Improvements to SLEPc in releases 3.14–3.18[*]

Jose E. Roman[†] Fernando Alvarruiz[‡] Carmen Campos[§] Lisandro Dalcin[¶]
Pierre Jolivet[‖] Alejandro Lamas Daviña[**]

September 29, 2023

Abstract

This short paper describes the main new features added to SLEPc, the Scalable Library for Eigenvalue Problem Computations, in the last two and a half years, corresponding to five release versions. The main novelty is the extension of the `SVD` module with new problem types, such as the generalized SVD or the hyperbolic SVD. Additionally, many improvements have been incorporated into different library parts, including contour integral eigensolvers, preconditioning, and GPU support.

1 Overview of SLEPc

SLEPc, the Scalable Library for Eigenvalue Problem Computations [12, 20], is a parallel library for solving large-scale eigenvalue problems and other related linear algebra problems. It is available for download at `https://slepc.upv.es` and `https://gitlab.com/slepc/slepc` under a 2-clause BSD license.

SLEPc is an extension of PETSc, the Portable, Extensible Toolkit for Scientific Computation [4]. It relies on its data classes, such as matrices and vectors, as well as its solvers for linear systems of equations, which are required in some eigensolvers. Figure 1 shows a summary of the functionality offered by PETSc and SLEPc.

SLEPc is often used to compute a few eigenvalues and eigenvectors of large-scale, sparse matrices or matrix pairs, usually with Krylov methods, but also with other methods, including Davidson-type [21], conjugate gradient-type (LOBPCG), or contour integral. SLEPc is unique in offering specific functionality for polynomial eigenproblems [5, 6] and general nonlinear eigenproblems [7]. SLEPc also provides some functionality related to matrix functions, both dense and sparse, which are occasionally needed by some eigensolvers. The main SLEPc classes are:

- `EPS`: linear eigenvalue problem solver.
- `PEP`: polynomial eigenvalue problem solver.
- `NEP`: general nonlinear eigenvalue problem solver.

[*]This work was supported by the Spanish Agencia Estatal de Investigación under grant PID2019-107379RB-I00 / AEI / 10.13039/501100011033.

[†]DSIC, Universitat Politècnica de València, València, Spain (`jroman@dsic.upv.es`).

[‡]DSIC, Universitat Politècnica de València, València, Spain (`fbermejo@dsic.upv.es`).

[§]D. Didáctica de la Matemática, Universitat de València, València, Spain (`carmen.campos-gonzalez@uv.es`).

[¶]King Abdullah University of Science and Technology, Extreme Computing Research Center, Thuwal, Saudi Arabia (`dalcinl@gmail.com`).

[‖]Sorbonne Université, CNRS, LIP6, 4 place Jussieu, Paris Cedex 05, 75252, France (`pierre.jolivet@cnrs.fr`).

[**]DSIC, Universitat Politècnica de València, València, Spain (`alejandro.lamas@dsic.upv.es`).

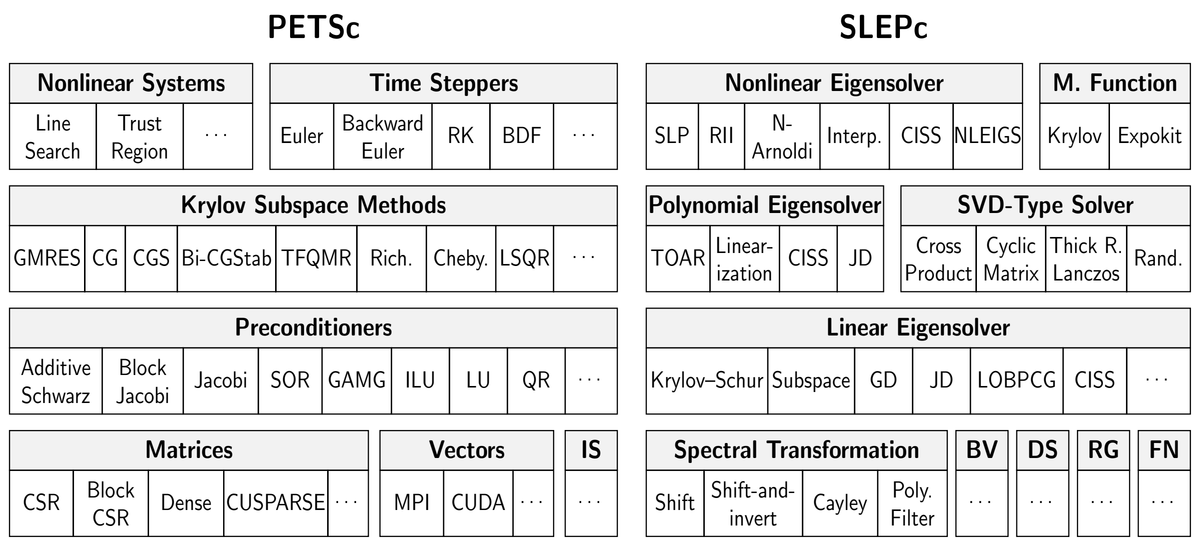

Figure 1: Main components of the PETSc (left) and SLEPc (right) libraries. Every class contains different sub-classes (types), each of them providing a particular implementation of a method.

- `SVD`: singular value decomposition and related problems.
- `MFN`: action of a matrix function on a vector.
- Auxiliary classes: `ST` for spectral transformations such as shift-and-invert; `BV` for management of basis vectors and their orthogonalization; `DS` for solving small-scale dense eigenproblems via LAPACK [2]; `RG` to let the user define a region in the complex plane; and `FN` to let the user define mathematical functions, allowing its evaluation on scalar values as well as on small dense matrices.

This paper provides an overview of the major changes in five SLEPc releases: 3.14 (Sept. 30, 2020), 3.15 (March 31, 2021), 3.16 (Sept. 30, 2021), 3.17 (March 31, 2022), 3.18 (Oct. 1, 2022). These are:

- New functionality for the `SVD` module, including a randomized solver for the SVD as well as support for two new problem types, the GSVD and the HSVD (see Section 2).
- Improvements of contour integral eigensolvers, named CISS in Fig. 1, for the linear, polynomial, and nonlinear eigenproblems (see Section 3).
- Improvements when linear solvers are used in combination with eigensolvers (see Section 4).
- Other miscellaneous changes related to Python, GPU, and external packages (see Section 5).

2 New functionality for SVD-related problems

The `SVD` module appeared early in the development of SLEPc, to compute the partial singular value decomposition (SVD) of a matrix A, corresponding to either the largest or smallest singular values. Essentially, it contained a couple of solvers based on Golub–Kahan–Lanczos bidiagonalization [13], together with two solvers based on formulating an equivalent eigenproblem with the cross product matrix A^*A (`SVDCROSS`) or the cyclic matrix $\left[\begin{smallmatrix} 0 & A \\ A^* & 0 \end{smallmatrix}\right]$ (`SVDCYCLIC`). We have extended this module with additional functionality, such as a new solver (Section 2.1).

In addition to the standard SVD, two new problem types that were not considered before have been introduced: the GSVD and the HSVD (see Sections 2.2 and 2.3). This has required changes in the user interface: now `SVDSetOperators()` must be used instead of `SVDSetOperator()`; a new function `SVDSetProblemType()` has been added to specify the problem type; and, in the case of the HSVD, the signature matrix can be established with `SVDSetSignature()`.

2.1 Randomized solver

The randomized SVD method [11] is used for approximating the singular values and vectors of low-rank matrices. An initial implementation of this method was added in SLEPc 3.15 with the name `SVDRANDOMIZED`. This solver works as follows. Given a matrix $A \in \mathbb{R}^{m \times n}$, the following operations [11] compute a rank-k approximation A_k of A,

$$Y = A\Omega, \tag{1}$$

$$B = Q^T A, \quad \text{with } Q = \operatorname{orth}(Y), \tag{2}$$

$$A_k = QB_k, \tag{3}$$

where $\Omega \in \mathbb{R}^{m \times (k+p)}$ is a Gaussian random matrix, and B_k is the rank-k truncated SVD of B. Note that we employ an oversampling parameter p that can be, e.g., equal to k. Suppose that we knew the exact rank of A and set $k = \operatorname{rank}(A)$. Then the rank-$k$ approximation is exact (with high probability), and the truncated SVD of A can be obtained easily from the computed quantities.

In the general case, the value of $\operatorname{rank}(A)$ is not known a priori. If we choose a value such that $k < \operatorname{rank}(A)$, then [11] suggest repeating the steps of Eqs. (1) and (2) a couple of times to obtain a sufficiently accurate rank-k approximation of A. However, in this case, the individual singular vectors of A will not be determined unless we continue iterating until each residual is below the given tolerance. This subspace iteration scheme is described in [10], which is what is implemented in the `SVDRANDOMIZED` solver.

To use the SLEPc solver, set `nsv`=k and `ncv`=$k+p$. With the special convergence criterion `SVD_CONV_MAXIT`, the solver will stop after the given maximum iterations are performed, which is enough to obtain the subspaces for the rank-k approximation, while the usual stopping criterion will provide singular vectors to the requested accuracy.

The algorithm's adaptive variant [11] works with a prescribed tolerance and expands the dimension of the subspace until the low-rank approximation satisfies the tolerance. This is currently not supported in SLEPc, but may be added in the future.

2.2 The generalized SVD

The generalized singular value decomposition (GSVD) is an extension of the SVD to the case of two matrices (cf. [9, §8.7.3] or [2, §2.3.5.3]), which appears in applications such as constrained least-squares problems or regularized discrete ill-posed problems.

Given two matrices with the same column dimension, $A \in \mathbb{C}^{m \times n}$ and $B \in \mathbb{C}^{p \times n}$, there exist two unitary matrices $U \in \mathbb{C}^{m \times m}$ and $V \in \mathbb{C}^{p \times p}$ and an invertible matrix $X \in \mathbb{C}^{n \times n}$ such that

$$U^* AX = C, \qquad V^* BX = S. \tag{4}$$

Suppose that $m \geq n$, then $C = \operatorname{diag}(c_1, \ldots, c_n)$ and $S = \operatorname{diag}(s_{n-q+1}, \ldots, s_n)$ with $q = \min(p, n)$, where c_i and s_i are real nonnegative values satisfying $c_i^2 + s_i^2 = 1$, and the generalized singular values are given by the ratios $\sigma(A, B) \equiv \{c_1/s_1, \ldots, c_q/s_q\}$. If $p < n$, the first $n - p$ generalized singular values are considered infinite, as if $s_1 = \cdots = s_{n-p} = 0$, and the matrix S must be padded with zeros on the left, as shown in Fig. 2. The decomposition represented in the figure

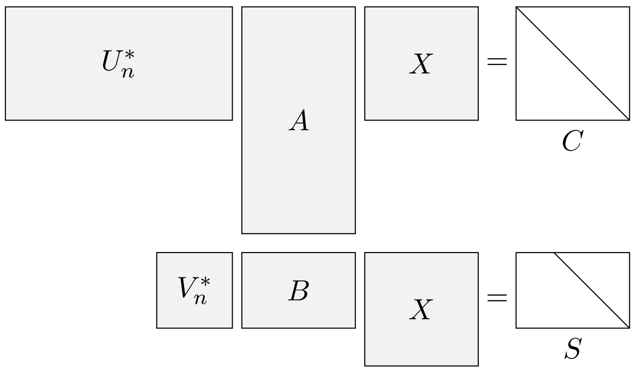

Figure 2: Scheme of the thin GSVD of two matrices A and B, for the case $m > n$ and $p < n$.

is the thin GSVD, where U_n denotes the first n columns of U. Similarly, the decomposition can also be defined for $m < n$, in which case we have $n - m$ zero generalized singular values, and matrix C must be padded with zeros on the right accordingly. More precisely, it is rank(A) and rank(B), rather than m and p, which determine the number of zero and infinite generalized singular values, respectively. The above description assumes that the pair $\{A, B\}$ is regular, that is, that the matrix obtained by stacking A and B has full column rank.

The values c_i and s_i are related to the CS decomposition [9, §2.6.4] associated with the orthogonal factor of the QR factorization of the stacked A and B matrices, i.e., if

$$Z := \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} Q_A \\ Q_B \end{bmatrix} R, \tag{5}$$

then c_i and s_i are the singular values of Q_A and Q_B, respectively. The matrix Z is relevant for some algorithms; Lanczos methods, in particular, usually build it explicitly, see Section 2.2.2.

SLEPc solvers compute a partial GSVD, i.e., only a few columns (`nsv`) of U, V, and X are obtained, corresponding to either the largest or smallest generalized singular values. Regarding the user interface, matrices A and B are specified with `SVDSetOperators()`, and the computed solutions are retrieved via `SVDGetSingularTriplet()`. The latter function returns only two vectors, for compatibility with the previously defined user interface. For the GSVD, the right generalized singular vectors x_i are returned in one of them, while the left vectors (the corresponding columns of U and V) are returned stacked on top of each other as a single $(m+p)$-vector $\begin{bmatrix} u_i^* & v_i^* \end{bmatrix}^*$.

2.2.1 GSVD via an equivalent eigenproblem

It is possible to formulate the problem of computing generalized singular values and vectors of a matrix pair $\{A, B\}$ as a generalized eigenvalue problem involving two Hermitian matrices related to A and B. This approach is analogous to what was done for the standard SVD in the `SVDCROSS` and `SVDCYCLIC` solvers, so we have extended those solvers to support the GSVD case. The advantage of this approach is the possibility of using most of the functionality offered by SLEPc's `EPS` module to solve the associated eigenvalue problem.

From the ith column of the relations of Eq. (4), we can see that vectors x_i satisfy

$$s_i^2 A^* A x_i = c_i^2 B^* B x_i, \tag{6}$$

i.e., we have $A^*Ax_i = \sigma_i^2 B^*Bx_i$, a generalized eigenvalue problem for the pencil (A^*A, B^*B). Once x_i is computed, vectors u_i and v_i are obtained trivially by a simple product with A or B.

This approach is followed in SVDCROSS. In the case of SVDCYCLIC, the formulation is a generalized eigenvalue problem involving two symmetric matrices of order $m+n$ or $p+n$ [1].

In both schemes, the eigenvalue problem is symmetric-definite; hence it is possible to use eigensolvers that exploit symmetry. For Lanczos-type methods, one typically has to (implicitly) invert the matrix on the right side of the eigenvalue equation, which in this case is either A^*A or B^*B, possibly concatenated with an identity block as in SVDCYCLIC. The user interface provides an option for explicitly computing the matrices, e.g., SVDCrossSetExplicitMatrix(), to enable more flexibility in the solution of the associated linear systems (e.g., Cholesky factorization of B^*B). Otherwise, the matrices are kept in implicit form (*shell* matrices in PETSc's terminology), which restricts the linear solvers and preconditioners that can be used.

The above schemes are implemented via object composition: an SVD object of type SVDCROSS contains an EPS object, which in turn holds an ST object internally and has a KSP object with a PC object inside. This structure, combined with the use of command-line options, makes it easy for the user to experiment with many different solver combinations, such as

```
$ ./ex -svd_nsv 4 -svd_smallest -svd_type cross -svd_cross_explicitmatrix
       -svd_cross_eps_type lobpcg -svd_cross_st_ksp_type cg -svd_cross_st_pc_type icc
```

2.2.2 Thick restart Lanczos solver

We have also developed a solver based on thick-restarted joint Lanczos bidiagonalization. The details of the method can be found in [1], and here we give a summary. The new method has been implemented in the SVDTRLANCZOS solver as an extension of the previously existing code that supported the standard SVD only.

The joint Lanczos bidiagonalization procedure can be seen as a way to simultaneously bidiagonalize matrices Q_A and Q_B from Eq. (5). This produces two bidiagonal matrices that can be taken together as a small-sized GSVD problem resulting from the projection of the original problem onto certain Krylov subspaces. In practice, matrices Q_A and Q_B cannot be used since they are not explicitly available, but a trick can be applied to replace them by the action of QQ^* on a vector. This operation can be implemented as a linear least-squares solve with coefficient matrix Z from Eq. (5) within the Lanczos iteration. This least-squares problem is handled by PETSc's KSP/PC classes (with either a direct method, such as a sparse QR factorization, or an iterative method, such as LSQR).

Apart from the basic joint bidiagonalization, we have incorporated several new ingredients:

- Thick restart, allowing to address problems with slow convergence. When the maximum subspace dimension (ncv) is reached, the subspaces are compressed to a smaller dimension.
- Scale factor, which can often boost convergence in difficult problems.
- One-sided orthogonalization, to save some computation during the orthogonalization of the various Krylov bases.

The new method's implementation also involved adding a new DS type, DSGSVD, for the generalized SVD of small dense matrices via a call to LAPACK's subroutine _ggsvd3.

2.3 The hyperbolic SVD

The hyperbolic singular value decomposition (HSVD) is a variation of the standard SVD that appears in such applications as the covariance differencing problem in signal processing [19]. The difference with the SVD is that U is orthogonal with respect to a signature,

$$A = U\Sigma V^*, \qquad U^*\Omega U = \tilde{\Omega}, \tag{7}$$

where $\Omega = \mathrm{diag}(\pm 1)$ is a user-provided $m \times m$ signature matrix, while $\tilde{\Omega}$ is another signature matrix obtained as part of the solution.

As in the case of the SVD, the problem's solution consists of singular triplets (σ_i, u_i, v_i), with σ_i real and nonnegative and sorted in nonincreasing order. With each singular triplet, there is an associated sign $\tilde{\omega}_i$ (either 1 or -1), the corresponding diagonal element of $\tilde{\Omega}$.

The relations between left and right singular vectors differ slightly from those of the standard SVD. We have $AV = U\Sigma$ and $A^*\Omega U = V\Sigma^*\tilde{\Omega}$, so for $m \geq n$

$$Av_i = u_i\sigma_i \ , \quad i = 1, \dots, n, \tag{8}$$

$$A^*\Omega u_i = v_i\sigma_i\tilde{\omega}_i \ , \quad i = 1, \dots, n. \tag{9}$$

SLEPc computes a partial HSVD of either the largest or smallest hyperbolic singular triplets.

2.3.1 HSVD via an equivalent eigenproblem

As in the SVD and GSVD, it is possible to formulate cross and cyclic schemes to compute the HSVD by solving an eigenvalue problem in the `SVDCROSS` and `SVDCYCLIC` solvers. The cross product matrix approach has two forms, one more convenient when $m \geq n$ and the other when $m < n$. The first form

$$A^*\Omega A v_i = \sigma_i^2 \tilde{\omega}_i v_i, \tag{10}$$

is derived by pre-multiplying Eq. (8) by $A^*\Omega$ and then using Eq. (9). This eigenproblem can be solved as a Hermitian eigenvalue problem (`EPS_HEP`) and may have both positive and negative eigenvalues, corresponding to $\tilde{\omega}_i = 1$ and $\tilde{\omega}_i = -1$, respectively. Once the right vector v_i has been computed, the corresponding left vector can be obtained using Eq. (8) as $u_i = \sigma_i^{-1}Av_i$. The second form computes left vectors first, multiplying Eq. (9) by A and then using Eq. (8),

$$AA^*\Omega u_i = \sigma_i^2 \tilde{\omega}_i u_i. \tag{11}$$

Then, the right singular vectors are obtained as $v_i = (\sigma_i\tilde{\omega}_i)^{-1}A^*\Omega u_i$. The coefficient matrix of Eq. (11) is non-Hermitian, so the eigenproblem should be solved as non-Hermitian (`EPS_NHEP`). Instead, we currently solve it as a generalized Hermitian-indefinite eigenvalue problem (`EPS_GHIEP`)

$$AA^*\hat{u}_i = \sigma_i^2\tilde{\omega}_i\Omega\hat{u}_i, \qquad \text{with } \hat{u}_i = \Omega u_i. \tag{12}$$

The cyclic matrix approach for the HSVD implies a generalized eigenvalue problem defined by two symmetric matrices of order $m + n$.

3 Contour integral solvers

Contour integral methods compute eigenvalues located inside a closed path in the complex plane, $\mathcal{C}$, by employing an approximation of the spectral projector to the corresponding eigenspace. In a subspace iteration scheme, the projector, given by Cauchy's integral formula, multiplies a block of vectors V. This operation can be approximated with a quadrature rule,

$$\frac{1}{2\pi \mathrm{i}} \oint_{\mathcal{C}} (zI - A)^{-1} V \mathrm{d}z \approx \sum_{j=0}^{n_c} \omega_j (z_j I - A)^{-1} V, \tag{13}$$

where z_j and ω_j are the quadrature points and the corresponding weights, $j = 0, 1, \dots, n_c$. A more general scheme uses high-order moments, i.e., the integrand of Eq. (13) is multiplied by z^k, $k = 0, 1, \dots, n_m$, and the results of the n_m moments are combined. SLEPc currently implements the latter case, and the solvers are called CISS. This methodology can be applied to both linear and nonlinear eigenvalue problems. Contour integral solvers are typically more expensive than other classes of solvers. However, they are usually better in scalability terms, provided that this aspect is kept in mind in the implementation.

The contour integral solvers for `EPS` and `NEP` were originally developed and contributed by Y. Maeda and T. Sakurai [18]. In recent versions, we have incorporated many improvements, in particular:

- The `NEPCISS` solver has been reworked completely. In addition to the original variant based on block Hankel matrices, it now provides two new variants that can be selected with `NEPCISSSetExtraction()`: one using a Rayleigh–Ritz projection and another one based on the CAA technique [22, 15]. In the Rayleigh–Ritz variant, the projected problem is also a nonlinear eigenproblem, which is solved with a reimplemented `DSNEP` that performs a dense-matrix contour integral eigensolution, combined with Newton iterative refinement. This approach improves the robustness of the overall solver considerably.
- The new `NEPCISS` solver also supports subcommunicators so that different subsets of integration points are processed by independent groups of MPI processes, improving the overall scalability. Subcommunicators were used in the `EPS` solver but not in the `NEP` one.
- A contour integral solver has also been added to the polynomial eigensolver class: `PEPCISS`. The implementation is very similar to the one in `NEP` but exploits the structure of the projected problem (a polynomial eigenproblem in the case of the Rayleigh–Ritz, `DSPEP`).
- All code has been refactored to minimize code duplication in the three contour integral solvers in `EPS`, `PEP`, and `NEP`.

4 Improvements to linear solvers

Many eigensolvers in SLEPc must be combined with a `KSP` object that solves the required linear systems. In `EPS` and `PEP`, this object is contained in the spectral transformation object `ST`, while in `NEP`, it is associated directly with some of the solvers.

The solution of linear systems must often be done with direct methods, typically obtained via external packages such as MUMPS. We have improved how such packages are integrated into SLEPc. For instance, the new function `EPSKrylovSchurGetKSP()` allows extracting the `KSP` object used in spectrum slicing runs so that MUMPS options can be prescribed in the code, even in multi-communicator configurations. This was not possible before. It is now also easier to set command-line options applicable to the linear solver, as the associated `Mat` object has the same prefix as the `KSP`, e.g. `-st_mat_mumps_icntl_13 1`.

4.1 Preconditioners in split form

We have added the possibility of using custom preconditioners in split form, which means that the user passes a set of matrices from which the preconditioner must be computed. For instance, in shift-and-invert, the preconditioner must be an approximation of the matrix $A-\sigma B$, so if A_0 and B_0 are given as approximations of A and B, then the preconditioner $M^{-1} \approx (A_0-\sigma B_0)^{-1}$ can be built internally. This procedure is better than passing an approximation of $A-\sigma B$ since it is valid for any value of σ (which may be changed internally by the solver). Similarly, for polynomial eigenproblems, the user can pass approximations for all coefficient matrices with the function `STSetSplitPreconditioner()`. In the case of nonlinear eigenproblems where the operator is represented in split form, we have added an analog function `NEPSetSplitPreconditioner()`.

```
import sys, slepc4py
slepc4py.init(sys.argv)
from petsc4py import PETSc
from slepc4py import SLEPc

.
.  # build matrix A
.

evals = solve_eigen(A)
```

```
def solve_eigen(A,nev=4):
  E = SLEPc.EPS(); E.create()
  E.setOperators(A)
  E.setProblemType(SLEPc.EPS.ProblemType.HEP)
  E.setDimensions(nev)
  E.setFromOptions()
  E.solve()
  evals = []
  nconv = E.getConverged()
  for i in range(nconv):
    evals.append(E.getEigenpair(i))
  return evals
```

Figure 3: Simple example illustrating the use of SLEPc Python bindings.

The addition of the split preconditioners was motivated by an application [14] where the problem matrices are not assembled explicitly, but a first-order approximation can be formed and used to build the preconditioner.

4.2 Support for block linear solves

In the case of eigensolvers where the operator is applied to a set of vectors simultaneously, it is much better to also apply the preconditioner or linear solver in a block fashion; otherwise, the overall efficiency is seriously spoiled. For this, we have implemented operations such as `STMatMatSolve()` to attain higher arithmetic intensity, especially in parallel. Solvers such as `EPSLOBPCG`, `EPSCISS`, or `EPSSUBSPACE` can now operate entirely by blocks, provided that the linear solver and preconditioner also support this, such as `KSPHPDDM` and `PCHPDDM` [16] or `PCMG`.

5 Miscellaneous changes

5.1 Python interface

The slepc4py Python bindings for SLEPc used to be a separate project, but now it has been integrated into the SLEPc source tree and repository, in the same way as petsc4py [8] has been included in PETSc. Now the `configure` script has a `--with-slepc4py` option to switch on the compilation of the Python bindings.

Python bindings are crucial to expand the user base of SLEPc to Python-based projects. slepc4py is included in popular package management systems such as Conda or PyPI. It can also be used via third-party software such as Firedrake or FEniCS.

Figure 3 lists a minimal example of slepc4py's basic usage.

5.2 Configuration script and external packages

SLEPc's `configure` script is much simpler than PETSc's counterpart because we inherit from it most of the configuration settings, such as compilers, MPI, and the like. Still, we continue improving it over the years for a smooth compilation on different systems and optimal integration with PETSc and other software. One of the changes we introduced in version 3.18 is that `configure` arguments intended to provide linker flags (and libraries) now must be specified via a quoted string instead of a comma-separated list. This is a better approach to install SLEPc from source-based package managers and is consistent with PETSc's `configure`.

One of the main roles of `configure` is to enable external packages when building SLEPc. Since the early days, SLEPc has allowed using packages such as ARPACK, PRIMME, or TRLAN in the same way as native SLEPc solvers. In recent versions, we have made the following changes:

- New interfaces to dense eigen- and SVD solvers from ScaLAPACK, Elemental, and ELPA [3] (note that configuration with ScaLAPACK and Elemental is handled by PETSc's `configure`). SLEPc is intended primarily for sparse matrices, but integrating these external libraries allows users to compute the whole spectrum of dense (or sparse) matrices using the same programming interface.
- New interface in `EPS` to the external package EVSL [17], which can be used for computing all eigenvalues in an interval of symmetric eigenproblems using polynomial filters. Similar functionality is also available natively in SLEPc via `STFILTER`.
- The interface to FEAST has been reintroduced, but now it is supported via Intel MKL only.
- The interface to BLZPACK has been removed since it is no longer available for download.

5.3 GPU support

Following the improved support for GPU computing in PETSc, we have also enhanced this aspect in SLEPc. In particular, some of the code that operates with dense matrices has been adapted to support GPU matrices of type `MATSEQDENSECUDA`. These matrices store the matrix entries in both the CPU and GPU memory, synchronizating only when required so that computation can be carried out on the GPU with minimal overhead.

- Dense matrix functions. The functionality in the `FN` class related to matrix functions, i.e., `FNEvaluateFunctionMat()` and `FNEvaluateFunctionMatVec()`, can now take arguments of type `MATSEQDENSECUDA`. In this way, the computation can be done on the GPU. This change required writing GPU implementations of methods such as scaling and squaring with Padé approximant for the matrix exponential or the Denman–Beavers method for the matrix square root, among others.
- Projected problems. The `DS` class does not have GPU support yet, but it has been reworked to always use the `Mat` interface rather than raw pointers. This is the first step towards eventually enabling GPU support in future versions, which will require replacing calls to LAPACK with the analog ones from MAGMA.

6 Conclusion

This paper summarizes the new functionality that has been added to SLEPc in recent versions, which often enables its applicability to situations where it could not be used before. Apart from the new features, SLEPc has also matured in the last years in other aspects such as more professional development (better software engineering practices, continuous integration using pipelines, exhaustive test battery with code coverage of more than 90%, comprehensive documentation, etc.) and community building (collaboration with other groups, interaction with users via issue notification and contribution of patches via merge requests).

Future work We are currently working on a thick-restart Lanczos method for the HSVD, which will complement the functionality described in Section 2.3. A possible future addition would be a new problem type for the generalized hyperbolic singular value decomposition (GHSVD), a mixture of the GSVD and HSVD problems. Other variations of the SVD might be considered as well. There are many other potential lines for future development, such as block eigensolvers, structured eigensolvers, solvers for the nonlinear eigenvector problem, improved support for GPU computing, etc. As always, the priority of the different lines can be biased by user requests.

Citing SLEPc In order to justify the work invested in SLEPc's development, we request that users cite at least one of the SLEPc papers in their publications. The most common way is by citing the main SLEPc article [12] or the user's guide [20]. Please reference the present document to acknowledge the use of one of the versions in the range 3.14–3.18. If you rely on specific library features, please consider citing any of the papers listed on the website. Alternatively, any SLEPc-based application code can be run with the option `-citations slepc.bib` (in addition to all the options required for normal execution), which will generate a list of BibTEX references appropriate for that computation.

Acknowledgments In addition to the authors of this paper, the following people contributed code to the releases: Fande Kong, Barry Smith, Satish Balay, Matthew Knepley, Stefano Zampini, Jacob Faibussowitsch, Murat Keçeli, William Dawn, Marco Morandini, Alexei Colin, Jan Blechta, Jack Hale, Elisa Schenone. Their contributions are much appreciated!

References

[1] F. Alvarruiz, C. Campos, and J. E. Roman. Thick-restarted joint Lanczos bidiagonalization for the GSVD. *arXiv:2206.03768 : retrieved 9 Jun 2022*, 2022.

[2] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[3] T. Auckenthaler, H.-J. Bungartz, T. Huckle, L. Krämer, B. Lang, and P. Willems. Developing algorithms and software for the parallel solution of the symmetric eigenvalue problem. *J. Comput. Sci.*, 2(3):272–278, 2011.

[4] S. Balay, S. Abhyankar, M. F. Adams, J. Brown S. Benson, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch, W. D. Gropp, V. Hapla, T. Isaac, P. Jolivet, D. Karpeev, D. Kaushik, M. G. Knepley, F. Kong, S. Kruger, D. A. May, L. Curfman McInnes, R. Tran Mills, L. Mitchell, T. Munson, J. E. Roman, K. Rupp, P. Sanan, J. Sarich, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, and J. Zhang. *PETSc/TAO Users Manual Revision 3.18*, 2022. Argonne Technical Memorandum ANL-21/39.

[5] C. Campos and J. E. Roman. Parallel Krylov solvers for the polynomial eigenvalue problem in SLEPc. *SIAM J. Sci. Comput.*, 38(5):S385–S411, 2016.

[6] C. Campos and J. E. Roman. A polynomial Jacobi-Davidson solver with support for non-monomial bases and deflation. *BIT*, 60(2):295–318, 2020.

[7] C. Campos and J. E. Roman. NEP: a module for the parallel solution of nonlinear eigenvalue problems in SLEPc. *ACM Trans. Math. Software*, 47(3):23:1–23:29, 2021.

[8] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo. Parallel distributed computing using Python. *Adv. Water Resour.*, 34(9):1124–1139, 2011.

[9] G. H. Golub and C. F. van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[10] M. Gu. Subspace iteration randomization and singular value problems. *SIAM J. Sci. Comput.*, 37(3):A1139–A1173, 2015.

[11] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, 2011.

[12] V. Hernandez, J. E. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005.

[13] Vicente Hernandez, Jose E. Roman, and Andres Tomas. A robust and efficient parallel SVD solver based on restarted Lanczos bidiagonalization. *Electron. Trans. Numer. Anal.*, 31:68–85, 2008.

[14] V. Hiremath and J. E. Roman. Acoustic modal analysis with heat release fluctuations using nonlinear eigensolvers. *arXiv:2208.08717 : retrieved 23 Sep 2022*, 2022.

[15] A. Imakura and T. Sakurai. Block SS–CAA: A complex moment-based parallel nonlinear eigensolver using the block communication-avoiding arnoldi procedure. *Parallel Comput.*, 74:34–48, 2018.

[16] P. Jolivet, J. E. Roman, and S. Zampini. KSPHPDDM and PCHPDDM: Extending PETSc with advanced Krylov methods and robust multilevel overlapping Schwarz preconditioners. *Comput. Math. Appl.*, 84:277–295, 2021.

[17] R. Li, Y. Xi, L. Erlandson, and Y. Saad. The eigenvalues slicing library (EVSL): algorithms, implementation, and software. *SIAM J. Sci. Comput.*, 41(4):C393–C415, 2019.

[18] Y. Maeda, T. Sakurai, and J. E. Roman. Contour integral spectrum slicing method in SLEPc. Technical Report STR-11, Universitat Politècnica de València, 2016. Available at `https://slepc.upv.es`.

[19] R. Onn, A. O. Steinhardt, and A. Bojanczyk. The hyperbolic singular value decomposition and applications. *IEEE Trans. Signal Proces.*, 39(7):1575–1588, 1991.

[20] J. E. Roman, C. Campos, L. Dalcin, E. Romero, and A. Tomas. SLEPc users manual. Technical Report DSIC-II/24/02–Revision 3.18, D. Sistemes Informàtics i Computació, Universitat Politècnica de València, 2022.

[21] E. Romero and J. E. Roman. A parallel implementation of Davidson methods for large-scale eigenvalue problems in SLEPc. *ACM Trans. Math. Software*, 40(2):13:1–13:29, 2014.

[22] S. Yokota and T. Sakurai. A projection method for nonlinear eigenvalue problems using contour integrals. *JSIAM Letters*, 5:41–44, 2013.