp. \quad (3.29)$$

The gradient of $\lambda_\perp(u)$ is collinear to the respective 'deflated' residual vector

$$r_\perp(u) = (I-P_k)^T L(I-P_k)u - \lambda_\perp(u)(I-P_k)^T L(I-P_k)u = (I-P_k)r(u) \text{ for } u \in \mathcal{X}_k^\perp, \quad (3.30)$$

where $r(u) = Lu - \lambda(u)Mu$. We stress that should one rely on residual norms in stopping criteria, the norm of $r_\perp(x^i_{k+1})$ should be used, as it yields meaningful bounds for the distance from x^i_{k+1} to the point of minimum of $\lambda_\perp(u)$ in $\mathcal{X}_k^\perp$ and for the difference between $\lambda_\perp(x^i_{k+1})$ and the respective minimal value, whereas the norm of $r(u)$ is generally non-zero at the point of minimum of $\lambda_\perp(u)$, and may remain above the required tolerance after any number of iterations.[12] It remains to note that, in practical terms, as far

[12] This kind of the stopping criterion's failure to detect the convergence was actually observed in tests where the norm of $r(x^{i+1})$ was used instead of that of $r_\perp(x^{i+1})$.

as PCG scheme (2.1)-(2.2) is concerned, the deflation amounts to using $(I - P_k)y^i$ as the search direction in (2.1) and $r_\perp(x^{i+1})$ in place of g^{i+1} for the conjugation (2.2).[13]

In HSL_EA19, once an eigenpair satisfies the accuracy requirements, the corresponding new search direction is not computed, and the approximate eigenvector no longer participates in the Rayleigh-Ritz procedure. All the remaining search directions are M-orthogonalized to converged eigenvectors so that the non-converged ones, which are orthogonal to the removed ones at the time of deflation, keep so at subsequent iterations. It should be stressed that the accuracy of the remaining eigenpairs is affected by the accuracy of those that are removed. In order to eliminate possible adverse effects of inaccurate removed 'converged' eigenvectors, once all wanted eigenpairs are deemed converged, we apply the Rayleigh-Ritz procedure in the subspace containing all approximate eigenvectors. If the residual-based stopping criteria are used, then error estimates are computed immediately. If the kinematic error estimation is employed, then two iterations are performed and errors are estimated based on the last eigenvalue decrements and previously estimated convergence factors q_{ij}. If all eigenvectors satisfy the stopping criteria, then the iterations are terminated, otherwise the whole procedure is repeated. In other words, the iterations are continued until all wanted eigenpairs are Ritz pairs that satisfy the accuracy requirements simultaneously.

4 Numerical experiments

In this section, we present the results of a set of tests with the double-precision version of HSL_EA19, illustrating its features. The tests were performed on Dell Precision 490 workstation with Intel Xeon 5130 dual core processor at 2GHz and 3GB RAM. For basic linear algebra operations (dense matrix multiplications and the like), the highly optimized BLAS and LAPACK routines provided by Intel Math Kernel Library (version 10.1) were employed. All calculations were in double precision.

We start our presentation with a note on the way the convergence is monitored in the tests in §4.1. In §4.2, we present the results of preliminary tests with the standard eigenvalue problem for the discretized Laplace operator in three dimensions and then proceed to problems for some matrices from Tim Davis' collection in §4.3.1 and §4.3.2, and the generalized eigenvalue problem for three-dimensional elasticity system discretized by bi-linear finite elements on a rectangular mesh in §4.4. Finally, in §4.5 we compare the performance of HSL_EA19 with that of some other eigensolvers designed for large-scale problems.

4.1 A note on the convergence monitoring

Portraying the convergence of approximate eigenpairs to the exact ones, assuming the latter available, is not a trivial issue. Three obvious options are: (i) plot the residual norm history, (ii) plot the eigenvalue errors $\lambda_j^i - \lambda_j$ history, or (iii) plot the eigenvector errors $x_j^i - x_j$ history in a suitable norm. It is shown by Ovtchinnikov (2008a) and Ovtchinnikov (2008b) that these three error measures are actually asymptotically equivalent, provided that certain special norms are used for residuals and eigenvector errors. However, the residual norm that is asymptotically equivalent to the eigenvalue error is very expensive to compute. The eigenvalue and eigenvector errors seem to be readily available if the exact eigenvalues and eigenvectors are known, but the convergence plots in terms of these are often confusing, owing to the fact that x_j^i might approximate x_k for some $k \neq j$. With eigenvectors, one runs into additional difficulties caused by clustering and multiple eigenvalues. To avoid these problems in this report, we opt for the eigenvector error measure of the sines of the angles between a pre-selected number of the exact leftmost eigenvectors[14] and the subspace spanned by the same number of the approximate ones. We note that these

[13] In the literature, one encounters deflated algorithms that use $g^{i+1} = r(x^{i+1})$ rather than $g^{i+1} = r_\perp(x^{i+1})$. The available convergence results for such algorithms (see e.g. Dyakonov (1996), Chapter 9, §5) are technically fairly cumbersome, and there may actually be no practical savings in using $r(x^{i+1})$ rather than $r_\perp(x^{i+1})$ in (2.2), since one may need to compute $\|r_\perp(x^{i+1})\|$ to check for convergence.

[14] In most tests this number is simply equal to the number of wanted eigenvalues. In tests featuring multiple eigenvalues, we exclude eigenvectors corresponding to leftmost wanted eigenvalues if their multiplicity is such that not all of the corresponding eigenvectors have been computed.

angles are asymptotically, as the iteration number increases, equivalent to the angles between respective exact and approximate eigenvectors, see e.g., Knyazev (1997), who shows that the asymptotic equivalence is 'cluster robust', i.e. not affected by small distances between clustered eigenvalues.

We note that a computed eigenpair does not change from the iteration in which it is found to satisfy its convergence criterion until the whole iteration procedure is restarted as a final check. Hence many of the error curves reduce, then are flat, and finally reduce a bit more.

4.2 Preliminary tests

The tests of this subsection deal with the standard eigenvalue problem for the Laplace equation in the three-dimensional brick-like domain with strong boundary conditions discretized by standard 7-point finite differences on a regular rectangular mesh. The eigenvectors and eigenvalues of this problem are available analytically, which makes it possible to measure errors and thus easily assess the convergence of iterations and other features of the algorithm implemented by `HSL_EA19`. Furthermore, unlike with the tests on matrices in §4.3.1 and §4.3.2, one can easily vary the parameters of the problem, such as the size or the layout of the eigenvalues, which makes it possible to demonstrate the algorithm's robustness with respect to the mesh size of the discretization (provided that a suitable preconditioner, such as AMG, is used) and eigenvalue clustering.

Figure 4.2: Eigenvector errors for 3D Laplacian in the 1-by-1.01-by-1.02 brick discretized by 7-point finite differences on $\mathtt{n} \times \mathtt{n} \times \mathtt{n}$ grid, $m = 15$.

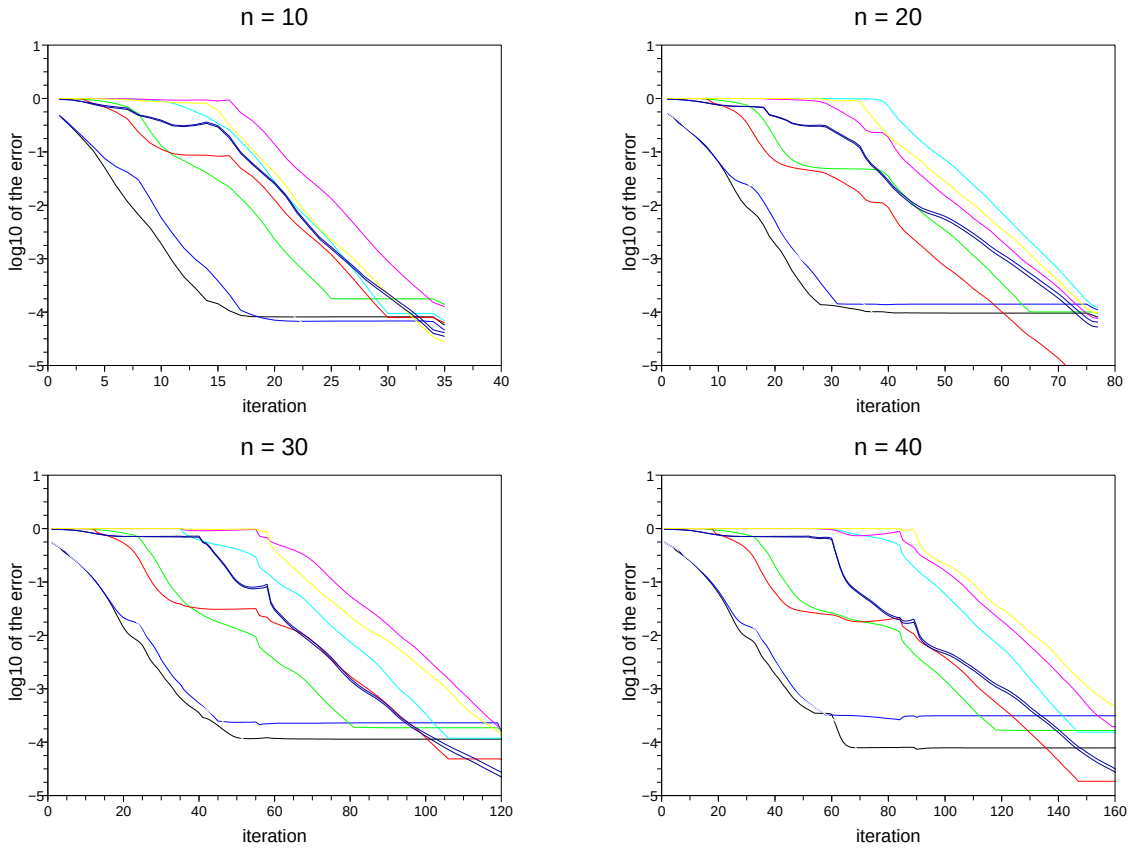
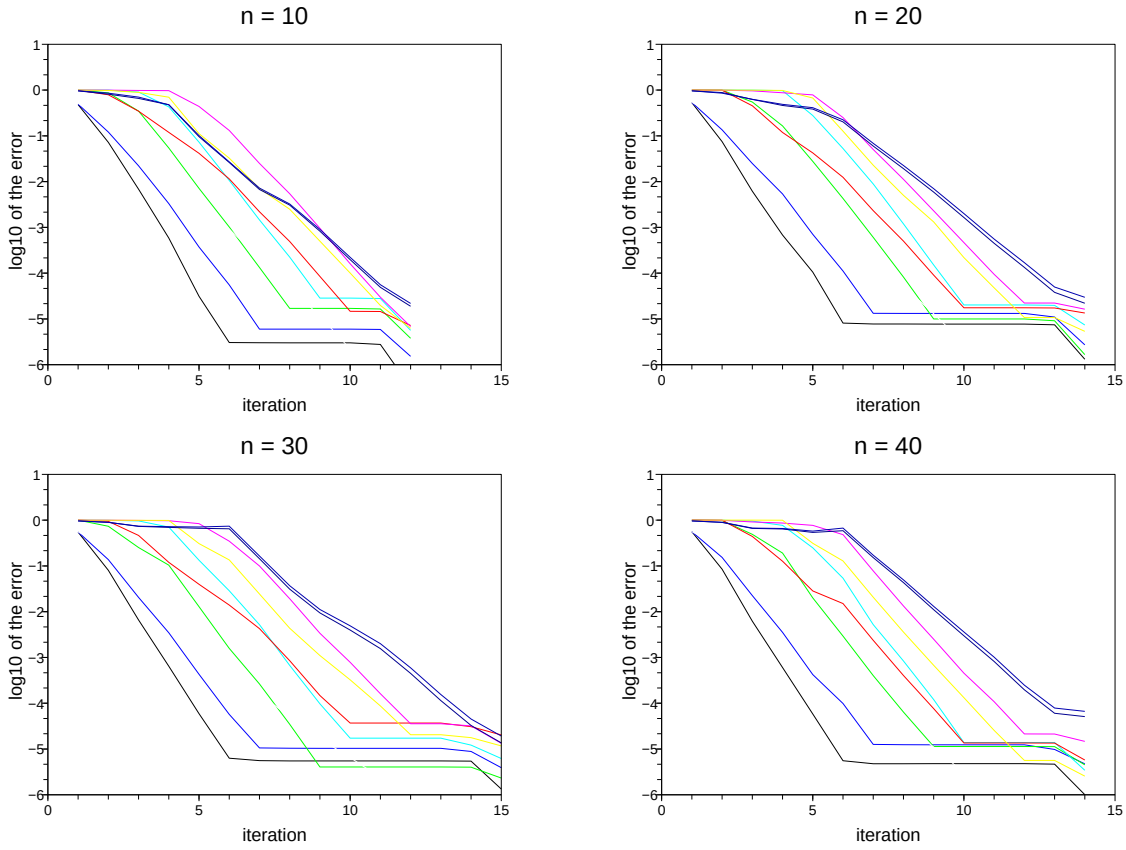


Figure 4.3: Same as Fig. 4.2 but computed with AMG preconditioner.



4.2.1 Convergence

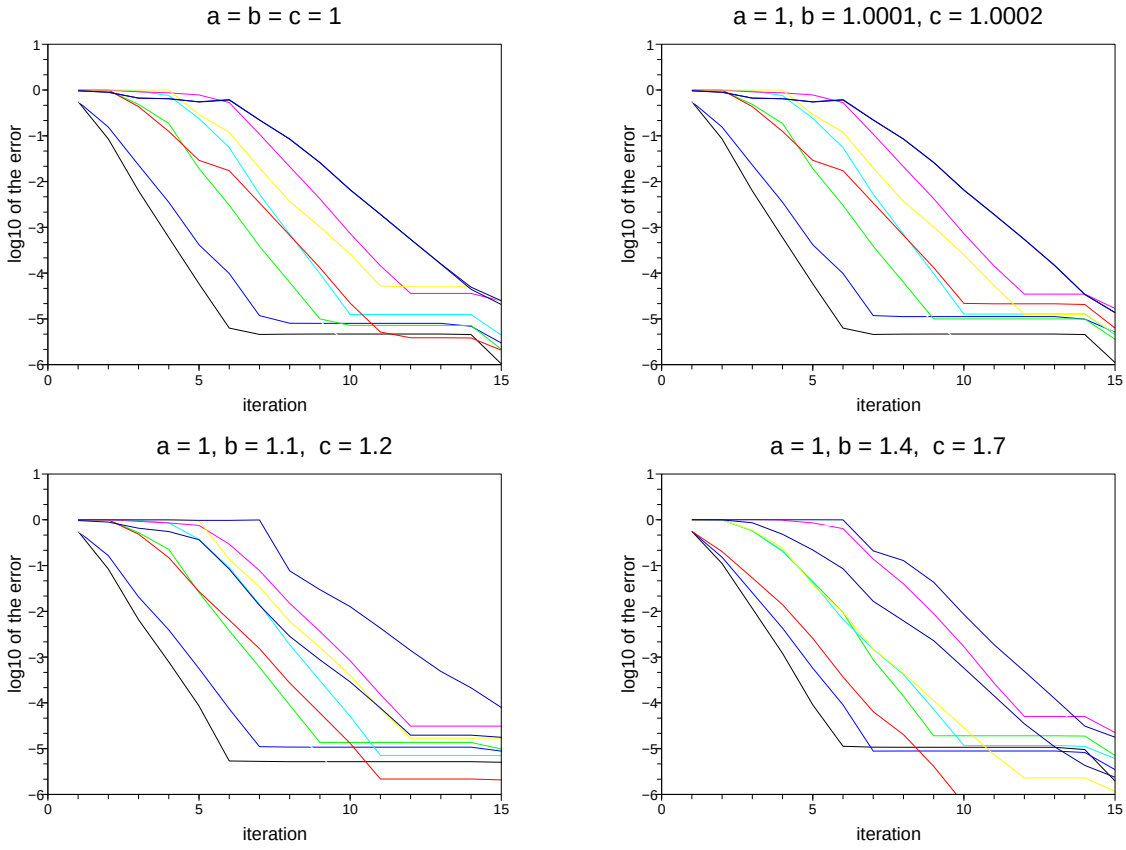
We start with a few tests illustrating the convergence properties of the JCPG algorithm and the way they are affected by preconditioning. In Fig. 4.2, the errors in the first 9 eigenvectors are plotted. The brick sizes are `a` = 1, `b` = 1.01 and `c` = 1.02 (note that closeness to a unit cube creates tight eigenvalue clusters), and `n`-by-`n`-by-`n` regular grid is used for four different values of `n`. No preconditioning is used, and $m = 15$. Available convergence results for block gradient-based eigensolvers (see e.g. Ovtchinnikov, 2006a) suggest that in the case of the standard eigenvalue problem, the convergence of iterations is largely determined by the condition number of the matrix (multiplied by the preconditioner if one is used) in a very much the same way as in the case of a linear system. For instance, for the steepest descent method, the number of iterations needed to achieve a given accuracy in eigenpairs is bounded by a value proportional to this condition number. By extending the analogy to Conjugate Gradients, one expects a bound that is proportional to the square root of the condition number – see Knyazev (2001). In the case at hand, the condition number is proportional to $\mathtt{n}^2$, hence the linear dependence of the iteration number on `n` is to be expected, and the comparison of the four plots in Fig. 4.2, where the range of the 'iterations' axis grows linearly in `n`, meets this expectation.

The next four tests repeat the same calculation but with the use of an Algebraic Multigrid (AMG) preconditioner. One remarkable property of the AMG preconditioning, particularities of which are not relevant to our discussion, is that it delivers convergence that does not depend on the mesh step when applied to a linear system for the discretized Laplacian (or, more generally, a discretized elliptic partial differential equation), and the slope of the convergence curves in Fig. 4.3 demonstrates that the same is

true for the eigenvalue problem.

Although preconditioning substantially increased the CPU time per iteration in these tests, the overall time reduced dramatically: e.g. in the test with `n` = 40, the CPU time per iteration doubled, but overall time reduced from 60 seconds to just 8.

Figure 4.4: Same as Fig. 4.3 for `n` = 40 but for different sizes `a`, `b` and `c` of the brick.



The next series of tests illustrate 'cluster robustness' of JCPG. The sizes `a`, `b` and `c` of the brick are varied starting with `a` = `b` = `c` = 1, which produces multiple eigenvalues that split into clusters when `b` and `c` deviate from 1; when deviation is large enough, as in the last test, clusters disappear. The plots in Fig 4.4 demonstrate that eigenvalue clustering has no tangible effect on the convergence.

4.2.2 Accuracy

The tests of this subsection illustrate the accuracy of `HSL_EA19`. Tables 4.4 and 4.5 present estimated and actual errors for 10 eigenpairs computed in two tests where good accuracy in eigenvalues was sought. It is important to emphasize that the 'estimated eigenvalue errors' refer to estimates for the differences between the Rayleigh quotients of computed eigenvectors and exact eigenvalues, where the Rayleigh quotients were assumed to be computed in higher precision. Since the accuracy in eigenvalues is about twice as good as that in eigenvectors, the estimated errors thus defined may fall below the double-precision machine accuracy – see e.g. the last three lines in Table 4.4. The actual errors were computed by subtracting the exact eigenvalues from approximate in double precision, which sometimes makes the estimated errors appear to underestimate the actual errors.

In Table 4.4, we used the Lehmann bounds and in Table 4.5, we used the kinematic estimates. We

note that, discounting the eigenvalues near 105, the kinematic test slightly underestimated some of the errors but was noticeably closer than when Lehmann bound was in use (cf. §3.2.2).

Table 4.4: Eigenvalue errors for the discretized 3D Laplacian: Lehmann estimates vs. actual errors.

eigenvalue	estimated error	actual error
29.016893439615	$4.35 \cdot 10^{-11}$	$2.96 \cdot 10^{-12}$
57.406397413890	$3.66 \cdot 10^{-11}$	$2.22 \cdot 10^{-12}$
57.971348285210	$9.47 \cdot 10^{-11}$	$6.00 \cdot 10^{-12}$
58.553332269826	$9.47 \cdot 10^{-11}$	$1.06 \cdot 10^{-11}$
86.360852259485	$4.57 \cdot 10^{-11}$	$4.14 \cdot 10^{-12}$
86.942836244097	$7.68 \cdot 10^{-11}$	$4.77 \cdot 10^{-12}$
87.507787115415	$9.56 \cdot 10^{-11}$	$7.23 \cdot 10^{-12}$
104.537088980872	$2.89 \cdot 10^{-18}$	$-5.68 \cdot 10^{-14}$
106.039940186432	$1.90 \cdot 10^{-16}$	$1.85 \cdot 10^{-13}$
107.588101942254	$9.09 \cdot 10^{-16}$	$2.13 \cdot 10^{-13}$

Table 4.5: Same as Tab. 4.4 but with kinematic estimates.

eigenvalue	estimated error	actual error
29.016893439656	$4.45 \cdot 10^{-11}$	$4.39 \cdot 10^{-11}$
57.406397413934	$5.24 \cdot 10^{-11}$	$4.59 \cdot 10^{-11}$
57.971348285239	$4.98 \cdot 10^{-11}$	$3.43 \cdot 10^{-11}$
58.553332269881	$6.47 \cdot 10^{-11}$	$6.55 \cdot 10^{-11}$
86.360852259487	$7.92 \cdot 10^{-12}$	$6.51 \cdot 10^{-12}$
86.942836244149	$4.86 \cdot 10^{-11}$	$5.74 \cdot 10^{-11}$
87.507787115460	$6.57 \cdot 10^{-11}$	$5.28 \cdot 10^{-11}$
104.537088980871	$3.37 \cdot 10^{-19}$	$-1.71 \cdot 10^{-13}$
106.039940186432	$2.23 \cdot 10^{-14}$	$1.42 \cdot 10^{-13}$
107.588101942254	$4.20 \cdot 10^{-14}$	$8.53 \cdot 10^{-14}$

Tables 4.6 and 4.7 present the results of two similar tests that computed eigenvectors to near machine accuracy in double precision. Again, the kinematic estimates are more accurate.

Table 4.6: Eigenvector errors for the discretized 3D Laplacian: residual estimates vs. actual errors.

eigenvalue	estimated error	actual error
29.016893439612	$8.99 \cdot 10^{-11}$	$1.39 \cdot 10^{-11}$
57.406397413881	$8.95 \cdot 10^{-11}$	$1.10 \cdot 10^{-11}$
57.971348285204	$9.40 \cdot 10^{-11}$	$1.37 \cdot 10^{-11}$
58.553332269816	$9.73 \cdot 10^{-11}$	$1.91 \cdot 10^{-11}$
86.360852259481	$8.98 \cdot 10^{-11}$	$1.15 \cdot 10^{-11}$
86.942836244092	$9.05 \cdot 10^{-11}$	$1.28 \cdot 10^{-11}$
87.507787115408	$9.35 \cdot 10^{-11}$	$1.37 \cdot 10^{-11}$
104.537088980871	$8.24 \cdot 10^{-13}$	$2.68 \cdot 10^{-14}$
106.039940186432	$9.43 \cdot 10^{-11}$	$1.38 \cdot 10^{-11}$
107.588101942254	$9.41 \cdot 10^{-11}$	$1.46 \cdot 10^{-11}$

Table 4.7: Same as Tab. 4.6 but with kinematic estimates.

eigenvalue	estimated error	actual error
29.016893439612	$4.33 \cdot 10^{-11}$	$2.18 \cdot 10^{-11}$
57.406397413888	$6.00 \cdot 10^{-12}$	$3.16 \cdot 10^{-12}$
57.971348285204	$7.90 \cdot 10^{-12}$	$5.06 \cdot 10^{-12}$
58.553332269816	$4.78 \cdot 10^{-11}$	$2.62 \cdot 10^{-11}$
86.360852259480	$1.07 \cdot 10^{-12}$	$4.56 \cdot 10^{-13}$
86.942836244092	$1.04 \cdot 10^{-11}$	$4.63 \cdot 10^{-12}$
87.507787115408	$7.07 \cdot 10^{-11}$	$2.99 \cdot 10^{-11}$
104.537088980872	$9.14 \cdot 10^{-14}$	$2.02 \cdot 10^{-14}$
106.039940186432	$6.76 \cdot 10^{-11}$	$2.97 \cdot 10^{-11}$
107.588101942254	$6.92 \cdot 10^{-11}$	$3.02 \cdot 10^{-11}$

4.3 Tests on matrices from Tim Davis' collection

In this section, we present the results of the tests with matrices from Tim Davis' (University of Florida) matrix collection.

4.3.1 PARSEC matrices

In this section, we present the results of the tests with matrices from PARSEC group of Tim Davis' collection. The exact eigenpairs were not available, hence we used high accuracy approximations to eigenpairs (computed with the eigenvector error tolerance of 10^{-12}) in place of exact ones. The matrices of the PARSEC group arise from the 37-point central finite difference discretization of a second order elliptic partial differential equation in a 3D domain of the form $H = -\alpha\Delta\psi + V\psi$, where $-\Delta$ is the Laplace operator, α is a scalar and V a non-differential operator.

The preconditioner that was used in the tests was constructed as follows. We approximated H and Δ on the same grid using the standard 7-point finite-difference discretization and computed the leftmost eigenvalue $\tilde{\lambda}_1$ of this low-order discretization of H by using `HSL_EA19` with an Algebraic Multigrid (AMG) preconditioner for the low-order discretization of $-\Delta$. To precondition the high-order discretization, we used the AMG preconditioner of the low-order discretization of $H - \tilde{\lambda}_1 I$. This preconditioning gave a good convergence rate and the preconditioning cost was modest because of being based on the low-order discretization: e.g. in the test where 20 eigenpairs of the matrix `Si41Ge41H72` were computed, the cost per iteration increased by less than 50%.

The distribution of the leftmost eigenvalues of the matrices `Si34H36` (n = 97,569; 5,156,379 nonzeros) and `Si41Ge41H72` (n = 185,639; 15,011,265 nonzeros) selected for our tests are shown on Fig. 4.5 and 4.6 respectively.

Figure 4.5: The spectrum of `Si34H36`: 100 leftmost eigenvalues.

Typical convergence behaviour of `HSL_EA19` for PARSEC matrices is illustrated by the four plots of Fig. 4.7 showing eigenvector error histories. Four different amounts of eigenpairs of the matrix `Si34H36` are computed. Preliminary tests suggested that setting the iterated subspace dimension m to the number of wanted eigenpairs plus 5 delivered the best overall performance in terms of CPU time. We observe

Figure 4.6: The spectrum of `Si41Ge41H72`: 100 leftmost eigenvalues.

Table 4.8: CPU per eigenpair (sec) for PARSEC matrices.

number of eigenpairs	20	40	60	80
`Si34H36`	2.0	2.0	1.8	1.6
`Si41Ge41H72`	10.2	6.7	7.4	8.6

a remarkably fast convergence in the test computing 80 eigenpairs, thanks to the sizeable gap in the spectrum between 85-th and 86-th eigenvalue (the second largest gap in Fig. 4.5) making the relative distance between the leftmost 80 eigenvalues and 86-th one, which determines the convergence rates, quite substantial.

The performance of `HSL_EA19` in terms of the CPU time per eigenpair is summarized in Table 4.8. We observe that the computation time remains approximately the same when the number m of iterated eigenpairs grows (the same 10^{-6} accuracy in eigenvectors was requested in all tests). Thus, the dependence of the total computation time on m is almost linear, which is apparently due to improved convergence as m increases and the use of highly-optimized BLAS subroutines for dense linear algebra tasks involving $\mathcal{O}\left(nm^2\right)$ floating-point operations.

Tables 4.9 and 4.10 compare the 'actual' errors in eigenpairs with those estimated using Lehmann and residual bounds of §3.2.1 and kinematic bounds pf §3.2.2 respectively. Just like with Laplacian, we observe that the kinematic bounds are noticeably more accurate.

4.3.2 DNVS matrices

In this section, we present the results of the tests with matrices from DNVS group of Tim Davis' collection. Since the exact eigenpairs were not available, we used those computed to 10^{-7} eigenvector accuracy instead. For preconditioning, we computed $v = Ku$ as the approximate solution of the system $Lv = u$ after 5 iterations of CG with diagonal preconditioning, which added about 200% to CPU time per iteration but made it possible to compute wanted eigenpairs within minutes rather than hours.

The distribution of the leftmost eigenvalues of the matrices `shipsec1` ($n = 140{,}874$; 3,568,176 nonzeros) and `shipsec5` ($n = 179{,}860$; 4,598,604 nonzeros) selected for our tests are shown on Fig. 4.8 and 4.9 respectively. Typical convergence behaviour is shown on Fig. 4.10 and the performance on Table 4.11; these are pretty much the same as for PARSEC matrices. In all tests, m is greater than the number of wanted eigenpairs by 10, as preliminary tests with various m suggested to be optimal, and 10^{-3} accuracy in eigenvectors was requested. Finally, the 'actual' and estimated accuracy in eigenpairs is demonstrated by Table 4.12 (the number of wanted eigenvalues is 10); the Lehmann and residual error estimates for these matrices provide bounds that are too far off the mark, apparently due to the large values of matrix entries, to be of any use (cf. the discussion at the end of §3.2.2).

Figure 4.7: Convergence to eigenvectors of `Si34H36`.

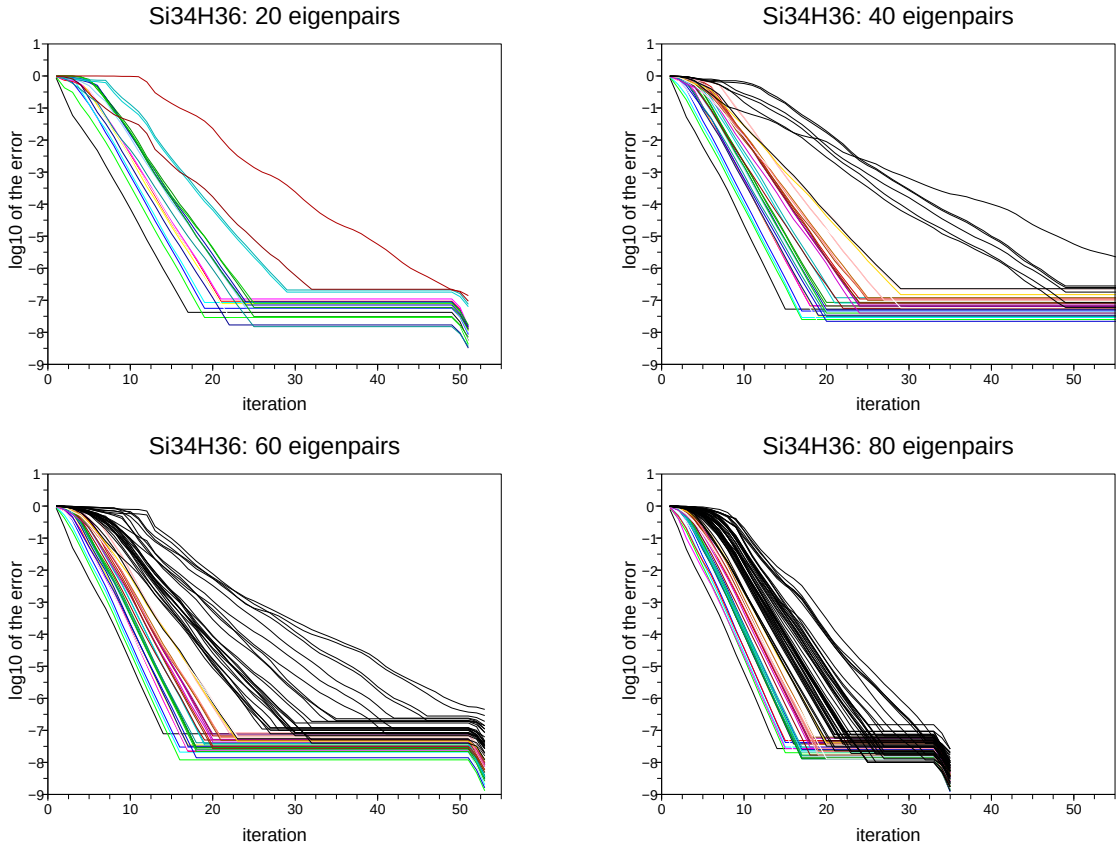


Figure 4.8: The spectrum of `shipsec1`: 100 leftmost eigenvalues.

Figure 4.9: The spectrum of `shipsec5`: 90 leftmost eigenvalues.

Table 4.9: Lehmann/residual-estimated and actual errors for 12 leftmost eigenpairs of `si34h36`.

eigenvalue	eigenvalue error		eigenvector error	
	estimated	actual	estimated	actual
-1.15862684250	$9.4 \cdot 10^{-9}$	$4.2 \cdot 10^{-10}$	$1.8 \cdot 10^{-4}$	$1.7 \cdot 10^{-5}$
-1.12438397505	$1.8 \cdot 10^{-7}$	$7.2 \cdot 10^{-9}$	$8.5 \cdot 10^{-4}$	$7.7 \cdot 10^{-5}$
-1.12438282394	$3.1 \cdot 10^{-8}$	$1.3 \cdot 10^{-9}$	$3.7 \cdot 10^{-4}$	$3.2 \cdot 10^{-5}$
-1.12438126267	$2.7 \cdot 10^{-8}$	$1.2 \cdot 10^{-9}$	$3.3 \cdot 10^{-4}$	$3.3 \cdot 10^{-5}$
-1.07704852744	$7.8 \cdot 10^{-8}$	$2.7 \cdot 10^{-9}$	$6.0 \cdot 10^{-4}$	$5.5 \cdot 10^{-5}$
-1.07704725618	$9.1 \cdot 10^{-8}$	$3.6 \cdot 10^{-9}$	$6.9 \cdot 10^{-4}$	$6.3 \cdot 10^{-5}$
-1.06245139726	$6.1 \cdot 10^{-8}$	$2.0 \cdot 10^{-9}$	$5.6 \cdot 10^{-4}$	$5.0 \cdot 10^{-5}$
-1.06245055613	$1.1 \cdot 10^{-7}$	$3.9 \cdot 10^{-9}$	$7.6 \cdot 10^{-4}$	$6.7 \cdot 10^{-5}$
-1.06245016280	$2.8 \cdot 10^{-8}$	$9.3 \cdot 10^{-10}$	$4.0 \cdot 10^{-4}$	$3.1 \cdot 10^{-5}$
-1.01335308149	$1.2 \cdot 10^{-7}$	$3.0 \cdot 10^{-9}$	$9.1 \cdot 10^{-4}$	$6.8 \cdot 10^{-5}$
-1.01335027853	$1.1 \cdot 10^{-7}$	$3.0 \cdot 10^{-9}$	$9.1 \cdot 10^{-4}$	$6.6 \cdot 10^{-5}$
-1.01334990091	$5.9 \cdot 10^{-8}$	$1.7 \cdot 10^{-9}$	$6.6 \cdot 10^{-4}$	$5.4 \cdot 10^{-5}$

Table 4.10: Same as Table 4.9 but with kinematic estimates.

eigenvalue	eigenvalue error		eigenvector error	
	estimated	actual	estimated	actual
-1.15862684284	$1.6 \cdot 10^{-10}$	$7.7 \cdot 10^{-11}$	$2.4 \cdot 10^{-5}$	$8.8 \cdot 10^{-6}$
-1.12438398209	$5.8 \cdot 10^{-10}$	$1.3 \cdot 10^{-10}$	$4.8 \cdot 10^{-5}$	$1.1 \cdot 10^{-5}$
-1.12438282523	$1.1 \cdot 10^{-10}$	$1.8 \cdot 10^{-11}$	$2.1 \cdot 10^{-5}$	$4.3 \cdot 10^{-6}$
-1.12438126386	$8.1 \cdot 10^{-11}$	$1.8 \cdot 10^{-11}$	$1.8 \cdot 10^{-5}$	$3.9 \cdot 10^{-6}$
-1.07704853006	$4.0 \cdot 10^{-10}$	$7.5 \cdot 10^{-11}$	$4.4 \cdot 10^{-5}$	$9.1 \cdot 10^{-6}$
-1.07704725963	$5.2 \cdot 10^{-10}$	$1.0 \cdot 10^{-10}$	$5.1 \cdot 10^{-5}$	$1.1 \cdot 10^{-5}$
-1.06245139922	$3.3 \cdot 10^{-10}$	$5.7 \cdot 10^{-11}$	$4.2 \cdot 10^{-5}$	$8.1 \cdot 10^{-6}$
-1.06245055992	$5.9 \cdot 10^{-10}$	$9.4 \cdot 10^{-11}$	$5.6 \cdot 10^{-5}$	$1.0 \cdot 10^{-5}$
-1.06245016370	$1.7 \cdot 10^{-10}$	$2.7 \cdot 10^{-11}$	$3.0 \cdot 10^{-5}$	$5.5 \cdot 10^{-6}$
-1.01335308418	$2.1 \cdot 10^{-9}$	$3.1 \cdot 10^{-10}$	$1.2 \cdot 10^{-4}$	$2.0 \cdot 10^{-5}$
-1.01335028147	$8.3 \cdot 10^{-10}$	$9.7 \cdot 10^{-11}$	$7.7 \cdot 10^{-5}$	$1.2 \cdot 10^{-5}$
-1.01334990253	$4.8 \cdot 10^{-10}$	$5.8 \cdot 10^{-11}$	$5.8 \cdot 10^{-5}$	$9.5 \cdot 10^{-6}$

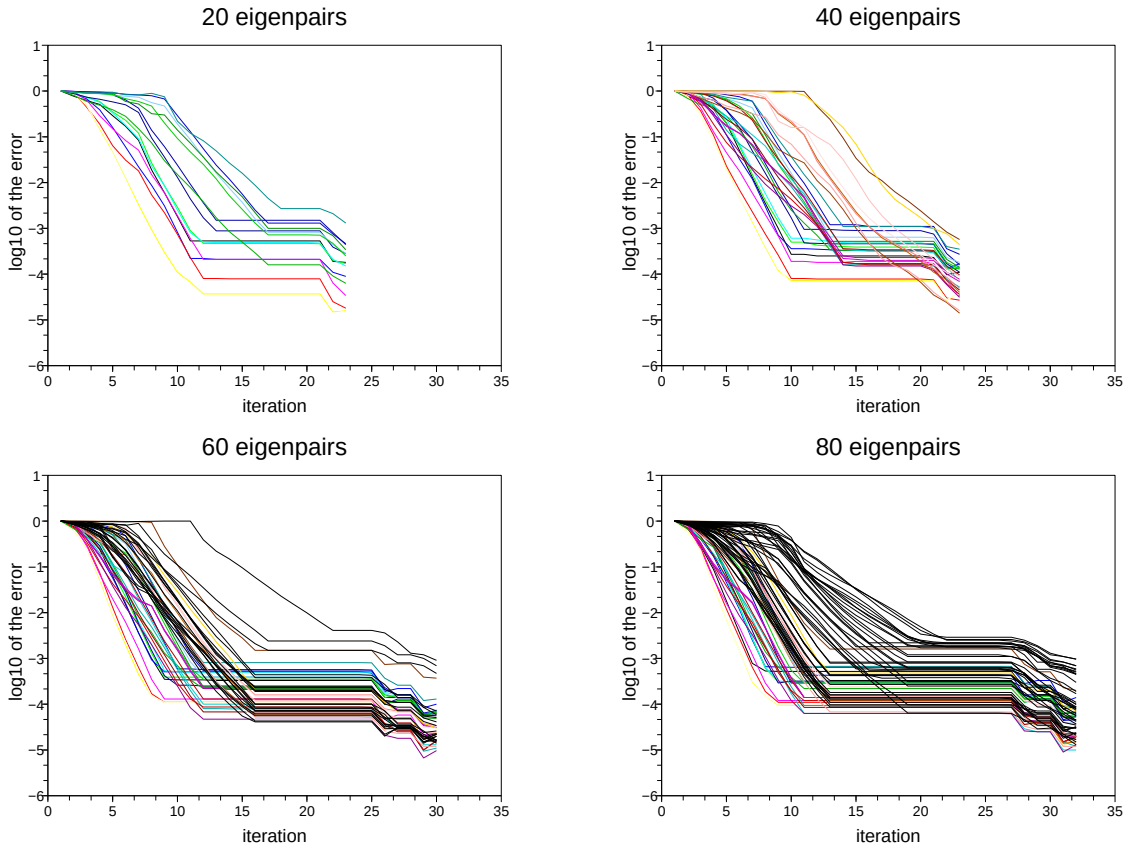
Table 4.11: CPU per eigenpair (sec) for DNVS matrices.

number of eigenpairs	20	40	60	80
`shipsec1`	4.8	4.1	4.5	4.6
`shipsec5`	4.2	4.4	4.9	4.1

Table 4.12: Kinematically estimated and actual errors for 10 leftmost eigenpairs of `shipsec1`.

eigenvalue	eigenvalue error		eigenvector error	
	estimated	actual	estimated	actual
0.1884713114	$4.9 \cdot 10^{-10}$	$3.2 \cdot 10^{-9}$	$5.7 \cdot 10^{-5}$	$6.3 \cdot 10^{-5}$
0.1884713199	$3.3 \cdot 10^{-9}$	$1.2 \cdot 10^{-8}$	$1.5 \cdot 10^{-4}$	$2.2 \cdot 10^{-5}$
0.1986679707	$1.1 \cdot 10^{-9}$	$3.0 \cdot 10^{-9}$	$8.6 \cdot 10^{-5}$	$3.1 \cdot 10^{-5}$
0.1986679726	$1.9 \cdot 10^{-9}$	$4.9 \cdot 10^{-9}$	$1.2 \cdot 10^{-4}$	$3.0 \cdot 10^{-5}$
0.2201740222	$1.6 \cdot 10^{-10}$	$2.9 \cdot 10^{-10}$	$3.7 \cdot 10^{-5}$	$8.6 \cdot 10^{-6}$
0.2201740223	$1.5 \cdot 10^{-10}$	$3.8 \cdot 10^{-10}$	$3.5 \cdot 10^{-5}$	$9.4 \cdot 10^{-6}$
0.2201741172	$4.2 \cdot 10^{-11}$	$1.8 \cdot 10^{-10}$	$1.9 \cdot 10^{-5}$	$7.9 \cdot 10^{-6}$
0.2201741177	$1.8 \cdot 10^{-10}$	$7.7 \cdot 10^{-10}$	$3.9 \cdot 10^{-5}$	$7.7 \cdot 10^{-6}$
0.2337364025	$2.1 \cdot 10^{-9}$	$2.4 \cdot 10^{-9}$	$1.4 \cdot 10^{-4}$	$3.8 \cdot 10^{-5}$
0.2337364064	$5.0 \cdot 10^{-9}$	$6.3 \cdot 10^{-9}$	$2.2 \cdot 10^{-4}$	$3.7 \cdot 10^{-5}$

Figure 4.10: Convergence to eigenvectors of `shipsec1`.



4.4 Tests on matrices generated by 3D FEM for linear elasticity.

The tests reported in this section were run on the generalized eigenvalue problem generated by the finite element method with bilinear shape functions applied to 3D linear elasticity (Lamé) equations in a 5-by-2-by-1 brick domain. The uniform mesh of k-by-k-by-k elements was used, with $k = 30$ and $k = 40$; the two problems solved are referred below as `brick30` (n = 86,490; 6,558,552 nonzeros) and `brick40` (n = 201,720; 15,548,742 nonzeros). AMG preconditioning was used, which added up to 300% to the CPU time per iteration but removed the dependence of the convergence on the mesh step size as can be seen from Table 4.13. The distribution of leftmost eigenvalues of `brick30` and `brick40` are shown on Fig. 4.11 and 4.12 respectively. The convergence behaviour of JCPG iterations is portrayed by eigenvector error histories in Fig. 4.13, and the performance of `HSL_EA19` by Table 4.13. In all tests the eigenvector accuracy is set to 10^{-3} and m exceeds the number of wanted eigenpairs by 5 (found to be a suitable margin in preliminary tests). Finally, Table 4.14 compares the kinematically estimated errors in eigenpairs with 'actual' ones calculated based on eigenvectors computed to 10^{-12} accuracy.

Figure 4.11: The spectrum of `brick30`: 100 leftmost eigenvalues.

Figure 4.12: The spectrum of `brick40`: 100 leftmost eigenvalues.

Table 4.13: CPU per eigenpair (sec) for 3D elasticity matrices.

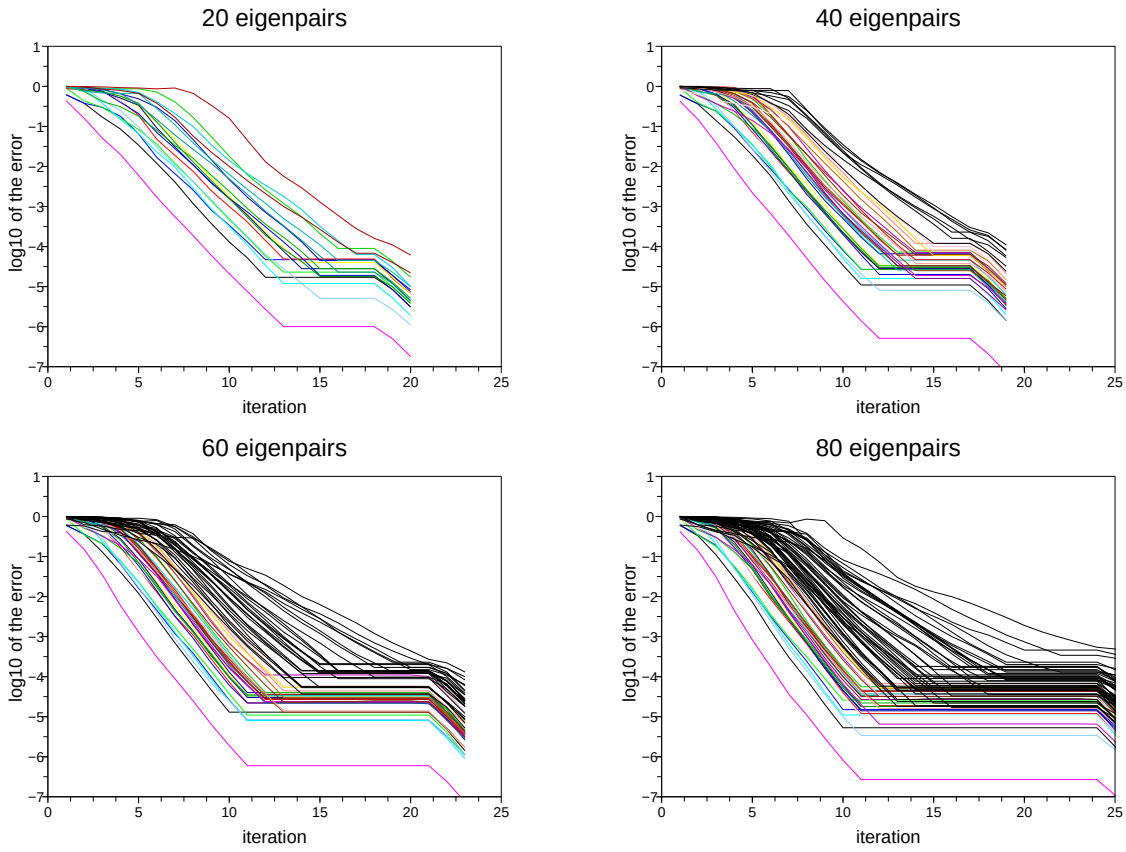
number of eigenpairs	20	40	60	80
`brick30`	8.2	6.9	7.0	7.0
`brick40`	16.0	13.3	13.4	14.1

4.5 Comparisons with other eigensolvers

In this section, we compare the performance of `HSL_EA19` with that of the two eigensolver packages: the package `HSL_EA16` by Meerbergen & Scott (2000) and `JADAMILU` by Bollhoefer & Notay (2007).

In comparing different eigensolver packages, it is important to pay attention to the actual accuracy of eigenpairs they compute – failing to take note of the accuracy may produce misleading results. Regrettably, neither `HSL_EA16` nor `JADAMILU` actually allow the user to compute eigenpairs to a desired accuracy: instead, the user has to set the required residual tolerance. However, computing eigenpairs to the same residual tolerance with different algorithms may result in significantly different levels of accuracy. The comparisons with `HSL_EA16` are even more complicated, since the residual tolerance is imposed on the residuals for inverted-shifted problem rather than the original one. With the discretized differential problems in the tests, this results in relatively higher level of 'short wavelength' error component in

Figure 4.13: Convergence to eigenvectors of `brick30`.



eigenvectors computed by `HSL_EA16`, immediately seen from the fact that Rayleigh quotients differed considerably from the computed eigenvalues. For a fair comparison, in the reported tests with `HSL_EA16` and `JADAMILU`, their residual tolerances were set to the values that delivered comparable accuracy to that of `HSL_EA19`, judged by applying the Rayleigh-Ritz procedure to their computed eigenvectors.

Table 4.14: Kinematically estimated and actual errors for 10 leftmost eigenpairs of `brick30`.

eigenvalue	eigenvalue error		eigenvector error	
	estimated	actual	estimated	actual
0.00401985	$3.4 \cdot 10^{-7}$	$1.5 \cdot 10^{-7}$	$2.3 \cdot 10^{-4}$	$7.2 \cdot 10^{-5}$
0.01356786	$3.2 \cdot 10^{-7}$	$1.3 \cdot 10^{-7}$	$2.2 \cdot 10^{-4}$	$6.5 \cdot 10^{-5}$
0.05894886	$8.2 \cdot 10^{-7}$	$2.5 \cdot 10^{-7}$	$3.5 \cdot 10^{-4}$	$8.5 \cdot 10^{-5}$
0.11771536	$3.8 \cdot 10^{-7}$	$9.4 \cdot 10^{-8}$	$2.4 \cdot 10^{-4}$	$5.4 \cdot 10^{-5}$
0.24612525	$3.9 \cdot 10^{-7}$	$9.7 \cdot 10^{-8}$	$2.5 \cdot 10^{-4}$	$5.5 \cdot 10^{-5}$
0.24945256	$2.4 \cdot 10^{-10}$	$5.6 \cdot 10^{-11}$	$6.2 \cdot 10^{-6}$	$1.2 \cdot 10^{-6}$
0.53316553	$1.4 \cdot 10^{-6}$	$3.3 \cdot 10^{-7}$	$4.8 \cdot 10^{-4}$	$1.2 \cdot 10^{-4}$
0.67561794	$7.1 \cdot 10^{-7}$	$1.3 \cdot 10^{-7}$	$3.4 \cdot 10^{-4}$	$7.2 \cdot 10^{-5}$
1.14044579	$8.7 \cdot 10^{-7}$	$1.7 \cdot 10^{-7}$	$4.0 \cdot 10^{-4}$	$8.0 \cdot 10^{-5}$
1.50602761	$2.2 \cdot 10^{-6}$	$3.5 \cdot 10^{-7}$	$6.5 \cdot 10^{-4}$	$1.2 \cdot 10^{-4}$

4.5.1 HSL_EA19 vs. HSL_EA16

The HSL package HSL_EA16 implements implicitly restarted block (rational) Lanczos method. Since the leftmost eigenvalues are poorly separated from the rest of the spectrum in all of the test problems, we took advantage of the option to employ shift-and-invert technique provided by HSL_EA16 to avoid slow convergence. For the factorization of shifted matrix and subsequent solves we used the HSL package HSL_MA77. With HSL_EA19, the preconditioners described in subsections 4.3.2 and 4.4 were used.

Table 4.15: The performance of HSL_EA16 and HSL_EA19 on shipsec1.

HSL_EA16

number of eigenpairs	20	40	60	80
CPU per eigenpair	6.0	5.8	4.5	6.5
max eigenvalue error	$3.1 \cdot 10^{-8}$	$1.1 \cdot 10^{-7}$	$2.0 \cdot 10^{-4}$	$1.2 \cdot 10^{-6}$
max eigenvector error	$1.8 \cdot 10^{-4}$	$3.7 \cdot 10^{-4}$	$3.2 \cdot 10^{-3}$	$5.3 \cdot 10^{-3}$

HSL_EA19

number of eigenpairs	20	40	60	80
CPU per eigenpair	4.8	4.1	4.5	4.6
max eigenvalue error	$1.3 \cdot 10^{-8}$	$3.4 \cdot 10^{-8}$	$3.1 \cdot 10^{-8}$	$3.2 \cdot 10^{-8}$
max eigenvector error	$1.1 \cdot 10^{-4}$	$9.0 \cdot 10^{-5}$	$1.1 \cdot 10^{-4}$	$1.2 \cdot 10^{-4}$

Table 4.15 summarizes the performance of HSL_EA16 and HSL_EA19 on the shipsec1 matrix. The residual tolerance for HSL_EA16 is set to 10^{-3}. We observe that HSL_EA19 was slightly faster, despite using a not very efficient preconditioner, save for the test with 60 eigenpairs; in the latter, however, the accuracy of HSL_EA16 was noticeably lower.

Table 4.16: The performance of HSL_EA16 and HSL_EA19 on brick30.

HSL_EA16

number of eigenpairs	20	40	60	80
CPU per eigenpair	6.8	5.4	5.0	5.7
max eigenvalue error	$1.3 \cdot 10^{-7}$	$3.0 \cdot 10^{-6}$	$7.4 \cdot 10^{-4}$	$7.1 \cdot 10^{-4}$
max eigenvector error	$2.0 \cdot 10^{-5}$	$4.8 \cdot 10^{-4}$	$4.9 \cdot 10^{-3}$	$8.5 \cdot 10^{-3}$

HSL_EA19

number of eigenpairs	20	40	60	80
CPU per eigenpair	8.2	6.9	7.0	7.0
max eigenvalue error	$2.2 \cdot 10^{-6}$	$3.9 \cdot 10^{-6}$	$6.6 \cdot 10^{-6}$	$3.0 \cdot 10^{-5}$
max eigenvector error	$7.0 \cdot 10^{-4}$	$8.0 \cdot 10^{-4}$	$1.4 \cdot 10^{-3}$	$4.5 \cdot 10^{-3}$

Table 4.16 compares the performance of HSL_EA16 and HSL_EA19 on the brick30 problem. The residual tolerance for HSL_EA16 is set to 10^{-4}. We observe that HSL_EA16 was faster than HSL_EA19. However, if one turns to the results for brick40 reported by Table 4.17 (the same residual tolerance was requested), one finds a completely different picture: now HSL_EA19 is about twice as fast as HSL_EA16. The explanation of the difference between the two tests is fairly simple: sparse factorization algorithms require $\mathcal{O}\left(n^2\right)$ arithmetic operations for the discretized 3D problems and $\mathcal{O}\left(n^{4/3}\right)$ memory storage.[15] We note that

[15] For this reason, comparisons on PARSEC matrices were not feasible.

due to the latter, in the brick40 tests a considerable number of exchanges with external memory was apparently performed, slowing down the computation considerably. In contrast, the computational cost of HSL_EA19 with AMG preconditioning is linear in the size of the problem, as can be clearly seen by comparing the results for brick30 and brick40.

Table 4.17: The performance of HSL_EA16 and HSL_EA19 on brick40.

HSL_EA16

number of eigenpairs	20	40	60	80
CPU per eigenpair	33.8	25.3	33.2	39.9
max eigenvalue error	$1.2 \cdot 10^{-5}$	$6.8 \cdot 10^{-5}$	$6.1 \cdot 10^{-4}$	$2.1 \cdot 10^{-3}$
max eigenvector error	$1.8 \cdot 10^{-4}$	$5.8 \cdot 10^{-4}$	$7.4 \cdot 10^{-3}$	$1.7 \cdot 10^{-2}$

HSL_EA19

number of eigenpairs	20	40	60	80
CPU per eigenpair	16.0	13.3	13.4	14.1
max eigenvalue error	$1.5 \cdot 10^{-5}$	$8.1 \cdot 10^{-6}$	$2.9 \cdot 10^{-5}$	$2.0 \cdot 10^{-5}$
max eigenvector error	$3.1 \cdot 10^{-4}$	$1.4 \cdot 10^{-3}$	$2.5 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$

It should be stressed that the reported comparison results must not be misinterpreted as the evidence that HSL_EA19 makes HSL_EA16 obsolete. In this report, we applied HSL_EA16 and HSL_EA19 to computing several leftmost eigenpairs, whereas the former package actually has much wider range of eigenvalue problems to which it can be efficiently applied. Futhermore, in many structural engineering applications the problem domains are 'almost' two-dimensional, and the computational cost of sparse factorizations is significantly lower, which makes HSL_EA16 more efficient for such problems for now, i.e. until more efficient preconditioners become available.

4.5.2 HSL_EA19 vs. JADAMILU

JADAMILU and HSL_EA19 share two important features: (i) preconditioning, and (ii) Rayleigh quotient minimization. In fact, the algorithms implemented by the two packages, the PCG and Generalized Davidson algorithms, would be identical in exact arithmetic were they applied to the minimization of a quadratic functional with positive-definite Hessian rather than the Rayleigh quotient. The most essential differences between the two eigensolvers actually is the fact that JADAMILU computes one eigenpair at a time, whereas HSL_EA19 computes several eigenpairs simultaneously. This implies that JADAMILU iterations are much cheaper especially in terms of memory storage, however, the convergence of approximate eigenpairs to exact ones is considerably faster with HSL_EA19, and missing an eigenpair is practically impossible with the latter, very much unlike the former.[16] Another important implication is that JADAMILU cannot possibly benefit from the advanced error estimation techniques described in this report; as a result, the error estimates provided by this package have little to do with actual errors.

JADAMILU package comes with built-in multilevel incomplete LU (MILU) preconditioner, however, the user can opt for a one of their own. Since direct access to the MILU preconditioner is not provided, in the tests reported here we used, for a fair comparison, the same AMG preconditioner as that used with HSL_EA19.

Tables 4.18 and 4.19 present the performance comparisons between the two packages in question on PARSEC matrices. The residual tolerance for JADAMILU was 10^{-4}. Clearly, JADAMILU is competitive only

[16] The two ways to avoid missing an eigenpair when using JADAMILU proved to be (in the reported tests): (i) using high enough accuracy, and (ii) computing more eigenpairs than needed because the missing eigenpair will appear sooner or later. The problem is that it is not clear what accuracy or how many extra eigenpairs are needed.

Table 4.18: The performance of JADAMILU and HSL_EA19 on si34h36.

JADAMILU

number of eigenpairs	20	40	60	80
CPU per eigenpair	1.6	2.3	2.9	3.2
max eigenvalue error	$7.9 \cdot 10^{-9}$	$5.6 \cdot 10^{-9}$	$6.4 \cdot 10^{-9}$	$1.3 \cdot 10^{-7}$
max eigenvector error	$5.1 \cdot 10^{-4}$	$3.9 \cdot 10^{-4}$	$5.2 \cdot 10^{-4}$	$7.0 \cdot 10^{-3}$

HSL_EA19

number of eigenpairs	20	40	60	80
CPU per eigenpair	2.0	2.0	1.8	1.6
max eigenvalue error	$6.6 \cdot 10^{-10}$	$1.9 \cdot 10^{-9}$	$7.2 \cdot 10^{-9}$	$3.2 \cdot 10^{-10}$
max eigenvector error	$4.6 \cdot 10^{-5}$	$1.4 \cdot 10^{-4}$	$3.4 \cdot 10^{-4}$	$2.3 \cdot 10^{-5}$

when the number of eigenpairs is not too large: the CPU time per eigenpair grows steadily as this number increases (this fact is admitted by Bollhoefer & Notay (2007)).

Table 4.19: The performance of JADAMILU and HSL_EA19 on si41ge41h72.

JADAMILU

number of eigenpairs	20	40	60	80
CPU per eigenpair	10.9	11.6	13.4	14.7
max eigenvalue error	$4.7 \cdot 10^{-8}$	$4.6 \cdot 10^{-9}$	$7.3 \cdot 10^{-9}$	$5.8 \cdot 10^{-9}$
max eigenvector error	$1.7 \cdot 10^{-3}$	$3.2 \cdot 10^{-4}$	$2.3 \cdot 10^{-3}$	$5.3 \cdot 10^{-4}$

HSL_EA19

number of eigenpairs	20	40	60	80
CPU per eigenpair	10.2	6.7	7.4	8.6
max eigenvalue error	$1.1 \cdot 10^{-8}$	$1.1 \cdot 10^{-8}$	$1.5 \cdot 10^{-8}$	$2.3 \cdot 10^{-8}$
max eigenvector error	$3.7 \cdot 10^{-4}$	$1.9 \cdot 10^{-4}$	$3.8 \cdot 10^{-4}$	$3.8 \cdot 10^{-4}$

Table 4.20: The performance of JADAMILU on shipsec1.

number of eigenpairs	20	40
CPU per eigenpair	33.7	48.9
max eigenvalue error	$8.8 \cdot 10^{-9}$	$3.2 \cdot 10^{-8}$
max eigenvector error	$2.1 \cdot 10^{-4}$	$1.7 \cdot 10^{-4}$

Comparing the results presented in Table 4.20 with those in Table 4.15 clearly demonstrates that shipsec1 is not a matrix on which JADAMILU is competitive to either HSL_EA16 or HSL_EA19 (to which it might be added that the residual tolerance had to be set to as large value as 1 to obtain the reported accuracy in eigenpairs – something that the user would have no way of knowing when applying JADAMILU to this matrix).

Finally, the comparison of Table 4.21 to Table 4.16 demonstrates that HSL_EA19 outperformed JADAMILU considerably in the tests with brick30.

Table 4.21: The performance of JADAMILU on brick30.

number of eigenpairs	20	40	60	80
CPU per eigenpair	12.9	12.7	14.1	15.2
max eigenvalue error	$2.7 \cdot 10^{-5}$	$2.3 \cdot 10^{-6}$	$1.3 \cdot 10^{-5}$	$7.5 \cdot 10^{-5}$
max eigenvector error	$2.1 \cdot 10^{-4}$	$1.9 \cdot 10^{-5}$	$1.1 \cdot 10^{-4}$	$4.2 \cdot 10^{-4}$

5 Code availability

HSL_EA19 is included in the 2007 release of HSL. All use of HSL requires a licence. Licences are available without charge to individual academic users for their personal (non-commercial) research and for teaching; for other users, a fee is normally charged. Details of how to obtain a licence and further details of all HSL packages are available at www.cse.clrc.ac.uk/nag/hsl/.

A Auxiliary results

A.1 Section 3.1

The following auxiliary result is used on step 5 for estimating the condition number of $\widehat{M}$. For the sake of simplicity of formulation, here we employ the dual notation for eigenvalue enumeration: positive indices are used for counting eigenvalues from the left and negative for counting them from the right (cf. Parlett (1980)), i.e. eigenvalues ν_i of a real symmetric/Hermitian matrix N of size n are indexed as $\nu_1 \le \nu_2 \le \cdots \le \nu_n$ and also as $\nu_{-1} \ge \nu_{-2} \ge \cdots \ge \nu_{-n}$ (in other words, for a positive i, ν_i is the i-th leftmost eigenvalue and ν_{-i} the i-th rightmost). We observe that in both cases ν_i is non-decreasing in i, and thus we may say that ν_i are enumerated in the increasing order in either case.

Lemma 1 *Consider a 2-by-2 block matrix*

$$H = \begin{bmatrix} A & C^T \\ C & B \end{bmatrix},$$

where blocks A and B are symmetric and A positive definite. Denote by S the Schur complement of A, i.e. $S = B - CA^{-1}C^T$. Denote by η_i, α_i and σ_i the eigenvalues of H, A and S respectively enumerated in the increasing order. The following inequalities are valid for any positive i that is less than the size of H:

- *if $\eta_i \le 0$ then $\sigma_i \le \eta_i$;*
- *if $0 \le \eta_i < \alpha_1$ then*

$$\frac{2\alpha_1\sigma_i}{\alpha_1 + \sigma_i + \gamma^2 + \sqrt{(\alpha_1 + \sigma_i + \gamma^2)^2 - 4\alpha_1\sigma_i}} \le \eta_i \le \sigma_i, \tag{1.31}$$

  where $\gamma = \frac{\|C\|}{\sqrt{\alpha_1}}$;
- *if $\alpha_{-1} < \eta_{-i}$ then*

$$\sigma_{-i} \le \eta_{-i} \le \frac{2\alpha_{-1}\sigma_{-i}}{\alpha_{-1} + \sigma_{-i} + \gamma^2 - \sqrt{(\alpha_{-1} + \sigma_{-i} + \gamma^2)^2 - 4\alpha_{-1}\sigma_{-i}}}. \tag{1.32}$$

Proof.

We start by applying the Schur complement technique to the equation $Hz = \eta z$, in the same way as it is done e.g. by Mathias (1998). The underlying idea is to rewrite this equation as a system of two equations

$$\begin{aligned} Ax + C^T y &= \eta x \\ Cx + By &= \eta y, \end{aligned}$$

and then exclude the x-component. Equivalently, one computes the product $H_\eta = U^T(H - \eta I_H)U$, where

$$U = \begin{bmatrix} I_A & -(A - \eta I_A)^{-1}C^T \\ 0 & I_B \end{bmatrix},$$

where I_H, I_A and I_B are the identity matrices of the same sizes as H, A and B respectively, and we assume that η is not an eigenvalue of A. It is easy to see that H_η is a block diagonal matrix with diagonal blocks $A - \eta I_A$ and $S_\eta - \eta I_B$, where $S_\eta = B - C(A - \eta I_A)^{-1}C^T$. By the matrix inertia theorem, H_η has the same number of negative, zero and positive eigenvalues as $H - \eta I_H$. Hence, the i-th leftmost eigenvalue of H that lies strictly left from the spectrum of A is also the i-th leftmost eigenvalue of S_{η_i}, and, symmetrically, the i-th rightmost eigenvalue that lies strictly right from the spectrum of A is also the i-th rightmost eigenvalue of $S_{\eta_{-i}}$. Based on this fact, Mathias (1998) derives bounds for eigenvalues of H in terms of those of B and the norm of C that are somewhat less accurate than (1.31) and (1.32) in the case at hand (for one thing, for any positive η_i our lower bound is positive).

In order to prove the inequalities of the lemma, we observe that

$$S_\eta = S + C(A^{-1} - (A - \eta I_A)^{-1})C^T = S - \eta C A^{-1}(A - \eta I_A)^{-1}C^T = S - \eta T_\eta.$$

If $\eta < \alpha_1$, then T_η is positive definite. Hence, for $\eta_i \le 0$, $S_{\eta_i} - S$ is positive semi-definite, which implies the first inequality of the lemma. If $0 \le \eta_i < \alpha_1$, then $S - S_{\eta_i}$ is positive semi-definite, which implies the right-hand side inequality in (1.31). To prove the left-hand side one, we observe that by Weyl's theorem

$$\sigma_i - \eta_i \le \|S - S_{\eta_i}\| = \eta_i \|CA^{-1}(A - \eta_i I_A)^{-1}C^T\| \le \eta_i \frac{\|C\|^2}{\alpha_1(\alpha_1 - \eta_i)} = \frac{\gamma^2 \eta_i}{\alpha_1 - \eta_i}, \tag{1.33}$$

and via simple calculation we arrive at the inequality in question. The inequalities (1.32) are proved in the same way.

A.2 Section 3.2

In the proof of (3.26) we employ the following auxiliary result.

Lemma 2 *Let $\tilde{x}_j$ and $\tilde{\lambda}_j$, $j = 1, \ldots, m$. be Ritz vectors and values of a Hermitian positive-definite operator L enumerated in the increasing order of $\tilde{\lambda}_i$ and denote $\tilde{r}_j = L\tilde{x}_j - \tilde{\lambda}_j \tilde{x}_j$. The following inequality is valid:*

$$\sum_{j=1}^{m} \frac{1}{\tilde{\lambda}_j} \frac{\|\tilde{r}_j\|^2_{L^{-1}}}{\|\tilde{x}_j\|^2_L} \le \sum_{j=1}^{m} \left(\frac{1}{\lambda_j} - \frac{1}{\tilde{\lambda}_j} \right). \tag{1.34}$$

Proof.

The normalization of $\tilde{x}_j$ does not affect (1.34), hence assume $\|\tilde{x}_j\| = 1$. Denote $\tilde{X} = [\tilde{x}_1, \ldots, \tilde{x}_m]$, $\tilde{R} = [\tilde{r}_1, \ldots, \tilde{r}_m]$ and $\tilde{\Lambda} = \operatorname{diag}\{\tilde{\lambda}_1, \ldots, \tilde{\lambda}_m\}$. We have $\tilde{R} = L\tilde{X} - \tilde{X}\tilde{\Lambda}$, and $\tilde{X}^T\tilde{R} = 0$, and hence

$$\tilde{R}^T L^{-1} \tilde{R} = (\tilde{X}^T L - \tilde{\Lambda}\tilde{X}^T)L^{-1}(L\tilde{X} - \tilde{X}\tilde{\Lambda}) = -\tilde{\Lambda} + \tilde{\Lambda}\tilde{X}^T L^{-1}\tilde{X}\tilde{\Lambda}$$

i.e.

$$\tilde{\Lambda}^{-1}\tilde{R}^T L^{-1}\tilde{R}\tilde{\Lambda}^{-1} = \tilde{X}^T L^{-1}\tilde{X} - \tilde{\Lambda}^{-1}.$$

The eigenvalues of $\tilde{X}^T L^{-1}\tilde{X}$ are Ritz values of L^{-1}, and hence they are not greater than $1/\lambda_1, \ldots, 1/\lambda_m$. Hence, taking the trace of both sides of the obtained equation, we arrive at

$$\sum_{j=1}^{m} \frac{\|\tilde{r}_j\|^2_{L^{-1}}}{\tilde{\lambda}^2} \le \sum_{j=1}^{m} \left(\frac{1}{\lambda_j} - \frac{1}{\tilde{\lambda}_j} \right),$$

and thus at (1.34) because $\|\tilde{x}_j\|^2_L = \tilde{\lambda}_j$. **QED**

Another ingredient of the proof of (3.26) is the following result (Lemma 5 of Ovtchinnikov (2006b)).

Lemma 3 *Let $\mu_1 \geq \mu_2 \geq \cdots$ be eigenvalues of a Hermitian operator M acting in a linear space $\mathcal{H}$ with the scalar product $(u,v)_{\mathcal{H}}$ and norm $\|u\|_{\mathcal{H}} = (u,u)^2_{\mathcal{H}}$. Let $\tilde{\mu}_1 \geq \tilde{\mu}_2 \geq \cdots \geq \tilde{\mu}_m$ be the Ritz values of M in a subspace $\mathcal{V} \subset \mathcal{H}$ of dimension m. For any $l \leq m$ the following inequality holds:*

$$\sum_{j=l}^{m} (\mu_j - \mu_{m+1}) \sin^2_{\mathcal{H}}\{x_j; \mathcal{V}\} \leq \sum_{j=l}^{m} (\mu_j - \tilde{\mu}_j), \tag{1.35}$$

where

$$\sin_{\mathcal{H}}\{u, \mathcal{V}\} = \min_{v \in \mathcal{V}} \frac{\|u - v\|_{\mathcal{H}}}{\|u\|_{\mathcal{H}}}.$$

Remark 1 *It is not difficult to verify that the proof of the quoted result given by Ovtchinnikov (2006b) does not use the assumption $\mu_m > \mu_{m+1}$ which is present in the formulation of Lemmata 4 and 5 of the cited paper.*

Turning now to (3.26), let us denote $\tilde{\lambda}_j = \lambda^i_j$, $\tilde{x}_j = x^i_j$, $\tilde{\mathcal{X}}_l = \mathcal{X}^i_l$, and $\tilde{\mathcal{X}}_k = \mathrm{span}\{\tilde{x}_1, \ldots, \tilde{x}_k\}$, and let us assume for now that L is positive definite. Assuming $\tilde{x}_j$ normalized, we have

$$\sum_{j=1}^{l} (\lambda_{k+1} - \tilde{\lambda}_j) = \sum_{j=1}^{l} (\lambda_{k+1}\tilde{x}_j - L\tilde{x}_j, \tilde{x}_j) \leq \sum_{j=1}^{l}\sum_{i=1}^{k} (\lambda_{k+1} - \lambda_i)|(\tilde{x}_j, x_i)|^2 =$$
$$\sum_{j=1}^{l}\sum_{i=1}^{l} (\lambda_{k+1} - \lambda_i)|(\tilde{x}_j, x_i)|^2 + \sum_{j=1}^{l}\sum_{i=l+1}^{k} (\lambda_{k+1} - \lambda_i)|(\tilde{x}_j, x_i)|^2. \tag{1.36}$$

We observe that

$$\sum_{j=1}^{l} |(\tilde{x}_j, x_i)|^2 = \cos^2\{x_i; \tilde{\mathcal{X}}_l\} = 1 - \sin^2\{x_i; \tilde{\mathcal{X}}_l\}. \tag{1.37}$$

Since $\tilde{x}_j$ is a Ritz vector in $\tilde{\mathcal{X}}_k$, $(r_j, \tilde{x}) = 0$ for any $\tilde{x} \in \tilde{\mathcal{X}}_k$, and hence

$$(\lambda_i - \tilde{\lambda}_j)(\tilde{x}_j, x_i) = (L\tilde{x}_j - \tilde{\lambda}_j\tilde{x}_j, x_i) = (r_j, x_i) = (r_j, x_i - \tilde{x}) \leq \|r_j\|_{L^{-1}} \|x_i - \tilde{x}\|_L,$$

i.e.

$$|(\tilde{x}_j, x_i)| \leq \frac{\|r_j\|_{L^{-1}}}{\lambda_i - \tilde{\lambda}_j} \min_{\tilde{x} \in \tilde{\mathcal{X}}_k} \|x_i - \tilde{x}\|_L.$$

Introducing the n-dimensional vector space $\mathcal{H}$ with the scalar product $(u,v)_{\mathcal{H}} = (Lu, v)$ and the norm $\|u\|_{\mathcal{H}} = (Lu, u)^{1/2}$, we can rewrite the last inequality as

$$|(\tilde{x}_j, x_i)| \leq \frac{\|r_j\|_{L^{-1}}}{\lambda_i - \tilde{\lambda}_j} \sin_{\mathcal{H}}\{x_i; \tilde{\mathcal{X}}_k\} \|x_i\|_L = \frac{\sqrt{\lambda_i}\|r_j\|_{L^{-1}}}{\lambda_i - \tilde{\lambda}_j} \sin_{\mathcal{H}}\{x_i; \tilde{\mathcal{X}}_k\}.$$

Applying Lemma 3 to L^{-1}, we obtain

$$\sum_{i=l+1}^{k} \left(\frac{1}{\lambda_i} - \frac{1}{\lambda_{k+1}}\right) \sin^2_{\mathcal{H}}\{x_i; \tilde{\mathcal{X}}_k\} \leq \sum_{i=l+1}^{k} \left(\frac{1}{\lambda_i} - \frac{1}{\tilde{\lambda}_i}\right),$$

and thus

$$\sum_{j=1}^{l}\sum_{i=l+1}^{k} (\lambda_{k+1} - \lambda_i)|(\tilde{x}_j, x_i)|^2 \leq \sum_{j=1}^{l}\sum_{i=l+1}^{k} \frac{\lambda_{k+1} - \lambda_i}{(\lambda_i - \tilde{\lambda}_j)^2} \lambda_i \sin^2_{\mathcal{H}}\{x_i; \tilde{\mathcal{X}}_k\} \|r_j\|^2_{L^{-1}} =$$
$$\sum_{j=1}^{l} \|r_j\|^2_{L^{-1}} \sum_{i=l+1}^{k} \frac{\lambda_{k+1}\lambda_i^2}{(\lambda_i - \tilde{\lambda}_j)^2} \left(\frac{1}{\lambda_i} - \frac{1}{\lambda_{k+1}}\right) \sin^2_{\mathcal{H}}\{x_i; \tilde{\mathcal{X}}_k\} \leq$$

$$\lambda_{k+1}\sum_{j=1}^{l}\frac{\lambda_{l+1}^2}{(\lambda_{l+1}-\tilde\lambda_j)^2}\|r_j\|_{L^{-1}}^2\sum_{i=l+1}^{k}\left(\frac{1}{\lambda_i}-\frac{1}{\lambda_{k+1}}\right)\sin^2_{\mathcal{H}}\{x_i;\tilde{\mathcal{X}}_k\}\le$$

$$\lambda_{k+1}\frac{\lambda_{l+1}^2\tilde\lambda_l^2}{(\lambda_{l+1}-\tilde\lambda_l)^2}\sum_{j=1}^{l}\frac{\|r_j\|_{L^{-1}}^2}{\tilde\lambda_j^2}\sum_{i=l+1}^{k}\left(\frac{1}{\lambda_i}-\frac{1}{\lambda_{k+1}}\right)\sin^2_{\mathcal{H}}\{x_i;\tilde{\mathcal{X}}_k\}\le$$

$$\lambda_{k+1}\frac{\lambda_{l+1}^2\tilde\lambda_l^2}{(\lambda_{l+1}-\tilde\lambda_l)^2}\sum_{j=1}^{l}\left(\frac{1}{\lambda_j}-\frac{1}{\tilde\lambda_j}\right)\sum_{i=l+1}^{k}\left(\frac{1}{\lambda_i}-\frac{1}{\tilde\lambda_i}\right)\le$$

$$\lambda_{k+1}\frac{\lambda_{l+1}^2\tilde\lambda_l^2}{(\lambda_{l+1}-\tilde\lambda_l)^2}\frac{1}{\lambda_1^2}\sum_{j=1}^{l}(\tilde\lambda_j-\lambda_j)\frac{1}{\lambda_{l+1}^2}\sum_{j=l+1}^{k}(\tilde\lambda_j-\lambda_j)=$$

$$\frac{\lambda_{k+1}}{\lambda_1^2}\frac{\tilde\lambda_l^2}{(\lambda_{l+1}-\tilde\lambda_l)^2}\sum_{j=1}^{l}(\tilde\lambda_j-\lambda_j)\sum_{j=l+1}^{k}(\tilde\lambda_j-\lambda_j)\le$$

$$\frac{\lambda_{k+1}\lambda_{l+1}^2}{\lambda_1^2}\frac{1}{(\lambda_{l+1}-\tilde\lambda_l)^2}\sum_{j=1}^{l}(\tilde\lambda_j-\lambda_j)\sum_{j=l+1}^{k}(\tilde\lambda_j-\lambda_j).$$

Substituting this estimate into (1.36) and taking into account (1.37) yields

$$\sum_{j=1}^{l}(\lambda_{k+1}-\lambda_j)\sin^2\{x_j;\tilde{\mathcal{X}}_l\}\le(1+\epsilon)\sum_{j=1}^{l}(\tilde\lambda_j-\lambda_j),\tag{1.38}$$

where

$$\epsilon=\frac{\lambda_{k+1}\lambda_{l+1}^2}{\lambda_1^2}\frac{1}{(\lambda_{l+1}-\tilde\lambda_l)^2}\sum_{j=l+1}^{k}(\tilde\lambda_j-\lambda_j).$$

Now, if we apply (1.38) to L shifted by $2\lambda_1-\lambda_{l+1}$ (which is positive definite for any L, hence the assumption that L is positive definite can be dropped), then ϵ becomes

$$\epsilon=4\frac{\lambda_{l+1}+\lambda_{k+1}-2\lambda_1}{(\lambda_{l+1}-\tilde\lambda_l)^2}\sum_{j=l+1}^{k}(\tilde\lambda_j-\lambda_j).$$

It remains to note that $\dim\mathcal{X}_l=\dim\tilde{\mathcal{X}}_l=l$, and hence

$$\sin^2\{\mathcal{X}_l;\tilde{\mathcal{X}}_l\}=\max_{x\in\mathcal{X}_l}\sin^2\{x,\tilde{\mathcal{X}}_l\}\le\sum_{j=1}^{l}\sin^2\{x_j;\tilde{\mathcal{X}}_l\},$$

and we arrive at (3.26).

References

P. Arbenz, U. L. Hetmaniuk, R. B. Lehoucq and R. S. Tuminaro (2005) A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods, *Int. J. Numer. Meth. Engng.,* **64**, 204–236.

M. Arioli (2004) A stopping criterion for the conjugate gradient algorithm in a finite element framework, *Numer. Math.,* **97**, 1–24.

Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds. (2000) *Templates for the Solution of Algebraic Eigenvalue Problems. A Practical Guide.* Software Environ. Tools 11, SIAM, Philadelphia.

M. Bollhöfer and Y. Notay (2007) JADAMILU: a software code for computing selected eigenvalues of large sparse symmetric matrices, *Computer Phys. Comm.,* **177**, 951–964.

J. Brandts (2003) The Riccati algorithm for eigenvalues and invariant subspaces of matrices with inexpensive action, *Linear Algebra Appl.,* **358**, 335–365.

E. G. D'yakonov (1996) *Optimization in solving elliptic problems.* CRC Press.

A. Edelman, T. A. Arias and S. T.Smith (1998) The geometry of algorithms with orthogonality constraints, *SIAM J. Matrix Anal. Appl.,* **20**, 303–353.

M. R. Hestenes and E. Stiefel (1952) Methods of conjugate gradients for solving linear systems, *J. Res. National Bureau of Standards,* **49**, 409–436.

HSL (2007) A collection of Fortran codes for large-scale scientific computation. See http://www.cse.scitech.ac.uk/nag/hsl/.

A. V. Knyazev (1987) Convergence rate estimates for iterative methods for mesh symmetric eigenvalue problem, *Soviet J. Numer. Anal. Math. Modelling,* **2** 371–396.

A. V. Knyazev (1997) New estimates for Ritz vectors. *Math. Comput.,* **66**, 985–995.

A. V. Knyazev (2001) Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method, *SIAM J. Sci. Comp.,* **2**, 517–541.

A. V. Knyazev, M. E. Argentati, I. Lashuk, and E. E. Ovtchinnikov (2007) Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX) in Hypre and PETSc, *SIAM J. Sci. Comp.,* **29**, 2224–2239.

N. J. Lehmann (1963) Optimal eigenvalue localization in the solution of symmetric matrix problems, *Numer. Math.,* **5**, 246–272.

R. B. Lehoucq, D. C. Sorensen and C. Yang (1998) *ARPACK USERS GUIDE: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods.* SIAM.

D. E. Longsine and S. F. McCormick (1980) Simultaneous Rayleigh-quotient minimization methods for $Ax = \lambda Bx$, *Linear Algebra Appl.,* **34**, 195–234.

R. Mathias (1998) Quadratic residual bounds for the Hermitian eigenvalue problem, *SIAM J. Matrix Anal. Appl.,* **19**, 541–550.

K. Meerbergen and J. A. Scott The design of a block rational Lanczos code with partial reorthogonalization and implicit restarting, Technical Report RAL-TR-2000-011, Rutherford Appleton Laboratory, 2000.

E. E. Ovtchinnikov (2006a) Cluster robustness of preconditioned gradient subspace iteration eigensolvers, *Linear Algebra Appl.,* **415**, 140–166.

E. E. Ovtchinnikov (2006b) Cluster robust error estimates for the RayleighRitz approximation I: Estimates for invariant subspaces, *Linear Algebra Appl.,* **415**, 167–187.

E. E. Ovtchinnikov (2006c) Cluster robust error estimates for the RayleighRitz approximation II: Estimates for eigenvalues, *Linear Algebra Appl.,* **415**, 188–209.

E. E. Ovtchinnikov (2006d) Sharp convergence estimates for the preconditioned steepest descent method for Hermitian eigenvalue problems, *SIAM. J. Numer. Anal.,* **43**, 2668–2689.

E. E. Ovtchinnikov (2008a) Jacobi correction equation, line search and conjugate gradients in Hermitian eigenvalue computation I: Computing an extreme eigenvalue, *SIAM J. Numer. Anal.,* **46**, 2567–2592.

E. E. Ovtchinnikov (2008b) Jacobi correction equation, line search and conjugate gradients in Hermitian eigenvalue computation II: Computing several extreme eigenvalues, *SIAM J. Numer. Anal.,* **46**, 2593–2619.

E. E. Ovtchinnikov (2008c) Computing several eigenpairs of Hermitian problems by conjugate gradient iterations, *J. Comput. Phys.,* **227**, 9477–9497.

E. E. Ovtchinnikov (2009) Lehmann bounds and eigenvalue error estimation, submitted to *Linear Algebra and its Applications*.

B. N. Parlett (1980) *The Symmetric Eigenvalue Problem.* Prentice Hall.

J. K. Reid (1964) Computational problems in linear algebra, PhD Thesis, Oxford University.

G. L. G. Sleijpen and H. A. van der Vorst (1996) A Jacobi–Davidson iteration method for linear eigenvalue problems, *SIAM J. Matrix Anal. Appl.,* **17**, 401–425.

I. Takahashi (1965) A note on the conjugate gradient method, *Information Processing in Japan,* **5**, 45–49.