Chapter 1

Introduction

The algebraic eigenvalue problem arises in many scientific and engineering applications, for instance (see [Saad, 2011, Section 10] and [Roman, 2011]):

- vibration analysis, to study the resonance phenomenon in buildings and bridges (in structural dynamics);
- resonance in electrical networks;
- analysis of systems modeled by the Schrödinger equation (in quantum chemistry);
- stability analysis of dynamical systems, bifurcation analysis;
- principal components analysis (in data mining and statistics);
- analysis of Markov Chain models (for computing the stationary distribution in discrete state and time random processes);

If the problem dimension is small (less than 10^6), it can be solved by *direct methods*, that obtain the results in a finite number of steps. For instance, the QR method [Francis, 1961] obtains all the eigenvalues and eigenvectors of a matrix of dimension n in $\mathcal{O}(n^3)$ iterations.

However, in many applications the problem dimensions are limited by the computational resources, even if a small quantity of solutions are required. Examples of this can be found in structural dynamics [Bertolini, 1998] and electromagnetism [Arbenz and Geus, 1999] where more accurate results are obtained from solving larger-scale eigenvalue problems as a result of finer-grain discretization. Moreover the problem matrices often have an efficient matrix-vector products, in the sense that the cost in operations and the storage in float point numbers are far from $\mathcal{O}(n^2)$, such as the sparse, the Vandermonde or the Toeplitz matrices.

In these cases *iterative methods* can be more optimal. Considering the computing time, the solution is corrected every iteration until the required quality of the

solution is satisfied, process that in general can be complete in $\mathcal{O}(n^2)$ operations, that is a low cost compared to $\mathcal{O}(n^3)$ of the direct methods. Moreover the time-expensive operations in the iterative methods (*e.g*, the matrix-vector products with the problem matrices, the vector operations such as additions, scaling and inner products) can be implemented with excellent performance in high-performance architectures, like in distributed memory or in GPUs.

On the other hand, considering the memory requirements, iterative methods can be implemented with a storage requirement of $\mathcal{O}(nk)$ with $k \ll n$, excluding the problem matrices. However many direct methods modify the problem matrices with the consequence of breaking their efficient structures, arriving at to the cost of storing almost the full matrix, *i.e.* $\mathcal{O}(n^2)$.

For an extended introduction to the eigenvalue problems and the methods for solving them we refer the reader to [Bai et al., 2000; Stewart, 2001b; Saad, 2011].

1.1 Formal definition and problems classification

The thesis treats the solution of linear eigenvalue problems, whose most generic form is as follows. Given a pair of square matrices A and B of dimension n, the problem consists in finding non-trivial pairs, *i.e.*, $\mathbf{x} \neq \mathbf{0}$, of scalar $\lambda \in \mathbb{C}$ and vector $\mathbf{x} \in \mathbb{C}^n$ such that

$$A\mathbf{x} = \lambda B\mathbf{x}. \tag{1.1}$$

There are up to n distinct pairs, called *eigenpairs*, that satisfy this relation and we denote them as $(\lambda_i, \mathbf{x}_i)$. The λ_i solutions are called *eigenvalues*, and are also the solutions of the *characteristic equation* $\det(A - \lambda B) = 0$. The $\mathbf{x}_i$ solutions are called *(right) eigenvectors*, and the $\mathbf{y}$ solutions that satisfy $\mathbf{y}^* A = \lambda \mathbf{y}^* B$ are called the *left eigenvectors*. The subspace spanned by the eigenvectors with the same eigenvalue λ is called *eigenspace* associated to λ.

An important subclass of problems is the *standard Hermitian eigenproblem (HEP)*, when A is Hermitian and $B = I$. In that case, the problem has n solutions, whose eigenvalues are real and eigenvectors X form an orthonormal basis, *i.e.*, $X^* X = I$, so that

$$A = X\Lambda X^*, \text{ where } \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n) \text{ and } X = [\mathbf{x}_1, \ldots, \mathbf{x}_n]. \tag{1.2}$$

Notice that the left and right eigenvectors coincide in this problem.

Moreover, the eigenvalues are well-conditioned: the eigenvalues change at most as much as the norm of the perturbation in A,

$$|\Lambda(A + E)_i - \Lambda(A)_i| \leq \|E\|_2, \tag{1.3}$$

where $\Lambda(M)_i$ is the i-th smallest eigenvalue of M. Therefore we can bound the error of an approximate eigenpair $(\tilde{\lambda}, \tilde{\mathbf{x}})$ by its associated residual $\mathbf{r} = A\tilde{\mathbf{x}} - \tilde{\lambda}\tilde{\mathbf{x}}$,

$$|\tilde{\lambda} - \lambda| \leq \|\mathbf{r}\|_2. \tag{1.4}$$

When A is Hermitian and B is Hermitian and positive definite, *i.e.*, $\Lambda(B)_i > 0$, the problem is referred to as *generalized Hermitian (GHEP)*. In that case, the problem can be transformed into a standard one,

$$A\mathbf{x} = \lambda B\mathbf{x} \quad \Leftrightarrow \quad L^{-1}AL^{-*}(L^*\mathbf{x}) = \lambda(L^*\mathbf{x}), \tag{1.5}$$

where L is the Cholesky factor of B ($B = LL^*$). By this relation part of the good properties of the standard Hermitian problems are inherited. For instance, the eigenvalues are also real, the eigenvectors form a B-orthonormal basis, *i.e.*, $X^*BX = I$, and the right eigenvectors are also equal to the left ones. In addition, the error of an approximate eigenpair can be bound by its residual $\mathbf{r} = A\tilde{\mathbf{x}} - \tilde{\lambda}B\tilde{\mathbf{x}}$,

$$|\tilde{\lambda} - \lambda| \leq \frac{\|\mathbf{r}\|_{B^{-1}}}{\|\tilde{\mathbf{x}}\|_B}. \tag{1.6}$$

The *standard non-Hermitian problems (NHEP)*, when A is not Hermitian and $B = I$, lose some of the nice properties described above: the eigenvalues can be complex, even if A is real, and the right and the left eigenvectors do not have to coincide, do not necessarily form an orthogonal basis and may not exist n independent vectors (although there are at least as many as distinct eigenvalues).

The rest of the problems are classified as *generalized non-Hermitian problems (GNHEP)* and, beside sharing the NHEP properties described above (notice that if B^{-1} exists the problem can be transformed into standard form), they can have infinite as eigenvalues, whose associated right and left eigenvectors correspond to the non-zero vectors x and y satisfying $Bx = 0$ and $y^*B = 0$, respectively.

In the non-Hermitian cases there is not a simple bound for the error of the eigenvalues and the eigenvectors related with the associated residual. Nevertheless, practical solvers work with the backward error, an estimation of the size of the perturbation on the problem matrices with which the solution provided would be the exact solution. The product of the backward error and the condition number (of eigenvalues and eigenvectors, *i.e.*, the sensitivity of these entities for perturbations in the problem matrices) provides a first order error bound for the computed solution.

For the backward errors and condition numbers for different eigenvalue problems we refer the reader to [Higham and Higham, 1998; Bai et al., 2000].

1.2 Subspace methods

The iterative methods we will refer to belong to the subclass of subspace methods, whose basic structure consists in generating a sequence of subspaces $\mathcal{V}^{(i)}$ and extracting approximated solutions from them.

The Rayleigh-Ritz procedure is the most basic approach to extract approximate pairs $(\tilde{\theta}, \tilde{\mathbf{x}})$ and is considered useful for computing eigenvalues at the periphery of the spectrum. In a standard problem, this technique imposes the Ritz-Galerkin condition on the associated residual to the returned pair,

$$\mathbf{r} := A\tilde{\mathbf{x}} - \tilde{\theta}\tilde{\mathbf{x}} \perp \mathcal{V}, \tag{1.7}$$

which leads to the low-dimensional projected eigenproblem

$$V^*AV\mathbf{u} = \tilde{\theta}V^*V\mathbf{u}. \tag{1.8}$$

From the solutions $(\tilde{\theta}, \mathbf{u})$ the returned pairs $(\tilde{\theta}, \tilde{\mathbf{x}}) = (\tilde{\theta}, V\mathbf{u})$ are obtained, which are the so-called Ritz pairs.

The simplest particular case of a subspace method is the Power method, that iterates with a subspace of fixed dimension 1 (a vector) $\mathbf{v}^{(i)}$, that is replaced by a vector in the direction of $A\mathbf{v}^{(i)}$, where A is the matrix of a standard eigenproblem. The approximate eigenvalue is computed by the Rayleigh quotient $\tilde{\theta}^{(i)} = \rho(\mathbf{v}^{(i)})$, where $\rho(\mathbf{x}) = \mathbf{x}^*A\mathbf{x}(\mathbf{x}^*\mathbf{x})^{-1}$, the particular case of the Rayleigh-Ritz procedure for a subspace spanned by a single vector. The recurrence, if it converges, tends to the eigenvector associated to the largest magnitude eigenvalue with a linear convergence rate.

With a similar scheme, the Rayleigh Quotient Iteration (RQI) updates the vector in the direction of $(A-\tilde{\theta}^{(i)}I)^{-1}\mathbf{v}^{(i)}$, with a convergence rate at least quadratic, and cubic in some cases. Despite its great convergence rate, the inversion of large matrices makes the method inefficient for large problems. See detailed information on the Power method and RQI in [Parlett, 1980].

These two basic methods can be found in the soul of more sophisticated methods, such as methods using Krylov subspaces, which are generated by the sequence of directions produced by the Power method,

$$\mathcal{K}_m(A, \mathbf{x}) = \text{span}\{\mathbf{x}, A\mathbf{x}, A^2\mathbf{x}, \ldots, A^{m-1}\mathbf{x}\}.$$

The Arnoldi method builds a Krylov subspace basis V, orthogonalizing the new vector $\mathbf{v}_i$ against the previous ones in the basis $V_{1:i-1}$ by Gram-Schmidt. As a result the projected matrix V^*AV is upper Hessenberg, and the matrix can be filled with the intermediate results during the Gram-Schmidt orthogonalization.

When A is Hermitian, the projected matrix is symmetric tridiagonal, property exploited by the Lanczos method, that only orthogonalizes the new vector against the two previous ones. However, in finite precision arithmetic, the process does not keep the vectors in the basis sufficiently orthogonal, and as a consequence spurious copies of the eigenvalues appears. Practical versions of Arnoldi and Lanczos can be found in [Parlett, 1980; Saad, 2011].

As the Power method, Krylov methods converge toward eigenvalues in the periphery of the spectrum. Their main drawbacks are exposed when interior eigenvalues are sought, often requiring to handle a matrix inverse, for instance with the shift-and-invert technique [Ericsson and Ruhe, 1980]. This implies solving linear systems quite accurately at each eigensolver iteration, frequently addressed by direct solvers to guarantee robustness.

Davidson methods, which are the main topic of this thesis, try to overcome this limitation by solving linear systems only approximately, without compromising the convergence in many cases, although influencing in the total number of iterations.

Unlike Krylov methods, Davidson methods work with unstructured subspaces, formed by corrections to the approximate eigenvector closest to the target values.

Table 1.1: List of software with Davidson-type methods (GD: Generalized Davidson, JD: Jacobi-Davidson) for the solution of sparse eigenvalue problems, indicating the version and the year of the last release, the main language in which is written and whether distributed memory is supported.

Name	Description	Version	Date	Language	Par.
ANASAZI	GD, LOBPCG	10.8	2011	C++	Yes
JADAMILU	JD (symmetric)	2.0	2009	F77	No
JDQR, JDQZ	JD (NHEP and GNHEP, resp.)	-	2002	Matlab	No
JDCG	JD (symmetric)	-	2005	Matl., F77	No
MPB	Conjugate Gradient, GD	1.4.2	2003	C	Yes
PRIMME	GD, JD (symmetric), LOBPCG	1.1	2006	C	Yes
PySPARSE	JD (symmetric)	1.1.1	2010	C, Python	No
SLEPc	GD, JD	3.2	2011	C	Yes

Preconditioners (approximate inverses, such as incomplete Cholesky or incomplete LU) occupy a central role in Davidson methods, being used in computing the corrections by either accelerating the convergence of an iterative linear solver (as in Jacobi-Davidson) or being applied in order to generate the new vectors (as in Generalized Davidson). A more detailed explanation of Davidson methods is exposed in §2.1 of this thesis.

Precisely the existence of a *good quality* preconditioner for a certain application that properly strikes the balance between numerical behavior and computational performance can make Davidson methods competitive against Krylov methods. This situation is extensible to LOBPCG [Knyazev, 2001] that, although not being a Davidson method, it has a similar structure and its performance is also influenced by the quality of the preconditioner employed.

1.3 Survey of freely available Davidson software

The progress in iterative methods for solving eigenvalue problems is reflected in the proliferation of software in the last 20 years, as observed in the software listed in [Hernandez et al., 2006] and [Hochstenbach, 2007]. In this section we take a brief tour considering the codes and libraries with Davidson methods. The ones with some development in the last 10 years are summarized in Table 1.1.

Two of the oldest codes that implement a Davidson method are the Fortran 77 packages DVDSON [Stathopoulos and Fischer, 1994] and NA18 [Sadkane and Sidje, 1999], block implementations of the Classical Davidson and Generalized Davidson methods, respectively, for computing extreme (*i.e.*, leftmost or rightmost) eigenpairs in large symmetric matrices, with several extensions such as reorthogonalization. A C code with similar features, beside supporting parallelism, can be found in the electromagnetic simulation software MPB [Johnson and Joannopoulos, 2001].

The authors of the Jacobi-Davidson variants JDQR and JDQZ described in [Fokkema et al., 1999] provide a reference implementation in Matlab, that includes standard and harmonic Rayleigh-Ritz extraction and several iterative methods for solving the correction equations. There is available a Fortran 77 code of the JDQZ method using complex arithmetic.

JDCG is a modified version of JDQR for symmetric problems, that solves the Jacobi-Davidson correction equation with CG, using a sensible stopping criterion detailed in [Notay, 2002]. The corresponding algorithm for real GHEP is called JDCG_GEP. JDRPCG is another variant of JDCG with *regular* preconditioning [Notay, 2005].

The jdsym module in PySPARSE provides a Python wrapper to a C-coded Jacobi-Davidson method for GHEP with B-orthogonalization, whose earlier version was developed in the context of a PhD thesis [Geus, 2002].

JADAMILU [Bollhöfer and Notay, 2007] implements a Jacobi-Davidson method for computing the smallest or interior eigenvalues of symmetric matrices. The correction equation is accelerated by a built-in preconditioner based on ILUPACK.

Except MPB, the mentioned codes are sequential, *i.e.*, they run in a single core. The main parallel libraries that provide Davidson eigensolvers are: Anasazi [Baker et al., 2009], that includes a customizable block Generalized Davidson (that with a little coding can behave like a basic Jacobi-Davidson); and PRIMME [Stathopoulos and McCombs, 2010], that includes many methods that fit in the Generalized Davidson scheme, although currently limited to HEP.

1.4 The SLEPc library

SLEPc, Scalable Library for Eigenvalue Problem Computations, provides implementations of iterative methods for solving eigenvalue problems and singular value problems, optimized for high-performance computers. The problem matrices can be defined by explicit (sparse) matrices or implicitly by the matrix-vector products.

The methods in SLEPc include state-of-the-art implementations of the subspace methods described in the previous section, such as power method, inverse iteration, Rayleigh Quotient Iteration (RQI), Arnoldi, Lanczos, Krylov-Schur and Golub-Kahan-Lanczos for SVD, besides Generalized Davidson and Jacobi-Davidson described in this thesis.

SLEPc has been developed and maintained by the GRyCAP group (Grid y Computación de Altas Prestaciones) at the Universitat Politècnica de València following the next guidelines:

stability and robustness the implementations correspond with the most numerically stable variants sought, whose robustness is reinforce by the inclusion of convergence monitors and error detectors;

efficiency the most expensive parts of the code are optimized by parallelizing them and improving the data locality;

abstraction the software is organized in several abstraction layers, hiding implementation details and data structures, and allowing a uniform and consistent access to the methods by an interface with concepts close to the services that are offered. For instance, the different solvers implemented in SLEPc have a common interface to the user, and similarly, the implementations access the problem matrices, the preconditioner and vectors by other interfaces that do not consider the underlying format; and

friendly usability the methods expose in an organized way the options to control the aspects that influence in the performance, with appropriate default values so that non-expert users can skip as much as possible of them.

SLEPc is built on top of PETSc (Portable, Extensible Toolkit for Scientific Computation, [Balay et al., 1997, 2011]), a widespread freely available framework for the numerical solution of partial differential equations, which provided mathematical algorithms using object-oriented programming techniques in order to be able to manage the complexity of efficient numerical codes for different architectures, specially considering distributed memory computers.

The object-oriented approach in PETSc has the objective of reducing the user code to manage a set of data structures and algorithmic objects, avoiding the considerations on the underlying data structures and implementation details. The basic abstract data classes are index sets, vectors and matrices, and on top of that are various classes of solver objects, including linear, nonlinear and time-stepping solvers. Many different iterative linear solvers are provided, including CG and GMRES, together with various preconditioners such as Jacobi or Incomplete Cholesky. PETSc has also the provision to interface with third-party software such as HYPRE.

SLEPc extends PETSc with all the functionality necessary for the solution of eigenvalue problems. SLEPc inherits all the good properties of PETSc, including portability, scalability, efficiency and flexibility. SLEPc also leverages well-established eigensolver packages such as ARPACK, BLOPEX and PRIMME, integrating them seamlessly. Some of the outstanding features of PETSc, also present in SLEPc, are the following:

- object-oriented programming style, that aims at providing properties such as extensibility and ease of maintaining;
- data-structure neutral implementation with transparent support for real and complex arithmetic of single, double and quadruple precision;
- run-time flexibility, giving full control over the solution process (*e.g.*, the method for solving the eigenvalue problem, the linear system and the preconditioner to be used);
- portability to a wide range of parallel platforms; and

- interfaces to different languages other than C/C++, such as Fortran, Matlab and Python.

Like other high-level classes in PETSc, the SLEPc eigenvalue solvers are built on the foundational classes:

Vec that represents dense vectors and can be used in typical algebraic operations like scalings, additions and inner products;

Mat that represents matrices and can be operated with other matrices and vectors, with a wide variety of available implementations like dense matrices, sparse matrices (using Compressed Sparse Row format), block diagonal matrices, permutation matrices or discrete Fourier transform (DFT) matrices;

KSP that represents Krylov iterative methods for solving linear systems, such as CG, GMRES or BiCGStab (for more information about these methods and preconditioners we refer the reader to [Saad, 2003]); and

PC that represents preconditioners (*i.e.*, matrix inverse approximations), which there are a wide variety of implementations (such as Jacobi, block Jacobi, ICC and ILU), besides wrappers to external packages (such as HYPRE[1] or pARMS [Li et al., 2003]).

SLEPc in turn extends the PETSc functionality by the addition of the following classes:

EPS that represents the eigenvalue solvers, including the Davidson methods described in this thesis;

ST that represents a spectral transformation, which is employed by the Krylov solvers in order to deal with GNHEP (by for instance computing the eigenvalues of $B^{-1}A$) and to compute interior eigenvalues;

IP that represents the inner product that is used in the orthogonalization of vectors.

The structure of SLEPc and PETSc is detailed in §2.2.1 and §2.2.2 of this thesis. For general information about the libraries we refer the reader to [Balay et al., 1997, 2011; Hernandez et al., 2005; Campos et al., 2011].

1.5 High-performance computing

High-performance computing has the objective of addressing challenging scientific computing problems, dealing with huge data volumes and/or number of operations. The codes in this context have to consider the underlying computer memory and processor architectures, whose complexity has increased to satisfy the computation

[1] `http://www.llnl.gov/casc/hypre`.

demands of scientific applications. For instance, in order to obtain high performance in supercomputers with hybrid architectures in which the computational processors are arranged in nodes with shared memory (the cores in the same blade) and the communication with processors in outside nodes is significantly slower, the operations and data must be distributed in an appropriate way, maximizing the communication between the processors in the same node.

However, the optimal distribution of the data and the operations is not trivial and depends on the specific characteristics of each machine, which can explain a certain lack of competent software optimized for architectures other than the single core with uniform memory access (UMA), such as multi/many-cores, GPU and distributed memory.

There are software providing high-level access to the *special* functions on these architectures throw standard interfaces, such as:

- Message Passing Interface (MPI) [MPI Forum, 1994], as portable message-passing system to communicate and synchronize processes running on nodes interconnected in some way (distributed memory).
- POSIX threads (Pthreads)[2] and OpenMP[3], as interfaces to shared memory multiprocessing.
- CUDA[4] and OpenCL[5], as interfaces to launch and manage executions on GPUs.

On the other hand, there are two important libraries that provide reference implementations of common operations in scientific computation, BLAS [Blackford et al., 2002] for basic algebra operations in dense vectors and matrices, and LAPACK [Anderson et al., 1992] for advance dense linear algebra operations such as resolution of linear system of equations, eigenvalue problems, singular value problems and least squares.

Major processor vendors provide optimized implementations of BLAS and LAPACK interfaces, like AMD Core Math Library (ACML)[6], Intel Math Kernel Library (MKL)[7], Engineering and Scientific Subroutine Library (ESSL)[8] for IBM Power processors, and CUBLAS[9] for Nvidia GPUs.

In this context, we can find great efforts to extend (or supersede) the BLAS and LAPACK features, for instance supporting other vector and matrix formats

[2]Described in the standard POSIX.1c, Threads extensions (IEEE Std 1003.1.c-1995).

[3]The last version approved by the consortium OpenMP Architecture Review Board is 3.1, and is available at `http://openmp.org/wp/openmp-specifications/`.

[4]Developed by Nvidia Corporation, see `http://www.nvidia.com/object/cuda_home_new.html`.

[5]Currently been developed by the Khronos Group, see `http://www.khronos.org/opencl`.

[6]`http://developer.amd.com/cpu/Libraries/acml/`.

[7]`http://www.intel.com/cd/software/products/asmo-na/eng/perflib/mkl/`.

[8]`http://www-03.ibm.com/systems/software/essl`.

[9]`http://developer.nvidia.com/cuBLAS`.

(*e.g.*, ACML implements the BLAS functionality for sparse matrices), or providing implementations optimized for different architectures (*e.g.*, MAGMA[10] and PLASMA[11]), or including more advance methods for solving linear and non-linear problems (other examples besides PETSc and SLEPc are GNU Scientific Library (GSL)[12] and Trilinos[13]).

1.6 Thesis objectives

As noted before, the Davidson methods are becoming an excellent alternative where Krylov methods do not offer good performance. Despite their potential benefit, it is still difficult to find freely available Davidson parallel implementations, specially for the non-Hermitian case and the generalized case.

Our objective is to provide a robust and efficient parallel implementation of Generalized Davidson and Jacobi-Davidson methods in the context of SLEPc, that can address standard and generalized problems, Hermitian and non-Hermitian, with either real or complex arithmetic, and follows the guidelines described in §1.4:

stability and robustness by studying the different variants for each aspect of the Davidson methods from the numerical point of view, such as the extraction of approximate eigenpairs, and the expansion, restarting and initialization of the search subspace (see §2.1); even studying new approach, as the expansion described in Ch. 3;

efficiency by considering the parallel performance in the implementation of the Davidson methods using PETSc functions (see §2.3), and even extending the PETSc interface such as the case of the multi-vectors (see §2.2.1 and §2.2.2);

abstract by employing the interfaces provided by PETSc and being transparent to the underlying architecture of the machine, as well as implementing as many functions of the SLEPc eigensolver interface;

user friendly by exposing specific parameters to control critical aspects of the Davidson methods, with appropriate default values (as show the results with the test battery §2.4.1).

The stability, robustness and efficiency of the new solvers are validated using a collection of problems whose matrices come from real applications (see §2.4.1), besides two relevant scientific computing applications as a result of international collaborations, namely the computation unstable modes of plasma (treated in Ch. 4 and 5) and the study of the electronic configurations of atoms (treated in Ch. 6), in which obtaining the corresponding eigenpairs is challenging for iterative solvers.

[10] `http://icl.cs.utk.edu/magma/`.
[11] `http://icl.cs.utk.edu/plasma/`.
[12] `http://www.gnu.org/software/gsl/`
[13] `http://trilinos.sandia.gov/`.

1.7 Outline

The chapters of this thesis, except the current one, correspond to papers published, submitted or in preparation, slightly modified to ensure uniform notation.

Chapter 2, submitted to ACM Trans. Math. Softw., starts with a presentation of the Davidson methods and the motivation of the development of parallel solvers to address large-scale eigenvalue problems. After that, the different variants in this family of methods are described, focusing on the ones included in the implementation, that is detailed in subsequent sections. The chapter ends with experimental results solving a collection of problems, that includes standard and generalized problems, both Hermitian and non-Hermitian.

Chapter 3, structured as a paper for future submission, introduces an expansion method that is more general than the corresponding to Generalized Davidson, originated in a collaboration with M. E. Hochstenbach. The convergence of this original expansion is studied, including theoretical and practical results.

The Jacobi-Davidson solver has been employed in GENE, a plasma turbulence code being developed at the Max-Planck-Institut für Plasmaphysik (IPP). Chapter 4, published as [Romero and Roman, 2011], briefly introduces the application and the method used to solve the non-Hermitian eigenproblems that it generates, the Jacobi-Davidson method with harmonic extraction. Chapter 5, published as [Merz et al., 2012], extends the description of GENE, and addresses advanced issues, *e.g.*, the use of a preconditioner based on part of the problem matrix, and initializing the search subspace with previous solutions during parameter scan studies.

Finally, Ch. 6 treats the solution of generalized symmetric-indefinite problems that appear in the finite element analysis of the electronic structure of light atoms modeled by the density functional theory. The chapter is based on a submitted paper to the journal Comput. Phys. Commun., written in collaboration with T. D. Young in the context of the deal.II library.

Because of chapters being self-contained, some topics are introduced or detailed more than once along the chapters. Theoretical backgrounds of Davidson expansions are discussed in Ch. 2, 3 and 4 and summarized in Ch. 5 (for Jacobi-Davidson in standard problems) and Ch. 6 (for Generalized Davidson for indefinite problems). In addition, the harmonic extraction is fully detailed in Ch. 2 and summarized in Ch. 4 for standard problems. Chapter 2 is devoted to the implementation of the Davidson solvers in SLEPc, although introductions to PETSc, SLEPc and Davidson solvers are also found in chapters 4, 5 and 6.

Discussions about controlling the resolution tolerance in the Jacobi-Davidson correction equation are found in chapters 2 and 4, with interesting experiment results in the latter. Examples of the performance gain by subspace recycling, *i.e.*, starting with a solution of a similar problem, is presented in chapters 5 and 6.

Two scientific applications are referred in the last chapters: the plasma turbulence code GENE, treated in chapters 4 and 5, and the computation of electronic configuration of atoms, presented in Ch. 6.

Chapter 2

Implementation of Davidson Methods in SLEPc

SLEPc, the Scalable Library for Eigenvalue Problem Computations [Hernandez et al., 2005], is a parallel library that provides state-of-the-art algorithms and tools to solve large-scale eigenvalue problems, including linear eigenproblems as well as other related problems like singular value decompositions and quadratic eigenproblems. These problems appear frequently in different scientific disciplines, like for instance nuclear engineering, electromagnetics and electronic structure calculations, and demand large computational effort. In this context, it is necessary to make use of high-end computing platforms, such as clusters of computers, and, ever more, emerging hybrid architectures that combine multi-cores and accelerators.

In this chapter we focus on the (linear) generalized eigenvalue problem, that is, the computation of eigenvalue-eigenvector pairs $(\lambda_i, \mathbf{x}_i)$ satisfying the equation $A\mathbf{x}_i = \lambda_i B\mathbf{x}_i$, where A and B are square matrices, either real or complex. The case $B = I$ is often referred to as the standard eigenvalue problem. We will devote especial attention to symmetric (Hermitian) problems, where both A and B are symmetric (Hermitian) and B is positive (semi-)definite. Singular matrix pairs (A, B), that is, when A and B have a common null space, are not considered in this thesis. We are concerned with applications where A and B are large and sparse, and only a small percentage of the eigenvalues are required. The methods for this are iterative in nature, and compute approximate eigenpairs $(\theta_i, \tilde{\mathbf{x}}_i)$ such that the residual $\|A\tilde{\mathbf{x}}_i - \theta_i B\tilde{\mathbf{x}}_i\|$ is bounded by a given tolerance.

Among the methods available for addressing the above problem we can find: (i) classical methods such as the power iteration, subspace iteration and Rayleigh quotient iteration (RQI); (ii) Krylov methods such as Arnoldi, Lanczos and Krylov-Schur, that are particularly suitable for computing extreme eigenvalues; (iii) preconditioned conjugate gradient methods such as LOBPCG [Knyazev, 2001], that is efficient and robust for generalized symmetric eigenproblems; and (iv) David-

Table 2.1: Summary of parallel libraries providing Davidson-type solvers, with eigenproblem types supported by each method (HEP: standard Hermitian, GHEP: generalized Hermitian, GNHEP: generalized non-Hermitian).

Methods	Libraries			
	PRIMME	Anasazi	BLOPEX	SLEPc
Subspace Iteration	HEP			GNHEP
Rayleigh Quotient Iteration	HEP			GNHEP
LOBPCG	HEP	GHEP	GHEP	
Generalized Davidson	HEP	GHEP		GNHEP
Jacobi-Davidson	HEP	GHEP⋆		GNHEP

⋆ Anasazi requires the user to implement the Jacobi-Davidson correction equation.

son methods such as Generalized Davidson and Jacobi-Davidson, that prove being very efficient compared with the rest of the methods when computing interior eigenvalues. See [Bai et al., 2000; Stewart, 2001b; van der Vorst, 2002] for a more detailed overview of these methods.

Krylov methods are very popular for computing eigenvalues in the periphery of the spectrum, and are available in the form of parallel implementations such as ARPACK [Lehoucq et al., 1998]. The main drawback of these methods is that they often require to implicitly handle a matrix inverse, *e.g.*, when computing interior eigenvalues with the shift-and-invert technique [Ericsson and Ruhe, 1980]. This implies solving linear systems at each eigensolver iteration, and for this most often direct solvers must be employed to guarantee robustness. Methods belonging to the Davidson family try to overcome this limitation by relaxing the precision with which these inverses are approximated (in some cases with a simple preconditioner). As opposed to Krylov methods, general-purpose Davidson solvers are still difficult to find in freely available parallel software, especially with capabilities for non-symmetric and/or generalized problems, despite there being numerous publications developing the methods for different problem types (as §2.1 summarizes) and even describing parallel implementations tailored for certain applications [Heuveline et al., 1997; Nool and van der Ploeg, 2000; Arbenz et al., 2006; Genseberger, 2010; Hwang et al., 2010; Ferronato et al., 2012].

The main parallel libraries that provide eigensolvers similar to ours are summarized in Table 2.1: Anasazi [Baker et al., 2009], that includes a customizable block Generalized Davidson (that with a little coding can behave like a basic Jacobi-Davidson) and LOBPCG; BLOPEX [Knyazev et al., 2007], that provides a reference implementation of LOBPCG; and PRIMME [Stathopoulos and McCombs, 2010], that includes many methods that fit in the Generalized Davidson scheme, although currently without support for generalized problems. As Table 2.1 shows, none of the mentioned libraries support non-Hermitian problems.

This chapter presents our developments in SLEPc in order to provide state-of-the-art, robust, high performance Davidson solvers supporting Hermitian and

non-Hermitian eigenproblems, both standard and generalized, with either real or complex arithmetic.

In §2.1 we briefly summarize the theoretical background of the implemented Davidson methods and their variants. Section 2.2 starts with an overview of SLEPc and its foundation PETSc (Portable, Extensible Toolkit for Scientific Computation [Balay et al., 1997]), followed by an outline on the design of the Davidson solvers and a description of their user interfaces. Then in §2.3 we discuss how to implement some non-trivial aspects described in previous sections. In §2.4 we present sequential and parallel performance results of our implementation, with a number of test problems as well as a couple of real applications. Finally, we end with the conclusions.

2.1 Davidson-type framework

We present a review of the Davidson methods and related techniques that we have implemented in SLEPc. For an extensive bibliography about Davidson methods see [Bai et al., 2000; van der Vorst, 2002, 2004; Hochstenbach and Notay, 2006]. First we describe the main steps of this type of methods, and later subsections detail the variants of each step that are available in the implementation. Unfortunately, often there is no mathematical proof that indicates the best way of carrying out each step in the general case, that is, the optimal configuration is problem dependent. For that reason, it is customary that libraries offering Davidson methods provide several variants for each step and mechanisms to customize the execution.

2.1.1 General description of the Davidson-type methods

Subspace methods seek the eigenvectors in a low-dimensional search subspace, which is updated at every iteration. Davidson-type methods are a distinguished subclass of the subspace methods that expand the search subspace $\mathcal{V}$ in the directions of the computed corrections to the most wanted eigenvectors in the search subspace. In fact a Davidson variant is characterized by how to select the wanted eigenpairs in the search subspace (*extraction*) and how to compute the corrections (*expansion*). The convergence of the eigenpairs is tracked, for instance monitoring the norm of the residual associated to the approximate eigenpair, $\mathbf{r}_i$, or its correction, $\mathbf{d}_i$. When an eigenpair is considered converged it is removed from the search subspace and a *deflation* technique is used in order to prevent the convergence of the same pair afterward. When the dimension of the search subspace $\mathcal{V}$ grows up to a certain limit $m_{\max}$, it is reset to an $m_{\min}$-dimensional subspace $\mathcal{V}' \subset \mathcal{V}$ keeping as much useful spectral information as possible (*restart*). Algorithm 2.1 provides a general scheme of a Davidson-type method. We next discuss some general issues, and postpone the details of each step until later subsections, where the theoretical background of the different alternatives is introduced.

From the numerical point of view, working with an orthogonal basis of the search subspace V is desirable to control numerical error and also to maintain the whole set of vectors linearly independent after the addition of the correction

Algorithm 2.1: Basic Davidson-type Method

Input: matrices A and B of size n, number of wanted eigenpairs p, block size s, initial dimension of V m_0, maximum dimension of V $m_{\max}$, restart with $m_{\min}$ vectors

Output: resulting eigenpairs (Θ, X)

```
Choose a starting subspace basis V of m_0 vectors
Set m ← m_0, l ← 0, Θ ← [ ] and X ← [ ]
while l < p do
    Extraction: Compute the Ritz pairs (Θ̃, X̃) from the proj. eigenprobl.
    and sort them
    Test convergence, store the k converged pairs in (Θ, X) and remove
    them from (Θ̃, X̃)
    Set m ← m − k and l ← l + k
    if m ≥ m_max then
        Restart V with an m_min-dimensional subspace basis
        Set m ← m_min
    end
    Expansion: Compute the correction D of the first s pairs (Θ̃, X̃), and
    add them to V
    Set m ← m + s
end
```

vectors D. The extraction of approximations is based on a projection on this subspace, and it is also desirable to take into account any kind of structure (e.g., symmetry) present in the original problem in such a way that this structure is preserved in the projected problem. This has implications on how the Davidson method is realized. Therefore, the general scheme is specialized depending on the properties of the eigenproblem.

For instance, in a standard Hermitian or generalized Hermitian-definite problem (with B positive definite) we know that eigenvectors are B-orthogonal, so keeping a B-orthogonal basis V will result in a standard Hermitian problem coming out from the projection, if the Rayleigh-Ritz procedure is used. The deflation is performed by B-orthogonalizing the new vectors D against the previously converged eigenvectors X. This variant is detailed in Algorithm 2.2.

Algorithm 2.2 returns eigenvectors with unit B-norm. However, if B is numerically singular, *i.e.*, $|\mathbf{x}^* B\mathbf{x}|$ is close to zero for some eigenvector $\mathbf{x}$ (that is, an eigenvector corresponding to an infinite eigenvalue of the matrix pair (A, B)), enforcing B-normality may result in breakdown if a Ritz vector converges to $\mathbf{x}$. If $\alpha \in \mathbb{R}$ can be found such that $A - \alpha B$ is nonsingular, a straightforward solution would be to run Algorithm 2.2 on the matrix pair $(A - \beta B, A - \alpha B)$, that has the

Algorithm 2.2: Generalized Hermitian Davidson-type Method with B-orthogonalization

Input: matrices A and B of size n, preconditioner K, number of wanted eigenpairs p, block size s, initial dimension of V m_0, maximum size of V $m_{\max}$, restart with $m_{\min}$ vectors

Output: resulting eigenpairs (Θ, X)

Choose a starting subspace basis V of m_0 vectors, such that $V^*BV = I_{m_0}$
Set $m \leftarrow m_0$, $l \leftarrow 0$, $\Theta \leftarrow [\,]$ and $X \leftarrow [\,]$
while $l < p$ **do**
  Extraction: Compute the Ritz pairs $(\widetilde{\Theta}, \widetilde{X})$ by means of the Rayleigh-Ritz method, that is, solve $V^*AVU = U\widetilde{\Theta}$, where $U^*U = I_m$ and $\widetilde{X} = VU$
  Sort the Ritz pairs $(\widetilde{\Theta}, \widetilde{X})$
  Obtain the number of converged pairs k
  if $k > 0$ **then**
    Add eigenvalues $\widetilde{\theta}_1, \ldots, \widetilde{\theta}_k$ to Θ
    Set $X \leftarrow [X \quad \widetilde{X}_{1:k}]$ and $V \leftarrow \widetilde{X}_{k+1:m}$
    Set $m \leftarrow m - k$ and $l \leftarrow l + k$
  end
  if $m \geq m_{\max}$ **then**
    Choose an $m \times m_{\min}$-matrix M to reset V, $V \leftarrow VM$, such that $M^*M = I_{m_{\min}}$
    Update $(\widetilde{\Theta}, \widetilde{X})$ and U so that $V^*AVU = U\widetilde{\Theta}$, where $U^*U = I_{m_{\min}}$ and $\widetilde{X} = VU$
    Set $m \leftarrow m_{\min}$
  end
  Expansion: Compute the correction D of the first s pairs $(\widetilde{\Theta}, \widetilde{X})$
  $V \leftarrow [V \quad B\text{-orthonormalize}([X \quad V], D)]$ and set $m \leftarrow m + s$
end

same eigenvectors as the original problem and the eigenvalues

$$\tilde{\mu}_i = \frac{\tilde{\theta}_i - \beta}{\tilde{\theta}_i - \alpha}. \tag{2.1}$$

When the problem is not Hermitian, the implemented Davidson-type method works completely with generalized Schur decompositions, see Algorithm 2.3. The eigenvectors of non-Hermitian eigenproblems do not have to form an orthonormal set (nor B-orthonormal), so an algorithm that operates directly with them would be dangerous from the numerical point of view. In contrast, Schur decompositions are numerically more stable because of the use of orthonormal bases of invariant

Algorithm 2.3: Generalized non-Hermitian Davidson-type Method

Input: matrices A and B of size n, preconditioner K, number of wanted eigenpairs p, block size s, initial dimension of V m_0, maximum size of V $m_{\max}$, restart with $m_{\min}$ vectors

Output: resulting eigenvalues Θ and Schur vectors X

Choose a starting subspace basis V of m_0 vectors, such that $V^*V = I_{m_0}$
Compute W corresponding to V, such that $W^*W = I_{m_0}$
Set $m \leftarrow m_0$, $l \leftarrow 0$, $\Theta \leftarrow [\,]$, $X \leftarrow [\,]$ and $Y \leftarrow [\,]$
while $l < p$ **do**
    Extraction: Compute the Schur pairs $(\widetilde{\Theta}, \widetilde{X})$, from the Generalized Schur decomposition $W^*AV = ZSU^*$ and $W^*BV = ZTU^*$, where $\widetilde{X} = VU$ and $\widetilde{\theta}_i = s_{i,i}/t_{i,i}$
    Sort the Schur pairs $(\widetilde{\Theta}, \widetilde{X})$
    Obtain the number of converged pairs k
    if $k > 0$ **then**
        Add eigenvalues $\widetilde{\theta}_1, \ldots, \widetilde{\theta}_k$ to Θ
        Set $X \leftarrow [X \quad \widetilde{X}_{1:k}]$, $Y \leftarrow [Y \quad WZ_{1:k}]$
        Set $V \leftarrow VU_{k+1:m}$ and $W \leftarrow [W \quad WZ_{k+1:m}]$
        Set $m \leftarrow m - k$ and $l \leftarrow l + k$
    end
    if $m \geq m_{\max}$ **then**
        Choose an $m \times m_{\min}$-matrix M to reset V, $V \leftarrow VM$, such that $M^*M = I_{m_{\min}}$
        Update $(\widetilde{\Theta}, \widetilde{X})$, U and Z so that $W^*AV = ZSU^*$ and $W^*BV = ZTU^*$, where $\widetilde{X} = VU$ and $\widetilde{\theta}_i = s_{i,i}/t_{i,i}$
        Set $m \leftarrow m_{\min}$
    end
    Expansion: Compute the correction D of the first s pairs $(\widetilde{\Theta}, \widetilde{X})$
    Set $V \leftarrow [V \quad \text{orthonormalize}([X \quad V], D)]$
    Compute W^0 corresponding to $V_{m:m+s}$ and set $W \leftarrow [W \quad \text{orthonormalize}([Y \quad W], W^0)]$
    Set $m \leftarrow m + s$
end

subspaces, which also simplifies the task of deflation. Schur forms have the additional benefit of using quotients $s_{i,i}/t_{i,i}$ for representing the eigenvalues, thus handling infinite eigenvalues more naturally with $t_{i,i} = 0$.

Moreover, an additional advantage for the case of real non-symmetric eigenproblems is the use of real Schur forms, which work with real orthogonal bases of the invariant subspaces associated to the eigenvectors, that may be complex, hence avoiding complex arithmetic completely. If the generalized Schur form cor-

responding to a matrix pair $(\hat{A}, \hat{B})$ is the decomposition $\hat{A}U = ZS$ and $\hat{B}U = ZT$, where U and Z are unitary matrices and S and T are upper triangular matrices, in the real Schur form S and T are upper quasi-triangular (possibly with 2×2 diagonal blocks representing complex conjugate pairs of eigenvalues).

In [Fokkema et al., 1999] a Jacobi-Davidson method using generalized Schur forms is introduced (called JDQZ), whose main difference with respect to the version presented in this chapter is that vectors used for deflation of the search subspace (see §2.1.4) must be necessarily included in the projectors in the correction equation (see §2.1.3). In our implementation this is optional and, if necessary, deflation is completed when a new vector is added to the bases V and W.

Consider the partial generalized Schur decomposition $AX = YS$ and $BX = YT$ corresponding to already converged eigenpairs. The method considers that the Schur tuple $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, (\alpha, \beta))$, which comes from the Schur form of the projected problem in the extraction, has converged and must be appended to the decomposition, that is,

$$A\,[X \quad \tilde{\mathbf{x}}] = [Y \quad \tilde{\mathbf{y}}] \begin{bmatrix} S & \mathbf{s} \\ 0 & \sigma \end{bmatrix} \text{ and } B\,[X \quad \tilde{\mathbf{x}}] = [Y \quad \tilde{\mathbf{y}}] \begin{bmatrix} T & \mathbf{t} \\ 0 & \tau \end{bmatrix}. \tag{2.2}$$

So the only condition that the new tuple should satisfy is $X^*\tilde{\mathbf{x}} = Y^*\mathbf{y} = 0$ or, equivalently, that $X^*V = Y^*W = 0$, because $\tilde{\mathbf{x}} \in \text{span}\{V\}$ and $\tilde{\mathbf{y}} \in \text{span}\{W\}$. In the implementation this condition is enforced by maintaining the bases V and W orthogonal against X and Y, respectively. This variant is detailed in Algorithm 2.3.

Both Algorithm 2.2 and 2.3 support computing the correction of more than one approximate eigenpair, by increasing the value of parameter s (block size). However, in terms of convergence our experience shows that the optimal value for s is 1 (the same comment appears in [Stathopoulos and McCombs, 2007, §2.2.2]).

In step 5 of Algorithm 2.3 the generalized Schur decomposition is sorted so that the wanted pairs are located at the beginning of the decomposition (see [Kressner, 2006] and references therein). The first s pairs will be corrected at the next steps, and when restarting, the first $m_{\min}$ pairs determine the part preserved in V and W.

We conclude this subsection with a brief comment about the case of Hermitian-indefinite problems. When both A and B are Hermitian matrices, but B is indefinite, then Algorithm 2.2 cannot be used but it is still possible to exploit symmetry to some extent. If $B^{-1}A$ is non-defective, the eigenvectors X satisfy $X^*BX = I^{\pm}$, where $I^{\pm}$ is a signature matrix (diagonal with ± 1 elements on the diagonal). For this case, we have implemented a variant of Algorithm 2.3 where $W = V$, the orthogonalization is performed with respect to the B inner product (that is an indefinite inner product or a pseudo-inner product), and the resulting vectors are normalized so that $|\mathbf{x}^*B\mathbf{x}| = 1$. The corresponding projected problem is a generalized symmetric-indefinite eigenproblem and can be solved with a structure-preserving method like the one proposed in [Brebner and Grad, 1982]. However, since this solver is not available as a LAPACK subroutine, we currently use the Schur decomposition instead.

2.1.2 Subspace extractions

The extraction techniques considered in this section return a set of approximate eigenpairs $(\tilde{\theta}, \tilde{\mathbf{x}})$ whose vectors belong to the search subspace $\mathcal{V}$ spanned by the columns of V, that is, $\tilde{\mathbf{x}} = V\mathbf{u}$. The Rayleigh-Ritz approach is the most basic one and is considered useful for computing eigenvalues at the periphery of the spectrum (see [Stewart, 2001b, §4.4]). This technique imposes the Ritz-Galerkin condition on the residual associated to the returned pair,

$$\mathbf{r} := A\tilde{\mathbf{x}} - \tilde{\theta}B\tilde{\mathbf{x}} \perp \mathcal{V}, \tag{2.3}$$

which leads to the low-dimensional projected eigenproblem

$$V^*AV\mathbf{u} = \tilde{\theta}V^*BV\mathbf{u}. \tag{2.4}$$

From the solutions $(\tilde{\theta}, \mathbf{u})$ of (2.4), the returned pairs $(\tilde{\theta}, \tilde{\mathbf{x}}) = (\tilde{\theta}, V\mathbf{u})$ are obtained, which are called Ritz pairs.

The projected matrices V^*AV and V^*BV preserve the Hermitian structure of the problem matrices A and B, and in case the search subspace basis V is B-orthogonal, (2.4) will be a standard Hermitian eigenproblem, as indicated in line 4 of Algorithm 2.2.

However, the Rayleigh-Ritz approach generally returns poor approximate eigenvectors for interior eigenvalues (see, for instance, [Stewart, 2001b, Example 4.2]). A simple explanation for that is that the Ritz-Galerkin condition (2.3) does not consider the residual norm of the resulting eigenpairs (for a more detailed explanation see [Sleijpen et al., 1998, §4]). The harmonic Rayleigh-Ritz method [Morgan, 1991; Paige et al., 1995] was proposed instead as an alternative technique for interior eigenvalues, that imposes the Petrov-Galerkin condition to the residual of the returned pairs $(\tilde{\theta}, \tilde{\mathbf{x}})$

$$A\tilde{\mathbf{x}} - \tilde{\theta}B\tilde{\mathbf{x}} \perp \mathcal{W} := (A - \tau B)\mathcal{V}, \tag{2.5}$$

if eigenvalues close to τ are sought. Similarly to the previous case, this leads to the projected eigenproblem

$$V^*(A - \tau B)^*(A - \tau B)V\mathbf{u} = \xi V^*(A - \tau B)^*BV\mathbf{u}. \tag{2.6}$$

The solution pairs $(\xi, \mathbf{u})$ of (2.6) with smallest ξ correspond to the Ritz pairs $(\tilde{\theta}, \tilde{\mathbf{x}}) = (\tau + \xi, V\mathbf{u})$ closest to the target τ, which satisfy

$$\|A\tilde{\mathbf{x}} - \tau B\tilde{\mathbf{x}}\| \leq |\xi|\|B\tilde{\mathbf{x}}\|, \tag{2.7}$$

resulting from left-multiplying (2.6) by $\mathbf{u}^*$ and applying the Cauchy-Schwarz inequality. Hence it is sensible to think (at least for eigenvalues sufficiently close to τ) that selecting the pairs $(\tilde{\theta}, \tilde{\mathbf{x}})$ with $\tilde{\theta}$ closest to τ (*i.e.*, with smallest $|\xi|$) also correspond to the pairs with smallest residual norm.

Recently, two new variations of the harmonic extraction have been proposed. One is the relative harmonic extraction [Hochstenbach, 2005b] that finds eigenvalues θ with minimal

$$|\tilde{\theta} - \tau||\tilde{\theta}|^{-1} = |1 - \tau\tilde{\theta}^{-1}|, \tag{2.8}$$

that is, eigenvalues $\tilde{\theta}$ closest to τ considering the *weight* of $\tilde{\theta}$, in contrast to harmonic extraction. These approximate eigenpairs $(\tilde{\theta}, \mathbf{x})$ can be obtained by the constraint

$$A\tilde{\mathbf{x}} - \tau(1 - \tau\tilde{\theta}^{-1})^{-1}(A - \tau B)\tilde{\mathbf{x}} \perp \mathcal{W} := (A - \tau B)\mathcal{V}. \tag{2.9}$$

The resulting eigenpairs satisfy

$$\|A\tilde{\mathbf{x}} - \tau B\tilde{\mathbf{x}}\| \leq |1 - \tau\tilde{\theta}^{-1}|\|A\tilde{\mathbf{x}}\|. \tag{2.10}$$

The other one is specific for extracting the rightmost eigenvalues [Hochstenbach, 2005a], particularly when eigenvalues with large imaginary part are present in the spectrum. These eigenpairs are selected with the Galerkin condition,

$$(A + \bar{\tau}B)\tilde{\mathbf{x}} - \frac{\tilde{\theta} + \bar{\tau}}{\tilde{\theta} - \tau}(A - \tau B)\tilde{\mathbf{x}} \perp \mathcal{W} := (A - \tau B)\mathcal{V}, \tag{2.11}$$

and they satisfy

$$\|A\tilde{\mathbf{x}} - \tau B\tilde{\mathbf{x}}\| \leq \left|\frac{\tilde{\theta} - \tau}{\tilde{\theta} + \bar{\tau}}\right| \|(A + \bar{\tau}B)\tilde{\mathbf{x}}\|. \tag{2.12}$$

However our implementation follows the next approach. The Galerkin conditions associated to the harmonic extraction and their variants can be generalized in the parametrized Galerkin condition [Hochstenbach, 2005b, §5.1]

$$(\alpha A - \beta B)\tilde{\mathbf{x}} - \xi(\gamma A - \delta B)\tilde{\mathbf{x}} \perp \mathcal{W} := (\alpha A - \beta B)\mathcal{V}, \quad \text{with} \quad \xi = \frac{\alpha\tilde{\theta} - \beta}{\gamma\tilde{\theta} - \delta}. \tag{2.13}$$

The values of α, β, γ and δ corresponding to the extraction methods are detailed in Table 2.2. The projected eigenvalue problem associated to (2.13) is

$$W^*(\alpha A - \beta B)V\mathbf{u} = \xi W^*(\gamma A - \delta B)V\mathbf{u}, \quad \text{with} \quad (\alpha A - \beta B)V = WR, \tag{2.14}$$

where W has orthonormal columns and R is upper triangular. The error associated to the approximate eigenpairs is bounded by

$$\|(\alpha A - \beta B)\tilde{\mathbf{x}}\| \leq |\xi|\|(\gamma A + \delta B)\tilde{\mathbf{x}}\|. \tag{2.15}$$

This approach is concreted in Algorithm 2.3, by solving a problem equivalent to (2.14) at step 5, and computing W as a basis of $\alpha A - \beta B$ at steps 2 and 21. The implementation does it by setting

- $W \leftarrow$ orthonormalize$((\alpha A - \beta B)V)$ in step 2, and
- $W^0 \leftarrow (\alpha A - \beta B)V_{m:m+s}$ in step 21.

Table 2.2: Correspondence between the values of α, β, γ and δ in the generic Galerkin condition (2.13) and some extraction methods.

Extraction	α	β	γ	δ
Harmonic Rayleigh-Ritz	1	τ	0	1
Relative harmonic Rayleigh-Ritz	1	τ	1	0
Rightmost eigenvalue	1	τ	1	$-\bar{\tau}$
Largest eigenvalue	0	1	1	0

2.1.3 Subspace expansions

The first subspace expansion in the context of the Davidson methods was proposed with classical Davidson [Davidson, 1975]. The method tries to compute the smallest eigenpairs of a standard Hermitian problem expanding the search subspace with $\mathbf{d}$ satisfying,

$$(\mathrm{diag}(A) - \tilde{\theta} I)\mathbf{d} = \mathbf{r} := A\tilde{\mathbf{x}} - \tilde{\theta}\tilde{\mathbf{x}}. \tag{2.16}$$

Subsequent developments [Morgan and Scott, 1986; Natarajan and Vanderbilt, 1989; Morgan, 1990] tried to understand (2.16) as an iterative linear equation solver step of the system

$$(A - \tilde{\theta} I)\mathbf{d} = \mathbf{r}. \tag{2.17}$$

Then Generalized Davidson was proposed, a more general expansion also useful for non-Hermitian and generalized problems that introduced the use of fast, approximate inverses (the preconditioners, K),

$$\mathbf{d} = K^{-1}\mathbf{r}, \qquad K \approx A - \tilde{\theta} B. \tag{2.18}$$

The Olsen variant [Olsen et al., 1990] attempts to avoid the possible stagnation of the method when the resulting vector from the expansion $\mathbf{d}$ and the approximate eigenvector $\mathbf{x}$ are almost collinear (possibly because the preconditioner cannot improve the current approximation, see [Stathopoulos et al., 1995] for numerical experiments),

$$\mathbf{d} = -\left(I - \frac{K^{-1}B\tilde{\mathbf{x}}\tilde{\mathbf{x}}^*}{\tilde{\mathbf{x}}^* K^{-1} B\tilde{\mathbf{x}}}\right) K^{-1}\mathbf{r}. \tag{2.19}$$

In Jacobi-Davidson [Sleijpen and van der Vorst, 1996, 2000] the search subspace is expanded by the approximate solution of the Jacobi orthogonal correction equation, that obtains a correction $\mathbf{d}$ orthogonal to the selected approximate eigenvector $\tilde{\mathbf{x}}$. In [Sleijpen et al., 1996] it is extended to generalized eigenproblems (and polynomial problems) and adapted to the use of a preconditioner (see [Sleijpen et al., 1996, Theorem 7.3]),

$$PK^{-1}(A-\theta B)P\mathbf{d} = -PK^{-1}\mathbf{r}, \quad \text{with } \mathbf{d} \perp \mathbf{z} \qquad \text{where } P = I - \frac{K^{-1}\mathbf{y}\mathbf{z}^*}{\mathbf{z}^* K^{-1}\mathbf{y}}. \tag{2.20}$$

Also, some theorems have been proposed about the convergence speed depending on the values of $\mathbf{z}$ and $\mathbf{y}$:

- For A and B Hermitian and $\lambda \in \mathbb{R}$, the choice of $\mathbf{z} = \tilde{\mathbf{x}}$ and any $\mathbf{y}$ such that $\mathbf{y} \not\perp \tilde{\mathbf{x}}$, implies quadratic convergence [Sleijpen et al., 1996, Remark 3.4].
- The choice of $\mathbf{y} = B\tilde{\mathbf{x}}$ and $\mathbf{z}$ such that $\mathbf{z} \not\perp \tilde{\mathbf{x}}$ leads to quadratic convergence [Sleijpen et al., 1996, Theorem 3.2], also $\mathbf{y} = \tilde{\theta}A\tilde{\mathbf{x}} + B\tilde{\mathbf{x}}$ leads quadratic convergence [Sleijpen et al., 1996, Remark 3.1].
- The choice $\mathbf{y} = \mathbf{z} = \tilde{\mathbf{x}}$ implies superlinear convergence [Sleijpen et al., 1996, Theorem 3.4].

In SLEPc the correction equation is solved with $\mathbf{z}$ and $\mathbf{y}$ set to the approximate right eigenvector $\tilde{\mathbf{x}}$ and the corresponding vector from the test subspace, respectively. This solution can be found also in [Sleijpen et al., 1996; Fokkema et al., 1999].

2.1.4 Deflation

In works dealing with non-Hermitian problems and in early works of Jacobi-Davidson, the projector P in the correction equation (2.20) is extended to deflate also against the converged pairs (see [Sleijpen and van der Vorst, 1996; Fokkema et al., 1999])

$$P = I - K^{-1}\widehat{Y}(\widehat{Z}^*K^{-1}\widehat{Y})^{-1}\widehat{Z}^*, \quad \text{where } \widehat{Y} = [Y \quad \widetilde{Y}],\ \widehat{Z} = [X \quad \widetilde{X}]. \tag{2.21}$$

The extended projector avoids that the computed correction $\mathbf{d}$ converges toward the previously locked invariant subspace instead of new directions that could enrich the selected eigenvector. An extreme example is illustrated in [Fokkema et al., 1999, §4.5] in which the convergence is slowed down and finally stagnated due to the lack of newly produced directions.

The theoretical justification for this stagnation is presented next following [Fokkema et al., 1999, §3.4]. Consider the standard eigenvalue problem with matrix A and eigenpairs $(\hat{\lambda}_i, \hat{\mathbf{x}}_i)$ and the approximate eigenpair $(\tilde{\theta}, \tilde{\mathbf{x}})$ converging toward $(\hat{\lambda}_1, \hat{\mathbf{x}}_1)$. The exact solution of the correction equation (2.20) with $\mathbf{z} = \mathbf{y} = \tilde{\mathbf{x}}$ is given by [Sleijpen and van der Vorst, 1996, §4.1]

$$\mathbf{d} = -\tilde{\mathbf{x}} + \epsilon(A - \tilde{\theta}I)^{-1}\tilde{\mathbf{x}}, \qquad \epsilon = (\tilde{\mathbf{x}}^*(A - \tilde{\theta}I)^{-1}\tilde{\mathbf{x}})^{-1}. \tag{2.22}$$

Considering the projection of $\tilde{\mathbf{x}}$ onto $\hat{\mathbf{x}}_i$ and the constraint $\mathbf{d} \perp \tilde{\mathbf{x}}$ results in

$$\mathbf{d} \approx \sum_{i \neq 1} \frac{\hat{\mathbf{x}}_i^*\tilde{\mathbf{x}}}{\hat{\lambda}_i - \tilde{\theta}}\hat{\mathbf{x}}_i. \tag{2.23}$$

Hence the directions $\hat{\mathbf{x}}_i$ of eigenvalues $\hat{\lambda}_i$ closest to $\tilde{\theta}$ become dominant. In that way, the convergence rate of $(\tilde{\theta}, \tilde{\mathbf{x}})$ increases as $\tilde{\theta}$ gets closer to $\hat{\lambda}_1$. However, suppose that a good approximation to $(\hat{\lambda}_1, \hat{\mathbf{x}}_1)$ was obtained and the next closest pair $(\hat{\lambda}_2, \hat{\mathbf{x}}_2)$ is sought. If the two pairs are approximately at the same *distance*

to $(\tilde{\theta}, \tilde{\mathbf{x}})$, that is $|\hat{\lambda}_1 - \tilde{\theta}| \approx |\hat{\lambda}_2 - \tilde{\theta}|$ and $\hat{\mathbf{x}}_1^* \mathbf{x} \approx \hat{\mathbf{x}}_2^* \mathbf{x}$, or even worse, the first pair is the closest one, then we expect that $\mathbf{d}$ has many unwanted components of $\hat{\mathbf{x}}_1$. This can justify the implementation of the deflation in the correction equation.

On the other hand, in [Stathopoulos and McCombs, 2007] the authors discuss the correction equation without the presence of the converged vectors in the context of a method for standard Hermitian problems (called JDQMR-000), showing some examples where it outperforms the deflation discussed above. Notice that the cost of applying the projector with converged vectors can become expensive when many pairs are sought: a cost of $\mathcal{O}(kn)$ per application for the simplest projectors and up to $\mathcal{O}(kn + k^3)$ for oblique projectors, when k vectors are locked.

Our proposal is slightly more flexible and robust, allowing the user to limit the maximum number of converged vectors in the projector (a parameter called `pwindow` in §2.2.4). This approach can be useful in problems with presence of close eigenvalues, if a bound of the size of these clusters is available *a priori*.

2.1.5 Restarting

The maximum dimension of the subspace bases V and W is restricted with the parameter $m_{\max}$, limiting the cost of maintaining them orthogonal, that in general is one of the most expensive parts of the method, along with the matrix-vector product and the computation of the expansion. Too low values of $m_{\max}$ may prevent the convergence and too high values may affect the global performance negatively, and its optimal value depends on the application and the expansion employed.

From the methods that have been proposed to minimize the negative impact of restarting on convergence, we have incorporated in SLEPc (i) the thick restart technique [Stathopoulos et al., 1998], also called GD($m_{\min}$,$m_{\max}$), that restarts V with a subspace basis that contains the best $m_{\min}$ current approximate eigenvectors; and (ii) its combination with a generalized CG-based restart [Stathopoulos and Saad, 1998], resulting in GD($m_{\min}$,$m_{\max}$)+k, that enriches the thick restart basis with the best k vectors from the previous iteration. For seeking extreme eigenpairs in standard Hermitian problems, the latter technique has theoretical [Stathopoulos and Saad, 1998, §4] and practical [Stathopoulos and Saad, 1998; Stathopoulos, 2007] justifications.

2.1.6 Initializing the search subspace

An interesting advantage of the unconstrained Davidson's search subspace (in opposition to structured subspaces, like in the case of Krylov methods) is the possibility of starting with an available rough approximation of the sought eigenvectors, for instance in applications where the solution of a previous similar eigenproblem can be used or an analytic solution of a simpler problem is provided.

However, in practice initializing the search subspace with initial solutions X_0 is not enough to improve the convergence unless it is sufficiently close to the exact solutions. Alternatively, better results may be obtained by initializing the search subspace with a Krylov subspace generated by the operator $K^{-1}(A - \tau B)$ and

X_0. An example of use and performance is given in the applications of §2.4.2 and §2.4.3.

2.2 SLEPc eigensolver design

The two following subsections depict SLEPc, the library in which the previously exposed variants of the Davidson methods are implemented, and PETSc, the framework on which SLEPc relies. Then related software efforts for solving large-scale problems with Davidson solvers are presented. And the section ends with a detailed description of the user interface of the Davidson solvers in SLEPc.

2.2.1 PETSc description

PETSc provides implementations of basic linear algebra operations with vectors and matrices, as well as tools for the solution of linear and non-linear systems of equations, all this enhanced with the support for distributed memory parallel computing platforms as well as an incipient support for emerging architectures such as shared memory clusters and GPUs. The interface is object-oriented, there are sets of functions in C (the class methods) accepting the same data type (representing a class), which encapsulates a pointer to a data structure of the class (that stores the object). The main advantage of this simple approach, over using a language with object-oriented support like C++, is the simplicity to invoke C functions from other languages. In fact PETSc gives support to C++, Fortran, Python and Matlab.

In general, the algorithm or the variant that will be used is determined by the *type*, that is set after the creation of the object. For instance, some of the available types for the KSP class, aimed at solving linear systems, are CG, GMRES and BiCGStab(ℓ).

The Vec and Mat classes gather different data structures for vectors and matrices. All vector objects are dense, but the Mat class offers a wide variety of matrix types like dense and sparse matrices, block diagonal matrices, FFT and user-defined implicit matrices, among others. Some types of vectors and matrices support high performance hardware, such as VecMPI, that distribute the vector and operations among MPI processes, and VecPThread and VecCUSP, that perform the operations by its distribution among POSIX threads and by launching GPU kernels, respectively. Types with similar capabilities can also be found in Mat.

Hence the parallelism and other high performance mechanisms are mostly transparent for other objects built on top of these, and also for the application programmer. The rest of classes such as linear, non-linear and time-stepping solvers, implement their methods without taking care of the underlying data-structures in matrices and vectors. The exception is the preconditioners class (PC), in which many of the types correspond to complete or partial factorizations (*e.g.*, LU, ILU, Cholesky and ICC) that are stored in ad-hoc Mat implementations.

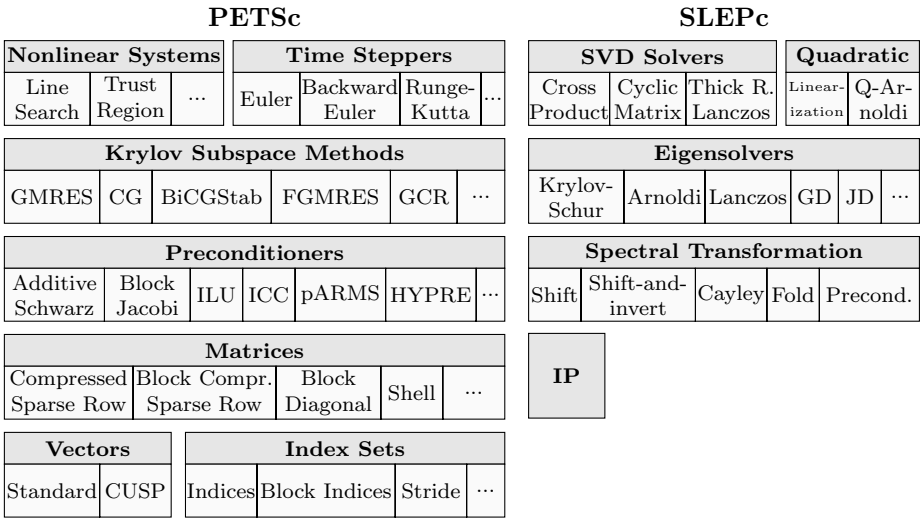


Figure 2.1: Main classes in PETSc and SLEPc.

For more information about the internal structure and the provided interfaces see [Balay et al., 1997, 2011].

2.2.2 Internal structure of SLEPc

SLEPc provides a collection of state-of-the-art eigensolvers, SVD solvers and quadratic eigensolvers. Besides the Davidson-type solver addressed in this chapter, the class EPS provides an implementation of the Arnoldi, Lanczos and Krylov-Schur methods as well as wrappers to other libraries such as ARPACK, PRIMME and BLOPEX. Spectral transformations such as the shift-and-invert technique are encapsulated in the class ST, and are needed by Krylov methods to compute interior eigenvalues or to deal with matrix inversion in generalized eigenproblems. Class SVD provides algorithms for computing a partial singular value decomposition, for instance by solving an equivalent eigenvalue problem via EPS or by implementing native methods such as restarted Lanczos bidiagonalization. In the same way QEP objects can solve quadratic eigenproblems by using EPS to solve an eigenvalue problem resulting from linearization or directly with the Q-Arnoldi method.

Like other high-level classes, these SLEPc classes are built on the foundational classes Vec and Mat, and also KSP for the spectral transformations (matrix inversion is implemented via linear solves). As explained in §2.3.1, KSP will also be used in the solution of the correction equation in Jacobi-Davidson, and the preconditioner in the Generalized Davidson expansion. Orthogonalization and B-orthogonalization methods (like classical and modified Gram-Schmidt, optionally with selective reorthogonalization) are encapsulated in the class IP. For a complete

description of SLEPc classes and usage, the reader is referred to [Campos et al., 2011].

PETSc and SLEPc delegate many dense operations to BLAS and LAPACK compatible libraries, such as the netlib reference implementation[1] or the ATLAS, MKL and ACML libraries. The delegated operations include frequent computations that account for most of the floating-point operations, like the addition and product of vectors during orthogonalization. Other delegated operations are performed on the projected problems, usually of rather small size, and they are important in terms of robustness of the algorithms, such as the computation of Schur decompositions in the extraction step of the algorithms in §2.1.

Furthermore SLEPc provides an interface for specifying operations with multivectors (a set of vectors that represent the columns of a thin tall matrix), giving the possibility to accelerate a sequence of level-2 BLAS operations by rewriting them as level-3 operations. The mechanism behind this consists in some functions to create PETSc Vec objects whose entries are stored contiguously in memory and to implement operations between such multivectors or multivectors with dense matrices. For instance, these functions are employed in the Davidson solvers in the creation of the projected matrices and the orthogonalization of the subspaces. As discussed below, interfaces to multivector operations are common in other libraries with Davidson methods.

2.2.3 Comparison with the design of other parallel Davidson software

Part of the design principles of Davidson methods in SLEPc can also be found in other libraries that implement related methods and share similar high performance aims, in particular Anasazi [Baker et al., 2009], BLOPEX [Knyazev et al., 2007] and PRIMME [Stathopoulos and McCombs, 2010].

Anasazi is part of Trilinos, a parallel object-oriented software framework for large-scale multi-physics scientific applications. It was designed for being independent of the particular implementation of the underlying linear algebra primitives, in order to facilitate its incorporation into larger libraries and application codes. The Anasazi package contains a collection of eigensolvers that includes block Generalized Davidson and LOBPCG, among other solvers not related with Davidson. The eigensolvers provide an interface to establish the concrete implementations that will be used for vector and matrix operations, for the orthogonalization and for the stopping criterion.

BLOPEX is a stand-alone library that includes only a LOBPCG solver. The library is also distributed as part of the HYPRE[2] library for parallel preconditioning, and can optionally be used as a external solver in SLEPc. BLOPEX is designed with independence and interoperability goals similar to Anasazi, but without the possibility for customizing the orthogonalization method and the stopping criterion.

[1] `http://netlib.org/blas/` and `http://netlib.org/lapack/`.

[2] `http://www.llnl.gov/casc/hypre`.

PRIMME implements a parametrized Davidson-type method, general enough to include algorithms ranging from Subspace Iteration to Jacobi-Davidson with several options about the correction equation. It uses its own distributed vectors with the solely support of BLAS and LAPACK, and a user-provided sum reduction operation. Currently, PRIMME only supports standard eigenproblems, although the interface is prepared for a matrix B. SLEPc also provides a wrapper to PRIMME as a external solver.

Table 2.3 summarizes the differences among these three libraries and our approach.

SLEPc's design goals lie between Anasazi/BLOPEX and PRIMME. On one hand, SLEPc eigensolvers use vectors, matrices and linear algebra primitives from PETSc. However, this PETSc-dependence does not reduce the software interoperability because PETSc classes allow for user-defined implementations. Anasazi and BLOPEX have an interface for multivector operations, and as mentioned before, SLEPc provides a simple interface for multivector operations such as inner-product W^*V and the update VU, where V and W are multivectors and U is a dense matrix.

The orthogonalization is also encapsulated in a SLEPc class and decoupled from the eigensolvers. Another simple mechanism to make the implementation independent of a given operation is to use callback functions. This is the way the convergence test and the sort routine can be replaced by user-defined code.

On the other hand, the code of Davidson-type methods in SLEPc is organized as a parametric multi-method, following PRIMME's approach. Hence, in practice, there is only one abstract object that contains all the Davidson code. By changing values in a C structure, the implementation behaves like either Algorithm 2.2 or 2.3. In the same way, the structure has variables to select the extraction and the expansion methods, to configure the restarting and to build the initial subspace.

2.2.4 Davidson-type eigensolvers interface

SLEPc offers two EPS solvers that implement Davidson-type methods, the Generalized Davidson (GD) and the Jacobi-Davidson (JD). As mentioned above, these objects are simply intermediate objects that access another abstract EPS object (abstract, in the sense that the end user cannot directly use it) that contains the implementation of the methods and techniques described in §2.1.

The JD and GD solvers share the following interface with the rest of EPS objects:

- the problem matrices, with `EPSSetOperators`;
- the problem type like Hermitian, non-Hermitian, standard or generalized, with `EPSSetProblemType`, that selects between Algorithms 2.2 and 2.3;
- the subspace expansion method, determined by the object type with `EPSSetType`, that is (2.19) for GD and (2.20) for JD;

Table 2.3: Summary of differences among BLOPEX, Anasazi, PRIMME, and SLEPc's Davidson, considering the implementation of the linear algebra operations (Vectors and Matrices), the possibility to change the orthogonalization routine (Orth.), the convergence test (Conv.) and the sort function (Sort), and how the Davidson solvers are organized.

Framework	**Vectors and Matrices**	**Customize**			**Eigensolvers**
		Orth.	**Conv.**	**Sort**	
BLOPEX	Abstract classes	-	-	-	Unique
Anasazi		X	X	X	Separate
PRIMME	Hard code implementation	-	-	-	Parametrized Davidson Eigensolver
SLEPc	PETSc	X	X	X	

- the subspace extraction method, with `EPSSetExtraction`, that selects between the extraction techniques presented in §2.1.2;
- the sorting criterion for the eigenpairs computed by the extraction method, with `EPSSetWhichEigenpairs`, which can be relative to a target value, or with respect to the magnitude, the real or the imaginary part of the eigenvalue;
- the target value τ if interior eigenvalues are wanted, with `EPSSetTarget`, parameter that is also used in the extraction;
- the maximum size of the search and testing subspaces, $m_{\max}$ (see §2.1.5), and the number of wanted eigenpairs, p, with `EPSSetDimensions`, called `mpd` and `nev`, respectively;
- the convergence criterion, with `EPSSetConvergenceTest`;
- the tolerance for the convergence criterion and the maximum number of (outer) iterations, with `EPSSetTolerances`;
- the initial guess of the wanted invariant subspace, with `EPSSetInitialSpace` (see §2.1.6); and
- the subspace in which the eigenvectors are not wanted, with `EPSSetDeflationSpace` (see §2.1.4).

The specific options that are shared by the Davidson solvers are (the `*` has to be substituted by `GD` or `JD`, because there are different function names for GD and JD, for instance `EPS*SetBlockSize` is `EPSGDSetBlockSize` for GD and `EPSJDSetBlockSize` for JD):

- whether to enrich the initial subspace as explained in §2.1.6, with `EPS*SetKrylovStart`;
- the block size s, with `EPS*SetBlockSize`;
- the size of the subspace after restarting $m_{\min}$ and GD+k, with `EPS*SetRestart` (see §2.1.5);
- the size of the initial search subspace, with `EPS*SetInitialSize` (if the number of initial vectors provided by the user is smaller than this number, the initial subspace is filled with random vectors); and
- the maximum number of converged vectors to be included in the projectors (parameter `pwindow`, see §2.1.4), with `EPS*SetWindowSizes`;

Moreover, the JD solver has two advanced options more that affect the convergence and performance of the solution of the correction equation, `EPSJDSetFix` and `EPSJDSetConstantCorrectionTolerance`, discussed in §2.3.1.

All preconditioned eigensolvers in SLEPc (JD and GD, but also the BLOPEX and PRIMME wrappers) are used in combination with the special ST object `Precond`. As other ST's (including shift-and-invert), it handles a linear solver (KSP) internally. The PC object in `Precond`'s KSP corresponds to the preconditioner used in the Generalized Davidson expansion or to the acceleration of the KSP that solves the Jacobi-Davidson correction equation. If the user does not provide a matrix as a basis for the preconditioner, with `STPrecondSetMatForPC`, the default preconditioner is built from $A - \tau B$ if interior eigenvalues are wanted, and B if largest magnitude eigenvalues are wanted. The method and the options for solving the correction equation are set in the KSP object. The default iterative solver for the JD correction equation is BiCGStab(2).

After the call to `EPSSolve`, the user can get individual converged eigenpairs with `EPSGetEigenpair`, or an orthogonal basis of the invariant subspace associated to them with `EPSGetInvariantSubspace`. Moreover, a basis of the test subspace associated to the converged pairs is accessible by the function `EPSGetInvariantSubspaceLeft`. Although not guaranteed, the test subspace may be a rough approximation to the left invariant subspace (if future versions of SLEPc include two-sided Jacobi-Davidson, more accurate approximations of the left invariant subspace will be obtained).

Figure 2.2 illustrates a simple example in C that computes the largest eigenvalue of a non-Hermitian matrix A, omitting the creation of the matrix, as well as the error checking and the functions to initialize and finalize the SLEPc environment. Notice that if PETSc is built in real arithmetic (that is, `PetscScalar` is not a complex type), SLEPc returns the real and imaginary part of eigenvalues (`kr` and `ki`) and eigenvectors (`xr` and `xi`) separately. In order to use some Davidson solver, a call to `EPSSetType(GD)` or `EPSSetType(JD)` has to be included just before line 11.

```
#include "slepceps.h"
EPS          eps;       /*  eigensolver context  */
Mat          A;         /*  matrix of Ax=kx      */
Vec          xr, xi;    /*  eigenvector, x       */
PetscScalar  kr, ki;    /*  eigenvalue, k        */
PetscInt     j, nconv;

EPSCreate( PETSC_COMM_WORLD, &eps );
EPSSetOperators( eps, A, PETSC_NULL );
EPSSetProblemType( eps, EPS_NHEP );
EPSSetFromOptions( eps );
EPSSolve( eps );
EPSGetConverged( eps, &nconv );
for (j=0;j<nconv;j++)
  EPSGetEigenpair( eps, j, &kr, &ki, xr, xi );
EPSDestroy( eps );
```

Figure 2.2: The shortest SLEPc example code that compute the eigenvalues of a non-Hermitian matrix.

Most of the above options are accessible via the command line. For instance, the code in Figure 2.2 can employ the GD solver to compute 10 eigenvalues with a relative tolerance of 10^{-8} by the execution of

```
$ ./exe -eps_type gd -eps_nev 10 -eps_tol 1e-8
```

or can employ JD for seeking the eigenvalues closest to 1 solving the correction equation with BiCGStab(2) and accelerated with an ILU preconditioner:

```
$ ./exe -eps_type jd -eps_nev 10 -eps_target 1 -st_ksp_type bcgsl \
        -st_ksp_bcgsl_ell 2 -st_pc_type ilu
```

This feature facilitates the process of manually tunning a code to find optimal configurations.

2.3 Implementation details

In this section we include some discussions about the way we have implemented certain variants or aspects of the Davidson methods, such as the solution of the Jacobi-Davidson correction equation by an iterative Krylov solver (a KSP object), the treatment of complex numbers in real problems, how the data structures are distributed across processors and some considerations about the memory management.

2.3.1 Solution of the correction equation

Linear system (2.20) is solved when the Jacobi-Davidson expansion is selected. For that, a PETSc Krylov linear solver is employed (a KSP object), accelerated with preconditioners (PC object). Practical aspects of how to modify the correction equation and the projectors in order to use a preconditioner are summarized below (details can be found in [Sleijpen et al., 1998, §3.2]).

Most KSP solvers available in PETSc support left preconditioning (many of them exclusively), that is, they solve the linear system $M^{-1}\hat{A}\mathbf{x} = M^{-1}\hat{\mathbf{b}}$. In that case, (2.20) is equivalent to a linear system with coefficient matrix $\hat{A} = PK^{-1}(A - \theta B)$ and right hand side $\hat{\mathbf{b}} = -PK^{-1}\mathbf{r}$. If the initial guess is a zero vector, then the obtained approximate solution $\mathbf{d}$ will satisfy the constraint $\mathbf{d} \perp \mathbf{z}$.

The most frequently used KSP solver supporting right preconditioning (*i.e.*, to solve instead $\hat{A}M^{-1}M\mathbf{x} = \hat{\mathbf{b}}$) is FGMRES [Saad, 1993], that is employed when the preconditioner can change during the iteration, for instance, when it performs a nested linear solve iteration as in pARMS [Li et al., 2003]. In the right preconditioning case, (2.20) is equivalent to a linear system with coefficient matrix $\hat{A} = Q(A - \tau B)$ where Q can be P or $I - \mathbf{z}\mathbf{z}^*$, the preconditioner $M^{-1} = PK^{-1}$ and right hand side $\hat{\mathbf{b}} = -\mathbf{r}$.

In both cases, the operator $\hat{A}$ and the preconditioner M^{-1} are implemented implicitly (that is, only the matrix-vector product is defined) using a MatShell and a PCShell, respectively (these *shell* constructs are simply a way to encapsulate user-defined operations as a PETSc object).

In terms of computational cost, the weight of the solution of the correction equation can be heavy if too accurate solutions are requested. The trade-off between performance and global convergence is controlled in the stopping criterion, that the user can configure by setting the maximum number of iterations besides the relative and the absolute tolerances. In addition, unless `EPSJDSetConstantCorrectionTolerance` is invoked, the KSP stops when the residual of the linear system at iteration j, $\|\hat{\mathbf{r}}^{(j)}\|_2$, satisfies at outer iteration i

$$\|\hat{\mathbf{r}}^{(j)}\|_2 \leq 2^{-i}\|\hat{\mathbf{r}}^{(0)}\|_2. \tag{2.24}$$

Notice that in PETSc the stopping criterion uses the preconditioned residual by default. This dynamic criterion comes from the Newton methods and the use in Jacobi-Davidson is suggested in [Fokkema et al., 1999], and tested in [Genseberger, 2010; Romero and Roman, 2011].

It is well-know that in the first iterations of the Davidson-type methods the extraction method usually produces poor eigenpair approximations and the target τ may be a relatively closer approximation to an exact eigenvalue [Morgan and Scott, 1986]. Therefore until the residual norm associated to the selected eigenpair reaches a threshold value so-called *fix* (that can be set by `EPSJDSetFix`), the correction equation is solved with $\theta = \tau$ [Fokkema et al., 1999].

2.3.2 Real arithmetic

In real non-symmetric problems the eigenvalues and the corresponding eigenvectors may be complex, or even in the convergence of a real eigenpair the approximate eigenpairs may be complex. Considering that PETSc does not support operations mixing real and complex matrices or vectors, a simple workaround would be to perform all the computations in complex arithmetic. However, this is wasteful not only in terms of memory requirements but also in computational efficiency since the effective operation throughput may be reduced up to 50% for sufficiently large problems in which the bottleneck is the bandwidth between the main memory and the processor.

Another possibility to avoid complex arithmetic is the use of real Schur decompositions, where all matrices involved are real and 2×2 blocks on the diagonal of quasi-triangular matrices are used to represent complex conjugate eigenvalues. Besides requiring half the storage of a complex Schur decomposition, another advantage of the real Schur form is that complex conjugate eigenvalues always appear together.

Thus SLEPc's Jacobi-Davidson has an implementation based on RJDQZ [van Noorden and Rommes, 2007], that adapts the extraction process and the correction equation (2.20) to work with the real Schur form. Considering the preconditioning, the resulting correction equation for the selected complex conjugate eigenvalues $\tilde{\theta}^r \pm \tilde{\theta}^i i$ and the associated Schur vectors $\tilde{\mathbf{x}}_1$ and $\tilde{\mathbf{x}}_2$ is

$$P^B \begin{bmatrix} K^{-1}(A - \tilde{\theta}^r B) & \tilde{\theta}^i K^{-1} B \\ -\tilde{\theta}^i K^{-1} B & K^{-1}(A - \tilde{\theta}^r B) \end{bmatrix} P^B \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{d}_2 \end{bmatrix} = -P^B \begin{bmatrix} K^{-1}\mathbf{r}_1 \\ K^{-1}\mathbf{r}_2 \end{bmatrix}, \tag{2.25}$$

where P^B contains the block version of the projector P in (2.20),

$$P^B = \begin{bmatrix} \hat{P}^B & 0 \\ 0 & \hat{P}^B \end{bmatrix}, \qquad \hat{P}^B = I - K^{-1}Y(Z^* K^{-1} Y)^{-1} Z^*, \tag{2.26}$$

and the residual is computed as

$$\begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{bmatrix} = \begin{bmatrix} A - \tilde{\theta}^r B & \tilde{\theta}^i B \\ -\tilde{\theta}^i B & A - \tilde{\theta}^r B \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}}_1 \\ \tilde{\mathbf{x}}_2 \end{bmatrix}. \tag{2.27}$$

This version of the correction equation admits the same stopping criterion as (2.20), because if $\mathbf{d}_1$ and $\mathbf{d}_2$ are obtained from the approximate solution of the correction equation (2.25) with a linear system residual norm ε, then also $\mathbf{d} = \mathbf{d}_1 + \mathbf{d}_2 i$ satisfies the correction equation (2.20) with the same residual norm tolerance. Moreover we do not expect a significant difference in the convergence of the iterative resolution of both equations because the condition number corresponding to the coefficient matrices are the same [van Noorden and Rommes, 2007, Proposition 1].

If Jacobi-Davidson is activated, the KSP in ST is responsible for solving the correction equation, that can be

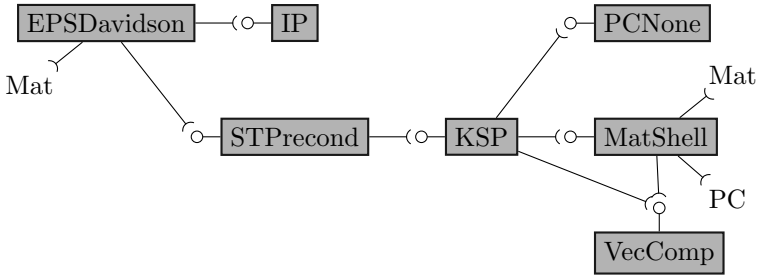


Figure 2.3: Collaboration diagram for the SLEPc's Davidson solver when the Jacobi-Davidson expansion is selected.

- the linear system (2.20), if either the problem and the selected eigenpair are both real or the problem is complex, or
- the double-sized linear system (2.25), if the problem is real, but the selected eigenpair is complex.

In the case of real problems, the KSP object would have to solve linear systems of different sizes, which implies reallocating the memory every time the system size changes. This problem can be neglected if the block size is one and the eigenpairs last many iterations before their convergence, because in general we do not expect an excessive number of jumps of a single approximate eigenvalue from real to complex (and vice versa). For applications that require many eigenvalues and they converge quickly, we have designed a special vector type VecComp with *virtually* the length of the double-sized system, which is employed by the KSP object. VecComp vectors are formed by two sub-vectors whose length is equal to the problem size, and only one is used when the approximate eigenvalue is real. Working with these vectors also simplifies the implementation of the double-sized coefficient matrix-vector product (which is implemented using the MatShell PETSc class, because the double-sized matrix is never built explicitly), that can be arranged as a 2-by-2 block product. This product also includes the projection and the preconditioner application. For that, the PC associated to the KSP object is detached and replaced by a dummy PC (PCNone). Figure 2.3 depicts the component diagram in this case.

2.3.3 Parallelization and memory management details

The problem matrices A and B and the vectors of the same size, such as the search and test subspaces V and W, and the converged invariant subspace $\widetilde{X}$, are distributed by blocks of rows among the processes, in the case of using MPI objects for vectors and matrices. The rest of the matrices and vectors with smaller size (bounded by $m_{\max}$) are implemented as sequential (non-MPI) objects, but are replicated in all nodes. For instance, this is the case of the projected matrices and the associated decompositions Θ, U, Z, S and T.

Algorithm 2.4: Optimized Iterative Classical Gram-Schmidt with B-inner product.

Input: matrix B of size n, B-orthonormal basis V, $V^B = BV$, vector $\mathbf{v}_0$
Output: B-orthonormal basis $[V\ \mathbf{v}]$ of span$\{[V\ \mathbf{v}_0]\}$ and $\mathbf{v}^B = B\mathbf{v}$

$\mathbf{v} \leftarrow \mathbf{v}_0$ and $\mathbf{v}^B \leftarrow B\mathbf{v}_0$
for $i = 1, 2, 3$ **do**
  $\mathbf{h} \leftarrow V^*\mathbf{v}^B$
  $\mathbf{v} \leftarrow \mathbf{v} - V\mathbf{h}$
  $\mathbf{v}^B \leftarrow \mathbf{v}^B - V^B\mathbf{h}$
  Test criterion
end
$\mathbf{v} = \mathbf{v}/\|\mathbf{v}\|_B$

The operations involving distributed operands are parallelized, such as the A and B matrix-vector product, the subspaces updating, the computation of the projected problem matrices and the orthogonalization of the subspaces. In addition, these operations can also be accelerated by the GPU or parallelized for taking advantage of multi-cores, if the appropriate types are set for matrices and vectors. Currently these features are experimental in PETSc, but in the near future they are expected to be fully functional.

In terms of memory management, nearly all memory required by the eigensolver (the bases of the search and test subspaces, auxiliary vectors and work spaces) is allocated in a contiguous array, in order to reduce the memory management overhead. Each piece in which the allocated memory is divided is aligned properly.

2.3.4 Subspace orthogonalization

The orthogonalization of vectors can become an expensive step in variants with cheap expansion, *e.g.*, Generalized Davidson, so it is important to use a procedure that is efficient both sequentially and in parallel. The SLEPc class IP provides Classical and Modified Gram-Schmidt to carry out that process. The default configuration (used in §2.4) employs iterative CGS (which yields better parallel scaling and higher floating point operations throughput than MGS), with a DGKS-like re-orthogonalization criterion. See [Hernandez et al., 2007] for details.

This configuration is also used for maintaining B-orthogonal bases, although it is not clear that the re-orthogonalization criterion can be useful when using B-inner products with ill-conditioned B [Kopal et al., 2011] (to avoid problems, it is possible in SLEPc to deactivate selective re-orthogonalization and use double orthogonalization instead). In any case, we present in Algorithm 2.4 a variant of iterative CGS that avoids the extra matrix-vector products when the search subspace is B-orthogonalized, at the cost of additional storage for vectors BV.

Table 2.4: Number of converged cases in each experiment.

Solver	Total	None	Jacobi	ILU/ICC	HYPRE	pARMS
Experiment I: GNHEP, largest magnitude eigenvalues						
GD	218	14	24	70	54	56
JD BiCGStab(2)	273	35	41	69	70	58
Experiment II: NHEP and GNHEP, eigenvalues closest to target						
GD	599	81	86	186	104	142
JD GMRES	280	21	42	94	43	80
JD BiCGStab(2)	490	90	100	143	77	80
Experiment III: HEP, eigenvalues closest to target						
GD	854	279	304	271		
PRIMME JDQMR_Etol	965	423	233	309		
JD BiCGStab(2)	895	280	358	257		

2.3.5 Convergence criterion for the eigensolver

SLEPc monitors the residual norm associated to the approximate eigenpairs in order to detect the converged ones with respect to some criterion, *e.g.*, $\|r\|_2 \leq \varepsilon$ for a given tolerance ε, or other criteria that can be defined by the user. When the problem is non-Hermitian, the solver works with approximate Schur vectors instead of eigenvectors, so the residual norms associated to the eigenpairs are not readily available. Our implementation first checks the convergence criterion with the residual associated to the Schur vector, and if it is passed, the criterion is checked again with the corresponding eigenvector residual. In this way, we avoid the costly computation of the eigenvector in each iteration unless it is close to convergence.

2.4 Results

In this section, we present performance results of the SLEPc Davidson solvers. First sequential results are shown comparing our implementations of GD and JD, as well as with PRIMME solvers, in the context of a collection of small eigenproblems. Then we show parallel results in the context of two large-scale scientific applications.

2.4.1 Test battery

We compare the Davidson solvers in terms of execution time and number of preconditioner applications required by each one in the solution of a collection of problems.

The collection consists of standard and generalized problems whose matrices come from the University of Florida Sparse Matrix Collection [Davis and Hu, 2011]. For each problem, the sequential performance of the solvers, in terms of number of matrix-vector products and time spent, is obtained computing 1 and 10

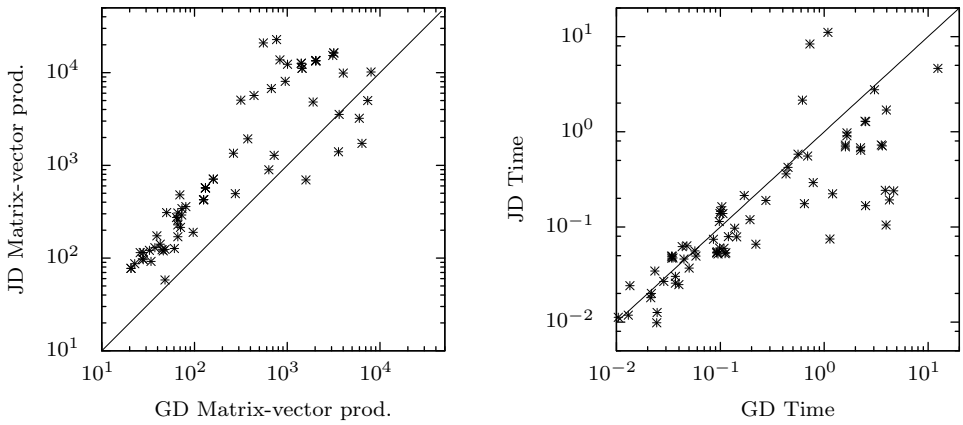


Figure 2.4: Comparison of number of matrix-vector products (left) and execution time in seconds (right) between SLEPc GD and JD computing the largest magnitude eigenvalues of generalized non-Hermitian problems. A mark above the line corresponds to an experiment with GD performing better than JD.

pairs with a relative tolerance of 10^{-10} using either no preconditioner or one of the PETSc preconditioners Jacobi, ILU (or ICC for Hermitian problems), HYPRE and pARMS. The rest of the solver's parameters keep the default values: 5 random vectors as initial subspace, restart the search subspace with 6 vectors after the dimension reaches 17 or 26, respectively for seeking 1 or 10 pairs. The default linear solver for the JD correction equation is BiCGStab(2). Each of these configurations is executed twice with three different initial random vectors, discarding the slower one.

Table 2.4 summarizes the number of cases in which the solvers obtained all the requested eigenpairs with less than 5050 and 10100 matrix-vector products for standard and generalized problems, respectively. We use matrix-vector products because it is unfair to compare GD and JD in terms of the number of outer iterations. Instead, it is more natural to roughly compare the number of iterations used by GD with the accumulated number of inner iterations employed by JD.

Experiment I takes results of the SLEPc solvers GD and JD computing the largest magnitude eigenpairs of a group of 25 generalized non-Hermitian problems. JD solves slightly more problems than GD, and many of them faster (see Figure 2.4, right). However, in terms of matrix-vector products, JD generally requires more products than GD (see Figure 2.4, left).

Experiment II collects the performance of the solvers with a group of 52 standard and generalized non-Hermitian problems while computing the eigenvalues

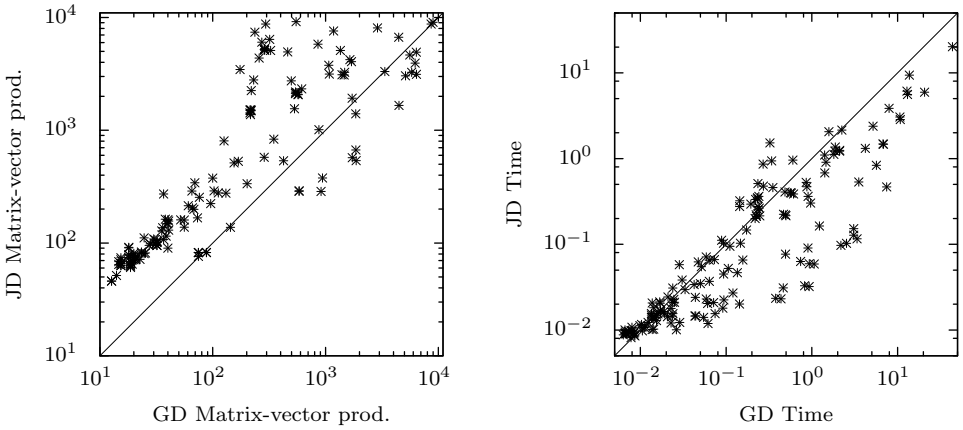


Figure 2.5: Comparison of number of matrix-vector products (left) and execution time in seconds (right) between SLEPc GD and JD computing the eigenvalues closest to a target in non-Hermitian problems. A mark above the line corresponds to an experiment with GD performing better than JD.

closest to a target in the interior of the spectrum. In absolute terms, GD successfully converges in more cases than JD (599 vs 490). In part, this is due to the few converged problems obtained by JD using sophisticated preconditioners (ILU, HYPRE and pARMS), compared with the JD results when no preconditioning is used, and with the GD results for those preconditioners. Apparently, the use of GMRES for solving the correction equation does not improve the results. Nevertheless, GD needs less matrix-vector products, but JD is generally faster, as in the previous experiment (see Figure 2.5).

Experiment III compares SLEPc GD and JD solvers with PRIMME's solver JDQMR_Etol, in which the JD correction equation is solved by QMR with an ad-hoc stopping criterion [Stathopoulos, 2007, §3.3]. In this case, the eigenvalues closest to an interior target are computed in a group of 99 standard Hermitian problems. We observe the same trend concerning the number of converged pairs by GD and JD as in experiment II. PRIMME JD obtains better figures than SLEPc solvers, possibly due to the effectiveness of the PRIMME stopping criterion, that reduces significantly the number of matrix-vectors products (see Figure 2.6, left). However this PRIMME advantage is less important considering the total time spent by the solvers (see Figure 2.6, right).

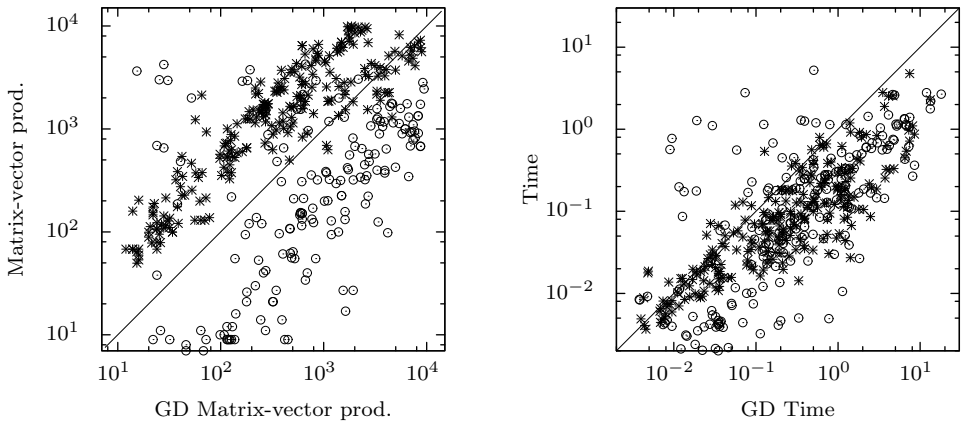


Figure 2.6: Comparison of number of matrix-vector products (left) and execution time in seconds (right) between SLEPc JD ($*$) and PRIMME JD ($\odot$), taking the results of SLEPc GD as a basis, when computing the eigenvalues closest to a target in standard Hermitian problems. A mark above the line corresponds to an experiment with GD performing better than JD.

2.4.2 Application 1: unstable modes of turbulent plasma

The plasma physics application GENE [Dannert and Jenko, 2005] computes micro-instabilities in fusion plasma, solving the gyrokinetic equations, a set of nonlinear partial integro-differential equations in five-dimensional phase space by means of the method of lines. In certain analyses, a few rightmost eigenvalues of a large complex non-Hermitian linear operator (available through matrix-vector products) must be computed, which is computationally hard because eigenvalues with large imaginary part dominate the spectrum. The required rightmost eigenvectors correspond to the unstable modes of the linearized gyrokinetic equation,

$$\frac{\partial g}{\partial t} = \mathcal{L}[g], \tag{2.28}$$

that describes the time evolution of the distribution of the particles in the plasma.

A study of the sequential and the parallel performance of SLEPc's Jacobi-Davidson is presented in Ch. 4, computing two rightmost eigenvalues using the harmonic extraction and solving the correction equation with BiCGStab(2) without preconditioner. The results reveal that the dynamic stopping criterion in the iterative solution of the JD correction equation (see §2.3.1) effectively improves the parallel performance (by reducing the overall number of reductions), and that the global performance has a strong dependence on the matrix-vector product performance.

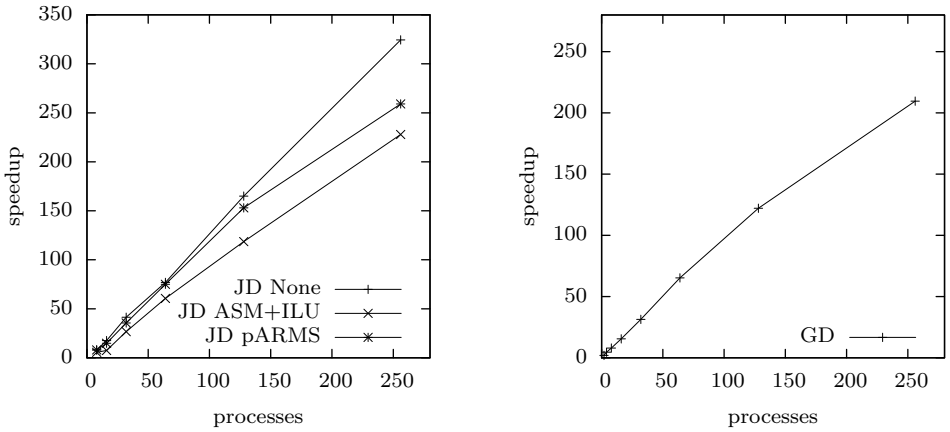


Figure 2.7: Speedup examples corresponding to GENE (left) and the DFT code (right) employing the SLEPc JD and GD solvers, respectively.

An explicit sparse representation of the operator is not computationally feasible because of the high density of the resulting matrix. Instead, we opted for building a preconditioner based on a sparse part of the operator that still retains most of the information of the system. The results in Ch. 5 corresponding to this preconditioning illustrate an acceleration of more than 10 times faster using ASM (restrictive Additive Schwarz Method) and more than 3 times using pARMS, both preconditioners scaling well (see Figure 2.7, left). Moreover the work presents a parameter scan test case in which similar problems are solved, and the total time is reduced around 23% if the eigensolver's initial subspaces are taken from the previously obtained solutions enriched as described in §2.1.6.

2.4.3 Application 2: electronic configuration of atoms

The Density Functional Theory (DFT) is one of the most popular methods that can be used to compute the electronic structure (principally the ground state) of atoms and molecules. The method requires the solution of the Schrödinger equation and the Poisson equation, that are coupled to each other. The computational approach consists in applying a self-consistent scheme, that is, the solution of the Schrödinger equation determines the next Poisson equation. The alternate solution of both equations is stopped when their results do not differ from the previous iterations under certain threshold.

After the discretization by the finite element method, the Poisson and Schrödinger equations result in a linear system and a generalized eigensystem, respectively, with large, sparse Hermitian matrices. Both problems are solved approximately,

and in the case of the eigenvalue problem, the number of computed solutions depends on how many orbitals are sought.

Chapter 6 presents a comparison of different refinement strategies for the meshes generated during the discretization. The computation of the smallest eigenvalues (corresponding to the lowest energy orbitals) of the generalized Hermitian problems was done using GD with harmonic extraction and block Jacobi as the preconditioner for the expansion. Results for sequential and parallel performance are given, together with the acceleration produced by the subspace recycling within the self-consistent loop, resulting in a speedup of 4 with respect to no recycling. A sample parallel speedup is shown in Figure 2.7, right.

2.5 Conclusions

This chapter has introduced the implementation of Davidson methods in the context of the SLEPc library for the solution of Hermitian and non-Hermitian eigenvalue problems, both standard and generalized. The proposed solvers incorporate state-of-the-art expansion methods (such as Generalized Davidson and some variants of Jacobi-Davidson), extraction techniques (such as standard Rayleigh-Ritz and harmonic variants) and restart techniques (GD+k), exposing a number of parameters that allow for the adaptation of the solver to the characteristics of the problems. The solvers are robust and efficient by trying to maintain the structural properties of the original problem in the projected problem, and performing the operations in real arithmetic whenever possible.

The implementations are fully integrated in the PETSc framework, as the rest of SLEPc solvers, inheriting its benefits such as ease of solver customization via the command line (generally the optimal configuration is not straightforward), availability of a large variety of iterative linear solvers and preconditioners even from external packages, and high-performance computing capabilities, mostly MPI but also increasing support for novel architectures like multi-cores and GPUs.

In terms of practical use, the SLEPc Davidson solvers are competitive with respect to other free parallel libraries, and even provide new features like the support for non-Hermitian problems and harmonic extraction methods. We have addressed two relevant scientific computing applications, the computation of unstable modes of plasma and electronic configurations of atoms, in which obtaining the corresponding eigenpairs is challenging for iterative solvers.

Chapter 3

A Double-Expansion Davidson Method

We consider the generalized eigenvalue problem (GEP)

$$A\mathbf{x} = \lambda B\mathbf{x}, \tag{3.1}$$

where A and B are $n \times n$ matrices, and are interested in interior eigenvalues close to a given target $\tau \in \mathbb{C}$. Efficient computation of these eigenvalues is a hard task that generally requires both a suitable subspace extraction process (often harmonic Rayleigh–Ritz is used, see, e.g., [Stewart, 2001b]) and a quality subspace expansion method. This subspace expansion in turn generally requires a (good) preconditioner and/or many steps of an iterative linear solver, depending on the complexity of the problem at hand. In this chapter we assume that we have a preconditioner M, which, for instance, may be an inexact LU-decomposition of $A - \tau B$.

Iterative methods based on Krylov subspaces (for instance, Lanczos for Hermitian problems, and Arnoldi and Krylov-Schur for non-Hermitian problems) are widely used to compute the eigenvalues in the extremes of the spectrum of standard eigenvalue problems. However, Davidson methods may present better performance computing interior eigenvalues and/or in generalized eigenproblems when exact solves with $A - \tau B$ are unaffordable, but some approximations are available, so-called preconditioners, [Davidson, 1975; van Lenthe and Pulay, 1990; Crouzeix et al., 1994; Heuveline et al., 1997; Arbenz et al., 2006; Genseberger, 2010].

The outline of the chapter is as follows. We start with a review of the Davidson methods, with a special interest in the expansions of Generalized Davidson (GD), Olsen (more robust method than Generalized Davidson but twice preconditioner applications) and Jacobi-Davidson. Then a new expansion is presented in §3.2, that tries to improve the robustness of Olsen at the same cost in terms of preconditioner applications, and we compare the new expansion with GD in terms of convergence. The section ends discussing some consideration for an optimal

implementation. After that, in §3.3 we offer some examples of the effectiveness of the new expansion computing interior eigenvalues in a collection of problems.

3.1 Expansions in Davidson methods

Starting with the introduction of the Davidson method [Davidson, 1975], there have been a wide variety of developments in the subspace expansion. Generalized Davidson (GD) [Morgan and Scott, 1986] introduces the first expansion that uses an arbitrary preconditioner, which is applied to the residual to try to enrich the approximation in the direction of the desired eigenvector. However, it may very well occur that the resulting vector is almost collinear to the approximate eigenvector, leading to the stagnation of the method. The Olsen variant [Olsen et al., 1990] attempts to avoid this situation by working on the orthogonal complement of the span of the approximate eigenvector.

Jacobi–Davidson (JD) [Sleijpen and van der Vorst, 1996, 2000] also seeks to avoid the stagnation, but differs from the previous methods in the fact that the convergence of JD may depend less on the quality of the preconditioner. The JD expansion results from the approximate solution of a linear system called the correction equation; the quality or the efficiency of the computation may be enhanced by a preconditioner.

GD and Olsen are attractive because of their straightforward implementation and good performance for easier problems. A major challenge in JD is the adaptive determination of parameters such as the number of inner steps; see [Stathopoulos, 2007; Hochstenbach and Notay, 2009] for recent progress in this direction. A rule-of-thumb is that JD may be necessary for harder problems.

For interior eigenvalues subspace expansion methods include (inexact) Rayleigh quotient iteration (RQI), (inexact) inverse iteration, or Jacobi–Davidson [Sleijpen et al., 1996]. Let $(\theta, \mathbf{u}) \approx (\lambda, \mathbf{x})$ be an approximate eigenpair, where $\mathbf{u}$ is in the search space $\mathcal{U}$. In the Jacobi–Davidson method, a possible correction equation is

$$\left(I - \frac{B\mathbf{u}\mathbf{u}^*}{\mathbf{u}^*B\mathbf{u}}\right)(A - \theta B)\,\mathbf{t} = -\mathbf{r}, \qquad \mathbf{t} \perp \mathbf{u}, \tag{3.2}$$

where $\mathbf{r} := A\mathbf{u} - \theta B\mathbf{u}$. This $\mathbf{t}$ is used to expand the search space. With the (standard) projected preconditioning we solve $\mathbf{t} \perp B\mathbf{u}$ from

$$\left(I - \frac{M^{-1}B\mathbf{u}\mathbf{u}^*}{\mathbf{u}^*M^{-1}B\mathbf{u}}\right)M^{-1}(A - \theta B)\mathbf{t} = -\left(I - \frac{M^{-1}B\mathbf{u}\mathbf{u}^*}{\mathbf{u}^*M^{-1}B\mathbf{u}}\right)M^{-1}\mathbf{r}.$$

First we note that we may approximate the solution $\mathbf{t}$ by just taking the right-hand side

$$\mathbf{t}_{\text{olsen}} = -\left(I - \frac{M^{-1}B\mathbf{u}\mathbf{u}^*}{\mathbf{u}^*M^{-1}B\mathbf{u}}\right)M^{-1}\mathbf{r}.$$

This is a linear combination of $M^{-1}(A\mathbf{u} - \theta B\mathbf{u})$ and $M^{-1}B\mathbf{u}$, orthogonal to $\mathbf{u}$. The subscript reflects that fact that it is a generalization for the GEP of the approach

advocated by Olsen et al. [Olsen et al., 1990], who proposed the expansion

$$-\left(I - \frac{M^{-1}\mathbf{u}\mathbf{u}^*}{\mathbf{u}^*M^{-1}\mathbf{u}}\right) M^{-1}(A\mathbf{u} - \theta\mathbf{u})$$

for the standard eigenvalue problem $A\mathbf{x} = \lambda\mathbf{x}$.

We can also precondition (3.2) by a regular (unprojected) preconditioner (cf. [Hochstenbach and Notay, 2009])

$$M^{-1}\left(I - \frac{B\mathbf{u}\mathbf{u}^*}{\mathbf{u}^*B\mathbf{u}}\right)(A - \theta B)\,\mathbf{t} = M^{-1}\mathbf{r}, \qquad \mathbf{t} \perp B\mathbf{u}.$$

Again, we may approximate the solution $\mathbf{t}$ of (3.2) by just taking the right-hand side

$$\mathbf{t}_{\mathrm{GD}} = M^{-1}\mathbf{r};$$

this approach is called generalized Davidson (GD) or also preconditioned inverse iteration (PINVIT) in the literature (see, e.g., [Neymeyr, 2001]).

We now make some comparisons between $\mathbf{t}_{\mathrm{olsen}}$ and $\mathbf{t}_{\mathrm{GD}}$. Suppose that M is a preconditioner of good quality, for the moment we will assume that $M^{-1} = (A - \tau B)^{-1}$ is an exact inverse. Then, for this special case,

$$\mathbf{t}_{\mathrm{olsen}} = -\mathbf{u} + (\mathbf{u}^*(A - \tau B)^{-1}B\mathbf{u})^{-1}\,(A - \tau B)^{-1}B\mathbf{u}$$

and

$$\mathbf{t}_{\mathrm{GD}} = \mathbf{u} + (\tau - \theta)(A - \tau B)^{-1}B\mathbf{u}.$$

If θ is very close to τ, which for instance may be possible if the target is quite accurate, it is clear that $\mathbf{t}_{\mathrm{GD}}$ may degenerate. Indeed, the case that $\theta = \tau$ suffers from the well-known "Davidson paradox": the perfect preconditioner gives no subspace expansion. The Olsen approach does not share this disadvantage; note that

$$\|(\mathbf{u}^*(A - \tau B)^{-1}B\mathbf{u})^{-1}\,(A - \tau B)^{-1}B\mathbf{u}\| \geq 1.$$

However, a clear disadvantage of $\mathbf{t}_{\mathrm{olsen}}$ is that, unless M^{-1} is an exact inverse of $A - \tau B$ as above, this approach spends two actions of the preconditioner ($M^{-1}A\mathbf{u}$ and $M^{-1}B\mathbf{u}$, or $M^{-1}B\mathbf{u}$ and $M^{-1}(A\mathbf{u} - \theta B\mathbf{u})$); while $\mathbf{t}_{\mathrm{GD}}$ spends only one. In the next section we propose a new approach that attempt to turn this fact into a strength.

3.2 A double subspace expansion approach

Based on the observations of the previous section, we conclude that $\mathbf{t}_{\mathrm{GD}}$ is a comparatively cheap approach and may be sensible in particular if the preconditioner is of good quality and $M^{-1}(A - \theta B)$ is not close to the identity. On the other hand, the expansion $\mathbf{t}_{\mathrm{olsen}}$ may be more robust in general, but is twice as expensive in terms of actions with the preconditioner.

Algorithm 3.1: Simplified Davidson for finding the eigenvalue closest to τ

Input: initial eigenvector approximation $\mathbf{u}^{(0)}$.
Output: $\theta^{(k)}$ and $\mathbf{u}^{(k)}$ from the last iteration k.

Compute $\theta^{(0)} = \rho(\mathbf{u}^{(0)})$
for $i = 0, 1, 2, \ldots$ **do**
  Compute the residual $\mathbf{r}^{(i)} = (A - \theta^{(i)} B)\,\mathbf{u}^{(i)}$
  Test for convergence
  Compute $\mathbf{t}^{(i)} \perp B^* B\mathbf{u}^{(i)}$ such that $\|(A - \tau B)\mathbf{t}^{(i)} - \mathbf{r}^{(i)}\| \leq \xi_1^{(i)} \|\mathbf{r}^{(i)}\|$
  Set $\mathbf{u}^{(i+1)} = \|\mathbf{u}^{(i)} - \mathbf{t}^{(i)}\|^{-1}(\mathbf{u}^{(i)} - \mathbf{t}^{(i)})$ and $\theta^{(i+1)} = \rho(\mathbf{u}^{(i+1)})$
end

We now propose a new subspace expansion that may combine the strengths of the two approaches: we will expand the search space by *both* $M^{-1}A\mathbf{u}$ and $M^{-1}B\mathbf{u}$. While these vectors will asymptotically be collinear, generally this will not be the case until very late in the process.

We note that this subspace expansion process has a number of potential favorable properties. First, the expansion includes both $\mathbf{t}_{\text{olsen}}$ and $\mathbf{t}_{\text{GD}}$. Second, by expanding the space by more than one vector per outer iteration, effectively one (harmonic) Rayleigh–Ritz extraction process is avoided, saving computational costs. Third, the new expansion relies on the robustness of the extraction process to select the best combination of $M^{-1}A\mathbf{u}$ and $M^{-1}B\mathbf{u}$.

3.2.1 Comparative analysis

We introduce a simplified scheme of a Davidson method for seeking an eigenpair with the eigenvalue close to τ, that does not take into account the subspace acceleration, and in which the computed expansion at each iteration $\mathbf{t}^{(i)}$ is treated as an approximate solution of

$$(A - \tau B)\mathbf{t}^{(i)} = \mathbf{r}^{(i)}, \tag{3.3}$$

with a residual tolerance of $\xi_1^{(i)}$ with respect to $\|\mathbf{r}^{(i)}\|$. In this scheme the double expansion is considered as the linear combination

$$M^{-1}A\mathbf{u}^{(i)} + \beta^{(i)} M^{-1}A\mathbf{u}^{(i)}$$

with the $\beta^{(i)}$ that obtains the best solution of (3.3) under some criterion. The GD expansion corresponds to setting $\beta^{(i)} = -\theta^{(i)}$. The scheme is detailed in Algorithm 3.1.

Without loss of generality, for the following discussion we compute the approximate eigenvalue associated to an approximate eigenvector as the related generalized Rayleigh quotient

$$\rho(\mathbf{u}) = \frac{\mathbf{u}^* B^* A\mathbf{u}}{\mathbf{u}^* B^* B\mathbf{u}}, \tag{3.4}$$

and we consider that the approximate eigenvectors $\mathbf{u}$ are normalized so that $\|B\mathbf{u}\| = 1$.

Assume we have the approximate eigenvector $\mathbf{u}^{(i)}$ and an expansion

$$M^{-1}\tilde{\mathbf{t}}^{(i)} = M^{-1}(A + \beta^{(i)}B)\mathbf{u}^{(i)},$$

computed by either expansion. Consider $\mathbf{t}^{(i)}$ in Algorithm 3.1 as a projection of the expansion $M^{-1}\tilde{\mathbf{t}}^{(i)}$ orthogonal to $B^*B\mathbf{u}^{(i)}$. We characterize the quality of the expansion by $\mathbf{d}^{(i)}$,

$$(A - \tau B)\mathbf{t}^{(i)} = \mathbf{r}^{(i)} + \mathbf{d}^{(i)}, \text{ for } \mathbf{t}^{(i)} \perp B^*B\mathbf{u}^{(i)}. \tag{3.5}$$

We can write (3.5) as

$$(A - \tau B)M^{-1}\tilde{\mathbf{t}}^{(i)} = \mathbf{r}^{(i)} + \alpha B\mathbf{u}^{(i)} + \mathbf{d}^{(i)},$$

where

$$\alpha = \frac{\mathbf{u}^{(i)^*}B^*B\left(M^{-1}\tilde{\mathbf{t}}^{(i)} - (A - \tau B)^{-1}(\mathbf{r}^{(i)} + \mathbf{d}^{(i)})\right)}{\mathbf{u}^{(i)^*}B^*B(A - \tau B)^{-1}B\mathbf{u}^{(i)}}.$$

This results in

$$Q(A - \tau B)M^{-1}\tilde{\mathbf{t}}^{(i)} = (A - \tau B)\tilde{Q}M^{-1}\tilde{\mathbf{t}}^{(i)} = Q\mathbf{r}^{(i)} + Q\mathbf{d}^{(i)}, \tag{3.6}$$

where

$$Q = I - \frac{B\mathbf{u}^{(i)}\mathbf{u}^{(i)^*}B^*B(A - \tau B)^{-1}}{\mathbf{u}^{(i)^*}B^*B(A - \tau B)^{-1}B\mathbf{u}^{(i)}} \quad \text{and} \quad \tilde{Q} = I - \frac{(A - \tau B)^{-1}B\mathbf{u}^{(i)}\mathbf{u}^{(i)^*}B^*B}{\mathbf{u}^{(i)^*}B^*B(A - \tau B)^{-1}B\mathbf{u}^{(i)}}.$$

Hence we take $\mathbf{t}^{(i)} := \tilde{Q}M^{-1}\tilde{\mathbf{t}}^{(i)}$, satisfying $\tilde{Q}M^{-1}\tilde{\mathbf{t}}^{(i)} \perp B^*B\mathbf{u}^{(i)}$.

Then we approximate the distance to the best expansion by $\delta^{(i)} := \|\mathbf{d}^{(i)}\|$, that for the GD expansion is

$$\delta_{GD}^{(i)} := \|\mathbf{d}_{GD}^{(i)}\| = \left\| \left(Q(A - \tau B)M^{-1} - I\right)\mathbf{r}^{(i)} \right\|.$$

Note that $\delta_{GD}^{(i)} = 0$ if $M = A - \tau B$. In the case of GD2, $\beta^{(i)}$ is a free parameter, and it is possible to find the optimal value $\beta_{opt}^{(i)}$ that minimizes $\delta^{(i)}$. Furthermore, we will assume that GD2 selects $\beta_{opt}^{(i)}$ when computing the correction $\mathbf{t}^{(i)}$.

Proposition 1. *If GD2 selects $\beta^{(i)} = \beta_{opt}^{(i)}$, then in general $\delta_{GD}^{(i)} \geq \delta_{GD2}^{(i)}$ and the equality holds at every iteration if $M = A - \tau B$.*

Proof. If we rewrite (3.6) as

$$\begin{aligned}\mathbf{d}^{(i)} &= (Q(A - \tau B)M^{-1} - I)\mathbf{r}^{(i)} + Q(A - \tau B)M^{-1}B\mathbf{u}^{(i)}(\beta^{(i)} - \theta^{(i)}) \\ &= \mathbf{d}_{GD}^{(i)} + Q(A - \tau B)M^{-1}B\mathbf{u}^{(i)}(\beta^{(i)} - \theta^{(i)}),\end{aligned} \tag{3.7}$$

Algorithm 3.2: Inexact Inverse Iteration

Input: initial approximate eigenvector $\mathbf{u}^{(0)}$
Output: $\theta^{(k)}$ and $\mathbf{u}^{(k)}$ from the last iteration k

for $i = 1, 2, \ldots$ **do**
  Choose the shift $\sigma^{(i)}$ and the tolerance $\xi^{(i)}$
  Find $\mathbf{y}^{(i)}$ such that

$$\|(A - \sigma^{(i)}B)\,\mathbf{y}^{(i)} - B\mathbf{u}^{(i-1)}\| \leq \xi^{(i)}\|B\mathbf{u}^{(i-1)}\| \tag{3.8}$$

  Set $\mathbf{u}^{(i)} = \mathbf{y}^{(i)}\|B\mathbf{y}^{(i)}\|^{-1}$ and $\theta^{(i)} = \rho(\mathbf{u}^{(i)})$
  Test for convergence
end

then the application of least squares leads to bound $\delta_{GD2}^{(i)}$ as

$$\delta_{GD2}^{(i)} = \min_{\beta^{(i)}} \delta^{(i)} = \|(Q(A-\tau B)M^{-1}B\mathbf{u}^{(i)})^{\perp}\mathbf{d}_{GD}^{(i)}\| \leq \|\mathbf{d}_{GD}^{(i)}\| = \delta_{GD}^{(i)},$$

where, by abuse of notation, we employ $\mathbf{v}^{\perp}$ to denote the orthogonal projector $I - \mathbf{v}(\mathbf{v}^*\mathbf{v})^{-1}\mathbf{v}^*$. Of course, when $M = A - \tau B$, $\mathbf{d}_{GD}^{(i)} = \mathbf{0}$, and hence $\mathbf{d}_{GD2}^{(i)} = \mathbf{0}$. □

From Proposition 1 it seems clear that the case where the double subspace expansion is more likely to be advantageous is in problems with preconditioners M^{-1} far from $(A - \tau B)^{-1}$.

3.2.2 Convergence analysis

We approach the analysis of the convergence of GD by relating the Davidson iterations with inexact inverse iterations.

A quite general convergence theory of Algorithm 3.2 is presented in [Freitag and Spence, 2007], for the computation of a finite eigenvalue θ_1 and the corresponding right eigenvector $\mathbf{x}_1$ of a generalized nonsymmetric eigenvalue problem. Consider the block factorization of $A - \theta B$ as

$$U^{-1}(A - \theta B)X = \begin{bmatrix} t_{11} & \mathbf{0}^* \\ \mathbf{0} & T_{22} \end{bmatrix} - \theta \begin{bmatrix} s_{11} & \mathbf{0}^* \\ \mathbf{0} & S_{22} \end{bmatrix}, \tag{3.9}$$

with nonsingular square matrices U and X of size n. In [Freitag and Spence, 2007], $\mathbf{u}^{(i)}$ is decomposed as

$$\mathbf{u}^{(i)} = \alpha^{(i)}(\mathbf{x}_1\zeta^{(i)} + X_2\mathbf{p}^{(i)}),$$

for some $\zeta^{(i)} \in \mathbb{C}$ and $\mathbf{p}^{(i)} \in \mathbb{C}^{n-1}$, where $X = [\mathbf{x}_1 \; X_2]$ and $\alpha^{(i)}$ is chosen so that $\|B\mathbf{u}^{(i)}\| = 1$. Then the quotient

$$\pi^{(i)} := \frac{\|S_{22}\mathbf{p}^{(i)}\|}{|s_{11}\zeta^{(i)}|}$$

is introduced as a measure for convergence, since it can be interpreted as a generalized tangent of the angle between $\mathbf{u}^{(i)}$ and the eigenvector $\mathbf{x}_1$. The following theorem shows the conditions guaranteeing that $\pi^{(i)}$ decreases linearly.

Theorem 1. *Let $(\theta_1, \mathbf{x}_1)$ be an algebraically simple eigenpair of (3.1) and let the decomposition (3.9) be induced by $\mathbf{x}_1$, with $\theta_1 = t_{11}/s_{11}$. Assume that the initial guess $\mathbf{u}^{(0)}$ satisfies $0 < \|S_{22}\mathbf{p}^{(0)}\| < 1$ and $\sigma^{(i)} \notin \lambda(T_{22}, S_{22})$. If $\sigma^{(i)}$ and $\tau_{(i)}$ are chosen in Algorithm 3.2 so that*

$$|\theta_1 - \sigma^{(i)}| < \frac{\|(T_{22} - \theta_1 S_{22})^{-1}\|^{-1}}{2\,\|S_{22}\|} \|S_{22}\mathbf{p}^{(i)}\|,$$

and

$$\xi^{(i)} < \frac{\alpha^{(i)}}{\|B\mathbf{u}^{(i)}\|\,\|\mathbf{u}_1\|}\,\beta\,|s_{11}\zeta^{(i)}|,$$

with $0 \le 2\beta < 1 - \pi^{(0)}$, then Algorithm 3.2 converges linearly.

Proof. See Theorem 3.4 in [Freitag and Spence, 2007]. □

Using the convergence theory of Inexact Inverse Iteration, we shall prove the linear convergence of GD2. For that we rewrite the step of the computed correction $\mathbf{t}^{(i)}$ (step 5 in Algorithm 3.1) represented by (3.5) in the form of (3.8),

$$(A - \tau B)(\mathbf{t}^{(i)} - \mathbf{u}^{(i)})(\tau - \theta^{(i)})^{-1} - B\mathbf{u}^{(i)} = \mathbf{d}_{GD2}^{(i)}(\tau - \theta^{(i)})^{-1}.$$

Hence if the computed correction obtained in Algorithm 3.1, $\mathbf{t}^{(i)}$, is used in Algorithm 3.2 as

$$\mathbf{y}^{(i)} = (\mathbf{t}^{(i)} - \mathbf{u}^{(i)})(\tau - \theta^{(i)})^{-1},$$

we can bound $\delta_{III}^{(i)}$, assuming that the inequality in step 5 of Algorithm 3.1 for the GD2 expansion holds at every iteration ($\|\mathbf{d}_{GD2}^{(i)}\| \le \xi_1 \|\mathbf{r}^{(i)}\|$), as

$$\delta_{III}^{(i)} = \|(A - \sigma^{(i)}B)\,\mathbf{y}^{(i)} - B\mathbf{u}^{(i-1)}\| = \|\mathbf{d}_{GD2}^{(i)}\||\tau - \theta^{(i)}|^{-1} \le \xi_1 \|\mathbf{r}^{(i)}\||\tau - \theta^{(i)}|^{-1}. \tag{3.10}$$

Corollary 1. *Let $(\lambda_1, \mathbf{x}_1)$ be an algebraically simple eigenpair of (3.1). Assume that the initial vector $\mathbf{u}^{(0)}$, and the values of $\sigma^{(i)} = \tau$ and $\xi^{(i)} = \xi_{III}^{(i)}$, where*

$$\xi_{III}^{(i)} := \xi_1 \|\mathbf{r}^{(i)}\||\tau - \theta^{(i)}|^{-1}, \tag{3.11}$$

satisfy the conditions of Theorem 1. Then Algorithm 3.1 converges linearly to $(\lambda_1, \mathbf{x}_1)$.

Proof. Note that $\|B\mathbf{u}^{(i)}\| = 1$ and (3.10) yield

$$\delta_{III}^{(i)} \leq \xi_1 \|\mathbf{r}^{(i)}\| |\tau - \theta^{(i)}|^{-1} \leq \xi_{III}^{(i)} \|B\mathbf{u}^{(i)}\|.$$

□

Note that the suggested value of $\xi_{III}^{(i)}$ of (3.11) is similar to the one proposed in [Freitag and Spence, 2007, Remark 3.5].

3.2.3 Algorithmic details

As a consequence of expanding the search subspace with two vectors, the matrix-vector product, the preconditioner application and the orthogonalization of the search subspace double their cost, at least, per iteration with respect to the single vector expansions. Rearranging the operations in blocks may improve the data locality and reduce the time spent per vector.

Some high-performance libraries provide multivector versions of the sparse matrix-vector product and are available even for new computer architectures such as multicore processors and GPUs [Williams et al., 2009]. Unfortunately, the availability of multivector preconditioners is quite rare. Still, in very large problems, consecutively applying the preconditioner to the vectors may imply an effective time reduction since the second application can reuse the preconditioner data already available in faster memory. An example of this can be found in Ch. 4.

The SVQB method [Stathopoulos and Wu, 2002] computes an orthogonal basis from a set of vectors, but instead of orthogonalizing one vector at a time, it works with a block of vectors employing matrix-matrix operations almost exclusively. An interesting candidate for updating the search subspace V with several new vectors W is the GS-SVQB variant that combines a Gram–Schmidt procedure to orthogonalize the new vectors against V, $W \leftarrow (I - VV^*)W$, and SVQB for the inner orthogonalization of the new block. Like (classical or modified) Gram–Schmidt, an iterative version of GS-SVQB that achieves good orthogonality levels is available.

Remark 1. *As mentioned before,* $A\mathbf{u}$ *and* $B\mathbf{u}$ *will asymptotically become linearly dependent. Therefore, it may be sensible to first determine an orthonormal basis for* $[A\mathbf{u}\ B\mathbf{u}]$ *before applying the preconditioner* M*. However, in our experiments we did not encounter an example in which the orthogonalization of* $[A\mathbf{u}\ B\mathbf{u}]$ *decreased the number of iterations significantly.*

3.3 Numerical results

First we illustrate the potential of the new expansion by computing interior eigenvalues of a diagonal generalized eigenvalue problem of order $n = 200$, formed by diagonal matrices A and B, with $a_{i,i} = i$ and $b_{i,i} = n - i + 1$. We study the impact of the quality of the preconditioner in the convergence. For that, we use a

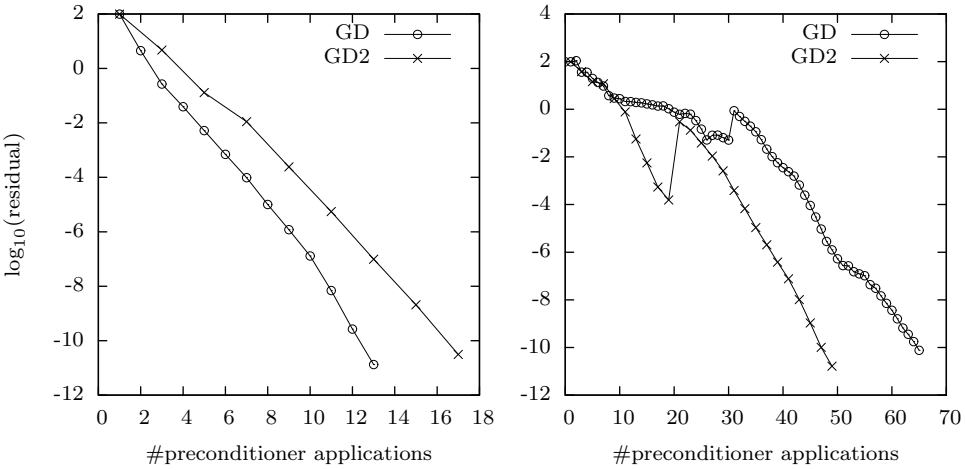


Figure 3.1: Residual norm against preconditioner applications spent by single expansion (GD) and the new expansion (GD2) using the high quality preconditioner M_0 (left plot) and the low quality preconditioner M_α with $\alpha = 10^{4/3}$ (right plot).

preconditioner with a configurable quality,

$$M_\alpha^{-1} = (A - \tau B + \alpha E)^{-1}, \tag{3.12}$$

where E is a diagonal matrix whose diagonal elements are random numbers uniformly distributed in the interval $[-1, 1]$. The quality of a preconditioner M for $A - \tau B$ may be estimated by its difference relative to M:

$$\gamma(M) = \|M^{-1}(M - (A - \tau B))\| = \|I - M^{-1}(A - \tau B)\|.$$

For the above eigenproblem, we obtained larger values of $\gamma(M_\alpha)$ for larger values of α, when α has a relatively small value (see Table 3.1). We present results up to the point where this trend is dramatically inverted, because for large values of α, $\gamma(M_\alpha)$ lowers to 1:

$$\lim_{\alpha\to\infty} \|I - M_\alpha^{-1}(A - \tau B)\| = \lim_{\alpha\to\infty} \|(A - \tau B + \alpha E)^{-1}\alpha E\| = 1.$$

We look for the eigenvalues closest to τ, which in this experiment is set to the arithmetic mean of the eigenvalues of the pair (A, B). The tolerance on the residual norm for the convergence of the eigenpair is 10^{-10}. The search subspace is bounded to 50 vectors and the solvers restart with 25 vectors. The harmonic Rayleigh-Ritz procedure is used to extract the approximate eigenpairs from the search subspace. In Figure 3.1 we have plotted the residual norm against the number of applications

Table 3.1: Iterations spent by the solvers versus the preconditioner quality

α	$10^{\frac{0}{3}}$	$10^{\frac{1}{3}}$	$10^{\frac{2}{3}}$	$10^{\frac{3}{3}}$	$10^{\frac{4}{3}}$	$10^{\frac{5}{3}}$	$10^{\frac{6}{3}}$
$\gamma(M_\alpha)$	1.2	1.1	2.5	9.6	31.3	251.6	20.5
GD	13	16	22	34	65	174	–
GD2	18	22	28	34	50	78	126

of the preconditioner (which is the same as the application of the matrix A, and also for matrix B), until one eigenpair converges. We can infer from the left plot that, in the case of using the high quality preconditioner M_0, the extra vector of the new expansion does not accelerate the convergence, compared with GD. However, using the low quality preconditioner M_α with $\alpha = 10^{\frac{4}{3}}$ the acceleration of the new expansion is evident. Table 3.1 shows the progressive effect of the preconditioner quality on the total number of preconditioner applications required by the solvers. One sees that the new expansion is less sensitive to the lack of preconditioner quality and its performance is significantly better in (very) low quality cases (like the case of $\alpha = 100$, in which GD required more than 1000 iterations to converge).

We tested the approach also on standard eigenvalue problems, in particular with a diagonal matrix A with entries $a_{i,i} = i/(n - i + 1)$, that has the same solutions as the generalized example above. In this case, we found that the convergence history is very similar to the one shown in Fig. 3.1.

We have checked the conclusion inferred in the previous simple case by testing the new expansion in a collection of 262 problems, both standard and generalized. The problem matrices were taken from the real and complex matrices available in the University of Florida Sparse Matrix Collection [Davis and Hu, 2011], disregarding their original application. The targets have been set to the arithmetic mean of the eigenvalues of the problem, which guarantees that the obtained approximate eigenvalues are interior. The tolerance on the residual norm for the converged eigenpair was $10^{-7}\|A\|_2$ and $10^{-7}(\|A\|_2 + |\theta|\|B\|_2)$, respectively, for standard and generalized eigenproblems. These stopping criteria come from the backward stability theory applied to eigenvalue problems and is useful to automatically set sensible tolerances considering the conditioning of the problem (see [Higham and Higham, 1998, Th. 2.1]). The preconditioner used was the standard incomplete LU factorization without fill-in, ILU(0), provided by the Matlab function `ilu`. The search subspace is bounded to 100 vectors and the solvers were configured to restart with 50 vectors. All the problems are solved five times with different random initial vectors (but the same vectors for both methods).

We present the time and the iterations spent by an experimental code in Matlab with harmonic Rayleigh-Ritz procedure to extract the eigenvalues close to the target, thick restart and the possibility of selecting between the single expansion (GD) or the double expansion (GD2) approach. The executions were carry out

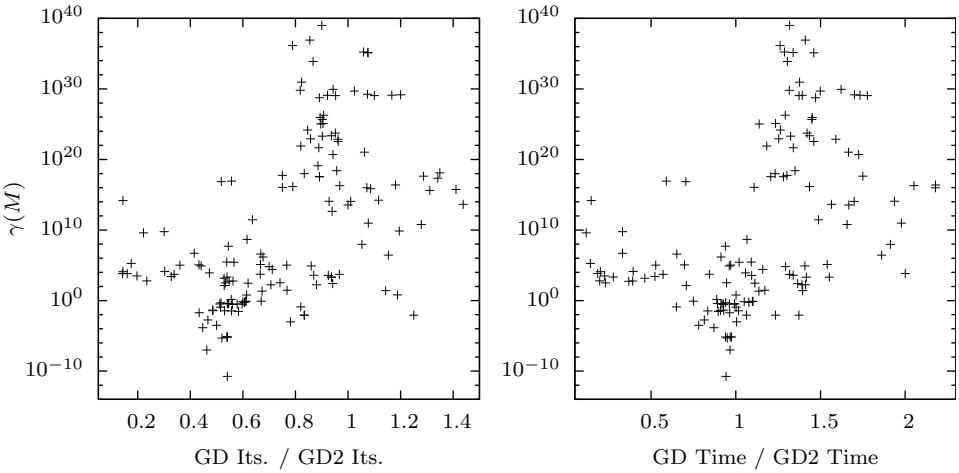


Figure 3.2: Gain of the new expansion (GD2) over the single expansion (GD) in number of preconditioner applications (left plot) and time (right plot), versus the quality of the ILU(0) decomposition as preconditioner.

on a machine consisting of 256 JS20 blade computing nodes, each of them with two 64-bit PowerPC 970+ processors running at 2.2 GHz. The interpreter of the Matlab code was Octave 3.3.54.

Considering an iteration as adding one vector to the search subspace, the left plot in Figure 3.2 represents the gain in iterations of the new expansion over the single expansion against the quality of the ILU(0) preconditioner. It is possible to observe two clusters of points roughly separated by the gain 0.7 and the quality 10: in general the preconditioner quality of the points with a gain lower than 0.7 is less than 10, and the preconditioner quality of the points with a gain higher than 0.7 is more than 10. This tendency strengthens our hypothesis that the new expansion is more suitable for low quality preconditioners.

Finally, we present some timing results. Our code uses the Matlab multivector matrix-vector product and the multivector preconditioner application. We note that instead of a block orthogonalization procedure (discussed in the previous section), the simpler repeated classical Gram–Schmidt procedure is used. The right plot in Figure 3.2 represents the gain in time of the new expansion over the single expansion. We can observe that the quality-gain pattern is similar to the pattern shown by the left plot: the single expansion is faster in 115 out of 169 (68%) problems with preconditioner quality less than 10, while the double expansion is faster in 85 out of 93 (91%) with preconditioner quality greater than 10.

3.4 Conclusions

We have proposed a new subspace expansion method, as an alternative to Olsen and Generalized Davidson. Compared to Olsen, the new approach has a double expansion with the same number of applications of the preconditioner. Compared to Generalized Davidson, the method doubles the number of actions of the preconditioner per outer iteration, but not when compared to the size of the subspace. Based on the analysis comparing the convergence of both Generalized Davidson and the double expansion with Inexact Inverse Iteration, the new approach seems to be most promising for relatively low-quality preconditioners. This conclusion is supported by comparing the performance of the methods computing interior eigenvalues in the proposed collection of eigenproblems.

The convergence analysis did not consider the extraction method neither the subspace acceleration, although it is reasonable to think that they may play an important role in the robustness of the method. Their influence, besides the influence of a block orthogonalization like GS-SVQB, in the performance of the method are left for future work.

We remark that a prototype of the proposed method has already been implemented in SLEPc.

Chapter 4

Parallel Jacobi-Davidson in GENE

We are concerned with the standard eigenvalue problem defined by a large, sparse matrix A of order n, $Ax = \lambda x$, where the scalar λ is called the eigenvalue, and the n-vector x is called the eigenvector. Many iterative methods are available for the partial solution of the above problem, that is, for computing a subset of the eigenvalues. The most popular ones are Krylov projection methods such as Lanczos, Arnoldi or Krylov-Schur, and Davidson-type methods such as Generalized Davidson or Jacobi-Davidson. Details of these methods can be found in [Bai et al., 2000]. Krylov methods achieve good performance when computing extreme eigenvalues, but usually fail to compute interior eigenvalues. In that case, the convergence can be improved by combining the method with a spectral transformation technique, i.e., to solve $(A - \sigma I)^{-1}x = \theta x$ instead of $Ax = \lambda x$. The drawback of this approach is the added high computational cost of solving large linear systems at each iteration of the eigensolver. Moreover, for stability reasons these systems must be solved very accurately (normally with direct methods). Davidson-type methods aim at reducing the cost by solving linear systems approximately, without compromising the robustness, usually with iterative methods. This topic is treated, *e.g.*, in [Hochstenbach and Notay, 2009].

Davidson methods are becoming an excellent alternative due to the possibility of striking a balance between numerical behavior and computational performance. A powerful preconditioner (close to the matrix inverse), if available, can usually reduce the number of iterations significantly. However, in practice its use is normally too expensive computationally and may be difficult to parallelize, thus dominating the cost of the eigensolver. Otherwise, depending on the performance of the matrix-vector product, the preconditioner and the orthogonalization, there exist Davidson-type variants that can be competitive with respect to Krylov-type eigensolvers. This chapter illustrates an example of this.

In this chapter, we show results for the eigenvalue calculation that takes place in the plasma physics application GENE, that solves a set of non-linear partial integro-differential equations in five-dimensional phase space by means of the

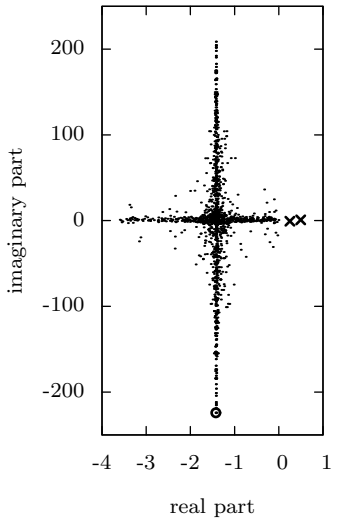


Figure 4.1: Spectrum of the linearized operator of a GENE problem similar to the test case I (Table 4.1). The largest magnitude (circle marks) and the rightmost (cross marks) eigenvalues are desired.

method of lines. Because of the shape of the spectrum (see Fig. 4.1), computing the largest magnitude eigenvalues of the linearized operator is not particularly difficult, despite the unfavorable characteristics of the problem (complex non-Hermitian with matrix in implicit form). However, the case of computing the rightmost eigenvalues is much more difficult from the numerical point of view, since these eigenvalues are much smaller in magnitude compared to the dominant ones. This makes the computational problem challenging and suitable as a testbed for our new parallel eigensolver running on distributed memory architectures.

The rest of the chapter is organized as follows. First we describe the Jacobi-Davidson method with harmonic extraction for standard problems without considering the preconditioning. Then the implementation details, including how the method is parallelized, are discussed. Next we provide a brief description of the application. The performance of the parallel eigensolver in this application is presented in the results section. Finally, we wrap up with some conclusions.

4.1 The Jacobi-Davidson method

Davidson-type methods belong to the class of subspace projection methods, where approximate eigenvectors are taken from a search subspace $\mathcal{V}$. Each iteration of these methods has two phases: subspace extraction and expansion. In the extraction phase, the solver selects the best (in terms of closeness to the desired region of the spectrum) from all available eigenpair approximations contained in $\mathcal{V}$. In the subspace expansion, a correction for the selected eigenpair is added to $\mathcal{V}$.

The subspace expansion distinguishes a Davidson-type variant from others. Jacobi-Davidson computes a correction $\mathbf{d}$ orthogonal to the selected approximate eigenvector $\tilde{\mathbf{x}}$ (considered normalized, $\|\tilde{\mathbf{x}}\| = 1$) as an approximate solution of the so-called Jacobi orthogonal component correction (JOCC) [Jacobi, 1846] equation

$$A(\tilde{\mathbf{x}} + \mathbf{d}) = \lambda(\tilde{\mathbf{x}} + \mathbf{d}), \quad \tilde{\mathbf{x}} \perp \mathbf{d}. \tag{4.1}$$

The correction $\mathbf{d}$ can be approximately obtained by the solution of the following system, often referred to as the Jacobi-Davidson correction equation [Sleijpen and van der Vorst, 1996],

$$(I - \tilde{\mathbf{x}}\tilde{\mathbf{x}}^*)\left(A - \tilde{\theta}I\right)(I - \tilde{\mathbf{x}}\tilde{\mathbf{x}}^*)\,\mathbf{d} = -\mathbf{r}, \tag{4.2}$$

where $\mathbf{r} := A\tilde{\mathbf{x}} - \tilde{\theta}\tilde{\mathbf{x}}$ is the residual associated to the selected approximate eigenpair $(\tilde{\theta}, \tilde{\mathbf{x}})$. However from (4.1) it is possible to formulate different linear systems. For our purpose, we implement the correction equation

$$\left(I - \frac{\mathbf{z}\mathbf{z}^*}{\mathbf{z}^*\mathbf{z}}\right)\left(A - \tilde{\theta}I\right)(I - \tilde{\mathbf{x}}\tilde{\mathbf{x}}^*)\,\mathbf{d} = -\mathbf{r}, \tag{4.3}$$

where $\mathbf{z} \in \operatorname{span}\{A\tilde{\mathbf{x}}, \tilde{\mathbf{x}}\}$. The equation comes from the one used in JDQZ [Fokkema et al., 1999], simplified for standard problems. This variant shows excellent results computing interior eigenvalues [Fokkema et al., 1999, Section 4.3].

If (4.3) is solved exactly, one step of the algorithm turns out to be one step of the Rayleigh Quotient Iteration, which converges almost quadratically [Fokkema et al., 1999]. Otherwise, if it is solved approximately, this high convergence rate may get lost. There is a trade-off between speed of convergence and the amount of work one is willing to spend for solving the equation, that is easily tuned if an iterative method is used. In practice, the performance of the eigensolver depends dramatically on a suitable stopping criterion for the iterative method.

Note that the correction equation (4.3) involves an operator for which the domain and the image space differ, due to the projectors when $\tilde{\mathbf{x}} \neq \mathbf{z}$. This can be problematic while forming powers of the coefficient matrix, like in the iterative solution of the correction equation using Krylov methods. Our implementation

includes the solution proposed in [Sleijpen et al., 1998, Section 3.2] and in [Fokkema et al., 1999, Section 2.6], that solves instead the equation

$$\left(I - \frac{\mathbf{z}\tilde{\mathbf{x}}^*}{\tilde{\mathbf{x}}^*\mathbf{z}}\right)\left(A - \tilde{\theta}I\right)\left(I - \frac{\mathbf{z}\tilde{\mathbf{x}}^*}{\tilde{\mathbf{x}}^*\mathbf{z}}\right)\mathbf{d} = -\mathbf{r}. \tag{4.4}$$

In the subspace extraction phase, Davidson-type methods classically impose the Ritz-Galerkin condition to the eigenpair $(\tilde{\theta}, \tilde{\mathbf{x}})$ that will be selected,

$$\mathbf{r} = A\tilde{\mathbf{x}} - \tilde{\theta}\tilde{\mathbf{x}} \perp \mathcal{V}. \tag{4.5}$$

Since $\tilde{\mathbf{x}} \in \mathcal{V}$, it is possible to express $\tilde{\mathbf{x}} = V\mathbf{u}$, V being an orthogonal basis of $\mathcal{V}$. This leads to the low-dimensional projected eigenproblem $V^*AV\mathbf{u} = \tilde{\theta}\mathbf{u}$.

In practice this extraction technique, called Rayleigh-Ritz projection, obtains good convergence rates when the eigenvalues of interest are those located at the periphery of the spectrum. However, it gives poor approximate eigenvectors for interior eigenvalues. The harmonic Rayleigh-Ritz method was proposed in [Morgan, 1991; Paige et al., 1995] as an alternative extraction technique for this case.

Assuming that interior eigenvalues close to a given target τ are desired, harmonic Rayleigh-Ritz imposes the Petrov-Galerkin condition

$$(A - \tau I)\tilde{\mathbf{x}} - \xi\tilde{\mathbf{x}} \perp \mathcal{W} := (A - \tau I)\mathcal{V} \tag{4.6}$$

to the selected eigenpair $(\tilde{\theta}, \tilde{\mathbf{x}})$ with $\tilde{\mathbf{x}} = V\mathbf{u}$, where $\xi = \tilde{\theta} - \tau$. For numerical stability reasons, both V and W (a basis of $\mathcal{W}$) are constructed to be orthonormal. The relation between the two bases is given by $(A-\tau I)V = WH$, where H is upper triangular. Similarly to the previous case, this leads to the projected eigenproblem

$$H\mathbf{u} = \xi W^*V\mathbf{u}, \tag{4.7}$$

considering that $W^*(A - \tau I)V = H$. Then the smallest magnitude pairs $(\xi, \mathbf{u})$ of (4.7) correspond to the pairs $(\xi + \tau, V\mathbf{u})$ in $\mathcal{V}$ closest to the target τ.

The eigenvectors of the projected eigenproblem, U, are not orthogonal in general, so updates such as $VU_{1:k}$ would require reorthogonalizing the resulting basis. For improved stability, instead of directly computing the eigenpairs of the projected problem, the generalized Schur decomposition of (4.7) is computed,

$$H = ZSU^*,\ W^*V = ZTU^*,\ \text{such that } U^*U = Z^*Z = I,\ \text{where } \xi_i = s_{i,i}/t_{i,i}.$$

At the end, the method works with Schur vectors along the computation, and the solver has obtained a partial Schur decomposition from which it is possible to compute the corresponding approximate eigenpairs.

The harmonic pairs are employed in other parts of the method,*e.g.*, the associated left vector of the selected pairs are set as $\mathbf{z}$ in the correction equation (4.4), and in the restart, the bases V and W are replaced by $VU_{1:m_{\min}}$ and $WZ_{1:m_{\min}}$ respectively.

Algorithm 4.1: Harmonic non-Hermitian Jacobi-Davidson

Input: matrix A of size n, number of wanted eigenpairs p, block size s, initial dimension of V m_0, maximum size of V $m_{\max}$, restart with $m_{\min}$ vectors

Output: resulting eigenvalues Θ and Schur vectors X

Choose a starting subspace basis V of m_0 vectors, such that $V^*V = I_{m_0}$
Compute W such that $(A - \tau I)V = WH$ and $W^*W = I_{m_0}$
Set $m \leftarrow m_0$, $l \leftarrow 0$, $\Theta \leftarrow \emptyset$ and $X \leftarrow \emptyset$
while $l < p$ **do**
    Extraction: Compute the Schur pairs $(\widetilde{\Theta}, \widetilde{X})$, from the generalized Schur decomp. $H = ZSU^*$ and $W^*V = ZTU^*$, where $\widetilde{X} = VU$ and $\tilde{\theta}_i = s_{i,i}/t_{i,i} + \tau$
    Sort the Schur pairs $(\widetilde{\Theta}, \widetilde{X})$
    Obtain the number of converged pairs k
    if $k > 0$ **then**
        Add eigenvalues $\widetilde{\theta}_1, \ldots, \widetilde{\theta}_k$ to Θ
        Set $X \leftarrow [X \quad \widetilde{X}_{1:k}]$, $V \leftarrow VU_{k+1:m}$ and $W \leftarrow WZ_{k+1:m}$
        Set $m \leftarrow m - k$ and $l \leftarrow l + k$
    end
    if $m \geq m_{\max}$ **then**
        Set $V \leftarrow VU_{1:m_{\min}}$, $W \leftarrow WZ_{1:m_{\min}}$, $\widetilde{\Theta} \leftarrow \widetilde{\Theta}_{1:m_{\min},1:m_{\min}}$, $\widetilde{X} = V$, $U = Z = I_{m_{\min}}$ and $m \leftarrow m_{\min}$
    end
    Expansion: Compute the correction D as in (4.4) for the first s pairs $(\widetilde{\Theta}, \widetilde{X})$
    Set $V \leftarrow [V \quad \text{orthonormalize}([X \quad V], D)]$
    Set $W \leftarrow [W \quad \text{orthonormalize}([X \quad W], (A - \tau I)V_{m:m+s})]$
    Set $m \leftarrow m + s$
end

Algorithm 4.1 summarizes the scheme of a Jacobi-Davidson method with harmonic Rayleigh-Ritz extraction. Note that the algorithm illustrates a block version, i.e., s eigenpair approximations are improved simultaneously in each iteration. As in all iterative algorithms based on expanding subspaces, due to memory limitations and in order to improve efficiency, the maximum size of the search and test subspaces have to be bounded. Thus, it is necessary to restart the computation whenever the available space for new basis vectors is exhausted. The thick restart technique [Stathopoulos et al., 1998] resets the subspace with the best $m_{\min}$ approximate eigenvectors when its size reaches $m_{\max}$. A related issue is locking of already converged eigenvectors $\widetilde{X}$, a deflation technique that amounts to extracting them from the active basis in order to avoid unnecessary computa-

tion for further improvement. When an eigenpair converges, it is removed from the subspace bases V and W, and in subsequent iterations the new $\mathbf{d}$ vectors are orthogonalized against all locked vectors.

For a more detailed description, the reader is referred to [Sleijpen and van der Vorst, 1996; Fokkema et al., 1999; Sleijpen et al., 1996, 1998].

4.2 Implementation description

In this section, we describe the details of our particular implementation, with special attention to the parallelization and important aspects such as the solution of the correction equation.

4.2.1 Overview of SLEPc

SLEPc, the Scalable Library for Eigenvalue Problem Computations [Hernandez et al., 2005], is a software library for the parallel solution of large-scale, sparse eigenvalue problems. It was designed to solve problems formulated in either standard or generalized form, both Hermitian and non-Hermitian, with either real or complex arithmetic. It can also be used for singular value and quadratic eigenvalue problems.

SLEPc provides a collection of eigensolvers on top of PETSc (Portable, Extensible Toolkit for Scientific Computation, [Balay et al., 2010]), including Krylov-Schur, Arnoldi, Lanczos, Subspace Iteration and Power/RQI. Davidson-type solvers were missing, and this motivated the development of our implementation, which was finally included in SLEPc 3.1 (released in August 2010).

PETSc is a parallel framework for the numerical solution of partial differential equations, whose approach is to encapsulate mathematical algorithms using object-oriented programming techniques in order to be able to manage the complexity of efficient numerical message-passing codes. PETSc is object-oriented in the sense that all the code is built around a set of data structures and algorithmic objects. The application programmer works directly with these objects rather than concentrating on the underlying data structures. The three basic abstract data objects are index sets, vectors and matrices. Built on top of this foundation are various classes of solver objects, including linear, nonlinear and time-stepping solvers. Many different iterative linear solvers are provided, including GMRES, BiCGstab and BiCGstab(ℓ) [Sleijpen and Fokkema, 1993], which can be combined with different preconditioners.

SLEPc eigensolvers rely on the parallel implementation of vector operations, the matrix-vector product and linear equation solvers. Basic implementations of these operations are supplied by PETSc objects. However, certain time-consuming, critical operations have custom implementation in SLEPc in order to improve the overall performance, as explained below.

4.2.2 Parallelization details

The problem matrix A and the vectors of size n, *e.g.*, those stored in V, W, X, $\widetilde{X}$ and R, are distributed by blocks of rows in the corresponding matrices. The rest of vectors and matrices of size bounded by $m_{\max} \ll n$, *e.g.*, the matrices H, U and Z, are replicated in all nodes.

Operations involving distributed operands are parallelized. These include updating W, computing the coefficient matrices of the projected eigensystem H and W^*V, the selected Ritz vectors $\widetilde{X}$ and their residuals R, solving the correction equation (4.4) and orthogonalizing V and W.

The search subspace V is initialized with a basis of randomly generated vectors in parallel, taking care that each processor generates different random sequences.

The orthogonalization is based on a variant of classical Gram-Schmidt with selective reorthogonalization, providing both numerical robustness and good parallel efficiency [Hernandez et al., 2007]. The Schur decomposition of the projected problem and other minor computations are replicated in all nodes.

PETSc only provides basic support for multivectors (a multivector can be seen as a thin tall matrix, or a set of vectors that should be stored contiguously for memory efficiency). In order to develop an optimized version of Jacobi-Davidson it is necessary to implement basic multivector operations using BLAS to perform the local calculations. We provide an implementation in which individual vectors in a multivector can be used in common PETSc functions.

The most time-consuming operations are the multivector inner product W^*V and the update VU. However, PETSc only implements the level 2 BLAS operations $W^*\mathbf{v}_i$ and $V\mathbf{u}_i$. For W^*V, our implementation (i) performs the level 3 BLAS matrix-matrix product of the locally stored parts of V and W on each process, and then (ii) sums up all of them with a single call to an MPI reduction operation. For VU, it performs the BLAS matrix-matrix product of the locally stored part of V and the whole U.

4.2.3 Solution of the correction equation

The correction equation (4.4) is solved using PETSc's Krylov linear solvers, which need to compute matrix-vector products with the coefficient matrix. In this case, performing the shifting and the projections implicitly is more efficient than explicitly building the coefficient matrix. Also, applying only the left projector is sufficient to guarantee the condition $\mathbf{d} \perp \tilde{\mathbf{x}}$, provided that a Krylov solver is used with a zero starting vector, as shown in [Sleijpen et al., 1998].

The trade-off between performance and global convergence is controlled in two ways. First, the maximum number of iterations and the relative residual tolerance can be tuned for the linear system solver. Generally, increasing the number of linear solver iterations (inner iterations) causes a decrease of the global method iterations (outer iterations). In §4.4.1 we will compare the performance of two stopping criteria.

Secondly, the convergence behavior of different Krylov solvers depends on the properties of the problem. We have tested two well-known solvers of this family:

GMRES and BiCGstab(ℓ). They have different parallel behavior because GMRES generally requires less matrix-vector products than BiCGstab(ℓ), but in contrast it has to explicitly maintain an orthonormalized basis whereas BiCGstab does not.

Finally, it is noteworthy that in the first outer steps the pairs resulting from the (harmonic) Rayleigh-Ritz procedure are usually poor approximations of the desired eigenpairs, and the target τ may be a relatively better approximation. Therefore, when the selected eigenpair's associated residual norm is greater than a threshold value *fix*, the correction equation (4.4) is solved with $\tilde{\theta} = \tau$ instead [Fokkema et al., 1999, Section 4.0.1].

In principle, the preconditioning of the correction equation is desirable. However, this issue is not addressed in this chapter because, as we will see below, the application matrix is defined in implicit form, thus preventing from computing conventional preconditioners. Instead it is treated in Ch. 5.

4.3 GENE: A gyrokinetic plasma simulation code

One of the main goals of plasma simulation is to study the micro-instabilities that drive turbulence which in turn produces anomalous transport. This analysis must be done to determine the energy confinement time, a crucial parameter for the design of a fusion reactor.

GENE [Dannert and Jenko, 2005] is a massively parallel plasma simulation code written in Fortran 90/95, which is based on the numerical solution of the gyrokinetic equations. These equations stem from a simplification of the Maxwell-Boltzmann equations by eliminating the fast gyration of ions and electrons in strongly magnetized, dilute plasmas. This periodic motion is not relevant for most investigations, usually focusing on observables related to much slower time scales such as the net particle transport.

The linearized gyrokinetic equation can be written schematically as

$$\frac{\partial g}{\partial t} = \mathcal{L}[g], \tag{4.8}$$

where $\mathcal{L}$ is a time independent, complex, non-Hermitian integro-differential operator. It describes the time evolution of the modified distribution function of the gyrocentres g, which is a (scalar) function of the perpendicular spatial wave vector $(k_x,\ k_y)$, the coordinate z parallel to the magnetic field, the velocity parallel to the magnetic field $v_\parallel$, the magnetic moment μ, and the species label j. The GENE code follows an Eulerian approach. In particular, an explicit Runge-Kutta scheme is used for time integration, while the semi-discretization of phase space variables is done with a fixed grid and a combination of spectral and finite difference techniques. In GENE, the operator matrix is never computed explicitly, but implemented in a highly parallelized and efficient matrix-free form. For reasonably accurate models, the size of the problem ranges from several hundred thousand for linear simulations up to a few billion for nonlinear problems.

In GENE, some selected eigenvalues of the linearized operator need to be computed. In [Roman et al., 2010], the Krylov eigensolvers available in SLEPc are

Table 4.1: Test case I: GENE configuration for a very unstable kinetic ballooning mode with growth rate of 0.2055 and frequency of 0.2872 and another unstable mode $(0.1227 - 0.4494i)$.

Dir.	Resol.	Boxsize		*Geom. & other params.*		Param.	Ions	Electrons
s	2			geom.	circular	R/L_n	2.0	2.0
x	12	l_x	125.628	$\hat{s}$	0.8	R/L_T	3.125	3.375
y	1	$k_{y,min}$	0.25	q_0	1.4	$mass$	1.0	0.00027
z	24			trpeps	0.18	$charge$	1.0	-1.0
v	48	l_v	3.0	β	0.001	T	1.0	1.5
μ	12	l_μ	9.0	(hyp_z, hyp_v)	(2, 0.5)	$dens$	1.0	1.0

Table 4.2: Test case II: GENE configuration similar to test case I but with a more realistic species configuration: deuterium, tritium, helium and electrons.

Dir.	Resol.	Boxsize		*Geom. & other params.*		Param.	^{2}H	^{3}H	He	e^-
s	4			geom.	circular	R/L_n	2.5	2.5	2.5	2.5
x	12	l_x	auto	$\hat{s}$	0.8	R/L_T	3.5	3.5	3.5	4.0
y	1	$k_{y,min}$	0.25	q_0	1.4	$mass$	2.014	3.016	4.002	5.4e-4
z	24			trpeps	0.18	$charge$	1	1	2	-1
v	48	l_v	3.0	β	0.001	T	1.0	1.0	1.0	1.5
μ	8	l_μ	9.0	(hyp_z, hyp_v)	(2, 0.5)	$dens$	0.45	0.45	0.05	1.0

used for this. One scenario is the computation of the largest magnitude eigenvalue (circled in the spectrum of Fig. 4.1) in order to estimate the optimal timestep of the initial value solver. In this case, Krylov solvers converge very fast. In a different context, SLEPc is also used for computing the subdominant unstable modes, i.e., the rightmost eigenvalues (crosses in Fig. 4.1). Due to the shape of the spectrum, these eigenvalues are much more difficult to compute. In [Roman et al., 2010], it is shown that the Krylov-Schur method with harmonic extraction has a reasonably good performance, compared to plain Krylov-Schur with spectral transformation. In the next section, we will show that our Jacobi-Davidson implementation performs even better. Computing these rightmost eigenpairs very fast is critical for some kind of analyses, e.g., when tracking the subdominant modes for varying values of several parameters [Merz and Jenko, 2010], in which case a sequence of eigenproblems has to be solved.

4.4 Computational results

This section summarizes the experiments carried out in order to evaluate the performance of our implementation, particularly in terms of scalability to a large number of processes.

The experiments are executed on Tirant, a machine consisting of 256 JS20 blade computing nodes, each of them with two 64-bit PowerPC 970+ processors

Table 4.3: Test case III: GENE configuration corresponding to a stellarator device.

Dir.	Resol.	Boxsize		*Geom. & other params.*		Param.	Ions	Electrons
s	2			geom.	tracer	R/L_n	0.0	0.0
x	3	l_x	auto	file	hm128.dat	R/L_T	4.0	0.0
y	1	$k_{y,min}$	0.3	$\hat{s}$	-0.1088	$mass$	1.0	0.0025
z	128			q_0	1.11	$charge$	1.0	-1.0
v	48	l_v	3.0	β	0.001	T	1.0	1.0
μ	12	l_μ	9.0	(hyp_z, hyp_v)	(5, 0.5)	$dens$	1.0	1.0

running at 2.2 GHz, and interconnected with a low latency Myrinet network. Only 256 processors are used due to account limitations.

The following software is employed: GENE 1.4, PETSc 3.1, SLEPc 3.1 and LAPACK 3.2.1. All of them are built with the IBM compilers XL for C and Fortran, and linked with the BLAS routines in ESSL and MPICH 1.2.7.

Three different GENE parameter settings are used, detailed in the Tables 4.1, 4.2 and 4.3. These can be considered real use scenarios. In our previous work [Romero and Roman, 2010], only a lower resolution version of the test case 4.1 was employed. The GENE parameters shown in the tables determine the size and the spectrum of the associated eigenproblem (which influence the convergence of the solvers), and change the performance of matrix-vector products (which is an important part of the overall performance).

All experiments in this section are run using the default domain distribution (how many groups of processes there are in each direction) computed by GENE. For instance, when running test case II with 64 processes the default is to split the s, x, y, z, v, and μ directions in 4, 2, 8, 1, 1, 1, respectively. The impact of different domain distributions on parallel performance is studied in [Roman et al., 2010] and [Romero and Roman, 2010].

The Jacobi-Davidson solver will be compared with the fastest alternative found in [Roman et al., 2010], that is the Krylov-Schur method with harmonic extraction, which is available in SLEPc. Both solvers are configured for computing the two eigenvalues with largest real part (that correspond to the two instabilities with largest growth rates), with a tolerance of 10^{-5} for the residual norm relative to the magnitude of the eigenvalue. The search subspace is limited to 64 vectors, and when it is full, Krylov-Schur restarts with 32 vectors whereas Jacobi-Davidson keeps only 8. We have used block size 1, since larger values do not improve the performance in this case.

The harmonic procedure in both eigensolvers needs a target (called τ in Algorithm 4.1), that is, a point in the complex plane whose nearest eigenvalues are the desired ones. In the experiments the target is set to 1, which is a reasonable assumption for the upper bound for the real part of the eigenvalues for the problem at hand.

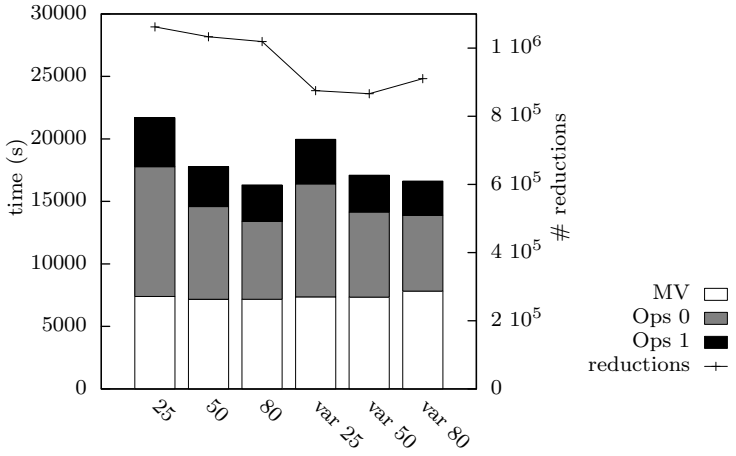


Figure 4.2: Time (in seconds, left axis) and number of reductions (right axis) spent by Jacobi-Davidson solving the test case I with four processes. The correction equation is solved with BiCGstab(2) and a maximum number of iterations of 25, 50 and 80. The cases labeled as "var" use a variable tolerance for the stopping criterion. The total time is split into matrix-vector products (MV) and vector operations without (Ops 0) and with (Ops 1) communication.

4.4.1 Stopping criterion for the correction equation solver

As already remarked, the way in which the correction equation is solved has a significant impact on the overall performance of the method. Our study in [Romero and Roman, 2010] compares the performance of Jacobi-Davidson when solving the correction equation with 110 iterations of GMRES and BiCGstab(2), showing that the latter obtained better results (this can be attributed to the high overhead of basis orthogonalization in GMRES). In this chapter, we extend the analysis by testing more flexible criteria with BiCGstab(2).

Figure 4.2 shows the time spent by Jacobi-Davidson using four processes when solving test case I with the BiCGstab(2) solver configured to perform 25, 50 and 80 (inner) iterations. We also consider modified versions of these configurations using a variable tolerance, that is, the iterative solution of the correction equation is stopped earlier if

$$\|r^{(j)}\|_2 \leq 2^{-i}\|r^{(0)}\|_2,$$

where $r^{(j)}$ is the residual of the correction equation at the linear solver iteration j, and i is the current outer iteration. This criterion comes from Newton methods and its use in the context of Jacobi-Davidson is suggested in [Fokkema et al., 1999], and is also used in [Genseberger, 2010].

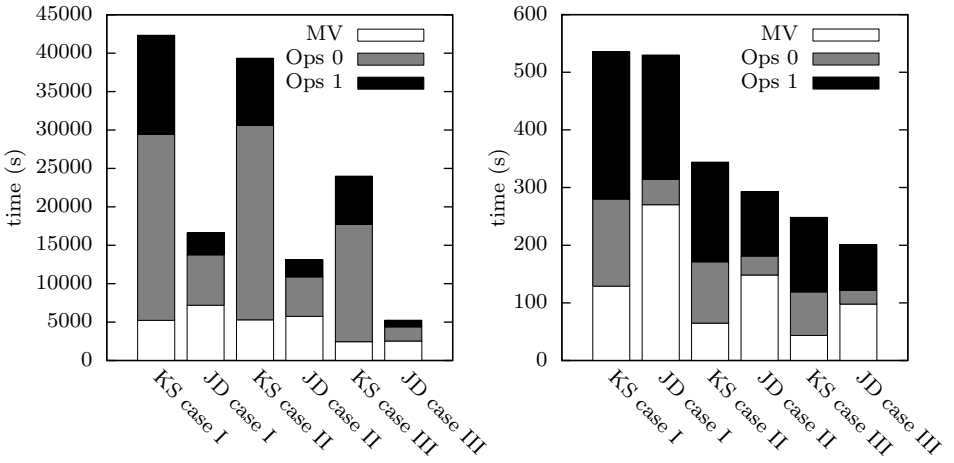


Figure 4.3: Time (in seconds) using Krylov-Schur (KS) and Jacobi-Davidson (JD) with BiCGstab(2) solving test cases I, II and III with 4 (left) and 256 (right) processes. The total time is split into matrix-vector products (MV) and vector operations without (Ops 0) and with (Ops 1) communication.

From the figure, we observe that the best times are obtained limiting the number of iterations to 80, and in general the more iterations, the better time. In all cases, the variable tolerance criterion slightly reduces the time and, more importantly, significantly reduces the number of parallel reductions. Parallel reductions require the synchronization of all processes and constitute a great penalty for the scalability of the method. If we repeat the analysis for a larger number of processes, we will see that using more iterations may be counterproductive because overhead associated to collective communication becomes more important. Therefore, in subsequent experiments we use the variable tolerance criterion with a maximum number of inner iterations of 50.

4.4.2 Jacobi-Davidson versus Krylov-Schur

Figure 4.3 compares the results of Jacobi-Davidson and Krylov-Schur for test cases I, II and III. With four processes (left plot) the advantage of Jacobi-Davidson is clear. However, the difference between both methods is reduced significantly when the number of processes increases, as can be appreciated in the results with 256 processes (right plot) and in the speedups in Figure 4.4 commented below.

One of the possible causes is that Jacobi-Davidson needs more than twice as many matrix-vector products as Krylov-Schur (see Table 4.4), but the order in which they are performed (quite consecutive compared to Krylov-Schur) allows the cache to reduce this penalty. When the local problem size is small enough,

Table 4.4: Total number of matrix-vector products (#MV), total time spent by them (T. MV) in seconds, average time spent by one product (T./#MV), and number of reductions (#red.) performed by both methods when solving the test cases with four processes.

	Krylov-Schur				Jacobi-Davidson			
Case	#MV	T. MV(s)	T./#MV(s)	#red.	#MV	T. MV(s)	T./#MV(s)	#red.
I	119776	5226.2	0.0436	177000	297147	7185.15	0.0241	865900
II	67136	5299.4	0.0789	100700	169685	5754.05	0.0339	494000
III	45216	2439.6	0.0539	66960	110499	2527.2	0.0228	321100

Krylov-Schur performs the matrix-vector product as fast as Jacobi-Davidson and the excess of matrix-vector products in Jacobi-Davidson is reflected in the time (see matrix-vector product times in the bottom right plot of Figure 4.4).

Another explanation is the fact that Jacobi-Davidson performs approximately five times more parallel reductions than Krylov-Schur (see Table 4.4). Some parallel reductions are performed in the orthogonalization procedure and others in the iterations of the solution of the correction equation. The latter is the main source of parallel reductions in Jacobi-Davidson, and keeping their number small is necessary for a competitive implementation.

4.4.3 Speedup and Scalability

Figure 4.4 illustrates the speedups, in the strong scaling sense, of Jacobi-Davidson and Krylov-Schur solving the test cases I, II and III. The time spent by Jacobi-Davidson in four processes is selected as the reference time for the speedup in each test case. The bottom right plot shows the speedup of the matrix-vector product taking as basis the performance in Jacobi-Davidson.

We can observe that there is a clear connection between the speedup of the matrix-vector product and the total speedup. The reason is that this operation approximately accounts for 50% of the total time in Jacobi-Davidson and just 33% in Krylov-Schur, with 256 processes. This can explain the poor speedup results for test case I. In each test case, the trend of the global speedup inherits the trend of the matrix-vector product as the number of processes grows.

Finally, Figure 4.5 illustrates the weak scaling scenario, plotting the time spent by the eigensolvers in the solution of the cases I (left plot) and II (right plot) with the resolution of the v direction increased. It is observed that both solvers are comparably good, with the exception of the anomalous and punctual malfunction of Krylov-Schur with 16 and 32 processes due to extremely slow convergence (this may not happen with a different set of parameters).

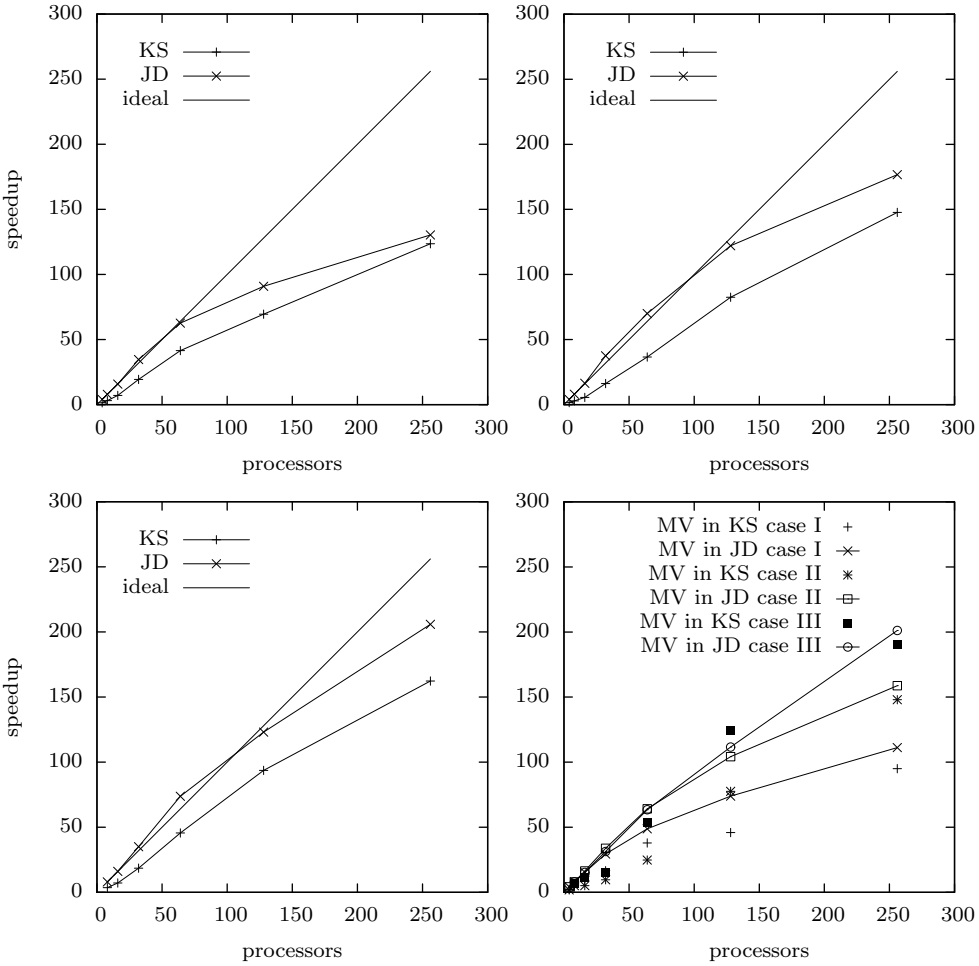


Figure 4.4: Speedups solving the test cases I (top left), II (top right) and III (bottom left) with Jacobi-Davidson and Krylov-Schur. Speedup of the matrix-vector product in these problems (bottom right).

4.5 Conclusions

We have presented a parallel implementation of the Jacobi-Davidson eigensolver for complex non-Hermitian matrices. The proposed solver incorporates all the ingredients necessary to be competitive with other solvers, such as restart and locking with Schur vectors. It is also equipped with harmonic extraction for finding interior eigenvalues, and several options for the efficient solution of the correction equation (projectors, preconditioning, and the fix parameter). The implementa-

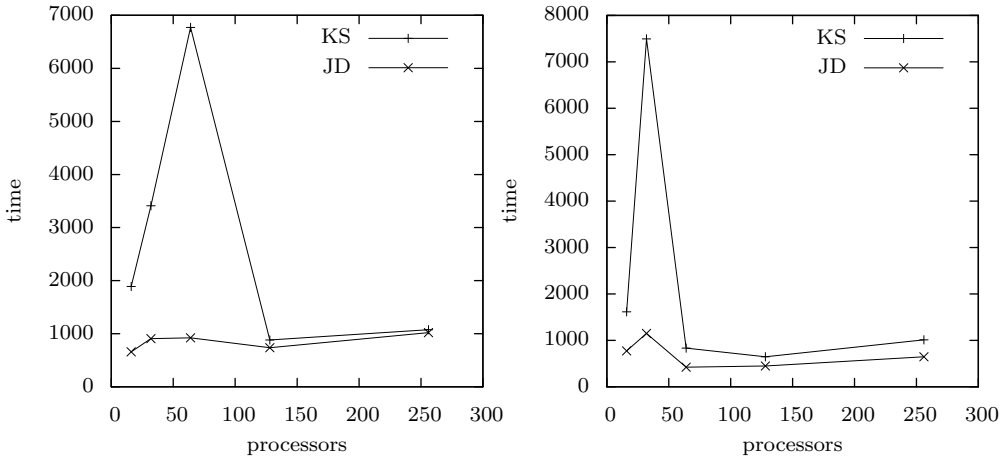


Figure 4.5: Time (in seconds) solving versions of the test cases I (left) and II (right) with increasing resolution in the v direction, with Jacobi-Davidson and Krylov-Schur.

tion has been carried out in the context of SLEPc, where the user is able to easily adjust the different parameters for the best performance.

In order to analyze the performance of the new solver, we have addressed a relevant scientific computing application, namely the computation of micro-instabilities in fusion plasmas as implemented in the GENE code. This application requires computing the rightmost eigenvalues (unstable modes) of a discretized advection dominated partial integro-differential equation, where the matrix has almost pure imaginary eigenvalues. This problem is a challenge for iterative eigensolvers.

The comparison, in terms of time, with the harmonic Krylov-Schur method is very favorable, being Jacobi-Davidson up to four times faster in some cases. However, this gain is diminished when the local problem size is small, that is, both methods become nearly equivalent when increasing the number of processes, although in absolute terms Jacobi-Davidson is still faster, at least up to 256 processes with the problems tested. The reason for this behavior is that the Jacobi-Davidson eigensolver, in order to be competitive with respect to Krylov-Schur, needs to perform many inner iterations. In this way, the cost is dominated by the matrix-vector product operation. In the considered application, the parallel efficiency of the matrix-vector product is rather variable, depending on the configuration of the GENE parameters.

In terms of practical use, we have shown that Jacobi-Davidson is a competitive method for GENE, even without using a preconditioner. It remains as a topic for further research the addressing of the correction equation preconditioning by,

for instance, designing a specific preconditioner for the GENE linearized operator (see Ch. 5). Furthermore, it may be interesting to consider other iterative methods for solving the correction equation, such as deflated restarting GMRES [Morgan, 2002], and advanced stopping criteria, such as those proposed in [Hochstenbach and Notay, 2009].

Chapter 5

Use of Preconditioners and Initial Guesses in GENE

In magnetically confined high temperature plasmas as they occur in fusion experiments, temperature and density profiles are determined by turbulent transport. Given that the relevant time scales are usually clearly above the particles' gyration times, this so-called microturbulence can be described in the framework of gyrokinetic theory [Brizard and Hahm, 2007] which is a reduced kinetic model, neglecting the fast gyrophase dependence. It describes the plasma as a collection of quasi-particles (charged rings) in a five-dimensional phase space, coupled via a modified form of Maxwell's equations. Assuming that the system size clearly exceeds the radial correlation length of the turbulence, it is common to make a (radially) local approximation, reducing the simulation volume to a thin flux tube [Beer et al., 1995]. Moreover, if one is only interested in the microinstabilities which drive the turbulence, the gyrokinetic equations may be linearized. While greatly reducing the overall computational effort, this still allows to make valuable predictions concerning the expected properties of the resulting turbulent transport.

In local gyrokinetics, the time evolution of the modified distribution function g of the gyrocenters can schematically be written as [Kammerer et al., 2008]

$$\partial_t g = Lg + N[g]\,.$$

Here, the distribution function g is a function of the two spatial coordinates (k_x, k_y) perpendicular to the background magnetic field, the parallel coordinate z, the two velocity space coordinates (parallel velocity and magnetic moment) $(v_\parallel, \mu)$, the species index s, and time t. L is the linear gyrokinetic operator and $N[g]$ is the quadratic $\mathbf{E} \times \mathbf{B}$ nonlinearity; both operators are of integro-differential form.

The turbulence in the nonlinear system is driven by linear instabilities, i.e., eigenmodes of L with positive real part of the eigenvalue. Investigations of the growth rate and frequency (i.e., real and imaginary parts of the eigenvalue) of these instabilities, which occur owing to temperature and density gradients of

the background, already give some information about the behavior of the system. Furthermore, the eigenvalues and -vectors can be used to construct quasilinear models (see, e.g., [Merz and Jenko, 2010]).

The linear operator L is block diagonal in k_y and only couples certain k_x values. The problem size of a linear computation is very much reduced compared to a nonlinear simulation, where the nonlinearity couples all values in the k_x, k_y plane. Linear investigations are therefore computationally much less demanding than full nonlinear turbulence simulations. This can be exploited to perform high dimensional parameter scans, which allows, e.g., for checks of the robustness of a simulation result with respect to variations about a nominal set of parameters, predictive simulations of fusion plasmas, and the optimization of experimental parameters.

For a general set of background gradients, several modes are unstable. While the most unstable mode is usually the most interesting one for quasilinear models, parameter variations lead to variations of the growth rates and therefore to transitions of the most unstable mode. The most unstable mode can be computed both as initial and eigenvalue problem, but mode transitions can only be monitored if an eigenvalue solver is used. Furthermore, the computation time for the initial value approach diverges exactly at a mode transition. Since the results of [Merz and Jenko, 2010] suggest that subdominant modes only contribute to the nonlinear properties if they are similar in growth rate to the most unstable mode, only the dominant and the first subdominant mode are considered.

This chapter is organized as follows. In the next section, we introduce the equations solved in the gyrokinetic GENE code and present the test case which will be used throughout this chapter. In Section 5.2, the interface between the GENE code, which implements the gyrokinetic equations, and the SLEPc library, which is used for the eigenvalue computations, is described, with focus on the recently implemented Jacobi-Davidson solver and the preconditioner, which is necessary for good performance. In Section 5.3, we discuss strategies to efficiently process large numbers of eigenvalue computations, including subspace recycling and parallelization. The capability of the resulting setup is demonstrated for an example in Section 5.4. Finally, Section 5.5 closes with a summary.

5.1 The GENE code

Since the nonlinear gyrokinetic equations generally do not allow for analytic solutions, they have to be solved numerically. A state-of-the-art gyrokinetic solver is provided by the GENE code [Jenko et al., 2000; Dannert and Jenko, 2005; Merz, 2008; Görler et al., 2011]. GENE is physically comprehensive and flexible, computationally efficient, and hyperscalable. GENE is being further developed by an international team and is freely available. More details can be found on the GENE website `http://gene.rzg.mpg.de`.

In the context of the present work, we will focus on the linearized gyrokinetic equations as implemented in GENE. We start by noting that the linear gyrokinetic

operator is a complex, non-Hermitian integro-differential operator. It can be split in two parts

$$L = L_g + L_\chi \,,$$

where

$$L_g = -\frac{T_{0s}(2v_\parallel^2 + \mu B_0)}{q_s B_0}\left(K_y i k_y + K_x i k_x\right) - \frac{v_{Ts}}{JB_0} v_\parallel \frac{\partial}{\partial z} + \frac{v_{Ts}}{2JB_0}\mu \partial_z B_0 \frac{\partial}{\partial v_\parallel}$$

is a differential operator acting directly on g, and L_χ is a more complicated operator that contains the various derivatives of the (gyro-averaged) electromagnetic fields. It can be written as

$$\begin{aligned} L_\chi g = &-\left(\omega_n + (v_\parallel^2 + \mu B_0 - \frac{3}{2})\omega_{Ts}\right) F_{0s} i k_y \chi_s - \frac{2v_\parallel^2 + \mu B_0}{B_0} F_0 \left(K_y i k_y + K_x i k_x\right)\chi_s \\ &- \frac{v_{Ts}}{JB_0} v_\parallel \frac{q_s}{T_{0s}} F_0 \partial_z \chi_s - \frac{2q_s}{m_s JB_0} v_\parallel^2 \mu F_0 \bar{A}_{\parallel s}(\partial_z B_0) \\ &- \frac{q_s}{m_s JB_0}\mu(\partial_z B_0)\bar{A}_{\parallel s}(\partial v_\parallel v_\parallel F_{0s}), \end{aligned}$$

where

$$\chi_s = \bar{\phi}_s - v_{Ts} v_\parallel \bar{A}_{\parallel s}$$

is a combination of the electromagnetic fields ϕ and $A_\parallel$ (the bars denote gyro-averaging). Its dependency on the species index s is introduced by the gyro-averaging operator. The fields are computed from g by the linear operators

$$\begin{aligned} \phi =& \frac{\sum_s n_{0s}\pi q_s B_0 \int J_0(\lambda_s) g_s dv_\parallel \, d\mu}{k_\perp^2 \lambda_D^2 + \sum_s \frac{q_s^2}{T_{0s}} n_{0s}(1 - \Gamma_0(b_s))} \\ A_{1\parallel} =& \frac{\sum_s \frac{\beta}{2} q_s n_{s0} v_{Ts} \pi B_0 \int v_\parallel J_0(\lambda_s) g_s(\vec{k}) dv_\parallel \, d\mu}{k_\perp^2 + \sum_s \frac{\beta q_s^2}{m_s} n_{0s} \pi B_0 \int v_\parallel^2 J_0^2(\lambda_s) F_{0s} dv_\parallel \, d\mu} \,. \end{aligned}$$

For the definitions of the prefactors, see [Merz, 2008]. The derivatives are discretized with (centered) finite differences in GENE, leading to a banded structure of L_g. The field operators contain integrals in the $v_\parallel$, μ and s coordinates and therefore leads to a large bandwidth of L_χ, which is inherited by L.

Since a computation based on an explicit representation of a matrix with this structure would be very inefficient, the operator is implemented in a matrix-free form in GENE, exploiting the knowledge about the integro-differential structure of the operator.

The default parameter set that will be used as a test case throughout this chapter is specified in Table 5.1. It corresponds to the parameter set 4 of the SLEPc testsuite provided by GENE. For this parameter set, a dominant ion temperature gradient (ITG) mode and a subdominant collisionless trapped electron mode (TEM) can be observed. To simplify the notation, we represent the $v_\parallel$ coordinate as v in Table 5.1 and in the rest of the chapter.

Table 5.1: Test case I: GENE configuration for an ITG mode with growth rate of 0.2055 and frequency of 0.2872 and a subdominant TEM ($0.1227 - 0.4494i$).

Dir.	Resol.	Boxsize		*Geom. & other params.*		Param.	Ions	Electrons
s	2			geom.	circular	R/L_n	2.5	2.5
x	5			$\hat{s}$	0.8	R/L_T	3.5	4.0
y	1	$k_{y,min}$	0.25	q_0	1.4	$mass$	1.0	0.00027
z	16			trpeps	0.18	$charge$	1.0	-1.0
v	48	l_v	3.0	β	0.001	T	1.0	1.5
μ	8	l_μ	9.0	(hyp_z, hyp_v)	(2, 0.5)	$dens$	1.0	1.0

5.2 Fast eigenvalue computations

The GENE code was coupled to the SLEPc package several years ago and since then it has been routinely used to compute the spectral radius of the linear operator. This allows the exact determination of the maximum allowed time step for the Runge-Kutta scheme used in initial value computations. Apart from this rather technical application, the investigation of a selected subset of eigenvalues and eigenvectors is of great physical interest and can be used, e.g., in the context of quasilinear models (see, e.g., [Dannert and Jenko, 2005; Jenko et al., 2005; Merz and Jenko, 2010]). Of obvious interest are the unstable eigenmodes (i.e., eigenmodes with positive real part), because they drive the turbulent transport in fusion plasmas.

The approach presented here can be used to compute any part of the spectrum (critical gradients, stable eigenmodes that are relevant for the saturation of the nonlinear system). It is well known that eigensolvers have much more difficulties, in terms of convergence, when computing interior eigenvalues, compared to eigenvalues in the periphery of the spectrum. In the case of unstable modes, the eigenvalues of interest are the rightmost ones, which in our case are as difficult to compute as interior eigenvalues because the spectrum is very elongated along the imaginary axis, and the few modes with positive growth rate are relatively close to the origin.

Because of the integro-differential structure discussed in the previous section, the linear operator L has a banded pattern only in two (k_x and z) of the five dimensions, with the velocity space and species dimensions completely filled. For our test case, this corresponds to a matrix with almost 4000 non-zero diagonals for a matrix dimension of around 60000 (here, a non-zero diagonal is a diagonal k consisting of entries a_{ij} with $|i - j| = k$ where some or all of the entries are different from zero).

The iterative solvers in SLEPc only require the matrix-vector product of a test vector with the linear operator for the computation of the eigenvectors. This means that no explicit matrix representation has to be computed. SLEPc can

directly use the matrix-vector product with L which is also used for initial value computations in GENE.

Previous efforts to improve the computation of these rightmost eigenvalues in SLEPc resulted in the implementation of the harmonic projection method for the Krylov-Schur solver [Roman et al., 2010], which allowed for the discovery of non-Hermitian degeneracies of gyrokinetic eigenmodes [Kammerer et al., 2008].

Recently, a Jacobi-Davidson solver has been implemented in SLEPc (see Ch. 4). The performance of this solver depends, in contrast to the previously used solver, on effective preconditioning methods for the correction equation. We next give details related to this approach.

5.2.1 Eigenvalue solver

Iterative eigensolvers are usually based on a projection onto a search subspace of increasing dimension. The *expansion* of the subspace is done by computing a new vector at each iteration, until a maximum dimension is reached (then the method is *restarted*). At each iteration, eigenvalue approximations can be obtained from the subspace, either with a Rayleigh-Ritz procedure or other *extraction* methods such as the aforementioned harmonic projection.

In some cases, Krylov methods are limited by the fact that the built subspace has to maintain the Krylov structure. As a consequence, convergence can be extremely slow in difficult problems such as the ones discussed in this chapter.

An alternative to Krylov methods are Davidson-type methods, that do not impose any restriction on the subspace and can thus expand the subspace with the "best" vector according to some criterion. In particular, these methods choose one of the eigenvalue-eigenvector approximations (θ, u) contained in the subspace (*e.g.*, the eigenvalue closest to the target τ specified by the user), then form the residual vector associated to it, $r = Au - \theta u$, and finally compute the so-called correction vector t that will be added to the subspace.

This new vector can be computed by simply preconditioning the residual,

$$t = K^{-1}r, \tag{5.1}$$

as in the Generalized Davidson (GD) method [Morgan, 1992], where the preconditioner K can be viewed as a rough approximation of $A - \theta I$. However, in difficult problems this simple approach is not effective enough. The more sophisticated Jacobi-Davidson (JD) method [Sleijpen and van der Vorst, 2000] computes t by (approximately) solving the so-called correction equation: a system of linear equations involving the matrix A, the preconditioner K, and a projector P related to K and the approximate eigenvector u. In particular, in this chapter we use

$$PK^{-1}\left(A - \theta I\right)Pt = -\hat{r}, \quad P = I - \frac{K^{-1}zu^*}{u^*K^{-1}z}, \quad t \perp u, \tag{5.2}$$

where $z \in \text{span}\{Au, u\}$ and $\hat{r} = PK^{-1}r$. Furthermore, we employ algorithmic techniques similar to the JDQZ variant [Fokkema et al., 1999], in order to enable

the use of harmonic extraction in a numerically stable way. Additional details about the algorithm and its use in the context of the GENE code can be found in Ch. 4, except for the preconditioning which will be treated in this chapter.

Another drawback of Krylov methods is that they start building the subspace from a single vector. If one has an a priori knowledge of a rough approximation of the wanted eigenspace, e.g., from a closely related eigenproblem, then this knowledge cannot be exploited. In contrast, Davidson methods can indeed benefit from using a rough approximation of the solution as initial guess. The explanation is that Davidson methods can be viewed from the perspective of inexact Newton schemes [Wu et al., 1998]. Thus, a good starting solution can improve convergence considerably, with the corresponding reduction of the overall cost. We will exploit this fact in parameter scans, see §5.3.

5.2.2 Overview of SLEPc and PETSc

SLEPc, the Scalable Library for Eigenvalue Problem Computations, is a software package for the solution of large-scale eigenvalue problems on parallel computers. It can be used to solve a variety of eigenvalue problems, including standard and generalized problems, both Hermitian and non-Hermitian, as well as other types of problems such as the quadratic eigenvalue problem or the singular value decomposition. SLEPc can work with either real or complex arithmetic, in single or double precision.

SLEPc offers a number of iterative eigensolvers, as described in the previous subsection. In particular, it provides a parallel implementation of the Krylov-Schur method, as well as GD and JD solvers, with various possibilities for the computation of the correction vector. In the Davidson-type methods (GD and JD), the user can easily select which preconditioner to use, via PETSc as described below.

SLEPc is built on top of PETSc, a parallel framework for the numerical solution of partial differential equations, which is based on defining basic abstract data objects such as vectors and matrices, and building solver objects on top of them, including linear, nonlinear and time-stepping solvers. SLEPc inherits all the good properties of PETSc, including portability to a wide range of parallel platforms, scalability to a large number of processors, and run-time flexibility giving full control over the solution process (one can for instance specify the solver at run time, or change relevant parameters such as the tolerance or the size of the subspace basis).

For the solution of linear systems, PETSc provides a list of iterative solvers such as GMRES, together with a variety of preconditioners including Jacobi (diagonal) preconditioning, and block Jacobi/additive Schwarz (with a choice of incomplete factorizations for the blocks). See [Saad, 2003] for details about the algorithms. It is also possible to use preconditioners available in third-party packages that are seamlessly integrated into PETSc.

Both in SLEPc and PETSc, iterative solvers can be employed in a matrix-free manner, that is, accessing the matrix only via matrix-vector product operations.

However, this limits part of the functionality, most notably the construction of preconditioners.

5.2.3 Approximate explicit matrix representation for the preconditioner

Most preconditioning techniques are based on explicitly building a preconditioner based on information about the individual entries of the matrix, e.g., computing an incomplete factorization or a sparse approximate inverse. Those techniques are not viable to compute a preconditioner for a matrix-free operator. Only methods that are based solely on the information collected from matrix-vector products could be used, for instance using a Krylov iterative solver as a preconditioner. However, our experience with these nested Krylov techniques indicates that they are not competitive, at least for our application.

Since an explicit representation of the full linear operator L cannot be used for the reasons given above, we have opted for constructing the preconditioner from the explicit representation of L_g, which can be viewed as a rough (sparse) approximation of L. The bandwidth of this operator is much smaller (only 9 diagonals for our test case), but it still contains important contributions like, e.g., the parallel electron dynamics, which usually is the dominant advection term of the linear operator and therefore largely determines its spectral radius. The L_g matrix is stored in parallel sparse matrix format provided by PETSc, and the time for its computation is negligible.

We next describe the two preconditioning techniques that we have tested, namely additive Schwarz and parallel ARMS.

5.2.4 ASM+ILU preconditioner

Having an explicit representation of the matrix L_g, preconditioners can be built with the standard PETSc packages. The linear system has to be solved in parallel and thus it has to be distributed to the different MPI processes. A first step would be to compute a preconditioner from a block diagonal approximation by $L_g \approx \sum_i R_i L_g R_i$ with R_i being a diagonal matrix having ones only for the indexes belonging to the ith subdomain. For the additive Schwarz method (ASM) [Cai and Sarkis, 1999] even points outside the domain are added if they have a neighbor of δth order being inside the domain. Thus the differential operator L_g can be approximated by

$$L_g \approx \sum_i L_{gi}^{\delta} = \sum_i R_i^{\delta} L_g R_i^{\delta} \tag{5.3}$$

with R_i^{δ} being the restriction operator involving also the δth order neighbor.

The overlap δ is thus representing the interaction between neighboring subdomains and is thus a measure of the required communication between the subdomains. Since the linear operator L_g is just containing a few diagonals, the number of δth order neighbors is rather small and requiring few computational resources for communication. The main idea of that domain decomposition is to create a

preconditioner K being a sum of preconditioners K_i, which are themselves constructed from the L_{gi}^{δ}. The overall system to solve is then

$$\sum_i K_i\left(L_{gi}^{\delta}\right) L_g x = \sum_i K_i\left(L_{gi}^{\delta}\right) b \tag{5.4}$$

which can be done in parallel since both the computation of the preconditioner and the evaluation of the linear gyrokinetic operator are distributed on the respective processes via MPI and PETSc.

The explicit representation of L_{gi}^{δ} allows the construction of an incomplete LU (ILU) decomposition [Saad, 2003] $\tilde{L}_i\tilde{U}_i$ with its inverse being computable cheaply via forward/backward substitution. This allows the construction of a preconditioner by

$$K = \sum_i K_i = \sum_i \left(\tilde{L}_i\tilde{U}_i\right)^{-1} \approx \sum_i \left(L_{gi}^{\delta}\right)^{-1} \approx L_g^{-1} \tag{5.5}$$

in parallel. Doing only an incomplete LU decomposition has the advantage of preserving the sparsity of L_{gi}^{δ}, since a full decomposition would lead to a large fill-in which is requiring a lot of additional memory. PETSc allows one to set a maximum level of fill-in, which is limiting the creation of entries from other filled in values to preserve the sparsity pattern. Also the ILU decomposition creates less fill-in if the ordering of the matrix is optimized. Different reorderings exist and the quotient minimum degree [George and Liu, 1980] reordering seemed to provide the best results for our purposes. All mentioned algorithms are provided by PETSc and could thus be easily connected with the eigenvalue computation in SLEPc.

5.2.5 pARMS preconditioner

The pARMS preconditioner [Li et al., 2003; Sosonkina et al., 2004] is a parallel, multi-level preconditioner based on the Schur complement and algebraic recursive multilevel solver (ARMS) techniques.

Given a system of linear equations $Ax = b$ that is written in block form

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \tag{5.6}$$

the idea is to compute an incomplete block LU decomposition of A as

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} \approx \begin{bmatrix} L & 0 \\ EU^{-1} & I \end{bmatrix} \begin{bmatrix} U & L^{-1}F \\ 0 & S \end{bmatrix}, \tag{5.7}$$

where LU is an incomplete factorization of B and the Schur complement matrix is $S = C - (EU^{-1})(L^{-1}F)$.

As in the case of Schwarz preconditioners, pARMS is also based on the domain decomposition idea. In this case, all the unknowns interior to the different subdomains are placed in the x_1 part in (5.6), whereas the x_2 part contains unknowns

corresponding to the interface between subdomains. Therefore, a permutation is required for reordering the unknowns. The ARMS method consists in applying the permutation and incomplete factorization to the Schur complement S recursively for a given number of levels. In parallel, pARMS distributes the available subdomains across processors. For further details, see [Li et al., 2003; Sosonkina et al., 2004].

There is an MPI implementation of the pARMS preconditioner[1]. As part of this work, we have integrated it as an external package in PETSc 3.2.

5.2.6 Results for one parameter set

We now present results from some experiments to evaluate different eigensolver configurations. The tests are executed on HPC-FF, a Linux cluster of 1080 nodes composed of two Intel Xeon X5570 (Nehalem-EP) Quad-Core processors at 2.93 GHz and 24 GB of DDR3 memory at 1066 MHz, and interconnected by Infiniband QDR with non-blocking Fat Tree topology.

The results correspond to GENE 1.5 linked with versions 3.2 of PETSc and SLEPc. All code is compiled with Intel C and Fortran Compilers 11.1.

The parameter set used is detailed in Table 5.1.

The SLEPc JD eigensolver is configured to compute the two eigenvalues closest to the target $\tau = 1$, with a relative tolerance of 10^{-5}. The search subspace is bounded to 64 vectors and when it is complete, the method restarts with 5 vectors. The correction equation is solved in a maximum of 300 iterations of BiCGstab(2) and with a tolerance of 10^{-8}, accelerated by a preconditioner $K^{-1} \approx (L_g - \sigma I)^{-1}$ with σ being a constant value (to avoid recomputing the preconditioner at each iteration). Usually, the shift σ is taken to be equal to the target τ, but in our experiments we observe a small improvement by taking slightly larger values of σ than τ (see Figure 5.1 (right)), so we set $\sigma = 3$ as the default value.

Optimal settings of the ASM preconditioner

The L_g matrix exhibits a block diagonal structure in the dimensions s and μ. When these dimensions prevail in the distribution, the resulting domains become quite unconnected and the block Jacobi preconditioner is effective. For other decompositions that have more connected domains, ASM can provide better preconditioners (in terms of convergence), but with more time-consuming application, due to the requirement of taking into account the neighbors of the order determined by the overlap δ.

Of course, the most efficient overlap value depends on the problem settings and the distribution. However, we obtained good results with an overlap $\delta = 2$ if fine-grained local preconditioners are used. Figure 5.1 (left) compares the performance of JD solving the test case I with different overlap values, using ASM with ILU as the local preconditioner.

[1] `http://www-users.cs.umn.edu/~saad/software/pARMS/`

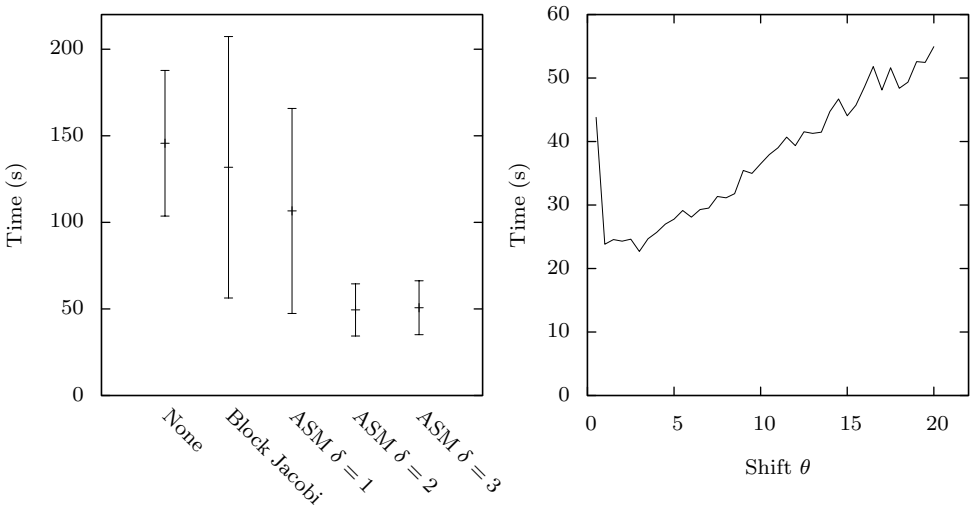


Figure 5.1: Influence of the overlap δ (left) and the shift σ of the preconditioner matrix (right) on the total time. The plots show the mean and the standard deviation (left) and the minimum time (right) spent by JD with different domain decompositions.

Whereas previous solvers could rely on GENE's internal automatic optimization of the domain decomposition for a fast evaluation of L, this choice might not be optimal for the ASM (and Block-Jacobi) preconditioner. Tests have shown that any decomposition in the z and v directions leads to a significant drop in performance due to increased communication, with the decomposition in z behaving even worse than the one in v (see some examples in Table 5.2 and the standard deviations of the time in Figure 5.1 (left)). Care has to be taken that the domain decomposition is chosen in a way that the s and μ directions are decomposed first, followed by a decomposition in the v direction.

Besides the optimal configuration of the preconditioner, the maximum iteration of the Jacobi-Davidson solver has to be changed to achieve optimal runtimes. If the ASM+ILU preconditioner is applied, five iterations of the BiCGstab algorithm lead to a sufficient accuracy in solving the correction equation to achieve convergence of the Jacobi-Davidson algorithm in minimal time.

Table 5.2: Time (in seconds) spent by JD with ASM+ILU solving the test case I with different distribution of processes across the directions s, z, v and μ.

s	z	v	μ	Time	s	z	v	μ	Time	s	z	v	μ	Time
1 processor					4 processors					32 processors				
1	1	1	1	73.29	2	1	1	2	24.69	2	1	2	8	3.88
2 processors					1	1	1	4	24.52	2	2	1	8	5.61
2	1	1	1	33.53	16 processors									
1	1	1	2	35.20	2	1	1	8	6.29					
1	1	2	1	49.48	1	2	1	8	10.45					
1	2	1	1	79.88	1	4	1	4	92.50					

Optimal settings of the pARMS preconditioner

The performance of the pARMS preconditioner is specially sensitive to the problem settings, the domain distribution and the number of processes, making it very difficult to find an optimal configuration. For the test case I, we found the best performance when using ARMS as local preconditioner, up to 16 levels of recursion, with a drop tolerance of 10^{-7} and a maximum fill-in of 90%. The solution obtained by the Schur complement recursive factorization of pARMS is enriched with up to 5 iterations of FGMRES.

The resulting preconditioner is slightly more expensive than ASM+ILU, as Figure 5.2 (left) shows, but pARMS converges with less preconditioner applications (3313, against 5229 ASM+ILU applications). However, in this case JD with ASM+ILU is faster. Notice that the use of preconditioners shifts the computational effort from GENE (the matrix-vector product, MV in Figure 5.2) to the preconditioner application operation.

On the other hand, the overhead of pARMS does not seem to penalize its parallel performance, as the comparison of speedups shows in Figure 5.2 (right).

5.3 Parameter scans

5.3.1 Subspace recycling

In an m-dimensional parameter scan, all eigenvalue problems are identified by a vector $\vec{p}$ in the m-dimensional subspace of the physical parameters varied, while the remaining (physical and numerical) parameters $\vec{p}_0$ are the same for all eigenvalue problems in the scan. The structure of the linear operator and most of the parameter values stay the same throughout the scan, and this should be reflected by a similarity of the eigenvectors. Since the initialization of the test vectors has a big influence on the speed of convergence of iterative solvers, the reuse of already computed eigenvectors as initial condition for a 'nearby' parameter set, so-called subspace recycling, has therefore the potential to speed up parameter scans signifi-

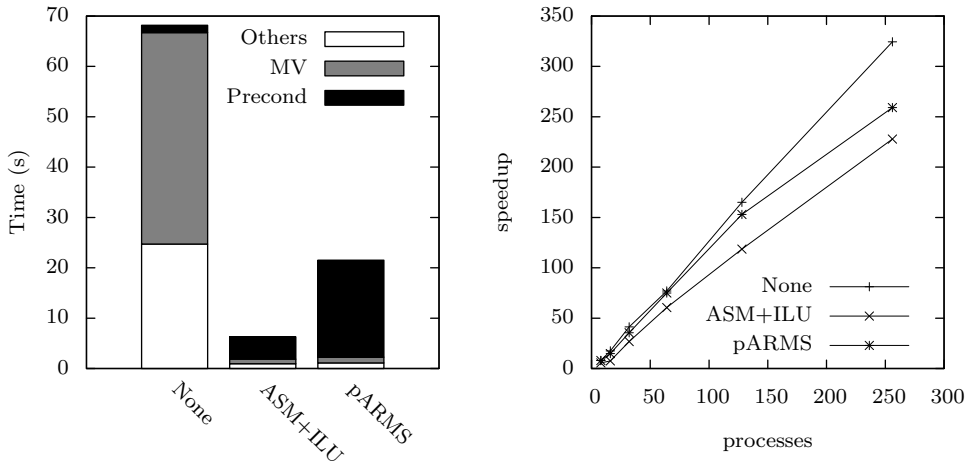


Figure 5.2: Time spent with 16 processes (left) and speedup (right) of JD solving the test case I without preconditioner (None), with ASM preconditioner using $\delta = 2$ and the local preconditioner ILU (ASM+ILU), and with pARMS using the local preconditioner ARMS (pARMS). MV stands for matrix-vector products.

cantly. To illustrate this, we have computed a one-dimensional parameter scan over the ion temperature gradient R/L_{Ti} from 2.5 to 5.5 around the nominal parameter set. Then the eigenvalue problem corresponding to the central point of the scan ($R/L_{Ti} = 4.0$) has been repeatedly solved, using the eigenvectors from the first scan at the different R/L_{Ti} positions as initial condition. The computation time relative to the computation time with random initialization is show in Fig. 5.3 as a function of $\Delta R/L_{Ti} = R/L_{Ti}^{in} - 4.0$. As expected, the computation time drops to almost zero for $\Delta R/L_{Ti} = 0$, with only the time for initialization remaining. The computation time increases quickly for $|\Delta R/L_{Ti}| > 0$, but the speedup compared to the computation time with random initial condition is significant throughout the parameter interval.

This illustrates two points. First of all, if eigenvectors $e_{i,a}$ $(a = 1, \ldots, n_{ev})$ for the parameter sets $\vec{p}_i$ $(i = 1, \ldots, n$, where n is the total number of previously computed solutions) are available, subspace recycling can reduce the computation time for a new parameter point $\vec{p}_{n+1}$ dramatically. And secondly, since the effect decays rapidly, an optimal selection of i is crucial.

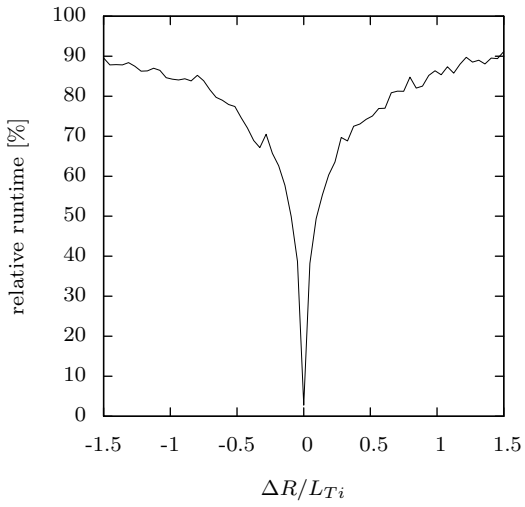


Figure 5.3: Computation time for the eigenvalue problem with $R/L_{Ti} = 4.0$ as a function of the difference to the R/L_{Ti} value of the eigenvectors used as initial condition, normalized to the computation time with random initial vectors.

5.3.2 Distances in parameter space

To speed up the computation for $\vec{p}_{n+1}$, the $\vec{p}_i$ 'closest' to $\vec{p}_{n+1}$ has to be found. For a one-dimensional scan, the difference vector in parameter space, $\vec{\Delta}_i = \vec{p}_{n+1} - \vec{p}_i$, has only one entry, which can naturally be used as a measure for the distance (as in Fig. 5.3). For an m-dimensional scan however, $\vec{\Delta}_i$ is m-dimensional, so that a metric has to be defined in parameter space. Then, the available e_i can be ranked according to their $|\vec{\Delta}_i|$ and the closest one can be selected.

For multi-dimensional scans, the scan ranges for the different parameter directions can differ by orders of magnitude, as can the effects of the variation on the solution, so that a simple Euclidean norm $|\vec{\Delta}_i| = \sqrt{\vec{\Delta}_i \cdot \vec{\Delta}_i}$ does not make sense. It is reasonable to assume that the speedup of the computation of the ath eigenvector of $\vec{p}_{n+1}$ due to initial vector $\vec{e}_{i,b}$ is related to the correlation coefficient between $e_{i,a}$ and $e_{n+1,b}$,

$$C(e_{i,a}, e_{n+1,b}) = \frac{|\int d\lambda\ e^*_{i,a} e_{n+1,b}|}{\sqrt{\int d\lambda\ e^*_{i,a} e_{i,a}} \sqrt{\int d\lambda\ e^*_{n+1,b} e_{n+1,b}}},$$

where $\int d\lambda$ denotes integration over the whole phase space, i.e., over all coordinates including the species. In our test problem, two eigenvalues are computed for each parameter set $(a, b = 1, 2)$, which results in four combinations for $C(e_{i,a}, e_{n+1,b})$. For $\vec{\Delta}_i \to \vec{0}$, two of the correlation coefficients approach unity, while the other

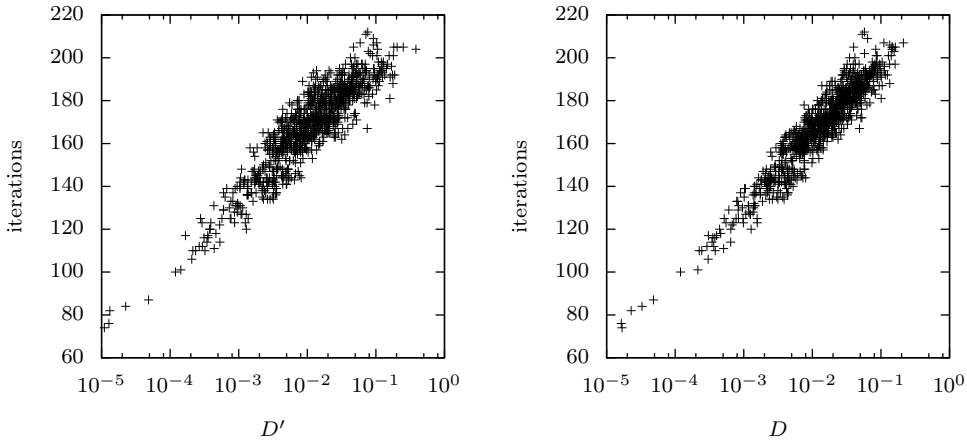


Figure 5.4: Number of iterations for the eigenvalue problem with the nominal parameters as function of $D' = 1.0 - C(e_i, e_{n+1})$ (left) and D (right).

two values approach the (smaller) correlation coefficient between the eigenvectors at $\vec{p}_{n+1}$. For the speedup, only the two combinations with the largest correlation coefficients are of interest, they are averaged to $C(e_i, e_{n+1})$, giving one real scalar quantity for each parameter combination.

To check the relevance of $C(e_i, e_{n+1})$, a set of random sample points $\vec{p}_i$ has been created, with a Gaussian distribution in R/L_{Ti} and R/L_{Te} ($\sigma = 0.6$) around the nominal parameter set. In a second stage, these e_i have then been used as initial condition for the computation of the problem with the nominal parameter set. Figure 5.4 (left) shows the number of iterations as a function of $D' = 1.0 - C(e_i, e_{n+1})$. The number of iterations is proportional to $\log(D')$, approaching the 217 iterations needed for $D' = 1.0$ (random initial condition). Finding the optimal e_i is thus equivalent to finding the smallest D'. The true $1 - C(e_i, e_{n+1})$ can of course only be determined after e_{n+1} has been computed, but it can be modeled to a good precision by $D(p_i, p_{n+1}) = \vec{\Delta}_i^T \cdot M \cdot \vec{\Delta}_i \approx D'$. For simplicity, the metric tensor M is assumed to be constant in parameter space. The entries of M can be determined by a fit (we use least squares fitting) to data, once the number of data points exceeds $m(m+1)/2$, which is the number of unknowns of M in m dimensions. For scan intervals that are not too big, we found that M converges quickly with the number of data points (here, we use $3^3 = 9$ equidistant points, corresponding to the first refinement stage for the hierarchical scans described in the next section). The data of Fig. 5.4 (left) plotted against $D(p_i, p_{n+1})$ is shown in Fig. 5.4 (right).

5.3.3 Parallelization

Going from a single eigenvalue computation to a parameter scan introduces new possibilities for parallelization. Without subspace recycling, the computations for the different parameter sets are completely independent and trivial to parallelize. This means that the individual eigenvalue computations can be run at their most efficient parallelization (which is determined by a balance of cache effects, communication overhead, and efficiency of the parallel preconditioner) and the whole scan can still employ a high number of processors to complete in a reasonable time.

To exploit this, the GENE solver has been extended to be able to deal with (independent) sets of input parameter files. In the initialization, the global MPI communicator is split into n_parallel_sims new communicators. On each of these subcommunicators, one (parallel) eigenvalue computation is run at a time. When the computation has finished, a new parameter set is selected from the (common) set of input files. The different instances keep track of the status of computation for each of the input files via MPI communication, so that each problem is only solved once; this is repeated until all parameter sets have been computed and GENE exits. In the present implementation of the solver, the file containing the initial vectors has to be specified in the input files, so they have to be known before the code is started.

As has become obvious in the previous subsections, subspace recycling is essential for the speed of parameter scans, the question is therefore how we can combine the benefits of subspace recycling (which introduces dependencies of the parameter sets) and this additional parallelism.

A good solution are hierarchical parameter scans, where the eigenvectors from previous refinement stages can easily be used as initial vectors, so that subspace recycling and parallelization over parameter sets can efficiently be combined. The necessity for a hierarchical sequence of parameter scans occurs naturally for adaptive grid refinement techniques, but even for dense grid scans without adaptivity, starting with a low resolution in the scan volume and hierarchically refining by bisection has the benefit of providing an interpolation for the full scan volume while the scan is still running.

The scans are managed by a superordinated Python script that is part of the GENE package since release 1.5. Controlled by a master input file, the script manages the creation of the parameter sets for a refinement stage. Taking into account all available eigenvectors from the previous stages, it computes the optimal e_i for each $\vec{p}_{n+1}$ of this new stage. It then starts the actual GENE code, which treats all parameter points of this refinement stage as independent and can therefore efficiently parallelize over the parameter points. The script then collects the results, and manages the storage of the eigenvectors and other output files, and continues with the next refinement stage.

Table 5.3: Wall clock times to compute the test parameter scan on 64 processors.

Solver	Parallelization	Subspace Recycling	Time [s]
Krylov-Schur	64/1	no	2375
Jacobi-Davidson	64/1	no	342
Jacobi-Davidson	8/8	no	264
Jacobi-Davidson	4/16	no	306
Jacobi-Davidson	8/8	yes	202

5.4 Application

We now want to demonstrate the gains due to the various improvements presented in the previous sections. As a test case, we perform a three-dimensional scan around the nominal parameter set presented in Table 5.1, varying the ion temperature gradient between 3.0 and 4.0, the electron temperature gradient between 3.5 and 4.5, and the magnetic safety factor q_0 between 1.2 and 1.4. We compute $5^3 = 125$ equidistant points in this parameter volume and use 64 processors for all cases. The results are shown in Table 5.3.

The solvers that have been compared are SLEPc's Krylov-Schur solver with harmonic projection, which needs no preconditioning and the Jacobi-Davidson solver with ASM+ILU preconditioning as described in Section 5.2. The parallelization column shows the number of processors per computation / number of parallel computations. As can be seen, the most important gain (a speedup of a factor 7) is due to the new solver/preconditioner. Both the optimal parallelization (8 cores per eigenvalue computation in this case) and the subspace recycling lead to further reductions of around 25% each. All in all, the computation time for eigenvalue scans with GENE/SLEPc has been reduced by more than an order of magnitude compared to previous versions.

5.5 Conclusions

In this chapter, we have presented and analyzed advanced numerical methods to perform large parameter scans with the GENE/SLEPc linear gyrokinetic eigenvalue solver. Considerable progress has been made concerning the robustness and speed of each single eigenvalue computation using the Jacobi-Davidson eigenvalue solver available from SLEPc 3.1 onwards, in combination with a preconditioner based on an approximate explicit representation of the linear gyrokinetic operator. In addition, two methods to speed up parameter scans have been used, namely the recycling of previously computed eigenvectors as initial condition for the computation at a nearby parameter set, and parallelization over the parameter sets, which removes the need to go to high processor numbers for the single parameter computations and therefore increases the efficiency. The performance gains for

multi-dimensional parameter scans using a three-dimensional test case compared to previous code versions were substantial, reaching a speedup factor of up to 12.

The overall implication of these improvements is that detailed investigations of the stable and unstable eigenmodes in the multi-dimensional gyrokinetic parameter space are now computationally feasible. The application of the techniques described in this chapter will certainly contribute to a better understanding of the important driving mechanisms of turbulent transport in fusion plasmas.

Chapter 6

Parallel DFT with Grid Refinement and Subspace Recycling

In this chapter adaptive finite element analysis of density functional computations of light atoms are explored based on two disparate themes: (i) h–adaptive grid refinement techniques that are applied within self-consistent iterations with the aim of reducing the total number of degrees of freedom in the real-space grid while improving on the approximate resolution of the system at hand; and (ii) subspace recycling of the approximate solution in subsequent self-consistent cycles with the aim of improving the performance of the generalized eigenproblem solver. Both techniques are shown to give a significant speed-up in the computation process. The computational toolkit consists of a set of freely available open source software libraries chosen for their high performance. These are integrated into a small and portable application code that solves the self-consistent algorithm for atomic systems.

6.1 Introduction

Density Functional Theory (DFT) [Hohenberg and Kohn, 1964; Kohn and Sham, 1965; Tsuchida and Tsukada, 1998] is one of the most celebrated first-principles framework used in theoretical and computation Physics for determination of the ground-state properties of multi-reference systems, and has far-reaching applicability in nuclear, atomic, and solid-state physics. DFT-based computer simulations are based on self-consistent iterations between a pair of coupled equations: (i) The Schrödinger equation that describes the wave-like properties of quantum objects; and (ii) The Poisson equation that describes the charge density as a functional of the relative charges of quantum objects (essentially a classical equation). The numerical implementation of DFT is to seek a configuration for which the charge density, $n^i(r) \sim n^{i-1}(r)$, where $n(r)$ is a function of real-space $r \in \mathbb{R}$ evaluated between two consecutive iterations $i-1$ and i. It is then said that a self-consistent

field has been found. Initializing the self-consistent procedure is largely a choice at the whim of the investigator, the principal rule being only that a "better" starting guess of the self-consistent field yields less self-consistent iterations and faster convergence to the wanted solution; that is, where the "best" starting guess would be the fully converged solution itself.

In the finite element basis the equation set to solve becomes an iterative process between the generalized eigenvalue problem (Schrödinger's equation) and a system of scalar-valued linear equations (Poisson's equation); see for example the introductory text by Ram-Mohan [2002] and the comparative review of Beck [2000]. The cost of solving a DFT-based problem in the finite element basis is directly related to the choice of finite element basis functions, *i.e.*, polynomial type and order, the choice of real-space representation, *i.e.*, grid type, and the number of eigenvalue problems that need to be solved in the iterative process. It is therefore worth exemplifying the advantages of that scheme:

1. In the finite element basis, being a strictly local theory, matrices arise that are sparse and for which numerical solvers optimized to take into account sparsity exist;

2. The choice of interpolation polynomials used in the finite element basis can guarantee continuity C^n of the global solution vectors of interest (up to arbitrary order n) which can be placed in accordance with the principles of quantum mechanics and classical mechanics;

3. Within the finite element scheme it is relatively straightforward (though not trivial) to employ p-adaptivity, which is the usage of a hierarchy of polynomial interpolations, and/or h-adaptivity, which is the usage of locally refined areas of the grid; and

4. In general, global basis functions fail to capture the subtleties of, and thus fail to describe, complicated functions over all $\mathbb{R}$-space. In that case a system based on local basis functions (such as the finite element method) is preferred.

To date, much of the work in adaptive real–space finite element quantum mechanics relies on improving the finite element approximation by examination of the underlying polynomial interpolation between grid points on which the computation is performed. These include the works of Pask and Sterne [2005] who investigate the rôle of the choice of well-known basis states in the finite element interpolation scheme. There is also a study of the Hydrogen atom by Guimarães and Prudente [2005], and the discrete variable representation of Rayson [2007] and Schneider et al. [2006]. The problem of using mixed basis functions was solved by Yamakawa and Hyodo [2005] who addressed the problem of treating core electrons in molecular orbits. The use of mixed basis functions as a local error estimate by Ackermann and Roitzsch [1993]; Ackermann et al. [1994] then leads to the so-called "multilevel method" and the "hierarchical method" investigated by Sugawara [1998].

An alternative to improving on the polynomial interpolation of the finite element approximation is to focus on the grid on which computations are performed. An example of local refinement applied within density functional theory applied to Beryllium clusters was investigated by Fattebert et al. [2007]. The works of Tsuchida and Tsukada [1995, 1996] use non-regular meshes based on geometric (logarithmic) considerations to locally control the resolution of calculations of electronic structure. The underlying proposition is that the space grid is small near to the locality of the nucleus, where the potential is varying rapidly. In this work we introduce a method of grid adaptation that is based on geometric observations of the underlying atomic potential that appears in the Schrödinger equation and is motivated by a need to create an initial starting grid on which a solution of the Hartree problem can be found with reasonable accuracy. That is then passed over to more traditional methods of grid adaptivity during the iterative determination of the self-consistent field (cf. Kelly error estimate [Kelly et al., 1983]).

The solver that addresses the computation of some eigenpairs of the Schrödinger equation, should converge fast, dealing with degeneracies and eigenvalue clustering. This makes the problem challenging. Preconditioned conjugate gradient methods have been usually proposed to handle this task, however Davidson methods have recently demonstrated being faster and more robust in many cases, *e.g.*, [Vömel et al., 2008].

In any case an efficient and sufficiently general solution of DFT is highly desirable. In this chapter we discuss how h-adaptive techniques based on the Honenberg–Kohn (HK) theorems as well as solver techniques (in particular subspace feeds) based on the fundamentals of non-linear self-consistent procedure, can be exploited to achieve that goal. Rather than to modify the mathematical foundations of DFT, we exploit those foundations (in particular the HK theorems); a method of investigation similar to that of Napoli et al. [2011]. It is thus, that our methods do not require the introduction of specializations and remain widely applicable to all systems that pertain to treatment by DFT.

This chapter is thus divided into five further sections. In the next section a brief overview of the DFT of finite electronic systems is given. In §6.3 some of the theoretical and computational requirements for the management of adaptive grids are considered, and estimates of the local error on the grid discussed. The resultant generalized eigenvalue problem and the solver methods used are given in some depth in §6.4, in which the notion of subspace feeds for solver initialization is explored. Following that, numerical experimentation of the methods of §6.3 and §6.4 is undertaken in §6.5 for a series of light atoms. Finally a brief discussion and our conclusions arising from this work are laid out in §6.6.

6.2 Theoretical background

In recent years a reasonable effort has been concentrated on the Finite Element Analysis (FEA) of the DFT of electronic structure in atomic/molecular physics and in condensed matter physics. As sketched above, there are some very sound reasons to put a concentrated effort toward this a goal: Ease and speed of computations without compromising accuracy and the physical basis on which computations are performed is highly desirable. In this section an overview of DFT for a system of an isolated atom is given that is based on the two HK theorems and in the finite element basis. For completeness the two HK theorems are stated here as prescribed in Ref. [Parr and Yang, 1989] with only some changes in notation. The first concerns the form of the potential:

Hohenberg–Kohn theorem 1. *The external potential* $V(\mathrm{x})$ *is determined, within a trivial additive constant, by the electron density* $\rho(\mathrm{x})$*.*

and the second which concerns the variational procedure:

Hohenberg–Kohn theorem 2. *For a trial density* $\rho(\mathrm{x})$*, such that* $\rho(\mathrm{x}) \geq 0$ *and* $\int \rho(\mathrm{x})\,\mathrm{d}r = N$*,*

$$E_0 \leq E_V[\rho] \quad ,$$

where $E_V[\rho]$ *is the energy functional of (6.1).*

6.2.1 Physical–theoretical framework

Schrödinger's wave equation in quantum mechanics describes the quantum state of a particle in a body by the wave function $\Psi = \{\psi_i\}_{i=0}^{\infty}$ in an external potential V as $(\Delta + V)\psi = E\Psi$, where E is the energy state of the system. The form of potential that governs the state vectors for atoms is based on the Coulomb potential which is exact for the Hydrogen atom. In dimensionless form, Schrödinger's time-dependent equation is

$$-\frac{1}{2}\Delta\Psi + \left(V(n(r)) + V_{\mathrm{EXC}}(n(r))\right)\Psi = E\Psi \quad , \quad \text{in } \Omega \quad , \tag{6.1}$$

where $\Delta := \nabla\cdot\nabla$ and where $V(n(r))$ is a density composed of single-particle states and $V_{\mathrm{EXC}}(n(r))$ is the exchange term whose functional properties are normally taken from computational Monte Carlo experiments. In the non-interacting case $V_{\mathrm{EXC}} = 0$ Eqn. (6.1) can be solved by substituting (as a first approximation) the second term for the Coulomb potential (in dimensionless form) $V = Z/|r - r'|$ where Z is the atomic number (number of protons) and $r - r' = \sqrt{\sum_d (x_d^2 - x'^2_d)}$ gives the radial distance from the atomic nucleus at $r = 0$ to any arbitrary point of the field r'. That yields a finite set of N orbitals $\Psi = \{\psi_i\}_{i=1}^{N}$ that reproduce the density $n(r)$ of the original many-body system

$$n(r) = \sum_i |\psi_i(r)|^2 \quad . \tag{6.2}$$

The main difficulty in solving Eqn. (6.1) is that the electron density ρ is dependent on the solution vector Ψ. This is usually obtained by solving Poisson's equation and adding the resultant density to the non-interacting potential, such that $V \to V'$, *cf.* theorem 1. Following that, the process repeats by solving Eqn. (6.1) again, yielding new wave functions Ψ and a new density $n(r)$, until convergence is achieved, *cf.* theorem 2. It is worth noting here that this method of solution for the Kohn-Sham (KS) equation set for determining the Self-Consistent Field (SCF) is not the most efficient. It is however an algorithm that preserves the Coulomb potential as the initial guess from which the derived eigenstates of the system are determined. More specifically, one could for example use an explicit start scheme for the Poisson equation by first guessing the eigenstates. This by-and-large makes use of carefully chosen set of orthogonal basis states from which the initial density functional ρ is formed. That scheme, and many like it, require prior knowledge of the particular system at hand and therefore cannot be generically applied in a computation.

6.2.2 Finite element approach

The physical equations are posed on a finite Lipschitz domain $\Omega \cup \mathbb{R}^d$ with dimensionality $d = 3$ where any point in the domain Ω is decomposed into three Cartesian components $\mathrm{x}_d \in \mathrm{x} = \{x, y, z\}$. Auxiliary conditions are applied on the boundary of the finite domain $\Gamma \cup \mathbb{R}^{d-1}$. In steady state or stationary state, considered here, dependency over time vanishes and the partial differential equation sets with boundary constraints together constitute a class of boundary value problems. Casting the equations into a form suitable for subsequent finite element analysis can be summarized in brief as follows: (i) Consider the "test" function φ which is supposed to solve the system of equations exactly; (ii) Multiply the equation set on the *left* by the scalar–valued test function φ; (iii) Integrate over the (finite) domain Ω, and integrate by parts using Green's first identity. Using these scalar test functions the discretized finite element solution can be written $u_h = \sum_i \varphi_i U_i$ where i denotes degrees of freedom in the finite element h. An analogous function v_h is then defined for the test function.

Following that procedure to find the finite element solution of Schrödinger's equation gives:

$$\int_\Omega \left(\frac{1}{2} \nabla\varphi \nabla\Psi + V(r)\varphi\Psi \right) \mathrm{d}\Omega - \oint_{\partial\Omega} \frac{1}{2} \varphi \nabla\Psi \cdot \hat{n} \, \mathrm{d}\Gamma = \int_\Omega \varphi E \Psi \quad , \tag{6.3}$$

where $\hat{n}$ denotes the outward normal on the surface Γ. After discretization this becomes:

$$\sum_{ij} U_i V_j \left(\frac{1}{2} \nabla\varphi_i \nabla\varphi_j + V(r)\varphi_i\varphi_j \right) = \sum_{ij} \varphi_i \varphi_j E \quad . \tag{6.4}$$

The surface integral in Eqn. (6.3) above is associated with boundary constraints on the finite volume. In the parlance of quantum theory there exists a posit that

$$\int_{\Omega} |a_k \psi_k|^2 \, d\Omega = 1 \quad , \tag{6.5}$$

corresponds to a probability density function $\forall\, k$ with arbitrary normalization constants a_k. The boundary integral is replaced by stronger condition in the form of Dirichlet zero boundary constraints: $\psi_i = 0 \,\mathrm{on}\, \Gamma$, which means that $\psi_i = \nabla\psi_i = 0$ for all states i. The boundary integral term has therefore been eliminated in Eqn. (6.4). The usage of zero boundary means that the solution vectors are unique and, moreover, scaling transformations can be used to determine the probability density as the square of the wavefunction $\rho_i = |\psi_i \psi_i|^2$.

There is no known analytical solution to the KS equation set and an analysis of the system energy can only be compared with experimental values or other theoretical means of evaluation. Ideally one would prefer to have a solution to the equation set $A(x)x = B(x)\lambda x$; however the solution to that is allusive. Thus the system of equations is solved self-consistently, *i.e.*, by repeated iterations against Poisson's equation. When convergence is achieved, we say that we have a Self-Consistent Field (SCF). The Poisson equation can be cast in the finite element basis by following the analogous procedure applied to the Schrödinger equation above. We find

$$\int_{\Omega} \nabla\varphi \nabla\phi \, \mathrm{d}\Omega - \oint_{\partial\Omega} \varphi \nabla\phi \cdot \hat{n} \, \mathrm{d}\Gamma = \int_{\Omega} \varphi n(r) \quad , \tag{6.6}$$

where $n(r)$ is the density of Eqn. (6.2), and after discretization:

$$\sum_{ij} U_i V_j \nabla\varphi_i \nabla\varphi_j = \sum_{j} \varphi_j n(r) \quad . \tag{6.7}$$

Since the electric potential at the boundary must match those of the Schrödinger equation, we have again over–specified the boundary constraints by setting $\phi = 0$ on Ω.

6.3 Management of computational grids

Let the ultimate practical goal of computer simulations in the physical sciences be to obtain the solution to a set of partial differential equations with maximal accuracy (Tol) and minimal computational effort (Work). Then the central idea of adaptive algorithms in a finite element approach is to improve on previous calculations and from there to find more accurate solutions in an automated way using some form of a local error estimate. One standard approach that can be employed is to adapt the grid on which calculations are performed by refining

finite elements in local regions of the domain; so called h-adaptivity. The scheme is given by the process:

$$\text{Estimate} \rightarrow \text{Flag} \rightarrow \text{Refine} \quad .$$

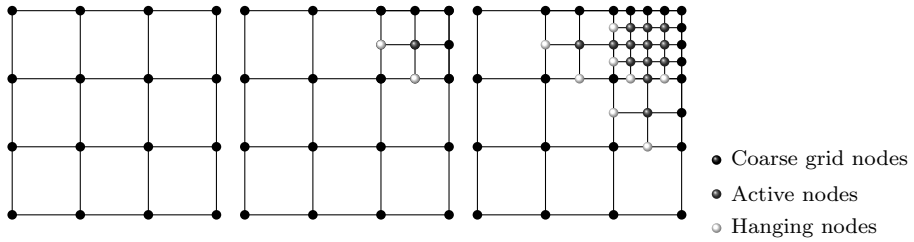
In this section we discuss the computational toolkits used for the finite element representation of the equation set and put forward two methods of grid adaptivity that can be used to obtain error estimates for the process of adaptivity.

6.3.1 Finite element toolkit

`deal.II`, an acronym for the Differential Equations Analysis Library [Bangerth et al., 2007] is written in templated C++, provides an abstract set of base tools for computations with adaptive finite elements. `deal.II` has support for using external solvers in the form of wrapper classes which enable standard computational toolkits to be used with relative ease. In particular, wrapper classes to PETSc and SLEPc (for explicit details on external libraries see §6.4). Additionally grid managment over parallel worlds is handled through an interface to the `p4est` library, which facilitates communications by dynamic management of a collection of "adaptive octrees"; see Ref. [Burstedde et al., 2011; Bangerth et al.].

At the time of writing, the `deal.II` tutorial suite contains one example code developed in course by one of the authors for solving the Schrödinger generalized eigenvalue problem (the serial step-36 tutorial program) which can be readily extended to more complex systems, grid adaptivity, and parallelization. Also developed specifically for this work is a (pre–alpha) library package that contains a variety of tools for computations in quantum physics "namespace ewalena" [Young, 2011a]. Also written in C++ the namespace ewalena library builds on `deal.II` objects and provides a collection of tools that are useful for describing systems of interest in mathematical and computational physics. There are, for example, basis states of arbitrary size, a growing set of predefined trial wavefunctions for DFTs, a set of atomic/molecular potentials (defined as a linear superposition of atomic potentials), and a zoo of physical constants based on the NIST database.

This combined scheme, appropriately extended and adapted, has been successfully used for problems pertaining to general quantum physics [Young and Armiento, 2010], atomic physics [Young et al., 2009], and solid–state physics [Young, 2011b]. In this chapter it is used to compute the energetic state of light atoms within a SCF theory. The algorithm of the governing user code makes use of dimension-independent programming



(a) Coarse grid. (b) Cycle one. (c) Cycle two.

Figure 6.1: Local refinement of cells: (a) Coarse grid on level zero; (b) Level one refinement of one cell; and (c) Level one refinement of two cells and level two refinement of one cell.

6.3.2 Grid refinement

The basic idea of using error estimates is quite straightforward. First an error estimate is obtained for each finite element cell based on a given criterion; and second refinement and/or coarsing of the grid is given according to the rules applied to the collection of error estimates.

Actually constructing the error estimate is nontrivial and often dependent on the equation set at hand. What one would like to do is to jump straight from Fig. 6.1(a) to an optimally refined grid of Fig. 6.1(c) without any need of solution to equation sets and thus further minimizing the usage of computational resources. This task could easily be accomplished in a whimsical manner, for which the production rules for a new grid depend on the investigator's "best guess". A more rigorous option would be to examine the functional properties of the underlying equation set to be solved and from that extract error estimates.

We now discuss two methods of grid refinement used in this chapter in order to successively improve on our solution. The first is the *a priori* error estimate of Kelly et al. [1983] and the second is a "Projected Potential" based error estimate designed by the authors.

Kelly error estimate

A commonly used method for *a priori* error is the Kelly error estimate and is conveniently provided in the `deal.II::KellyErrorEstimator` class. The Kelly error estimate is generally considered to be a universal tool for estimating the errors in sets of linear equations (*cf.* Poisson's equation). It is based on estimating errors to the Laplace equation, $\nabla^2\phi = 0$, of which Poisson's equation $\nabla^2\phi = f$, which has a nonzero right–hand–side vector f, is a special case. That, and other methods, have been extended to more complex systems [Ainsworth and Oden, 2000; Babuška and Strouboulis, 2001]. The basic idea of the Kelly error estimate is to compute the second derivative (Hessian) of the solution vector at each point on the grid and, by integration of the discontinuity between each cell, assign an approximate

error to that cell. The greater the discontinuity, the higher the error estimate; and it is worth noting that error estimates in that definition are in $\mathbb{N}^+$ only. Defining the boundary of the cell K as the union of its faces $\partial K^d \cup K_i^{d-1}, K_{i+1}^{d-1}$, this leads to an error estimate of the form:

$$\eta_K^2 = \frac{h}{24} \|\partial_i \phi_h\|_{\partial K}^2 \quad , \tag{6.8}$$

where $\partial_i u_h$ denotes the discontinuity of the normal derivative of the solution vector ϕ at the interface between cells.

Projected potential error estimate

An alternative approach to relying on known solutions is to attempt to employ an *a posteriori* error estimate; that is, one in which the expected error is known, or at least supposed by exploiting knowledge of the equation set. The crucial considerations that make this scheme are presented below, where a more detailed exposition is to appear elsewhere. Let us start with the following ideas: First,

Proposition 1. *(Physical) There is (or can be obtained) at least a posteriori knowledge of the atomic potential function V.*

and second,

Proposition 2. *(Numerical) Between self consistent cycles I given a suitable initial "good guess" for the functional $n(r)^I$ given $n(r)^{I-1}$ the change in the external potential $\delta V := V^I - V^{I-1}$ is vanishingly small.*

Provided proposition 1 is fulfilled, in general, it can be supposed that proposition 2 will be automatically fulfilled. Taking into account large spin-orbit splitting, or the introduction of large external perturbations in the potential (for example, an applied magnetic field), this may not always be the case. Nevertheless, there is little more to be said about proposition 2 other than that if the initial guess of the potential function is not well suited to the actual solution, then the initial guess should be improved by examining the initial equation set. If this improvement is ever found to be the needed, then proposition 2. As it turns out, proposition 1 is far more interesting since it relates the solutions of the eigenvalue problem to that of the external potential in a way that is mathematically amenable and in the spirit of the first HK theorem 1. Recall that the density ρ uniquely defines the electron wavefunctions Ψ (and vice–versa). Then, if proposition 2 holds (let $\delta\phi \to 0$), the potential will uniquely define the electron wavefunctions. It is then taken as a corollary that a good representation of the underlying potential will yield a good representation of the wavefunctions.

Consider the equation for a bulk criterion $\mathcal{C}$ defined in terms of the bulk error $\mathcal{E}$

$$\mathcal{C} := \max\left[\mathcal{E}\right] \propto f(V(x_d)) \to 0 \quad , \quad \text{on}\, x \in \Omega \quad , \tag{6.9}$$

where $f(\cdot)$ is an as–of–yet unspecified function of the real-space potential V and the resulting criterion for the cell K is $\mathcal{C}_K \subset \mathcal{C}$. The error is then calculated per finite element and the set of elements in the volume $\bar{\Omega} \subset \Omega$ for which $\max \left[\mathcal{E}^K\right]_{K\in\Omega} > 0$ are flagged for refinement. The size of the set $\bar{\Omega}$ is a free parameter, though is normally chosen to double the number of cells in the grid.

Before proceeding it is worth considering the following suggestive comments: (i) The wavefunction ψ_i drops off exponentially as it extends from a potential barrier (core). It is therefore reasonable to expand the potential at each point in space as an exponential function. (ii) The absolute value of the potential does not affect the wavefunctions, which are eigenfunctions. It is therefore always possible to scale the potential with a number (normalization) without affecting the result.

It seems natural to continue by expanding the function as a left (negative) exponential of the absolute value $f(V) \to \exp[-|V|]$; thus avoiding the divergent properties of $\exp[a]$ for all $a > 0$. The exponential function of the potential, which converges to zero at the boundaries, has the following useful properties: (i) The left-hand-side of the exponential function is convergent as $\exp[-1/0] \sim \exp[-\infty] \to 0 =: \psi_i|_{\partial\Omega}$; and (ii) The left-hand-side of the exponential function is always finite as $\exp[0] \sim \exp[1/-\infty] \to 1$. In other words, the expansion of the potential has the same numerical properties as the solution vectors toward the boundaries of the domain, and non-singular properties at infinities of the potential function – as do the solution vectors – where $\exp[-\infty] \to 0 =: 1 - \|\psi_i\|_{\ell_\infty(\Omega)}$. By choice, the potential function is normalized by the energy of the vacuum E_{vac}, which in the case considered here is zero; and finally remove the proportionality relation with the introduction of a constant. This leads to an expression as a function of the Coulomb potential from which the error estimate can be evaluated as

$$\mathcal{C} = 1 - \exp\left[-\eta \left| \frac{1}{|r - r'|} - E_{\text{vac}} \right| \right] \quad . \tag{6.10}$$

This procedure turns out to be self-normalizing because of the second property of the exponential function: $\exp[-\infty] = 0$ together with the addition of the vacuum term $f(V) \propto (V - E_{\text{vac}})$ which, as sketched above, gives a value of one to the projected potential at singular points. In order to extract an error estimate for each active cell in the grid, the expression above is first discretized and then its gradient is integrated over each cell face in 3D (line in 2D, vertex in 1D). That procedure is similar to the Kelly error estimate described above (Eqn. (6.8)).

The resultant projected potential function and its associated error estimate are given in Fig. 6.2. First we note that the initial grid (upper panel) gives a poor approximation for the potential and a maximal error estimate $\max \left[\mathcal{E}^{(0)}\right] \sim 0.25$. After applying the error estimate twice by two cycles of refining a fractional $N = 1/d^2$ and then recalculating the projected potential, the representation of the projected potential on an adapted grid is a better numerical approximation of the analytical form of Eqn. (6.10) (lower panel of Fig. 6.2). Furthermore, after these cycles of refinement, the error estimate has fallen – $\max \left[\mathcal{E}^{(2)}\right] \sim 0.06$ and

$\max\left[\mathcal{E}^{(4)}\right] \sim 0.006$ – a 76% and 98% reduction in the maximal error estimate respectively from the coarse starting grid.

6.3.3 Summary

The resultant grids obtained using the two error estimates *in two space dimensions* are given for comparison in Fig. 6.3 for the Kelly error estimate and in Fig. 6.4 for the projected potential scheme. Starting from a single affine cell globally refined three times, three results of three cycles of adaptivity are compared. Two examples of refinement are given for each, one (upper-panel) where the number of cells flagged for refinement $N = 1/d^2 = 0.25$ and the second where $N = 1/d = 0.5$ are refined (leading to a more aggressive refinement scheme).

The Kelly error estimate is found by solving the Hartree problem and then basing the error estimate on the sum over the lowest eigenvector states, or in other words, on the electron density function. The main refinement is found to expand out from the center of the grid where the singularity in the potential is present, or conversely, where the cusp in the wavefunction is known to be. The adaptivity based on the projected potential scheme, *cf.* Fig. 6.4, appears to give very similar results, that is, that the refinement levels increase as the singularity in the potential is approached. There is however one marked difference: namely, that at the location of the singularity itself, the level of refinement is lower than the surrounding cells. These leads to a "hole" in the refinement pattern, which is particularly noticeable in the less aggressive version of refinement (upper-panel). That this appears is no surprise, since the function of Eqn. (6.10) has a cusp as the potential becomes divergent – as does the electron density function. The projected potential evidently has a shallower gradient than the electron density at this point, and hence the gradient of the function between adjacent cells is minimal.

The Kelly error estimate has the advantage in that it works directly on the electron density to estimate errors in the solution but requires that a minimum of one eigenvalue problem and one set of linear equations be solved on each and every adaptive cycle. The method of projecting the potential onto a continuous space for estimating the errors does not require either of these computations on equation sets but simply a representation of the underlying potential. The difference in computational cost for the latter is therefore significantly lower than the Kelly error estimate, though otherwise, both yield similar grids.

It was noted above that the method of projecting potentials misses information about the core arising from the reciprocal of the radius-squared term that is the Coulomb potential. This is beneficial in that places where the computation may blow-up are avoided, however the loss of information is not highly desirable. Nevertheless, in §6.5, numerical experiments are applied to the Hartree problem using the method of projecting the potential and it is shown that convergence toward the exact (*i.e.*, analytical) result can easily be found within this scheme.

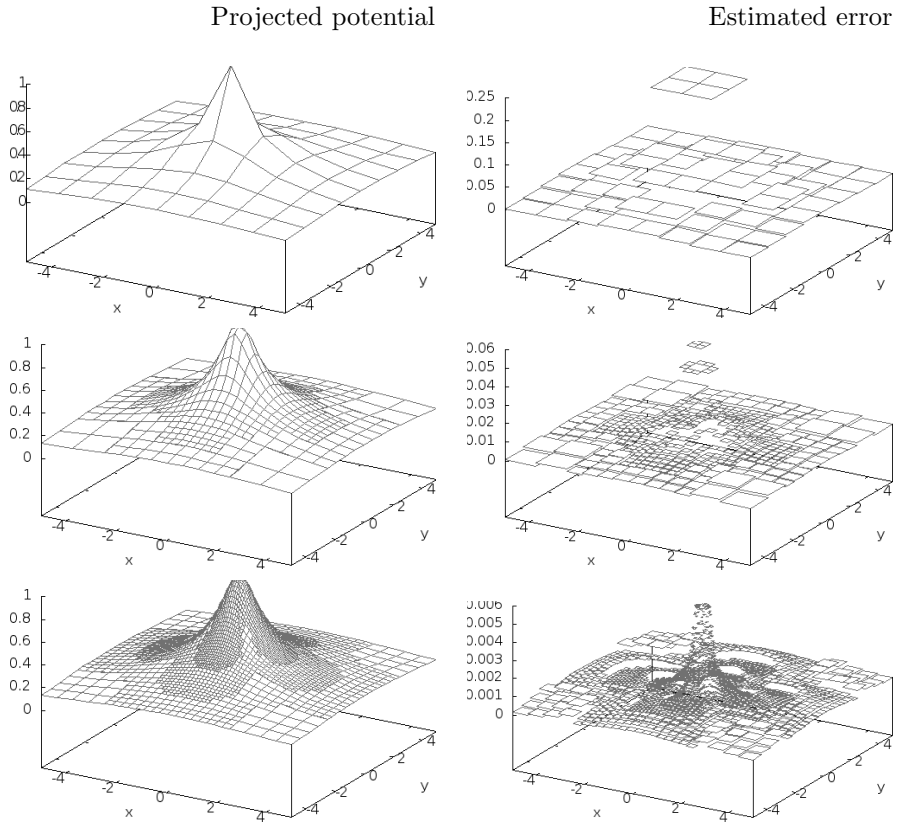


Figure 6.2: Projected potential plotted as an interpolated function and the resultant error estimate for the starting grid (upper panel) and after two levels of refinement (middle panel) and four levels of refinement (lower panel). The error estimate is plotted for each finite element cell, where the "height" of the finite element cell depicts the estimated error. Note the change of scale in the error estimate.

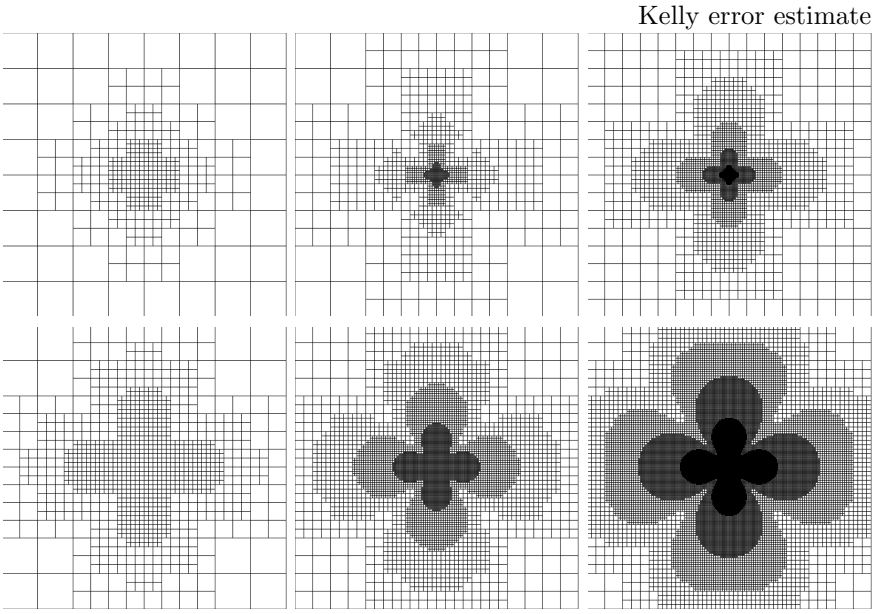


Figure 6.3: Two sets of successfully refined grids based on the Kelly applied to the density functional as an error indicator and using different levels of refinement. In (a) for fractional refinement $N = 1/d^2$ of the total number of active cells; and (b) The more aggressive condition $N = 1/d$.

6.4 Large-scale eigenvalue problems

To compute the eigenstates within the self-consistent loop, we need an efficient solver for the associated algebraic eigenproblem. Here, efficiency is understood in various ways: (i) The ability to exploit the sparsity of matrices; (ii) Expedient convergence to a small number of wanted eigenvalues; and (iii) Parallel capabilities to address large-scale problems. In this section, we discuss methods and software for this purpose.

In this chapter, we are primarily concerned with eigenpair solutions to the real generalized symmetric-definite eigenvalue problem,

$$Ax = \lambda Bx \quad , \tag{6.11}$$

where $A, B \in \mathbb{R}^{n\times n}$, $\lambda \in \mathbb{R}$ (eigenvalue) and $x \in \mathbb{R}^n$ (eigenvector). A is symmetric and B is symmetric positive-definite. In the methodology described in previous sections, B would naturally represent the overlap matrix, but this is not positive-definite. We will discuss later how to make the problem fit the above formulation so that symmetry can be exploited.

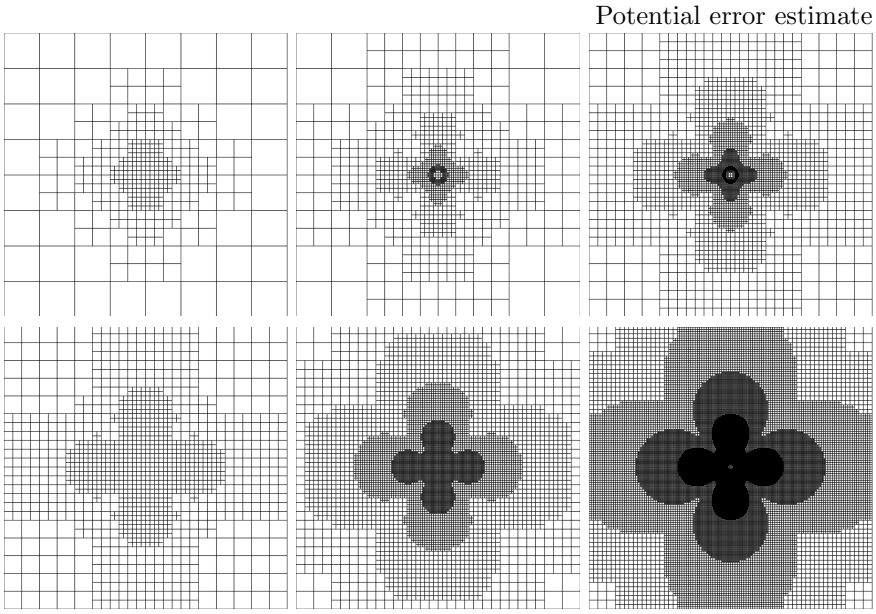


Figure 6.4: The same as in Fig. 6.3 but for two sets of successfully refined grids based on the projected potential as an error indicator. The results vary slightly.

We will focus on projection methods for the above eigenproblem, which are appropriate when the matrices A and B are very large but their action on a vector is relatively cheap (e.g., they are sparse) and only part of the spectrum is required. Projection methods rely on two main stages: building a subspace basis and extracting approximations from that subspace. These computations are carried out iteratively, where the extracted approximations are used to improve the subspace and extract better approximations. The process is repeated until the computed approximations are sufficiently accurate. For background material on projection methods, the reader is referred to [Parlett, 1980; Bai et al., 2000]. For completeness, we discuss several alternative approaches for subspace extraction and expansion below.

6.4.1 Subspace extraction

For the moment, we restrict the description to the standard eigenvalue problem $Mx = \lambda x$, where M can be non-symmetric in general (in which case λ and x may be complex). We want to compute k eigenpairs, (λ_i, x_i), $i = 1, \ldots, k$, usually with $k \ll n$. The basic principle of projection methods is to find the best approximations to the eigenvectors in a given subspace of small dimension. Let V be an $n \times m$ matrix, with $k \leq m \ll n$, whose columns v_i constitute an orthonormal basis of a given subspace $\mathcal{V}$, i.e., $V^T V = I_m$, where I is the identity matrix and $\text{span}\{v_1, v_2, \ldots, v_m\} = \mathcal{V}$. Then the eigenvalues of the so-called Rayleigh quotient matrix $H = V^T M V$ approximate eigenvalues of M. More precisely, if $Hy_i = \theta_i y_i$ then the eigenpair approximations are $\tilde{\lambda}_i = \theta_i$ and $\tilde{x}_i = V y_i$. These approximate eigenvectors belong to subspace $\mathcal{V}$ and are the best possible approximations in that subspace.

The method outlined above is referred to as the Rayleigh-Ritz procedure, and provides m Ritz approximations $(\tilde{\lambda}_i, \tilde{x}_i)$. This routine can be viewed as an orthogonal projection satisfying the Galerkin condition that the residuals are orthogonal to the subspace,

$$r_i = M\tilde{x}_i - \tilde{\lambda}_i \tilde{x}_i \perp \mathcal{V} \quad . \tag{6.12}$$

6.4.2 Subspace expansion

The quality of the eigenpair approximations $(\tilde{\lambda}_i, \tilde{x}_i)$ depends on how the subspace $\mathcal{V}$ is built. A popular choice is to use Krylov subspaces of increasing dimension, since they contain increasingly good approximations of eigenvectors of extreme eigenvalues. Computing an orthogonal basis of the Krylov subspace associated with matrix M and a given unit-length initial vector v_1, $\mathcal{K}_m(M, v_1) = \text{span}\{v_1, Mv_1, M^2 v_1, \ldots, M^{m-1} v_1\}$, can be done with the Lanczos or Arnoldi algorithms, which basically consist in orthogonalizing the vector Mv_k with respect to the previous basis vectors. In a practical implementation, it is not possible to expand the subspace up to a very large dimension, that is, to increase m too much, otherwise the storage and computational cost explodes. Thus it is necessary to *restart* the algorithm, that is, stop after m iterations and rerun the method by keeping as much relevant information as possible from the computed quantities. For a description of efficiently restarted Krylov methods, see [Ramos et al., 2010] and the references therein.

An alternative to Krylov methods are Davidson-type methods such as Generalized Davidson (GD) or Jacobi-Davidson (JD). The main characteristic of this class of methods is that they expand the subspace with a so-called correction vector t, which is computed from the residual vector r associated to the most wanted eigenpair with the aim of improving it further. This new vector can be computed by simply preconditioning the residual,

$$t = K^{-1} r \quad , \tag{6.13}$$

as in the GD method [Davidson, 1975; Morgan and Scott, 1986], or by (approximately) solving the correction equation,

$$(I - \tilde{x}\tilde{x}^*)\left(M - \tilde{\lambda}I\right)(I - \tilde{x}\tilde{x}^*)\,t = -r \quad , \tag{6.14}$$

as in the JD method [Sleijpen and van der Vorst, 2000]. As in (6.13), a preconditioner K can also be introduced in (6.14). K can be viewed as a rough approximation of M, the simplest version being $\mathrm{diag}(M)$ as originally proposed by Davidson [1975].

In many contexts, especially when computing interior eigenvalues in difficult non-symmetric problems, JD can be the most competitive method, and even more so if a good, cheap preconditioner is available. However, for symmetric problems where only extreme eigenvalues are to be sought, the high cost of the correction equation does not usually compensate. In this chapter, we focus on the simpler GD scheme.

6.4.3 Exploiting symmetry in definite matrix pairs

We now discuss how the above methods can be extended to the generalized symmetric-definite eigenproblem (6.11). In the case of Krylov methods, it is not possible to directly cope with generalized eigenproblems, so the trick is to set $M = B^{-1}A$, or if B is singular, use $M = A^{-1}B$ for the equivalent eigenproblem

$$Bx = \frac{1}{\lambda}Ax \quad . \tag{6.15}$$

In any case, the inverted matrix implies that linear solves must be performed in each iteration of the eigensolver, thus increasing the cost considerably. This approach was used in our preliminary work [Young and Armiento, 2010].

In contrast, Davidson methods can be extended naturally to generalized eigenproblems [Sleijpen et al., 1996]. When the projection is applied to the matrix pair (A, B) an m-dimensional matrix pair (H, T) is obtained, with $H = V^T AV$ and $T = V^T BV$. If the original matrix pair is symmetric-definite, so it is the projected one. In this context, symmetry can be exploited in two ways:

1. Modify the algorithm so that the computed basis V is not orthonormal but B-orthonormal, that is $V^T BV = I$. In that case, T becomes the identity and therefore the projected problem is standard. Note that this alternative requires B to be positive-definite, so if it is not but the matrix pair is definite, then one has to solve the reverse problem (6.15). The drawback of this alternative is that B-orthogonalization is generally more expensive than orthogonalization, since the standard inner product must be replaced by $\langle x, y\rangle_B = x^T By$.

2. Use regular orthogonalization and exploit symmetry only when computing the solution of the symmetric-definite projected eigenproblem (H, T). In this case, there is little gain in terms of computation, but the robustness may be improved significantly.

In §6.5 the latter option is used. In particular, the implementation is based on an unsymmetric version of GD (see Algorithm 2.3), where the symmetric entries of the matrices of the projected problem are computed only once. Further details about the implementation can be found in Ch. 2.

6.4.4 Subspace recycling

A drawback of Krylov methods is that they start building the subspace from a single vector v_1. If one has an a priori knowledge of a rough approximation of the solution, *e.g.*, from a previous iteration of the self-consistent loop, then this knowledge cannot be exploited. In contrast, Davidson methods can indeed benefit from using a rough approximation of the solution as initial guess. The justification is that Davidson methods can be viewed from the perspective of inexact Newton schemes [Wu et al., 1998]. Thus, a good starting solution can improve convergence considerably, with the corresponding reduction of the overall cost.

In our code, the eigenvalue computation at a given self-consistent iteration is started with an initial guess V^0 coming from the solution computed in the previous iteration. In order to further improve the convergence, this initial subspace can be enriched with a Krylov subspace generated by the operator $K^{-1}(A - \tau B)$ acting upon the initial guess (see §2.1.6).

6.4.5 The SLEPc library

SLEPc, the Scalable Library for Eigenvalue Problem Computations [Hernandez et al., 2005], is a software package for the solution of large-scale eigenvalue problems on parallel computers. It can be used to solve standard and generalized eigenvalue problems, as well as other types of related problems such as the quadratic eigenvalue problem or the singular value decomposition. SLEPc can work with either real or complex arithmetic, in single or double precision, and it is not restricted to symmetric (Hermitian) problems. It can be used from code written in C, C++, FORTRAN, and Python.

SLEPc provides a collection of eigensolvers, most of which are based on the subspace projection paradigm described in the previous paragraphs. In particular, it includes a parallel implementation of a robust and efficient restarted Krylov method, namely the Krylov-Schur method [Stewart, 2001a]. Several Davidson-type solvers are included as well, in particular GD and JD, with various possibilities for the computation of the correction vector. In these solvers, the user can easily select which preconditioner to use.

SLEPc is built on top of PETSc, a parallel framework for the numerical solution of partial differential equations, whose approach is to encapsulate mathematical algorithms using object-oriented programming techniques in order to be able to manage the complexity of efficient numerical message-passing codes. All the PETSc software is freely available and used around the world in many application areas. PETSc is object-oriented in the sense that all the code is built around a set of data structures and algorithmic objects. The application programmer works directly with these objects rather than concentrating on the underlying data struc-

tures. The three basic abstract data objects are index sets, vectors and matrices. Built on top of this foundation are various classes of solver objects, including linear, non-linear and time-stepping solvers. SLEPc inherits all the good properties of PETSc, including portability to a wide range of parallel platforms, scalability to a large number of processors, and run-time flexibility giving full control over the solution process (one can for instance specify the solver at run time, or change relevant parameters such as the tolerance or the size of the subspace basis).

6.5 Numerical experiments

If the target is to employ the methods of §6.3 (and summary therein) within a modern multi-electron theory, *i.e.*, density functional theory, one must take into account the troublesome Coulomb divergences that are characteristic of atomic potentials (for example, the review in [Tsuchida and Tsukada, 1998]). It is common in the application of those methods to handle the unscreened divergences through pseudopotentials, giving a resulting eigenproblem that avoids that issue. As noted in §6.3, the method of projecting the potential handles divergencies in the Coulomb potential and thus this problem does not arise in error estimation.

The most direct approach to solving the KS equation set is to discard the electron interaction terms, or in other words, to set the electron density functional to zero. The physical model then becomes the Hartree model and is one in which the SC cycle disappears. Since the Hartree approximation was found not to have a major impact on computational resources, the numerical experiments are performed on a serial architecture. That, incidentally, provides an excellent ground on which to test methods of grid adaptivity, as outlined in §6.3. The computations of the Hartree model were performed on Tramwaj, a single machine with two 64-bit AMD processors running at 2.4 GHz. The results correspond to `deal.II` 7.1 linked with versions 3.2 of PETSc and SLEPc.

While the results of Fig. 6.2 demonstrate how the error estimate can provide a better representation of the potential in which the form of the resultant grid was discussed, a second consideration is the number of degrees-of-freedom for that grid which is directly proportional to the size of the matrix that has to be presented to the solver. Tables 6.1 and 6.2 present statistical information about the resultant grids for the method that starts from projecting the potential (the shorthand O/M means Out of Memory).

Consider first the results of Table 6.1 in which the fraction of refinement is $N = 1/d^2 = 0.111$. In this case the most reasonably coarse grid is used (global refinement $g = 2$ which, in three dimensions, leads to sixty-four active cells in the grid). Successive refinement based on the projected potential as an error estimate clearly leads to convergence with respect to the previous level of refinement and, more fortuitously, the eigenenergy converges toward the correct (*i.e.*, analytical) result. In this example the energy appears to converge to an energy that is slightly lower than the analytical solution. This, we assume, may be a artifact of lacking refinement around the core; as discussed in §6.3. The same refinement strategy

Table 6.1: Evolution of the number of active cells (Act. cells), the total number of degrees of freedom (dofs), the total eigenenergy of the mean-field (eV) and the difference in the Hartree energy ($\Delta E_{\text{relative}} = |E^{n-1} - E^n|$), computing the electronic configuration of the Hydrogen atom. At each cycle of intermediate refinement defined by Eqn. (6.9), the number of cells flagged for refinement is $N = 1/d^2$ starting from a coarse grid globally refined $g = 2$ times.

Phase	Act. cells	Dofs	eV	$\Delta E_{\text{relative}}$
Global 2	64	125	-11.1149	
Intermediate-0	120	223	-12.2267	0.0409
Intermediate-1	400	649	-12.7780	0.0203
Intermediate-2	736	1157	-13.1234	0.0127
Intermediate-3	1688	2475	-13.2962	0.0064
Intermediate-4	3060	4179	-13.4066	0.0040
Intermediate-5	5454	7273	-13.4524	0.0017
Intermediate-6	9710	12263	-13.4968	0.0016
Intermediate-7	17606	21465	-13.5230	0.0010
Intermediate-8	31536	27483	-13.5389	0.0006
Intermediate-9	57296	65925	-13.5506	0.0004
Intermediate-10	102544	115043	-13.5597	0.0003
Intermediate-11	183184	201191	-13.5643	0.0002
Intermediate-12	326432	254273	-13.5677	0.0001
Analytical	∞	∞	-13.6057	0

is applied to grids that start from a higher level of refinement and the refinement level has been set to $N = 1/d = 0.333$. The results are gathered in Table 6.2.

The first clear result comes from a comparison of the starting grid $g = 2$ in both Table 6.1 and 6.2. A more aggressive refinement criterion ($N = 0.333$) leads to more degrees of freedom in the grid and a better energy convergence for a lower number of refinement cycles. This behavior is typical of error estimates. A closer examination shows that starting from a coarser grid in fact yields a *better* converged eigenenergy for a *lower* number of degrees of freedom; which is the second result. Starting from a well refined grid, *cf.* $g = \{4, 6\}$, a relative error in the energy is found to be $\Delta E_{g=4} \sim 0.0001$ and $\Delta E_{g=6} \sim 0.0003$, respectively, before running out of memory (O/M). The corresponding number of degrees of freedom in those cases are dof$= 1.6 \times 10^6$ and dof$= 3.0 \times 10^6$. The same comparison between of $g = 2$ and $g = 4$ show them to be quantitatively similar. Comparing this with the results of Table 6.1 we find the analogous result: $\Delta E_{g=2} \sim 0.0001$ and dof$= 0.25 \times 10^6$.

The numerical experiments testing the energy convergence and computational cost of computing the error estimated based on the projected potential are summarized in Fig. 6.5. Since using the error estimate based on the projected potential

Table 6.2: Same as in table 6.1, where at each cycle the number of cells refined is $N = 1/d$. Three computations are given starting from a coarse grid globally refined $g \in \{2, 4, 6\}$.

Phase	Act. cells	Dofs	eV	$\Delta E_{\text{relative}}$
Global 2	64	125	-11.1149	
Intermediate-0	288	469	-12.4016	0.0473
Intermediate-1	1072	1523	-13.0560	0.0240
Intermediate-2	3592	4721	-13.3736	0.0117
Intermediate-3	12370	15059	-13.5003	0.0047
Intermediate-4	41742	48367	-13.5433	0.0016
Intermediate-5	140050	155333	-13.5624	0.0007
Intermediate-6	468700	503735	-13.5662	0.0001
Intermediate-7	1565376	1641721	-13.5692	0.0001
Global 4	4096	4913	-13.0918	
Intermediate-0	13840	16217	-13.4077	0.0116
Intermediate-1	46656	52699	-13.5211	0.0042
Intermediate-2	156416	171327	-13.5555	0.0013
Intermediate-3	523160	557655	-13.5655	0.0004
Intermediate-4	1746424	1828109	-13.5695	0.0001
Intermediate-5	5829832	6016979	O/M	O/M
Global 6	262144	274625	-13.5237	
Intermediate-0	873832	911987	-13.5581	0.0013
Intermediate-1	2919568	3015425	-13.5674	0.0003
Intermediate-2	9742384	9977315	O/M	O/M
Intermediate-3	O/M	O/M	O/M	O/M

is of very low computational cost, it is satisfying that starting from the coarsest possible grid and a modest level of refinement between cycles yields a very good result. This, remembering the convergence behavior of the eigenenergy, indicates that refinement based on the projected potential is very suitable for producing a starting grid on which computations can be performed after which a solution-based error estimate (such as the Kelly error estimate) should be employed.

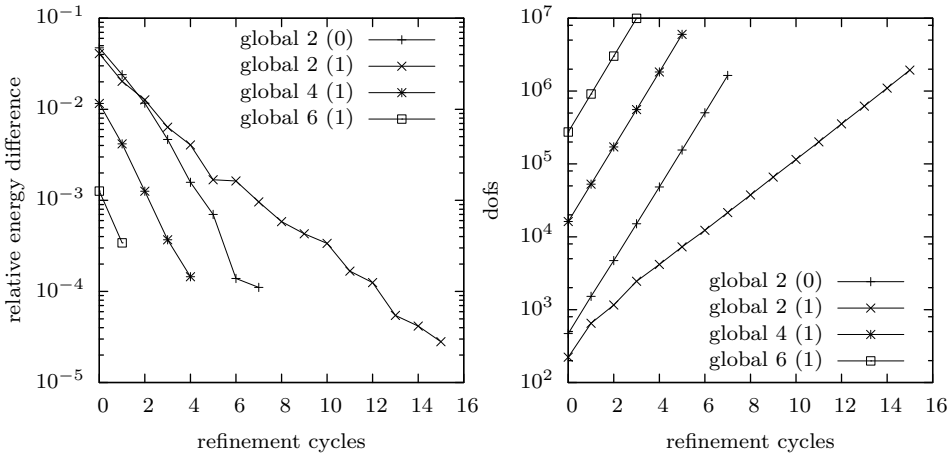


Figure 6.5: Evolution of the relative energy difference and total number of degrees of freedom as a function of refinement cycle using the projected potential as an error estimate. The key (0) refers to a refinement level of $N = 1/d^2$ cells, whereas (1) refers to a refinement level of $N = 1/d$ cells.

6.5.1 Performance of the combined scheme

The following discussion summarizes the experiments carried out in order to evaluate the performance of our implementation, and particularly in terms of scalability to a large number of processes. The experiments are executed on Tirant, a machine consisting of 256 JS20 blade computing nodes, each of them with two 64-bit PowerPC 970+ processors running at 2.2 GHz, and interconnected with a low latency Myrinet network. Only 256 processors are used due to user account limitations.

The results correspond to `deal.II` 7.0.0 linked with versions 3.2 of PETSc and SLEPc. The application employs the PETSc implementation of vectors, matrices and linear system solvers, and the SLEPc GD. The Poisson's Problem is solved iteratively with GMRES(30) accelerated with a Block Jacobi preconditioner. Although this iterative method does not exploit the problem symmetry and the preconditioner could be more powerful, the time spent by this part is considerably small; Fig. 6.6 shows this to be the case. On the contrary, the GD solver dominates the time, and therefore we focus on its settings for the rest of this section.

The GD solver is configured to compute as many eigenpairs as electrons there are in the atom, with a tolerance of 10^{-6}, using a symmetric variant of Algorithm 2.3. The search subspace is initialized with 10 vectors, and bounded to 18 vectors. When the subspace is full, the method restarts with 8 vectors. The preconditioner employed is the Block Jacobi approximation of $A - \tau B$, using incomplete Cholesky factorization on the blocks (level of fill equal to 5). The initial

Table 6.3: Evolution of the number of active cells (Act. cells), the total number of degrees of freedom (dofs), the iterations spent by the linear system solver (Poi.) and the eigensystem solver (Schr.), the total eigenenergy of the mean-field (eV) and the difference in the Hartree energy ($\Delta E_{\text{relative}} = |E^{(n-1)} - E^{(n)}|$), computing the electronic configuration of the Hydrogen atom performing two intermediate steps.

Phase	Act. cells	Dofs	Poi.	Schr.	eV	$\Delta E_{\text{relative}}$
Global	32768	35937		27	-13.4088	
Intermediate-0	58192	69749		22	-13.5190	$4.04\ 10^{-3}$
Intermediate-1	116012	148749	56	36	-13.5378	$6.91\ 10^{-4}$
Kelly-0 SC-0	207152	241837	53	54		$8.27\ 10^{-4}$
SC-1			53	1		$1.34\ 10^{-10}$
SC-2			53	1		$1.00\ 10^{-12}$
SC-3			53	1	-13.5603	$< 10^{-13}$
Kelly-1 SC-0	370175	415072	43	69		$1.57\ 10^{-4}$
SC-1			43	1		$3.20\ 10^{-11}$
SC-2			43	1		$6.00\ 10^{-12}$
SC-3			43	1	-13.5646	$1.00\ 10^{-13}$
Kelly-2 SC-0	660052	713196	76	85		$7.45\ 10^{-4}$
SC-1			76	1		$5.10\ 10^{-11}$
SC-2			76	1		$1.10\ 10^{-11}$
SC-3			76	1	-13.5666	$3.00\ 10^{-12}$
Kelly-3 SC-0	1175609	1246510	76	104		$5.26\ 10^{-5}$
SC-1			76	1		$5.80\ 10^{-11}$
SC-2			76	1		$9.00\ 10^{-12}$
SC-3			76	1	-13.5681	$3.00\ 10^{-12}$
Kelly-4 SC-0	2091727	2194572	40	129		$3.29\ 10^{-5}$
SC-1			40	1		$1.16\ 10^{-10}$
SC-2			40	1		$1.40\ 10^{-11}$
SC-3			40	1	-13.5689	$2.00\ 10^{-12}$
Kelly-5 SC-0	3744049	3926380	1	159		$1.59\ 10^{-5}$
SC-1			1	1		$2.34\ 10^{-10}$
SC-2			1	1		$3.30\ 10^{-11}$
SC-3			1	1	-13.5694	$9.00\ 10^{-12}$

target value is $\tau = -1$, and the subsequent ones are obtained from the previous converged eigenvalue.

Tables 6.3 and 6.4 show a trace of the size (degree of freedom) and the number of iterations spent by the linear solver and the eigensolver. Notice that the Hartree energy in a Kelly phase converges with few self-consistent iterations.

Figure 6.6 compares the time spent by the different parts and highlights the effectiveness of the feeding to reduce the time spent by GD (more than four times

Table 6.4: Same as in table 6.3, but performing four intermediate steps.

Phase	Act. cells	Dofs	Poi.	Schr.	eV	$\Delta E_{\text{relative}}$
Global	32768	35937		27	-13.4088	
Itermediate-0	58192	69749		22	-13.5190	$4.04\ 10^{-3}$
Itermediate-1	116012	148749		36	-13.5378	$6.91\ 10^{-4}$
Itermediate-2	232576	309753		45	-13.5457	$2.90\ 10^{-4}$
Itermediate-3	234816	311931	51	35	-13.5451	$2.91\ 10^{-5}$
Kelly-0 SC-0	424901	505787	56	69		$6.68\ 10^{-4}$
SC-1			56	1		$7.00\ 10^{-12}$
SC-2			56	1		$1.00\ 10^{-12}$
SC-3			56	1	-13.5633	$< 10^{-13}$
Kelly-1 SC-0	764254	850338	77	86		$1.31\ 10^{-4}$
SC-1			77	1		$4.70\ 10^{-11}$
SC-2			77	1		$7.00\ 10^{-12}$
SC-3			77	1	-13.5669	$2.00\ 10^{-12}$
Kelly-2 SC-0	1366681	1460988	71	108		$5.35\ 10^{-5}$
SC-1			71	1		$9.30\ 10^{-11}$
SC-2			71	1		$1.10\ 10^{-11}$
SC-3			71	1	-13.5683	$3.00\ 10^{-12}$
Kelly-3 SC-0	2432578	2561494	26	130		$3.97\ 10^{-5}$
SC-1			26	1		$1.55\ 10^{-10}$
SC-2			26	1		$1.70\ 10^{-11}$
SC-3			26	1	-13.5692	$3.00\ 10^{-12}$

Table 6.5: Collective timings for the Schrödinger–Poisson set up and solution on 16 processors. Ref. means refinement.

			Poisson			Schrödinger		
Phase	Time	Ref.	Assem.	Setup	Solve	Assem.	Setup	Solve
Table 6.3	621.7	84.49	57.29	28.86	64.77	53.29	22.91	288.71
Table 6.4	395.9	58.61	34.87	18.33	60.08	33.54	14.51	160.98

faster) computing the electronic configuration of Hydrogen. However recycling the eigenvalue as a target has a negative effect in the convergence of GD, in-so-much that the application does not converge if the non-symmetric variant of GD is used instead. The proposed scheme has been tested successfully to compute the electronic configuration of atoms up to 4 protons (see Fig. 6.6, right). For heavier atoms, difficulties arise in terms of convergence, so other preconditioning strategies should be employed (*e.g.*, multigrid).

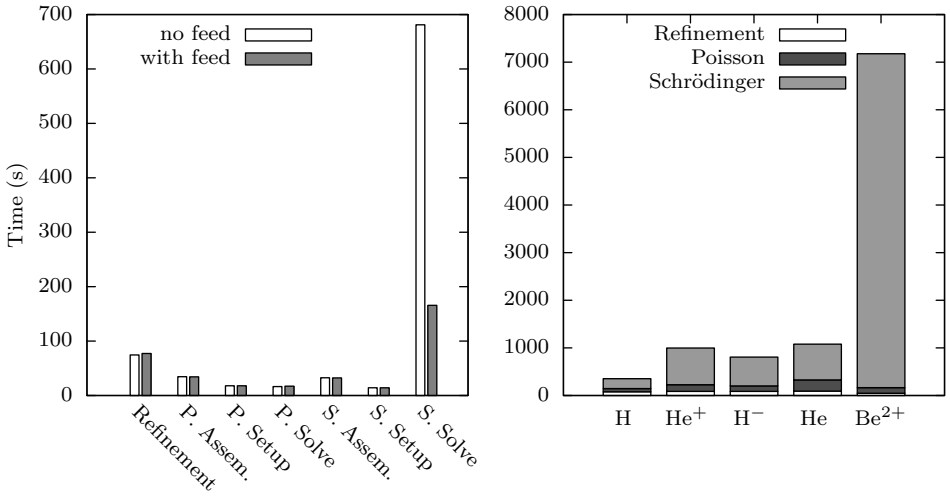


Figure 6.6: Wallclock times with 16 processes computing the electronic configuration of H without and with feeding (left), and the various atoms with feeding (right). The time is split between the refinement processes and the solution of the Poisson and the Schrödinger problems.

Finally, Fig. 6.7 shows the scalability of the different components in the application. The refinement process is the least scalable part, but takes a relatively small portion of the time for the whole computation. The other parts present a good speedup, leading the application as a whole to good parallel performance. It is worth emphasising here, that the error estimate based on a projection of the potential does not require any computations involving the solution vectors; at each intermediate refinement stage the solution has been calculated here only to demonstrate the convergent properties of the scheme.

6.6 Discussion and conclusions

In summary, the generic adaptive finite element scheme for electronic structure computations introduced above focuses on exploiting (i) h–adaptive grid methods; and (ii) Subspace recycling, offering experimental results regarding the proposed grid refinement techniques and performance of iterative solvers for the Poisson and Schrödinger problems. For numerical experimentation of grid adaptivity and the eigenvalue solvers employed here it is the convergence of the eigenenergies with respect to adaptive and SC cycles that are of central interest. We were also occupied in the relative convergence (between cycles, cf. the variational principle) and the scalability of convergence with respect to subspace recycling; and not in obtaining the best possible global convergence, *i.e.*, reproducing highly accurate

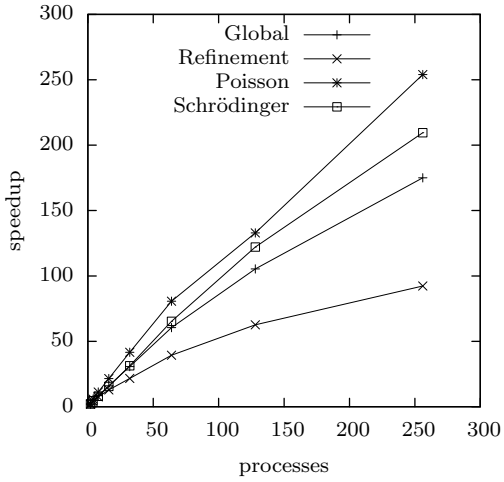


Figure 6.7: Speedup computing the electronic configuration of H.

results numerically. The latter is to be the concern of a more specific applications than the scheme proposed in this contribution. In the majority of other works "specialized" techniques are used that, while sophisticated and accurate, are not generally applicable (see discussion in §6.1). Our methods on the other hand, are integrable into other computational schemes and models with ease: that is, as long as a potential exists and adaptive methods are used, and/or a high-performance generalized eigenvalue solver is required.

Over a simple algorithm for solving the KS equation set, we detailed critical aspects from the point of view of the overall computational effort in time and memory in the case of employing h–adaptive grid techniques and subspace recycling. We evaluate some approaches for those critical decisions, such as the heuristic for selecting the cell to be refined and the configuration for the linear system and eigenproblem iterative solvers. Specifically, we provide experimental evidence that the use of the projected potential heuristic combined with Kelly's refinement method yields efficient grids, reducing the memory requirements and computational cost. Also, the overhead of the eigensolver can be significantly alleviated by Generalized Davidson with an appropriate starting subspace management that exploits the previous step solution as initial guess.

The following algorithm is therefore proposed that is expected to make the most of estimating errors based on the potential and alleviating computational cost: (i) An initial grid is constructed, deeming it possible to start from an appropriately refined grid while having not computed any solution vectors (with the projected potential heuristic); that is, maximum accuracy in the solution, minimal number of degrees of freedom. (ii) The self-consistent iterative technique of solving the

equations of DFT can then proceed using, at each self-consistent cycle, the previous eigenvectors as an approximate starting space to feed the eigensolver – in the case investigated here, that was the GD solver. (iii) Iterations over each fully converged self-consistent cycle can be successively improved on by using the Kelly error estimate to improve on the initial grid.

A rigorous numerical test of the techniques developed in this work using more sophisticated versions of the equations of DFT is desirable. Time dependency is a good test ground on which our scheme may have a significant impact by reducing the overall cost of computation. For now, these and other possibilities remain as the outlook for the future. A relatively straightforward extension to the current scheme would be to introduce nonlocal terms into the Schrödinger equation; which was supposed not to have a significant impact on the robustness of the methods used in this work.

In any case, if the ultimate goal of real–space approaches to electronic structure theory on the atomic scale is to perform fast and highly accurate time-dependent computations of atomic/molecular structure; then our scheme is a promising candidate for that purpose.

Chapter 7

Conclusions and Future Work

The solution of eigenvalue problems is considered one of the most important tasks in many scientific and engineering applications. Very frequently they requiere the computation of a small quantity of solutions of problems with certain structure, such as symmetric-definite, although the non-structured cases are also worth considering; and whose matrices have a scalable matrix-vector product, in the sense that its cost is far from $\mathcal{O}(n^2)$ for a matrix with dimension n, being a remarkable example the case of sparse matrices. In this contexti, iterative methods present important advantages from the point of view of the computational resources utilized.

The thesis is concerned with the implementation of Davidson-type methods, a subclass of the subspace iterative methods, for the solution of Hermitian and non-Hermitian large-scale partial eigenvalue problems, both standard and generalized. The implementations are integrated in the SLEPc library and incorporate (i) state-of-the-art expansion methods, such as Generalized Davidson and Jacobi-Davidson (with the Newton stopping criteria), besides the new expansion GD2; (ii) state-of-the-art extraction techniques, such as the Rayleigh-Ritz method and harmonic variants; and (iii) restart techniques, such as GD+k. The solvers are robust and efficient, exploiting the structural properties of the problem and performing the operations in real arithmetic as much as possible.

The main contribution of this thesis is that some features provided by the SLEPc Davidson solvers are not available in other free parallel libraries, such as the support of non-Hermitian or generalized problems and the harmonic extraction techniques. The software profits from the PETSc framework, inheriting the access to a wide variety of iterative linear system methods and preconditioners, and the computational capabilities of high performance hardware, specially supporting distributed memory.

The presented solvers were validated by their testing in a collection of problems whose matrices come from real applications, and the comparison of the results for Hermitian cases, considering the convergence and the computational performance

against similar libraries. In short, the Jacobi-Davidson solver presents slightly worse performance than the one available in PRIMME (because of the lack in the SLEPc implementation of sensible stopping criteria for the correction equation), and the Generalized Davidson is as competitive as the versions available in PRIMME and Anasazi. In addition, the new method to expand the subspace, GD2, provides better results compared with Generalized Davidson when the preconditioner is very far from the ideal.

As a consequence of the integration on the described frameworks of SLEPc and PETSc, the presented solvers, as the rest of SLEPc components, can interoperate in other applications and be configured not only in terms of problem properties, but also in the solution requirements through the convergence criterion (to indicate the solution quality) and the sort criterion (to specify the region in which to find solutions, *e.g.*, largest magnitude, rightmost/leftmost values, closest to a target).

This has been illustrated addressing two relevant scientific computing applications, in which obtaining the corresponding eigenpairs is challenging for iterative solvers. One of them is in the context of computation of plasma unstable modes with the code GENE, in which we provided examples of Jacobi-Davidson improving the performance of calculating eigenvalues, some of them initialized with previous solutions of similar problems. In the other, Generalized Davidson obtains the leftmost eigenvalues from problems that come from the discretization of the Schrödinger equation in an application for computing the electronic configurations of atoms.

SLEPc is currently an active project and there are plans for the continuation of the development of the Davidson implementation in order to address more problems and improve its efficiency and robustness. Concretely, a list of near future work includes

- the implementation of two-sided and alternating Jacobi-Davidson [Hochstenbach and Sleijpen, 2003], which have asymptotically cubic convergence finding both left and right eigenvectors at the same time;
- the implementation of robust Davidson methods specific for SVD [Hochstenbach, 2001] and polynomial eigenproblems [Sleijpen et al., 1996; Voss, 2007; Hochstenbach and Sleijpen, 2008] (these problems can be currently solved by the SVD and QEP SLEPc solvers that linearize the problem to a linear eigenvalue problem);
- the implementation of sensible stopping criteria, such as the one described for CG in [Notay, 2002] and QMR in [Stathopoulos, 2007] (implemented by PRIMME), and the more general criteria described in [Hochstenbach and Notay, 2009], which support non-Hermitian problems and can be efficiently employed in many iterative solvers for linear systems;
- the implementation of dense eigensolvers for the projected problem in the case of generalized symmetric-indefinite problems [Brebner and Grad, 1982]

and polynomial eigenproblems [Betcke and Kressner, 2011], that can make the solvers more robust; and

- the improvement of the support of the SLEPc Davidson solvers for novel architecture hardware such as multi-cores and GPUs, that currently is limited to accelerate the matrix-vector product and some vector-vector operations.

About the mentioned applications, some parts of the interface to PETSc/SLEPc can be improved, *e.g.*, the creation of the problem matrices in PETSc formats or the interchange of vectors between the application and PETSc. In addition, extending the SLEPc sorting criteria interface to consider the vectors also can be interesting for the code GENE. Finally the DFT code can be extended with more sophisticated versions, for instance considering time dependency.

7.1 Publications

The following list of publication in conferences and journals has been produced in the context of this thesis:

- E. Romero and Jose E. Roman. A parallel implementation of the trace minimization eigensolver. In J. M. Palma, P. R. Amestoy, M. Daydé, M. Mattoso, and J. C. Lopes, editors, *High Performance Computing for Computational Science - VECPAR 2008*, volume 5336 of *Lect. Notes Comp. Sci.*, pages 255–268, 2008.

- T. D. Young, E. Romero and J. E. Roman. Finite Element Solution of the Stationary Schrödinger Equation Using Standard Computational Tools. *Proceedings of the International Conference on Computational and Mathematical Methods in Science and Engineering, CMMSE 2009*, 1140–1150, 2009.

- M. B. Cruz, E. Romero, J. E. Roman and P. B. Vasconcelos. Uma implementação paralela do método de Jacobi-Davidson para problemas de valores próprios não simétricos de grande dimensão. *Congreso de Métodos Numéricos en Ingeniería 2009, METNUM 2009.*

- E. Romero and J. E. Roman. A Parallel Implementation of the Davidson Method for Generalized Eigenproblems. In B. Chapman, F. Desprez, G. R. Joubert, A. Lichnewsky, F. Peters and T. Priol, editors, *Parallel Computing: From Multicores and GPU's to Petascale*, volume 19 of *Advances in Parallel Computing*, pages 133–140. IOS Press, 2010.

- E. Romero, M. B. Cruz, J. E. Roman and P. B.Vasconcelos. A Parallel Implementation of the Jacobi-Davidson Eigensolver for Unsymmetric Matrices. In J. M. L. M. Palma, M. Daydé, O. Marques, and J. C. Lopes, editors, *High Performance Computing for Computational Science – VECPAR 2010*, volume 6449 of *Lect. Notes Comp. Sci.*, pages 380–393. Springer, 2011.

- E. Romero and Jose E. Roman. Computing Subdominant Unstable Modes of Turbulent Plasma with a Parallel Jacobi-Davidson Eigensolver, *Concur. Comput.: Pract. Exp*, 23(17):2179–2191, 2011.

- F. Merz, C. Kowitz, E. Romero, J. E. Roman and F. Jenko. Multi-dimensional gyrokinetic parameter studies based on eigenvalue computations, *Comput. Phys. Commun.*, 183(4):922–930, 2012.

- T. D. Young, E. Romero and J. E. Roman. Parallel Finite Element Density Functional Computations Exploiting Grid Refinement and Subspace Recycling. Submitted to *Comput. Phys. Commun.*

7.2 Projects

The work developed in this thesis was partially supported by the Spanish Ministry of Science and Innovation (MICINN) under the projects

- *Advanced methods and novel computational techniques for the numerical solution of large-scale eigenvalue problems*, grant number TIN2009-07519, whose objective is to extend SLEPc, besides the inclusion of the Davidson solvers and the harmonic extraction, in order to support quadratic eigenproblems and improve the performance in multi-core and GPUs computing environments; and

- *Numerical Methods for Spectral Computations: Development and Implementation in Parallel Computers*, in the context of the Spanish-Portuguese Integrated Action, that funded the collaboration with M. B. Cruz and P. B. Vasconcelos in the initial development of the Jacobi-Davidson for standard non-Hermitian problems, among other works.

7.3 Software based on SLEPc

The SLEPc source code, in which the Davidson solvers described in this thesis are included, is freely distributed in Internet under the GPL license since 2003, encouraging its use in other software projects as a specialized component for solving eigenvalue problems. Some software using SLEPc is listed next, for a more complete reference see [Roman, 2011]:

FEniCS a toolkit for the Automation of Computational Mathematical Modeling (ACMM);

libMesh a C++ framework for the numerical simulation of partial differential equations;

deal.II a finite element Differential Equations Analysis Library;

Elefant Efficient Learning, Large-scale Inference, and Optimisation Toolkit;

TiberCAD Multiscale Device Simulator;

GENE Gyrokinetic Electromagnetic Numerical Experiment;

GYRO The General Atomics TGYRO Code Suite;

OpenCMISS Open Continuum Mechanics, Imaging, Signal processing and System identification;

Milonga a free nuclear reactor core analysis code;

Dome tools for power system analysis.

The Davidson solvers in SLEPc are available from SLEPc version 3.1, and some libraries are starting to using them, like for instance in GENE, which employs the Jacobi-Davidson solver as default (configured similarly as described in the experiments of chapters 4 and 5); and deal.II, which added the corresponding wrappers so that they can be used by the rest of the library.

Bibliography

J. Ackermann, B. Erdmann, and R. Roitzsch. A self–adaptive multilevel finite element method for the stationary Schrödinger equation in three space dimensions. *Chem. Phys. Lett.*, 101:7643, 1994.

J. Ackermann and R. Roitzsch. A two-dimensional multilevel adaptive finite element method for the time-independent Schrödinger equation. *Chem. Phys. Lett.*, 214(1):109–117, 1993.

M. Ainsworth and J. T. Oden. *A Posteriori Error Estimation in Finite Element Analysis*. John Wiley and Sons, 2000.

E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1992.

P. Arbenz, M. Becka, R. Geus, U. Hetmaniuk, and T. Mengotti. On a parallel multilevel preconditioned Maxwell eigensolver. *Parallel Comput.*, 32(2):157–165, 2006.

P. Arbenz and R. Geus. A comparison of solvers for large eigenvalue problems occuring in the design of resonant cavities. *Numer. Linear Algebra Appl.*, 6(1):3–16, 1999.

I. Babuška and T. Strouboulis. *The Finite Element Method and its Reliability*. Claredon Press, New York, 2001.

Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.

C. G. Baker, U. L. Hetmaniuk, R. B. Lehoucq, and H. K. Thornquist. Anasazi software for the numerical solution of large-scale eigenvalue problems. *ACM Trans. Math. Softw.*, 36(3):13:1–13:23, 2009.

S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.2, Argonne National Laboratory, 2011.

S. Balay, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.1, Argonne National Laboratory, 2010.

S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhaüser, 1997.

W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Trans. Math. Softw.*, ???? (Submitted).

W. Bangerth, R. Hartmann, and G. Kanschat. deal.IIa general-purpose object-oriented finite element library. *ACM Trans. Math. Softw.*, 33, 2007.

T. L. Beck. Real-space mesh techniques in density-functional theory. *Rev. Mod. Phys.*, 72:1041, 2000.

M. A. Beer, S. C. Cowley, and G. W. Hammett. Field-aligned coordinates for nonlinear simulations of tokamak turbulence. *Phys. Plasmas*, 2(7):2687–2700, 1995.

A. F. Bertolini. Review of eigensolution procedures for linear dynamic finite element analysis. *Appl. Mech. Rev.*, 51(2):155–172, 1998.

T. Betcke and D. Kressner. Perturbation, extraction and refinement of invariant pairs for matrix polynomials. *Linear Algebra and its Applications*, 435(3):514–536, 2011.

L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of Basic Linear Algebra Subprograms (BLAS). *ACM Trans. Math. Softw.*, 28(2):135–151, 2002.

M. Bollhöfer and Y. Notay. JADAMILU: a software code for computing selected eigenvalues of large sparse symmetric matrices. *Comput. Phys. Commun.*, 177(12):951–964, 2007.

M. A. Brebner and J. Grad. Eigenvalues of $Ax = \lambda Bx$ for real symmetric matrices A and B computed by reduction to a pseudosymmetric form and the HR process. *Linear Algebra Appl.*, 43:99–118, 1982.

A. J. Brizard and T. S. Hahm. Foundations of nonlinear gyrokinetic theory. *Rev. Mod. Phys.*, 79:421–468, 2007.

C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.*, 33(3):1103–1133, 2011.

X.-C. Cai and M. Sarkis. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM J. Sci. Comput.*, 21(22):792–797, 1999.

C. Campos, J. E. Roman, E. Romero, and A. Tomas. SLEPc users manual. Technical Report DSIC-II/24/02 - Revision 3.2, D. Sistemes Informàtics i Computació, Universitat Politècnica de València, 2011. Available at http://www.grycap.upv.es/slepc.

M. Crouzeix, B. Philippe, and M. Sadkane. The Davidson method. *SIAM J. Sci. Comput.*, 15(1):62–76, 1994.

T. Dannert and F. Jenko. Gyrokinetic simulation of collisionless trapped-electron mode turbulence. *Phys. Plasmas*, 12(7):072309, 2005.

E. R. Davidson. The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *J. Comput. Phys.*, 17(1):87–94, 1975.

T. A. Davis and Y. Hu. The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, 2011.

T. Ericsson and A. Ruhe. The spectral transformation Lanczos method for the numerical solution of large sparse generalized symmetric eigenvalue problems. *Math. Comp.*, 35(152):1251–1268, 1980.

J.-L. Fattebert, R. D. Hornung, and A. M. Wissink. Finite element approach for density functional theory calculations on locally refined meshes. *J. Comput. Phys.*, 223(2):759–773, 2007.

M. Ferronato, C. Janna, and G. Pini. Efficient parallel solution to large-size sparse eigenproblems with block FSAI preconditioning. *Numer. Linear Algebra Appl.*, 2012. In press.

D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst. Jacobi–Davidson style QR and QZ algorithms for the reduction of matrix pencils. *SIAM J. Sci. Comput.*, 20(1):94–125, 1999.

J. G. F. Francis. The QR transformation: a unitary analogue to the LR transformation. *Comput. J.*, 4(3):265–271, 1961.

M. A. Freitag and A. Spence. Convergence theory for inexact inverse iteration applied to the generalised nonsymmetric eigenproblem. *Electron. Trans. Numer. Anal.*, 28:40–64, 2007.

M. Genseberger. Improving the parallel performance of a domain decomposition preconditioning technique in the Jacobi-Davidson method for large scale eigenvalue problems. *App. Numer. Math.*, 60(11):1083–1099, 2010.

A. George and J. W. H. Liu. A fast implementation of the minimum degree algorithm using quotient graphs. *ACM Trans. Math. Softw.*, 6:337–358, 1980.

R. Geus. *The Jacobi-Davidson algorithm for solving large sparse symmetric eigenvalue problems with application to the design of accelerator cavities*. Ph.D. thesis, ETH Zürich, 2002.

T. Görler, X. Lapillonne, S. Brunner, T. Dannert, F. Jenko, F. Merz, and D. Told. The global version of the gyrokinetic turbulence code GENE. *J. Comput. Phys.*, 230(18):7053–7071, 2011.

M. N. Guimarães and F. V. Prudente. A study of the confined Hydrogen atom using the finite element method. *J. Phys. B: At. Mol. Opt. Phys.*, 38:2811, 2005.

V. Hernandez, J. E. Roman, and A. Tomas. Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement. *Parallel Comput.*, 33(7–8):521–540, 2007.

V. Hernandez, J. E. Roman, A. Tomas, and V. Vidal. A survey of software for sparse eigenvalue problems. Technical Report STR-6, Universidad Politécnica de Valencia, 2006. Available at http://www.grycap.upv.es/slepc.

V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Softw.*, 31(3):351–362, 2005.

V. Heuveline, B. Philippe, and M. Sadkane. Parallel computation of spectral portrait of large matrices by Davidson type methods. *Numer. Algorithms*, 16(1):55–75, 1997.

D. J. Higham and N. J. Higham. Structured backward error and condition of generalized eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 20(2):493–512, 1998.

M. Hochstenbach. Jacobi-Davidson Gateway. 2007.

M. E. Hochstenbach. A Jacobi–Davidson type SVD method. *SIAM J. Sci. Statist. Comput.*, 23(2):606–628, 2001.

M. E. Hochstenbach. Generalizations of harmonic and refined Rayleigh-Ritz. *Electron. Trans. Numer. Anal.*, 20:235–252, 2005a.

M. E. Hochstenbach. Variations on harmonic Rayleigh–Ritz for standard and generalized eigenproblems, 2005b. Preprint, Department of Mathematics, Case Western Reserve University.

M. E. Hochstenbach and Y. Notay. The Jacobi–Davidson method. *GAMM Mitt.*, 29(2):368–382, 2006.

M. E. Hochstenbach and Y. Notay. Controlling inner iterations in the Jacobi–Davidson method. *SIAM J. Matrix Anal. Appl.*, 31(2):460–477, 2009.

M. E. Hochstenbach and G. L. Sleijpen. Two-sided and alternating Jacobi-Davidson. *Linear Algebra Appl.*, 358(1-3):145–172, 2003.

M. E. Hochstenbach and G. L. G. Sleijpen. Harmonic and refined Rayleigh-Ritz for the polynomial eigenvalue problem. *Numer. Linear Algebra Appl.*, 15(1):35–54, 2008.

P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136(3B):B864–B871, 1964.

F.-N. Hwang, Z.-H. Wei, T.-M. Huang, and W. Wang. A parallel additive Schwarz preconditioned Jacobi-Davidson algorithm for polynomial eigenvalue problems in quantum dot simulation. *J. Comput. Phys.*, 229(8):2932–2947, 2010.

C. G. J. Jacobi. Über ein leichtes Verfahren die in der Theorie der Säculärstörungen vorkommenden Gleichungen numerisch aufzulösen. *Crelle's J.*, 30:51–94, 1846.

F. Jenko, T. Dannert, and C. Angioni. Heat and particle transport in a tokamak: advances in nonlinear gyrokinetics. *Plasma Phys. Control. Fusion*, 47(12B):B195, 2005.

F. Jenko, W. Dorland, M. Kotschenreuther, and B. N. Rogers. Electron temperature gradient driven turbulence. *Phys. Plasmas*, 7(5):1904–1910, 2000.

S. G. Johnson and J. D. Joannopoulos. Block-iterative frequency-domain methods for Maxwell's equations in a planewave basis. *Opt. Express*, 8(3):173–190, 2001.

M. Kammerer, F. Merz, and F. Jenko. Exceptional points in linear gyrokinetics. *Phys. Plasmas*, 15(5):052102, 2008.

D. W. Kelly, J. P. de S. R. Gago, O. Zienkiewicz, and I. Babuska. A posteriori error analysis and adaptive processes in the finite element method, Part I: error analysis. *Internat. J. Numer. Methods Engrg.*, 19:1593–1619, 1983.

A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient method. *SIAM J. Sci. Comput.*, 23(2):517–541, 2001.

A. V. Knyazev, M. E. Argentati, I. Lashuk, and E. E. Ovtchinnikov. Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX) in HYPRE and PETSc. *SIAM J. Sci. Comput.*, 29(5):2224–2239, 2007.

W. Kohn and L. J. Sham. Self-Consistent equations including exchange and correlation effects. *Phys. Rev.*, 140(4A):A1133–A1138, 1965.

J. Kopal, M. Rozložník, M. Tuma, and A. Smoktunowicz. Rounding error analysis of orthogonalization with a non-standard inner product. *Numer. Math.*, 2011. Submitted.

D. Kressner. Block algorithms for reordering standard and generalized Schur forms. *ACM Trans. Math. Softw.*, 32(4):521–532, 2006.

R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide, Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1998.

Z. Li, Y. Saad, and M. Sosonkina. pARMS: a parallel version of the algebraic recursive multilevel solver. *Numer. Linear Algebra Appl.*, 10(5-6):485–509, 2003.

F. Merz. *Gyrokinetic simulation of multimode plasma turbulence.* Ph.D. thesis, University of Münster, 2008. Available at `http://en.scientificcommons.org/41889457`. Accessed 1 December 2011.

F. Merz and F. Jenko. Nonlinear interplay of TEM and ITG turbulence and its effect on transport. *Nuclear Fusion*, 50:054005, 2010.

F. Merz, C. Kowitz, E. Romero, J. E. Roman, and F. Jenko. Multi-dimensional gyrokinetic parameter studies based on eigenvalues computations. *Comput. Phys. Commun.*, 183(4):922–930, 2012.

R. B. Morgan. Davidson's method and preconditioning for generalized eigenvalue problems. *J. Comput. Phys.*, 89:241–245, 1990.

R. B. Morgan. Computing interior eigenvalues of large matrices. *Linear Algebra Appl.*, 154–156:289–309, 1991.

R. B. Morgan. Generalizations of Davidson's method for computing eigenvalues of large nonsymmetric matrices. *J. Comput. Phys.*, 101:287, 1992.

R. B. Morgan. GMRES with deflated restarting. *SIAM J. Sci. Comput.*, 24(1):20–37, 2002.

R. B. Morgan and D. S. Scott. Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices. *SIAM J. Sci. Statist. Comput.*, 7(3):817–825, 1986.

MPI Forum. MPI: a message-passing interface standard. *Int. J. Supercomp. Applic. High Perf. Comp.*, 8(3/4):159–416, 1994.

E. D. Napoli, S. Blügel, and P. Bientinesi. Correlations in sequences of generalized eigenproblems arising in density functional theory. *arXiv:1108.2594v1 : retrieved 12 Aug 2011*, 2011.

R. Natarajan and D. Vanderbilt. A new iterative scheme for obtaining eigenvectors of large, real-symmetric matrices. *J. Comput. Phys.*, 82(1):218–228, 1989.

K. Neymeyr. A geometric theory for preconditioned inverse iteration I: Extrema of the Rayleigh quotient. *Linear Algebra Appl.*, 322(1-3):61–85, 2001.

M. Nool and A. van der Ploeg. A parallel Jacobi–Davidson-type method for solving large generalized eigenvalue problems in magnetohydrodynamics. *SIAM J. Sci. Comput.*, 22(1):95–112, 2000.

Y. Notay. Combination of Jacobi-Davidson and conjugate gradients for the partial symmetric eigenproblem. *Numer. Linear Algebra Appl.*, 9(1):21–44, 2002.

Y. Notay. Inner iterations in eigenvalue solvers. Technical Report GANMN 05-01, Service de Métrologie Nucléaire, Université Libre de Bruxelles, 2005.

J. Olsen, P. Jørgensen, and J. Simons. Passing the one-billion limit in full configuration-interaction (FCI) calculations. *Chem. Phys. Lett.*, 169(6):463–472, 1990.

C. C. Paige, B. N. Parlett, and H. A. van der Vorst. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Numer. Linear Algebra Appl.*, 2(2):115–133, 1995.

B. N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, Englewood Cliffs, NJ, 1980. Reissued with revisions by SIAM, Philadelphia, 1998.

R. G. Parr and W. Yang. *Density–Functional Theory of Atoms and Molecules*. Number 16 in International series of monographs on chemistry. Oxford University Press, New York, 1989.

J. E. Pask and P. A. Sterne. Finite element methods in *ab initio* electronic structure calculations. *Modelling Simul. Mater. Sci. Eng.*, 13:R71, 2005.

L. R. Ram-Mohan. *Finite element and boundary element applications in quantum mechanics*. Oxford University Press, New York, 2002.

E. Ramos, J. E. Roman, S. Cardona-Serra, and J. M. Clemente-Juan. Parallel implementation of the MAGPACK package for the analysis of high-nuclearity spin clusters. *Comput. Phys. Commun.*, 181(12):1929–1940, 2010.

M. J. Rayson. Lagrange–Lobatto interpolating polynomials in the discrete variable representation. *Phys. Rev. E*, 76(2):026704, 2007.

J. E. Roman. SLEPc web page. 2011.

J. E. Roman, M. Kammerer, F. Merz, and F. Jenko. Fast eigenvalue calculations in a massively parallel plasma turbulence code. *Parallel Comput.*, 36(5-6):339–358, 2010.

E. Romero and J. E. Roman. A parallel implementation of the Jacobi-Davidson eigensolver and its application in a plasma turbulence code. In P. D'Ambra, M. Guarracino, and D. Talia, editors, *Euro-Par 2010, Part II*, volume 6272 of *Lect. Notes Comp. Sci.*, pages 101–112. Springer, 2010.

E. Romero and J. E. Roman. Computing subdominant unstable modes of turbulent plasma with a parallel Jacobi–Davidson eigensolver. *Concur. Comput.: Pract. Exp*, 23(17):2179–2191, 2011.

Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, 1993.

Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM Publications, 2nd edition, 2003.

Y. Saad. *Numerical Methods for Large Eigenvalue Problems, Revised Edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2011.

M. Sadkane and R. B. Sidje. Implementation of a variable block Davidson method with deflation for solving large sparse eigenproblems. *Numer. Algorithms*, 20(2–3):217–240, 1999.

B. J. Schneider, L. A. Collins, and S. X. Hu. Parallel solver for the time–dependent linear and nonlinear Schrödinger equation. *Phys. Rev. E*, 73:036708, 2006.

G. L. G. Sleijpen, A. G. L. Booten, D. R. Fokkema, and H. A. van der Vorst. Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems. *BIT*, 36(3):595–633, 1996.

G. L. G. Sleijpen and D. R. Fokkema. BiCGstab(ℓ) for linear equations involving unsymmetric matrices with complex spectrum. *Electron. Trans. Numer. Anal.*, 1:11–32, 1993.

G. L. G. Sleijpen and H. A. van der Vorst. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17(2):401–425, 1996.

G. L. G. Sleijpen and H. A. van der Vorst. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM Rev.*, 42(2):267–293, 2000.

G. L. G. Sleijpen, H. A. van der Vorst, and E. Meijerink. Efficient expansion of subspaces in the Jacobi–Davidson method for standard and generalized eigenproblems. *Electron. Trans. Numer. Anal.*, 7:75–89, 1998.

M. Sosonkina, Y. Saad, and X. Cai. Using the parallel algebraic recursive multilevel solver in modern physical applications. *Future Generation Computer Systems*, 20:489, 2004.

A. Stathopoulos. Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part I: Seeking one eigenvalue. *SIAM J. Sci. Comput.*, 29(2):481–514, 2007.

A. Stathopoulos and C. F. Fischer. A Davidson program for finding a few selected extreme eigenpairs of a large, sparse, real, symmetric matrix. *Comput. Phys. Commun.*, 79:268, 1994.

A. Stathopoulos and J. R. McCombs. Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part II: Seeking many eigenvalues. *SIAM J. Sci. Comput.*, 29(5):2162–2188, 2007.

A. Stathopoulos and J. R. McCombs. PRIMME: PReconditioned Iterative MultiMethod Eigensolver: Methods and software description. *ACM Trans. Math. Softw.*, 37(2):21:1–21:30, 2010.

A. Stathopoulos and Y. Saad. Restarting techniques for the (Jacobi-)Davidson symmetric eigenvalue methods. *Electron. Trans. Numer. Anal.*, 7:163–181, 1998.

A. Stathopoulos, Y. Saad, and C. F. Fischer. Robust preconditioning of large, sparse, symmetric eigenvalue problems. *J. Comput. Appl. Math.*, 64(3):197–215, 1995.

A. Stathopoulos, Y. Saad, and K. Wu. Dynamic thick restarting of the Davidson, and the implicitly restarted Arnoldi methods. *SIAM J. Sci. Comput.*, 19(1):227–245, 1998.

A. Stathopoulos and K. Wu. A block orthogonalization procedure with constant synchronization requirements. *SIAM J. Sci. Comput.*, 23(6):2165–2182, 2002.

G. W. Stewart. A Krylov–Schur algorithm for large eigenproblems. *SIAM J. Matrix Anal. Appl.*, 23(3):601–614, 2001a.

G. W. Stewart. *Matrix Algorithms. Volume II: Eigensystems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001b.

M. Sugawara. Adaptive basis set for quantum mechanical calculation based on hierarchial finite element method. *Chem. Phys. Lett.*, 295(5-6):423–430, 1998.

E. Tsuchida and M. Tsukada. Electronic-structure calculations based on the finite-element method. *Phys. Rev. B*, 52(8):5573, 1995.

E. Tsuchida and M. Tsukada. Adaptive finite–element method for electronic–structure calculations. *Phys. Rev. B*, 54(11):7602, 1996.

E. Tsuchida and M. Tsukada. Large-scale electronic-structure calculations based on the adaptive finite-element method. *J. Phys. Soc. Jpn.*, 67(11):3844–3858, 1998.

H. A. van der Vorst. Computational methods for large eigenvalue problems. In P. G. Ciarlet and J. L. Lions, editors, *Handbook of Numerical Analysis*, volume VIII, pages 3–179. Elsevier, Amsterdam, 2002.

H. A. van der Vorst. Modern methods for the iterative computation of eigenpairs of matrices of high dimension. *Z. Angew. Math. Mech.*, 84(7):444–451, 2004.

J. H. van Lenthe and P. Pulay. A space-saving modification of Davidson's eigenvector algorithm. *J. Comput. Chem.*, 11(10):1164–1168, 1990.

T. van Noorden and J. Rommes. Computing a partial generalized real Schur form using the Jacobi–Davidson method. *Numer. Linear Algebra Appl.*, 14(3):197–215, 2007.

C. Vömel, S. Z. Tomov, O. A. Marques, A. Canning, L.-W. Wang, and J. J. Dongarra. State-of-the-art eigensolvers for electronic structure calculations of large scale nano-systems. *J. Comput. Phys.*, 227(15):7113–7124, 2008.

H. Voss. A Jacobi-Davidson method for nonlinear and nonsymmetric eigenproblems. *Comput. & Structures*, 85(17-18):1284–1292, 2007.

S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of sparse matrix–vector multiplication on emerging multicore platforms. *Parallel Comput.*, 35(3):178–194, 2009.

K. Wu, Y. Saad, and A. Stathopoulos. Inexact Newton preconditioning techniques for large symmetric eigenvalue problems. *Electron. Trans. Numer. Anal.*, 7:202–214, 1998.

S. Yamakawa and S. Hyodo. Gaussian finite-element mixed-basis method for electronic structure calculations. *Phys. Rev. B*, 71:035113, 2005.

T. D. Young. `namespace ewalena`*, Documentation and technical reference*. IFTR of the Polish Academy of Sciences, pre–alpha edition, 2011a. Available at `http://www.ewalena.sourceforge.net`.

T. D. Young. A qualitative semi–classical treatment of an isolated semi–polar quantum dot. *J. Phys.: Conf. Ser.*, 281:012015, 2011b.

T. D. Young and R. Armiento. Strategies for h-adaptive refinement for a finite element treatment of harmonic oscillator Schrödinger eigenproblem. *Commun. Theor. Phys.*, 53(6):1017, 2010.

T. D. Young, E. Romero, and J. E. Román. Finite element solution of the stationary Schrödinger equation using standard computational tools. In *Proceedings of the International Conference on Computational and Mathematical Methods in Science and Engineering 2009*, page 1140. Gijón, Asturias, Spain, 2009.