Computer Physics Communications 183 (2012) 922–930

Contents lists available at SciVerse ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Multi-dimensional gyrokinetic parameter studies based on eigenvalue computations

F. Merz [a], C. Kowitz [a,c], E. Romero [b], J.E. Roman [b,*], F. Jenko [a]

[a] Max-Planck-Institut für Plasmaphysik, Boltzmannstr. 2, 85748 Garching, Germany
[b] Institut I3M, Universitat Politècnica de València, Camí de Vera s/n, 46022 València, Spain
[c] Institut für Informatik, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany

ARTICLE INFO

Article history:
Received 27 September 2011
Received in revised form 2 December 2011
Accepted 21 December 2011
Available online 23 December 2011

Keywords:
Gyrokinetic equations
GENE code
Eigenvalue computation
Parameter scans

ABSTRACT

Plasma microinstabilities, which can be described in the framework of the linear gyrokinetic equations, are routinely computed in the context of stability analyses and transport predictions for magnetic confinement fusion experiments. The GENE code, which solves the gyrokinetic equations, has been coupled to the SLEPc package for an efficient iterative, matrix-free, and parallel computation of rightmost eigenvalues. This setup is presented, including the preconditioner which is necessary for the newly implemented Jacobi–Davidson solver. The fast computation of instabilities at a single parameter set is exploited to make parameter scans viable, that is to compute the solution at many points in the parameter space. Several issues related to parameter scans are discussed, such as an efficient parallelization over parameter sets and subspace recycling.

1. Introduction

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 [1] which is a reduced kinetic model, neglecting the fast gyrophase dependence. It describes the plasma as a collection of quasiparticles (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 [2]. 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 [3]

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

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 behaviour of the system. Furthermore, the eigenvalues and -vectors can be used to construct quasilinear models (see, e.g., Ref. [4]).

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

* Corresponding author.
E-mail addresses: Florian.Merz@ipp.mpg.de (F. Merz), kowitz@in.tum.de (C. Kowitz), elroal@upvnet.upv.es (E. Romero), jroman@dsic.upv.es (J.E. Roman), fsj@ipp.mpg.de (F. Jenko).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2011.12.018

Table 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

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 Ref. [4] 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 paper 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 paper. In Section 3, 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 4, 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. Finally, Section 6 closes with a summary.

2. 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 [5–8]. 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

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

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

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

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

$$\phi = \frac{\sum_s n_{0s}\pi q_s B_0 \int J_0(\lambda_s) g_s \, dv_\parallel \, d\mu}{k_\perp^2 \lambda_D^2 + \sum_s \frac{q_s^2}{T_{0s}} n_{0s}(1 - \Gamma_0(b_s))},$$

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

For the definitions of the prefactors, see [7]. 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 paper is specified in Table 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 1 and in the rest of the paper.

3. Fast eigenvalue computations

The GENE code was coupled to the SLEPc package [9,10] 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., Refs. [6,11,4]). 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 60 000 (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 [12], which allowed for the discovery of non-Hermitian degeneracies of gyrokinetic eigenmodes [3].

Recently, a Jacobi–Davidson solver has been implemented in SLEPc [13,14]. 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.

3.1. 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 paper.

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,

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

as in the Generalized Davidson (GD) method [15], 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 [16] 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 paper we use

$$PK^{-1}(A-\theta I)Pt = -\hat{r}, \qquad P = I - \frac{K^{-1}zu^*}{u^*K^{-1}z}, \quad t \perp u, \tag{2}$$

where $z \in \operatorname{span}\{Au, u\}$ and $\hat{r} = PK^{-1}r$. Furthermore, we employ algorithmic techniques similar to the JDQZ variant [17], 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 [14], except for the preconditioning which will be treated in this paper.

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 [18]. 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 Section 4.

3.2. Overview of SLEPc and PETSc

SLEPc, the Scalable Library for Eigenvalue Problem Computations [9,10], 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 (Portable, Extensible Toolkit for Scientific Computation [19,20]), 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 [21] 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.

3.3. 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.

3.4. 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) [22] 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

$$L_g \approx \sum_i L_{gi}^{\delta} = \sum_i R_i^{\delta} L_g R_i^{\delta} \tag{3}$$

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

$$\sum_i K_i\left(L_{gi}^{\delta}\right) L_g x = \sum_i K_i\left(L_{gi}^{\delta}\right) b \tag{4}$$

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 [21] $\tilde{L}_i \tilde{U}_i$ with its inverse being computable cheaply via forward/backward substitution. This allows the construction of a preconditioner by

$$K = \sum_i K_i = \sum_i (\tilde{L}_i \tilde{U}_i)^{-1} \approx \sum_i \left(L_{gi}^{\delta}\right)^{-1} \approx L_g^{-1} \tag{5}$$

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 [23] 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.

3.5. pARMS preconditioner

The pARMS preconditioner [24,25] 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

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \tag{6}$$

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

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} \approx \begin{bmatrix} L & 0 \\ EU^{-1} & I \end{bmatrix} \begin{bmatrix} U & L^{-1}F \\ 0 & S \end{bmatrix}, \tag{7}$$

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 (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 [24,25].

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.

3.6. 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 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 Fig. 1 (right)), so we set $\sigma = 3$ as the default value.

3.6.1. 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

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

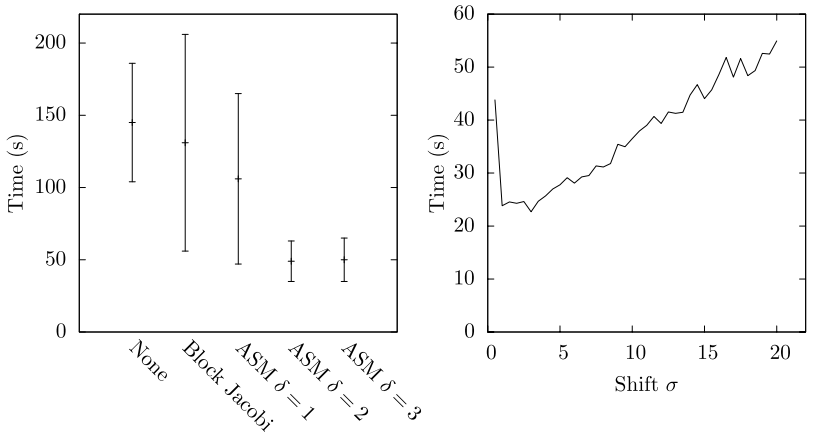

Fig. 1. Influence of the overlap δ (left) and the shift σ of the preconditioner matrix (right) on the total time. The plots show the mean and the standard deviation (left) and the minimum time (right) spent by JD with different domain decompositions.

Table 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					

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. Fig. 1 (left) compares the performance of JD solving the test case I with different overlap values, using ASM with ILU as the local preconditioner.

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 2 and the standard deviations of the time in Fig. 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.

3.6.2. 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 Fig. 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 Fig. 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 Fig. 2 (right).

4. Parameter scans

4.1. 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 significantly. 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

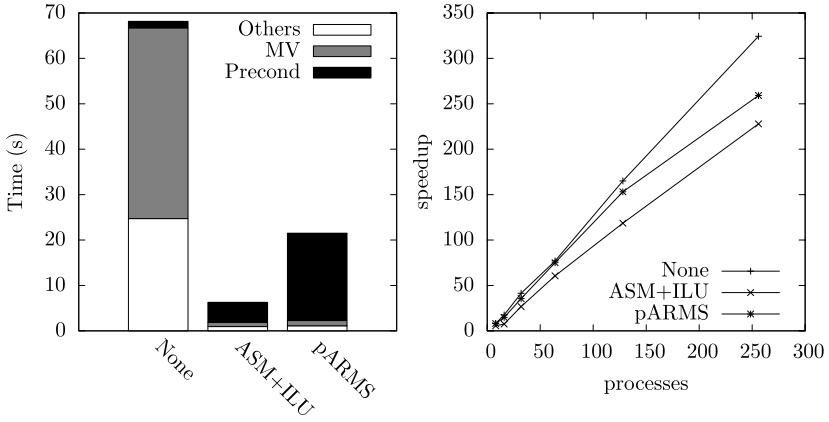

Fig. 2. Time spent with 16 processes (left) and speedup (right) of JD solving the test case I without preconditioner (None), with ASM preconditioner using $\delta = 2$ and the local preconditioner ILU (ASM+ILU), and with pARMS using the local preconditioner ARMS (pARMS). MV stands for matrix–vector products.

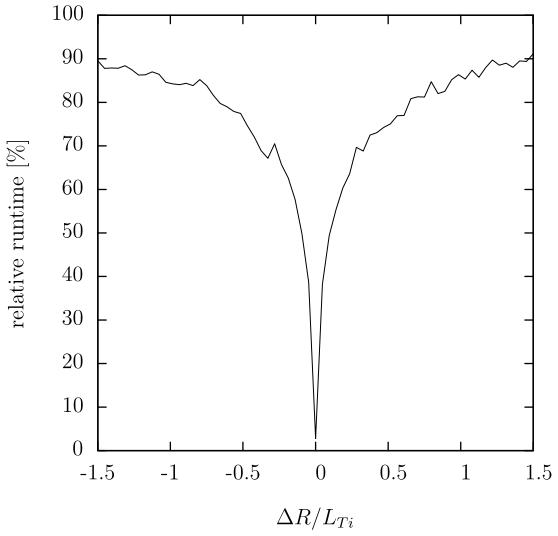

Fig. 3. Computation time for the eigenvalue problem with $R/L_{Ti} = 4.0$ as a function of the difference to the R/L_{Ti} value of the eigenvectors used as initial condition, normalized to the computation time with random initial vectors.

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. 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 $e_{i,a}$ ($a = 1..n_{ev}$) for the parameter sets $\vec{p}_i$ ($i = 1..n$, 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.

4.2. 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. 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 their $|\vec{\Delta}_i|$ and the closest 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 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

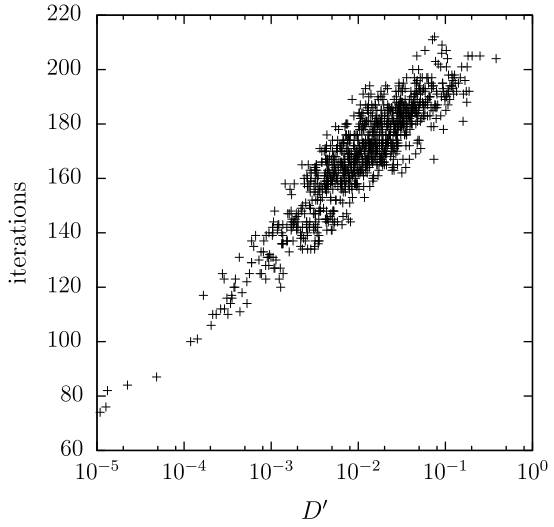

Fig. 4. Number of iterations for the eigenvalue problem with the nominal parameters as function of $D' = 1.0 - C(e_i, e_{n+1})$.

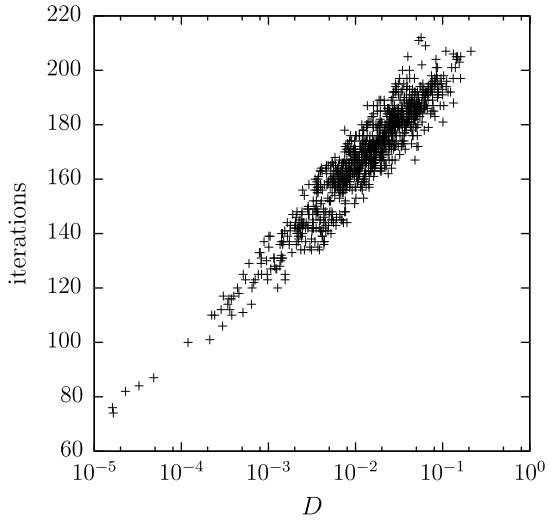

Fig. 5. Number of iterations for the eigenvalue problem with the nominal parameters as function of D.

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. Fig. 4 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. 4 plotted against $D(p_i, p_{n+1})$ is shown in Fig. 5.

4.3. 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 and communication overhead) 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

Table 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

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.

5. 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 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 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 3. 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.

6. Summary

In this paper, 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 paper will certainly contribute to a better understanding of the important driving mechanisms of turbulent transport in fusion plasmas.

Acknowledgements

E. Romero and J.E. Roman were supported by the Spanish Ministry of Science and Innovation (MICINN) under project number TIN2009-07519.

References

[1] A.J. Brizard, T.S. Hahm, Foundations of nonlinear gyrokinetic theory, Reviews of Modern Physics 79 (2007) 421.
[2] M.A. Beer, S.C. Cowley, G.W. Hammett, Field-aligned coordinates for nonlinear simulations of tokamak turbulence, Physics of Plasmas 2 (1995) 2767.
[3] M. Kammerer, F. Merz, F. Jenko, Exceptional points in linear gyrokinetics, Physics of Plasmas 15 (2008) 052102.
[4] F. Merz, F. Jenko, Nonlinear interplay of TEM and ITG turbulence and its effect on transport, Nuclear Fusion 50 (2010) 054005.
[5] F. Jenko, W. Dorland, M. Kotschenreuther, B.N. Rogers, Electron temperature gradient driven turbulence, Physics of Plasmas 7 (2000) 1904.
[6] T. Dannert, F. Jenko, Gyrokinetic simulation of collisionless trapped electron mode turbulence, Physics of Plasmas 12 (2005) 072309.
[7] 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.
[8] T. Goerler, X. Lapillonne, S. Brunner, T. Dannert, F. Jenko, F.M.D. Told, The global version of the gyrokinetic turbulence code gene, Journal of Computational Physics 230 (2011) 7053.
[9] V. Hernandez, J.E. Roman, V. Vidal, SLEPc: Scalable Library for Eigenvalue Problem Computations, Lecture Notes in Computer Science, vol. 2565, Springer, 2003, p. 377.
[10] V. Hernandez, J.E. Roman, V. Vidal, SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems, ACM Transactions on Mathematical Software 31 (2005) 351.
[11] F. Jenko, T. Dannert, C. Angioni, Heat and particle transport in a tokamak: Advances in nonlinear gyrokinetics, Plasma Physics and Controlled Fusion 47 (2005) B195.
[12] J.E. Roman, M. Kammerer, F. Merz, F. Jenko, Fast eigenvalue calculations in a massively parallel plasma turbulence code, Parallel Computing 36 (2010) 339.
[13] E. Romero, 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, D. Talia (Eds.), Euro-Par 2010, Part II, in: Lecture Notes in Computer Science, vol. 6272, Springer, 2010, p. 101.
[14] E. Romero, J.E. Roman, Computing subdominant unstable modes of turbulent plasma with a parallel Jacobi–Davidson eigensolver, Concurrency and Computation: Practice and Experience 23 (2011) 2179.

[15] R.B. Morgan, Generalizations of Davidson's method for computing eigenvalues of large nonsymmetric matrices, Journal of Computational Physics 101 (1992) 287.
[16] G.L.G. Sleijpen, H.A. van der Vorst, A. Jacobi–Davidson iteration method for linear eigenvalue problems, SIAM Review 42 (2000) 267.
[17] D.R. Fokkema, G.L.G. Sleijpen, H.A. van der Vorst, Jacobi–Davidson style QR and QZ algorithms for the reduction of matrix pencils, SIAM Journal on Scientific Computing 20 (1999) 94.
[18] K. Wu, Y. Saad, A. Stathopoulos, Inexact Newton preconditioning techniques for large symmetric eigenvalue problems, Electronic Transactions on Numerical Analysis 7 (1998) 202.
[19] S. Balay, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L.C. McInnes, B. Smith, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11, Revision 3.2, Argonne National Laboratory, 2011.
[20] S. Balay, W.D. Gropp, L.C. McInnes, B.F. Smith, Efficient management of parallelism in object oriented numerical software libraries, in: Modern Software Tools in Scientific Computing, Birkhäuser Press, 1997, p. 163.
[21] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd edition, SIAM, 2003.
[22] X.C. Cai, M. Sarkis, Restricted additive Schwarz preconditioner for general sparse linear systems, SIAM Journal on Scientific Computing 21 (2) (1999) 792.
[23] A. George, J.W.H. Liu, A fast implementation of the minimum degree algorithm using quotient graphs, ACM Transactions on Mathematical Software 6 (3) (1980) 337.
[24] Z. Li, Y. Saad, M. Sosonkina, pARMS: a parallel version of the algebraic recursive multilevel solver, Numerical Linear Algebra with Applications 10 (2003) 485.
[25] M. Sosonkina, Y. Saad, X. Cai, Using the parallel algebraic recursive multilevel solver in modern physical applications, Future Generation Computer Systems 20 (2004) 489.