
SLEPc for Python

Release 3.24.1

Lisandro Dalcin

Dec 12, 2025

Contents

1	Contents	1
1.1	Overview	1
1.2	Tutorial	4
1.3	Installation	8
1.4	Citations	10
1.5	Reference	10
1.6	slepc4py demos	284
	Python Module Index	288
	Index	289

Abstract

This document describes `slepc4py`, a `Python` port to the `SLEPc` libraries.

`SLEPc` is a software package for the parallel solution of large-scale eigenvalue problems. It can be used for computing eigenvalues and eigenvectors of large, sparse matrices, or matrix pairs, and also for computing singular values and vectors of a rectangular matrix.

`SLEPc` relies on `PETSc` for basic functionality such as the representation of matrices and vectors, and the solution of linear systems of equations. Thus, `slepc4py` must be used together with its companion `petsc4py`.

1 Contents

1.1 Overview

SLEPc for Python (`slepc4py`) is a `Python` package that provides convenient access to the functionality of `SLEPc`.

`SLEPc`^{1, 2} implements algorithms and tools for the numerical solution of large, sparse eigenvalue problems on parallel

¹ J. E. Roman, C. Campos, L. Dalcin, E. Romero, A. Tomas. *SLEPc Users Manual*. DSIC-II/24/02 - Revision 3.24. D. Sistemas Informaticos y Computacion, Universitat Politecnica de Valencia. 2025.

² Vicente Hernandez, Jose E. Roman and Vicente Vidal. *SLEPc: A Scalable and Flexible Toolkit for the Solution of Eigenvalue Problems*, ACM Trans. Math. Softw. 31(3), pp. 351-362, 2005.

computers. It can be used for linear eigenvalue problems in either standard or generalized form, with real or complex arithmetic. It can also be used for computing a partial SVD of a large, sparse, rectangular matrix, and to solve nonlinear eigenvalue problems (polynomial or general). Additionally, SLEPc provides solvers for the computation of the action of a matrix function on a vector.

SLEPc is intended for computing a subset of the spectrum of a matrix (or matrix pair). One can for instance approximate the largest magnitude eigenvalues, or the smallest ones, or even those eigenvalues located near a given region of the complex plane. Interior eigenvalues are harder to compute, so SLEPc provides different methodologies. One such method is to use a spectral transformation. Cheaper alternatives are also available.

Features

Currently, the following types of eigenproblems can be addressed:

- Standard eigenvalue problem, $Ax=kx$, either for Hermitian or non-Hermitian matrices.
- Generalized eigenvalue problem, $Ax=kBx$, either Hermitian positive-definite or not.
- Partial singular value decomposition of a rectangular matrix, $Au=sv$.
- Polynomial eigenvalue problem, $P(k)x=0$.
- Nonlinear eigenvalue problem, $T(k)x=0$.
- Computing the action of a matrix function on a vector, $w=f(\alpha A)v$.

For the linear eigenvalue problem, the following methods are available:

- Krylov eigensolvers, particularly Krylov-Schur, Arnoldi, and Lanczos.
- Davidson-type eigensolvers, including Generalized Davidson and Jacobi-Davidson.
- Subspace iteration and single vector iterations (inverse iteration, RQI).
- Conjugate gradient for the minimization of the Rayleigh quotient.
- A contour integral solver.

For singular value computations, the following alternatives can be used:

- Use an eigensolver via the cross-product matrix $A'A$ or the cyclic matrix $[0 \ A; A' \ 0]$.
- Explicitly restarted Lanczos bidiagonalization.
- Implicitly restarted Lanczos bidiagonalization (thick-restart Lanczos).

For polynomial eigenvalue problems, the following methods are available:

- Use an eigensolver to solve the generalized eigenvalue problem obtained after linearization.
- TOAR and Q-Arnoldi, memory efficient variants of Arnoldi for polynomial eigenproblems.

For general nonlinear eigenvalue problems, the following methods can be used:

- Solve a polynomial eigenproblem obtained via polynomial interpolation.
- Rational interpolation and linearization (NLEIGS).
- Newton-type methods such as SLP or RII.

Computation of interior eigenvalues is supported by means of the following methodologies:

- Spectral transformations, such as shift-and-invert. This technique implicitly uses the inverse of the shifted matrix $(A-tI)$ in order to compute eigenvalues closest to a given target value, t .
- Harmonic extraction, a cheap alternative to shift-and-invert that also tries to approximate eigenvalues closest to a target, t , but without requiring a matrix inversion.

Other remarkable features include:

- High computational efficiency, by using NumPy and SLEPc under the hood.
- Data-structure neutral implementation, by using efficient sparse matrix storage provided by PETSc. Implicit matrix representation is also available by providing basic operations such as matrix-vector products as user-defined Python functions.
- Run-time flexibility, by specifying numerous setting at the command line.
- Ability to do the computation in parallel.

Components

SLEPc provides the following components, which are mirrored by slepc4py for its use from Python. The first five components are solvers for different classes of problems, while the rest can be considered auxiliary object.

EPS

The Eigenvalue Problem Solver is the component that provides all the functionality necessary to define and solve an eigenproblem. It provides mechanisms for completely specifying the problem: the problem type (e.g. standard symmetric), number of eigenvalues to compute, part of the spectrum of interest. Once the problem has been defined, a collection of solvers can be used to compute the required solutions. The behavior of the solvers can be tuned by means of a few parameters, such as the maximum dimension of the subspace to be used during the computation.

SVD

This component is the analog of EPS for the case of Singular Value Decompositions. The user provides a rectangular matrix and specifies how many singular values and vectors are to be computed, whether the largest or smallest ones, as well as some other parameters for fine tuning the computation. Different solvers are available, as in the case of EPS.

PEP

This component is the analog of EPS for the case of Polynomial Eigenvalue Problems. The user provides the coefficient matrices of the polynomial. Several parameters can be specified, as in the case of EPS. It is also possible to indicate whether the problem belongs to a special type, e.g., symmetric or gyroscopic.

NEP

This component covers the case of general nonlinear eigenproblems, $T(\lambda)x=0$. The user provides the parameter-dependent matrix T via the split form or by means of callback functions.

MFN

This component provides the functionality for computing the action of a matrix function on a vector. Given a matrix A and a vector b , the call `MFNSolve(mfn,b,x)` computes $x=f(A)b$, where f is a function such as the exponential.

LME

This component provides the functionality for solving linear matrix equations such as Lyapunov or Sylvester where the solution has low rank.

ST

The Spectral Transformation is a component that provides convenient implementations of common spectral transformations. These are simple transformations that map eigenvalues to different positions, in such a way that convergence to wanted eigenvalues is enhanced. The most common spectral transformation is shift-and-invert, that allows for the computation of eigenvalues closest to a given target value.

BV

This component encapsulates the concept of a set of Basis Vectors spanning a vector space. This

component provides convenient access to common operations such as orthogonalization of vectors. The BV component is usually not required by end-users.

DS

The Dense System (or Direct Solver) component, used internally to solve dense eigenproblems of small size that appear in the course of iterative eigensolvers.

FN

A component used to define mathematical functions. This is required by the end-user for instance to define function $T(\cdot)$ when solving nonlinear eigenproblems with NEP in split form.

RG

A component used to define a region of the complex plane such as an ellipse or a rectangle. This is required by end-users in some cases such as contour-integral eigensolvers.

In addition to the above components, some extra functionality is provided in the :Sys: and :Util: sections.

1.2 Tutorial

This tutorial is intended for basic use of slepc4py. For more advanced use, the reader is referred to SLEPc tutorials as well as to slepc4py reference documentation.

Commented source of a simple example

In this section, we include the source code of example `demo/ex1.py` available in the slepc4py distribution, with comments inserted inline.

The first thing to do is initialize the libraries. This is normally not required, as it is done automatically at import time. However, if you want to gain access to the facilities for accessing command-line options, the following lines must be executed by the main script prior to any `petsc4py` or `slepc4py` calls:

```
import sys, slepc4py
slepc4py.init(sys.argv)
```

Next, we have to import the relevant modules. Normally, both PETSc and SLEPc modules have to be imported in all slepc4py programs. It may be useful to import NumPy as well:

```
from petsc4py import PETSc
from slepc4py import SLEPc
import numpy
```

At this point, we can use any `petsc4py` and `slepc4py` operations. For instance, the following lines allow the user to specify an integer command-line argument `n` with a default value of 30 (see the next section for example usage of command-line options):

```
opts = PETSc.Options()
n = opts.getInt('n', 30)
```

It is necessary to build a matrix to define an eigenproblem (or two in the case of generalized eigenproblems). The following fragment of code creates the matrix object and then fills the non-zero elements one by one. The matrix of this particular example is tridiagonal, with value 2 in the diagonal, and -1 in off-diagonal positions. See `petsc4py` documentation for details about matrix objects:

```
A = PETSc.Mat().create()
A.setSizes([n, n])
A.setFromOptions()
A.setUp()
```

(continues on next page)

(continued from previous page)

```
rstart, rend = A.getOwnershipRange()

# first row
if rstart == 0:
    A[0, :2] = [2, -1]
    rstart += 1
# last row
if rend == n:
    A[n-1, -2:] = [-1, 2]
    rend -= 1
# other rows
for i in range(rstart, rend):
    A[i, i-1:i+2] = [-1, 2, -1]

A.assemble()
```

The solver object is created in a similar way as other objects in `petsc4py`:

```
E = SLEPc.EPS(); E.create()
```

Once the object is created, the eigenvalue problem must be specified. At least one matrix must be provided. The problem type must be indicated as well, in this case it is HEP (Hermitian eigenvalue problem). Apart from these, other settings could be provided here (for instance, the tolerance for the computation). After all options have been set, the user should call the `setFromOptions()` operation, so that any options specified at run time in the command line are passed to the solver object:

```
E.setOperators(A)
E.setProblemType(SLEPc.EPS.ProblemType.HEP)
E.setFromOptions()
```

After that, the `solve()` method will run the selected eigensolver, keeping the solution stored internally:

```
E.solve()
```

Once the computation has finished, we are ready to print the results. First, some informative data can be retrieved from the solver object:

```
Print = PETSc.Sys.Print

Print()
Print("*****")
Print("*** SLEPc Solution Results ***")
Print("*****")
Print()

its = E.getIterationNumber()
Print("Number of iterations of the method: %d" % its)

eps_type = E.getType()
Print("Solution method: %s" % eps_type)

nev, ncv, mpd = E.getDimensions()
```

(continues on next page)

(continued from previous page)

```
Print("Number of requested eigenvalues: %d" % nev)

tol, maxit = E.getTolerances()
Print("Stopping condition: tol=%.4g, maxit=%d" % (tol, maxit))
```

For retrieving the solution, it is necessary to find out how many eigenpairs have converged to the requested precision:

```
nconv = E.getConverged()
Print("Number of converged eigenpairs %d" % nconv)
```

For each of the `nconv` eigenpairs, we can retrieve the eigenvalue `k`, and the eigenvector, which is represented by means of two `petsc4py` vectors `vr` and `vi` (the real and imaginary part of the eigenvector, since for real matrices the eigenvalue and eigenvector may be complex). We also compute the corresponding relative errors in order to make sure that the computed solution is indeed correct:

```
if nconv > 0:
    # Create the results vectors
    vr, wr = A.getVecs()
    vi, wi = A.getVecs()
    #
    Print()
    Print("          k          ||Ax-kx||/||kx|| ")
    Print("-----")
    for i in range(nconv):
        k = E.getEigenpair(i, vr, vi)
        error = E.computeError(i)
        if k.imag != 0.0:
            Print(" %9f%+9f j %12g" % (k.real, k.imag, error))
        else:
            Print(" %12f      %12g" % (k.real, error))
    Print()
```

Example of command-line usage

Now we illustrate how to specify command-line options in order to extract the full potential of `slepc4py`.

A simple execution of the `demo/ex1.py` script will result in the following output:

```
$ python demo/ex1.py

*****
*** SLEPc Solution Results ***
*****

Number of iterations of the method: 4
Solution method: krylovschur
Number of requested eigenvalues: 1
Stopping condition: tol=1e-07, maxit=100
Number of converged eigenpairs 4

          k          ||Ax-kx||/||kx||
-----
    3.989739      5.76012e-09
```

(continues on next page)

(continued from previous page)

3.959060	1.41957e-08
3.908279	6.74118e-08
3.837916	8.34269e-08

For specifying different setting for the solver parameters, we can use SLEPc command-line options with the `-eps` prefix. For instance, to change the number of requested eigenvalues and the tolerance:

```
$ python demo/ex1.py -eps_nev 10 -eps_tol 1e-11
```

The method used by the solver object can also be set at run time:

```
$ python demo/ex1.py -eps_type subspace
```

All the above settings can also be changed within the source code by making use of the appropriate `slepc4py` method. Since options can be set from within the code and the command-line, it is often useful to view the particular settings that are currently being used:

```
$ python demo/ex1.py -eps_view
```

```
EPS Object: 1 MPI process
  type: krylovschur
    50% of basis vectors kept after restart
    using the locking variant
  problem type: symmetric eigenvalue problem
  selected portion of the spectrum: largest eigenvalues in magnitude
  number of eigenvalues (nev): 1
  number of column vectors (ncv): 16
  maximum dimension of projected problem (mpd): 16
  maximum number of iterations: 100
  tolerance: 1e-08
  convergence test: relative to the eigenvalue
BV Object: 1 MPI process
  type: mat
  17 columns of global length 30
  orthogonalization method: classical Gram-Schmidt
  orthogonalization refinement: if needed (eta: 0.7071)
  block orthogonalization method: GS
  doing matmult as a single matrix-matrix product
DS Object: 1 MPI process
  type: hep
  solving the problem with: Implicit QR method (_steqr)
ST Object: 1 MPI process
  type: shift
  shift: 0
  number of matrices: 1
```

Note that for computing eigenvalues of smallest magnitude we can use the option `-eps_smallest_magnitude`, but for interior eigenvalues things are not so straightforward. One possibility is to try with harmonic extraction, for instance to get the eigenvalues closest to 0.6:

```
$ python demo/ex1.py -eps_harmonic -eps_target 0.6
```

Depending on the problem, harmonic extraction may fail to converge. In those cases, it is necessary to specify a spectral

transformation other than the default. In the command-line, this is indicated with the `-st_` prefix. For example, shift-and-invert with a value of the shift equal to 0.6 would be:

```
$ python demo/ex1.py -st_type sinvert -eps_target 0.6
```

1.3 Installation

Using pip or easy_install

You can use **pip** to install *slepc4py* and its dependencies (*mpi4py* is optional but highly recommended):

```
$ pip install [--user] numpy mpi4py
$ pip install [--user] petsc petsc4py
$ pip install [--user] slepc slepc4py
```

Alternatively, you can use **easy_install** (deprecated):

```
$ easy_install [--user] slepc4py
```

If you already have working PETSc and SLEPc installs, set environment variables `SLEPC_DIR` and `PETSC_DIR` (and perhaps `PETSC_ARCH` for non-prefix installs) to appropriate values and next use **pip**:

```
$ export SLEPC_DIR=/path/to/slepc
$ export PETSC_DIR=/path/to/petsc
$ export PETSC_ARCH=arch-linux2-c-opt
$ pip install [--user] petsc4py slepc4py
```

Using distutils

Requirements

You need to have the following software properly installed in order to build *SLEPc for Python*:

- Any **MPI** implementation¹ (e.g., **MPICH** or **Open MPI**), built with shared libraries.
- A matching version of **PETSc** built with shared libraries.
- A matching version of **SLEPc** built with shared libraries.
- **NumPy** package.
- *petsc4py* package.

Downloading

The *SLEPc for Python* package is available for download at the Python Package Index. You can use **curl** or **wget** to get a release tarball.

- Using **curl**:

```
$ curl -LO https://pypi.io/packages/source/s/slepc4py/slepc4py-X.Y.Z.tar.gz
```

- Using **wget**:

```
$ wget https://pypi.io/packages/source/s/slepc4py/slepc4py-X.Y.Z.tar.gz
```

¹ Unless you have appropriately configured and built SLEPc and PETSc without MPI (configure option `--with-mpi=0`).

Building

After unpacking the release tarball:

```
$ tar -zxf slepc4py-X.Y.tar.gz
$ cd slepc4py-X.Y
```

the distribution is ready for building.

Note

Mac OS X users employing a Python distribution built with **universal binaries** may need to set the environment variables `MACOSX_DEPLOYMENT_TARGET`, `SDKROOT`, and `ARCHFLAGS` to appropriate values. As an example, assume your Mac is running **Snow Leopard** on a **64-bit Intel** processor and you want to override the hard-wired cross-development SDK in Python configuration, your environment should be modified like this:

```
$ export MACOSX_DEPLOYMENT_TARGET=10.6
$ export SDKROOT=/
$ export ARCHFLAGS='-arch x86_64'
```

Some environment configuration is needed to inform the location of PETSc and SLEPc. You can set (using **setenv**, **export** or what applies to you shell or system) the environment variables `SLEPC_DIR`, `PETSC_DIR`, and `PETSC_ARCH` indicating where you have built/installed SLEPc and PETSc:

```
$ export SLEPC_DIR=/usr/local/slepc
$ export PETSC_DIR=/usr/local/petsc
$ export PETSC_ARCH=arch-linux2-c-opt
```

Alternatively, you can edit the file `setup.cfg` and provide the required information below the `[config]` section:

```
[config]
slepc_dir  = /usr/local/slepc
petsc_dir  = /usr/local/petsc
petsc_arch = arch-linux2-c-opt
...
```

Finally, you can build the distribution by typing:

```
$ python setup.py build
```

Installing

After building, the distribution is ready for installation.

If you have root privileges (either by log-in as the root user or by using **sudo**) and you want to install *SLEPc for Python* in your system for all users, just do:

```
$ python setup.py install
```

The previous steps will install the *slepc4py* package at standard location `prefix/lib/pythonX.X/site-packages`.

If you do not have root privileges or you want to install *SLEPc for Python* for your private use, just do:

```
$ python setup.py install --user
```

1.4 Citations

If SLEPc for Python been significant to a project that leads to an academic publication, please acknowledge that fact by citing the project.

- L. Dalcin, P. Kler, R. Paz, and A. Cosimo, *Parallel Distributed Computing using Python*, Advances in Water Resources, 34(9):1124-1139, 2011. <http://dx.doi.org/10.1016/j.advwatres.2011.04.013>
- V. Hernandez, J.E. Roman, and V. Vidal, *SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems*, ACM Transactions on Mathematical Software, 31(3):351-362, 2005. <http://dx.doi.org/10.1145/1089014.1089019>

1.5 Reference

<code>slepc4py</code>	SLEPc for Python
<code>slepc4py.typing</code>	Typing support.
<code>slepc4py.SLEPc</code>	Scalable Library for Eigenvalue Problem Computations

slepc4py

SLEPc for Python

This package is an interface to [SLEPc](#) libraries.

[SLEPc](#) (the Scalable Library for Eigenvalue Problem Computations) is a software library for the solution of large scale sparse eigenvalue problems on parallel computers. It is an extension of [PETSc](#) and can be used for either standard or generalized eigenproblems, with real or complex arithmetic. It can also be used for computing a partial SVD of a large, sparse, rectangular matrix.

Functions

<code>get_config()</code>	
<code>get_include()</code>	Return the directory in the package that contains header files.
<code>init([args, arch])</code>	Initialize SLEPc.

slepc4py.get_config

`slepc4py.get_config()`

Return type

`dict[str, str]`

slepc4py.get_include

`slepc4py.get_include()`

Return the directory in the package that contains header files.

Extension modules that need to compile against slepc4py should use this function to locate the appropriate include directory. Using Python distutils (or perhaps NumPy distutils):

```
import petscc4py, slepc4py
Extension('extension_name', ...
         include_dirs=[...,
                       petsc4py.get_include(),
                       slepc4py.get_include(),])
```

Return type

str

slepc4py.init

slepc4py.**init**(args=None, arch=None)

Initialize SLEPc.

Parameters

- **Beyond** `sys.argv`: command-line arguments, usually the ‘sys.argv’ list.
- **arch**: specific configuration to use.

Parameters

- **args** (`str` | `list[str]` | `None`)
- **arch** (`str` | `None`)

Return type

None

Note

This function should be called only once, typically at the very beginning of the bootstrap script of an application.

slepc4py.typing

Typing support.

Attributes

<code>Scalar</code>	Scalar type.
<code>ArrayInt</code>	Array of <code>int</code> .
<code>ArrayReal</code>	Array of <code>float</code> .
<code>ArrayComplex</code>	Array of <code>complex</code> .
<code>ArrayScalar</code>	Array of <code>Scalar</code> numbers.
<code>LayoutSizeSpec</code>	<code>int</code> or 2-tuple of <code>int</code> describing the layout sizes.
<code>EPStoppingFunction</code>	<code>EPS</code> stopping test callback.
<code>EPArbitraryFunction</code>	<code>EPS</code> arbitrary selection callback.
<code>EPSEigenvalueComparison</code>	<code>EPS</code> eigenvalue comparison callback.
<code>EPMonitorFunction</code>	<code>EPS</code> monitor callback.
<code>PEPStoppingFunction</code>	<code>PEP</code> stopping test callback.
<code>PEPMonitorFunction</code>	<code>PEP</code> monitor callback.
<code>NEPStoppingFunction</code>	<code>NEP</code> stopping test callback.
<code>NEPMonitorFunction</code>	<code>NEP</code> monitor callback.

continues on next page

Table 3 – continued from previous page

<i>NEPFunction</i>	<i>NEP</i> Function callback.
<i>NEPJacobian</i>	<i>NEP</i> Jacobian callback.
<i>SVDStoppingFunction</i>	<i>SVD</i> stopping test callback.
<i>SVDMonitorFunction</i>	<i>SVD</i> monitor callback.
<i>MFNMonitorFunction</i>	<i>MFN</i> monitor callback.
<i>LMEMonitorFunction</i>	<i>LME</i> monitor callback.

slepc4py.typing.Scalar

`slepc4py.typing.Scalar = float | complex`

Scalar type.

Scalars can be either `float` or `complex` (but not both) depending on how PETSc was configured (`./configure --with-scalar-type=real | complex`).

slepc4py.typing.ArrayInt

`slepc4py.typing.ArrayInt`

Array of `int`.

alias of `ndarray[tuple[Any, ...], dtype[int]]`

slepc4py.typing.ArrayReal

`slepc4py.typing.ArrayReal`

Array of `float`.

alias of `ndarray[tuple[Any, ...], dtype[float]]`

slepc4py.typing.ArrayComplex

`slepc4py.typing.ArrayComplex`

Array of `complex`.

alias of `ndarray[tuple[Any, ...], dtype[complex]]`

slepc4py.typing.ArrayScalar

`slepc4py.typing.ArrayScalar`

Array of *Scalar* numbers.

alias of `ndarray[tuple[Any, ...], dtype[float | complex]]`

slepc4py.typing.LayoutSizeSpec

`slepc4py.typing.LayoutSizeSpec = int | tuple[int, int]`

`int` or 2-tuple of `int` describing the layout sizes.

A single `int` indicates global size. A `tuple` of `int` indicates (`local_size`, `global_size`).

slepc4py.typing.EPSStoppingFunction

`slepc4py.typing.EPSStoppingFunction`

EPS stopping test callback.

alias of `Callable[[EPS, int, int, int, int], ConvergedReason]`

slepc4py.typing.EPSArbitraryFunction

`slepc4py.typing.EPSArbitraryFunction`

EPS arbitrary selection callback.

alias of `Callable[[float | complex, float | complex, Vec, Vec, float | complex, float | complex], [float | complex, float | complex]]`

slepc4py.typing.EPSEigenvalueComparison

`slepc4py.typing.EPSEigenvalueComparison`

EPS eigenvalue comparison callback.

alias of `Callable[[float | complex, float | complex, float | complex, float | complex], int]`

slepc4py.typing.EPSMonitorFunction

`slepc4py.typing.EPSMonitorFunction`

EPS monitor callback.

alias of `Callable[[EPS, int, int, ndarray[tuple[Any, ...], dtype[float | complex]], ndarray[tuple[Any, ...], dtype[float | complex]], ndarray[tuple[Any, ...], dtype[float]], int, None]`

slepc4py.typing.PEPStoppingFunction

`slepc4py.typing.PEPStoppingFunction`

PEP stopping test callback.

alias of `Callable[[PEP, int, int, int, int], ConvergedReason]`

slepc4py.typing.PEPMonitorFunction

`slepc4py.typing.PEPMonitorFunction`

PEP monitor callback.

alias of `Callable[[PEP, int, int, ndarray[tuple[Any, ...], dtype[float | complex]], ndarray[tuple[Any, ...], dtype[float | complex]], ndarray[tuple[Any, ...], dtype[float]], int, None]`

slepc4py.typing.NEPStoppingFunction

`slepc4py.typing.NEPStoppingFunction`

NEP stopping test callback.

alias of `Callable[[NEP, int, int, int, int], ConvergedReason]`

slepc4py.typing.NEPMonitorFunction

slepc4py.typing.NEPMonitorFunction

NEP monitor callback.

alias of `Callable[[NEP, int, int, ndarray[tuple[Any, ...], dtype[float | complex]], ndarray[tuple[Any, ...], dtype[float | complex]], ndarray[tuple[Any, ...], dtype[float]], int], None]`

slepc4py.typing.NEPFunction

slepc4py.typing.NEPFunction

NEP Function callback.

alias of `Callable[[NEP, float | complex, Mat, Mat], None]`

slepc4py.typing.NEPJacobian

slepc4py.typing.NEPJacobian

NEP Jacobian callback.

alias of `Callable[[NEP, float | complex, Mat], None]`

slepc4py.typing.SVDStoppingFunction

slepc4py.typing.SVDStoppingFunction

SVD stopping test callback.

alias of `Callable[[SVD, int, int, int, int], ConvergedReason]`

slepc4py.typing.SVDMonitorFunction

slepc4py.typing.SVDMonitorFunction

SVD monitor callback.

alias of `Callable[[SVD, int, int, ndarray[tuple[Any, ...], dtype[float]], ndarray[tuple[Any, ...], dtype[float]], int], None]`

slepc4py.typing.MFNMonitorFunction

slepc4py.typing.MFNMonitorFunction

MFN monitor callback.

alias of `Callable[[MFN, int, float], None]`

slepc4py.typing.LMEMonitorFunction

slepc4py.typing.LMEMonitorFunction

LME monitor callback.

alias of `Callable[[LME, int, float], None]`

slepc4py.SLEPc

Scalable Library for Eigenvalue Problem Computations

Classes

<i>BV</i>	BV.
<i>BVSVDMethod</i>	BV methods for computing the SVD.
<i>DS</i>	DS.
<i>EPS</i>	EPS.
<i>EPSKrylovSchurBSEType</i>	EPS Krylov-Schur method for BSE problems.
<i>FN</i>	FN.
<i>LME</i>	LME.
<i>MFN</i>	MFN.
<i>NEP</i>	NEP.
<i>PEP</i>	PEP.
<i>RG</i>	RG.
<i>ST</i>	ST.
<i>STFilterDamping</i>	ST filter damping.
<i>STFilterType</i>	ST filter type.
<i>SVD</i>	SVD.
<i>Sys</i>	Sys.
<i>Util</i>	Util.

slepc4py.SLEPc.BV

class slepc4py.SLEPc.BV

Bases: `Object`

BV.

Enumerations

<i>MatMultType</i>	BV mat-mult types.
<i>OrthogBlockType</i>	BV block-orthogonalization types.
<i>OrthogRefineType</i>	BV orthogonalization refinement types.
<i>OrthogType</i>	BV orthogonalization types.
<i>Type</i>	BV type.

slepc4py.SLEPc.BV.MatMultType

class slepc4py.SLEPc.BV.MatMultType

Bases: `object`

BV mat-mult types.

- *VECS*: Perform a matrix-vector multiply per each column.
- *MAT*: Carry out a Mat-Mat product with a dense matrix.

Attributes Summary

<i>MAT</i>	Constant MAT of type <code>int</code>
<i>VECS</i>	Constant VECS of type <code>int</code>

Attributes Documentation

MAT: `int` = MAT

Constant MAT of type `int`

VECS: `int` = VECS

Constant VECS of type `int`

slepc4py.SLEPc.BV.OrthogBlockType

class slepc4py.SLEPc.BV.OrthogBlockType

Bases: `object`

BV block-orthogonalization types.

- *GS*: Gram-Schmidt.
- *CHOL*: Cholesky.
- *TSQR*: Tall-skinny QR.
- *TSQRCHOL*: Tall-skinny QR with Cholesky.
- *SVQB*: SVQB.

Attributes Summary

<i>CHOL</i>	Constant CHOL of type <code>int</code>
<i>GS</i>	Constant GS of type <code>int</code>
<i>SVQB</i>	Constant SVQB of type <code>int</code>
<i>TSQR</i>	Constant TSQR of type <code>int</code>
<i>TSQRCHOL</i>	Constant TSQRCHOL of type <code>int</code>

Attributes Documentation

CHOL: `int` = CHOL

Constant CHOL of type `int`

GS: `int` = GS

Constant GS of type `int`

SVQB: `int` = SVQB

Constant SVQB of type `int`

TSQR: `int` = TSQR

Constant TSQR of type `int`

TSQRCHOL: `int` = TSQRCHOL

Constant TSQRCHOL of type `int`

slepc4py.SLEPc.BV.OrthogRefineType

class slepc4py.SLEPc.BV.OrthogRefineType

Bases: `object`

BV orthogonalization refinement types.

- *IFNEEDED*: Reorthogonalize if a criterion is satisfied.
- *NEVER*: Never reorthogonalize.
- *ALWAYS*: Always reorthogonalize.

Attributes Summary

<i>ALWAYS</i>	Constant ALWAYS of type <code>int</code>
<i>IFNEEDED</i>	Constant IFNEEDED of type <code>int</code>
<i>NEVER</i>	Constant NEVER of type <code>int</code>

Attributes Documentation

ALWAYS: `int` = ALWAYS

Constant ALWAYS of type `int`

IFNEEDED: `int` = IFNEEDED

Constant IFNEEDED of type `int`

NEVER: `int` = NEVER

Constant NEVER of type `int`

slepc4py.SLEPc.BV.OrthogType

class slepc4py.SLEPc.BV.OrthogType

Bases: `object`

BV orthogonalization types.

- *CGS*: Classical Gram-Schmidt.
- *MGS*: Modified Gram-Schmidt.

Attributes Summary

<i>CGS</i>	Constant CGS of type <code>int</code>
<i>MGS</i>	Constant MGS of type <code>int</code>

Attributes Documentation

CGS: `int` = CGS

Constant CGS of type `int`

MGS: `int` = MGS

Constant MGS of type `int`

slepc4py.SLEPc.BV.Type

class slepc4py.SLEPc.BV.Type

Bases: `object`

BV type.

Attributes Summary

<i>CONTIGUOUS</i>	Object CONTIGUOUS of type <code>str</code>
<i>MAT</i>	Object MAT of type <code>str</code>
<i>SVEC</i>	Object SVEC of type <code>str</code>
<i>TENSOR</i>	Object TENSOR of type <code>str</code>
<i>VECS</i>	Object VECS of type <code>str</code>

Attributes Documentation

CONTIGUOUS: `str` = CONTIGUOUS

Object CONTIGUOUS of type `str`

MAT: `str` = MAT

Object MAT of type `str`

SVEC: `str` = SVEC

Object SVEC of type `str`

TENSOR: `str` = TENSOR

Object TENSOR of type `str`

VECS: `str` = VECS

Object VECS of type `str`

Methods Summary

<i>appendOptionsPrefix</i> ([prefix])	Append to the prefix used for searching for all BV options in the database.
<i>applyMatrix</i> (x, y)	Multiply a vector with the matrix associated to the bilinear form.
<i>copy</i> ([result])	Copy a basis vector object into another one.
<i>copyColumn</i> (j, i)	Copy the values from one of the columns to another one.
<i>copyVec</i> (j, v)	Copy one of the columns of a basis vectors object into a Vec.
<i>create</i> ([comm])	Create the BV object.
<i>createFromMat</i> (A)	Create a basis vectors object from a dense Mat object.
<i>createMat</i> ()	Create a new Mat object of dense type and copy the contents of the BV.
<i>createVec</i> ()	Create a Vec with the type and dimensions of the columns of the BV.
<i>destroy</i> ()	Destroy the BV object.
<i>dot</i> (Y)	Compute the 'block-dot' product of two basis vectors objects.

continues on next page

Table 11 – continued from previous page

<code>dotColumn(j)</code>	Dot products of a column against all the column vectors of a BV.
<code>dotVec(v)</code>	Dot products of a vector against all the column vectors of the BV.
<code>duplicate()</code>	Duplicate the BV object with the same type and dimensions.
<code>duplicateResize(m)</code>	Create a BV object of the same type and dimensions as an existing one.
<code>getActiveColumns()</code>	Get the current active dimensions.
<code>getColumn(j)</code>	Get a Vec object with the entries of the column of the BV object.
<code>getDefiniteTolerance()</code>	Get the tolerance to be used when checking a definite inner product.
<code>getLeadingDimension()</code>	Get the leading dimension.
<code>getMat()</code>	Get a Mat object of dense type that shares the memory of the BV object.
<code>getMatMultMethod()</code>	Get the method used for the <code>matMult()</code> operation.
<code>getMatrix()</code>	Get the matrix representation of the inner product.
<code>getNumConstraints()</code>	Get the number of constraints.
<code>getOptionsPrefix()</code>	Get the prefix used for searching for all BV options in the database.
<code>getOrthogonalization()</code>	Get the orthogonalization settings from the BV object.
<code>getRandomContext()</code>	Get the <code>petsc4py.PETSc.Random</code> object associated with the BV.
<code>getSizes()</code>	Get the local and global sizes, and the number of columns.
<code>getType()</code>	Get the BV type of this object.
<code>getVecType()</code>	Get the vector type used by the basis vectors object.
<code>insertConstraints(C)</code>	Insert a set of vectors as constraints.
<code>insertVec(j, w)</code>	Insert a vector into the specified column.
<code>insertVecs(s, W[, orth])</code>	Insert a set of vectors into specified columns.
<code>matMult(A[, Y])</code>	Compute the matrix-vector product for each column, $Y = AV$.
<code>matMultColumn(A, j)</code>	Mat-vec product for a column, storing the result in the next column.
<code>matMultHermitianTranspose(A[, Y])</code>	Pre-multiplication with the conjugate transpose of a matrix.
<code>matMultHermitianTransposeColumn(A, j)</code>	Conjugate-transpose matrix-vector product for a specified column.
<code>matMultTransposeColumn(A, j)</code>	Transpose matrix-vector product for a specified column.
<code>matProject(A, Y)</code>	Compute the projection of a matrix onto a subspace.
<code>mult(alpha, beta, X, Q)</code>	Compute $Y = \beta Y + \alpha XQ$.
<code>multColumn(alpha, beta, j, q)</code>	Compute $y = \beta y + \alpha Xq$.
<code>multInPlace(Q, s, e)</code>	Update a set of vectors as $V(:, s : e - 1) = VQ(:, s : e - 1)$.
<code>multVec(alpha, beta, y, q)</code>	Compute $y = \beta y + \alpha Xq$.
<code>norm([norm_type])</code>	Compute the matrix norm of the BV.
<code>normColumn(j[, norm_type])</code>	Compute the vector norm of a selected column.
<code>orthogonalize([R])</code>	Orthogonalize all columns (except leading ones) (QR decomposition).

continues on next page

Table 11 – continued from previous page

<code>orthogonalizeColumn(j)</code>	Orthogonalize a column vector with respect to the previous ones.
<code>orthogonalizeVec(v)</code>	Orthogonalize a vector with respect to a set of vectors.
<code>orthonormalizeColumn(j[, replace])</code>	Orthonormalize a column vector with respect to the previous ones.
<code>resize(m[, copy])</code>	Change the number of columns.
<code>restoreColumn(j, v)</code>	Restore a column obtained with <code>getColumn()</code> .
<code>restoreMat(A)</code>	Restore the Mat obtained with <code>getMat()</code> .
<code>scale(alpha)</code>	Multiply the entries by a scalar value.
<code>scaleColumn(j, alpha)</code>	Scale column j by alpha.
<code>setActiveColumns(l, k)</code>	Set the columns that will be involved in operations.
<code>setDefiniteTolerance(deftol)</code>	Set the tolerance to be used when checking a definite inner product.
<code>setFromOptions()</code>	Set BV options from the options database.
<code>setLeadingDimension(ld)</code>	Set the leading dimension.
<code>setMatMultMethod(method)</code>	Set the method used for the <code>matMult()</code> operation.
<code>setMatrix(mat[, indef])</code>	Set the bilinear form to be used for inner products.
<code>setNumConstraints(nc)</code>	Set the number of constraints.
<code>setOptionsPrefix([prefix])</code>	Set the prefix used for searching for all BV options in the database.
<code>setOrthogonalization([otype, refine, eta, block])</code>	Set the method used for the (block-)orthogonalization of vectors.
<code>setRandom()</code>	Set the active columns of the BV to random numbers.
<code>setRandomColumn(j)</code>	Set one column of the BV to random numbers.
<code>setRandomCond(condn)</code>	Set the columns of a BV to random numbers.
<code>setRandomContext(rnd)</code>	Set the <code>petsc4py.PETSc.Random</code> object associated with the BV.
<code>setRandomNormal()</code>	Set the active columns of the BV to normal random numbers.
<code>setRandomSign()</code>	Set the entries of a BV to values 1 or -1 with equal probability.
<code>setSizes(sizes, m)</code>	Set the local and global sizes, and the number of columns.
<code>setSizesFromVec(w, m)</code>	Set the local and global sizes, and the number of columns.
<code>setType(bv_type)</code>	Set the type for the BV object.
<code>setVecType(vec_type)</code>	Set the vector type.
<code>view([viewer])</code>	Print the BV data structure.

Attributes Summary

<code>column_size</code>	Basis vectors column size.
<code>local_size</code>	Basis vectors local size.
<code>size</code>	Basis vectors global size.
<code>sizes</code>	Basis vectors local and global sizes, and the number of columns.

Methods Documentation

appendOptionsPrefix(*prefix=None*)

Append to the prefix used for searching for all BV options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all BV option requests.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:413 <slepc4py/SLEPc/BV.pyx#L413>`

applyMatrix(*x, y*)

Multiply a vector with the matrix associated to the bilinear form.

Neighbor-wise collective.

Parameters

- **x** (*Vec*) – The input vector.
- **y** (*Vec*) – The result vector.

Return type

None

Notes

If the bilinear form has no associated matrix this function copies the vector.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:605 <slepc4py/SLEPc/BV.pyx#L605>`

copy(*result=None*)

Copy a basis vector object into another one.

Logically collective.

Parameters

result (*BV* / *None*) – The copy.

Return type

BV

:sources: `Source code at slepc4py/SLEPc/BV.pyx:255 <slepc4py/SLEPc/BV.pyx#L255>`

copyColumn(*j, i*)

Copy the values from one of the columns to another one.

Logically collective.

Parameters

- **j** (*int*) – The number of the source column.
- **i** (*int*) – The number of the destination column.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:866 <slepc4py/SLEPc/BV.pyx#L866>`

copyVec(*j*, *v*)

Copy one of the columns of a basis vectors object into a Vec.

Logically collective.

Parameters

- *j* (*int*) – The column number to be copied.
- *v* (*Vec*) – A vector.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:850 <slepc4py/SLEPc/BV.pyx#L850>`

create(*comm=None*)

Create the BV object.

Collective.

Parameters

comm (*Comm* | *None*) – MPI communicator; if not provided, it defaults to all processes.

Return type

Self

:sources: `Source code at slepc4py/SLEPc/BV.pyx:175 <slepc4py/SLEPc/BV.pyx#L175>`

createFromMat(*A*)

Create a basis vectors object from a dense Mat object.

Collective.

Parameters

A (*Mat*) – A dense tall-skinny matrix.

Return type

Self

:sources: `Source code at slepc4py/SLEPc/BV.pyx:193 <slepc4py/SLEPc/BV.pyx#L193>`

createMat()

Create a new Mat object of dense type and copy the contents of the BV.

Collective.

Returns

The new matrix.

Return type

`petsc4py.PETSc.Mat`

:sources: `Source code at slepc4py/SLEPc/BV.pyx:209 <slepc4py/SLEPc/BV.pyx#L209>`

createVec()

Create a Vec with the type and dimensions of the columns of the BV.

Collective.

Returns

New vector.

Return type

`petsc4py.PETSc.Vec`

:sources: `Source code at slepc4py/SLEPc/BV.pyx:810 <slepc4py/SLEPc/BV.pyx#L810>`

destroy()

Destroy the BV object.

Collective.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:165 <slepc4py/SLEPc/BV.pyx#L165>`

Return type

Self

dot(*Y*)

Compute the ‘block-dot’ product of two basis vectors objects.

Collective.

$M = Y^H X$ ($m_{ij} = y_i^H x_j$) or $M = Y^H B X$

Parameters

Y (*BV*) – Left basis vectors, can be the same as self, giving $M = X^H X$.

Returns

The resulting matrix.

Return type

petsc4py.PETSc.Mat

Notes

This is the generalization of `VecDot()` for a collection of vectors, $M = Y^H X$. The result is a matrix M whose entry m_{ij} is equal to $y_i^H x_j$ (where y_i^H denotes the conjugate transpose of y_i).

X and Y can be the same object.

If a non-standard inner product has been specified with `setMatrix()`, then the result is $M = Y^H B X$. In this case, both X and Y must have the same associated matrix.

Only rows (resp. columns) of M starting from ly (resp. lx) are computed, where ly (resp. lx) is the number of leading columns of Y (resp. X).

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1064 <slepc4py/SLEPc/BV.pyx#L1064>`

dotColumn(*j*)

Dot products of a column against all the column vectors of a BV.

Collective.

Parameters

j (*int*) – The index of the column.

Returns

The computed values.

Return type

ArrayScalar

:sources: `Source code at slepc4py/SLEPc/BV.pyx:948 <slepc4py/SLEPc/BV.pyx#L948>`

dotVec(*v*)

Dot products of a vector against all the column vectors of the BV.

Collective.

Parameters

v (*Vec*) – A vector.

Returns

The computed values.

Return type

ArrayScalar

Notes

This is analogue to VecMDot(), but using BV to represent a collection of vectors. The result is $m = X^H y$, so m_i is equal to $x_j^H y$. Note that here X is transposed as opposed to BVDot().

If a non-standard inner product has been specified with BVSetMatrix(), then the result is $m = X^H B y$.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:912 <slepc4py/SLEPc/BV.pyx#L912>`

duplicate()

Duplicate the BV object with the same type and dimensions.

Collective.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:224 <slepc4py/SLEPc/BV.pyx#L224>`

Return type

BV

duplicateResize(m)

Create a BV object of the same type and dimensions as an existing one.

Collective.

Parameters

m (*int*) – The number of columns.

Return type

BV

Notes

With possibly different number of columns.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:234 <slepc4py/SLEPc/BV.pyx#L234>`

getActiveColumns()

Get the current active dimensions.

Not collective.

Returns

- **l** (*int*) – The leading number of columns.
- **k** (*int*) – The active number of columns.

Return type

tuple[*int*, *int*]

:sources: `Source code at slepc4py/SLEPc/BV.pyx:642 <slepc4py/SLEPc/BV.pyx#L642>`

getColumn(*j*)

Get a Vec object with the entries of the column of the BV object.

Logically collective.

Parameters

j (*int*) – The index of the requested column.

Returns

The vector containing the *j*th column.

Return type

`petsc4py.PETSc.Vec`

Notes

Modifying the returned Vec will change the BV entries as well.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:975 <slepc4py/SLEPc/BV.pyx#L975>`

getDefiniteTolerance()

Get the tolerance to be used when checking a definite inner product.

Not collective.

Returns

The tolerance.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/BV.pyx:897 <slepc4py/SLEPc/BV.pyx#L897>`

getLeadingDimension()

Get the leading dimension.

Not collective.

Returns

The leading dimension.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/BV.pyx:377 <slepc4py/SLEPc/BV.pyx#L377>`

getMat()

Get a Mat object of dense type that shares the memory of the BV object.

Collective.

Returns

The matrix.

Return type

`petsc4py.PETSc.Mat`

Notes

The returned matrix contains only the active columns. If the content of the Mat is modified, these changes are also done in the BV object. The user must call `restoreMat()` when no longer needed.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1022 <slepc4py/SLEPc/BV.pyx#L1022>`

getMatMultMethod()

Get the method used for the *matMult()* operation.

Not collective.

Returns

The method for the *matMult()* operation.

Return type

MatMultType

:sources: `Source code at slepc4py/SLEPc/BV.pyx:535 <slepc4py/SLEPc/BV.pyx#L535>`

getMatrix()

Get the matrix representation of the inner product.

Not collective.

Returns

- **mat** (*petsc4py.PETSc.Mat*) – The matrix of the inner product
- **indef** (*bool*) – Whether the matrix is indefinite

Return type

tuple[*petsc4py.PETSc.Mat*, *bool*] | *tuple*[*None*, *bool*]

:sources: `Source code at slepc4py/SLEPc/BV.pyx:566 <slepc4py/SLEPc/BV.pyx#L566>`

getNumConstraints()

Get the number of constraints.

Not collective.

Returns

The number of constraints.

Return type

int

:sources: `Source code at slepc4py/SLEPc/BV.pyx:795 <slepc4py/SLEPc/BV.pyx#L795>`

getOptionsPrefix()

Get the prefix used for searching for all BV options in the database.

Not collective.

Returns

The prefix string set for this BV object.

Return type

str

:sources: `Source code at slepc4py/SLEPc/BV.pyx:428 <slepc4py/SLEPc/BV.pyx#L428>`

getOrthogonalization()

Get the orthogonalization settings from the BV object.

Not collective.

Returns

- **type** (*OrthogType*) – The type of orthogonalization technique.
- **refine** (*OrthogRefineType*) – The type of refinement.

- **eta** (*float*) – Parameter for selective refinement (used when the refinement type is *BV.OrthogRefineType.IFNEEDED*).
- **block** (*OrthogBlockType*) – The type of block orthogonalization .

Return type

tuple[OrthogType, OrthogRefineType, float, OrthogBlockType]

:sources: `Source code at slepc4py/SLEPc/BV.pyx:458 <slepc4py/SLEPc/BV.pyx#L458>`

getRandomContext()

Get the `petsc4py.PETSc.Random` object associated with the BV.

Collective.

Returns

The random number generator context.

Return type

`petsc4py.PETSc.Random`

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1554 <slepc4py/SLEPc/BV.pyx#L1554>`

getSizes()

Get the local and global sizes, and the number of columns.

Not collective.

Returns

- **(n, N)** (*tuple of int*) – The local and global sizes
- **m** (*int*) – The number of columns.

Return type

tuple[LayoutSizeSpec, int]

:sources: `Source code at slepc4py/SLEPc/BV.pyx:346 <slepc4py/SLEPc/BV.pyx#L346>`

getType()

Get the BV type of this object.

Not collective.

Returns

The inner product type currently being used.

Return type

str

:sources: `Source code at slepc4py/SLEPc/BV.pyx:288 <slepc4py/SLEPc/BV.pyx#L288>`

getVecType()

Get the vector type used by the basis vectors object.

Not collective.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:840 <slepc4py/SLEPc/BV.pyx#L840>`

Return type

str

insertConstraints(C)

Insert a set of vectors as constraints.

Collective.

Parameters

C (*Vec* | *list*[*Vec*]) – Set of vectors to be inserted as constraints.

Returns

Number of constraints.

Return type

int

Notes

The constraints are relevant only during orthogonalization. Constraint vectors span a subspace that is deflated in every orthogonalization operation, so they are intended for removing those directions from the orthogonal basis computed in regular BV columns.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:749 <slepc4py/SLEPc/BV.pyx#L749>`

insertVec(*j*, *w*)

Insert a vector into the specified column.

Logically collective.

Parameters

- *j* (*int*) – The column to be overwritten.
- *w* (*Vec*) – The vector to be copied.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:694 <slepc4py/SLEPc/BV.pyx#L694>`

insertVecs(*s*, *W*, *orth=False*)

Insert a set of vectors into specified columns.

Collective.

Parameters

- *s* (*int*) – The first column to be overwritten.
- *W* (*Vec* | *list*[*Vec*]) – Set of vectors to be copied.
- *orth* (*bool*) – Flag indicating if the vectors must be orthogonalized.

Returns

Number of linearly independent vectors.

Return type

int

Notes

Copies the contents of vectors *W* into `self[:,s:s+n]`, where *n* is the length of *W*. If orthogonalization flag is set then the vectors are copied one by one then orthogonalized against the previous one. If any are linearly dependent then it is discarded and the value of *m* is decreased.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:710 <slepc4py/SLEPc/BV.pyx#L710>`

matMult(*A*, *Y=None*)

Compute the matrix-vector product for each column, $Y = AV$.

Neighbor-wise collective.

Parameters

- **A** (*Mat*) – The matrix.
- **Y** (*BV* / *None*)

Returns

The result.

Return type

BV

Notes

Only active columns (excluding the leading ones) are processed.

It is possible to choose whether the computation is done column by column or using dense matrices using the options database keys:

`-bv_matmult_vecs -bv_matmult_mat`

The default is `bv_matmult_mat`.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1139 <slepc4py/SLEPc/BV.pyx#L1139>`

matMultColumn(*A, j*)

Mat-vec product for a column, storing the result in the next column.

Neighbor-wise collective.

$$v_{j+1} = Av_j.$$

Parameters

- **A** (*Mat*) – The matrix.
- **j** (*int*) – Index of column.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1234 <slepc4py/SLEPc/BV.pyx#L1234>`

matMultHermitianTranspose(*A, Y=None*)

Pre-multiplication with the conjugate transpose of a matrix.

Neighbor-wise collective.

$$Y = A^H V.$$

Parameters

- **A** (*Mat*) – The matrix.
- **Y** (*BV* / *None*)

Returns

The result.

Return type

BV

Notes

Only active columns (excluding the leading ones) are processed.

As opposed to `matMult()`, this operation is always done by column by column, with a sequence of calls to `MatMultHermitianTranspose()`.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1188 <slepc4py/SLEPc/BV.pyx#L1188>`

matMultHermitianTransposeColumn(A, j)

Conjugate-transpose matrix-vector product for a specified column.

Neighbor-wise collective.

Store the result in the next column: $v_{j+1} = A^H v_j$.

Parameters

- **A** (*Mat*) – The matrix.
- **j** (*int*) – Index of column.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1270 <slepc4py/SLEPc/BV.pyx#L1270>`

matMultTransposeColumn(A, j)

Transpose matrix-vector product for a specified column.

Neighbor-wise collective.

Store the result in the next column: $v_{j+1} = A^T v_j$.

Parameters

- **A** (*Mat*) – The matrix.
- **j** (*int*) – Index of column.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1252 <slepc4py/SLEPc/BV.pyx#L1252>`

matProject(A, Y)

Compute the projection of a matrix onto a subspace.

Collective.

$$M = Y^H A X$$

Parameters

- **A** (*petsc4py.PETSc.Mat* | *None*) – Matrix to be projected.
- **Y** (*BV*) – Left basis vectors, can be the same as self, giving $M = X^H A X$.

Returns

Projection of the matrix A onto the subspace.

Return type

petsc4py.PETSc.Mat

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1109 <slepc4py/SLEPc/BV.pyx#L1109>`

mult(*alpha*, *beta*, *X*, *Q*)

Compute $Y = \text{beta}Y + \text{alpha}XQ$.

Logically collective.

Parameters

- **alpha** ([Scalar](#)) – Coefficient that multiplies X.
- **beta** ([Scalar](#)) – Coefficient that multiplies Y.
- **X** ([BV](#)) – Input basis vectors.
- **Q** ([Mat](#)) – Input matrix, if not given the identity matrix is assumed.

Return type

[None](#)

:sources: [Source code at slepc4py/SLEPc/BV.pyx:1288 <slepc4py/SLEPc/BV.pyx#L1288>](#)

multColumn(*alpha*, *beta*, *j*, *q*)

Compute $y = \text{beta}y + \text{alpha}Xq$.

Logically collective.

Compute $y = \text{beta}y + \text{alpha}Xq$, where y is the j^{th} column.

Parameters

- **alpha** ([Scalar](#)) – Coefficient that multiplies X.
- **beta** ([Scalar](#)) – Coefficient that multiplies y.
- **j** ([int](#)) – The column index.
- **q** ([Sequence](#)[[Scalar](#)]) – Input coefficients.

Return type

[None](#)

:sources: [Source code at slepc4py/SLEPc/BV.pyx:1329 <slepc4py/SLEPc/BV.pyx#L1329>](#)

multInPlace(*Q*, *s*, *e*)

Update a set of vectors as $V(:, s : e - 1) = VQ(:, s : e - 1)$.

Logically collective.

Parameters

- **Q** ([Mat](#)) – A sequential dense matrix.
- **s** ([int](#)) – First column to be overwritten.
- **e** ([int](#)) – Last column to be overwritten.

Return type

[None](#)

:sources: [Source code at slepc4py/SLEPc/BV.pyx:1310 <slepc4py/SLEPc/BV.pyx#L1310>](#)

multVec(*alpha*, *beta*, *y*, *q*)

Compute $y = \text{beta}y + \text{alpha}Xq$.

Logically collective.

Parameters

- **alpha** ([Scalar](#)) – Coefficient that multiplies X.

- **beta** (*Scalar*) – Coefficient that multiplies y.
- **y** (*Vec*) – Input/output vector.
- **q** (*Sequence*[*Scalar*]) – Input coefficients.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1360 <slepc4py/SLEPc/BV.pyx#L1360>`

norm(*norm_type=None*)

Compute the matrix norm of the BV.

Collective.

Parameters

norm_type (*NormType* / *None*) – The norm type.

Returns

The norm.

Return type

float

Notes

All active columns (except the leading ones) are considered as a matrix. The allowed norms are NORM_1, NORM_FROBENIUS, and NORM_INFINITY.

This operation fails if a non-standard inner product has been specified with BVSetMatrix().

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1420 <slepc4py/SLEPc/BV.pyx#L1420>`

normColumn(*j*, *norm_type=None*)

Compute the vector norm of a selected column.

Collective.

Parameters

- **j** (*int*) – Index of column.
- **norm_type** (*NormType* / *None*) – The norm type.

Returns

The norm.

Return type

float

Notes

The norm of V_j is computed (NORM_1, NORM_2, or NORM_INFINITY).

If a non-standard inner product has been specified with BVSetMatrix(), then the returned value is $\sqrt{V_j^H B V_j}$, where B is the inner product matrix (argument 'type' is ignored).

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1387 <slepc4py/SLEPc/BV.pyx#L1387>`

orthogonalize(*R=None*, ***kargs*)

Orthogonalize all columns (except leading ones) (QR decomposition).

Collective.

Parameters

- **R** (*Mat* / *None*) – A sequential dense matrix.
- **kargs** (*Any*)

Return type

None

Notes

The output satisfies $V_0 = VR$ (where V_0 represent the input V) and $V'V = I$.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1668 <slepc4py/SLEPc/BV.pyx#L1668>`

orthogonalizeColumn(j)

Orthogonalize a column vector with respect to the previous ones.

Collective.

Parameters

j (*int*) – Index of the column to be orthogonalized.

Returns

- **norm** (*float*) – The norm of the resulting vector.
- **lindep** (*bool*) – Flag indicating that refinement did not improve the quality of orthogonalization.

Return type

tuple[*float*, *bool*]

Notes

This function applies an orthogonal projector to project vector V_j onto the orthogonal complement of the span of the columns $V[0..j-1]$, where $V[.]$ are the vectors of the BV. The columns $V[0..j-1]$ are assumed to be mutually orthonormal.

This routine does not normalize the resulting vector.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1602 <slepc4py/SLEPc/BV.pyx#L1602>`

orthogonalizeVec(v)

Orthogonalize a vector with respect to a set of vectors.

Collective.

Parameters

v (*Vec*) – Vector to be orthogonalized, modified on return.

Returns

- **norm** (*float*) – The norm of the resulting vector.
- **lindep** (*bool*) – Flag indicating that refinement did not improve the quality of orthogonalization.

Return type

tuple[*float*, *bool*]

Notes

This function applies an orthogonal projector to project vector v onto the orthogonal complement of the span of the columns of the BV.

This routine does not normalize the resulting vector.

:sources: [Source code at slepc4py/SLEPc/BV.pyx:1570](#) <slepc4py/SLEPc/BV.pyx#L1570>

orthonormalizeColumn(j , $replace=False$)

Orthonormalize a column vector with respect to the previous ones.

Collective.

This is equivalent to a call to [orthogonalizeColumn\(\)](#) followed by a call to [scaleColumn\(\)](#) with the reciprocal of the norm.

Parameters

- **j** ([int](#)) – Index of the column to be orthonormalized.
- **replace** ([bool](#)) – Whether it is allowed to set the vector randomly.

Returns

- **norm** ([float](#)) – The norm of the resulting vector.
- **lindep** ([bool](#)) – Flag indicating that refinement did not improve the quality of orthogonalization.

Return type

[tuple](#)[[float](#), [bool](#)]

:sources: [Source code at slepc4py/SLEPc/BV.pyx:1636](#) <slepc4py/SLEPc/BV.pyx#L1636>

resize(m , $copy=True$)

Change the number of columns.

Collective.

Parameters

- **m** ([int](#)) – The new number of columns.
- **copy** ([bool](#)) – A flag indicating whether current values should be kept.

Return type

[None](#)

Notes

Internal storage is reallocated. If `copy` is `True`, then the contents are copied to the leading part of the new space.

:sources: [Source code at slepc4py/SLEPc/BV.pyx:1451](#) <slepc4py/SLEPc/BV.pyx#L1451>

restoreColumn(j , v)

Restore a column obtained with [getColumn\(\)](#).

Logically collective.

Parameters

- **j** ([int](#)) – The index of the requested column.
- **v** ([Vec](#)) – The vector obtained with [getColumn\(\)](#).

Return type

None

Notes

The arguments must match the corresponding call to `getColumn()`.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1001 <slepc4py/SLEPc/BV.pyx#L1001>`

restoreMat(A)

Restore the Mat obtained with `getMat()`.

Logically collective.

Parameters

A (*Mat*) – The matrix obtained with `getMat()`.

Return type

None

Notes

A call to this function must match a previous call of `getMat()`. The effect is that the contents of the Mat are copied back to the BV internal data structures.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1044 <slepc4py/SLEPc/BV.pyx#L1044>`

scale(alpha)

Multiply the entries by a scalar value.

Logically collective.

Parameters

alpha (*Scalar*) – scaling factor.

Return type

None

Notes

All active columns (except the leading ones) are scaled.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:676 <slepc4py/SLEPc/BV.pyx#L676>`

scaleColumn(j, alpha)

Scale column *j* by *alpha*.

Logically collective.

Parameters

- **j** (*int*) – column number to be scaled.
- **alpha** (*Scalar*) – scaling factor.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:659 <slepc4py/SLEPc/BV.pyx#L659>`

setActiveColumns(l, k)

Set the columns that will be involved in operations.

Logically collective.

Parameters

- **l** (*int*) – The leading number of columns.
- **k** (*int*) – The active number of columns.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:625 <slepc4py/SLEPc/BV.pyx#L625>`

setDefiniteTolerance(*deftol*)

Set the tolerance to be used when checking a definite inner product.

Logically collective.

Parameters

deftol (*float*) – The tolerance.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:883 <slepc4py/SLEPc/BV.pyx#L883>`

setFromOptions()

Set BV options from the options database.

Collective.

Notes

To see all options, run your program with the `-help` option.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:443 <slepc4py/SLEPc/BV.pyx#L443>`

Return type

None

setLeadingDimension(*ld*)

Set the leading dimension.

Not collective.

Parameters

ld (*int*) – The leading dimension.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:363 <slepc4py/SLEPc/BV.pyx#L363>`

setMatMultMethod(*method*)

Set the method used for the `matMult()` operation.

Logically collective.

Parameters

method (*MatMultType*) – The method for the `matMult()` operation.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:550 <slepc4py/SLEPc/BV.pyx#L550>`

setMatrix(*mat*, *indef=False*)

Set the bilinear form to be used for inner products.

Collective.

Parameters

- **mat** (*Mat*) – The matrix of the inner product.
- **indef** (*bool*) – Whether the matrix is indefinite

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:588 <slepc4py/SLEPc/BV.pyx#L588>`

setNumConstraints(*nc*)

Set the number of constraints.

Logically collective.

Parameters

nc (*int*) – The number of constraints.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:781 <slepc4py/SLEPc/BV.pyx#L781>`

setOptionsPrefix(*prefix=None*)

Set the prefix used for searching for all BV options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all BV option requests.

Return type

None

Notes

A hyphen (-) must NOT be given at the beginning of the prefix name. The first character of all runtime options is AUTOMATICALLY the hyphen.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:392 <slepc4py/SLEPc/BV.pyx#L392>`

setOrthogonalization(*otype=None*, *refine=None*, *eta=None*, *block=None*)

Set the method used for the (block-)orthogonalization of vectors.

Logically collective.

Orthogonalization of vectors (classical or modified Gram-Schmidt with or without refinement), and for the block-orthogonalization (simultaneous orthogonalization of a set of vectors).

Parameters

- **otype** (*OrthogType* / *None*) – The type of orthogonalization technique.
- **refine** (*OrthogRefineType* / *None*) – The type of refinement.
- **eta** (*float* / *None*) – Parameter for selective refinement.
- **block** (*OrthogBlockType* / *None*) – The type of block orthogonalization.

Return type

None

Notes

The default settings work well for most problems.

The parameter `eta` should be a real value between 0 and 1 (or *DETERMINE*). The value of `eta` is used only when the refinement type is *BV.OrthogRefineType.IFNEEDED*.

When using several processors, *BV.OrthogType.MGS* is likely to result in bad scalability.

If the method set for block orthogonalization is GS, then the computation is done column by column with the vector orthogonalization.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:483 <slepc4py/SLEPc/BV.pyx#L483>`

setRandom()

Set the active columns of the BV to random numbers.

Logically collective.

Notes

All active columns (except the leading ones) are modified.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1473 <slepc4py/SLEPc/BV.pyx#L1473>`

Return type

None

setRandomColumn(*j*)

Set one column of the BV to random numbers.

Logically collective.

Parameters

j (*int*) – Column number to be set.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1509 <slepc4py/SLEPc/BV.pyx#L1509>`

setRandomCond(*condn*)

Set the columns of a BV to random numbers.

Logically collective.

The generated matrix has a prescribed condition number.

Parameters

condn (*float*) – Condition number.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1523 <slepc4py/SLEPc/BV.pyx#L1523>`

setRandomContext(*rnd*)

Set the `petsc4py.PETSc.Random` object associated with the BV.

Collective.

To be used in operations that need random numbers.

Parameters

rnd (*Random*) – The random number generator context.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1539 <slepc4py/SLEPc/BV.pyx#L1539>`

setRandomNormal()

Set the active columns of the BV to normal random numbers.

Logically collective.

Notes

All active columns (except the leading ones) are modified.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1485 <slepc4py/SLEPc/BV.pyx#L1485>`

Return type

None

setRandomSign()

Set the entries of a BV to values 1 or -1 with equal probability.

Logically collective.

Notes

All active columns (except the leading ones) are modified.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1497 <slepc4py/SLEPc/BV.pyx#L1497>`

Return type

None

setSizes(sizes, m)

Set the local and global sizes, and the number of columns.

Collective.

Parameters

- **sizes** (*LayoutSizeSpec*) – The global size *N* or a two-tuple (*n*, *N*) with the local and global sizes.
- **m** (*int*) – The number of columns.

Return type

None

Notes

Either *n* or *N* (but not both) can be `PETSc.DECIDE` or `None` to have it automatically set.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:303 <slepc4py/SLEPc/BV.pyx#L303>`

setSizesFromVec(w, m)

Set the local and global sizes, and the number of columns.

Collective.

Local and global sizes are specified indirectly by passing a template vector.

Parameters

- **w** (*Vec*) – The template vector.
- **m** (*int*) – The number of columns.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:327 <slepc4py/SLEPc/BV.pyx#L327>`

setType(*bv_type*)

Set the type for the BV object.

Logically collective.

Parameters

bv_type (*Type* / *str*) – The inner product type to be used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:273 <slepc4py/SLEPc/BV.pyx#L273>`

setVecType(*vec_type*)

Set the vector type.

Collective.

Parameters

vec_type (*petsc4py.PETSc.Vec.Type* / *str*) – Vector type used when creating vectors with *createVec*.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:825 <slepc4py/SLEPc/BV.pyx#L825>`

view(*viewer=None*)

Print the BV data structure.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/BV.pyx:150 <slepc4py/SLEPc/BV.pyx#L150>`

Attributes Documentation

column_size

Basis vectors column size.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1705 <slepc4py/SLEPc/BV.pyx#L1705>`

local_size

Basis vectors local size.

:sources: `Source code at slepc4py/SLEPc/BV.pyx:1700 <slepc4py/SLEPc/BV.pyx#L1700>`

size

Basis vectors global size.

:sources: ``Source code at slepc4py/SLEPc/BV.pyx:1695 <slepc4py/SLEPc/BV.pyx#L1695>``

sizes

Basis vectors local and global sizes, and the number of columns.

:sources: ``Source code at slepc4py/SLEPc/BV.pyx:1690 <slepc4py/SLEPc/BV.pyx#L1690>``

slepc4py.SLEPc.BVSVDMethod

class slepc4py.SLEPc.BVSVDMethod

Bases: `object`

BV methods for computing the SVD.

- **REFINE:** Based on the SVD of the cross product matrix $S^H S$, with refinement.
- **QR:** Based on the SVD of the triangular factor of $\text{qr}(S)$.
- **QR_CAA:** Variant of QR intended for use in communication-avoiding Arnoldi.

Attributes Summary

<code>QR</code>	Constant QR of type <code>int</code>
<code>QR_CAA</code>	Constant QR_CAA of type <code>int</code>
<code>REFINE</code>	Constant REFINE of type <code>int</code>

Attributes Documentation

QR: `int` = QR

Constant QR of type `int`

QR_CAA: `int` = QR_CAA

Constant QR_CAA of type `int`

REFINE: `int` = REFINE

Constant REFINE of type `int`

slepc4py.SLEPc.DS

class slepc4py.SLEPc.DS

Bases: `Object`

DS.

Enumerations

<code>MatType</code>	To refer to one of the matrices stored internally in DS.
<code>ParallelType</code>	DS parallel types.
<code>StateType</code>	DS state types.

continues on next page

Table 14 – continued from previous page

Type	DS type.
------	----------

slepc4py.SLEPc.DS.MatType**class** slepc4py.SLEPc.DS.MatTypeBases: `object`

To refer to one of the matrices stored internally in DS.

- *A*: first matrix of eigenproblem/singular value problem.
- *B*: second matrix of a generalized eigenproblem.
- *C*: third matrix of a quadratic eigenproblem.
- *T*: tridiagonal matrix.
- *D*: diagonal matrix.
- *Q*: orthogonal matrix of (right) Schur vectors.
- *Z*: orthogonal matrix of left Schur vectors.
- *X*: right eigenvectors.
- *Y*: left eigenvectors.
- *U*: left singular vectors.
- *V*: right singular vectors.
- *W*: workspace matrix.

Attributes Summary

<i>A</i>	Constant A of type <code>int</code>
<i>B</i>	Constant B of type <code>int</code>
<i>C</i>	Constant C of type <code>int</code>
<i>D</i>	Constant D of type <code>int</code>
<i>Q</i>	Constant Q of type <code>int</code>
<i>T</i>	Constant T of type <code>int</code>
<i>U</i>	Constant U of type <code>int</code>
<i>V</i>	Constant V of type <code>int</code>
<i>W</i>	Constant W of type <code>int</code>
<i>X</i>	Constant X of type <code>int</code>
<i>Y</i>	Constant Y of type <code>int</code>
<i>Z</i>	Constant Z of type <code>int</code>

Attributes Documentation**A:** `int` = AConstant A of type `int`**B:** `int` = BConstant B of type `int`

C: `int` = C
 Constant C of type `int`

D: `int` = D
 Constant D of type `int`

Q: `int` = Q
 Constant Q of type `int`

T: `int` = T
 Constant T of type `int`

U: `int` = U
 Constant U of type `int`

V: `int` = V
 Constant V of type `int`

W: `int` = W
 Constant W of type `int`

X: `int` = X
 Constant X of type `int`

Y: `int` = Y
 Constant Y of type `int`

Z: `int` = Z
 Constant Z of type `int`

slepc4py.SLEPc.DS.ParallelType

class slepc4py.SLEPc.DS.ParallelType

Bases: `object`

DS parallel types.

- *REDUNDANT*: Every process performs the computation redundantly.
- *SYNCHRONIZED*: The first process sends the result to the rest.
- *DISTRIBUTED*: Used in some cases to distribute the computation among processes.

Attributes Summary

<i>DISTRIBUTED</i>	Constant DISTRIBUTED of type <code>int</code>
<i>REDUNDANT</i>	Constant REDUNDANT of type <code>int</code>
<i>SYNCHRONIZED</i>	Constant SYNCHRONIZED of type <code>int</code>

Attributes Documentation

DISTRIBUTED: `int` = DISTRIBUTED
 Constant DISTRIBUTED of type `int`

REDUNDANT: `int` = REDUNDANT
 Constant REDUNDANT of type `int`

SYNCHRONIZED: `int` = `SYNCHRONIZED`

Constant `SYNCHRONIZED` of type `int`

`slepc4py.SLEPc.DS.StateType`

class `slepc4py.SLEPc.DS.StateType`

Bases: `object`

DS state types.

- `RAW`: Not processed yet.
- `INTERMEDIATE`: Reduced to Hessenberg or tridiagonal form (or equivalent).
- `CONDENSED`: Reduced to Schur or diagonal form (or equivalent).
- `TRUNCATED`: Condensed form truncated to a smaller size.

Attributes Summary

<code>CONDENSED</code>	Constant <code>CONDENSED</code> of type <code>int</code>
<code>INTERMEDIATE</code>	Constant <code>INTERMEDIATE</code> of type <code>int</code>
<code>RAW</code>	Constant <code>RAW</code> of type <code>int</code>
<code>TRUNCATED</code>	Constant <code>TRUNCATED</code> of type <code>int</code>

Attributes Documentation

CONDENSED: `int` = `CONDENSED`

Constant `CONDENSED` of type `int`

INTERMEDIATE: `int` = `INTERMEDIATE`

Constant `INTERMEDIATE` of type `int`

RAW: `int` = `RAW`

Constant `RAW` of type `int`

TRUNCATED: `int` = `TRUNCATED`

Constant `TRUNCATED` of type `int`

`slepc4py.SLEPc.DS.Type`

class `slepc4py.SLEPc.DS.Type`

Bases: `object`

DS type.

Attributes Summary

<code>GHEP</code>	Object <code>GHEP</code> of type <code>str</code>
<code>GHIEP</code>	Object <code>GHIEP</code> of type <code>str</code>
<code>GNHEP</code>	Object <code>GNHEP</code> of type <code>str</code>
<code>GSVD</code>	Object <code>GSVD</code> of type <code>str</code>
<code>HEP</code>	Object <code>HEP</code> of type <code>str</code>
<code>HSVD</code>	Object <code>HSVD</code> of type <code>str</code>

continues on next page

Table 18 – continued from previous page

<i>NEP</i>	Object NEP of type <i>str</i>
<i>NHEP</i>	Object NHEP of type <i>str</i>
<i>NHEPTS</i>	Object NHEPTS of type <i>str</i>
<i>PEP</i>	Object PEP of type <i>str</i>
<i>SVD</i>	Object SVD of type <i>str</i>

Attributes Documentation

GHEP: *str* = GHEP

Object GHEP of type *str*

GHIEP: *str* = GHIEP

Object GHIEP of type *str*

GNHEP: *str* = GNHEP

Object GNHEP of type *str*

GSVD: *str* = GSVD

Object GSVD of type *str*

HEP: *str* = HEP

Object HEP of type *str*

HSVD: *str* = HSVD

Object HSVD of type *str*

NEP: *str* = NEP

Object NEP of type *str*

NHEP: *str* = NHEP

Object NHEP of type *str*

NHEPTS: *str* = NHEPTS

Object NHEPTS of type *str*

PEP: *str* = PEP

Object PEP of type *str*

SVD: *str* = SVD

Object SVD of type *str*

Methods Summary

<i>allocate</i> (ld)	Allocate memory for internal storage or matrices in DS.
<i>appendOptionsPrefix</i> ([prefix])	Append to the prefix used for searching for all DS options in the database.
<i>cond</i> ()	Compute the inf-norm condition number of the first matrix.
<i>create</i> ([comm])	Create the DS object.
<i>destroy</i> ()	Destroy the DS object.
<i>duplicate</i> ()	Duplicate the DS object with the same type and dimensions.

continues on next page

Table 19 – continued from previous page

<code>getBlockSize()</code>	Get the block size.
<code>getCompact()</code>	Get the compact storage flag.
<code>getDimensions()</code>	Get the current dimensions.
<code>getExtraRow()</code>	Get the extra row flag.
<code>getGSVDDimensions()</code>	Get the number of columns and rows of a <i>DS</i> of type <i>GSVD</i> .
<code>getHSVDDimensions()</code>	Get the number of columns of a <i>DS</i> of type <i>HSVD</i> .
<code>getLeadingDimension()</code>	Get the leading dimension of the allocated matrices.
<code>getMat(matname)</code>	Get the requested matrix as a sequential dense Mat object.
<code>getMethod()</code>	Get the method currently used in the DS.
<code>getOptionsPrefix()</code>	Get the prefix used for searching for all DS options in the database.
<code>getPEPCoefficients()</code>	Get the polynomial basis coefficients of a <i>DS</i> of type <i>PEP</i> .
<code>getPEPDegree()</code>	Get the polynomial degree of a <i>DS</i> of type <i>PEP</i> .
<code>getParallel()</code>	Get the mode of operation in parallel runs.
<code>getRefined()</code>	Get the refined vectors flag.
<code>getSVDDimensions()</code>	Get the number of columns of a <i>DS</i> of type <i>SVD</i> .
<code>getState()</code>	Get the current state.
<code>getType()</code>	Get the DS type of this object.
<code>reset()</code>	Reset the DS object.
<code>restoreMat(matname, mat)</code>	Restore the previously seized matrix.
<code>setBlockSize(bs)</code>	Set the block size.
<code>setCompact(comp)</code>	Set the matrices' compact storage flag.
<code>setDimensions([n, l, k])</code>	Set the matrices sizes in the DS object.
<code>setExtraRow(ext)</code>	Set a flag to indicate that the matrix has one extra row.
<code>setFromOptions()</code>	Set DS options from the options database.
<code>setGSVDDimensions(m, p)</code>	Set the number of columns and rows of a <i>DS</i> of type <i>GSVD</i> .
<code>setHSVDDimensions(m)</code>	Set the number of columns of a <i>DS</i> of type <i>HSVD</i> .
<code>setIdentity(matname)</code>	Set the identity on the active part of a matrix.
<code>setMethod(meth)</code>	Set the method to be used to solve the problem.
<code>setOptionsPrefix([prefix])</code>	Set the prefix used for searching for all DS options in the database.
<code>setPEPCoefficients(pbc)</code>	Set the polynomial basis coefficients of a <i>DS</i> of type <i>PEP</i> .
<code>setPEPDegree(deg)</code>	Set the polynomial degree of a <i>DS</i> of type <i>PEP</i> .
<code>setParallel(pmode)</code>	Set the mode of operation in parallel runs.
<code>setRefined(ref)</code>	Set a flag to indicate that refined vectors must be computed.
<code>setSVDDimensions(m)</code>	Set the number of columns of a <i>DS</i> of type <i>SVD</i> .
<code>setState(state)</code>	Set the state of the DS object.
<code>setType(ds_type)</code>	Set the type for the DS object.
<code>solve()</code>	Solve the problem.
<code>truncate(n[, trim])</code>	Truncate the system represented in the DS object.
<code>updateExtraRow()</code>	Ensure that the extra row gets up-to-date after a call to <i>DS.solve()</i> .
<code>vectors([matname])</code>	Compute vectors associated to the dense system such as eigenvectors.
<code>view([viewer])</code>	Print the DS data structure.

Attributes Summary

<code>block_size</code>	The block size.
<code>compact</code>	Compact storage of matrices.
<code>extra_row</code>	If the matrix has one extra row.
<code>method</code>	The method to be used to solve the problem.
<code>parallel</code>	The mode of operation in parallel runs.
<code>refined</code>	If refined vectors must be computed.
<code>state</code>	The state of the DS object.

Methods Documentation

allocate(*ld*)

Allocate memory for internal storage or matrices in DS.

Logically collective.

Parameters

ld (*int*) – Leading dimension (maximum allowed dimension for the matrices, including the extra row if present).

Return type

None

:sources: `Source code at slepc4py/SLEPc/DS.pyx:245 <slepc4py/SLEPc/DS.pyx#L245>`

appendOptionsPrefix(*prefix=None*)

Append to the prefix used for searching for all DS options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all DS option requests.

Return type

None

:sources: `Source code at slepc4py/SLEPc/DS.pyx:190 <slepc4py/SLEPc/DS.pyx#L190>`

cond()

Compute the inf-norm condition number of the first matrix.

Logically collective.

Returns

Condition number.

Return type

float

:sources: `Source code at slepc4py/SLEPc/DS.pyx:654 <slepc4py/SLEPc/DS.pyx#L654>`

create(*comm=None*)

Create the DS object.

Collective.

Parameters

comm (*Comm* / *None*) – MPI communicator; if not provided, it defaults to all processes.

Return type*Self*

:sources: [Source code at slepc4py/SLEPc/DS.pyx:122 <slepc4py/SLEPc/DS.pyx#L122>](#)

destroy()

Destroy the DS object.

Collective.

:sources: [Source code at slepc4py/SLEPc/DS.pyx:104 <slepc4py/SLEPc/DS.pyx#L104>](#)

Return type*Self***duplicate()**

Duplicate the DS object with the same type and dimensions.

Collective.

:sources: [Source code at slepc4py/SLEPc/DS.pyx:233 <slepc4py/SLEPc/DS.pyx#L233>](#)

Return type*DS***getBlockSize()**

Get the block size.

Not collective.

Returns

The block size.

Return type*int*

:sources: [Source code at slepc4py/SLEPc/DS.pyx:409 <slepc4py/SLEPc/DS.pyx#L409>](#)

getCompact()

Get the compact storage flag.

Not collective.

Returns

The flag.

Return type*bool*

:sources: [Source code at slepc4py/SLEPc/DS.pyx:477 <slepc4py/SLEPc/DS.pyx#L477>](#)

getDimensions()

Get the current dimensions.

Not collective.

Returns

- **n** (*int*) – The new size.
- **l** (*int*) – Number of locked (inactive) leading columns.
- **k** (*int*) – Intermediate dimension (e.g., position of arrow).
- **t** (*int*) – Truncated length.

Return type

`tuple[int, int, int, int]`

:sources: `Source code at slepc4py/SLEPc/DS.pyx:371 <slepc4py/SLEPc/DS.pyx#L371>`

getExtraRow()

Get the extra row flag.

Not collective.

Returns

The flag.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/DS.pyx:516 <slepc4py/SLEPc/DS.pyx#L516>`

getGSVDDimensions()

Get the number of columns and rows of a *DS* of type *GSVD*.

Not collective.

Returns

- `m (int)` – The number of columns.
- `p (int)` – The number of rows for the second matrix.

Return type

`tuple[int, int]`

:sources: `Source code at slepc4py/SLEPc/DS.pyx:784 <slepc4py/SLEPc/DS.pyx#L784>`

getHSVDDimensions()

Get the number of columns of a *DS* of type *HSVD*.

Not collective.

Returns

The number of columns.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/DS.pyx:752 <slepc4py/SLEPc/DS.pyx#L752>`

getLeadingDimension()

Get the leading dimension of the allocated matrices.

Not collective.

Returns

Leading dimension (maximum allowed dimension for the matrices).

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/DS.pyx:260 <slepc4py/SLEPc/DS.pyx#L260>`

getMat(matname)

Get the requested matrix as a sequential dense Mat object.

Not collective.

Parameters

matname ([MatType](#)) – The requested matrix.

Returns

The matrix.

Return type

[petsc4py.PETSc.Mat](#)

:sources: [Source code at slepc4py/SLEPc/DS.pyx:599](#) <[slepc4py/SLEPc/DS.pyx#L599](#)>

getMethod()

Get the method currently used in the DS.

Not collective.

Returns

Identifier of the method.

Return type

[int](#)

:sources: [Source code at slepc4py/SLEPc/DS.pyx:438](#) <[slepc4py/SLEPc/DS.pyx#L438](#)>

getOptionsPrefix()

Get the prefix used for searching for all DS options in the database.

Not collective.

Returns

The prefix string set for this DS object.

Return type

[str](#)

:sources: [Source code at slepc4py/SLEPc/DS.pyx:205](#) <[slepc4py/SLEPc/DS.pyx#L205](#)>

getPEPCoefficients()

Get the polynomial basis coefficients of a *DS* of type *PEP*.

Not collective.

Returns

Coefficients.

Return type

[ArrayReal](#)

:sources: [Source code at slepc4py/SLEPc/DS.pyx:847](#) <[slepc4py/SLEPc/DS.pyx#L847](#)>

getPEPDegree()

Get the polynomial degree of a *DS* of type *PEP*.

Not collective.

Returns

The polynomial degree.

Return type

[int](#)

:sources: [Source code at slepc4py/SLEPc/DS.pyx:816](#) <[slepc4py/SLEPc/DS.pyx#L816](#)>

getParallel()

Get the mode of operation in parallel runs.

Not collective.

Returns

The parallel mode.

Return type

ParallelType

:sources: `Source code at slepc4py/SLEPc/DS.pyx:329 <slepc4py/SLEPc/DS.pyx#L329>`

getRefined()

Get the refined vectors flag.

Not collective.

Returns

The flag.

Return type

bool

:sources: `Source code at slepc4py/SLEPc/DS.pyx:556 <slepc4py/SLEPc/DS.pyx#L556>`

getSVDDimensions()

Get the number of columns of a *DS* of type *SVD*.

Not collective.

Returns

The number of columns.

Return type

int

:sources: `Source code at slepc4py/SLEPc/DS.pyx:723 <slepc4py/SLEPc/DS.pyx#L723>`

getState()

Get the current state.

Not collective.

Returns

The current state.

Return type

StateType

:sources: `Source code at slepc4py/SLEPc/DS.pyx:300 <slepc4py/SLEPc/DS.pyx#L300>`

getType()

Get the DS type of this object.

Not collective.

Returns

The direct solver type currently being used.

Return type

str

:sources: `Source code at slepc4py/SLEPc/DS.pyx:154 <slepc4py/SLEPc/DS.pyx#L154>`

reset()

Reset the DS object.

Collective.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:114 <slepc4py/SLEPc/DS.pyx#L114>`

Return type

None

restoreMat(matname, mat)

Restore the previously seized matrix.

Not collective.

Parameters

- **matname** (*MatType*) – The selected matrix.
- **mat** (*petsc4py.PETSc.Mat*) – The matrix previously obtained with *getMat()*.

Return type

None

:sources: `Source code at slepc4py/SLEPc/DS.pyx:621 <slepc4py/SLEPc/DS.pyx#L621>`

setBlockSize(bs)

Set the block size.

Logically collective.

Parameters

bs (*int*) – The block size.

Return type

None

:sources: `Source code at slepc4py/SLEPc/DS.pyx:395 <slepc4py/SLEPc/DS.pyx#L395>`

setCompact(comp)

Set the matrices' compact storage flag.

Logically collective.

Parameters

comp (*bool*) – True means compact storage.

Return type

None

Notes

Compact storage is used in some *DS* types such as *DS.Type.HEP* when the matrix is tridiagonal. This flag can be used to indicate whether the user provides the matrix entries via the compact form (the tridiagonal *DS.MatType.T*) or the non-compact one (*DS.MatType.A*).

The default is False.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:453 <slepc4py/SLEPc/DS.pyx#L453>`

setDimensions(n=None, l=None, k=None)

Set the matrices sizes in the DS object.

Logically collective.

Parameters

- **n** (*int* / *None*) – The new size.
- **l** (*int* / *None*) – Number of locked (inactive) leading columns.
- **k** (*int* / *None*) – Intermediate dimension (e.g., position of arrow).

Return type

None

Notes

The internal arrays are not reallocated.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:344 <slepc4py/SLEPc/DS.pyx#L344>`

setExtraRow(*ext*)

Set a flag to indicate that the matrix has one extra row.

Logically collective.

Parameters

ext (*bool*) – True if the matrix has extra row.

Return type

None

Notes

In Krylov methods it is useful that the matrix representing the direct solver has one extra row, i.e., has dimension $(n + 1)n$. If this flag is activated, all transformations applied to the right of the matrix also affect this additional row. In that case, $(n + 1)$ must be less or equal than the leading dimension.

The default is `False`.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:492 <slepc4py/SLEPc/DS.pyx#L492>`

setFromOptions()

Set DS options from the options database.

Collective.

Notes

To see all options, run your program with the `-help` option.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:220 <slepc4py/SLEPc/DS.pyx#L220>`

Return type

None

setGSVDDimensions(*m*, *p*)

Set the number of columns and rows of a *DS* of type *GSVD*.

Logically collective.

Parameters

- **m** (*int*) – The number of columns.
- **p** (*int*) – The number of rows for the second matrix.

Return type

None

:sources: [Source code at slepc4py/SLEPc/DS.pyx:767 <slepc4py/SLEPc/DS.pyx#L767>](#)

setHSVDDimensions(*m*)

Set the number of columns of a *DS* of type *HSVD*.

Logically collective.

Parameters

m (*int*) – The number of columns.

Return type

None

:sources: [Source code at slepc4py/SLEPc/DS.pyx:738 <slepc4py/SLEPc/DS.pyx#L738>](#)

setIdentity(*matname*)

Set the identity on the active part of a matrix.

Logically collective.

Parameters

matname (*MatType*) – The requested matrix.

Return type

None

:sources: [Source code at slepc4py/SLEPc/DS.pyx:638 <slepc4py/SLEPc/DS.pyx#L638>](#)

setMethod(*meth*)

Set the method to be used to solve the problem.

Logically collective.

Parameters

meth (*int*) – An index identifying the method.

Return type

None

:sources: [Source code at slepc4py/SLEPc/DS.pyx:424 <slepc4py/SLEPc/DS.pyx#L424>](#)

setOptionsPrefix(*prefix=None*)

Set the prefix used for searching for all DS options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all DS option requests.

Return type

None

Notes

A hyphen (-) must NOT be given at the beginning of the prefix name. The first character of all runtime options is AUTOMATICALLY the hyphen.

:sources: [Source code at slepc4py/SLEPc/DS.pyx:169 <slepc4py/SLEPc/DS.pyx#L169>](#)

setPEPCoefficients(*pbc*)

Set the polynomial basis coefficients of a *DS* of type *PEP*.

Logically collective.

Parameters**pb** (*Sequence*[*float*]) – Coefficients.**Return type***None***:sources:** `Source code at slepc4py/SLEPc/DS.pyx:831 <slepc4py/SLEPc/DS.pyx#L831>`**setPEPDegree**(*deg*)Set the polynomial degree of a *DS* of type *PEP*.

Logically collective.

Parameters**deg** (*int*) – The polynomial degree.**Return type***None***:sources:** `Source code at slepc4py/SLEPc/DS.pyx:802 <slepc4py/SLEPc/DS.pyx#L802>`**setParallel**(*pmode*)

Set the mode of operation in parallel runs.

Logically collective.

Parameters**pmode** (*ParallelType*) – The parallel mode.**Return type***None***:sources:** `Source code at slepc4py/SLEPc/DS.pyx:315 <slepc4py/SLEPc/DS.pyx#L315>`**setRefined**(*ref*)

Set a flag to indicate that refined vectors must be computed.

Logically collective.

Parameters**ref** (*bool*) – True if refined vectors must be used.**Return type***None***Notes**

Normally the vectors returned in *DS.MatType.X* are eigenvectors of the projected matrix. With this flag activated, *vectors()* will return the right singular vector of the smallest singular value of matrix *At* – *thetaI*, where *At* is the extended $(n + 1) \times n$ matrix and *theta* is the Ritz value. This is used in the refined Ritz approximation.

The default is False.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:531 <slepc4py/SLEPc/DS.pyx#L531>`**setSVDDimensions**(*m*)Set the number of columns of a *DS* of type *SVD*.

Logically collective.

Parameters**m** (*int*) – The number of columns.

Return type

None

:sources: [Source code at slepc4py/SLEPc/DS.pyx:709 <slepc4py/SLEPc/DS.pyx#L709>](#)

setState(*state*)

Set the state of the DS object.

Logically collective.

Parameters

state (*StateType*) – The new state.

Return type

None

Notes

The state indicates that the dense system is in an initial state (raw), in an intermediate state (such as tridiagonal, Hessenberg or Hessenberg-triangular), in a condensed state (such as diagonal, Schur or generalized Schur), or in a truncated state.

This function is normally used to return to the raw state when the condensed structure is destroyed.

:sources: [Source code at slepc4py/SLEPc/DS.pyx:275 <slepc4py/SLEPc/DS.pyx#L275>](#)

setType(*ds_type*)

Set the type for the DS object.

Logically collective.

Parameters

ds_type (*Type* / *str*) – The direct solver type to be used.

Return type

None

:sources: [Source code at slepc4py/SLEPc/DS.pyx:139 <slepc4py/SLEPc/DS.pyx#L139>](#)

solve()

Solve the problem.

Logically collective.

Returns

Eigenvalues or singular values.

Return type

ArrayScalar

:sources: [Source code at slepc4py/SLEPc/DS.pyx:669 <slepc4py/SLEPc/DS.pyx#L669>](#)

truncate(*n*, *trim=False*)

Truncate the system represented in the DS object.

Logically collective.

Parameters

- **n** (*int*) – The new size.
- **trim** (*bool*) – A flag to indicate if the factorization must be trimmed.

Return type

None

:sources: `Source code at slepc4py/SLEPc/DS.pyx:571 <slepc4py/SLEPc/DS.pyx#L571>`

updateExtraRow()

Ensure that the extra row gets up-to-date after a call to `DS.solve()`.

Logically collective.

Perform all necessary operations so that the extra row gets up-to-date after a call to `DS.solve()`.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:588 <slepc4py/SLEPc/DS.pyx#L588>`

Return type

None

vectors(matname=X)

Compute vectors associated to the dense system such as eigenvectors.

Logically collective.

Parameters

matname (`DS.MatType` enumerate) – The matrix, used to indicate which vectors are required.

Return type

None

:sources: `Source code at slepc4py/SLEPc/DS.pyx:693 <slepc4py/SLEPc/DS.pyx#L693>`

view(viewer=None)

Print the DS data structure.

Collective.

Parameters

viewer (`Viewer` / `None`) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/DS.pyx:89 <slepc4py/SLEPc/DS.pyx#L89>`

Attributes Documentation

block_size

The block size.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:885 <slepc4py/SLEPc/DS.pyx#L885>`

compact

Compact storage of matrices.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:899 <slepc4py/SLEPc/DS.pyx#L899>`

extra_row

If the matrix has one extra row.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:906 <slepc4py/SLEPc/DS.pyx#L906>`

method

The method to be used to solve the problem.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:892 <slepc4py/SLEPc/DS.pyx#L892>`

parallel

The mode of operation in parallel runs.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:878 <slepc4py/SLEPc/DS.pyx#L878>`

refined

If refined vectors must be computed.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:913 <slepc4py/SLEPc/DS.pyx#L913>`

state

The state of the DS object.

:sources: `Source code at slepc4py/SLEPc/DS.pyx:871 <slepc4py/SLEPc/DS.pyx#L871>`

slepc4py.SLEPc.EPS

class slepc4py.SLEPc.EPS

Bases: `Object`

EPS.

Enumerations

<i>Balance</i>	EPS type of balancing used for non-Hermitian problems.
<i>CISSExtraction</i>	EPS CISS extraction technique.
<i>CISSQuadRule</i>	EPS CISS quadrature rule.
<i>Conv</i>	EPS convergence test.
<i>ConvergedReason</i>	EPS convergence reasons.
<i>ErrorType</i>	EPS error type to assess accuracy of computed solutions.
<i>Extraction</i>	EPS extraction technique.
<i>KrylovSchurBSEType</i>	EPS Krylov-Schur method for BSE problems.
<i>LanczosReorthogType</i>	EPS Lanczos reorthogonalization type.
<i>PowerShiftType</i>	EPS Power shift type.
<i>ProblemType</i>	EPS problem type.
<i>Stop</i>	EPS stopping test.
<i>Type</i>	EPS type.
<i>Which</i>	EPS desired part of spectrum.

slepc4py.SLEPc.EPS.Balance

class slepc4py.SLEPc.EPS.Balance

Bases: `object`

EPS type of balancing used for non-Hermitian problems.

- *NONE*: None.
- *ONESIDE*: One-sided balancing.
- *TWOSIDE*: Two-sided balancing.
- *USER*: User-provided balancing matrices.

Attributes Summary

<i>NONE</i>	Constant NONE of type <code>int</code>
<i>ONESIDE</i>	Constant ONESIDE of type <code>int</code>
<i>TWOSIDE</i>	Constant TWOSIDE of type <code>int</code>
<i>USER</i>	Constant USER of type <code>int</code>

Attributes Documentation

NONE: `int` = NONE

Constant NONE of type `int`

ONESIDE: `int` = ONESIDE

Constant ONESIDE of type `int`

TWOSIDE: `int` = TWOSIDE

Constant TWOSIDE of type `int`

USER: `int` = USER

Constant USER of type `int`

`slepc4py.SLEPc.EPS.CISSExtraction`

class `slepc4py.SLEPc.EPS.CISSExtraction`

Bases: `object`

EPS CISS extraction technique.

- *RITZ*: Ritz extraction.
- *HANKEL*: Extraction via Hankel eigenproblem.

Attributes Summary

<i>HANKEL</i>	Constant HANKEL of type <code>int</code>
<i>RITZ</i>	Constant RITZ of type <code>int</code>

Attributes Documentation

HANKEL: `int` = HANKEL

Constant HANKEL of type `int`

RITZ: `int` = RITZ

Constant RITZ of type `int`

`slepc4py.SLEPc.EPS.CISSQuadRule`

class `slepc4py.SLEPc.EPS.CISSQuadRule`

Bases: `object`

EPS CISS quadrature rule.

- *TRAPEZOIDAL*: Trapezoidal rule.
- *CHEBYSHEV*: Chebyshev points.

Attributes Summary

<i>CHEBYSHEV</i>	Constant CHEBYSHEV of type <code>int</code>
<i>TRAPEZOIDAL</i>	Constant TRAPEZOIDAL of type <code>int</code>

Attributes Documentation

CHEBYSHEV: `int` = CHEBYSHEV

Constant CHEBYSHEV of type `int`

TRAPEZOIDAL: `int` = TRAPEZOIDAL

Constant TRAPEZOIDAL of type `int`

`slepc4py.SLEPc.EPS.Conv`

class `slepc4py.SLEPc.EPS.Conv`

Bases: `object`

EPS convergence test.

- *ABS*: Absolute convergence test.
- *REL*: Convergence test relative to the eigenvalue.
- *NORM*: Convergence test relative to the matrix norms.
- *USER*: User-defined convergence test.

Attributes Summary

<i>ABS</i>	Constant ABS of type <code>int</code>
<i>NORM</i>	Constant NORM of type <code>int</code>
<i>REL</i>	Constant REL of type <code>int</code>
<i>USER</i>	Constant USER of type <code>int</code>

Attributes Documentation

ABS: `int` = ABS

Constant ABS of type `int`

NORM: `int` = NORM

Constant NORM of type `int`

REL: `int` = REL

Constant REL of type `int`

USER: `int` = USER

Constant USER of type `int`

`slepc4py.SLEPc.EPS.ConvergedReason`

class `slepc4py.SLEPc.EPS.ConvergedReason`

Bases: `object`

EPS convergence reasons.

- *CONVERGED_TOL*: All eigenpairs converged to requested tolerance.
- *CONVERGED_USER*: User-defined convergence criterion satisfied.
- *DIVERGED_ITS*: Maximum number of iterations exceeded.
- *DIVERGED_BREAKDOWN*: Solver failed due to breakdown.
- *DIVERGED_SYMMETRY_LOST*: Lanczos-type method could not preserve symmetry.
- *CONVERGED_ITERATING*: Iteration not finished yet.

Attributes Summary

<i>CONVERGED_ITERATING</i>	Constant <i>CONVERGED_ITERATING</i> of type <i>int</i>
<i>CONVERGED_TOL</i>	Constant <i>CONVERGED_TOL</i> of type <i>int</i>
<i>CONVERGED_USER</i>	Constant <i>CONVERGED_USER</i> of type <i>int</i>
<i>DIVERGED_BREAKDOWN</i>	Constant <i>DIVERGED_BREAKDOWN</i> of type <i>int</i>
<i>DIVERGED_ITS</i>	Constant <i>DIVERGED_ITS</i> of type <i>int</i>
<i>DIVERGED_SYMMETRY_LOST</i>	Constant <i>DIVERGED_SYMMETRY_LOST</i> of type <i>int</i>
<i>ITERATING</i>	Constant <i>ITERATING</i> of type <i>int</i>

Attributes Documentation

CONVERGED_ITERATING: *int* = CONVERGED_ITERATING

Constant *CONVERGED_ITERATING* of type *int*

CONVERGED_TOL: *int* = CONVERGED_TOL

Constant *CONVERGED_TOL* of type *int*

CONVERGED_USER: *int* = CONVERGED_USER

Constant *CONVERGED_USER* of type *int*

DIVERGED_BREAKDOWN: *int* = DIVERGED_BREAKDOWN

Constant *DIVERGED_BREAKDOWN* of type *int*

DIVERGED_ITS: *int* = DIVERGED_ITS

Constant *DIVERGED_ITS* of type *int*

DIVERGED_SYMMETRY_LOST: *int* = DIVERGED_SYMMETRY_LOST

Constant *DIVERGED_SYMMETRY_LOST* of type *int*

ITERATING: *int* = ITERATING

Constant *ITERATING* of type *int*

slepc4py.SLEPc.EPS.ErrorType

class slepc4py.SLEPc.EPS.ErrorType

Bases: *object*

EPS error type to assess accuracy of computed solutions.

- *ABSOLUTE*: Absolute error.
- *RELATIVE*: Relative error.
- *BACKWARD*: Backward error.

Attributes Summary

<i>ABSOLUTE</i>	Constant ABSOLUTE of type <i>int</i>
<i>BACKWARD</i>	Constant BACKWARD of type <i>int</i>
<i>RELATIVE</i>	Constant RELATIVE of type <i>int</i>

Attributes Documentation

ABSOLUTE: *int* = ABSOLUTE

Constant ABSOLUTE of type *int*

BACKWARD: *int* = BACKWARD

Constant BACKWARD of type *int*

RELATIVE: *int* = RELATIVE

Constant RELATIVE of type *int*

slepc4py.SLEPc.EPS.Extraction

class slepc4py.SLEPc.EPS.Extraction

Bases: *object*

EPS extraction technique.

- *RITZ*: Standard Rayleigh-Ritz extraction.
- *HARMONIC*: Harmonic extraction.
- *HARMONIC_RELATIVE*: Harmonic extraction relative to the eigenvalue.
- *HARMONIC_RIGHT*: Harmonic extraction for rightmost eigenvalues.
- *HARMONIC_LARGEST*: Harmonic extraction for largest magnitude (without target).
- *REFINED*: Refined extraction.
- *REFINED_HARMONIC*: Refined harmonic extraction.

Attributes Summary

<i>HARMONIC</i>	Constant HARMONIC of type <i>int</i>
<i>HARMONIC_LARGEST</i>	Constant HARMONIC_LARGEST of type <i>int</i>
<i>HARMONIC_RELATIVE</i>	Constant HARMONIC_RELATIVE of type <i>int</i>
<i>HARMONIC_RIGHT</i>	Constant HARMONIC_RIGHT of type <i>int</i>
<i>REFINED</i>	Constant REFINED of type <i>int</i>
<i>REFINED_HARMONIC</i>	Constant REFINED_HARMONIC of type <i>int</i>
<i>RITZ</i>	Constant RITZ of type <i>int</i>

Attributes Documentation

HARMONIC: *int* = HARMONIC

Constant HARMONIC of type *int*

HARMONIC_LARGEST: *int* = HARMONIC_LARGEST

Constant HARMONIC_LARGEST of type *int*

HARMONIC_RELATIVE: `int` = **HARMONIC_RELATIVE**

Constant HARMONIC_RELATIVE of type `int`

HARMONIC_RIGHT: `int` = **HARMONIC_RIGHT**

Constant HARMONIC_RIGHT of type `int`

REFINED: `int` = **REFINED**

Constant REFINED of type `int`

REFINED_HARMONIC: `int` = **REFINED_HARMONIC**

Constant REFINED_HARMONIC of type `int`

RITZ: `int` = **RITZ**

Constant RITZ of type `int`

slepc4py.SLEPc.EPS.KrylovSchurBSEType

class slepc4py.SLEPc.EPS.KrylovSchurBSEType

Bases: `object`

EPS Krylov-Schur method for BSE problems.

- *SHAO*: Lanczos recurrence for H square.
- *GRUNING*: Lanczos recurrence for H.
- *PROJECTEDBSE*: Lanczos where the projected problem has BSE structure.

Attributes Summary

<i>GRUNING</i>	Constant GRUNING of type <code>int</code>
<i>PROJECTEDBSE</i>	Constant PROJECTEDBSE of type <code>int</code>
<i>SHAO</i>	Constant SHAO of type <code>int</code>

Attributes Documentation

GRUNING: `int` = **GRUNING**

Constant GRUNING of type `int`

PROJECTEDBSE: `int` = **PROJECTEDBSE**

Constant PROJECTEDBSE of type `int`

SHAO: `int` = **SHAO**

Constant SHAO of type `int`

slepc4py.SLEPc.EPS.LanczosReorthogType

class slepc4py.SLEPc.EPS.LanczosReorthogType

Bases: `object`

EPS Lanczos reorthogonalization type.

- *LOCAL*: Local reorthogonalization only.
- *FULL*: Full reorthogonalization.
- *SELECTIVE*: Selective reorthogonalization.

- *PERIODIC*: Periodic reorthogonalization.
- *PARTIAL*: Partial reorthogonalization.
- *DELAYED*: Delayed reorthogonalization.

Attributes Summary

<i>DELAYED</i>	Constant DELAYED of type <code>int</code>
<i>FULL</i>	Constant FULL of type <code>int</code>
<i>LOCAL</i>	Constant LOCAL of type <code>int</code>
<i>PARTIAL</i>	Constant PARTIAL of type <code>int</code>
<i>PERIODIC</i>	Constant PERIODIC of type <code>int</code>
<i>SELECTIVE</i>	Constant SELECTIVE of type <code>int</code>

Attributes Documentation

DELAYED: `int` = DELAYED

Constant DELAYED of type `int`

FULL: `int` = FULL

Constant FULL of type `int`

LOCAL: `int` = LOCAL

Constant LOCAL of type `int`

PARTIAL: `int` = PARTIAL

Constant PARTIAL of type `int`

PERIODIC: `int` = PERIODIC

Constant PERIODIC of type `int`

SELECTIVE: `int` = SELECTIVE

Constant SELECTIVE of type `int`

slepc4py.SLEPc.EPS.PowerShiftType

class slepc4py.SLEPc.EPS.PowerShiftType

Bases: `object`

EPS Power shift type.

- *CONSTANT*: Constant shift.
- *RAYLEIGH*: Rayleigh quotient.
- *WILKINSON*: Wilkinson shift.

Attributes Summary

<i>CONSTANT</i>	Constant CONSTANT of type <code>int</code>
<i>RAYLEIGH</i>	Constant RAYLEIGH of type <code>int</code>
<i>WILKINSON</i>	Constant WILKINSON of type <code>int</code>

Attributes Documentation

CONSTANT: `int` = CONSTANT

Constant CONSTANT of type `int`

RAYLEIGH: `int` = RAYLEIGH

Constant RAYLEIGH of type `int`

WILKINSON: `int` = WILKINSON

Constant WILKINSON of type `int`

slepc4py.SLEPc.EPS.ProblemType

class slepc4py.SLEPc.EPS.ProblemType

Bases: `object`

EPS problem type.

- *HEP*: Hermitian eigenproblem.
- *NHEP*: Non-Hermitian eigenproblem.
- *GHEP*: Generalized Hermitian eigenproblem.
- *GNHEP*: Generalized Non-Hermitian eigenproblem.
- ***PGNHEP*: Generalized Non-Hermitian eigenproblem**
with positive definite B .
- *GHIEP*: Generalized Hermitian-indefinite eigenproblem.
- *BSE*: Structured Bethe-Salpeter eigenproblem.
- *HAMILT*: Hamiltonian eigenproblem.

Attributes Summary

<i>BSE</i>	Constant BSE of type <code>int</code>
<i>GHEP</i>	Constant GHEP of type <code>int</code>
<i>GHIEP</i>	Constant GHIEP of type <code>int</code>
<i>GNHEP</i>	Constant GNHEP of type <code>int</code>
<i>HAMILT</i>	Constant HAMILT of type <code>int</code>
<i>HEP</i>	Constant HEP of type <code>int</code>
<i>NHEP</i>	Constant NHEP of type <code>int</code>
<i>PGNHEP</i>	Constant PGNHEP of type <code>int</code>

Attributes Documentation

BSE: `int` = BSE

Constant BSE of type `int`

GHEP: `int` = GHEP

Constant GHEP of type `int`

GHIEP: `int` = GHIEP

Constant GHIEP of type `int`

GNHEP: `int` = GNHEP

Constant GNHEP of type `int`

HAMILT: `int` = HAMILT

Constant HAMILT of type `int`

HEP: `int` = HEP

Constant HEP of type `int`

NHEP: `int` = NHEP

Constant NHEP of type `int`

PGNHEP: `int` = PGNHEP

Constant PGNHEP of type `int`

slepc4py.SLEPc.EPS.Stop

class slepc4py.SLEPc.EPS.Stop

Bases: `object`

EPS stopping test.

- *BASIC*: Default stopping test.
- *USER*: User-defined stopping test.
- *THRESHOLD*: Threshold stopping test.

Attributes Summary

<i>BASIC</i>	Constant BASIC of type <code>int</code>
<i>THRESHOLD</i>	Constant THRESHOLD of type <code>int</code>
<i>USER</i>	Constant USER of type <code>int</code>

Attributes Documentation

BASIC: `int` = BASIC

Constant BASIC of type `int`

THRESHOLD: `int` = THRESHOLD

Constant THRESHOLD of type `int`

USER: `int` = USER

Constant USER of type `int`

slepc4py.SLEPc.EPS.Type

class slepc4py.SLEPc.EPS.Type

Bases: `object`

EPS type.

Native sparse eigensolvers.

- *POWER*: Power Iteration, Inverse Iteration, RQI.
- *SUBSPACE*: Subspace Iteration.

- *ARNOLDI*: Arnoldi.
- *LANCZOS*: Lanczos.
- *KRYLOV SCHUR*: Krylov-Schur (default).
- *GD*: Generalized Davidson.
- *JD*: Jacobi-Davidson.
- *RQCG*: Rayleigh Quotient Conjugate Gradient.
- *LOBPCG*: Locally Optimal Block Preconditioned Conjugate Gradient.
- *CISS*: Contour Integral Spectrum Slicing.
- *LYAPII*: Lyapunov inverse iteration.
- *LAPACK*: Wrappers to dense eigensolvers in Lapack.

Wrappers to external eigensolvers (should be enabled during installation of SLEPc)

- *ARPACK*:
- *BLOPEX*:
- *PRIMME*:
- *FEAST*:
- *SCALAPACK*:
- *ELPA*:
- *ELEMENTAL*:
- *EVSL*:
- *CHASE*:

Attributes Summary

<i>ARNOLDI</i>	Object <i>ARNOLDI</i> of type <i>str</i>
<i>ARPACK</i>	Object <i>ARPACK</i> of type <i>str</i>
<i>BLOPEX</i>	Object <i>BLOPEX</i> of type <i>str</i>
<i>CHASE</i>	Object <i>CHASE</i> of type <i>str</i>
<i>CISS</i>	Object <i>CISS</i> of type <i>str</i>
<i>ELEMENTAL</i>	Object <i>ELEMENTAL</i> of type <i>str</i>
<i>ELPA</i>	Object <i>ELPA</i> of type <i>str</i>
<i>EVSL</i>	Object <i>EVSL</i> of type <i>str</i>
<i>FEAST</i>	Object <i>FEAST</i> of type <i>str</i>
<i>GD</i>	Object <i>GD</i> of type <i>str</i>
<i>JD</i>	Object <i>JD</i> of type <i>str</i>
<i>KRYLOV SCHUR</i>	Object <i>KRYLOV SCHUR</i> of type <i>str</i>
<i>LANCZOS</i>	Object <i>LANCZOS</i> of type <i>str</i>
<i>LAPACK</i>	Object <i>LAPACK</i> of type <i>str</i>
<i>LOBPCG</i>	Object <i>LOBPCG</i> of type <i>str</i>
<i>LYAPII</i>	Object <i>LYAPII</i> of type <i>str</i>
<i>POWER</i>	Object <i>POWER</i> of type <i>str</i>
<i>PRIMME</i>	Object <i>PRIMME</i> of type <i>str</i>
<i>RQCG</i>	Object <i>RQCG</i> of type <i>str</i>
<i>SCALAPACK</i>	Object <i>SCALAPACK</i> of type <i>str</i>

continues on next page

Table 34 – continued from previous page

<i>SUBSPACE</i>	Object SUBSPACE of type <i>str</i>
Attributes Documentation	
ARNOLDI: <i>str</i> = ARNOLDI	Object ARNOLDI of type <i>str</i>
ARPACK: <i>str</i> = ARPACK	Object ARPACK of type <i>str</i>
BLOPEX: <i>str</i> = BLOPEX	Object BLOPEX of type <i>str</i>
CHASE: <i>str</i> = CHASE	Object CHASE of type <i>str</i>
CISS: <i>str</i> = CISS	Object CISS of type <i>str</i>
ELEMENTAL: <i>str</i> = ELEMENTAL	Object ELEMENTAL of type <i>str</i>
ELPA: <i>str</i> = ELPA	Object ELPA of type <i>str</i>
EVSL: <i>str</i> = EVSL	Object EVSL of type <i>str</i>
FEAST: <i>str</i> = FEAST	Object FEAST of type <i>str</i>
GD: <i>str</i> = GD	Object GD of type <i>str</i>
JD: <i>str</i> = JD	Object JD of type <i>str</i>
KRYLOVSHUR: <i>str</i> = KRYLOVSHUR	Object KRYLOVSHUR of type <i>str</i>
LANCZOS: <i>str</i> = LANCZOS	Object LANCZOS of type <i>str</i>
LAPACK: <i>str</i> = LAPACK	Object LAPACK of type <i>str</i>
LOBPCG: <i>str</i> = LOBPCG	Object LOBPCG of type <i>str</i>
LYAPII: <i>str</i> = LYAPII	Object LYAPII of type <i>str</i>
POWER: <i>str</i> = POWER	Object POWER of type <i>str</i>
PRIMME: <i>str</i> = PRIMME	Object PRIMME of type <i>str</i>

RQCG: `str = RQCG`

Object RQCG of type `str`

SCALAPACK: `str = SCALAPACK`

Object SCALAPACK of type `str`

SUBSPACE: `str = SUBSPACE`

Object SUBSPACE of type `str`

slepc4py.SLEPc.EPS.Which

class `slepc4py.SLEPc.EPS.Which`

Bases: `object`

EPS desired part of spectrum.

- *LARGEST_MAGNITUDE*: Largest magnitude (default).
- *SMALLEST_MAGNITUDE*: Smallest magnitude.
- *LARGEST_REAL*: Largest real parts.
- *SMALLEST_REAL*: Smallest real parts.
- *LARGEST_IMAGINARY*: Largest imaginary parts in magnitude.
- *SMALLEST_IMAGINARY*: Smallest imaginary parts in magnitude.
- *TARGET_MAGNITUDE*: Closest to target (in magnitude).
- *TARGET_REAL*: Real part closest to target.
- *TARGET_IMAGINARY*: Imaginary part closest to target.
- *ALL*: All eigenvalues in an interval.
- *USER*: User defined selection.

Attributes Summary

<i>ALL</i>	Constant ALL of type <code>int</code>
<i>LARGEST_IMAGINARY</i>	Constant LARGEST_IMAGINARY of type <code>int</code>
<i>LARGEST_MAGNITUDE</i>	Constant LARGEST_MAGNITUDE of type <code>int</code>
<i>LARGEST_REAL</i>	Constant LARGEST_REAL of type <code>int</code>
<i>SMALLEST_IMAGINARY</i>	Constant SMALLEST_IMAGINARY of type <code>int</code>
<i>SMALLEST_MAGNITUDE</i>	Constant SMALLEST_MAGNITUDE of type <code>int</code>
<i>SMALLEST_REAL</i>	Constant SMALLEST_REAL of type <code>int</code>
<i>TARGET_IMAGINARY</i>	Constant TARGET_IMAGINARY of type <code>int</code>
<i>TARGET_MAGNITUDE</i>	Constant TARGET_MAGNITUDE of type <code>int</code>
<i>TARGET_REAL</i>	Constant TARGET_REAL of type <code>int</code>
<i>USER</i>	Constant USER of type <code>int</code>

Attributes Documentation

ALL: `int = ALL`

Constant ALL of type `int`

LARGEST_IMAGINARY: `int = LARGEST_IMAGINARY`

Constant LARGEST_IMAGINARY of type `int`

LARGEST_MAGNITUDE: `int` = LARGEST_MAGNITUDE
Constant LARGEST_MAGNITUDE of type `int`

LARGEST_REAL: `int` = LARGEST_REAL
Constant LARGEST_REAL of type `int`

SMALLEST_IMAGINARY: `int` = SMALLEST_IMAGINARY
Constant SMALLEST_IMAGINARY of type `int`

SMALLEST_MAGNITUDE: `int` = SMALLEST_MAGNITUDE
Constant SMALLEST_MAGNITUDE of type `int`

SMALLEST_REAL: `int` = SMALLEST_REAL
Constant SMALLEST_REAL of type `int`

TARGET_IMAGINARY: `int` = TARGET_IMAGINARY
Constant TARGET_IMAGINARY of type `int`

TARGET_MAGNITUDE: `int` = TARGET_MAGNITUDE
Constant TARGET_MAGNITUDE of type `int`

TARGET_REAL: `int` = TARGET_REAL
Constant TARGET_REAL of type `int`

USER: `int` = USER
Constant USER of type `int`

Methods Summary

<i><code>appendOptionsPrefix([prefix])</code></i>	Append to the prefix used for searching for all EPS options in the database.
<i><code>cancelMonitor()</code></i>	Clear all monitors for an <i>EPS</i> object.
<i><code>computeError(i[, etype])</code></i>	Compute the error associated with the i-th computed eigenpair.
<i><code>create([comm])</code></i>	Create the EPS object.
<i><code>destroy()</code></i>	Destroy the EPS object.
<i><code>errorView([etype, viewer])</code></i>	Display the errors associated with the computed solution.
<i><code>getArnoldiDelayed()</code></i>	Get the type of reorthogonalization used during the Arnoldi iteration.
<i><code>getBV()</code></i>	Get the basis vector objects associated to the eigensolver.
<i><code>getBalance()</code></i>	Get the balancing type used by the EPS, and the associated parameters.
<i><code>getCISSExtraction()</code></i>	Get the extraction technique used in the CISS solver.
<i><code>getCISSKSPs()</code></i>	Get the array of linear solver objects associated with the CISS solver.
<i><code>getCISSQuadRule()</code></i>	Get the quadrature rule used in the CISS solver.
<i><code>getCISSRefinement()</code></i>	Get the values of various refinement parameters in the CISS solver.
<i><code>getCISSSizes()</code></i>	Get the values of various size parameters in the CISS solver.
<i><code>getCISSThreshold()</code></i>	Get the values of various threshold parameters in the CISS solver.

continues on next page

Table 36 – continued from previous page

<code>getCISSUseST()</code>	Get the flag indicating the use of the <i>ST</i> object in the CISS solver.
<code>getConverged()</code>	Get the number of converged eigenpairs.
<code>getConvergedReason()</code>	Get the reason why the <code>solve()</code> iteration was stopped.
<code>getConvergenceTest()</code>	Get how to compute the error estimate used in the convergence test.
<code>getDS()</code>	Get the direct solver associated to the eigensolver.
<code>getDimensions()</code>	Get number of eigenvalues to compute and the dimension of the subspace.
<code>getEigenpair(i[, Vr, Vi])</code>	Get the i-th solution of the eigenproblem as computed by <code>solve()</code> .
<code>getEigenvalue(i)</code>	Get the i-th eigenvalue as computed by <code>solve()</code> .
<code>getEigenvector(i[, Vr, Vi])</code>	Get the i-th eigenvector as computed by <code>solve()</code> .
<code>getErrorEstimate(i)</code>	Get the error estimate associated to the i-th computed eigenpair.
<code>getExtraction()</code>	Get the extraction type used by the EPS object.
<code>getGDBOrth()</code>	Get the orthogonalization used in the search subspace.
<code>getGDBBlockSize()</code>	Get the number of vectors to be added to the searching space.
<code>getGDDoubleExpansion()</code>	Get a flag indicating whether the double expansion variant is active.
<code>getGDInitialSize()</code>	Get the initial size of the searching space.
<code>getGDKrylovStart()</code>	Get a flag indicating if the search subspace is started with a Krylov basis.
<code>getGDRestart()</code>	Get the number of vectors of the search space after restart.
<code>getInterval()</code>	Get the computational interval for spectrum slicing.
<code>getInvariantSubspace()</code>	Get an orthonormal basis of the computed invariant subspace.
<code>getIterationNumber()</code>	Get the current iteration number.
<code>getJDBOrth()</code>	Get the orthogonalization used in the search subspace.
<code>getJDBBlockSize()</code>	Get the number of vectors to be added to the searching space.
<code>getJDConstCorrectionTol()</code>	Get the flag indicating if the dynamic stopping is being used.
<code>getJDFix()</code>	Get the threshold for changing the target in the correction equation.
<code>getJDInitialSize()</code>	Get the initial size of the searching space.
<code>getJDKrylovStart()</code>	Get a flag indicating if the search subspace is started with a Krylov basis.
<code>getJDRestart()</code>	Get the number of vectors of the search space after restart.
<code>getKrylovSchurBSEType()</code>	Get the method used for BSE structured eigenproblems (Krylov-Schur).
<code>getKrylovSchurDetectZeros()</code>	Get the flag that enforces zero detection in spectrum slicing.
<code>getKrylovSchurDimensions()</code>	Get the dimensions used for each subsolve step (spectrum slicing).

continues on next page

Table 36 – continued from previous page

<code>getKrylovSchurInertias()</code>	Get the values of the shifts and their corresponding inertias.
<code>getKrylovSchurKSP()</code>	Get the linear solver object associated with the internal <i>EPS</i> object.
<code>getKrylovSchurLocking()</code>	Get the locking flag used in the Krylov-Schur method.
<code>getKrylovSchurPartitions()</code>	Get the number of partitions of the communicator (spectrum slicing).
<code>getKrylovSchurRestart()</code>	Get the restart parameter used in the Krylov-Schur method.
<code>getKrylovSchurSubcommInfo()</code>	Get information related to the case of doing spectrum slicing.
<code>getKrylovSchurSubcommMats()</code>	Get the eigenproblem matrices stored in the subcommunicator.
<code>getKrylovSchurSubcommPairs(i, V)</code>	Get the i-th eigenpair stored in the multi-communicator of the process.
<code>getKrylovSchurSubintervals()</code>	Get the points that delimit the subintervals.
<code>getLOBPCGBlockSize()</code>	Get the block size used in the LOBPCG method.
<code>getLOBPCGLocking()</code>	Get the locking flag used in the LOBPCG method.
<code>getLOBPCGRestart()</code>	Get the restart parameter used in the LOBPCG method.
<code>getLanczosReorthogType()</code>	Get the type of reorthogonalization used during the Lanczos iteration.
<code>getLeftEigenvector(i[, Wr, Wi])</code>	Get the i-th left eigenvector as computed by <i>solve()</i> .
<code>getLyapIIRanks()</code>	Get the rank values used for the Lyapunov step.
<code>getMonitor()</code>	Get the list of monitor functions.
<code>getOperators()</code>	Get the matrices associated with the eigenvalue problem.
<code>getOptionsPrefix()</code>	Get the prefix used for searching for all EPS options in the database.
<code>getPowerShiftType()</code>	Get the type of shifts used during the power iteration.
<code>getProblemType()</code>	Get the problem type from the EPS object.
<code>getPurify()</code>	Get the flag indicating whether purification is activated or not.
<code>getRG()</code>	Get the region object associated to the eigensolver.
<code>getRQCGReset()</code>	Get the reset parameter used in the RQCG method.
<code>getST()</code>	Get the spectral transformation object associated to the eigensolver.
<code>getStoppingTest()</code>	Get the stopping function.
<code>getTarget()</code>	Get the value of the target.
<code>getThreshold()</code>	Get the threshold used in the threshold stopping test.
<code>getTolerances()</code>	Get the tolerance and max.
<code>getTrackAll()</code>	Get the flag indicating if all residual norms must be computed or not.
<code>getTrueResidual()</code>	Get the flag indicating if true residual must be computed explicitly.
<code>getTwoSided()</code>	Get the flag indicating if a two-sided variant of the algorithm is being used.
<code>getType()</code>	Get the EPS type of this object.
<code>getWhichEigenpairs()</code>	Get which portion of the spectrum is to be sought.
<code>isGeneralized()</code>	Tell if the EPS object corresponds to a generalized eigenproblem.

continues on next page

Table 36 – continued from previous page

<i>isHermitian()</i>	Tell if the EPS object corresponds to a Hermitian eigenproblem.
<i>isPositive()</i>	Eigenproblem requiring a positive (semi-) definite matrix B .
<i>isStructured()</i>	Tell if the EPS object corresponds to a structured eigenvalue problem.
<i>reset()</i>	Reset the EPS object.
<i>setArbitrarySelection</i> (arbitrary[, args, kargs])	Set an arbitrary selection criterion function.
<i>setArnoldiDelayed</i> (delayed)	Set (toggle) delayed reorthogonalization in the Arnoldi iteration.
<i>setBV</i> (bv)	Set a basis vectors object associated to the eigensolver.
<i>setBalance</i> ([balance, iterations, cutoff])	Set the balancing technique to be used by the eigensolver.
<i>setCISSExtraction</i> (extraction)	Set the extraction technique used in the CISS solver.
<i>setCISSQuadRule</i> (quad)	Set the quadrature rule used in the CISS solver.
<i>setCISSRefinement</i> ([inner, blsize])	Set the values of various refinement parameters in the CISS solver.
<i>setCISSSizes</i> ([ip, bs, ms, npart, bsmax, ...])	Set the values of various size parameters in the CISS solver.
<i>setCISSThreshold</i> ([delta, spur])	Set the values of various threshold parameters in the CISS solver.
<i>setCISSUseST</i> (usest)	Set a flag indicating that the CISS solver will use the <i>ST</i> object.
<i>setConvergenceTest</i> (conv)	Set how to compute the error estimate used in the convergence test.
<i>setDS</i> (ds)	Set a direct solver object associated to the eigensolver.
<i>setDeflationSpace</i> (space)	Add vectors to the basis of the deflation space.
<i>setDimensions</i> ([nev, ncv, mpd])	Set number of eigenvalues to compute and the dimension of the subspace.
<i>setEigenvalueComparison</i> (comparison[, args, ...])	Set an eigenvalue comparison function.
<i>setExtraction</i> (extraction)	Set the extraction type used by the EPS object.
<i>setFromOptions</i> ()	Set EPS options from the options database.
<i>setGDBOrth</i> (borth)	Set the orthogonalization that will be used in the search subspace.
<i>setGDBBlockSize</i> (bs)	Set the number of vectors to be added to the searching space.
<i>setGDDoubleExpansion</i> (doubleexp)	Set that the search subspace is expanded with double expansion.
<i>setGDInitialSize</i> (initialsize)	Set the initial size of the searching space.
<i>setGDKrylovStart</i> ([krylovstart])	Set (toggle) starting the search subspace with a Krylov basis.
<i>setGDRestart</i> ([minv, plusk])	Set the number of vectors of the search space after restart.
<i>setInitialSpace</i> (space)	Set the initial space from which the eigensolver starts to iterate.
<i>setInterval</i> (inta, intb)	Set the computational interval for spectrum slicing.
<i>setJDBOrth</i> (borth)	Set the orthogonalization that will be used in the search subspace.
<i>setJDBBlockSize</i> (bs)	Set the number of vectors to be added to the searching space.

continues on next page

Table 36 – continued from previous page

<code>setJDConstCorrectionTol</code> (constant)	Deactivate the dynamic stopping criterion.
<code>setJDFix</code> (fix)	Set the threshold for changing the target in the correction equation.
<code>setJDInitialSize</code> (initialsize)	Set the initial size of the searching space.
<code>setJDKrylovStart</code> ([krylovstart])	Set (toggle) starting the search subspace with a Krylov basis.
<code>setJDRestart</code> ([minv, plusk])	Set the number of vectors of the search space after restart.
<code>setKrylovSchurBSEType</code> (bse)	Set the Krylov-Schur variant used for BSE structured eigenproblems.
<code>setKrylovSchurDetectZeros</code> (detect)	Set the flag that enforces zero detection in spectrum slicing.
<code>setKrylovSchurDimensions</code> ([nev, ncv, mpd])	Set the dimensions used for each subsolve step (spectrum slicing).
<code>setKrylovSchurLocking</code> (lock)	Set (toggle) locking/non-locking variants of the Krylov-Schur method.
<code>setKrylovSchurPartitions</code> (npart)	Set the number of partitions of the communicator (spectrum slicing).
<code>setKrylovSchurRestart</code> (keep)	Set the restart parameter for the Krylov-Schur method.
<code>setKrylovSchurSubintervals</code> (subint)	Set the subinterval boundaries.
<code>setLOBPCGBlockSize</code> (bs)	Set the block size of the LOBPCG method.
<code>setLOBPCGLocking</code> (lock)	Toggle between locking and non-locking (LOBPCG method).
<code>setLOBPCGRestart</code> (restart)	Set the restart parameter for the LOBPCG method.
<code>setLanczosReorthogType</code> (reorthog)	Set the type of reorthogonalization used during the Lanczos iteration.
<code>setLeftInitialSpace</code> (space)	Set a left initial space from which the eigensolver starts to iterate.
<code>setLyapIIRanks</code> ([rkc, rkl])	Set the ranks used in the solution of the Lyapunov equation.
<code>setMonitor</code> (monitor[, args, kargs])	Append a monitor function to the list of monitors.
<code>setOperators</code> (A[, B])	Set the matrices associated with the eigenvalue problem.
<code>setOptionsPrefix</code> ([prefix])	Set the prefix used for searching for all EPS options in the database.
<code>setPowerShiftType</code> (shift)	Set the type of shifts used during the power iteration.
<code>setProblemType</code> (problem_type)	Set the type of the eigenvalue problem.
<code>setPurify</code> ([purify])	Set (toggle) eigenvector purification.
<code>setRG</code> (rg)	Set a region object associated to the eigensolver.
<code>setRQCGReset</code> (nrest)	Set the reset parameter of the RQCG iteration.
<code>setST</code> (st)	Set a spectral transformation object associated to the eigensolver.
<code>setStoppingTest</code> (stopping[, args, kargs])	Set when to stop the outer iteration of the eigensolver.
<code>setTarget</code> (target)	Set the value of the target.
<code>setThreshold</code> (thres[, rel])	Set the threshold used in the threshold stopping test.
<code>setTolerances</code> ([tol, max_it])	Set the tolerance and max.
<code>setTrackAll</code> (trackall)	Set if the solver must compute the residual of all approximate eigenpairs.
<code>setTrueResidual</code> (trueres)	Set if the solver must compute the true residual explicitly or not.

continues on next page

Table 36 – continued from previous page

<code>setTwoSided(twosided)</code>	Set to use a two-sided variant that also computes left eigenvectors.
<code>setType(eps_type)</code>	Set the particular solver to be used in the EPS object.
<code>setUp()</code>	Set up all the internal data structures.
<code>setWhichEigenpairs(which)</code>	Set which portion of the spectrum is to be sought.
<code>solve()</code>	Solve the eigensystem.
<code>updateKrylovSchurSubcommMats([s, a, Au, t, ...])</code>	Update the eigenproblem matrices stored internally in the communicator.
<code>valuesView([viewer])</code>	Display the computed eigenvalues in a viewer.
<code>vectorsView([viewer])</code>	Output computed eigenvectors to a viewer.
<code>view([viewer])</code>	Print the EPS data structure.

Attributes Summary

<code>bv</code>	The basis vectors (BV) object associated.
<code>ds</code>	The direct solver (DS) object associated.
<code>extraction</code>	The type of extraction technique to be employed.
<code>max_it</code>	The maximum iteration count.
<code>problem_type</code>	The type of the eigenvalue problem.
<code>purify</code>	Eigenvector purification.
<code>rg</code>	The region (RG) object associated.
<code>st</code>	The spectral transformation (ST) object associated.
<code>target</code>	The value of the target.
<code>tol</code>	The tolerance.
<code>track_all</code>	Compute the residual norm of all approximate eigenpairs.
<code>true_residual</code>	Compute the true residual explicitly.
<code>two_sided</code>	Two-sided that also computes left eigenvectors.
<code>which</code>	The portion of the spectrum to be sought.

Methods Documentation

`appendOptionsPrefix(prefix=None)`

Append to the prefix used for searching for all EPS options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all EPS option requests.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:421 <slepc4py/SLEPc/EPS.pyx#L421>`

`cancelMonitor()`

Clear all monitors for an *EPS* object.

Logically collective.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1439 <slepc4py/SLEPc/EPS.pyx#L1439>`

Return type

None

computeError(*i*, *etype*=None)

Compute the error associated with the *i*-th computed eigenpair.

Collective.

Compute the error (based on the residual norm) associated with the *i*-th computed eigenpair.

Parameters

- **i** (*int*) – Index of the solution to be considered.
- **etype** (*ErrorType* / *None*) – The error type to compute.

Returns

The error bound, computed in various ways from the residual norm $\|Ax - kBx\|_2$ where k is the eigenvalue and x is the eigenvector.

Return type

float

Notes

The index *i* should be a value between 0 and `nconv-1` (see `getConverged()`).

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1722 <slepc4py/SLEPc/EPS.pyx#L1722>`

create(*comm*=None)

Create the EPS object.

Collective.

Parameters

comm (*Comm* / *None*) – MPI communicator; if not provided, it defaults to all processes.

Return type

Self

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:323 <slepc4py/SLEPc/EPS.pyx#L323>`

destroy()

Destroy the EPS object.

Collective.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:305 <slepc4py/SLEPc/EPS.pyx#L305>`

Return type

Self

errorView(*etype*=None, *viewer*=None)

Display the errors associated with the computed solution.

Collective.

Display the errors and the eigenvalues.

Parameters

- **etype** (*ErrorType* / *None*) – The error type to compute.
- **viewer** (*petsc4py.PETSc.Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

Notes

By default, this function checks the error of all eigenpairs and prints the eigenvalues if all of them are below the requested tolerance. If the viewer has format ASCII_INFO_DETAIL then a table with eigenvalues and corresponding errors is printed.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1756 <slepc4py/SLEPc/EPS.pyx#L1756>`

getArnoldiDelayed()

Get the type of reorthogonalization used during the Arnoldi iteration.

Not collective.

Returns

True if delayed reorthogonalization is to be used.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1887 <slepc4py/SLEPc/EPS.pyx#L1887>`

getBV()

Get the basis vector objects associated to the eigensolver.

Not collective.

Returns

The basis vectors context.

Return type

`BV`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1111 <slepc4py/SLEPc/EPS.pyx#L1111>`

getBalance()

Get the balancing type used by the EPS, and the associated parameters.

Not collective.

Returns

- **balance** (`Balance`) – The balancing method
- **iterations** (`int`) – Number of iterations of the balancing algorithm
- **cutoff** (`float`) – Cutoff value

Return type

`tuple[Balance, int, float]`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:560 <slepc4py/SLEPc/EPS.pyx#L560>`

getCISSExtraction()

Get the extraction technique used in the CISS solver.

Not collective.

Returns

The extraction technique.

Return type

`CISSExtraction`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3096 <slepc4py/SLEPc/EPS.pyx#L3096>`

getCISSKSPs()

Get the array of linear solver objects associated with the CISS solver.

Not collective.

Returns

The linear solver objects.

Return type

`list of petsc4py.PETSc.KSP`

Notes

The number of `petsc4py.PETSc.KSP` solvers is equal to the number of integration points divided by the number of partitions. This value is halved in the case of real matrices with a region centered at the real axis.

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:3326 <slepc4py/SLEPc/EPs.pyx#L3326>`

getCISSQuadRule()

Get the quadrature rule used in the CISS solver.

Not collective.

Returns

The quadrature rule.

Return type

`CISSQuadRule`

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:3125 <slepc4py/SLEPc/EPs.pyx#L3125>`

getCISSRefinement()

Get the values of various refinement parameters in the CISS solver.

Not collective.

Returns

- **inner** (`int`) – Number of iterative refinement iterations (inner loop).
- **bsize** (`int`) – Number of iterative refinement iterations (blocksize loop).

Return type

`tuple[int, int]`

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:3276 <slepc4py/SLEPc/EPs.pyx#L3276>`

getCISSSizes()

Get the values of various size parameters in the CISS solver.

Not collective.

Returns

- **ip** (`int`) – Number of integration points.
- **bs** (`int`) – Block size.
- **ms** (`int`) – Moment size.
- **npart** (`int`) – Number of partitions when splitting the communicator.
- **bsmax** (`int`) – Maximum block size.
- **realmats** (`bool`) – True if A and B are real.

Return type

tuple[int, int, int, int, int, bool]

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3190 <slepc4py/SLEPc/EPS.pyx#L3190>`

getCISSThreshold()

Get the values of various threshold parameters in the CISS solver.

Not collective.

Returns

- **delta** (float) – Threshold for numerical rank.
- **spur** (float) – Spurious threshold (to discard spurious eigenpairs).

Return type

tuple[float, float]

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3239 <slepc4py/SLEPc/EPS.pyx#L3239>`

getCISSUseST()

Get the flag indicating the use of the *ST* object in the CISS solver.

Not collective.

Returns

Whether to use the *ST* object or not.

Return type

bool

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3311 <slepc4py/SLEPc/EPS.pyx#L3311>`

getConverged()

Get the number of converged eigenpairs.

Not collective.

Returns

Number of converged eigenpairs.

Return type

int

Notes

This function should be called after *solve()* has finished.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1508 <slepc4py/SLEPc/EPS.pyx#L1508>`

getConvergedReason()

Get the reason why the *solve()* iteration was stopped.

Not collective.

Returns

Negative value indicates diverged, positive value converged.

Return type

ConvergedReason

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1493 <slepc4py/SLEPc/EPS.pyx#L1493>`

getConvergenceTest()

Get how to compute the error estimate used in the convergence test.

Not collective.

Returns

The method used to compute the error estimate used in the convergence test.

Return type

Conv

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:925 <slepc4py/SLEPc/EPS.pyx#L925>`

getDS()

Get the direct solver associated to the eigensolver.

Not collective.

Returns

The direct solver context.

Return type

DS

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1140 <slepc4py/SLEPc/EPS.pyx#L1140>`

getDimensions()

Get number of eigenvalues to compute and the dimension of the subspace.

Not collective.

Returns

- **nev** (*int*) – Number of eigenvalues to compute.
- **ncv** (*int*) – Maximum dimension of the subspace to be used by the solver.
- **mpd** (*int*) – Maximum dimension allowed for the projected problem.

Return type

tuple[int, int, int]

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1014 <slepc4py/SLEPc/EPS.pyx#L1014>`

getEigenpair(*i*, *Vr*=None, *Vi*=None)

Get the *i*-th solution of the eigenproblem as computed by *solve()*.

Collective.

The solution consists of both the eigenvalue and the eigenvector.

Parameters

- **i** (*int*) – Index of the solution to be obtained.
- **Vr** (*Vec* | *None*) – Placeholder for the returned eigenvector (real part).
- **Vi** (*Vec* | *None*) – Placeholder for the returned eigenvector (imaginary part).

Returns

e – The computed eigenvalue. It will be a real variable in case of a Hermitian or generalized Hermitian eigenproblem. Otherwise it will be a complex variable (possibly with zero imaginary part).

Return type

Scalar

Notes

The index `i` should be a value between 0 and `nconv-1` (see `getConverged()`). Eigenpairs are indexed according to the ordering criterion established with `setWhichEigenpairs()`.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1615 <slepc4py/SLEPc/EPS.pyx#L1615>`

`getEigenvalue(i)`

Get the `i`-th eigenvalue as computed by `solve()`.

Not collective.

Parameters

`i` (`int`) – Index of the solution to be obtained.

Returns

The computed eigenvalue. It will be a real variable in case of a Hermitian or generalized Hermitian eigenproblem. Otherwise it will be a complex variable (possibly with zero imaginary part).

Return type

`Scalar`

Notes

The index `i` should be a value between 0 and `nconv-1` (see `getConverged()`). Eigenpairs are indexed according to the ordering criterion established with `setWhichEigenpairs()`.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1527 <slepc4py/SLEPc/EPS.pyx#L1527>`

`getEigenvector(i, Vr=None, Vi=None)`

Get the `i`-th eigenvector as computed by `solve()`.

Collective.

Parameters

- `i` (`int`) – Index of the solution to be obtained.
- `Vr` (`Vec` / `None`) – Placeholder for the returned eigenvector (real part).
- `Vi` (`Vec` / `None`) – Placeholder for the returned eigenvector (imaginary part).

Return type

`None`

Notes

The index `i` should be a value between 0 and `nconv-1` (see `getConverged()`). Eigenpairs are indexed according to the ordering criterion established with `setWhichEigenpairs()`.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1561 <slepc4py/SLEPc/EPS.pyx#L1561>`

`getErrorEstimate(i)`

Get the error estimate associated to the `i`-th computed eigenpair.

Not collective.

Parameters

`i` (`int`) – Index of the solution to be considered.

Returns

Error estimate.

Return type
`float`

Notes

This is the error estimate used internally by the eigensolver. The actual error bound can be computed with `computeError()`.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1696 <slepc4py/SLEPc/EPS.pyx#L1696>`

`getExtraction()`

Get the extraction type used by the EPS object.

Not collective.

Returns
The method of extraction.

Return type
`Extraction`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:613 <slepc4py/SLEPc/EPS.pyx#L613>`

`getGDBOrth()`

Get the orthogonalization used in the search subspace.

Not collective.

Get the orthogonalization used in the search subspace in case of generalized Hermitian problems.

Returns
Whether to B-orthogonalize the search subspace.

Return type
`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2608 <slepc4py/SLEPc/EPS.pyx#L2608>`

`getGDBlockSize()`

Get the number of vectors to be added to the searching space.

Not collective.

Get the number of vectors to be added to the searching space in every iteration.

Returns
The number of vectors added to the search space in every iteration.

Return type
`int`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2501 <slepc4py/SLEPc/EPS.pyx#L2501>`

`getGDDoubleExpansion()`

Get a flag indicating whether the double expansion variant is active.

Not collective.

Get a flag indicating whether the double expansion variant has been activated or not.

Returns
True if using double expansion.

Return type
`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2646 <slepc4py/SLEPc/EPS.pyx#L2646>`

getGDInitialSize()

Get the initial size of the searching space.

Not collective.

Returns

The number of vectors of the initial searching subspace.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2576 <slepc4py/SLEPc/EPS.pyx#L2576>`

getGDKrylovStart()

Get a flag indicating if the search subspace is started with a Krylov basis.

Not collective.

Returns

True if starting the search subspace with a Krylov basis.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2469 <slepc4py/SLEPc/EPS.pyx#L2469>`

getGDRestart()

Get the number of vectors of the search space after restart.

Not collective.

Get the number of vectors of the search space after restart and the number of vectors saved from the previous iteration.

Returns

- **minv** (`int`) – The number of vectors of the search subspace after restart.
- **plusk** (`int`) – The number of vectors saved from the previous iteration.

Return type

`tuple[int, int]`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2541 <slepc4py/SLEPc/EPS.pyx#L2541>`

getInterval()

Get the computational interval for spectrum slicing.

Not collective.

Returns

- **inta** (`float`) – The left end of the interval.
- **intb** (`float`) – The right end of the interval.

Return type

`tuple[float, float]`

Notes

If the interval was not set by the user, then zeros are returned.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:773 <slepc4py/SLEPc/EPS.pyx#L773>`

`getInvariantSubspace()`

Get an orthonormal basis of the computed invariant subspace.

Collective.

Returns

Basis of the invariant subspace.

Return type

`list of petsc4py.PETSc.Vec`

Notes

This function should be called after `solve()` has finished.

The returned vectors span an invariant subspace associated with the computed eigenvalues. An invariant subspace X of A satisfies $Ax \in X$ for all $x \in X$ (a similar definition applies for generalized eigenproblems).

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1657 <slepc4py/SLEPc/EPS.pyx#L1657>`

`getIterationNumber()`

Get the current iteration number.

Not collective.

If the call to `solve()` is complete, then it returns the number of iterations carried out by the solution method.

Returns

Iteration number.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1475 <slepc4py/SLEPc/EPS.pyx#L1475>`

`getJDBOrth()`

Get the orthogonalization used in the search subspace.

Not collective.

Get the orthogonalization used in the search subspace in case of generalized Hermitian problems.

Returns

Whether to B-orthogonalize the search subspace.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2890 <slepc4py/SLEPc/EPS.pyx#L2890>`

`getJDBlockSize()`

Get the number of vectors to be added to the searching space.

Not collective.

Get the number of vectors to be added to the searching space in every iteration.

Returns

The number of vectors added to the search space in every iteration.

Return type

`int`

`:sources: Source code at slepc4py/SLEPc/EPS.pyx:2712 <slepc4py/SLEPc/EPS.pyx#L2712>`

getJDConstCorrectionTol()

Get the flag indicating if the dynamic stopping is being used.

Not collective.

Get the flag indicating if the dynamic stopping is being used for solving the correction equation.

Returns

True if the dynamic stopping criterion is not being used.

Return type

`bool`

`:sources: Source code at slepc4py/SLEPc/EPS.pyx:2855 <slepc4py/SLEPc/EPS.pyx#L2855>`

getJDFix()

Get the threshold for changing the target in the correction equation.

Not collective.

Returns

The threshold for changing the target.

Return type

`float`

`:sources: Source code at slepc4py/SLEPc/EPS.pyx:2822 <slepc4py/SLEPc/EPS.pyx#L2822>`

getJDInitialSize()

Get the initial size of the searching space.

Not collective.

Returns

The number of vectors of the initial searching subspace.

Return type

`int`

`:sources: Source code at slepc4py/SLEPc/EPS.pyx:2787 <slepc4py/SLEPc/EPS.pyx#L2787>`

getJDKrylovStart()

Get a flag indicating if the search subspace is started with a Krylov basis.

Not collective.

Returns

True if starting the search subspace with a Krylov basis.

Return type

`bool`

`:sources: Source code at slepc4py/SLEPc/EPS.pyx:2680 <slepc4py/SLEPc/EPS.pyx#L2680>`

getJDRestart()

Get the number of vectors of the search space after restart.

Not collective.

Get the number of vectors of the search space after restart and the number of vectors saved from the previous iteration.

Returns

- **minv** (*int*) – The number of vectors of the search subspace after restart.
- **plusk** (*int*) – The number of vectors saved from the previous iteration.

Return type

tuple[*int*, *int*]

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:2752 <slepc4py/SLEPc/EPs.pyx#L2752>`

getKrylovSchurBSEType()

Get the method used for BSE structured eigenproblems (Krylov-Schur).

Not collective.

Returns

The BSE method.

Return type

KrylovSchurBSEType

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:1959 <slepc4py/SLEPc/EPs.pyx#L1959>`

getKrylovSchurDetectZeros()

Get the flag that enforces zero detection in spectrum slicing.

Not collective.

Returns

The zero detection flag.

Return type

bool

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:2112 <slepc4py/SLEPc/EPs.pyx#L2112>`

getKrylovSchurDimensions()

Get the dimensions used for each subsolve step (spectrum slicing).

Not collective.

Get the dimensions used for each subsolve step in case of doing spectrum slicing for a computational interval.

Returns

- **nev** (*int*) – Number of eigenvalues to compute.
- **ncv** (*int*) – Maximum dimension of the subspace to be used by the solver.
- **mpd** (*int*) – Maximum dimension allowed for the projected problem.

Return type

tuple[*int*, *int*, *int*]

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:2159 <slepc4py/SLEPc/EPs.pyx#L2159>`

getKrylovSchurInertias()

Get the values of the shifts and their corresponding inertias.

Not collective.

Get the values of the shifts and their corresponding inertias in case of doing spectrum slicing for a computational interval.

Returns

- **shifts** (*ArrayReal*) – The values of the shifts used internally in the solver.
- **inertias** (*ArrayInt*) – The values of the inertia in each shift.

Return type

tuple[*ArrayReal*, *ArrayInt*]

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:2404 <slepc4py/SLEPc/EPs.pyx#L2404>`

getKrylovSchurKSP()

Get the linear solver object associated with the internal *EPs* object.

Collective.

Get the linear solver object associated with the internal *EPs* object in case of doing spectrum slicing for a computational interval.

Returns

The linear solver object.

Return type

petsc4py.PETSc.KSP

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:2434 <slepc4py/SLEPc/EPs.pyx#L2434>`

getKrylovSchurLocking()

Get the locking flag used in the Krylov-Schur method.

Not collective.

Returns

The locking flag.

Return type

bool

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:2030 <slepc4py/SLEPc/EPs.pyx#L2030>`

getKrylovSchurPartitions()

Get the number of partitions of the communicator (spectrum slicing).

Not collective.

Returns

The number of partitions.

Return type

int

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:2070 <slepc4py/SLEPc/EPs.pyx#L2070>`

getKrylovSchurRestart()

Get the restart parameter used in the Krylov-Schur method.

Not collective.

Returns

The number of vectors to be kept at restart.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1994 <slepc4py/SLEPc/EPS.pyx#L1994>`

`getKrylovSchurSubcommInfo()`

Get information related to the case of doing spectrum slicing.

Collective on the subcommunicator (if `v` is given).

Get information related to the case of doing spectrum slicing for a computational interval with multiple communicators.

Returns

- `k` (`int`) – Number of the subinterval for the calling process.
- `n` (`int`) – Number of eigenvalues found in the `k`-th subinterval.
- `v` (`petsc4py.PETSc.Vec`) – A vector owned by processes in the subcommunicator with dimensions compatible for locally computed eigenvectors.

Return type

`tuple[int, int, petsc4py.PETSc.Vec]`

Notes

This function is only available for spectrum slicing runs.

The returned `Vec` should be destroyed by the user.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2183 <slepc4py/SLEPc/EPS.pyx#L2183>`

`getKrylovSchurSubcommMats()`

Get the eigenproblem matrices stored in the subcommunicator.

Collective on the subcommunicator.

Get the eigenproblem matrices stored internally in the subcommunicator to which the calling process belongs.

Returns

- `A` (`petsc4py.PETSc.Mat`) – The matrix associated with the eigensystem.
- `B` (`petsc4py.PETSc.Mat`) – The second matrix in the case of generalized eigenproblems.

Return type

`tuple[petsc4py.PETSc.Mat, petsc4py.PETSc.Mat] | tuple[petsc4py.PETSc.Mat, None]`

Notes

This is the analog of `getOperators()`, but returns the matrices distributed differently (in the subcommunicator rather than in the parent communicator).

These matrices should not be modified by the user.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2246 <slepc4py/SLEPc/EPS.pyx#L2246>`

getKrylovSchurSubcommPairs(*i*, *V*)

Get the *i*-th eigenpair stored in the multi-communicator of the process.

Collective on the subcommunicator (if *v* is given).

Get the *i*-th eigenpair stored internally in the multi-communicator to which the calling process belongs.

Parameters

- ***i*** (*int*) – Index of the solution to be obtained.
- ***V*** (*Vec*) – Placeholder for the returned eigenvector.

Returns

The computed eigenvalue.

Return type

Scalar

Notes

The index *i* should be a value between 0 and *n*-1, where *n* is the number of vectors in the local subinterval, see [*getKrylovSchurSubcommInfo\(\)*](#).

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2214 <slepc4py/SLEPc/EPS.pyx#L2214>`

getKrylovSchurSubintervals()

Get the points that delimit the subintervals.

Not collective.

Get the points that delimit the subintervals used in spectrum slicing with several partitions.

Returns

Real values specifying subintervals

Return type

ArrayReal

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2379 <slepc4py/SLEPc/EPS.pyx#L2379>`

getLOBPCGBlockSize()

Get the block size used in the LOBPCG method.

Not collective.

Returns

The block size.

Return type

int

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2956 <slepc4py/SLEPc/EPS.pyx#L2956>`

getLOBPCGLocking()

Get the locking flag used in the LOBPCG method.

Not collective.

Returns

The locking flag.

Return type

bool

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3028 <slepc4py/SLEPc/EPS.pyx#L3028>`

getLOBPCGRestart()

Get the restart parameter used in the LOBPCG method.

Not collective.

Returns

The restart parameter.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2993 <slepc4py/SLEPc/EPS.pyx#L2993>`

getLanczosReorthogType()

Get the type of reorthogonalization used during the Lanczos iteration.

Not collective.

Returns

The type of reorthogonalization.

Return type

`LanczosReorthogType`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1921 <slepc4py/SLEPc/EPS.pyx#L1921>`

getLeftEigenvector(*i*, *Wr*=None, *Wi*=None)

Get the *i*-th left eigenvector as computed by `solve()`.

Collective.

Parameters

- ***i*** (`int`) – Index of the solution to be obtained.
- ***Wr*** (`Vec` / `None`) – Placeholder for the returned eigenvector (real part).
- ***Wi*** (`Vec` / `None`) – Placeholder for the returned eigenvector (imaginary part).

Return type

`None`

Notes

The index *i* should be a value between 0 and `nconv-1` (see `getConverged()`). Eigensolutions are indexed according to the ordering criterion established with `setWhichEigenpairs()`.

Left eigenvectors are available only if the twosided flag was set with `setTwoSided()`.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1587 <slepc4py/SLEPc/EPS.pyx#L1587>`

getLyapIIRanks()

Get the rank values used for the Lyapunov step.

Not collective.

Returns

- ***rkc*** (`int`) – The compressed rank.
- ***rkl*** (`int`) – The Lyapunov rank.

Return type

`tuple[int, int]`

:sources:`Source code at slepc4py/SLEPc/EPS.pyx:3062 <slepc4py/SLEPc/EPS.pyx#L3062>`

getMonitor()

Get the list of monitor functions.

:sources:`Source code at slepc4py/SLEPc/EPS.pyx:1435 <slepc4py/SLEPc/EPS.pyx#L1435>`

Return type

EPSPMonitorFunction

getOperators()

Get the matrices associated with the eigenvalue problem.

Collective.

Returns

- **A** (*petsc4py.PETSc.Mat*) – The matrix associated with the eigensystem.
- **B** (*petsc4py.PETSc.Mat*) – The second matrix in the case of generalized eigenproblems.

Return type

tuple[*petsc4py.PETSc.Mat*, *petsc4py.PETSc.Mat*] | *tuple*[*petsc4py.PETSc.Mat*, *None*]

:sources:`Source code at slepc4py/SLEPc/EPS.pyx:1198 <slepc4py/SLEPc/EPS.pyx#L1198>`

getOptionsPrefix()

Get the prefix used for searching for all EPS options in the database.

Not collective.

Returns

The prefix string set for this EPS object.

Return type

str

:sources:`Source code at slepc4py/SLEPc/EPS.pyx:379 <slepc4py/SLEPc/EPS.pyx#L379>`

getPowerShiftType()

Get the type of shifts used during the power iteration.

Not collective.

Returns

The type of shift.

Return type

PowerShiftType

:sources:`Source code at slepc4py/SLEPc/EPS.pyx:1848 <slepc4py/SLEPc/EPS.pyx#L1848>`

getProblemType()

Get the problem type from the EPS object.

Not collective.

Returns

The problem type that was previously set.

Return type

ProblemType

:sources:`Source code at slepc4py/SLEPc/EPS.pyx:454 <slepc4py/SLEPc/EPS.pyx#L454>`

getPurify()

Get the flag indicating whether purification is activated or not.

Not collective.

Returns

Whether purification is activated or not.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:896 <slepc4py/SLEPc/EPS.pyx#L896>`

getRG()

Get the region object associated to the eigensolver.

Not collective.

Returns

The region context.

Return type

`RG`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1169 <slepc4py/SLEPc/EPS.pyx#L1169>`

getRQCGReset()

Get the reset parameter used in the RQCG method.

Not collective.

Returns

The number of iterations between resets.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2927 <slepc4py/SLEPc/EPS.pyx#L2927>`

getST()

Get the spectral transformation object associated to the eigensolver.

Not collective.

Returns

The spectral transformation.

Return type

`ST`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1082 <slepc4py/SLEPc/EPS.pyx#L1082>`

getStoppingTest()

Get the stopping function.

Not collective.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1353 <slepc4py/SLEPc/EPS.pyx#L1353>`

Return type

`EPSStoppingFunction`

getTarget()

Get the value of the target.

Not collective.

Returns

The value of the target.

Return type

Scalar

Notes

If the target was not set by the user, then zero is returned.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:734 <slepc4py/SLEPc/EPS.pyx#L734>`

getThreshold()

Get the threshold used in the threshold stopping test.

Not collective.

Returns

- **thres** (*float*) – The threshold.
- **rel** (*bool*) – Whether the threshold is relative or not.

Return type

tuple[*float*, *bool*]

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:691 <slepc4py/SLEPc/EPS.pyx#L691>`

getTolerances()

Get the tolerance and max. iter. count used for convergence tests.

Not collective.

Get the tolerance and iteration limit used by the default EPS convergence tests.

Returns

- **tol** (*float*) – The convergence tolerance.
- **max_it** (*int*) – The maximum number of iterations.

Return type

tuple[*float*, *int*]

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:822 <slepc4py/SLEPc/EPS.pyx#L822>`

getTrackAll()

Get the flag indicating if all residual norms must be computed or not.

Not collective.

Returns

Whether the solver compute all residuals or not.

Return type

bool

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:985 <slepc4py/SLEPc/EPS.pyx#L985>`

getTrueResidual()

Get the flag indicating if true residual must be computed explicitly.

Not collective.

Returns

Whether the solver compute all residuals or not.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:956 <slepc4py/SLEPc/EPS.pyx#L956>`

getTwoSided()

Get the flag indicating if a two-sided variant of the algorithm is being used.

Not collective.

Returns

Whether the two-sided variant is to be used or not.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:867 <slepc4py/SLEPc/EPS.pyx#L867>`

getType()

Get the EPS type of this object.

Not collective.

Returns

The solver currently being used.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:364 <slepc4py/SLEPc/EPS.pyx#L364>`

getWhichEigenpairs()

Get which portion of the spectrum is to be sought.

Not collective.

Returns

The portion of the spectrum to be sought by the solver.

Return type

`Which`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:653 <slepc4py/SLEPc/EPS.pyx#L653>`

isGeneralized()

Tell if the EPS object corresponds to a generalized eigenproblem.

Not collective.

Returns

True if two matrices were set with `setOperators()`.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:497 <slepc4py/SLEPc/EPS.pyx#L497>`

isHermitian()

Tell if the EPS object corresponds to a Hermitian eigenproblem.

Not collective.

Returns

True if the problem type set with `setProblemType()` was Hermitian.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:512 <slepc4py/SLEPc/EPS.pyx#L512>`

isPositive()

Eigenproblem requiring a positive (semi-) definite matrix B .

Not collective.

Tell if the EPS corresponds to an eigenproblem requiring a positive (semi-) definite matrix B .

Returns

True if the problem type set with `setProblemType()` was positive.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:527 <slepc4py/SLEPc/EPS.pyx#L527>`

isStructured()

Tell if the EPS object corresponds to a structured eigenvalue problem.

Not collective.

Returns

True if the problem type set with `setProblemType()` was structured.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:545 <slepc4py/SLEPc/EPS.pyx#L545>`

reset()

Reset the EPS object.

Collective.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:315 <slepc4py/SLEPc/EPS.pyx#L315>`

Return type

`None`

setArbitrarySelection(*arbitrary*, *args*=None, *kargs*=None)

Set an arbitrary selection criterion function.

Logically collective.

Set a function to look for eigenvalues according to an arbitrary selection criterion. This criterion can be based on a computation involving the current eigenvector approximation.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1363 <slepc4py/SLEPc/EPS.pyx#L1363>`

Parameters

- **arbitrary** (`EPsArbitraryFunction` | `None`)
- **args** (`tuple`[`Any`, ...] | `None`)

- **kargs** (*dict[str, Any] | None*)

Return type

None

setArnoldiDelayed(*delayed*)

Set (toggle) delayed reorthogonalization in the Arnoldi iteration.

Logically collective.

Parameters

delayed (*bool*) – True if delayed reorthogonalization is to be used.

Return type

None

Notes

This call is only relevant if the type was set to *EPS.Type.ARNOLDI* with *setType()*.

Delayed reorthogonalization is an aggressive optimization for the Arnoldi eigensolver than may provide better scalability, but sometimes makes the solver converge less than the default algorithm.

:sources: ``Source code at slepc4py/SLEPc/EPS.pyx:1863 <slepc4py/SLEPc/EPS.pyx#L1863>``

setBV(*bv*)

Set a basis vectors object associated to the eigensolver.

Collective.

Parameters

bv (*BV*) – The basis vectors context.

Return type

None

:sources: ``Source code at slepc4py/SLEPc/EPS.pyx:1127 <slepc4py/SLEPc/EPS.pyx#L1127>``

setBalance(*balance=None, iterations=None, cutoff=None*)

Set the balancing technique to be used by the eigensolver.

Logically collective.

Set the balancing technique to be used by the eigensolver, and some parameters associated to it.

Parameters

- **balance** (*Balance | None*) – The balancing method
- **iterations** (*int | None*) – Number of iterations of the balancing algorithm
- **cutoff** (*float | None*) – Cutoff value

Return type

None

:sources: ``Source code at slepc4py/SLEPc/EPS.pyx:581 <slepc4py/SLEPc/EPS.pyx#L581>``

setCISSExtraction(*extraction*)

Set the extraction technique used in the CISS solver.

Logically collective.

Parameters

extraction (*CISSExtraction*) – The extraction technique.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3082 <slepc4py/SLEPc/EPS.pyx#L3082>`

setCISSQuadRule(*quad*)

Set the quadrature rule used in the CISS solver.

Logically collective.

Parameters

quad (*CISSQuadRule*) – The quadrature rule.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3111 <slepc4py/SLEPc/EPS.pyx#L3111>`

setCISSRefinement(*inner=None, blsize=None*)

Set the values of various refinement parameters in the CISS solver.

Logically collective.

Parameters

- **inner** (*int* | *None*) – Number of iterative refinement iterations (inner loop).
- **blsize** (*int* | *None*) – Number of iterative refinement iterations (blocksize loop).

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3257 <slepc4py/SLEPc/EPS.pyx#L3257>`

setCISSSizes(*ip=None, bs=None, ms=None, npart=None, bsmax=None, realmats=False*)

Set the values of various size parameters in the CISS solver.

Logically collective.

Parameters

- **ip** (*int* | *None*) – Number of integration points.
- **bs** (*int* | *None*) – Block size.
- **ms** (*int* | *None*) – Moment size.
- **npart** (*int* | *None*) – Number of partitions when splitting the communicator.
- **bsmax** (*int* | *None*) – Maximum block size.
- **realmats** (*bool*) – True if A and B are real.

Return type

None

Notes

The default number of partitions is 1. This means the internal `petsc4py.PETSc.KSP` object is shared among all processes of the *EPS* communicator. Otherwise, the communicator is split into *npart* communicators, so that *npart* `petsc4py.PETSc.KSP` solves proceed simultaneously.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3140 <slepc4py/SLEPc/EPS.pyx#L3140>`

setCISSThreshold(*delta=None, spur=None*)

Set the values of various threshold parameters in the CISS solver.

Logically collective.

Parameters

- **delta** (*float* / *None*) – Threshold for numerical rank.
- **spur** (*float* / *None*) – Spurious threshold (to discard spurious eigenpairs).

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3220 <slepc4py/SLEPc/EPS.pyx#L3220>`

setCISSUseST(*usest*)

Set a flag indicating that the CISS solver will use the *ST* object.

Logically collective.

Set a flag indicating that the CISS solver will use the *ST* object for the linear solves.

Parameters

usest (*bool*) – Whether to use the *ST* object or not.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3294 <slepc4py/SLEPc/EPS.pyx#L3294>`

setConvergenceTest(*conv*)

Set how to compute the error estimate used in the convergence test.

Logically collective.

Parameters

conv (*Conv*) – The method used to compute the error estimate used in the convergence test.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:941 <slepc4py/SLEPc/EPS.pyx#L941>`

setDS(*ds*)

Set a direct solver object associated to the eigensolver.

Collective.

Parameters

ds (*DS*) – The direct solver context.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1156 <slepc4py/SLEPc/EPS.pyx#L1156>`

setDeflationSpace(*space*)

Add vectors to the basis of the deflation space.

Collective.

Parameters

space (*Vec* / *list[Vec]*) – Set of basis vectors to be added to the deflation space.

Return type

None

Notes

When a deflation space is given, the eigensolver seeks the eigensolution in the restriction of the problem to the orthogonal complement of this space. This can be used for instance in the case that an invariant subspace is known beforehand (such as the nullspace of the matrix).

The vectors do not need to be mutually orthonormal, since they are explicitly orthonormalized internally.

These vectors do not persist from one `solve()` call to the other, so the deflation space should be set every time.

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:1238 <slepc4py/SLEPc/EPs.pyx#L1238>`

setDimensions(*nev=None, ncv=None, mpd=None*)

Set number of eigenvalues to compute and the dimension of the subspace.

Logically collective.

Parameters

- **nev** (*int* / *None*) – Number of eigenvalues to compute.
- **ncv** (*int* / *None*) – Maximum dimension of the subspace to be used by the solver.
- **mpd** (*int* / *None*) – Maximum dimension allowed for the projected problem.

Return type

None

Notes

Use `DECIDE` for `ncv` and `mpd` to assign a reasonably good value, which is dependent on the solution method.

The parameters `ncv` and `mpd` are intimately related, so that the user is advised to set one of them at most. Normal usage is the following:

- In cases where `nev` is small, the user sets `ncv` (a reasonable default is $2 * nev$).
- In cases where `nev` is large, the user sets `mpd`.

The value of `ncv` should always be between `nev` and $(nev + mpd)$, typically $ncv = nev + mpd$. If `nev` is not too large, $mpd = nev$ is a reasonable choice, otherwise a smaller value should be used.

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:1035 <slepc4py/SLEPc/EPs.pyx#L1035>`

setEigenvalueComparison(*comparison, args=None, kargs=None*)

Set an eigenvalue comparison function.

Logically collective.

Specify the eigenvalue comparison function when `setWhichEigenpairs()` is set to `EPS.Which.USER`.

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:1390 <slepc4py/SLEPc/EPs.pyx#L1390>`

Parameters

- **comparison** (`EPSEigenvalueComparison` / *None*)
- **args** (*tuple*[*Any*, ...] / *None*)
- **kargs** (*dict*[*str*, *Any*] / *None*)

Return type

None

setExtraction(extraction)

Set the extraction type used by the EPS object.

Logically collective.

Parameters

extraction ([Extraction](#)) – The extraction method to be used by the solver.

Return type

None

Notes

Not all eigensolvers support all types of extraction. See the SLEPc documentation for details.

By default, a standard Rayleigh-Ritz extraction is used. Other extractions may be useful when computing interior eigenvalues.

Harmonic-type extractions are used in combination with a *target*. See [setTarget\(\)](#).

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:628 <slepc4py/SLEPc/EPS.pyx#L628>](#)

setFromOptions()

Set EPS options from the options database.

Collective.

This routine must be called before [setUp\(\)](#) if the user is to be allowed to set the solver type.

Notes

To see all options, run your program with the `-help` option.

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:436 <slepc4py/SLEPc/EPS.pyx#L436>](#)

Return type

None

setGDBOrth(borth)

Set the orthogonalization that will be used in the search subspace.

Logically collective.

Set the orthogonalization that will be used in the search subspace in case of generalized Hermitian problems.

Parameters

borth ([bool](#)) – Whether to B-orthogonalize the search subspace.

Return type

int

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:2591 <slepc4py/SLEPc/EPS.pyx#L2591>](#)

setGDBlockSize(bs)

Set the number of vectors to be added to the searching space.

Logically collective.

Set the number of vectors to be added to the searching space in every iteration.

Parameters

bs ([int](#)) – The number of vectors added to the search space in every iteration.

Return type

None

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:2484 <slepc4py/SLEPc/EPS.pyx#L2484>](#)

setGDDoubleExpansion(*doubleexp*)

Set that the search subspace is expanded with double expansion.

Logically collective.

Set a variant where the search subspace is expanded with $K[Ax, Bx]$ (double expansion) instead of the classic Kr , where K is the preconditioner, x the selected approximate eigenvector and r its associated residual vector.

Parameters

doubleexp (*bool*) – True if using double expansion.

Return type

None

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:2626 <slepc4py/SLEPc/EPS.pyx#L2626>](#)

setGDInitialSize(*initialsize*)

Set the initial size of the searching space.

Logically collective.

Parameters

initialsize (*int*) – The number of vectors of the initial searching subspace.

Return type

None

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:2562 <slepc4py/SLEPc/EPS.pyx#L2562>](#)

setGDKrylovStart(*krylovstart=True*)

Set (toggle) starting the search subspace with a Krylov basis.

Logically collective.

Parameters

krylovstart (*bool*) – True if starting the search subspace with a Krylov basis.

Return type

None

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:2455 <slepc4py/SLEPc/EPS.pyx#L2455>](#)

setGDRestart(*minv=None, plusk=None*)

Set the number of vectors of the search space after restart.

Logically collective.

Set the number of vectors of the search space after restart and the number of vectors saved from the previous iteration.

Parameters

- **minv** (*int*) – The number of vectors of the search subspace after restart.
- **plusk** (*int*) – The number of vectors saved from the previous iteration.

Return type

None

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:2519](#) <[slepc4py/SLEPc/EPS.pyx#L2519](#)>

setInitialSpace(*space*)

Set the initial space from which the eigensolver starts to iterate.

Collective.

Parameters

space (*Vec* | *list*[*Vec*]) – The initial space

Return type

None

Notes

Some solvers start to iterate on a single vector (initial vector). In that case, the other vectors are ignored.

In contrast to [setDeflationSpace\(\)](#), these vectors do not persist from one [solve\(\)](#) call to the other, so the initial space should be set every time.

The vectors do not need to be mutually orthonormal, since they are explicitly orthonormalized internally.

Common usage of this function is when the user can provide a rough approximation of the wanted eigenspace. Then, convergence may be faster.

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:1272](#) <[slepc4py/SLEPc/EPS.pyx#L1272](#)>

setInterval(*inta*, *intb*)

Set the computational interval for spectrum slicing.

Logically collective.

Parameters

- **inta** (*float*) – The left end of the interval.
- **intb** (*float*) – The right end of the interval.

Return type

None

Notes

Spectrum slicing is a technique employed for computing all eigenvalues of symmetric eigenproblems in a given interval. This function provides the interval to be considered. It must be used in combination with [EPS.Which.ALL](#), see [setWhichEigenpairs\(\)](#).

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:795](#) <[slepc4py/SLEPc/EPS.pyx#L795](#)>

setJDBOrth(*borth*)

Set the orthogonalization that will be used in the search subspace.

Logically collective.

Set the orthogonalization that will be used in the search subspace in case of generalized Hermitian problems.

Parameters

borth (*bool*) – Whether to B-orthogonalize the search subspace.

Return type

None

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:2873](#) <[slepc4py/SLEPc/EPS.pyx#L2873](#)>

setJDBlockSize(*bs*)

Set the number of vectors to be added to the searching space.

Logically collective.

Set the number of vectors to be added to the searching space in every iteration.

Parameters

bs (*int*) – The number of vectors added to the search space in every iteration.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2695 <slepc4py/SLEPc/EPS.pyx#L2695>`

setJDConstCorrectionTol(*constant*)

Deactivate the dynamic stopping criterion.

Logically collective.

Deactivate the dynamic stopping criterion that sets the `petsc4py.PETSc.KSP` relative tolerance to 0.5^{**i} , where *i* is the number of *EPS* iterations from the last converged value.

Parameters

constant (*bool*) – If False, the `petsc4py.PETSc.KSP` relative tolerance is set to 0.5^{**i} .

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2837 <slepc4py/SLEPc/EPS.pyx#L2837>`

setJDFix(*fix*)

Set the threshold for changing the target in the correction equation.

Logically collective.

Parameters

fix (*float*) – The threshold for changing the target.

Return type

None

Notes

The target in the correction equation is fixed at the first iterations. When the norm of the residual vector is lower than the fix value, the target is set to the corresponding eigenvalue.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2802 <slepc4py/SLEPc/EPS.pyx#L2802>`

setJDInitialSize(*initialsize*)

Set the initial size of the searching space.

Logically collective.

Parameters

initialsize (*int*) – The number of vectors of the initial searching subspace.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2773 <slepc4py/SLEPc/EPS.pyx#L2773>`

setJDKrylovStart(*krylovstart=True*)

Set (toggle) starting the search subspace with a Krylov basis.

Logically collective.

Parameters

krylovstart (*bool*) – True if starting the search subspace with a Krylov basis.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2666 <slepc4py/SLEPc/EPS.pyx#L2666>`

setJDRestart(*minv=None, plusk=None*)

Set the number of vectors of the search space after restart.

Logically collective.

Set the number of vectors of the search space after restart and the number of vectors saved from the previous iteration.

Parameters

- **minv** (*int* / *None*) – The number of vectors of the search subspace after restart.
- **plusk** (*int* / *None*) – The number of vectors saved from the previous iteration.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2730 <slepc4py/SLEPc/EPS.pyx#L2730>`

setKrylovSchurBSEType(*bse*)

Set the Krylov-Schur variant used for BSE structured eigenproblems.

Logically collective.

Parameters

bse (*KrylovSchurBSEType*) – The BSE method.

Return type

None

Notes

This call is only relevant if the type was set to *EPS.Type.KRYLOV SCHUR* with *setType()* and the problem type to *EPS.ProblemType.BSE* with *setProblemType()*.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1939 <slepc4py/SLEPc/EPS.pyx#L1939>`

setKrylovSchurDetectZeros(*detect*)

Set the flag that enforces zero detection in spectrum slicing.

Logically collective.

Set a flag to enforce the detection of zeros during the factorizations throughout the spectrum slicing computation.

Parameters

detect (*bool*) – True if zeros must checked for.

Return type

None

Notes

A zero in the factorization indicates that a shift coincides with an eigenvalue.

This flag is turned off by default, and may be necessary in some cases, especially when several partitions are being used. This feature currently requires an external package for factorizations with support for zero detection, e.g. MUMPS.

:sources: Source code at slepc4py/SLEPc/EPs.pyx:2085 <slepc4py/SLEPc/EPs.pyx#L2085>

setKrylovSchurDimensions(*nev=None, ncv=None, mpd=None*)

Set the dimensions used for each subsolve step (spectrum slicing).

Logically collective.

Set the dimensions used for each subsolve step in case of doing spectrum slicing for a computational interval. The meaning of the parameters is the same as in [setDimensions\(\)](#).

Parameters

- **nev** (*int* / *None*) – Number of eigenvalues to compute.
- **ncv** (*int* / *None*) – Maximum dimension of the subspace to be used by the solver.
- **mpd** (*int* / *None*) – Maximum dimension allowed for the projected problem.

Return type

None

:sources: Source code at slepc4py/SLEPc/EPs.pyx:2127 <slepc4py/SLEPc/EPs.pyx#L2127>

setKrylovSchurLocking(*lock*)

Set (toggle) locking/non-locking variants of the Krylov-Schur method.

Logically collective.

Parameters

- **lock** (*bool*) – True if the locking variant must be selected.

Return type

None

Notes

The default is to lock converged eigenpairs when the method restarts. This behavior can be changed so that all directions are kept in the working subspace even if already converged to working accuracy (the non-locking variant).

:sources: Source code at slepc4py/SLEPc/EPs.pyx:2009 <slepc4py/SLEPc/EPs.pyx#L2009>

setKrylovSchurPartitions(*npart*)

Set the number of partitions of the communicator (spectrum slicing).

Logically collective.

Set the number of partitions for the case of doing spectrum slicing for a computational interval with the communicator split in several sub-communicators.

Parameters

- **npart** (*int*) – The number of partitions.

Return type

None

Notes

By default, `npart=1` so all processes in the communicator participate in the processing of the whole interval. If `npart>1` then the interval is divided into `npart` subintervals, each of them being processed by a subset of processes.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2045 <slepc4py/SLEPc/EPS.pyx#L2045>`

setKrylovSchurRestart(*keep*)

Set the restart parameter for the Krylov-Schur method.

Logically collective.

It is the proportion of basis vectors that must be kept after restart.

Parameters

keep (*float*) – The number of vectors to be kept at restart.

Return type

None

Notes

Allowed values are in the range `[0.1,0.9]`. The default is `0.5`.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1974 <slepc4py/SLEPc/EPS.pyx#L1974>`

setKrylovSchurSubintervals(*subint*)

Set the subinterval boundaries.

Logically collective.

Set the subinterval boundaries for spectrum slicing with a computational interval.

Parameters

subint (*Sequence[*float*]*) – Real values specifying subintervals

Return type

None

Notes

This function must be called after `setKrylovSchurPartitions()`. For `npart` partitions, the argument `subint` must contain `npart+1` real values sorted in ascending order: `subint_0, subint_1, ..., subint_npart`, where the first and last values must coincide with the interval endpoints set with `EPSSetInterval()`. The subintervals are then defined by two consecutive points: `[subint_0,subint_1]`, `[subint_1,subint_2]`, and so on.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2342 <slepc4py/SLEPc/EPS.pyx#L2342>`

setLOBPCGBlockSize(*bs*)

Set the block size of the LOBPCG method.

Logically collective.

Parameters

bs (*int*) – The block size.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2942 <slepc4py/SLEPc/EPS.pyx#L2942>`

setLOBPCGLocking(*lock*)

Toggle between locking and non-locking (LOBPCG method).

Logically collective.

Parameters

lock (*bool*) – True if the locking variant must be selected.

Return type

None

Notes

This flag refers to soft locking (converged vectors within the current block iterate), since hard locking is always used (when nev is larger than the block size).

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:3008 <slepc4py/SLEPc/EPs.pyx#L3008>`

setLOBPCGRestart(*restart*)

Set the restart parameter for the LOBPCG method.

Logically collective.

The meaning of this parameter is the proportion of vectors within the current block iterate that must have converged in order to force a restart with hard locking.

Parameters

restart (*float*) – The percentage of the block of vectors to force a restart.

Return type

None

Notes

Allowed values are in the range [0.1,1.0]. The default is 0.9.

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:2971 <slepc4py/SLEPc/EPs.pyx#L2971>`

setLanczosReorthogType(*reorthog*)

Set the type of reorthogonalization used during the Lanczos iteration.

Logically collective.

Parameters

reorthog (*LanczosReorthogType*) – The type of reorthogonalization.

Return type

None

Notes

This call is only relevant if the type was set to *EPS.Type.LANCZOS* with *setType()*.

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:1902 <slepc4py/SLEPc/EPs.pyx#L1902>`

setLeftInitialSpace(*space*)

Set a left initial space from which the eigensolver starts to iterate.

Collective.

Parameters

space (*Vec* | *list[Vec]*) – The left initial space

Return type

None

Notes

Left initial vectors are used to initiate the left search space in two-sided eigensolvers. Users should pass here an approximation of the left eigenspace, if available.

The same comments in `setInitialSpace()` are applicable here.

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:1305 <slepc4py/SLEPc/EPs.pyx#L1305>`

setLyapIIRanks(*rkc=None, rkl=None*)

Set the ranks used in the solution of the Lyapunov equation.

Logically collective.

Parameters

- **rkc** (*int* / *None*) – The compressed rank.
- **rkl** (*int* / *None*) – The Lyapunov rank.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:3043 <slepc4py/SLEPc/EPs.pyx#L3043>`

setMonitor(*monitor, args=None, kargs=None*)

Append a monitor function to the list of monitors.

Logically collective.

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:1414 <slepc4py/SLEPc/EPs.pyx#L1414>`

Parameters

- **monitor** (*EPsMonitorFunction* / *None*)
- **args** (*tuple*[*Any*, ...] / *None*)
- **kargs** (*dict*[*str*, *Any*] / *None*)

Return type

None

setOperators(*A, B=None*)

Set the matrices associated with the eigenvalue problem.

Collective.

Parameters

- **A** (*Mat*) – The matrix associated with the eigensystem.
- **B** (*Mat* / *None*) – The second matrix in the case of generalized eigenproblems; if not provided, a standard eigenproblem is assumed.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPs.pyx:1221 <slepc4py/SLEPc/EPs.pyx#L1221>`

setOptionsPrefix(*prefix=None*)

Set the prefix used for searching for all EPS options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all EPS option requests.

Return type

None

Notes

A hyphen (-) must NOT be given at the beginning of the prefix name. The first character of all runtime options is AUTOMATICALLY the hyphen.

For example, to distinguish between the runtime options for two different EPS contexts, one could call:

```
E1.setOptionsPrefix("eig1_")
E2.setOptionsPrefix("eig2_")
```

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:394 <slepc4py/SLEPc/EPS.pyx#L394>](#)

setPowerShiftType(*shift*)

Set the type of shifts used during the power iteration.

Logically collective.

This can be used to emulate the Rayleigh Quotient Iteration (RQI) method.

Parameters

shift (*PowerShiftType*) – The type of shift.

Return type

None

Notes

This call is only relevant if the type was set to *EPS.Type.POWER* with *setType()*.

By default, shifts are constant (*EPS.PowerShiftType.CONSTANT*) and the iteration is the simple power method (or inverse iteration if a shift-and-invert transformation is being used).

A variable shift can be specified (*EPS.PowerShiftType.RAYLEIGH* or *EPS.PowerShiftType.WILKINSON*). In this case, the iteration behaves rather like a cubic converging method as RQI.

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:1816 <slepc4py/SLEPc/EPS.pyx#L1816>](#)

setProblemType(*problem_type*)

Set the type of the eigenvalue problem.

Logically collective.

Parameters

problem_type (*ProblemType*) – The problem type to be set.

Return type

None

Notes

Allowed values are: Hermitian (HEP), non-Hermitian (NHEP), generalized Hermitian (GHEP), generalized non-Hermitian (GNHEP), and generalized non-Hermitian with positive semi-definite B (PGNHEP).

This function must be used to instruct SLEPc to exploit symmetry. If no problem type is specified, by default a non-Hermitian problem is assumed (either standard or generalized). If the user knows that the problem is Hermitian (i.e. $A = A^H$) or generalized Hermitian (i.e. $A = A^H$, $B = B^H$, and B positive definite) then it is recommended to set the problem type so that eigensolver can exploit these properties.

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:469](#) [<slepc4py/SLEPc/EPS.pyx#L469>](#)

setPurify(*purify=True*)

Set (toggle) eigenvector purification.

Logically collective.

Parameters

purify (*bool*) – True to activate purification (default).

Return type

None

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:911](#) [<slepc4py/SLEPc/EPS.pyx#L911>](#)

setRG(*rg*)

Set a region object associated to the eigensolver.

Collective.

Parameters

rg (*RG*) – The region context.

Return type

None

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:1185](#) [<slepc4py/SLEPc/EPS.pyx#L1185>](#)

setRQCGReset(*nrest*)

Set the reset parameter of the RQCG iteration.

Logically collective.

Every *nrest* iterations the solver performs a Rayleigh-Ritz projection step.

Parameters

nrest (*int*) – The number of iterations between resets.

Return type

None

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:2910](#) [<slepc4py/SLEPc/EPS.pyx#L2910>](#)

setST(*st*)

Set a spectral transformation object associated to the eigensolver.

Collective.

Parameters

st (*ST*) – The spectral transformation.

Return type

None

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:1098](#) [<slepc4py/SLEPc/EPS.pyx#L1098>](#)

setStoppingTest(*stopping*, *args=None*, *kargs=None*)

Set when to stop the outer iteration of the eigensolver.

Logically collective.

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:1333 <slepc4py/SLEPc/EPS.pyx#L1333>](#)

Parameters

- **stopping** ([EPSStoppingFunction](#) | *None*)
- **args** ([tuple](#)[*Any*, ...] | *None*)
- **kargs** ([dict](#)[*str*, *Any*] | *None*)

Return type

[None](#)

setTarget(*target*)

Set the value of the target.

Logically collective.

Parameters

target ([Scalar](#)) – The value of the target.

Return type

[None](#)

Notes

The target is a scalar value used to determine the portion of the spectrum of interest. It is used in combination with [setWhichEigenpairs\(\)](#).

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:753 <slepc4py/SLEPc/EPS.pyx#L753>](#)

setThreshold(*thres*, *rel=False*)

Set the threshold used in the threshold stopping test.

Logically collective.

Parameters

- **thres** ([float](#)) – The threshold.
- **rel** ([bool](#)) – Whether the threshold is relative or not.

Return type

[None](#)

Notes

This function internally sets a special stopping test based on the threshold, where eigenvalues are computed in sequence until one of the computed eigenvalues is below/above the threshold (depending on whether largest or smallest eigenvalues are computed).

:sources: [Source code at slepc4py/SLEPc/EPS.pyx:709 <slepc4py/SLEPc/EPS.pyx#L709>](#)

setTolerances(*tol=None*, *max_it=None*)

Set the tolerance and max. iter. used by the default EPS convergence tests.

Logically collective.

Parameters

- **tol** (*float* / *None*) – The convergence tolerance.
- **max_it** (*int* / *None*) – The maximum number of iterations.

Return type

None

Notes

Use *DECIDE* for maxits to assign a reasonably good value, which is dependent on the solution method.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:843 <slepc4py/SLEPc/EPS.pyx#L843>`

setTrackAll(*trackall*)

Set if the solver must compute the residual of all approximate eigenpairs.

Logically collective.

Parameters

trackall (*bool*) – Whether compute all residuals or not.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1000 <slepc4py/SLEPc/EPS.pyx#L1000>`

setTrueResidual(*trueres*)

Set if the solver must compute the true residual explicitly or not.

Logically collective.

Parameters

trueres (*bool*) – Whether compute the true residual or not.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:971 <slepc4py/SLEPc/EPS.pyx#L971>`

setTwoSided(*twosided*)

Set to use a two-sided variant that also computes left eigenvectors.

Logically collective.

Parameters

twosided (*bool*) – Whether the two-sided variant is to be used or not.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:882 <slepc4py/SLEPc/EPS.pyx#L882>`

setType(*eps_type*)

Set the particular solver to be used in the EPS object.

Logically collective.

Parameters

eps_type (*Type* / *str*) – The solver to be used.

Return type

None

Notes

See [EPS.Type](#) for available methods. The default is [EPS.Type.KRYLOV SCHUR](#). Normally, it is best to use [setFromOptions\(\)](#) and then set the EPS type from the options database rather than by using this routine. Using the options database provides the user with maximum flexibility in evaluating the different available methods.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:340 <slepc4py/SLEPc/EPS.pyx#L340>`

setUp()

Set up all the internal data structures.

Collective.

Set up all the internal data structures necessary for the execution of the eigensolver.

Notes

This function need not be called explicitly in most cases, since [solve\(\)](#) calls it. It can be useful when one wants to measure the set-up time separately from the solve time.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1450 <slepc4py/SLEPc/EPS.pyx#L1450>`

Return type

None

setWhichEigenpairs(which)

Set which portion of the spectrum is to be sought.

Logically collective.

Parameters

which ([Which](#)) – The portion of the spectrum to be sought by the solver.

Return type

None

Notes

Not all eigensolvers implemented in EPS account for all the possible values. Also, some values make sense only for certain types of problems. If SLEPc is compiled for real numbers [EPS.Which.LARGEST_IMAGINARY](#) and [EPS.Which.SMALLEST_IMAGINARY](#) use the absolute value of the imaginary part for eigenvalue selection.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:668 <slepc4py/SLEPc/EPS.pyx#L668>`

solve()

Solve the eigensystem.

Collective.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1467 <slepc4py/SLEPc/EPS.pyx#L1467>`

Return type

None

updateKrylovSchurSubcommMats(*s=1.0, a=1.0, Au=None, t=1.0, b=1.0, Bu=None, structure=None, globalup=False*)

Update the eigenproblem matrices stored internally in the communicator.

Collective.

Update the eigenproblem matrices stored internally in the subcommunicator to which the calling process belongs.

Parameters

- **s** (*Scalar*) – Scalar that multiplies the existing A matrix.
- **a** (*Scalar*) – Scalar used in the axpy operation on A.
- **Au** (*petsc4py.PETSc.Mat* / *None*) – The matrix used in the axpy operation on A.
- **t** (*Scalar*) – Scalar that multiplies the existing B matrix.
- **b** (*Scalar*) – Scalar used in the axpy operation on B.
- **Bu** (*petsc4py.PETSc.Mat* / *None*) – The matrix used in the axpy operation on B.
- **structure** (*petsc4py.PETSc.Mat.Structure* / *None*) – Either same, different, or a subset of the non-zero sparsity pattern.
- **globalup** (*bool*) – Whether global matrices must be updated or not.

Return type

None

Notes

This function modifies the eigenproblem matrices at subcommunicator level, and optionally updates the global matrices in the parent communicator. The updates are expressed as $A \leftarrow sA + aAu$, $B \leftarrow tB + bBu$.

It is possible to update one of the matrices, or both.

The matrices Au and Bu must be equal in all subcommunicators.

The structure flag is passed to the `petsc4py.PETSc.Mat.axpy` operations to perform the updates.

If `globalup` is True, communication is carried out to reconstruct the updated matrices in the parent communicator.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:2279 <slepc4py/SLEPc/EPS.pyx#L2279>`

valuesView(*viewer=None*)

Display the computed eigenvalues in a viewer.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:1784 <slepc4py/SLEPc/EPS.pyx#L1784>`

vectorsView(*viewer=None*)

Output computed eigenvectors to a viewer.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

view(*viewer=None*)
 Print the EPS data structure.
 Collective.

Parameters
viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type
None

Source code at slepc4py/SLEPc/EPS.pyx:1799 <slepc4py/SLEPc/EPS.pyx#L1799>

Source code at slepc4py/SLEPc/EPS.pyx:290 <slepc4py/SLEPc/EPS.pyx#L290>

Attributes Documentation

bv
 The basis vectors (BV) object associated.

Source code at slepc4py/SLEPc/EPS.pyx:3427 <slepc4py/SLEPc/EPS.pyx#L3427>

ds
 The direct solver (DS) object associated.

Source code at slepc4py/SLEPc/EPS.pyx:3441 <slepc4py/SLEPc/EPS.pyx#L3441>

extraction
 The type of extraction technique to be employed.

Source code at slepc4py/SLEPc/EPS.pyx:3357 <slepc4py/SLEPc/EPS.pyx#L3357>

max_it
 The maximum iteration count.

Source code at slepc4py/SLEPc/EPS.pyx:3385 <slepc4py/SLEPc/EPS.pyx#L3385>

problem_type
 The type of the eigenvalue problem.

Source code at slepc4py/SLEPc/EPS.pyx:3350 <slepc4py/SLEPc/EPS.pyx#L3350>

purify
 Eigenvector purification.

Source code at slepc4py/SLEPc/EPS.pyx:3406 <slepc4py/SLEPc/EPS.pyx#L3406>

rg
 The region (RG) object associated.

Source code at slepc4py/SLEPc/EPS.pyx:3434 <slepc4py/SLEPc/EPS.pyx#L3434>

st
 The spectral transformation (ST) object associated.

Source code at slepc4py/SLEPc/EPS.pyx:3420 <slepc4py/SLEPc/EPS.pyx#L3420>

target
 The value of the target.

Source code at slepc4py/SLEPc/EPS.pyx:3371 <slepc4py/SLEPc/EPS.pyx#L3371>

tol

The tolerance.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3378 <slepc4py/SLEPc/EPS.pyx#L3378>`

track_all

Compute the residual norm of all approximate eigenpairs.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3413 <slepc4py/SLEPc/EPS.pyx#L3413>`

true_residual

Compute the true residual explicitly.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3399 <slepc4py/SLEPc/EPS.pyx#L3399>`

two_sided

Two-sided that also computes left eigenvectors.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3392 <slepc4py/SLEPc/EPS.pyx#L3392>`

which

The portion of the spectrum to be sought.

:sources: `Source code at slepc4py/SLEPc/EPS.pyx:3364 <slepc4py/SLEPc/EPS.pyx#L3364>`

slepc4py.SLEPc.EPSKrylovSchurBSEType

class `slepc4py.SLEPc.EPSKrylovSchurBSEType`

Bases: `object`

EPS Krylov-Schur method for BSE problems.

- *SHAO*: Lanczos recurrence for H square.
- *GRUNING*: Lanczos recurrence for H.
- *PROJECTEDBSE*: Lanczos where the projected problem has BSE structure.

Attributes Summary

<i>GRUNING</i>	Constant GRUNING of type <code>int</code>
<i>PROJECTEDBSE</i>	Constant PROJECTEDBSE of type <code>int</code>
<i>SHAO</i>	Constant SHAO of type <code>int</code>

Attributes Documentation

GRUNING: `int` = `GRUNING`

Constant GRUNING of type `int`

PROJECTEDBSE: `int` = `PROJECTEDBSE`

Constant PROJECTEDBSE of type `int`

SHAO: `int` = `SHAO`

Constant SHAO of type `int`

slepc4py.SLEPc.FN

class slepc4py.SLEPc.FN

Bases: `Object`

FN.

Enumerations

<i>CombineType</i>	FN type of combination of child functions.
<i>ParallelType</i>	FN parallel types.
<i>Type</i>	FN type.

slepc4py.SLEPc.FN.CombineType

class slepc4py.SLEPc.FN.CombineType

Bases: `object`

FN type of combination of child functions.

- *ADD*: Addition $f(x) = f1(x) + f2(x)$
- *MULTIPLY*: Multiplication $f(x) = f1(x) * f2(x)$
- *DIVIDE*: Division $f(x) = f1(x) / f2(x)$
- *COMPOSE*: Composition $f(x) = f2(f1(x))$

Attributes Summary

<i>ADD</i>	Constant ADD of type <code>int</code>
<i>COMPOSE</i>	Constant COMPOSE of type <code>int</code>
<i>DIVIDE</i>	Constant DIVIDE of type <code>int</code>
<i>MULTIPLY</i>	Constant MULTIPLY of type <code>int</code>

Attributes Documentation

ADD: `int` = **ADD**

Constant ADD of type `int`

COMPOSE: `int` = **COMPOSE**

Constant COMPOSE of type `int`

DIVIDE: `int` = **DIVIDE**

Constant DIVIDE of type `int`

MULTIPLY: `int` = **MULTIPLY**

Constant MULTIPLY of type `int`

slepc4py.SLEPc.FN.ParallelType

class slepc4py.SLEPc.FN.ParallelType

Bases: `object`

FN parallel types.

- *REDUNDANT*: Every process performs the computation redundantly.
- *SYNCHRONIZED*: The first process sends the result to the rest.

Attributes Summary

<i>REDUNDANT</i>	Constant REDUNDANT of type <i>int</i>
<i>SYNCHRONIZED</i>	Constant SYNCHRONIZED of type <i>int</i>

Attributes Documentation

REDUNDANT: *int* = REDUNDANT

Constant REDUNDANT of type *int*

SYNCHRONIZED: *int* = SYNCHRONIZED

Constant SYNCHRONIZED of type *int*

slepc4py.SLEPc.FN.Type

class slepc4py.SLEPc.FN.Type

Bases: *object*

FN type.

Attributes Summary

<i>COMBINE</i>	Object COMBINE of type <i>str</i>
<i>EXP</i>	Object EXP of type <i>str</i>
<i>INVSQRT</i>	Object INVSQRT of type <i>str</i>
<i>LOG</i>	Object LOG of type <i>str</i>
<i>PHI</i>	Object PHI of type <i>str</i>
<i>RATIONAL</i>	Object RATIONAL of type <i>str</i>
<i>SQRT</i>	Object SQRT of type <i>str</i>

Attributes Documentation

COMBINE: *str* = COMBINE

Object COMBINE of type *str*

EXP: *str* = EXP

Object EXP of type *str*

INVSQRT: *str* = INVSQRT

Object INVSQRT of type *str*

LOG: *str* = LOG

Object LOG of type *str*

PHI: *str* = PHI

Object PHI of type *str*

RATIONAL: *str* = RATIONAL

Object RATIONAL of type *str*

SQRT: `str` = SQRT

Object SQRT of type `str`

Methods Summary

<code>appendOptionsPrefix([prefix])</code>	Append to the prefix used for searching for all FN options in the database.
<code>create([comm])</code>	Create the FN object.
<code>destroy()</code>	Destroy the FN object.
<code>duplicate([comm])</code>	Duplicate the FN object copying all parameters.
<code>evaluateDerivative(x)</code>	Compute the value of the derivative $f'(x)$ for a given x .
<code>evaluateFunction(x)</code>	Compute the value of the function $f(x)$ for a given x .
<code>evaluateFunctionMat(A[, B])</code>	Compute the value of the function $f(A)$ for a given matrix A .
<code>evaluateFunctionMatVec(A[, v])</code>	Compute the first column of the matrix $f(A)$ for a given matrix A .
<code>getCombineChildren()</code>	Get the two child functions that constitute this combined function.
<code>getMethod()</code>	Get the method currently used for matrix functions.
<code>getOptionsPrefix()</code>	Get the prefix used for searching for all FN options in the database.
<code>getParallel()</code>	Get the mode of operation in parallel runs.
<code>getPhiIndex()</code>	Get the index of the phi-function.
<code>getRationalDenominator()</code>	Get the coefficients of the denominator of the rational function.
<code>getRationalNumerator()</code>	Get the coefficients of the numerator of the rational function.
<code>getScale()</code>	Get the scaling parameters that define the mathematical function.
<code>getType()</code>	Get the FN type of this object.
<code>setCombineChildren(comb, f1, f2)</code>	Set the two child functions that constitute this combined function.
<code>setFromOptions()</code>	Set FN options from the options database.
<code>setMethod(meth)</code>	Set the method to be used to evaluate functions of matrices.
<code>setOptionsPrefix([prefix])</code>	Set the prefix used for searching for all FN options in the database.
<code>setParallel(pmode)</code>	Set the mode of operation in parallel runs.
<code>setPhiIndex(k)</code>	Set the index of the phi-function.
<code>setRationalDenominator(alpha)</code>	Set the coefficients of the denominator of the rational function.
<code>setRationalNumerator(alpha)</code>	Set the coefficients of the numerator of the rational function.
<code>setScale([alpha, beta])</code>	Set the scaling parameters that define the mathematical function.
<code>setType(fn_type)</code>	Set the type for the FN object.
<code>view([viewer])</code>	Print the FN data structure.

Attributes Summary

<i>method</i>	The method to be used to evaluate functions of matrices.
<i>parallel</i>	The mode of operation in parallel runs.

Methods Documentation

appendOptionsPrefix(*prefix=None*)

Append to the prefix used for searching for all FN options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all FN option requests.

Return type

None

:sources: `Source code at slepc4py/SLEPc/FN.pyx:212 <slepc4py/SLEPc/FN.pyx#L212>`

create(*comm=None*)

Create the FN object.

Collective.

Parameters

comm (*Comm* / *None*) – MPI communicator; if not provided, it defaults to all processes.

Return type

Self

:sources: `Source code at slepc4py/SLEPc/FN.pyx:144 <slepc4py/SLEPc/FN.pyx#L144>`

destroy()

Destroy the FN object.

Collective.

:sources: `Source code at slepc4py/SLEPc/FN.pyx:134 <slepc4py/SLEPc/FN.pyx#L134>`

Return type

Self

duplicate(*comm=None*)

Duplicate the FN object copying all parameters.

Collective.

Duplicate the FN object copying all parameters, possibly with a different communicator.

Parameters

comm (*Comm* / *None*) – MPI communicator; if not provided, it defaults to the object's communicator.

Return type

FN

:sources: `Source code at slepc4py/SLEPc/FN.pyx:255 <slepc4py/SLEPc/FN.pyx#L255>`

evaluateDerivative(x)

Compute the value of the derivative $f'(x)$ for a given x .

Not collective.

Parameters

x (*Scalar*) – Value where the derivative must be evaluated.

Returns

The result of $f'(x)$.

Return type

Scalar

:sources: `Source code at slepc4py/SLEPc/FN.pyx:298 <slepc4py/SLEPc/FN.pyx#L298>`

evaluateFunction(x)

Compute the value of the function $f(x)$ for a given x .

Not collective.

Parameters

x (*Scalar*) – Value where the function must be evaluated.

Returns

The result of $f(x)$.

Return type

Scalar

:sources: `Source code at slepc4py/SLEPc/FN.pyx:277 <slepc4py/SLEPc/FN.pyx#L277>`

evaluateFunctionMat(A, B=None)

Compute the value of the function $f(A)$ for a given matrix A .

Logically collective.

Parameters

- **A** (*petsc4py.PETSc.Mat*) – Matrix on which the function must be evaluated.
- **B** (*petsc4py.PETSc.Mat* | *None*) – Placeholder for the result.

Returns

The result of $f(A)$.

Return type

petsc4py.PETSc.Mat

:sources: `Source code at slepc4py/SLEPc/FN.pyx:319 <slepc4py/SLEPc/FN.pyx#L319>`

evaluateFunctionMatVec(A, v=None)

Compute the first column of the matrix $f(A)$ for a given matrix A .

Logically collective.

Parameters

- **A** (*petsc4py.PETSc.Mat*) – Matrix on which the function must be evaluated.
- **v** (*petsc4py.PETSc.Vec* | *None*)

Returns

The first column of the result $f(A)$.

Return type`petsc4py.PETSc.Vec`**:sources:** `Source code at slepc4py/SLEPc/FN.pyx:341 <slepc4py/SLEPc/FN.pyx#L341>`**getCombineChildren()**

Get the two child functions that constitute this combined function.

Not collective.

Get the two child functions that constitute this combined function, and the way they must be combined.

Returns

- **comb** (*CombineType*) – How to combine the functions (addition, multiplication, division, composition).
- **f1** (*FN*) – First function.
- **f2** (*FN*) – Second function.

Return type`tuple[CombineType, FN, FN]`**:sources:** `Source code at slepc4py/SLEPc/FN.pyx:564 <slepc4py/SLEPc/FN.pyx#L564>`**getMethod()**

Get the method currently used for matrix functions.

Not collective.

Returns

An index identifying the method.

Return type`int`**:sources:** `Source code at slepc4py/SLEPc/FN.pyx:422 <slepc4py/SLEPc/FN.pyx#L422>`**getOptionsPrefix()**

Get the prefix used for searching for all FN options in the database.

Not collective.

Returns

The prefix string set for this FN object.

Return type`str`**:sources:** `Source code at slepc4py/SLEPc/FN.pyx:227 <slepc4py/SLEPc/FN.pyx#L227>`**getParallel()**

Get the mode of operation in parallel runs.

Not collective.

Returns

The parallel mode.

Return type`ParallelType`**:sources:** `Source code at slepc4py/SLEPc/FN.pyx:451 <slepc4py/SLEPc/FN.pyx#L451>`

getPhiIndex()

Get the index of the phi-function.

Not collective.

Returns

The index.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/FN.pyx:605 <slepc4py/SLEPc/FN.pyx#L605>`

getRationalDenominator()

Get the coefficients of the denominator of the rational function.

Not collective.

Returns

Coefficients.

Return type

`ArrayScalar`

:sources: `Source code at slepc4py/SLEPc/FN.pyx:521 <slepc4py/SLEPc/FN.pyx#L521>`

getRationalNumerator()

Get the coefficients of the numerator of the rational function.

Not collective.

Returns

Coefficients.

Return type

`ArrayScalar`

:sources: `Source code at slepc4py/SLEPc/FN.pyx:484 <slepc4py/SLEPc/FN.pyx#L484>`

getScale()

Get the scaling parameters that define the mathematical function.

Not collective.

Returns

- **alpha** (`Scalar`) – Inner scaling (argument).
- **beta** (`Scalar`) – Outer scaling (result).

Return type

`tuple[Scalar, Scalar]`

:sources: `Source code at slepc4py/SLEPc/FN.pyx:380 <slepc4py/SLEPc/FN.pyx#L380>`

getType()

Get the FN type of this object.

Not collective.

Returns

The inner product type currently being used.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/FN.pyx:176 <slepc4py/SLEPc/FN.pyx#L176>`

setCombineChildren(*comb*, *f1*, *f2*)

Set the two child functions that constitute this combined function.

Logically collective.

Set the two child functions that constitute this combined function, and the way they must be combined.

Parameters

- **comb** ([CombineType](#)) – How to combine the functions (addition, multiplication, division, composition).
- **f1** ([FN](#)) – First function.
- **f2** ([FN](#)) – Second function.

Return type

[None](#)

:sources: `Source code at slepc4py/SLEPc/FN.pyx:542 <slepc4py/SLEPc/FN.pyx#L542>`

setFromOptions()

Set FN options from the options database.

Collective.

Notes

To see all options, run your program with the `-help` option.

:sources: `Source code at slepc4py/SLEPc/FN.pyx:242 <slepc4py/SLEPc/FN.pyx#L242>`

Return type

[None](#)

setMethod(*meth*)

Set the method to be used to evaluate functions of matrices.

Logically collective.

Parameters

- **meth** ([int](#)) – An index identifying the method.

Return type

[None](#)

Notes

In some [FN](#) types there are more than one algorithms available for computing matrix functions. In that case, this function allows choosing the wanted method.

If *meth* is currently set to 0 and the input argument of [FN.evaluateFunctionMat\(\)](#) is a symmetric/Hermitian matrix, then the computation is done via the eigendecomposition, rather than with the general algorithm.

:sources: `Source code at slepc4py/SLEPc/FN.pyx:397 <slepc4py/SLEPc/FN.pyx#L397>`

setOptionsPrefix(*prefix=None*)

Set the prefix used for searching for all FN options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all FN option requests.

Return type

None

Notes

A hyphen (-) must NOT be given at the beginning of the prefix name. The first character of all runtime options is AUTOMATICALLY the hyphen.

:sources: `Source code at slepc4py/SLEPc/FN.pyx:191 <slepc4py/SLEPc/FN.pyx#L191>`

setParallel(*pmode*)

Set the mode of operation in parallel runs.

Logically collective.

Parameters

pmode (*ParallelType*) – The parallel mode.

Return type

None

:sources: `Source code at slepc4py/SLEPc/FN.pyx:437 <slepc4py/SLEPc/FN.pyx#L437>`

setPhiIndex(*k*)

Set the index of the phi-function.

Logically collective.

Parameters

k (*int*) – The index.

Return type

None

:sources: `Source code at slepc4py/SLEPc/FN.pyx:591 <slepc4py/SLEPc/FN.pyx#L591>`

setRationalDenominator(*alpha*)

Set the coefficients of the denominator of the rational function.

Logically collective.

Parameters

alpha (*Sequence*[*Scalar*]) – Coefficients.

Return type

None

:sources: `Source code at slepc4py/SLEPc/FN.pyx:505 <slepc4py/SLEPc/FN.pyx#L505>`

setRationalNumerator(*alpha*)

Set the coefficients of the numerator of the rational function.

Logically collective.

Parameters

alpha (*Sequence*[*Scalar*]) – Coefficients.

Return type

None

:sources: `Source code at slepc4py/SLEPc/FN.pyx:468 <slepc4py/SLEPc/FN.pyx#L468>`

setScale(*alpha=None, beta=None*)

Set the scaling parameters that define the mathematical function.

Logically collective.

Parameters

- **alpha** (*Scalar* / *None*) – Inner scaling (argument), default is 1.0.
- **beta** (*Scalar* / *None*) – Outer scaling (result), default is 1.0.

Return type

None

:sources: `Source code at slepc4py/SLEPc/FN.pyx:361 <slepc4py/SLEPc/FN.pyx#L361>`

setType(*fn_type*)

Set the type for the FN object.

Logically collective.

Parameters

fn_type (*Type* / *str*) – The inner product type to be used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/FN.pyx:161 <slepc4py/SLEPc/FN.pyx#L161>`

view(*viewer=None*)

Print the FN data structure.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/FN.pyx:119 <slepc4py/SLEPc/FN.pyx#L119>`

Attributes Documentation

method

The method to be used to evaluate functions of matrices.

:sources: `Source code at slepc4py/SLEPc/FN.pyx:622 <slepc4py/SLEPc/FN.pyx#L622>`

parallel

The mode of operation in parallel runs.

:sources: `Source code at slepc4py/SLEPc/FN.pyx:629 <slepc4py/SLEPc/FN.pyx#L629>`

slepc4py.SLEPc.LME

class `slepc4py.SLEPc.LME`

Bases: *Object*

LME.

Enumerations

<i>ConvergedReason</i>	LME convergence reasons.
<i>ProblemType</i>	LME problem type.
<i>Type</i>	LME type.

slepc4py.SLEPc.LME.ConvergedReason

class slepc4py.SLEPc.LME.ConvergedReason

Bases: `object`

LME convergence reasons.

- *CONVERGED_TOL*: All eigenpairs converged to requested tolerance.
- *DIVERGED_ITS*: Maximum number of iterations exceeded.
- *DIVERGED_BREAKDOWN*: Solver failed due to breakdown.
- *CONVERGED_ITERATING*: Iteration not finished yet.

Attributes Summary

<i>CONVERGED_ITERATING</i>	Constant <i>CONVERGED_ITERATING</i> of type <code>int</code>
<i>CONVERGED_TOL</i>	Constant <i>CONVERGED_TOL</i> of type <code>int</code>
<i>DIVERGED_BREAKDOWN</i>	Constant <i>DIVERGED_BREAKDOWN</i> of type <code>int</code>
<i>DIVERGED_ITS</i>	Constant <i>DIVERGED_ITS</i> of type <code>int</code>
<i>ITERATING</i>	Constant <i>ITERATING</i> of type <code>int</code>

Attributes Documentation

CONVERGED_ITERATING: `int` = *CONVERGED_ITERATING*

Constant *CONVERGED_ITERATING* of type `int`

CONVERGED_TOL: `int` = *CONVERGED_TOL*

Constant *CONVERGED_TOL* of type `int`

DIVERGED_BREAKDOWN: `int` = *DIVERGED_BREAKDOWN*

Constant *DIVERGED_BREAKDOWN* of type `int`

DIVERGED_ITS: `int` = *DIVERGED_ITS*

Constant *DIVERGED_ITS* of type `int`

ITERATING: `int` = *ITERATING*

Constant *ITERATING* of type `int`

slepc4py.SLEPc.LME.ProblemType

class slepc4py.SLEPc.LME.ProblemType

Bases: `object`

LME problem type.

- *LYAPUNOV*: Continuous-time Lyapunov.
- *SYLVESTER*: Continuous-time Sylvester.

- *GEN_LYAPUNOV*: Generalized Lyapunov.
- *GEN_SYLVESTER*: Generalized Sylvester.
- *DT_LYAPUNOV*: Discrete-time Lyapunov.
- *STEIN*: Stein.

Attributes Summary

<i>DT_LYAPUNOV</i>	Constant DT_LYAPUNOV of type <i>int</i>
<i>GEN_LYAPUNOV</i>	Constant GEN_LYAPUNOV of type <i>int</i>
<i>GEN_SYLVESTER</i>	Constant GEN_SYLVESTER of type <i>int</i>
<i>LYAPUNOV</i>	Constant LYAPUNOV of type <i>int</i>
<i>STEIN</i>	Constant STEIN of type <i>int</i>
<i>SYLVESTER</i>	Constant SYLVESTER of type <i>int</i>

Attributes Documentation

DT_LYAPUNOV: *int* = DT_LYAPUNOV

Constant DT_LYAPUNOV of type *int*

GEN_LYAPUNOV: *int* = GEN_LYAPUNOV

Constant GEN_LYAPUNOV of type *int*

GEN_SYLVESTER: *int* = GEN_SYLVESTER

Constant GEN_SYLVESTER of type *int*

LYAPUNOV: *int* = LYAPUNOV

Constant LYAPUNOV of type *int*

STEIN: *int* = STEIN

Constant STEIN of type *int*

SYLVESTER: *int* = SYLVESTER

Constant SYLVESTER of type *int*

slepc4py.SLEPc.LME.Type

class slepc4py.SLEPc.LME.Type

Bases: *object*

LME type.

- *KRYLOV*: Restarted Krylov solver.

Attributes Summary

<i>KRYLOV</i>	Object KRYLOV of type <i>str</i>
---------------	----------------------------------

Attributes Documentation

KRYLOV: *str* = KRYLOV

Object KRYLOV of type *str*

Methods Summary

<code>appendOptionsPrefix([prefix])</code>	Append to the prefix used for searching in the database.
<code>cancelMonitor()</code>	Clear all monitors for an <i>LME</i> object.
<code>computeError()</code>	Compute the error associated with the last equation solved.
<code>create([comm])</code>	Create the LME object.
<code>destroy()</code>	Destroy the LME object.
<code>getBV()</code>	Get the basis vector object associated to the LME object.
<code>getCoefficients()</code>	Get the coefficient matrices of the matrix equation.
<code>getConvergedReason()</code>	Get the reason why the <code>solve()</code> iteration was stopped.
<code>getDimensions()</code>	Get the dimension of the subspace used by the solver.
<code>getErrorEstimate()</code>	Get the error estimate obtained during solve.
<code>getErrorIfNotConverged()</code>	Get if <code>solve()</code> generates an error if the solver does not converge.
<code>getIterationNumber()</code>	Get the current iteration number.
<code>getMonitor()</code>	Get the list of monitor functions.
<code>getOptionsPrefix()</code>	Get the prefix used for searching for all LME options in the database.
<code>getProblemType()</code>	Get the LME problem type of this object.
<code>getRHS()</code>	Get the right-hand side of the matrix equation.
<code>getSolution()</code>	Get the solution of the matrix equation.
<code>getTolerances()</code>	Get the tolerance and maximum iteration count.
<code>getType()</code>	Get the LME type of this object.
<code>reset()</code>	Reset the LME object.
<code>setBV(bv)</code>	Set a basis vector object to the LME object.
<code>setCoefficients(A[, B, D, E])</code>	Set the coefficient matrices.
<code>setDimensions(ncv)</code>	Set the dimension of the subspace to be used by the solver.
<code>setErrorIfNotConverged([flg])</code>	Set <code>solve()</code> to generate an error if the solver has not converged.
<code>setFromOptions()</code>	Set LME options from the options database.
<code>setMonitor(monitor[, args, kargs])</code>	Append a monitor function to the list of monitors.
<code>setOptionsPrefix([prefix])</code>	Set the prefix used for searching for all LME options in the database.
<code>setProblemType(lme_problem_type)</code>	Set the LME problem type of this object.
<code>setRHS(C)</code>	Set the right-hand side of the matrix equation.
<code>setSolution(X)</code>	Set the placeholder for the solution of the matrix equation.
<code>setTolerances([tol, max_it])</code>	Set the tolerance and maximum iteration count.
<code>setType(lme_type)</code>	Set the particular solver to be used in the LME object.
<code>setUp()</code>	Set up all the internal necessary data structures.
<code>solve()</code>	Solve the linear matrix equation.
<code>view([viewer])</code>	Print the LME data structure.

Attributes Summary

<i>bv</i>	The basis vectors (BV) object associated to the LME object.
<i>fn</i>	The math function (FN) object associated to the LME object.
<i>max_it</i>	The maximum iteration count used by the LME convergence tests.
<i>tol</i>	The tolerance value used by the LME convergence tests.

Methods Documentation

appendOptionsPrefix(*prefix=None*)

Append to the prefix used for searching in the database.

Logically collective.

Append to the prefix used for searching for all LME options in the database.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all LME option requests.

Return type

None

:sources: `Source code at slepc4py/SLEPc/LME.pyx:350 <slepc4py/SLEPc/LME.pyx#L350>`

cancelMonitor()

Clear all monitors for an *LME* object.

:sources: `Source code at slepc4py/SLEPc/LME.pyx:508 <slepc4py/SLEPc/LME.pyx#L508>`

Return type

None

computeError()

Compute the error associated with the last equation solved.

Collective.

Computes the error (based on the residual norm) associated with the last equation solved.

Returns

The error

Return type

float

:sources: `Source code at slepc4py/SLEPc/LME.pyx:302 <slepc4py/SLEPc/LME.pyx#L302>`

create(*comm=None*)

Create the LME object.

Collective.

Parameters

comm (*Comm* / *None*) – MPI communicator. If not provided, it defaults to all processes.

Return type

Self

:sources:`Source code at slepc4py/SLEPc/LME.pyx:91 <slepc4py/SLEPc/LME.pyx#L91>`

destroy()

Destroy the LME object.

Collective.

:sources:`Source code at slepc4py/SLEPc/LME.pyx:73 <slepc4py/SLEPc/LME.pyx#L73>`

Return type

Self

getBV()

Get the basis vector object associated to the LME object.

Not collective.

Returns

The basis vectors context.

Return type

BV

:sources:`Source code at slepc4py/SLEPc/LME.pyx:452 <slepc4py/SLEPc/LME.pyx#L452>`

getCoefficients()

Get the coefficient matrices of the matrix equation.

Collective.

Returns

- A – First coefficient matrix
- B – Second coefficient matrix, if available
- D – Third coefficient matrix, if available
- E – Fourth coefficient matrix, if available

Return type

tuple[Mat, Mat | None, Mat | None, Mat | None]

:sources:`Source code at slepc4py/SLEPc/LME.pyx:193 <slepc4py/SLEPc/LME.pyx#L193>`

getConvergedReason()

Get the reason why the *solve()* iteration was stopped.

Not collective.

Returns

Negative value indicates diverged, positive value converged.

Return type

ConvergedReason

:sources:`Source code at slepc4py/SLEPc/LME.pyx:552 <slepc4py/SLEPc/LME.pyx#L552>`

getDimensions()

Get the dimension of the subspace used by the solver.

Not collective.

Returns

Maximum dimension of the subspace to be used by the solver.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/LME.pyx:423 <slepc4py/SLEPc/LME.pyx#L423>`

getErrorEstimate()

Get the error estimate obtained during solve.

Not collective.

Returns

The error estimate

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/LME.pyx:287 <slepc4py/SLEPc/LME.pyx#L287>`

getErrorIfNotConverged()

Get if `solve()` generates an error if the solver does not converge.

Not collective.

Get a flag indicating whether `solve()` will generate an error if the solver does not converge.

Returns

True indicates you want the error generated.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/LME.pyx:581 <slepc4py/SLEPc/LME.pyx#L581>`

getIterationNumber()

Get the current iteration number.

Not collective.

If the call to `solve()` is complete, then it returns the number of iterations carried out by the solution method.

Returns

Iteration number.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/LME.pyx:534 <slepc4py/SLEPc/LME.pyx#L534>`

getMonitor()

Get the list of monitor functions.

:sources: `Source code at slepc4py/SLEPc/LME.pyx:502 <slepc4py/SLEPc/LME.pyx#L502>`

Return type

`LMEMonitorFunction`

getOptionsPrefix()

Get the prefix used for searching for all LME options in the database.

Not collective.

Returns

The prefix string set for this LME object.

Return type

str

:sources: `Source code at slepc4py/SLEPc/LME.pyx:320 <slepc4py/SLEPc/LME.pyx#L320>`

getProblemType()

Get the LME problem type of this object.

Not collective.

Returns

The problem type currently being used.

Return type

ProblemType

:sources: `Source code at slepc4py/SLEPc/LME.pyx:152 <slepc4py/SLEPc/LME.pyx#L152>`

getRHS()

Get the right-hand side of the matrix equation.

Collective.

Returns

The low-rank matrix

Return type

C

:sources: `Source code at slepc4py/SLEPc/LME.pyx:239 <slepc4py/SLEPc/LME.pyx#L239>`

getSolution()

Get the solution of the matrix equation.

Collective.

Returns

The low-rank matrix

Return type

X

:sources: `Source code at slepc4py/SLEPc/LME.pyx:271 <slepc4py/SLEPc/LME.pyx#L271>`

getTolerances()

Get the tolerance and maximum iteration count.

Not collective.

Get the tolerance and maximum iteration count used by the default LME convergence tests.

Returns

- **tol** (*float*) – The convergence tolerance.
- **max_it** (*int*) – The maximum number of iterations

Return type

*tuple[*float*, *int*]*

:sources: `Source code at slepc4py/SLEPc/LME.pyx:380 <slepc4py/SLEPc/LME.pyx#L380>`

getType()

Get the LME type of this object.

Not collective.

Returns

The solver currently being used.

Return type

`str`

`:sources: Source code at slepc4py/SLEPc/LME.pyx:123 <slepc4py/SLEPc/LME.pyx#L123>`

reset()

Reset the LME object.

Collective.

`:sources: Source code at slepc4py/SLEPc/LME.pyx:83 <slepc4py/SLEPc/LME.pyx#L83>`

Return type

`None`

setBV(*bv*)

Set a basis vector object to the LME object.

Collective.

Parameters

bv (`BV`) – The basis vectors context.

Return type

`None`

`:sources: Source code at slepc4py/SLEPc/LME.pyx:468 <slepc4py/SLEPc/LME.pyx#L468>`

setCoefficients(*A*, *B=None*, *D=None*, *E=None*)

Set the coefficient matrices.

Collective.

Set the coefficient matrices that define the linear matrix equation to be solved.

Parameters

- ***A*** (`Mat`) – First coefficient matrix
- ***B*** (`Mat` | `None`) – Second coefficient matrix, optional
- ***D*** (`Mat` | `None`) – Third coefficient matrix, optional
- ***E*** (`Mat` | `None`) – Fourth coefficient matrix, optional

Return type

`None`

`:sources: Source code at slepc4py/SLEPc/LME.pyx:167 <slepc4py/SLEPc/LME.pyx#L167>`

setDimensions(*ncv*)

Set the dimension of the subspace to be used by the solver.

Logically collective.

Parameters

ncv (`int`) – Maximum dimension of the subspace to be used by the solver.

Return type

`None`

`:sources: Source code at slepc4py/SLEPc/LME.pyx:438 <slepc4py/SLEPc/LME.pyx#L438>`

setErrorIfNotConverged(*flag=True*)

Set `solve()` to generate an error if the solver has not converged.

Logically collective.

Parameters

flag (*bool*) – True indicates you want the error generated.

Return type

None

:sources: `Source code at slepc4py/SLEPc/LME.pyx:567 <slepc4py/SLEPc/LME.pyx#L567>`

setFromOptions()

Set LME options from the options database.

Collective.

Sets LME options from the options database. This routine must be called before `setUp()` if the user is to be allowed to set the solver type.

:sources: `Source code at slepc4py/SLEPc/LME.pyx:368 <slepc4py/SLEPc/LME.pyx#L368>`

Return type

None

setMonitor(*monitor, args=None, kargs=None*)

Append a monitor function to the list of monitors.

Logically collective.

:sources: `Source code at slepc4py/SLEPc/LME.pyx:481 <slepc4py/SLEPc/LME.pyx#L481>`

Parameters

- **monitor** (*LMEMonitorFunction* / *None*)
- **args** (*tuple*[*Any*, ...] / *None*)
- **kargs** (*dict*[*str*, *Any*] / *None*)

Return type

None

setOptionsPrefix(*prefix=None*)

Set the prefix used for searching for all LME options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all LME option requests.

Return type

None

:sources: `Source code at slepc4py/SLEPc/LME.pyx:335 <slepc4py/SLEPc/LME.pyx#L335>`

setProblemType(*lme_problem_type*)

Set the LME problem type of this object.

Logically collective.

Parameters

lme_problem_type (*ProblemType* / *str*) – The problem type to be used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/LME.pyx:138 <slepc4py/SLEPc/LME.pyx#L138>`

setRHS(*C*)

Set the right-hand side of the matrix equation.

Collective.

Set the right-hand side of the matrix equation, as a low-rank matrix.

Parameters

C (*Mat*) – The right-hand side matrix

Return type

None

:sources: `Source code at slepc4py/SLEPc/LME.pyx:223 <slepc4py/SLEPc/LME.pyx#L223>`

setSolution(*X*)

Set the placeholder for the solution of the matrix equation.

Collective.

Set the placeholder for the solution of the matrix equation, as a low-rank matrix.

Parameters

X (*Mat*) – The solution matrix

Return type

None

:sources: `Source code at slepc4py/SLEPc/LME.pyx:255 <slepc4py/SLEPc/LME.pyx#L255>`

setTolerances(*tol=None, max_it=None*)

Set the tolerance and maximum iteration count.

Logically collective.

Set the tolerance and maximum iteration count used by the default LME convergence tests.

Parameters

- **tol** (*float* / *None*) – The convergence tolerance.
- **max_it** (*int* / *None*) – The maximum number of iterations

Return type

None

:sources: `Source code at slepc4py/SLEPc/LME.pyx:401 <slepc4py/SLEPc/LME.pyx#L401>`

setType(*lme_type*)

Set the particular solver to be used in the LME object.

Logically collective.

Parameters

lme_type (*Type* / *str*) – The solver to be used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/LME.pyx:108 <slepc4py/SLEPc/LME.pyx#L108>`

setUp()

Set up all the internal necessary data structures.

Collective.

Set up all the internal data structures necessary for the execution of the eigensolver.

:sources: `Source code at slepc4py/SLEPc/LME.pyx:515 <slepc4py/SLEPc/LME.pyx#L515>`

Return type

None

solve()

Solve the linear matrix equation.

Collective.

:sources: `Source code at slepc4py/SLEPc/LME.pyx:526 <slepc4py/SLEPc/LME.pyx#L526>`

Return type

None

view(viewer=None)

Print the LME data structure.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/LME.pyx:58 <slepc4py/SLEPc/LME.pyx#L58>`

Attributes Documentation

bv

The basis vectors (BV) object associated to the LME object.

:sources: `Source code at slepc4py/SLEPc/LME.pyx:622 <slepc4py/SLEPc/LME.pyx#L622>`

fn

The math function (FN) object associated to the LME object.

:sources: `Source code at slepc4py/SLEPc/LME.pyx:615 <slepc4py/SLEPc/LME.pyx#L615>`

max_it

The maximum iteration count used by the LME convergence tests.

:sources: `Source code at slepc4py/SLEPc/LME.pyx:608 <slepc4py/SLEPc/LME.pyx#L608>`

tol

The tolerance value used by the LME convergence tests.

:sources: `Source code at slepc4py/SLEPc/LME.pyx:601 <slepc4py/SLEPc/LME.pyx#L601>`

slepc4py.SLEPc.MFN

class slepc4py.SLEPc.MFN

Bases: `Object`

MFN.

Enumerations

<i>ConvergedReason</i>	MFN convergence reasons.
<i>Type</i>	MFN type.

slepc4py.SLEPc.MFN.ConvergedReason

class slepc4py.SLEPc.MFN.ConvergedReason

Bases: `object`

MFN convergence reasons.

- ‘MFN_CONVERGED_TOL’: All eigenpairs converged to requested tolerance.
- ‘MFN_CONVERGED_ITS’: Solver completed the requested number of steps.
- ‘MFN_DIVERGED_ITS’: Maximum number of iterations exceeded.
- ‘MFN_DIVERGED_BREAKDOWN’: Generic breakdown in method.

Attributes Summary

<i>CONVERGED_ITERATING</i>	Constant CONVERGED_ITERATING of type <code>int</code>
<i>CONVERGED_ITS</i>	Constant CONVERGED_ITS of type <code>int</code>
<i>CONVERGED_TOL</i>	Constant CONVERGED_TOL of type <code>int</code>
<i>DIVERGED_BREAKDOWN</i>	Constant DIVERGED_BREAKDOWN of type <code>int</code>
<i>DIVERGED_ITS</i>	Constant DIVERGED_ITS of type <code>int</code>
<i>ITERATING</i>	Constant ITERATING of type <code>int</code>

Attributes Documentation

CONVERGED_ITERATING: `int` = **CONVERGED_ITERATING**

Constant CONVERGED_ITERATING of type `int`

CONVERGED_ITS: `int` = **CONVERGED_ITS**

Constant CONVERGED_ITS of type `int`

CONVERGED_TOL: `int` = **CONVERGED_TOL**

Constant CONVERGED_TOL of type `int`

DIVERGED_BREAKDOWN: `int` = **DIVERGED_BREAKDOWN**

Constant DIVERGED_BREAKDOWN of type `int`

DIVERGED_ITS: `int` = **DIVERGED_ITS**

Constant DIVERGED_ITS of type `int`

ITERATING: `int` = **ITERATING**

Constant ITERATING of type `int`

slepc4py.SLEPc.MFN.Type

class slepc4py.SLEPc.MFN.Type

Bases: `object`

MFN type.

Action of a matrix function on a vector.

- *KRYLOV*: Restarted Krylov solver.
- *EXPOKIT*: Implementation of the method in Expokit.

Attributes Summary

<i>EXPOKIT</i>	Object EXPOKIT of type <code>str</code>
<i>KRYLOV</i>	Object KRYLOV of type <code>str</code>

Attributes Documentation

EXPOKIT: `str` = **EXPOKIT**

Object EXPOKIT of type `str`

KRYLOV: `str` = **KRYLOV**

Object KRYLOV of type `str`

Methods Summary

<i>appendOptionsPrefix</i> ([prefix])	Append to the prefix used for searching for all MFN options in the database.
<i>cancelMonitor</i> ()	Clear all monitors for an <i>MFN</i> object.
<i>create</i> ([comm])	Create the MFN object.
<i>destroy</i> ()	Destroy the MFN object.
<i>getBV</i> ()	Get the basis vector object associated to the MFN object.
<i>getConvergedReason</i> ()	Get the reason why the <i>solve()</i> iteration was stopped.
<i>getDimensions</i> ()	Get the dimension of the subspace used by the solver.
<i>getErrorIfNotConverged</i> ()	Get if <i>solve()</i> generates an error if the solver does not converge.
<i>getFN</i> ()	Get the math function object associated to the MFN object.
<i>getIterationNumber</i> ()	Get the current iteration number.
<i>getMonitor</i> ()	Get the list of monitor functions.
<i>getOperator</i> ()	Get the matrix associated with the MFN object.
<i>getOptionsPrefix</i> ()	Get the prefix used for searching for all MFN options in the database.
<i>getTolerances</i> ()	Get the tolerance and maximum iteration count.
<i>getType</i> ()	Get the MFN type of this object.
<i>reset</i> ()	Reset the MFN object.
<i>setBV</i> (bv)	Set a basis vector object associated to the MFN object.

continues on next page

Table 54 – continued from previous page

<code>setDimensions(ncv)</code>	Set the dimension of the subspace to be used by the solver.
<code>setErrorIfNotConverged([flag])</code>	Set <code>solve()</code> to generate an error if the solver does not converge.
<code>setFN(fn)</code>	Set a math function object associated to the MFN object.
<code>setFromOptions()</code>	Set MFN options from the options database.
<code>setMonitor(monitor[, args, kargs])</code>	Append a monitor function to the list of monitors.
<code>setOperator(A)</code>	Set the matrix associated with the MFN object.
<code>setOptionsPrefix([prefix])</code>	Set the prefix used for searching for all MFN options in the database.
<code>setTolerances([tol, max_it])</code>	Set the tolerance and maximum iteration count.
<code>setType(mfn_type)</code>	Set the particular solver to be used in the MFN object.
<code>setUp()</code>	Set up all the necessary internal data structures.
<code>solve(b, x)</code>	Solve the matrix function problem.
<code>solveTranspose(b, x)</code>	Solve the transpose matrix function problem.
<code>view([viewer])</code>	Print the MFN data structure.

Attributes Summary

<code>bv</code>	The basis vectors (BV) object associated to the MFN object.
<code>fn</code>	The math function (FN) object associated to the MFN object.
<code>max_it</code>	The maximum iteration count used by the MFN convergence tests.
<code>tol</code>	The tolerance count used by the MFN convergence tests.

Methods Documentation

`appendOptionsPrefix(prefix=None)`

Append to the prefix used for searching for all MFN options in the database.

Logically collective.

Parameters

prefix (`str` / `None`) – The prefix string to prepend to all MFN option requests.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:154 <slepc4py/SLEPc/MFN.pyx#L154>`

`cancelMonitor()`

Clear all monitors for an `MFN` object.

Logically collective.

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:367 <slepc4py/SLEPc/MFN.pyx#L367>`

Return type

`None`

create(*comm=None*)

Create the MFN object.

Collective.

Parameters

comm (*Comm* / *None*) – MPI communicator. If not provided, it defaults to all processes.

Return type

Self

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:77 <slepc4py/SLEPc/MFN.pyx#L77>`

destroy()

Destroy the MFN object.

Logically collective.

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:59 <slepc4py/SLEPc/MFN.pyx#L59>`

Return type

Self

getBV()

Get the basis vector object associated to the MFN object.

Not collective.

Returns

The basis vectors context.

Return type

BV

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:282 <slepc4py/SLEPc/MFN.pyx#L282>`

getConvergedReason()

Get the reason why the *solve*() iteration was stopped.

Not collective.

Returns

Negative value indicates diverged, positive value converged.

Return type

ConvergedReason

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:444 <slepc4py/SLEPc/MFN.pyx#L444>`

getDimensions()

Get the dimension of the subspace used by the solver.

Not collective.

Returns

Maximum dimension of the subspace to be used by the solver.

Return type

int

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:224 <slepc4py/SLEPc/MFN.pyx#L224>`

getErrorIfNotConverged()

Get if `solve()` generates an error if the solver does not converge.

Not collective.

Get a flag indicating whether `solve()` will generate an error if the solver does not converge.

Returns

True indicates you want the error generated.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:473 <slepc4py/SLEPc/MFN.pyx#L473>`

getFN()

Get the math function object associated to the MFN object.

Not collective.

Returns

The math function context.

Return type

`FN`

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:253 <slepc4py/SLEPc/MFN.pyx#L253>`

getIterationNumber()

Get the current iteration number.

Not collective.

Get the current iteration number. If the call to `solve()` is complete, then it returns the number of iterations carried out by the solution method.

Returns

Iteration number.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:425 <slepc4py/SLEPc/MFN.pyx#L425>`

getMonitor()

Get the list of monitor functions.

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:363 <slepc4py/SLEPc/MFN.pyx#L363>`

Return type

`MFNMonitorFunction`

getOperator()

Get the matrix associated with the MFN object.

Collective.

Returns

The matrix for which the matrix function is to be computed.

Return type

`petsc4py.PETSc.Mat`

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:311 <slepc4py/SLEPc/MFN.pyx#L311>`

getOptionsPrefix()

Get the prefix used for searching for all MFN options in the database.

Not collective.

Returns

The prefix string set for this MFN object.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:124 <slepc4py/SLEPc/MFN.pyx#L124>`

getTolerances()

Get the tolerance and maximum iteration count.

Not collective.

Get the tolerance and maximum iteration count used by the default MFN convergence tests.

Returns

- **tol** (`float`) – The convergence tolerance.
- **max_it** (`int`) – The maximum number of iterations

Return type

`tuple[float, int]`

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:181 <slepc4py/SLEPc/MFN.pyx#L181>`

getType()

Get the MFN type of this object.

Not collective.

Returns

The solver currently being used.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:109 <slepc4py/SLEPc/MFN.pyx#L109>`

reset()

Reset the MFN object.

Collective.

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:69 <slepc4py/SLEPc/MFN.pyx#L69>`

Return type

`None`

setBV(bv)

Set a basis vector object associated to the MFN object.

Collective.

Parameters

bv (`BV`) – The basis vectors context.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:298 <slepc4py/SLEPc/MFN.pyx#L298>`

setDimensions(*ncv*)

Set the dimension of the subspace to be used by the solver.

Logically collective.

Parameters

ncv (*int*) – Maximum dimension of the subspace to be used by the solver.

Return type

None

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:239 <slepc4py/SLEPc/MFN.pyx#L239>`

setErrorIfNotConverged(*flg=True*)

Set *solve()* to generate an error if the solver does not converge.

Logically collective.

Parameters

flg (*bool*) – True indicates you want the error generated.

Return type

None

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:459 <slepc4py/SLEPc/MFN.pyx#L459>`

setFN(*fn*)

Set a math function object associated to the MFN object.

Collective.

Parameters

fn (*FN*) – The math function context.

Return type

None

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:269 <slepc4py/SLEPc/MFN.pyx#L269>`

setFromOptions()

Set MFN options from the options database.

Collective.

Set MFN options from the options database. This routine must be called before *setUp()* if the user is to be allowed to set the solver type.

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:169 <slepc4py/SLEPc/MFN.pyx#L169>`

Return type

None

setMonitor(*monitor, args=None, kargs=None*)

Append a monitor function to the list of monitors.

Logically collective.

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:342 <slepc4py/SLEPc/MFN.pyx#L342>`

Parameters

- **monitor** (*MFNMonitorFunction* | *None*)
- **args** (*tuple*[*Any*, ...] | *None*)
- **kargs** (*dict*[*str*, *Any*] | *None*)

Return type

None

setOperator(A)

Set the matrix associated with the MFN object.

Collective.

Parameters

A (*Mat*) – The problem matrix.

Return type

None

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:327 <slepc4py/SLEPc/MFN.pyx#L327>`

setOptionsPrefix(prefix=None)

Set the prefix used for searching for all MFN options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all MFN option requests.

Return type

None

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:139 <slepc4py/SLEPc/MFN.pyx#L139>`

setTolerances(tol=None, max_it=None)

Set the tolerance and maximum iteration count.

Logically collective.

Set the tolerance and maximum iteration count used by the default MFN convergence tests.

Parameters

- **tol** (*float* / *None*) – The convergence tolerance.
- **max_it** (*int* / *None*) – The maximum number of iterations

Return type

None

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:202 <slepc4py/SLEPc/MFN.pyx#L202>`

setType(mfn_type)

Set the particular solver to be used in the MFN object.

Logically collective.

Parameters

mfn_type (*Type* / *str*) – The solver to be used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:94 <slepc4py/SLEPc/MFN.pyx#L94>`

setUp()

Set up all the necessary internal data structures.

Collective.

Set up all the internal data structures necessary for the execution of the eigensolver.

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:378 <slepc4py/SLEPc/MFN.pyx#L378>`

Return type

None

solve(*b*, *x*)

Solve the matrix function problem.

Collective.

Given a vector *b*, the vector $x = f(A)b$ is returned.

Parameters

- **b** (*Vec*) – The right hand side vector.
- **x** (*Vec*) – The solution.

Return type

None

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:389 <slepc4py/SLEPc/MFN.pyx#L389>`

solveTranspose(*b*, *x*)

Solve the transpose matrix function problem.

Collective.

Given a vector *b*, the vector $x = f(A^T)b$ is returned.

Parameters

- **b** (*Vec*) – The right hand side vector.
- **x** (*Vec*) – The solution.

Return type

None

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:407 <slepc4py/SLEPc/MFN.pyx#L407>`

view(*viewer=*None)

Print the MFN data structure.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:44 <slepc4py/SLEPc/MFN.pyx#L44>`

Attributes Documentation

bv

The basis vectors (BV) object associated to the MFN object.

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:514 <slepc4py/SLEPc/MFN.pyx#L514>`

fn

The math function (FN) object associated to the MFN object.

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:507 <slepc4py/SLEPc/MFN.pyx#L507>`

max_it

The maximum iteration count used by the MFN convergence tests.

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:500 <slepc4py/SLEPc/MFN.pyx#L500>`

tol

The tolerance count used by the MFN convergence tests.

:sources: `Source code at slepc4py/SLEPc/MFN.pyx:493 <slepc4py/SLEPc/MFN.pyx#L493>`

slepc4py.SLEPc.NEP

class slepc4py.SLEPc.NEP

Bases: `Object`

NEP.

Enumerations

<i>CISSExtraction</i>	NEP CISS extraction technique.
<i>Conv</i>	NEP convergence test.
<i>ConvergedReason</i>	NEP convergence reasons.
<i>ErrorType</i>	NEP error type to assess accuracy of computed solutions.
<i>ProblemType</i>	NEP problem type.
<i>Refine</i>	NEP refinement strategy.
<i>RefineScheme</i>	NEP scheme for solving linear systems during iterative refinement.
<i>Stop</i>	NEP stopping test.
<i>Type</i>	NEP type.
<i>Which</i>	NEP desired part of spectrum.

slepc4py.SLEPc.NEP.CISSExtraction

class slepc4py.SLEPc.NEP.CISSExtraction

Bases: `object`

NEP CISS extraction technique.

- *RITZ*: Ritz extraction.
- *HANKEL*: Extraction via Hankel eigenproblem.
- *CAA*: Communication-avoiding Arnoldi.

Attributes Summary

<i>CAA</i>	Constant CAA of type <code>int</code>
<i>HANKEL</i>	Constant HANKEL of type <code>int</code>

continues on next page

Table 57 – continued from previous page

<i>RITZ</i>	Constant RITZ of type <code>int</code>
-------------	--

Attributes Documentation

CAA: `int` = CAA

Constant CAA of type `int`

HANKEL: `int` = HANKEL

Constant HANKEL of type `int`

RITZ: `int` = RITZ

Constant RITZ of type `int`

`slepc4py.SLEPc.NEP.Conv`

class `slepc4py.SLEPc.NEP.Conv`

Bases: `object`

NEP convergence test.

- *ABS*: Absolute convergence test.
- *REL*: Convergence test relative to the eigenvalue.
- *NORM*: Convergence test relative to the matrix norms.
- *USER*: User-defined convergence test.

Attributes Summary

<i>ABS</i>	Constant ABS of type <code>int</code>
<i>NORM</i>	Constant NORM of type <code>int</code>
<i>REL</i>	Constant REL of type <code>int</code>
<i>USER</i>	Constant USER of type <code>int</code>

Attributes Documentation

ABS: `int` = ABS

Constant ABS of type `int`

NORM: `int` = NORM

Constant NORM of type `int`

REL: `int` = REL

Constant REL of type `int`

USER: `int` = USER

Constant USER of type `int`

`slepc4py.SLEPc.NEP.ConvergedReason`

class `slepc4py.SLEPc.NEP.ConvergedReason`

Bases: `object`

NEP convergence reasons.

- **CONVERGED_TOL**: All eigenpairs converged to requested tolerance.
- **CONVERGED_USER**: User-defined convergence criterion satisfied.
- **DIVERGED_ITS**: Maximum number of iterations exceeded.
- **DIVERGED_BREAKDOWN**: Solver failed due to breakdown.
- **DIVERGED_LINEAR_SOLVE**: Inner linear solve failed.
- **DIVERGED_SUBSPACE_EXHAUSTED**: Run out of space for the basis in an unrestarted solver.
- **CONVERGED_ITERATING**: Iteration not finished yet.

Attributes Summary

<i>CONVERGED_ITERATING</i>	Constant CONVERGED_ITERATING of type <code>int</code>
<i>CONVERGED_TOL</i>	Constant CONVERGED_TOL of type <code>int</code>
<i>CONVERGED_USER</i>	Constant CONVERGED_USER of type <code>int</code>
<i>DIVERGED_BREAKDOWN</i>	Constant DIVERGED_BREAKDOWN of type <code>int</code>
<i>DIVERGED_ITS</i>	Constant DIVERGED_ITS of type <code>int</code>
<i>DIVERGED_LINEAR_SOLVE</i>	Constant DIVERGED_LINEAR_SOLVE of type <code>int</code>
<i>DIVERGED_SUBSPACE_EXHAUSTED</i>	Constant DIVERGED_SUBSPACE_EXHAUSTED of type <code>int</code>
<i>ITERATING</i>	Constant ITERATING of type <code>int</code>

Attributes Documentation

CONVERGED_ITERATING: `int` = CONVERGED_ITERATING

Constant CONVERGED_ITERATING of type `int`

CONVERGED_TOL: `int` = CONVERGED_TOL

Constant CONVERGED_TOL of type `int`

CONVERGED_USER: `int` = CONVERGED_USER

Constant CONVERGED_USER of type `int`

DIVERGED_BREAKDOWN: `int` = DIVERGED_BREAKDOWN

Constant DIVERGED_BREAKDOWN of type `int`

DIVERGED_ITS: `int` = DIVERGED_ITS

Constant DIVERGED_ITS of type `int`

DIVERGED_LINEAR_SOLVE: `int` = DIVERGED_LINEAR_SOLVE

Constant DIVERGED_LINEAR_SOLVE of type `int`

DIVERGED_SUBSPACE_EXHAUSTED: `int` = DIVERGED_SUBSPACE_EXHAUSTED

Constant DIVERGED_SUBSPACE_EXHAUSTED of type `int`

ITERATING: `int` = ITERATING

Constant ITERATING of type `int`

slepc4py.SLEPc.NEP.ErrorType

class slepc4py.SLEPc.NEP.ErrorType

Bases: `object`

NEP error type to assess accuracy of computed solutions.

- *ABSOLUTE*: Absolute error.
- *RELATIVE*: Relative error.
- *BACKWARD*: Backward error.

Attributes Summary

<i>ABSOLUTE</i>	Constant ABSOLUTE of type <code>int</code>
<i>BACKWARD</i>	Constant BACKWARD of type <code>int</code>
<i>RELATIVE</i>	Constant RELATIVE of type <code>int</code>

Attributes Documentation

ABSOLUTE: `int` = ABSOLUTE

Constant ABSOLUTE of type `int`

BACKWARD: `int` = BACKWARD

Constant BACKWARD of type `int`

RELATIVE: `int` = RELATIVE

Constant RELATIVE of type `int`

slepc4py.SLEPc.NEP.ProblemType

class slepc4py.SLEPc.NEP.ProblemType

Bases: `object`

NEP problem type.

- *GENERAL*: General nonlinear eigenproblem.
- *RATIONAL*: NEP defined in split form with all f_i rational.

Attributes Summary

<i>GENERAL</i>	Constant GENERAL of type <code>int</code>
<i>RATIONAL</i>	Constant RATIONAL of type <code>int</code>

Attributes Documentation

GENERAL: `int` = GENERAL

Constant GENERAL of type `int`

RATIONAL: `int` = RATIONAL

Constant RATIONAL of type `int`

slepc4py.SLEPc.NEP.Refine

class slepc4py.SLEPc.NEP.Refine

Bases: `object`

NEP refinement strategy.

- *NONE*: No refinement.
- *SIMPLE*: Refine eigenpairs one by one.
- *MULTIPLE*: Refine all eigenpairs simultaneously (invariant pair).

Attributes Summary

<i>MULTIPLE</i>	Constant <i>MULTIPLE</i> of type <code>int</code>
<i>NONE</i>	Constant <i>NONE</i> of type <code>int</code>
<i>SIMPLE</i>	Constant <i>SIMPLE</i> of type <code>int</code>

Attributes Documentation

MULTIPLE: `int` = *MULTIPLE*

Constant *MULTIPLE* of type `int`

NONE: `int` = *NONE*

Constant *NONE* of type `int`

SIMPLE: `int` = *SIMPLE*

Constant *SIMPLE* of type `int`

slepc4py.SLEPc.NEP.RefineScheme

class slepc4py.SLEPc.NEP.RefineScheme

Bases: `object`

NEP scheme for solving linear systems during iterative refinement.

- *SCHUR*: Schur complement.
- *MBE*: Mixed block elimination.
- *EXPLICIT*: Build the explicit matrix.

Attributes Summary

<i>EXPLICIT</i>	Constant <i>EXPLICIT</i> of type <code>int</code>
<i>MBE</i>	Constant <i>MBE</i> of type <code>int</code>
<i>SCHUR</i>	Constant <i>SCHUR</i> of type <code>int</code>

Attributes Documentation

EXPLICIT: `int` = *EXPLICIT*

Constant *EXPLICIT* of type `int`

MBE: `int` = *MBE*

Constant *MBE* of type `int`

SCHUR: `int` = **SCHUR**
Constant SCHUR of type `int`

slepc4py.SLEPc.NEP.Stop

class `slepc4py.SLEPc.NEP.Stop`
Bases: `object`
NEP stopping test.

- *BASIC*: Default stopping test.
- *USER*: User-defined stopping test.

Attributes Summary

<i>BASIC</i>	Constant BASIC of type <code>int</code>
<i>USER</i>	Constant USER of type <code>int</code>

Attributes Documentation

BASIC: `int` = **BASIC**
Constant BASIC of type `int`

USER: `int` = **USER**
Constant USER of type `int`

slepc4py.SLEPc.NEP.Type

class `slepc4py.SLEPc.NEP.Type`
Bases: `object`
NEP type.
Nonlinear eigensolvers.

- *RII*: Residual inverse iteration.
- *SLP*: Successive linear problems.
- *NARNOLDI*: Nonlinear Arnoldi.
- *CISS*: Contour integral spectrum slice.
- *INTERPOL*: Polynomial interpolation.
- *NLEIGS*: Fully rational Krylov method for nonlinear eigenproblems.

Attributes Summary

<i>CISS</i>	Object CISS of type <code>str</code>
<i>INTERPOL</i>	Object INTERPOL of type <code>str</code>
<i>NARNOLDI</i>	Object NARNOLDI of type <code>str</code>
<i>NLEIGS</i>	Object NLEIGS of type <code>str</code>
<i>RII</i>	Object RII of type <code>str</code>
<i>SLP</i>	Object SLP of type <code>str</code>

Attributes Documentation

CISS: `str` = CISS

Object CISS of type `str`

INTERPOL: `str` = INTERPOL

Object INTERPOL of type `str`

NARNOLDI: `str` = NARNOLDI

Object NARNOLDI of type `str`

NLEIGS: `str` = NLEIGS

Object NLEIGS of type `str`

RII: `str` = RII

Object RII of type `str`

SLP: `str` = SLP

Object SLP of type `str`

`slepc4py.SLEPc.NEP.Which`

class `slepc4py.SLEPc.NEP.Which`

Bases: `object`

NEP desired part of spectrum.

- *LARGEST_MAGNITUDE*: Largest magnitude (default).
- *SMALLEST_MAGNITUDE*: Smallest magnitude.
- *LARGEST_REAL*: Largest real parts.
- *SMALLEST_REAL*: Smallest real parts.
- *LARGEST_IMAGINARY*: Largest imaginary parts in magnitude.
- *SMALLEST_IMAGINARY*: Smallest imaginary parts in magnitude.
- *TARGET_MAGNITUDE*: Closest to target (in magnitude).
- *TARGET_REAL*: Real part closest to target.
- *TARGET_IMAGINARY*: Imaginary part closest to target.
- *ALL*: All eigenvalues in a region.
- *USER*: User defined selection.

Attributes Summary

<i>ALL</i>	Constant ALL of type <code>int</code>
<i>LARGEST_IMAGINARY</i>	Constant LARGEST_IMAGINARY of type <code>int</code>
<i>LARGEST_MAGNITUDE</i>	Constant LARGEST_MAGNITUDE of type <code>int</code>
<i>LARGEST_REAL</i>	Constant LARGEST_REAL of type <code>int</code>
<i>SMALLEST_IMAGINARY</i>	Constant SMALLEST_IMAGINARY of type <code>int</code>
<i>SMALLEST_MAGNITUDE</i>	Constant SMALLEST_MAGNITUDE of type <code>int</code>
<i>SMALLEST_REAL</i>	Constant SMALLEST_REAL of type <code>int</code>
<i>TARGET_IMAGINARY</i>	Constant TARGET_IMAGINARY of type <code>int</code>

continues on next page

Table 66 – continued from previous page

<i>TARGET_MAGNITUDE</i>	Constant TARGET_MAGNITUDE of type <code>int</code>
<i>TARGET_REAL</i>	Constant TARGET_REAL of type <code>int</code>
<i>USER</i>	Constant USER of type <code>int</code>

Attributes Documentation

ALL: `int` = ALL

Constant ALL of type `int`

LARGEST_IMAGINARY: `int` = LARGEST_IMAGINARY

Constant LARGEST_IMAGINARY of type `int`

LARGEST_MAGNITUDE: `int` = LARGEST_MAGNITUDE

Constant LARGEST_MAGNITUDE of type `int`

LARGEST_REAL: `int` = LARGEST_REAL

Constant LARGEST_REAL of type `int`

SMALLEST_IMAGINARY: `int` = SMALLEST_IMAGINARY

Constant SMALLEST_IMAGINARY of type `int`

SMALLEST_MAGNITUDE: `int` = SMALLEST_MAGNITUDE

Constant SMALLEST_MAGNITUDE of type `int`

SMALLEST_REAL: `int` = SMALLEST_REAL

Constant SMALLEST_REAL of type `int`

TARGET_IMAGINARY: `int` = TARGET_IMAGINARY

Constant TARGET_IMAGINARY of type `int`

TARGET_MAGNITUDE: `int` = TARGET_MAGNITUDE

Constant TARGET_MAGNITUDE of type `int`

TARGET_REAL: `int` = TARGET_REAL

Constant TARGET_REAL of type `int`

USER: `int` = USER

Constant USER of type `int`

Methods Summary

<i>appendOptionsPrefix</i> ([prefix])	Append to the prefix used for searching for all NEP options in the database.
<i>applyResolvent</i> (omega, v, r[, rg])	Apply the resolvent $T^{-1}(z)$ to a given vector.
<i>cancelMonitor</i> ()	Clear all monitors for a <i>NEP</i> object.
<i>computeError</i> (i[, etype])	Compute the error associated with the i-th computed eigenpair.
<i>create</i> ([comm])	Create the NEP object.
<i>destroy</i> ()	Destroy the NEP object.
<i>errorView</i> ([etype, viewer])	Display the errors associated with the computed solution.
<i>getBV</i> ()	Get the basis vectors object associated to the eigen-solver.

continues on next page

Table 67 – continued from previous page

<code>getCISSExtraction()</code>	Get the extraction technique used in the CISS solver.
<code>getCISSKSPs()</code>	Get the list of linear solver objects associated with the CISS solver.
<code>getCISSRefinement()</code>	Get the values of various refinement parameters in the CISS solver.
<code>getCISSSizes()</code>	Get the values of various size parameters in the CISS solver.
<code>getCISSThreshold()</code>	Get the values of various threshold parameters in the CISS solver.
<code>getConverged()</code>	Get the number of converged eigenpairs.
<code>getConvergedReason()</code>	Get the reason why the <code>solve()</code> iteration was stopped.
<code>getConvergenceTest()</code>	Get the method used to compute the error estimate used in the convergence test.
<code>getDS()</code>	Get the direct solver associated to the eigensolver.
<code>getDimensions()</code>	Get the number of eigenvalues to compute.
<code>getEigenpair(i[, Vr, Vi])</code>	Get the i-th solution of the eigenproblem as computed by <code>solve()</code> .
<code>getErrorEstimate(i)</code>	Get the error estimate associated to the i-th computed eigenpair.
<code>getFunction()</code>	Get the function to compute the nonlinear Function $T(\lambda)$.
<code>getInterpolInterpolation()</code>	Get the tolerance and maximum degree for the interpolation polynomial.
<code>getInterpolPEP()</code>	Get the associated polynomial eigensolver object.
<code>getIterationNumber()</code>	Get the current iteration number.
<code>getJacobian()</code>	Get the function to compute the Jacobian $T'(\lambda)$ and J .
<code>getLeftEigenvector(i, Wr[, Wi])</code>	Get the i-th left eigenvector as computed by <code>solve()</code> .
<code>getMonitor()</code>	Get the list of monitor functions.
<code>getNArnoldiKSP()</code>	Get the linear solver object associated with the non-linear eigensolver.
<code>getNArnoldiLagPreconditioner()</code>	Get how often the preconditioner is rebuilt.
<code>getNLEIGSEPS()</code>	Get the linear eigensolver object associated with the nonlinear eigensolver.
<code>getNLEIGSFullBasis()</code>	Get the flag that indicates if NLEIGS is using the full-basis variant.
<code>getNLEIGSInterpolation()</code>	Get the tolerance and maximum degree for the interpolation polynomial.
<code>getNLEIGSKSPs()</code>	Get the list of linear solver objects associated with the NLEIGS solver.
<code>getNLEIGSLocking()</code>	Get the locking flag used in the NLEIGS method.
<code>getNLEIGSRKShifts()</code>	Get the list of shifts used in the Rational Krylov method.
<code>getNLEIGSRestart()</code>	Get the restart parameter used in the NLEIGS method.
<code>getOptionsPrefix()</code>	Get the prefix used for searching for all NEP options in the database.
<code>getProblemType()</code>	Get the problem type from the <code>NEP</code> object.
<code>getRG()</code>	Get the region object associated to the eigensolver.
<code>getRIIConstCorrectionTol()</code>	Get the constant tolerance flag.
<code>getRIIDeflationThreshold()</code>	Get the threshold value that controls deflation.

continues on next page

Table 67 – continued from previous page

<i>getRIIHermitian()</i>	Get if the Hermitian version must be used by the solver.
<i>getRIIKSP()</i>	Get the linear solver object associated with the non-linear eigensolver.
<i>getRIILagPreconditioner()</i>	Get how often the preconditioner is rebuilt.
<i>getRIIMaximumIterations()</i>	Get the maximum number of inner iterations of RIL.
<i>getRefine()</i>	Get the refinement strategy used by the NEP object.
<i>getRefineKSP()</i>	Get the KSP object used by the eigensolver in the refinement phase.
<i>getSLPDeflationThreshold()</i>	Get the threshold value that controls deflation.
<i>getSLPEPS()</i>	Get the linear eigensolver object associated with the nonlinear eigensolver.
<i>getSLPEPSLeft()</i>	Get the left eigensolver.
<i>getSLPKSP()</i>	Get the linear solver object associated with the non-linear eigensolver.
<i>getSplitOperator()</i>	Get the operator of the nonlinear eigenvalue problem in split form.
<i>getSplitPreconditioner()</i>	Get the operator of the split preconditioner.
<i>getStoppingTest()</i>	Get the stopping function.
<i>getTarget()</i>	Get the value of the target.
<i>getTolerances()</i>	Get the tolerance and maximum iteration count.
<i>getTrackAll()</i>	Get the flag indicating whether all residual norms must be computed.
<i>getTwoSided()</i>	Get the flag indicating if a two-sided variant is being used.
<i>getType()</i>	Get the NEP type of this object.
<i>getWhichEigenpairs()</i>	Get which portion of the spectrum is to be sought.
<i>reset()</i>	Reset the NEP object.
<i>setBV(bv)</i>	Set the basis vectors object associated to the eigensolver.
<i>setCISSExtraction(extraction)</i>	Set the extraction technique used in the CISS solver.
<i>setCISSRefinement([inner, blsize])</i>	Set the values of various refinement parameters in the CISS solver.
<i>setCISSSizes([ip, bs, ms, npart, bsmax, ...])</i>	Set the values of various size parameters in the CISS solver.
<i>setCISSThreshold([delta, spur])</i>	Set the values of various threshold parameters in the CISS solver.
<i>setConvergenceTest(conv)</i>	Set how to compute the error estimate used in the convergence test.
<i>setDS(ds)</i>	Set a direct solver object associated to the eigensolver.
<i>setDimensions([nev, ncv, mpd])</i>	Set the number of eigenvalues to compute.
<i>setFromOptions()</i>	Set NEP options from the options database.
<i>setFunction(function[, F, P, args, kargs])</i>	Set the function to compute the nonlinear Function $T(\lambda)$.
<i>setInitialSpace(space)</i>	Set the initial space from which the eigensolver starts to iterate.
<i>setInterpolInterpolation([tol, deg])</i>	Set the tolerance and maximum degree for the interpolation polynomial.
<i>setInterpolPEP(pep)</i>	Set a polynomial eigensolver object associated to the nonlinear eigensolver.
<i>setJacobian(jacobian[, J, args, kargs])</i>	Set the function to compute the Jacobian $T'(\lambda)$.
<i>setMonitor(monitor[, args, kargs])</i>	Append a monitor function to the list of monitors.

continues on next page

Table 67 – continued from previous page

<code>setArnoldiKSP(ksp)</code>	Set a linear solver object associated to the nonlinear eigensolver.
<code>setArnoldiLagPreconditioner(lag)</code>	Set when the preconditioner is rebuilt in the nonlinear solve.
<code>setNLEIGSEPS(eps)</code>	Set a linear eigensolver object associated to the non-linear eigensolver.
<code>setNLEIGSFullBasis([fullbasis])</code>	Set TOAR-basis (default) or full-basis variants of the NLEIGS method.
<code>setNLEIGSInterpolation([tol, deg])</code>	Set the tolerance and maximum degree for the interpolation polynomial.
<code>setNLEIGSLocking(lock)</code>	Toggle between locking and non-locking variants of the NLEIGS method.
<code>setNLEIGSRKShifts(shifts)</code>	Set a list of shifts to be used in the Rational Krylov method.
<code>setNLEIGSRestart(keep)</code>	Set the restart parameter for the NLEIGS method.
<code>setOptionsPrefix([prefix])</code>	Set the prefix used for searching for all NEP options in the database.
<code>setProblemType(problem_type)</code>	Set the type of the eigenvalue problem.
<code>setRG(rg)</code>	Set a region object associated to the eigensolver.
<code>setRIIConstCorrectionTol(cct)</code>	Set a flag to keep the tolerance used in the linear solver constant.
<code>setRIIDeflationThreshold(deftol)</code>	Set the threshold used to switch between deflated and non-deflated.
<code>setRIIHermitian(herm)</code>	Set a flag to use the Hermitian version of the solver.
<code>setRIIKSP(ksp)</code>	Set a linear solver object associated to the nonlinear eigensolver.
<code>setRIILagPreconditioner(lag)</code>	Set when the preconditioner is rebuilt in the nonlinear solve.
<code>setRIIMaximumIterations(its)</code>	Set the max.
<code>setRefine(ref[, npart, tol, its, scheme])</code>	Set the refinement strategy used by the NEP object.
<code>setSLPDeflationThreshold(deftol)</code>	Set the threshold used to switch between deflated and non-deflated.
<code>setSLPEPS(eps)</code>	Set a linear eigensolver object associated to the non-linear eigensolver.
<code>setSLPEPSLeft(eps)</code>	Set a linear eigensolver object associated to the non-linear eigensolver.
<code>setSLPKSP(ksp)</code>	Set a linear solver object associated to the nonlinear eigensolver.
<code>setSplitOperator(A, f[, structure])</code>	Set the operator of the nonlinear eigenvalue problem in split form.
<code>setSplitPreconditioner(P[, structure])</code>	Set the operator in split form.
<code>setStoppingTest(stopping[, args, kargs])</code>	Set a function to decide when to stop the outer iteration of the eigensolver.
<code>setTarget(target)</code>	Set the value of the target.
<code>setTolerances([tol, maxit])</code>	Set the tolerance and max.
<code>setTrackAll(trackall)</code>	Set if the solver must compute the residual of all approximate eigenpairs.
<code>setTwoSided(twosided)</code>	Set the solver to use a two-sided variant.
<code>setType(nep_type)</code>	Set the particular solver to be used in the NEP object.
<code>setUp()</code>	Set up all the necessary internal data structures.
<code>setWhichEigenpairs(which)</code>	Set which portion of the spectrum is to be sought.
<code>solve()</code>	Solve the eigensystem.

continues on next page

Table 67 – continued from previous page

<code>valuesView([viewer])</code>	Display the computed eigenvalues in a viewer.
<code>vectorsView([viewer])</code>	Output computed eigenvectors to a viewer.
<code>view([viewer])</code>	Print the NEP data structure.

Attributes Summary

<code>bv</code>	The basis vectors (BV) object associated.
<code>ds</code>	The direct solver (DS) object associated.
<code>max_it</code>	The maximum iteration count used by the NEP convergence tests.
<code>problem_type</code>	The problem type from the NEP object.
<code>rg</code>	The region (RG) object associated.
<code>target</code>	The value of the target.
<code>tol</code>	The tolerance used by the NEP convergence tests.
<code>track_all</code>	Compute the residual of all approximate eigenpairs.
<code>which</code>	The portion of the spectrum to be sought.

Methods Documentation

`appendOptionsPrefix(prefix=None)`

Append to the prefix used for searching for all NEP options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all NEP option requests.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:289 <slepc4py/SLEPc/NEP.pyx#L289>`

`applyResolvent(omega, v, r, rg=None)`

Apply the resolvent $T^{-1}(z)$ to a given vector.

Collective.

Parameters

- **omega** (*Scalar*) – Value where the resolvent must be evaluated.
- **v** (*Vec*) – Input vector.
- **r** (*Vec*) – Placeholder for the result vector.
- **rg** (*RG* / *None*) – Region.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1339 <slepc4py/SLEPc/NEP.pyx#L1339>`

`cancelMonitor()`

Clear all monitors for a *NEP* object.

Logically collective.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:821 <slepc4py/SLEPc/NEP.pyx#L821>`

Return type

`None`

computeError(*i*, *etype*=`None`)

Compute the error associated with the *i*-th computed eigenpair.

Collective.

Compute the error (based on the residual norm) associated with the *i*-th computed eigenpair.

Parameters

- **i** (`int`) – Index of the solution to be considered.
- **etype** (`ErrorType` / `None`) – The error type to compute.

Returns

The error bound, computed in various ways from the residual norm $\|T(\lambda)x\|_2$ where λ is the eigenvalue and x is the eigenvector.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:978 <slepc4py/SLEPc/NEP.pyx#L978>`

create(*comm*=`None`)

Create the NEP object.

Collective.

Parameters

comm (`Comm` / `None`) – MPI communicator. If not provided, it defaults to all processes.

Return type

`Self`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:212 <slepc4py/SLEPc/NEP.pyx#L212>`

destroy()

Destroy the NEP object.

Collective.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:194 <slepc4py/SLEPc/NEP.pyx#L194>`

Return type

`Self`

errorView(*etype*=`None`, *viewer*=`None`)

Display the errors associated with the computed solution.

Collective.

Display the eigenvalues and the errors associated with the computed solution

Parameters

- **etype** (`ErrorType` / `None`) – The error type to compute.
- **viewer** (`petsc4py.PETSc.Viewer` / `None`) – Visualization context; if not provided, the standard output is used.

Return type

`None`

Notes

By default, this function checks the error of all eigenpairs and prints the eigenvalues if all of them are below the requested tolerance. If the viewer has format ASCII_INFO_DETAIL then a table with eigenvalues and corresponding errors is printed.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1007 <slepc4py/SLEPc/NEP.pyx#L1007>`

getBV()

Get the basis vectors object associated to the eigensolver.

Not collective.

Returns

The basis vectors context.

Return type

BV

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:653 <slepc4py/SLEPc/NEP.pyx#L653>`

getCISSExtraction()

Get the extraction technique used in the CISS solver.

Not collective.

Returns

The extraction technique.

Return type

CISSExtraction

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2068 <slepc4py/SLEPc/NEP.pyx#L2068>`

getCISSKSPs()

Get the list of linear solver objects associated with the CISS solver.

Collective.

Returns

The linear solver objects.

Return type

list of petsc4py.PETSc.KSP

Notes

The number of `petsc4py.PETSc.KSP` solvers is equal to the number of integration points divided by the number of partitions. This value is halved in the case of real matrices with a region centered at the real axis.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2237 <slepc4py/SLEPc/NEP.pyx#L2237>`

getCISSRefinement()

Get the values of various refinement parameters in the CISS solver.

Not collective.

Returns

- **inner** (*int*) – Number of iterative refinement iterations (inner loop).
- **blsize** (*int*) – Number of iterative refinement iterations (blocksize loop).

Return type`tuple[int, int]`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2219 <slepc4py/SLEPc/NEP.pyx#L2219>`

getCISSSizes()

Get the values of various size parameters in the CISS solver.

Not collective.

Returns

- **ip** (`int`) – Number of integration points.
- **bs** (`int`) – Block size.
- **ms** (`int`) – Moment size.
- **npart** (`int`) – Number of partitions when splitting the communicator.
- **bsmax** (`int`) – Maximum block size.
- **realmats** (`bool`) – True if A and B are real.

Return type`tuple[int, int, int, int, bool]`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2133 <slepc4py/SLEPc/NEP.pyx#L2133>`

getCISSThreshold()

Get the values of various threshold parameters in the CISS solver.

Not collective.

Returns

- **delta** (`float`) – Threshold for numerical rank.
- **spur** (`float`) – Spurious threshold (to discard spurious eigenpairs).

Return type`tuple[float, float]`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2182 <slepc4py/SLEPc/NEP.pyx#L2182>`

getConverged()

Get the number of converged eigenpairs.

Not collective.

Returns

Number of converged eigenpairs.

Return type`int`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:885 <slepc4py/SLEPc/NEP.pyx#L885>`

getConvergedReason()

Get the reason why the `solve()` iteration was stopped.

Not collective.

Returns

Negative value indicates diverged, positive value converged.

Return type*ConvergedReason***:sources:** `Source code at slepc4py/SLEPc/NEP.pyx:869 <slepc4py/SLEPc/NEP.pyx#L869>`**getConvergenceTest()**

Get the method used to compute the error estimate used in the convergence test.

Not collective.

Returns

The method used to compute the error estimate used in the convergence test.

Return type*Conv***:sources:** `Source code at slepc4py/SLEPc/NEP.pyx:452 <slepc4py/SLEPc/NEP.pyx#L452>`**getDS()**

Get the direct solver associated to the eigensolver.

Not collective.

Returns

The direct solver context.

Return type*DS***:sources:** `Source code at slepc4py/SLEPc/NEP.pyx:711 <slepc4py/SLEPc/NEP.pyx#L711>`**getDimensions()**

Get the number of eigenvalues to compute.

Not collective.

Get the number of eigenvalues to compute, and the dimension of the subspace.

Returns

- **nev** (*int*) – Number of eigenvalues to compute.
- **ncv** (*int*) – Maximum dimension of the subspace to be used by the solver.
- **mpd** (*int*) – Maximum dimension allowed for the projected problem.

Return type*tuple*[*int*, *int*, *int*]**:sources:** `Source code at slepc4py/SLEPc/NEP.pyx:598 <slepc4py/SLEPc/NEP.pyx#L598>`**getEigenpair(*i*, *Vr*=None, *Vi*=None)**

Get the *i*-th solution of the eigenproblem as computed by *solve()*.

Collective.

The solution consists of both the eigenvalue and the eigenvector.

Parameters

- **i** (*int*) – Index of the solution to be obtained.
- **Vr** (*Vec* | *None*) – Placeholder for the returned eigenvector (real part).
- **Vi** (*Vec* | *None*) – Placeholder for the returned eigenvector (imaginary part).

Returns

The computed eigenvalue.

Return type

`complex`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:900 <slepc4py/SLEPc/NEP.pyx#L900>`

getErrorEstimate(*i*)

Get the error estimate associated to the *i*-th computed eigenpair.

Not collective.

Parameters

i (`int`) – Index of the solution to be considered.

Returns

Error estimate.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:958 <slepc4py/SLEPc/NEP.pyx#L958>`

getFunction()

Get the function to compute the nonlinear Function $T(\lambda)$.

Collective.

Get the function to compute the nonlinear Function $T(\lambda)$ and the matrix.

Parameters

- **F** – Function matrix
- **P** – preconditioner matrix (usually the same as the F)
- **function** – Function evaluation routine

Return type

`tuple[petsc4py.PETSc.Mat, petsc4py.PETSc.Mat, NEPFunction]`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1104 <slepc4py/SLEPc/NEP.pyx#L1104>`

getInterpolInterpolation()

Get the tolerance and maximum degree for the interpolation polynomial.

Not collective.

Returns

- **tol** (`float`) – The tolerance to stop computing polynomial coefficients.
- **deg** (`int`) – The maximum degree of interpolation.

Return type

`tuple[float, int]`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1799 <slepc4py/SLEPc/NEP.pyx#L1799>`

getInterpolPEP()

Get the associated polynomial eigensolver object.

Collective.

Get the polynomial eigensolver object associated with the nonlinear eigensolver.

Returns

The polynomial eigensolver.

Return type

PEP

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1758 <slepc4py/SLEPc/NEP.pyx#L1758>`

getIterationNumber()

Get the current iteration number.

Not collective.

If the call to `solve()` is complete, then it returns the number of iterations carried out by the solution method.

Returns

Iteration number.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:851 <slepc4py/SLEPc/NEP.pyx#L851>`

getJacobian()

Get the function to compute the Jacobian $T'(\lambda)$ and J.

Collective.

Get the function to compute the Jacobian $T'(\lambda)$ and the matrix.

Parameters

- **J** – Jacobian matrix
- **jacobian** – Jacobian evaluation routine

Return type

`tuple[petsc4py.PETSc.Mat, NEPJacobian]`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1162 <slepc4py/SLEPc/NEP.pyx#L1162>`

getLeftEigenvector(*i*, *Wr*, *Wi=None*)

Get the *i*-th left eigenvector as computed by `solve()`.

Collective.

Parameters

- **i** (`int`) – Index of the solution to be obtained.
- **Wr** (`Vec`) – Placeholder for the returned eigenvector (real part).
- **Wi** (`Vec` / `None`) – Placeholder for the returned eigenvector (imaginary part).

Return type

`None`

Notes

The index *i* should be a value between 0 and `nconv-1` (see `getConverged()`). Eigensolutions are indexed according to the ordering criterion established with `setWhichEigenpairs()`.

Left eigenvectors are available only if the twosided flag was set with `setTwoSided()`.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:929 <slepc4py/SLEPc/NEP.pyx#L929>`

getMonitor()

Get the list of monitor functions.

Not collective.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:813 <slepc4py/SLEPc/NEP.pyx#L813>`

Return type

`NEPMonitorFunction`

getNArnoldiKSP()

Get the linear solver object associated with the nonlinear eigensolver.

Collective.

Returns

The linear solver object.

Return type

`petsc4py.PETSc.KSP`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1691 <slepc4py/SLEPc/NEP.pyx#L1691>`

getNArnoldiLagPreconditioner()

Get how often the preconditioner is rebuilt.

Not collective.

Returns

The lag parameter.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1728 <slepc4py/SLEPc/NEP.pyx#L1728>`

getNLEIGSEPS()

Get the linear eigensolver object associated with the nonlinear eigensolver.

Collective.

Returns

The linear eigensolver.

Return type

`EPS`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1978 <slepc4py/SLEPc/NEP.pyx#L1978>`

getNLEIGSFULLBASIS()

Get the flag that indicates if NLEIGS is using the full-basis variant.

Not collective.

Returns

True if the full-basis variant must be selected.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1950 <slepc4py/SLEPc/NEP.pyx#L1950>`

getNLEIGSInterpolation()

Get the tolerance and maximum degree for the interpolation polynomial.

Not collective.

Get the tolerance and maximum degree when building the interpolation via divided differences.

Returns

- **tol** (`float`) – The tolerance to stop computing divided differences.
- **deg** (`int`) – The maximum degree of interpolation.

Return type

`tuple[float, int]`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1912 <slepc4py/SLEPc/NEP.pyx#L1912>`

getNLEIGSKSPs()

Get the list of linear solver objects associated with the NLEIGS solver.

Collective.

Returns

The linear solver objects.

Return type

`list of petsc4py.PETSc.KSP`

Notes

The number of `petsc4py.PETSc.KSP` solvers is equal to the number of shifts provided by the user, or 1 if the user did not provide shifts.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2031 <slepc4py/SLEPc/NEP.pyx#L2031>`

getNLEIGSLocking()

Get the locking flag used in the NLEIGS method.

Not collective.

Returns

The locking flag.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1875 <slepc4py/SLEPc/NEP.pyx#L1875>`

getNLEIGSRKShifts()

Get the list of shifts used in the Rational Krylov method.

Not collective.

Returns

The shift values.

Return type

`ArrayScalar`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2010 <slepc4py/SLEPc/NEP.pyx#L2010>`

getNLEIGSRestart()

Get the restart parameter used in the NLEIGS method.

Not collective.

Returns

The number of vectors to be kept at restart.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1839 <slepc4py/SLEPc/NEP.pyx#L1839>`

getOptionsPrefix()

Get the prefix used for searching for all NEP options in the database.

Not collective.

Returns

The prefix string set for this NEP object.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:259 <slepc4py/SLEPc/NEP.pyx#L259>`

getProblemType()

Get the problem type from the *NEP* object.

Not collective.

Returns

The problem type that was previously set.

Return type

ProblemType

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:315 <slepc4py/SLEPc/NEP.pyx#L315>`

getRG()

Get the region object associated to the eigensolver.

Not collective.

Returns

The region context.

Return type

RG

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:682 <slepc4py/SLEPc/NEP.pyx#L682>`

getRIIConstCorrectionTol()

Get the constant tolerance flag.

Not collective.

Returns

If True, the `petsc4py.PETSc.KSP` relative tolerance is constant.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1413 <slepc4py/SLEPc/NEP.pyx#L1413>`

getRIIDeflationThreshold()

Get the threshold value that controls deflation.

Not collective.

Returns

The threshold value.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1512 <slepc4py/SLEPc/NEP.pyx#L1512>`

getRIIHermitian()

Get if the Hermitian version must be used by the solver.

Not collective.

Get the flag about using the Hermitian version of the scalar nonlinear equation.

Returns

If True, the Hermitian version is used.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1477 <slepc4py/SLEPc/NEP.pyx#L1477>`

getRIIKSP()

Get the linear solver object associated with the nonlinear eigensolver.

Collective.

Returns

The linear solver object.

Return type

`petsc4py.PETSc.KSP`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1540 <slepc4py/SLEPc/NEP.pyx#L1540>`

getRIILagPreconditioner()

Get how often the preconditioner is rebuilt.

Not collective.

Returns

The lag parameter.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1384 <slepc4py/SLEPc/NEP.pyx#L1384>`

getRIIMaximumIterations()

Get the maximum number of inner iterations of RII.

Not collective.

Returns

Maximum inner iterations.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1445 <slepc4py/SLEPc/NEP.pyx#L1445>`

getRefine()

Get the refinement strategy used by the NEP object.

Not collective.

Get the refinement strategy used by the NEP object and the associated parameters.

Returns

- **ref** (*Refine*) – The refinement type.
- **npart** (*int*) – The number of partitions of the communicator.
- **tol** (*float*) – The convergence tolerance.
- **its** (*int*) – The maximum number of refinement iterations.
- **scheme** (*RefineScheme*) – Scheme for solving linear systems

Return type

tuple[*Refine*, *int*, *float*, *int*, *RefineScheme*]

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:483 <slepc4py/SLEPc/NEP.pyx#L483>`

getRefineKSP()

Get the KSP object used by the eigensolver in the refinement phase.

Collective.

Returns

The linear solver object.

Return type

petsc4py.PETSc.KSP

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:553 <slepc4py/SLEPc/NEP.pyx#L553>`

getSLPDeflationThreshold()

Get the threshold value that controls deflation.

Not collective.

Returns

The threshold value.

Return type

float

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1572 <slepc4py/SLEPc/NEP.pyx#L1572>`

getSLPEPS()

Get the linear eigensolver object associated with the nonlinear eigensolver.

Collective.

Returns

The linear eigensolver.

Return type

EPS

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1600 <slepc4py/SLEPc/NEP.pyx#L1600>`

getSLPEPSLeft()

Get the left eigensolver.

Collective.

Returns

The linear eigensolver.

Return type

EPS

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1631 <slepc4py/SLEPc/NEP.pyx#L1631>`

getSLPKSP()

Get the linear solver object associated with the nonlinear eigensolver.

Collective.

Returns

The linear solver object.

Return type

petsc4py.PETSc.KSP

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1660 <slepc4py/SLEPc/NEP.pyx#L1660>`

getSplitOperator()

Get the operator of the nonlinear eigenvalue problem in split form.

Collective.

Returns

- **A** (list of *petsc4py.PETSc.Mat*) – Coefficient matrices of the split form.
- **f** (list of *FN*) – Scalar functions of the split form.
- **structure** (*petsc4py.PETSc.Mat.Structure*) – Structure flag for matrices.

Return type

tuple[list[*petsc4py.PETSc.Mat*], list[*FN*], *petsc4py.PETSc.Mat.Structure*]

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1218 <slepc4py/SLEPc/NEP.pyx#L1218>`

getSplitPreconditioner()

Get the operator of the split preconditioner.

Not collective.

Returns

- **P** (list of *petsc4py.PETSc.Mat*) – Coefficient matrices of the split preconditioner.
- **structure** (*petsc4py.PETSc.Mat.Structure*) – Structure flag for matrices.

Return type

tuple[list[*petsc4py.PETSc.Mat*], *petsc4py.PETSc.Mat.Structure*]

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1279 <slepc4py/SLEPc/NEP.pyx#L1279>`

getStoppingTest()

Get the stopping function.

Not collective.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:782 <slepc4py/SLEPc/NEP.pyx#L782>`

Return type*NEPStoppingFunction***getTarget()**

Get the value of the target.

Not collective.

Returns

The value of the target.

Return type*Scalar***Notes**

If the target was not set by the user, then zero is returned.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:373 <slepc4py/SLEPc/NEP.pyx#L373>`

getTolerances()

Get the tolerance and maximum iteration count.

Not collective.

Get the tolerance and maximum iteration count used by the default NEP convergence tests.

Returns

- **tol** (*float*) – The convergence tolerance.
- **maxit** (*int*) – The maximum number of iterations.

Return type*tuple[*float*, *int*]*

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:412 <slepc4py/SLEPc/NEP.pyx#L412>`

getTrackAll()

Get the flag indicating whether all residual norms must be computed.

Not collective.

Returns

Whether the solver compute all residuals or not.

Return type*bool*

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:569 <slepc4py/SLEPc/NEP.pyx#L569>`

getTwoSided()

Get the flag indicating if a two-sided variant is being used.

Not collective.

Get the flag indicating whether a two-sided variant of the algorithm is being used or not.

Returns

Whether the two-sided variant is to be used or not.

Return type*bool*

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1304 <slepc4py/SLEPc/NEP.pyx#L1304>`

getType()

Get the NEP type of this object.

Not collective.

Returns

The solver currently being used.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:244 <slepc4py/SLEPc/NEP.pyx#L244>`

getWhichEigenpairs()

Get which portion of the spectrum is to be sought.

Not collective.

Returns

The portion of the spectrum to be sought by the solver.

Return type

`Which`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:344 <slepc4py/SLEPc/NEP.pyx#L344>`

reset()

Reset the NEP object.

Collective.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:204 <slepc4py/SLEPc/NEP.pyx#L204>`

Return type

`None`

setBV(*bv*)

Set the basis vectors object associated to the eigensolver.

Collective.

Parameters

bv (`BV`) – The basis vectors context.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:669 <slepc4py/SLEPc/NEP.pyx#L669>`

setCISSExtraction(*extraction*)

Set the extraction technique used in the CISS solver.

Logically collective.

Parameters

extraction (`CISSExtraction`) – The extraction technique.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2054 <slepc4py/SLEPc/NEP.pyx#L2054>`

setCISSRefinement(*inner=None, bsize=None*)

Set the values of various refinement parameters in the CISS solver.

Logically collective.

Parameters

- **inner** (*int* / *None*) – Number of iterative refinement iterations (inner loop).
- **bsize** (*int* / *None*) – Number of iterative refinement iterations (blocksize loop).

Return type

None

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:2200 <slepc4py/SLEPc/NEP.pyx#L2200>](#)

setCISSSizes(*ip=None, bs=None, ms=None, npart=None, bsmax=None, realmats=False*)

Set the values of various size parameters in the CISS solver.

Logically collective.

Parameters

- **ip** (*int* / *None*) – Number of integration points.
- **bs** (*int* / *None*) – Block size.
- **ms** (*int* / *None*) – Moment size.
- **npart** (*int* / *None*) – Number of partitions when splitting the communicator.
- **bsmax** (*int* / *None*) – Maximum block size.
- **realmats** (*bool*) – True if A and B are real.

Return type

None

Notes

The default number of partitions is 1. This means the internal `petsc4py.PETSc.KSP` object is shared among all processes of the `NEP` communicator. Otherwise, the communicator is split into `npart` communicators, so that `npart` `petsc4py.PETSc.KSP` solves proceed simultaneously.

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:2083 <slepc4py/SLEPc/NEP.pyx#L2083>](#)

setCISSThreshold(*delta=None, spur=None*)

Set the values of various threshold parameters in the CISS solver.

Logically collective.

Parameters

- **delta** (*float* / *None*) – Threshold for numerical rank.
- **spur** (*float* / *None*) – Spurious threshold (to discard spurious eigenpairs).

Return type

None

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:2163 <slepc4py/SLEPc/NEP.pyx#L2163>](#)

setConvergenceTest(*conv*)

Set how to compute the error estimate used in the convergence test.

Logically collective.

Parameters

conv ([Conv](#)) – The method used to compute the error estimate used in the convergence test.

Return type

[None](#)

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:468 <slepc4py/SLEPc/NEP.pyx#L468>](#)

setDS(*ds*)

Set a direct solver object associated to the eigensolver.

Collective.

Parameters

ds ([DS](#)) – The direct solver context.

Return type

[None](#)

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:727 <slepc4py/SLEPc/NEP.pyx#L727>](#)

setDimensions(*nev=None, ncv=None, mpd=None*)

Set the number of eigenvalues to compute.

Logically collective.

Set the number of eigenvalues to compute and the dimension of the subspace.

Parameters

- **nev** ([int](#) / [None](#)) – Number of eigenvalues to compute.
- **ncv** ([int](#) / [None](#)) – Maximum dimension of the subspace to be used by the solver.
- **mpd** ([int](#) / [None](#)) – Maximum dimension allowed for the projected problem.

Return type

[None](#)

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:622 <slepc4py/SLEPc/NEP.pyx#L622>](#)

setFromOptions()

Set NEP options from the options database.

Collective.

This routine must be called before [setUp\(\)](#) if the user is to be allowed to set the solver type.

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:304 <slepc4py/SLEPc/NEP.pyx#L304>](#)

Return type

[None](#)

setFunction(*function, F=None, P=None, args=None, kargs=None*)

Set the function to compute the nonlinear Function $T(\lambda)$.

Collective.

Set the function to compute the nonlinear Function $T(\lambda)$ as well as the location to store the matrix.

Parameters

- **function** ([NEPFunction](#)) – Function evaluation routine
- **F** ([petsc4py.PETSc.Mat](#) / [None](#)) – Function matrix
- **P** ([petsc4py.PETSc.Mat](#) / [None](#)) – preconditioner matrix (usually the same as F)

- **args** (*tuple*[Any, ...] | None)
- **kargs** (*dict*[str, Any] | None)

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1068 <slepc4py/SLEPc/NEP.pyx#L1068>`

setInitialSpace(*space*)

Set the initial space from which the eigensolver starts to iterate.

Collective.

Parameters

space (*Vec*) – The initial space

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:742 <slepc4py/SLEPc/NEP.pyx#L742>`

setInterpolInterpolation(*tol=None, deg=None*)

Set the tolerance and maximum degree for the interpolation polynomial.

Collective.

Set the tolerance and maximum degree when building the interpolation polynomial.

Parameters

- **tol** (*float* | None) – The tolerance to stop computing polynomial coefficients.
- **deg** (*int* | None) – The maximum degree of interpolation.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1777 <slepc4py/SLEPc/NEP.pyx#L1777>`

setInterpolPEP(*pep*)

Set a polynomial eigensolver object associated to the nonlinear eigensolver.

Collective.

Parameters

pep (*PEP*) – The polynomial eigensolver.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1745 <slepc4py/SLEPc/NEP.pyx#L1745>`

setJacobian(*jacobian, J=None, args=None, kargs=None*)

Set the function to compute the Jacobian $T'(\lambda)$.

Collective.

Set the function to compute the Jacobian $T'(\lambda)$ as well as the location to store the matrix.

Parameters

- **jacobian** (*NEPJacobian*) – Jacobian evaluation routine
- **J** (*petsc4py.PETSc.Mat* | None) – Jacobian matrix
- **args** (*tuple*[Any, ...] | None)

- **kargs** (*dict*[*str*, *Any*] | *None*)

Return type

None

:sources: ``Source code at slepc4py/SLEPc/NEP.pyx:1130 <slepc4py/SLEPc/NEP.pyx#L1130>``

setMonitor(*monitor*, *args=None*, *kargs=None*)

Append a monitor function to the list of monitors.

Logically collective.

:sources: ``Source code at slepc4py/SLEPc/NEP.pyx:792 <slepc4py/SLEPc/NEP.pyx#L792>``

Parameters

- **monitor** (*NEPMonitorFunction* | *None*)
- **args** (*tuple*[*Any*, ...] | *None*)
- **kargs** (*dict*[*str*, *Any*] | *None*)

Return type

None

setArnoldiKSP(*ksp*)

Set a linear solver object associated to the nonlinear eigensolver.

Collective.

Parameters

ksp (*petsc4py.PETSc.KSP*) – The linear solver object.

Return type

None

:sources: ``Source code at slepc4py/SLEPc/NEP.pyx:1678 <slepc4py/SLEPc/NEP.pyx#L1678>``

setArnoldiLagPreconditioner(*lag*)

Set when the preconditioner is rebuilt in the nonlinear solve.

Logically collective.

Parameters

lag (*int*) – 0 indicates NEVER rebuild, 1 means rebuild every time the Jacobian is computed within the nonlinear iteration, 2 means every second time the Jacobian is built, etc.

Return type

None

Notes

The default is 1. The preconditioner is ALWAYS built in the first iteration of a nonlinear solve.

:sources: ``Source code at slepc4py/SLEPc/NEP.pyx:1707 <slepc4py/SLEPc/NEP.pyx#L1707>``

setNLEIGSEPS(*eps*)

Set a linear eigensolver object associated to the nonlinear eigensolver.

Collective.

Parameters

eps (*EPS*) – The linear eigensolver.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1965 <slepc4py/SLEPc/NEP.pyx#L1965>`

setNLEIGSFULLBasis(*fullbasis=True*)

Set TOAR-basis (default) or full-basis variants of the NLEIGS method.

Logically collective.

Toggle between TOAR-basis (default) and full-basis variants of the NLEIGS method.

Parameters

fullbasis (*bool*) – True if the full-basis variant must be selected.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1933 <slepc4py/SLEPc/NEP.pyx#L1933>`

setNLEIGSInterpolation(*tol=None, deg=None*)

Set the tolerance and maximum degree for the interpolation polynomial.

Collective.

Set the tolerance and maximum degree when building the interpolation via divided differences.

Parameters

- **tol** (*float* / *None*) – The tolerance to stop computing divided differences.
- **deg** (*int* / *None*) – The maximum degree of interpolation.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1890 <slepc4py/SLEPc/NEP.pyx#L1890>`

setNLEIGSLocking(*lock*)

Toggle between locking and non-locking variants of the NLEIGS method.

Logically collective.

Parameters

lock (*bool*) – True if the locking variant must be selected.

Return type

None

Notes

The default is to lock converged eigenpairs when the method restarts. This behavior can be changed so that all directions are kept in the working subspace even if already converged to working accuracy (the non-locking variant).

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1854 <slepc4py/SLEPc/NEP.pyx#L1854>`

setNLEIGSRKShifts(*shifts*)

Set a list of shifts to be used in the Rational Krylov method.

Collective.

Parameters

shifts (*Sequence*[*Scalar*]) – Values specifying the shifts.

Return type

None

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:1994 <slepc4py/SLEPc/NEP.pyx#L1994>](#)

setNLEIGSRestart (*keep*)

Set the restart parameter for the NLEIGS method.

Logically collective.

The proportion of basis vectors that must be kept after restart.

Parameters

keep (*float*) – The number of vectors to be kept at restart.

Return type

None

Notes

Allowed values are in the range [0.1,0.9]. The default is 0.5.

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:1819 <slepc4py/SLEPc/NEP.pyx#L1819>](#)

setOptionsPrefix (*prefix=None*)

Set the prefix used for searching for all NEP options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all NEP option requests.

Return type

None

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:274 <slepc4py/SLEPc/NEP.pyx#L274>](#)

setProblemType (*problem_type*)

Set the type of the eigenvalue problem.

Logically collective.

Parameters

problem_type (*ProblemType*) – The problem type to be set.

Return type

None

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:330 <slepc4py/SLEPc/NEP.pyx#L330>](#)

setRG (*rg*)

Set a region object associated to the eigensolver.

Collective.

Parameters

rg (*RG*) – The region context.

Return type

None

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:698 <slepc4py/SLEPc/NEP.pyx#L698>](#)

setRIIConstCorrectionTol(*cct*)

Set a flag to keep the tolerance used in the linear solver constant.

Logically collective.

Parameters

cct (*bool*) – If True, the `petsc4py.PETSc.KSP` relative tolerance is constant.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1399 <slepc4py/SLEPc/NEP.pyx#L1399>`

setRIIDeflationThreshold(*deftol*)

Set the threshold used to switch between deflated and non-deflated.

Logically collective.

Set the threshold value used to switch between deflated and non-deflated iteration.

Parameters

deftol (*float*) – The threshold value.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1495 <slepc4py/SLEPc/NEP.pyx#L1495>`

setRIIHermitian(*herm*)

Set a flag to use the Hermitian version of the solver.

Logically collective.

Set a flag to indicate if the Hermitian version of the scalar nonlinear equation must be used by the solver.

Parameters

herm (*bool*) – If True, the Hermitian version is used.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1460 <slepc4py/SLEPc/NEP.pyx#L1460>`

setRIIKSP(*ksp*)

Set a linear solver object associated to the nonlinear eigensolver.

Collective.

Parameters

ksp (*petsc4py.PETSc.KSP*) – The linear solver object.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1527 <slepc4py/SLEPc/NEP.pyx#L1527>`

setRIILagPreconditioner(*lag*)

Set when the preconditioner is rebuilt in the nonlinear solve.

Logically collective.

Parameters

lag (*int*) – 0 indicates NEVER rebuild, 1 means rebuild every time the Jacobian is computed within the nonlinear iteration, 2 means every second time the Jacobian is built, etc.

Return type

None

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:1368 <slepc4py/SLEPc/NEP.pyx#L1368>](#)

setRIIMaximumIterations(*its*)

Set the max. number of inner iterations to be used in the RII solver.

Logically collective.

These are the Newton iterations related to the computation of the nonlinear Rayleigh functional.

Parameters

its (*int*) – Maximum inner iterations.

Return type

None

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:1428 <slepc4py/SLEPc/NEP.pyx#L1428>](#)

setRefine(*ref*, *npart*=None, *tol*=None, *its*=None, *scheme*=None)

Set the refinement strategy used by the NEP object.

Logically collective.

Set the refinement strategy used by the NEP object and the associated parameters.

Parameters

- **ref** (*Refine*) – The refinement type.
- **npart** (*int* / *None*) – The number of partitions of the communicator.
- **tol** (*float* / *None*) – The convergence tolerance.
- **its** (*int* / *None*) – The maximum number of refinement iterations.
- **scheme** (*RefineScheme* / *None*) – Scheme for linear system solves

Return type

None

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:513 <slepc4py/SLEPc/NEP.pyx#L513>](#)

setSLPDeflationThreshold(*deftol*)

Set the threshold used to switch between deflated and non-deflated.

Logically collective.

Parameters

deftol (*float*) – The threshold value.

Return type

None

:sources: [Source code at slepc4py/SLEPc/NEP.pyx:1558 <slepc4py/SLEPc/NEP.pyx#L1558>](#)

setSLPEPS(*eps*)

Set a linear eigensolver object associated to the nonlinear eigensolver.

Collective.

Parameters

eps (*EPS*) – The linear eigensolver.

Return type

None

:sources:`Source code at slepc4py/SLEPc/NEP.pyx:1587 <slepc4py/SLEPc/NEP.pyx#L1587>`

setSLPEPSLeft(*eps*)

Set a linear eigensolver object associated to the nonlinear eigensolver.

Collective.

Used to compute left eigenvectors in the two-sided variant of SLP.

Parameters

eps ([EPS](#)) – The linear eigensolver.

Return type

[None](#)

:sources:`Source code at slepc4py/SLEPc/NEP.pyx:1616 <slepc4py/SLEPc/NEP.pyx#L1616>`

setSLPKSP(*ksp*)

Set a linear solver object associated to the nonlinear eigensolver.

Collective.

Parameters

ksp ([petsc4py.PETSc.KSP](#)) – The linear solver object.

Return type

[None](#)

:sources:`Source code at slepc4py/SLEPc/NEP.pyx:1647 <slepc4py/SLEPc/NEP.pyx#L1647>`

setSplitOperator(*A*, *f*, *structure=None*)

Set the operator of the nonlinear eigenvalue problem in split form.

Collective.

Parameters

- **A** ([petsc4py.PETSc.Mat](#) / [list\[petsc4py.PETSc.Mat\]](#)) – Coefficient matrices of the split form.
- **f** ([FN](#) / [list\[FN\]](#)) – Scalar functions of the split form.
- **structure** ([petsc4py.PETSc.Mat.Structure](#) / [None](#)) – Structure flag for matrices.

Return type

[None](#)

:sources:`Source code at slepc4py/SLEPc/NEP.pyx:1184 <slepc4py/SLEPc/NEP.pyx#L1184>`

setSplitPreconditioner(*P*, *structure=None*)

Set the operator in split form.

Collective.

Set the operator in split form from which to build the preconditioner to be used when solving the nonlinear eigenvalue problem in split form.

Parameters

- **P** ([petsc4py.PETSc.Mat](#) / [list\[petsc4py.PETSc.Mat\]](#)) – Coefficient matrices of the split preconditioner.
- **structure** ([petsc4py.PETSc.Mat.Structure](#) / [None](#)) – Structure flag for matrices.

Return type

[None](#)

:sources:`Source code at slepc4py/SLEPc/NEP.pyx:1250 <slepc4py/SLEPc/NEP.pyx#L1250>`

setStoppingTest(*stopping*, *args=None*, *kargs=None*)

Set a function to decide when to stop the outer iteration of the eigensolver.

Logically collective.

:sources:`Source code at slepc4py/SLEPc/NEP.pyx:762 <slepc4py/SLEPc/NEP.pyx#L762>`

Parameters

- **stopping** (*NEPStoppingFunction* | *None*)
- **args** (*tuple*[*Any*, ...] | *None*)
- **kargs** (*dict*[*str*, *Any*] | *None*)

Return type

None

setTarget(*target*)

Set the value of the target.

Logically collective.

Parameters

target (*Scalar*) – The value of the target.

Return type

None

Notes

The target is a scalar value used to determine the portion of the spectrum of interest. It is used in combination with *setWhichEigenpairs()*.

:sources:`Source code at slepc4py/SLEPc/NEP.pyx:392 <slepc4py/SLEPc/NEP.pyx#L392>`

setTolerances(*tol=None*, *maxit=None*)

Set the tolerance and max. iteration count used in convergence tests.

Logically collective.

Parameters

- **tol** (*float* | *None*) – The convergence tolerance.
- **maxit** (*int* | *None*) – The maximum number of iterations.

Return type

None

:sources:`Source code at slepc4py/SLEPc/NEP.pyx:433 <slepc4py/SLEPc/NEP.pyx#L433>`

setTrackAll(*trackall*)

Set if the solver must compute the residual of all approximate eigenpairs.

Logically collective.

Parameters

trackall (*bool*) – Whether compute all residuals or not.

Return type

None

:sources:`Source code at slepc4py/SLEPc/NEP.pyx:584 <slepc4py/SLEPc/NEP.pyx#L584>`

setTwoSided(*twosided*)

Set the solver to use a two-sided variant.

Logically collective.

Set the solver to use a two-sided variant so that left eigenvectors are also computed.

Parameters

twosided (*bool*) – Whether the two-sided variant is to be used or not.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1322 <slepc4py/SLEPc/NEP.pyx#L1322>`

setType(*nep_type*)

Set the particular solver to be used in the NEP object.

Logically collective.

Parameters

nep_type (*Type* / *str*) – The solver to be used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:229 <slepc4py/SLEPc/NEP.pyx#L229>`

setUp()

Set up all the necessary internal data structures.

Collective.

Set up all the internal data structures necessary for the execution of the eigensolver.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:832 <slepc4py/SLEPc/NEP.pyx#L832>`

Return type

None

setWhichEigenpairs(*which*)

Set which portion of the spectrum is to be sought.

Logically collective.

Parameters

which (*Which*) – The portion of the spectrum to be sought by the solver.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:359 <slepc4py/SLEPc/NEP.pyx#L359>`

solve()

Solve the eigensystem.

Collective.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:843 <slepc4py/SLEPc/NEP.pyx#L843>`

Return type

None

valuesView(viewer=None)

Display the computed eigenvalues in a viewer.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1036 <slepc4py/SLEPc/NEP.pyx#L1036>`

vectorsView(viewer=None)

Output computed eigenvectors to a viewer.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:1051 <slepc4py/SLEPc/NEP.pyx#L1051>`

view(viewer=None)

Print the NEP data structure.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:179 <slepc4py/SLEPc/NEP.pyx#L179>`

Attributes Documentation

bv

The basis vectors (BV) object associated.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2302 <slepc4py/SLEPc/NEP.pyx#L2302>`

ds

The direct solver (DS) object associated.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2316 <slepc4py/SLEPc/NEP.pyx#L2316>`

max_it

The maximum iteration count used by the NEP convergence tests.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2288 <slepc4py/SLEPc/NEP.pyx#L2288>`

problem_type

The problem type from the NEP object.

:sources: `Source code at slepc4py/SLEPc/NEP.pyx:2260 <slepc4py/SLEPc/NEP.pyx#L2260>`

rg

The region (RG) object associated.

:sources: ``Source code at slepc4py/SLEPc/NEP.pyx:2309 <slepc4py/SLEPc/NEP.pyx#L2309>``

target

The value of the target.

:sources: ``Source code at slepc4py/SLEPc/NEP.pyx:2274 <slepc4py/SLEPc/NEP.pyx#L2274>``

tol

The tolerance used by the NEP convergence tests.

:sources: ``Source code at slepc4py/SLEPc/NEP.pyx:2281 <slepc4py/SLEPc/NEP.pyx#L2281>``

track_all

Compute the residual of all approximate eigenpairs.

:sources: ``Source code at slepc4py/SLEPc/NEP.pyx:2295 <slepc4py/SLEPc/NEP.pyx#L2295>``

which

The portion of the spectrum to be sought.

:sources: ``Source code at slepc4py/SLEPc/NEP.pyx:2267 <slepc4py/SLEPc/NEP.pyx#L2267>``

slepc4py.SLEPc.PEP

class `slepc4py.SLEPc.PEP`

Bases: `Object`

PEP.

Enumerations

<i>Basis</i>	PEP basis type for the representation of the polynomial.
<i>CISSExtraction</i>	PEP CISS extraction technique.
<i>Conv</i>	PEP convergence test.
<i>ConvergedReason</i>	PEP convergence reasons.
<i>ErrorType</i>	PEP error type to assess accuracy of computed solutions.
<i>Extract</i>	PEP extraction strategy used.
<i>JDProjection</i>	PEP type of projection to be used in the Jacobi-Davidson solver.
<i>ProblemType</i>	PEP problem type.
<i>Refine</i>	PEP refinement strategy.
<i>RefineScheme</i>	PEP scheme for solving linear systems during iterative refinement.
<i>Scale</i>	PEP scaling strategy.
<i>Stop</i>	PEP stopping test.
<i>Type</i>	PEP type.
<i>Which</i>	PEP desired part of spectrum.

slepc4py.SLEPc.PEP.Basis

class slepc4py.SLEPc.PEP.Basis

Bases: `object`

PEP basis type for the representation of the polynomial.

- *MONOMIAL*: Monomials (default).
- *CHEBYSHEV1*: Chebyshev polynomials of the 1st kind.
- *CHEBYSHEV2*: Chebyshev polynomials of the 2nd kind.
- *LEGENDRE*: Legendre polynomials.
- *LAGUERRE*: Laguerre polynomials.
- *HERMITE*: Hermite polynomials.

Attributes Summary

<i>CHEBYSHEV1</i>	Constant CHEBYSHEV1 of type <code>int</code>
<i>CHEBYSHEV2</i>	Constant CHEBYSHEV2 of type <code>int</code>
<i>HERMITE</i>	Constant HERMITE of type <code>int</code>
<i>LAGUERRE</i>	Constant LAGUERRE of type <code>int</code>
<i>LEGENDRE</i>	Constant LEGENDRE of type <code>int</code>
<i>MONOMIAL</i>	Constant MONOMIAL of type <code>int</code>

Attributes Documentation

CHEBYSHEV1: `int` = CHEBYSHEV1

Constant CHEBYSHEV1 of type `int`

CHEBYSHEV2: `int` = CHEBYSHEV2

Constant CHEBYSHEV2 of type `int`

HERMITE: `int` = HERMITE

Constant HERMITE of type `int`

LAGUERRE: `int` = LAGUERRE

Constant LAGUERRE of type `int`

LEGENDRE: `int` = LEGENDRE

Constant LEGENDRE of type `int`

MONOMIAL: `int` = MONOMIAL

Constant MONOMIAL of type `int`

slepc4py.SLEPc.PEP.CISSExtraction

class slepc4py.SLEPc.PEP.CISSExtraction

Bases: `object`

PEP CISS extraction technique.

- *RITZ*: Ritz extraction.
- *HANKEL*: Extraction via Hankel eigenproblem.
- *CAA*: Communication-avoiding Arnoldi.

Attributes Summary

<i>CAA</i>	Constant CAA of type <code>int</code>
<i>HANKEL</i>	Constant HANKEL of type <code>int</code>
<i>RITZ</i>	Constant RITZ of type <code>int</code>

Attributes Documentation

CAA: `int` = CAA

Constant CAA of type `int`

HANKEL: `int` = HANKEL

Constant HANKEL of type `int`

RITZ: `int` = RITZ

Constant RITZ of type `int`

slepc4py.SLEPc.PEP.Conv

class slepc4py.SLEPc.PEP.Conv

Bases: `object`

PEP convergence test.

- *ABS*: Absolute convergence test.
- *REL*: Convergence test relative to the eigenvalue.
- *NORM*: Convergence test relative to the matrix norms.
- *USER*: User-defined convergence test.

Attributes Summary

<i>ABS</i>	Constant ABS of type <code>int</code>
<i>NORM</i>	Constant NORM of type <code>int</code>
<i>REL</i>	Constant REL of type <code>int</code>
<i>USER</i>	Constant USER of type <code>int</code>

Attributes Documentation

ABS: `int` = ABS

Constant ABS of type `int`

NORM: `int` = NORM

Constant NORM of type `int`

REL: `int` = REL

Constant REL of type `int`

USER: `int` = USER

Constant USER of type `int`

slepc4py.SLEPc.PEP.ConvergedReason

class slepc4py.SLEPc.PEP.ConvergedReason

Bases: `object`

PEP convergence reasons.

- *CONVERGED_TOL*: All eigenpairs converged to requested tolerance.
- *CONVERGED_USER*: User-defined convergence criterion satisfied.
- *DIVERGED_ITS*: Maximum number of iterations exceeded.
- *DIVERGED_BREAKDOWN*: Solver failed due to breakdown.
- *DIVERGED_SYMMETRY_LOST*: Lanczos-type method could not preserve symmetry.
- *CONVERGED_ITERATING*: Iteration not finished yet.

Attributes Summary

<i>CONVERGED_ITERATING</i>	Constant <i>CONVERGED_ITERATING</i> of type <code>int</code>
<i>CONVERGED_TOL</i>	Constant <i>CONVERGED_TOL</i> of type <code>int</code>
<i>CONVERGED_USER</i>	Constant <i>CONVERGED_USER</i> of type <code>int</code>
<i>DIVERGED_BREAKDOWN</i>	Constant <i>DIVERGED_BREAKDOWN</i> of type <code>int</code>
<i>DIVERGED_ITS</i>	Constant <i>DIVERGED_ITS</i> of type <code>int</code>
<i>DIVERGED_SYMMETRY_LOST</i>	Constant <i>DIVERGED_SYMMETRY_LOST</i> of type <code>int</code>
<i>ITERATING</i>	Constant <i>ITERATING</i> of type <code>int</code>

Attributes Documentation

CONVERGED_ITERATING: `int` = *CONVERGED_ITERATING*

Constant *CONVERGED_ITERATING* of type `int`

CONVERGED_TOL: `int` = *CONVERGED_TOL*

Constant *CONVERGED_TOL* of type `int`

CONVERGED_USER: `int` = *CONVERGED_USER*

Constant *CONVERGED_USER* of type `int`

DIVERGED_BREAKDOWN: `int` = *DIVERGED_BREAKDOWN*

Constant *DIVERGED_BREAKDOWN* of type `int`

DIVERGED_ITS: `int` = *DIVERGED_ITS*

Constant *DIVERGED_ITS* of type `int`

DIVERGED_SYMMETRY_LOST: `int` = *DIVERGED_SYMMETRY_LOST*

Constant *DIVERGED_SYMMETRY_LOST* of type `int`

ITERATING: `int` = *ITERATING*

Constant *ITERATING* of type `int`

slepc4py.SLEPc.PEP.ErrorType

class slepc4py.SLEPc.PEP.ErrorType

Bases: `object`

PEP error type to assess accuracy of computed solutions.

- *ABSOLUTE*: Absolute error.
- *RELATIVE*: Relative error.
- *BACKWARD*: Backward error.

Attributes Summary

<i>ABSOLUTE</i>	Constant ABSOLUTE of type <code>int</code>
<i>BACKWARD</i>	Constant BACKWARD of type <code>int</code>
<i>RELATIVE</i>	Constant RELATIVE of type <code>int</code>

Attributes Documentation

ABSOLUTE: `int` = **ABSOLUTE**

Constant ABSOLUTE of type `int`

BACKWARD: `int` = **BACKWARD**

Constant BACKWARD of type `int`

RELATIVE: `int` = **RELATIVE**

Constant RELATIVE of type `int`

`slepc4py.SLEPc.PEP.Extract`

class `slepc4py.SLEPc.PEP.Extract`

Bases: `object`

PEP extraction strategy used.

PEP extraction strategy used to obtain eigenvectors of the PEP from the eigenvectors of the linearization.

- *NONE*: Use the first block.
- *NORM*: Use the first or last block depending on norm of H.
- *RESIDUAL*: Use the block with smallest residual.
- *STRUCTURED*: Combine all blocks in a certain way.

Attributes Summary

<i>NONE</i>	Constant NONE of type <code>int</code>
<i>NORM</i>	Constant NORM of type <code>int</code>
<i>RESIDUAL</i>	Constant RESIDUAL of type <code>int</code>
<i>STRUCTURED</i>	Constant STRUCTURED of type <code>int</code>

Attributes Documentation

NONE: `int` = **NONE**

Constant NONE of type `int`

NORM: `int` = **NORM**

Constant NORM of type `int`

RESIDUAL: `int` = RESIDUAL

Constant RESIDUAL of type `int`

STRUCTURED: `int` = STRUCTURED

Constant STRUCTURED of type `int`

slepc4py.SLEPc.PEP.JDProjection

class slepc4py.SLEPc.PEP.JDProjection

Bases: `object`

PEP type of projection to be used in the Jacobi-Davidson solver.

- *HARMONIC*: Harmonic projection.
- *ORTHOGONAL*: Orthogonal projection.

Attributes Summary

<i>HARMONIC</i>	Constant HARMONIC of type <code>int</code>
<i>ORTHOGONAL</i>	Constant ORTHOGONAL of type <code>int</code>

Attributes Documentation

HARMONIC: `int` = HARMONIC

Constant HARMONIC of type `int`

ORTHOGONAL: `int` = ORTHOGONAL

Constant ORTHOGONAL of type `int`

slepc4py.SLEPc.PEP.ProblemType

class slepc4py.SLEPc.PEP.ProblemType

Bases: `object`

PEP problem type.

- *GENERAL*: No structure.
- *HERMITIAN*: Hermitian structure.
- *HYPERBOLIC*: QEP with Hermitian matrices, $M > 0$,
 $(x^T C x)^2 > 4(x^T M x)(x^T K x)$.
- *GYROSCOPIC*: QEP with M, K Hermitian,
 $M > 0$, C skew-Hermitian.

Attributes Summary

<i>GENERAL</i>	Constant GENERAL of type <code>int</code>
<i>GYROSCOPIC</i>	Constant GYROSCOPIC of type <code>int</code>
<i>HERMITIAN</i>	Constant HERMITIAN of type <code>int</code>
<i>HYPERBOLIC</i>	Constant HYPERBOLIC of type <code>int</code>

Attributes Documentation

GENERAL: `int` = **GENERAL**

Constant GENERAL of type `int`

GYROSCOPIC: `int` = **GYROSCOPIC**

Constant GYROSCOPIC of type `int`

HERMITIAN: `int` = **HERMITIAN**

Constant HERMITIAN of type `int`

HYPERBOLIC: `int` = **HYPERBOLIC**

Constant HYPERBOLIC of type `int`

`slepc4py.SLEPc.PEP.Refine`

class `slepc4py.SLEPc.PEP.Refine`

Bases: `object`

PEP refinement strategy.

- *NONE*: No refinement.
- *SIMPLE*: Refine eigenpairs one by one.
- *MULTIPLE*: Refine all eigenpairs simultaneously (invariant pair).

Attributes Summary

<i>MULTIPLE</i>	Constant MULTIPLE of type <code>int</code>
<i>NONE</i>	Constant NONE of type <code>int</code>
<i>SIMPLE</i>	Constant SIMPLE of type <code>int</code>

Attributes Documentation

MULTIPLE: `int` = **MULTIPLE**

Constant MULTIPLE of type `int`

NONE: `int` = **NONE**

Constant NONE of type `int`

SIMPLE: `int` = **SIMPLE**

Constant SIMPLE of type `int`

`slepc4py.SLEPc.PEP.RefineScheme`

class `slepc4py.SLEPc.PEP.RefineScheme`

Bases: `object`

PEP scheme for solving linear systems during iterative refinement.

- *SCHUR*: Schur complement.
- *MBE*: Mixed block elimination.
- *EXPLICIT*: Build the explicit matrix.

Attributes Summary

<i>EXPLICIT</i>	Constant EXPLICIT of type <code>int</code>
<i>MBE</i>	Constant MBE of type <code>int</code>
<i>SCHUR</i>	Constant SCHUR of type <code>int</code>

Attributes Documentation

EXPLICIT: `int` = EXPLICIT

Constant EXPLICIT of type `int`

MBE: `int` = MBE

Constant MBE of type `int`

SCHUR: `int` = SCHUR

Constant SCHUR of type `int`

slepc4py.SLEPc.PEP.Scale

class slepc4py.SLEPc.PEP.Scale

Bases: `object`

PEP scaling strategy.

- *NONE*: No scaling.
- *SCALAR*: Parameter scaling.
- *DIAGONAL*: Diagonal scaling.
- *BOTH*: Both parameter and diagonal scaling.

Attributes Summary

<i>BOTH</i>	Constant BOTH of type <code>int</code>
<i>DIAGONAL</i>	Constant DIAGONAL of type <code>int</code>
<i>NONE</i>	Constant NONE of type <code>int</code>
<i>SCALAR</i>	Constant SCALAR of type <code>int</code>

Attributes Documentation

BOTH: `int` = BOTH

Constant BOTH of type `int`

DIAGONAL: `int` = DIAGONAL

Constant DIAGONAL of type `int`

NONE: `int` = NONE

Constant NONE of type `int`

SCALAR: `int` = SCALAR

Constant SCALAR of type `int`

slepc4py.SLEPc.PEP.Stop

class slepc4py.SLEPc.PEP.Stop

Bases: `object`

PEP stopping test.

- *BASIC*: Default stopping test.
- *USER*: User-defined stopping test.

Attributes Summary

<i>BASIC</i>	Constant BASIC of type <code>int</code>
<i>USER</i>	Constant USER of type <code>int</code>

Attributes Documentation

BASIC: `int` = BASIC

Constant BASIC of type `int`

USER: `int` = USER

Constant USER of type `int`

slepc4py.SLEPc.PEP.Type

class slepc4py.SLEPc.PEP.Type

Bases: `object`

PEP type.

Polynomial eigensolvers.

- *LINEAR*: Linearization via EPS.
- *QARNOLDI*: Q-Arnoldi for quadratic problems.
- *TOAR*: Two-level orthogonal Arnoldi.
- *STOAR*: Symmetric TOAR.
- *JD*: Polynomial Jacobi-Davidson.
- *CISS*: Contour integral spectrum slice.

Attributes Summary

<i>CISS</i>	Object CISS of type <code>str</code>
<i>JD</i>	Object JD of type <code>str</code>
<i>LINEAR</i>	Object LINEAR of type <code>str</code>
<i>QARNOLDI</i>	Object QARNOLDI of type <code>str</code>
<i>STOAR</i>	Object STOAR of type <code>str</code>
<i>TOAR</i>	Object TOAR of type <code>str</code>

Attributes Documentation

CISS: `str` = CISS

Object CISS of type `str`

JD: `str` = JD

Object JD of type `str`

LINEAR: `str` = LINEAR

Object LINEAR of type `str`

QARNOLDI: `str` = QARNOLDI

Object QARNOLDI of type `str`

STOAR: `str` = STOAR

Object STOAR of type `str`

TOAR: `str` = TOAR

Object TOAR of type `str`

slepc4py.SLEPc.PEP.Which

class slepc4py.SLEPc.PEP.Which

Bases: `object`

PEP desired part of spectrum.

- *LARGEST_MAGNITUDE*: Largest magnitude (default).
- *SMALLEST_MAGNITUDE*: Smallest magnitude.
- *LARGEST_REAL*: Largest real parts.
- *SMALLEST_REAL*: Smallest real parts.
- *LARGEST_IMAGINARY*: Largest imaginary parts in magnitude.
- *SMALLEST_IMAGINARY*: Smallest imaginary parts in magnitude.
- *TARGET_MAGNITUDE*: Closest to target (in magnitude).
- *TARGET_REAL*: Real part closest to target.
- *TARGET_IMAGINARY*: Imaginary part closest to target.
- *ALL*: All eigenvalues in an interval.
- *USER*: User-defined criterion.

Attributes Summary

<i>ALL</i>	Constant ALL of type <code>int</code>
<i>LARGEST_IMAGINARY</i>	Constant LARGEST_IMAGINARY of type <code>int</code>
<i>LARGEST_MAGNITUDE</i>	Constant LARGEST_MAGNITUDE of type <code>int</code>
<i>LARGEST_REAL</i>	Constant LARGEST_REAL of type <code>int</code>
<i>SMALLEST_IMAGINARY</i>	Constant SMALLEST_IMAGINARY of type <code>int</code>
<i>SMALLEST_MAGNITUDE</i>	Constant SMALLEST_MAGNITUDE of type <code>int</code>
<i>SMALLEST_REAL</i>	Constant SMALLEST_REAL of type <code>int</code>
<i>TARGET_IMAGINARY</i>	Constant TARGET_IMAGINARY of type <code>int</code>

continues on next page

Table 83 – continued from previous page

<i>TARGET_MAGNITUDE</i>	Constant TARGET_MAGNITUDE of type <i>int</i>
<i>TARGET_REAL</i>	Constant TARGET_REAL of type <i>int</i>
<i>USER</i>	Constant USER of type <i>int</i>

Attributes Documentation

ALL: *int* = ALL

Constant ALL of type *int*

LARGEST_IMAGINARY: *int* = LARGEST_IMAGINARY

Constant LARGEST_IMAGINARY of type *int*

LARGEST_MAGNITUDE: *int* = LARGEST_MAGNITUDE

Constant LARGEST_MAGNITUDE of type *int*

LARGEST_REAL: *int* = LARGEST_REAL

Constant LARGEST_REAL of type *int*

SMALLEST_IMAGINARY: *int* = SMALLEST_IMAGINARY

Constant SMALLEST_IMAGINARY of type *int*

SMALLEST_MAGNITUDE: *int* = SMALLEST_MAGNITUDE

Constant SMALLEST_MAGNITUDE of type *int*

SMALLEST_REAL: *int* = SMALLEST_REAL

Constant SMALLEST_REAL of type *int*

TARGET_IMAGINARY: *int* = TARGET_IMAGINARY

Constant TARGET_IMAGINARY of type *int*

TARGET_MAGNITUDE: *int* = TARGET_MAGNITUDE

Constant TARGET_MAGNITUDE of type *int*

TARGET_REAL: *int* = TARGET_REAL

Constant TARGET_REAL of type *int*

USER: *int* = USER

Constant USER of type *int*

Methods Summary

<i>appendOptionsPrefix</i> ([prefix])	Append to the prefix used for searching for all PEP options in the database.
<i>cancelMonitor</i> ()	Clear all monitors for a <i>PEP</i> object.
<i>computeError</i> (i[, etype])	Compute the error associated with the i-th computed eigenpair.
<i>create</i> ([comm])	Create the PEP object.
<i>destroy</i> ()	Destroy the PEP object.
<i>errorView</i> ([etype, viewer])	Display the errors associated with the computed solution.
<i>getBV</i> ()	Get the basis vectors object associated to the eigen-solver.
<i>getBasis</i> ()	Get the type of polynomial basis used.

continues on next page

Table 84 – continued from previous page

<code>getCISSExtraction()</code>	Get the extraction technique used in the CISS solver.
<code>getCISSKSPs()</code>	Get the array of linear solver objects associated with the CISS solver.
<code>getCISSRefinement()</code>	Get the values of various refinement parameters in the CISS solver.
<code>getCISSSizes()</code>	Get the values of various size parameters in the CISS solver.
<code>getCISSThreshold()</code>	Get the values of various threshold parameters in the CISS solver.
<code>getConverged()</code>	Get the number of converged eigenpairs.
<code>getConvergedReason()</code>	Get the reason why the <code>solve()</code> iteration was stopped.
<code>getConvergenceTest()</code>	Get the method used to compute the error estimate used in the convergence test.
<code>getDS()</code>	Get the direct solver associated to the eigensolver.
<code>getDimensions()</code>	Get the number of eigenvalues to compute and the dimension of the subspace.
<code>getEigenpair(i[, Vr, Vi])</code>	Get the i-th solution of the eigenproblem as computed by <code>solve()</code> .
<code>getErrorEstimate(i)</code>	Get the error estimate associated to the i-th computed eigenpair.
<code>getExtract()</code>	Get the extraction technique used by the <code>PEP</code> object.
<code>getInterval()</code>	Get the computational interval for spectrum slicing.
<code>getIterationNumber()</code>	Get the current iteration number.
<code>getJDFix()</code>	Get threshold for changing the target in the correction equation.
<code>getJDMINimalityIndex()</code>	Get the maximum allowed value of the minimality index.
<code>getJDProjection()</code>	Get the type of projection to be used in the Jacobi-Davidson solver.
<code>getJDRestart()</code>	Get the restart parameter used in the Jacobi-Davidson method.
<code>getJDReusePreconditioner()</code>	Get the flag for reusing the preconditioner.
<code>getLinearEPS()</code>	Get the eigensolver object associated to the polynomial eigenvalue solver.
<code>getLinearExplicitMatrix()</code>	Get if the matrices A and B for the linearization are built explicitly.
<code>getLinearLinearization()</code>	Get the coeffs.
<code>getMonitor()</code>	Get the list of monitor functions.
<code>getOperators()</code>	Get the matrices associated with the eigenvalue problem.
<code>getOptionsPrefix()</code>	Get the prefix used for searching for all PEP options in the database.
<code>getProblemType()</code>	Get the problem type from the PEP object.
<code>getQArnoldiLocking()</code>	Get the locking flag used in the Q-Arnoldi method.
<code>getQArnoldiRestart()</code>	Get the restart parameter used in the Q-Arnoldi method.
<code>getRG()</code>	Get the region object associated to the eigensolver.
<code>getRefine()</code>	Get the refinement strategy used by the PEP object.
<code>getRefineKSP()</code>	Get the KSP object used by the eigensolver in the refinement phase.
<code>getST()</code>	Get the <code>ST</code> object associated to the eigensolver object.

continues on next page

Table 84 – continued from previous page

<i>getSTOARCheckEigenvalueType()</i>	Get the flag for the eigenvalue type check in spectrum slicing.
<i>getSTOARDetectZeros()</i>	Get the flag that enforces zero detection in spectrum slicing.
<i>getSTOARDimensions()</i>	Get the dimensions used for each subsolve step.
<i>getSTOARInertias()</i>	Get the values of the shifts and their corresponding inertias.
<i>getSTOARLinearization()</i>	Get the coefficients that define the linearization of a quadratic eigenproblem.
<i>getSTOARLocking()</i>	Get the locking flag used in the STOAR method.
<i>getScale([DI, Dr])</i>	Get the strategy used for scaling the polynomial eigenproblem.
<i>getStoppingTest()</i>	Get the stopping function.
<i>getTOARLocking()</i>	Get the locking flag used in the TOAR method.
<i>getTOARRestart()</i>	Get the restart parameter used in the TOAR method.
<i>getTarget()</i>	Get the value of the target.
<i>getTolerances()</i>	Get the tolerance and maximum iteration count.
<i>getTrackAll()</i>	Get the flag indicating whether all residual norms must be computed.
<i>getType()</i>	Get the PEP type of this object.
<i>getWhichEigenpairs()</i>	Get which portion of the spectrum is to be sought.
<i>reset()</i>	Reset the PEP object.
<i>setBV(bv)</i>	Set a basis vectors object associated to the eigensolver.
<i>setBasis(basis)</i>	Set the type of polynomial basis used.
<i>setCISSExtraction(extraction)</i>	Set the extraction technique used in the CISS solver.
<i>setCISSRefinement([inner, blsize])</i>	Set the values of various refinement parameters in the CISS solver.
<i>setCISSSizes([ip, bs, ms, npart, bsmax, ...])</i>	Set the values of various size parameters in the CISS solver.
<i>setCISSThreshold([delta, spur])</i>	Set the values of various threshold parameters in the CISS solver.
<i>setConvergenceTest(conv)</i>	Set how to compute the error estimate used in the convergence test.
<i>setDS(ds)</i>	Set a direct solver object associated to the eigensolver.
<i>setDimensions([nev, ncv, mpd])</i>	Set the number of eigenvalues to compute and the dimension of the subspace.
<i>setExtract(extract)</i>	Set the extraction strategy to be used.
<i>setFromOptions()</i>	Set PEP options from the options database.
<i>setInitialSpace(space)</i>	Set the initial space from which the eigensolver starts to iterate.
<i>setInterval(inta, intb)</i>	Set the computational interval for spectrum slicing.
<i>setJDFix(fix)</i>	Set the threshold for changing the target in the correction equation.
<i>setJDMinimalityIndex(flag)</i>	Set the maximum allowed value for the minimality index.
<i>setJDProjection(proj)</i>	Set the type of projection to be used in the Jacobi-Davidson solver.
<i>setJDRestart(keep)</i>	Set the restart parameter for the Jacobi-Davidson method.
<i>setJDReusePreconditioner(flag)</i>	Set a flag indicating whether the preconditioner must be reused or not.

continues on next page

Table 84 – continued from previous page

<code>setLinearEPS(eps)</code>	Set an eigensolver object associated to the polynomial eigenvalue solver.
<code>setLinearExplicitMatrix(flag)</code>	Set flag to explicitly build the matrices A and B.
<code>setLinearLinearization([alpha, beta])</code>	Set the coefficients that define the linearization of a quadratic eigenproblem.
<code>setMonitor(monitor[, args, kargs])</code>	Append a monitor function to the list of monitors.
<code>setOperators(operators)</code>	Set the matrices associated with the eigenvalue problem.
<code>setOptionsPrefix([prefix])</code>	Set the prefix used for searching for all PEP options in the database.
<code>setProblemType(problem_type)</code>	Set the type of the eigenvalue problem.
<code>setQArnoldiLocking(lock)</code>	Toggle between locking and non-locking variants of the Q-Arnoldi method.
<code>setQArnoldiRestart(keep)</code>	Set the restart parameter for the Q-Arnoldi method.
<code>setRG(rg)</code>	Set a region object associated to the eigensolver.
<code>setRefine(ref[, npart, tol, its, scheme])</code>	Set the refinement strategy used by the PEP object.
<code>setST(st)</code>	Set a spectral transformation object associated to the eigensolver.
<code>setSTOARCheckEigenvalueType(flag)</code>	Set flag to check if all eigenvalues have the same definite type.
<code>setSTOARDetectZeros(detect)</code>	Set flag to enforce detection of zeros during the factorizations.
<code>setSTOARDimensions([nev, ncv, mpd])</code>	Set the dimensions used for each subsolve step.
<code>setSTOARLinearization([alpha, beta])</code>	Set the coefficients that define the linearization of a quadratic eigenproblem.
<code>setSTOARLocking(lock)</code>	Toggle between locking and non-locking variants of the STOAR method.
<code>setScale(scale[, alpha, DI, Dr, its, lbda])</code>	Set the scaling strategy to be used.
<code>setStoppingTest(stopping[, args, kargs])</code>	Set a function to decide when to stop the outer iteration of the eigensolver.
<code>setTOARLocking(lock)</code>	Toggle between locking and non-locking variants of the TOAR method.
<code>setTOARRestart(keep)</code>	Set the restart parameter for the TOAR method.
<code>setTarget(target)</code>	Set the value of the target.
<code>setTolerances([tol, max_it])</code>	Set the tolerance and maximum iteration count.
<code>setTrackAll(trackall)</code>	Set flag to compute the residual of all approximate eigenpairs.
<code>setType(pep_type)</code>	Set the particular solver to be used in the PEP object.
<code>setUp()</code>	Set up all the necessary internal data structures.
<code>setWhichEigenpairs(which)</code>	Set which portion of the spectrum is to be sought.
<code>solve()</code>	Solve the eigensystem.
<code>valuesView([viewer])</code>	Display the computed eigenvalues in a viewer.
<code>vectorsView([viewer])</code>	Output computed eigenvectors to a viewer.
<code>view([viewer])</code>	Print the PEP data structure.

Attributes Summary

<code>bv</code>	The basis vectors (BV) object associated.
<code>ds</code>	The direct solver (DS) object associated.
<code>extract</code>	The type of extraction technique to be employed.
<code>max_it</code>	The maximum iteration count.

continues on next page

Table 85 – continued from previous page

<code>problem_type</code>	The type of the eigenvalue problem.
<code>rg</code>	The region (RG) object associated.
<code>st</code>	The spectral transformation (ST) object associated.
<code>target</code>	The value of the target.
<code>tol</code>	The tolerance.
<code>track_all</code>	Compute the residual norm of all approximate eigenpairs.
<code>which</code>	The portion of the spectrum to be sought.

Methods Documentation

appendOptionsPrefix(*prefix=None*)

Append to the prefix used for searching for all PEP options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all PEP option requests.

Return type

None

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:353 <slepc4py/SLEPc/PEP.pyx#L353>](#)

cancelMonitor()

Clear all monitors for a *PEP* object.

Logically collective.

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1162 <slepc4py/SLEPc/PEP.pyx#L1162>](#)

Return type

None

computeError(*i, etype=None*)

Compute the error associated with the *i*-th computed eigenpair.

Collective.

Compute the error (based on the residual norm) associated with the *i*-th computed eigenpair.

Parameters

- **i** (*int*) – Index of the solution to be considered.
- **etype** (*ErrorType* / *None*) – The error type to compute.

Returns

The error bound, computed in various ways from the residual norm $\|P(l)x\|_2$ where l is the eigenvalue and x is the eigenvector.

Return type

float

Notes

The index *i* should be a value between 0 and `nconv-1` (see `getConverged()`).

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1290 <slepc4py/SLEPc/PEP.pyx#L1290>](#)

create(*comm=None*)

Create the PEP object.

Collective.

Parameters

comm (*Comm* / *None*) – MPI communicator. If not provided, it defaults to all processes.

Return type

Self

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:276 <slepc4py/SLEPc/PEP.pyx#L276>](#)

destroy()

Destroy the PEP object.

Collective.

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:258 <slepc4py/SLEPc/PEP.pyx#L258>](#)

Return type

Self

errorView(*etype=None, viewer=None*)

Display the errors associated with the computed solution.

Collective.

Display the errors and the eigenvalues.

Parameters

- **etype** (*ErrorType* / *None*) – The error type to compute.
- **viewer** (*petsc4py.PETSc.Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

Notes

By default, this function checks the error of all eigenpairs and prints the eigenvalues if all of them are below the requested tolerance. If the viewer has format ASCII_INFO_DETAIL then a table with eigenvalues and corresponding errors is printed.

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1324 <slepc4py/SLEPc/PEP.pyx#L1324>](#)

getBV()

Get the basis vectors object associated to the eigensolver.

Not collective.

Returns

The basis vectors context.

Return type

BV

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:954 <slepc4py/SLEPc/PEP.pyx#L954>](#)

getBasis()

Get the type of polynomial basis used.

Not collective.

Get the type of polynomial basis used to describe the polynomial eigenvalue problem.

Returns

The basis that was previously set.

Return type

Basis

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:379 <slepc4py/SLEPc/PEP.pyx#L379>`

getCISSExtraction()

Get the extraction technique used in the CISS solver.

Not collective.

Returns

The extraction technique.

Return type

CISSExtraction

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2049 <slepc4py/SLEPc/PEP.pyx#L2049>`

getCISSKSPs()

Get the array of linear solver objects associated with the CISS solver.

Collective.

Returns

The linear solver objects.

Return type

list of petsc4py.PETSc.KSP

Notes

The number of `petsc4py.PETSc.KSP` solvers is equal to the number of integration points divided by the number of partitions. This value is halved in the case of real matrices with a region centered at the real axis.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2218 <slepc4py/SLEPc/PEP.pyx#L2218>`

getCISSRefinement()

Get the values of various refinement parameters in the CISS solver.

Not collective.

Returns

- **inner** (*int*) – Number of iterative refinement iterations (inner loop).
- **blsize** (*int*) – Number of iterative refinement iterations (blocksize loop).

Return type

tuple[int, int]

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2200 <slepc4py/SLEPc/PEP.pyx#L2200>`

getCISSSizes()

Get the values of various size parameters in the CISS solver.

Not collective.

Returns

- **ip** ([int](#)) – Number of integration points.
- **bs** ([int](#)) – Block size.
- **ms** ([int](#)) – Moment size.
- **npart** ([int](#)) – Number of partitions when splitting the communicator.
- **bsmax** ([int](#)) – Maximum block size.
- **realmats** ([bool](#)) – True if A and B are real.

Return type

[tuple](#)[[int](#), [int](#), [int](#), [int](#), [int](#), [bool](#)]

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:2114 <slepc4py/SLEPc/PEP.pyx#L2114>](#)

getCISSThreshold()

Get the values of various threshold parameters in the CISS solver.

Not collective.

Returns

- **delta** ([float](#)) – Threshold for numerical rank.
- **spur** ([float](#)) – Spurious threshold (to discard spurious eigenpairs).

Return type

[tuple](#)[[float](#), [float](#)]

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:2163 <slepc4py/SLEPc/PEP.pyx#L2163>](#)

getConverged()

Get the number of converged eigenpairs.

Not collective.

Returns

Number of converged eigenpairs.

Return type

[int](#)

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1226 <slepc4py/SLEPc/PEP.pyx#L1226>](#)

getConvergedReason()

Get the reason why the [solve\(\)](#) iteration was stopped.

Not collective.

Returns

Negative value indicates diverged, positive value converged.

Return type

[ConvergedReason](#)

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1210 <slepc4py/SLEPc/PEP.pyx#L1210>](#)

getConvergenceTest()

Get the method used to compute the error estimate used in the convergence test.

Not collective.

Returns

The method used to compute the error estimate used in the convergence test.

Return type

Conv

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:601 <slepc4py/SLEPc/PEP.pyx#L601>`

getDS()

Get the direct solver associated to the eigensolver.

Not collective.

Returns

The direct solver context.

Return type

DS

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1012 <slepc4py/SLEPc/PEP.pyx#L1012>`

getDimensions()

Get the number of eigenvalues to compute and the dimension of the subspace.

Not collective.

Returns

- **nev** (*int*) – Number of eigenvalues to compute.
- **ncv** (*int*) – Maximum dimension of the subspace to be used by the solver.
- **mpd** (*int*) – Maximum dimension allowed for the projected problem.

Return type

tuple[*int*, *int*, *int*]

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:779 <slepc4py/SLEPc/PEP.pyx#L779>`

getEigenpair(*i*, *Vr*=None, *Vi*=None)

Get the *i*-th solution of the eigenproblem as computed by *solve()*.

Collective.

The solution consists of both the eigenvalue and the eigenvector.

Parameters

- **i** (*int*) – Index of the solution to be obtained.
- **Vr** (*Vec* | *None*) – Placeholder for the returned eigenvector (real part).
- **Vi** (*Vec* | *None*) – Placeholder for the returned eigenvector (imaginary part).

Returns

The computed eigenvalue.

Return type

complex

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1241 <slepc4py/SLEPc/PEP.pyx#L1241>`

getErrorEstimate(*i*)

Get the error estimate associated to the *i*-th computed eigenpair.

Not collective.

Parameters

i (*int*) – Index of the solution to be considered.

Returns

Error estimate.

Return type

float

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1270 <slepc4py/SLEPc/PEP.pyx#L1270>`

getExtract()

Get the extraction technique used by the *PEP* object.

Not collective.

Returns

The extraction strategy.

Return type

Extract

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:732 <slepc4py/SLEPc/PEP.pyx#L732>`

getInterval()

Get the computational interval for spectrum slicing.

Not collective.

Returns

- **inta** (*float*) – The left end of the interval.
- **intb** (*float*) – The right end of the interval.

Return type

tuple[*float*, *float*]

Notes

If the interval was not set by the user, then zeros are returned.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:554 <slepc4py/SLEPc/PEP.pyx#L554>`

getIterationNumber()

Get the current iteration number.

Not collective.

If the call to *solve()* is complete, then it returns the number of iterations carried out by the solution method.

Returns

Iteration number.

Return type

int

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1192 <slepc4py/SLEPc/PEP.pyx#L1192>`

getJDFix()

Get threshold for changing the target in the correction equation.

Not collective.

Returns

The threshold for changing the target.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1931 <slepc4py/SLEPc/PEP.pyx#L1931>`

getJDMINimalityIndex()

Get the maximum allowed value of the minimality index.

Not collective.

Returns

The maximum minimality index.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1989 <slepc4py/SLEPc/PEP.pyx#L1989>`

getJDProjection()

Get the type of projection to be used in the Jacobi-Davidson solver.

Not collective.

Returns

The type of projection.

Return type

`JDProjection`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2018 <slepc4py/SLEPc/PEP.pyx#L2018>`

getJDRestart()

Get the restart parameter used in the Jacobi-Davidson method.

Not collective.

Returns

The number of vectors to be kept at restart.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1896 <slepc4py/SLEPc/PEP.pyx#L1896>`

getJDReusePreconditioner()

Get the flag for reusing the preconditioner.

Not collective.

Returns

The reuse flag.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1960 <slepc4py/SLEPc/PEP.pyx#L1960>`

getLinearEPS()

Get the eigensolver object associated to the polynomial eigenvalue solver.

Collective.

Returns

The linear eigensolver.

Return type

EPS

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1398 <slepc4py/SLEPc/PEP.pyx#L1398>`

getLinearExplicitMatrix()

Get if the matrices A and B for the linearization are built explicitly.

Not collective.

Get the flag indicating if the matrices A and B for the linearization are built explicitly.

Returns

Boolean flag indicating if the matrices are built explicitly.

Return type

bool

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1472 <slepc4py/SLEPc/PEP.pyx#L1472>`

getLinearLinearization()

Get the coeffs. defining the linearization of a quadratic eigenproblem.

Not collective.

Return the coefficients that define the linearization of a quadratic eigenproblem.

Returns

- **alpha** (*float*) – First parameter of the linearization.
- **beta** (*float*) – Second parameter of the linearization.

Return type

*tuple[*float*, *float*]*

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1434 <slepc4py/SLEPc/PEP.pyx#L1434>`

getMonitor()

Get the list of monitor functions.

Not collective.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1154 <slepc4py/SLEPc/PEP.pyx#L1154>`

Return type

PEPMonitorFunction

getOperators()

Get the matrices associated with the eigenvalue problem.

Collective.

Returns

The matrices associated with the eigensystem.

Return type

*list of *petsc4py.PETSc.Mat**

:sources:`Source code at slepc4py/SLEPc/PEP.pyx:1041 <slepc4py/SLEPc/PEP.pyx#L1041>`

getOptionsPrefix()

Get the prefix used for searching for all PEP options in the database.

Not collective.

Returns

The prefix string set for this PEP object.

Return type

`str`

:sources:`Source code at slepc4py/SLEPc/PEP.pyx:323 <slepc4py/SLEPc/PEP.pyx#L323>`

getProblemType()

Get the problem type from the PEP object.

Not collective.

Returns

The problem type that was previously set.

Return type

`ProblemType`

:sources:`Source code at slepc4py/SLEPc/PEP.pyx:414 <slepc4py/SLEPc/PEP.pyx#L414>`

getQArnoldiLocking()

Get the locking flag used in the Q-Arnoldi method.

Not collective.

Returns

The locking flag.

Return type

`bool`

:sources:`Source code at slepc4py/SLEPc/PEP.pyx:1550 <slepc4py/SLEPc/PEP.pyx#L1550>`

getQArnoldiRestart()

Get the restart parameter used in the Q-Arnoldi method.

Not collective.

Returns

The number of vectors to be kept at restart.

Return type

`float`

:sources:`Source code at slepc4py/SLEPc/PEP.pyx:1514 <slepc4py/SLEPc/PEP.pyx#L1514>`

getRG()

Get the region object associated to the eigensolver.

Not collective.

Returns

The region context.

Return type

`RG`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:983 <slepc4py/SLEPc/PEP.pyx#L983>`

getRefine()

Get the refinement strategy used by the PEP object.

Not collective.

Get the refinement strategy used by the PEP object, and the associated parameters.

Returns

- **ref** (*Refine*) – The refinement type.
- **npart** (*int*) – The number of partitions of the communicator.
- **tol** (*float*) – The convergence tolerance.
- **its** (*int*) – The maximum number of refinement iterations.
- **scheme** (*RefineScheme*) – Scheme for solving linear systems

Return type

tuple[*Refine*, *int*, *float*, *int*, *RefineScheme*]

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:632 <slepc4py/SLEPc/PEP.pyx#L632>`

getRefineKSP()

Get the KSP object used by the eigensolver in the refinement phase.

Collective.

Returns

The linear solver object.

Return type

petsc4py.PETSc.KSP

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:702 <slepc4py/SLEPc/PEP.pyx#L702>`

getST()

Get the *ST* object associated to the eigensolver object.

Not collective.

Get the spectral transformation object associated to the eigensolver object.

Returns

The spectral transformation.

Return type

ST

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:828 <slepc4py/SLEPc/PEP.pyx#L828>`

getSTOARCheckEigenvalueType()

Get the flag for the eigenvalue type check in spectrum slicing.

Not collective.

Returns

Whether the eigenvalue type is checked or not.

Return type

bool

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1857 <slepc4py/SLEPc/PEP.pyx#L1857>`

getSTOARDetectZeros()

Get the flag that enforces zero detection in spectrum slicing.

Not collective.

Returns

The zero detection flag.

Return type

`bool`

:sources: ``Source code at slepc4py/SLEPc/PEP.pyx:1739 <slepc4py/SLEPc/PEP.pyx#L1739>``

getSTOARDimensions()

Get the dimensions used for each subsolve step.

Not collective.

Get the dimensions used for each subsolve step in case of doing spectrum slicing for a computational interval.

Returns

- **nev** (`int`) – Number of eigenvalues to compute.
- **ncv** (`int`) – Maximum dimension of the subspace to be used by the solver.
- **mpd** (`int`) – Maximum dimension allowed for the projected problem.

Return type

`tuple[int, int, int]`

:sources: ``Source code at slepc4py/SLEPc/PEP.pyx:1786 <slepc4py/SLEPc/PEP.pyx#L1786>``

getSTOARInertias()

Get the values of the shifts and their corresponding inertias.

Not collective.

Get the values of the shifts and their corresponding inertias in case of doing spectrum slicing for a computational interval.

Returns

- **shifts** (`ArrayReal`) – The values of the shifts used internally in the solver.
- **inertias** (`ArrayInt`) – The values of the inertia in each shift.

Return type

`tuple[ArrayReal, ArrayInt]`

:sources: ``Source code at slepc4py/SLEPc/PEP.pyx:1810 <slepc4py/SLEPc/PEP.pyx#L1810>``

getSTOARLinearization()

Get the coefficients that define the linearization of a quadratic eigenproblem.

Not collective.

Returns

- **alpha** (`float`) – First parameter of the linearization.
- **beta** (`float`) – Second parameter of the linearization.

Return type

`tuple[float, float]`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1659 <slepc4py/SLEPc/PEP.pyx#L1659>`

getSTOARLocking()

Get the locking flag used in the STOAR method.

Not collective.

Returns

The locking flag.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1698 <slepc4py/SLEPc/PEP.pyx#L1698>`

getScale(Dl=None, Dr=None)

Get the strategy used for scaling the polynomial eigenproblem.

Not collective.

Parameters

- **Dl** (`petsc4py.PETSc.Vec` | `None`) – Placeholder for the returned left diagonal matrix.
- **Dr** (`petsc4py.PETSc.Vec` | `None`) – Placeholder for the returned right diagonal matrix.

Returns

- **scale** (`Scale`) – The scaling strategy.
- **alpha** (`float`) – The scaling factor.
- **its** (`int`) – The number of iteration of diagonal scaling.
- **lbda** (`float`) – Approximation of the wanted eigenvalues (modulus).

Return type

`tuple[Scale, float, int, float]`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:860 <slepc4py/SLEPc/PEP.pyx#L860>`

getStoppingTest()

Get the stopping function.

Not collective.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1123 <slepc4py/SLEPc/PEP.pyx#L1123>`

Return type

`PEPStoppingFunction`

getTOARLocking()

Get the locking flag used in the TOAR method.

Not collective.

Returns

The locking flag.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1625 <slepc4py/SLEPc/PEP.pyx#L1625>`

getTOARRestart()

Get the restart parameter used in the TOAR method.

Not collective.

Returns

The number of vectors to be kept at restart.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1589 <slepc4py/SLEPc/PEP.pyx#L1589>`

getTarget()

Get the value of the target.

Not collective.

Returns

The value of the target.

Return type

`Scalar`

Notes

If the target was not set by the user, then zero is returned.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:472 <slepc4py/SLEPc/PEP.pyx#L472>`

getTolerances()

Get the tolerance and maximum iteration count.

Not collective.

Get the tolerance and maximum iteration count used by the default PEP convergence tests.

Returns

- `tol (float)` – The convergence tolerance.
- `max_it (int)` – The maximum number of iterations

Return type

`tuple[float, int]`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:511 <slepc4py/SLEPc/PEP.pyx#L511>`

getTrackAll()

Get the flag indicating whether all residual norms must be computed.

Not collective.

Returns

Whether the solver compute all residuals or not.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:747 <slepc4py/SLEPc/PEP.pyx#L747>`

getType()

Get the PEP type of this object.

Not collective.

Returns

The solver currently being used.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:308 <slepc4py/SLEPc/PEP.pyx#L308>`

getWhichEigenpairs()

Get which portion of the spectrum is to be sought.

Not collective.

Returns

The portion of the spectrum to be sought by the solver.

Return type

`Which`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:443 <slepc4py/SLEPc/PEP.pyx#L443>`

reset()

Reset the PEP object.

Collective.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:268 <slepc4py/SLEPc/PEP.pyx#L268>`

Return type

`None`

setBV(*bv*)

Set a basis vectors object associated to the eigensolver.

Collective.

Parameters

bv (`BV`) – The basis vectors context.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:970 <slepc4py/SLEPc/PEP.pyx#L970>`

setBasis(*basis*)

Set the type of polynomial basis used.

Logically collective.

Set the type of polynomial basis used to describe the polynomial eigenvalue problem.

Parameters

basis (`Basis`) – The basis to be set.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:397 <slepc4py/SLEPc/PEP.pyx#L397>`

setCISSExtraction(*extraction*)

Set the extraction technique used in the CISS solver.

Logically collective.

Parameters

extraction ([CISSExtraction](#)) – The extraction technique.

Return type

[None](#)

:sources: ``Source code at slepc4py/SLEPc/PEP.pyx:2035 <slepc4py/SLEPc/PEP.pyx#L2035>``

setCISSRefinement (*inner=None, blsize=None*)

Set the values of various refinement parameters in the CISS solver.

Logically collective.

Parameters

- **inner** ([int](#) / [None](#)) – Number of iterative refinement iterations (inner loop).
- **blsize** ([int](#) / [None](#)) – Number of iterative refinement iterations (blocksize loop).

Return type

[None](#)

:sources: ``Source code at slepc4py/SLEPc/PEP.pyx:2181 <slepc4py/SLEPc/PEP.pyx#L2181>``

setCISSSizes (*ip=None, bs=None, ms=None, npart=None, bsmax=None, realmats=False*)

Set the values of various size parameters in the CISS solver.

Logically collective.

Parameters

- **ip** ([int](#) / [None](#)) – Number of integration points.
- **bs** ([int](#) / [None](#)) – Block size.
- **ms** ([int](#) / [None](#)) – Moment size.
- **npart** ([int](#) / [None](#)) – Number of partitions when splitting the communicator.
- **bsmax** ([int](#) / [None](#)) – Maximum block size.
- **realmats** ([bool](#)) – True if A and B are real.

Return type

[None](#)

Notes

The default number of partitions is 1. This means the internal `petsc4py.PETSc.KSP` object is shared among all processes of the `PEP` communicator. Otherwise, the communicator is split into `npart` communicators, so that `npart` `petsc4py.PETSc.KSP` solves proceed simultaneously.

:sources: ``Source code at slepc4py/SLEPc/PEP.pyx:2064 <slepc4py/SLEPc/PEP.pyx#L2064>``

setCISSThreshold (*delta=None, spur=None*)

Set the values of various threshold parameters in the CISS solver.

Logically collective.

Parameters

- **delta** ([float](#) / [None](#)) – Threshold for numerical rank.
- **spur** ([float](#) / [None](#)) – Spurious threshold (to discard spurious eigenpairs).

Return type

[None](#)

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2144 <slepc4py/SLEPc/PEP.pyx#L2144>`

setConvergenceTest(*conv*)

Set how to compute the error estimate used in the convergence test.

Logically collective.

Parameters

conv (*Conv*) – The method used to compute the error estimate used in the convergence test.

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:617 <slepc4py/SLEPc/PEP.pyx#L617>`

setDS(*ds*)

Set a direct solver object associated to the eigensolver.

Collective.

Parameters

ds (*DS*) – The direct solver context.

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1028 <slepc4py/SLEPc/PEP.pyx#L1028>`

setDimensions(*nev=None, ncw=None, mpd=None*)

Set the number of eigenvalues to compute and the dimension of the subspace.

Logically collective.

Parameters

- **nev** (*int* / *None*) – Number of eigenvalues to compute.
- **ncw** (*int* / *None*) – Maximum dimension of the subspace to be used by the solver.
- **mpd** (*int* / *None*) – Maximum dimension allowed for the projected problem.

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:800 <slepc4py/SLEPc/PEP.pyx#L800>`

setExtract(*extract*)

Set the extraction strategy to be used.

Logically collective.

Parameters

extract (*Extract*) – The extraction strategy.

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:718 <slepc4py/SLEPc/PEP.pyx#L718>`

setFromOptions()

Set PEP options from the options database.

Collective.

This routine must be called before `setUp()` if the user is to be allowed to set the solver type.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:368 <slepc4py/SLEPc/PEP.pyx#L368>`

Return type

None

setInitialSpace(*space*)

Set the initial space from which the eigensolver starts to iterate.

Collective.

Parameters

space (*Vec* | *list*[*Vec*]) – The initial space

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1083 <slepc4py/SLEPc/PEP.pyx#L1083>`

setInterval(*inta*, *intb*)

Set the computational interval for spectrum slicing.

Logically collective.

Parameters

- **inta** (*float*) – The left end of the interval.
- **intb** (*float*) – The right end of the interval.

Return type

None

Notes

Spectrum slicing is a technique employed for computing all eigenvalues of symmetric quadratic eigenproblems in a given interval. This function provides the interval to be considered. It must be used in combination with *PEP.Which.ALL*, see *setWhichEigenpairs()*.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:576 <slepc4py/SLEPc/PEP.pyx#L576>`

setJDFix(*fix*)

Set the threshold for changing the target in the correction equation.

Logically collective.

Parameters

fix (*float*) – Threshold for changing the target.

Return type

None

Notes

The target in the correction equation is fixed at the first iterations. When the norm of the residual vector is lower than the fix value, the target is set to the corresponding eigenvalue.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1911 <slepc4py/SLEPc/PEP.pyx#L1911>`

setJDMINimalityIndex(*flag*)

Set the maximum allowed value for the minimality index.

Logically collective.

Parameters

flag (*int*) – The maximum minimality index.

Return type

None

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1975 <slepc4py/SLEPc/PEP.pyx#L1975>](#)

setJDProjection(*proj*)

Set the type of projection to be used in the Jacobi-Davidson solver.

Logically collective.

Parameters

proj (*JDProjection*) – The type of projection.

Return type

None

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:2004 <slepc4py/SLEPc/PEP.pyx#L2004>](#)

setJDRestart(*keep*)

Set the restart parameter for the Jacobi-Davidson method.

Logically collective.

Set the restart parameter for the Jacobi-Davidson method, in particular the proportion of basis vectors that must be kept after restart.

Parameters

keep (*float*) – The number of vectors to be kept at restart.

Return type

None

Notes

Allowed values are in the range [0.1,0.9]. The default is 0.5.

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1874 <slepc4py/SLEPc/PEP.pyx#L1874>](#)

setJDReusePreconditioner(*flag*)

Set a flag indicating whether the preconditioner must be reused or not.

Logically collective.

Parameters

flag (*bool*) – The reuse flag.

Return type

None

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1946 <slepc4py/SLEPc/PEP.pyx#L1946>](#)

setLinearEPS(*eps*)

Set an eigensolver object associated to the polynomial eigenvalue solver.

Collective.

Parameters

eps (*EPS*) – The linear eigensolver.

Return type

None

:sources:`Source code at slepc4py/SLEPc/PEP.pyx:1385 <slepc4py/SLEPc/PEP.pyx#L1385>`

setLinearExplicitMatrix(*flag*)

Set flag to explicitly build the matrices A and B.

Logically collective.

Toggle if the matrices A and B for the linearization of the problem must be built explicitly.

Parameters

flag (*bool*) – Boolean flag indicating if the matrices are built explicitly.

Return type

None

:sources:`Source code at slepc4py/SLEPc/PEP.pyx:1455 <slepc4py/SLEPc/PEP.pyx#L1455>`

setLinearLinearization(*alpha=1.0, beta=0.0*)

Set the coefficients that define the linearization of a quadratic eigenproblem.

Logically collective.

Set the coefficients that define the linearization of a quadratic eigenproblem.

Parameters

- **alpha** (*float*) – First parameter of the linearization.
- **beta** (*float*) – Second parameter of the linearization.

Return type

None

:sources:`Source code at slepc4py/SLEPc/PEP.pyx:1414 <slepc4py/SLEPc/PEP.pyx#L1414>`

setMonitor(*monitor, args=None, kargs=None*)

Append a monitor function to the list of monitors.

Logically collective.

:sources:`Source code at slepc4py/SLEPc/PEP.pyx:1133 <slepc4py/SLEPc/PEP.pyx#L1133>`

Parameters

- **monitor** (*PEPMonitorFunction* | *None*)
- **args** (*tuple*[*Any*, ...] | *None*)
- **kargs** (*dict*[*str*, *Any*] | *None*)

Return type

None

setOperators(*operators*)

Set the matrices associated with the eigenvalue problem.

Collective.

Parameters

operators (*list*[*Mat*]) – The matrices associated with the eigensystem.

Return type

None

:sources:`Source code at slepc4py/SLEPc/PEP.pyx:1063 <slepc4py/SLEPc/PEP.pyx#L1063>`

setOptionsPrefix(*prefix=None*)

Set the prefix used for searching for all PEP options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all PEP option requests.

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:338 <slepc4py/SLEPc/PEP.pyx#L338>`

setProblemType(*problem_type*)

Set the type of the eigenvalue problem.

Logically collective.

Parameters

problem_type (*ProblemType*) – The problem type to be set.

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:429 <slepc4py/SLEPc/PEP.pyx#L429>`

setQArnoldiLocking(*lock*)

Toggle between locking and non-locking variants of the Q-Arnoldi method.

Logically collective.

Parameters

lock (*bool*) – True if the locking variant must be selected.

Return type

None

Notes

The default is to lock converged eigenpairs when the method restarts. This behavior can be changed so that all directions are kept in the working subspace even if already converged to working accuracy (the non-locking variant).

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1529 <slepc4py/SLEPc/PEP.pyx#L1529>`

setQArnoldiRestart(*keep*)

Set the restart parameter for the Q-Arnoldi method.

Logically collective.

Set the restart parameter for the Q-Arnoldi method, in particular the proportion of basis vectors that must be kept after restart.

Parameters

keep (*float*) – The number of vectors to be kept at restart.

Return type

None

Notes

Allowed values are in the range [0.1,0.9]. The default is 0.5.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1492 <slepc4py/SLEPc/PEP.pyx#L1492>`

setRG(*rg*)

Set a region object associated to the eigensolver.

Collective.

Parameters

rg (*RG*) – The region context.

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:999 <slepc4py/SLEPc/PEP.pyx#L999>`

setRefine(*ref*, *npart*=*None*, *tol*=*None*, *its*=*None*, *scheme*=*None*)

Set the refinement strategy used by the PEP object.

Logically collective.

Set the refinement strategy used by the PEP object, and the associated parameters.

Parameters

- **ref** (*Refine*) – The refinement type.
- **npart** (*int* / *None*) – The number of partitions of the communicator.
- **tol** (*float* / *None*) – The convergence tolerance.
- **its** (*int* / *None*) – The maximum number of refinement iterations.
- **scheme** (*RefineScheme* / *None*) – Scheme for linear system solves

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:662 <slepc4py/SLEPc/PEP.pyx#L662>`

setST(*st*)

Set a spectral transformation object associated to the eigensolver.

Collective.

Parameters

st (*ST*) – The spectral transformation.

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:847 <slepc4py/SLEPc/PEP.pyx#L847>`

setSTOARCheckEigenvalueType(*flag*)

Set flag to check if all eigenvalues have the same definite type.

Logically collective.

Set a flag to check that all the eigenvalues obtained throughout the spectrum slicing computation have the same definite type.

Parameters

flag (*bool*) – Whether the eigenvalue type is checked or not.

Return type

None

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1840 <slepc4py/SLEPc/PEP.pyx#L1840>](#)

setSTOARDetectZeros(*detect*)

Set flag to enforce detection of zeros during the factorizations.

Logically collective.

Set a flag to enforce detection of zeros during the factorizations throughout the spectrum slicing computation.

Parameters

detect (*bool*) – True if zeros must checked for.

Return type

None

Notes

A zero in the factorization indicates that a shift coincides with an eigenvalue.

This flag is turned off by default, and may be necessary in some cases. This feature currently requires an external package for factorizations with support for zero detection, e.g. MUMPS.

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1713 <slepc4py/SLEPc/PEP.pyx#L1713>](#)

setSTOARDimensions(*nev=None, ncv=None, mpd=None*)

Set the dimensions used for each subsolve step.

Logically collective.

Set the dimensions used for each subsolve step in case of doing spectrum slicing for a computational interval. The meaning of the parameters is the same as in [setDimensions\(\)](#).

Parameters

- **nev** (*int* | *None*) – Number of eigenvalues to compute.
- **ncv** (*int* | *None*) – Maximum dimension of the subspace to be used by the solver.
- **mpd** (*int* | *None*) – Maximum dimension allowed for the projected problem.

Return type

None

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1754 <slepc4py/SLEPc/PEP.pyx#L1754>](#)

setSTOARLinearization(*alpha=1.0, beta=0.0*)

Set the coefficients that define the linearization of a quadratic eigenproblem.

Logically collective.

Parameters

- **alpha** (*float*) – First parameter of the linearization.
- **beta** (*float*) – Second parameter of the linearization.

Return type

None

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1642 <slepc4py/SLEPc/PEP.pyx#L1642>](#)

setSTOARLocking(*lock*)

Toggle between locking and non-locking variants of the STOAR method.

Logically collective.

Parameters

lock (*bool*) – True if the locking variant must be selected.

Return type

None

Notes

The default is to lock converged eigenpairs when the method restarts. This behavior can be changed so that all directions are kept in the working subspace even if already converged to working accuracy (the non-locking variant).

:sources: ``Source code at slepc4py/SLEPc/PEP.pyx:1677 <slepc4py/SLEPc/PEP.pyx#L1677>``

setScale(*scale*, *alpha*=*None*, *Dl*=*None*, *Dr*=*None*, *its*=*None*, *lbda*=*None*)

Set the scaling strategy to be used.

Collective.

Set the scaling strategy to be used for scaling the polynomial problem before attempting to solve.

Parameters

- **scale** (*Scale*) – The scaling strategy.
- **alpha** (*float* | *None*) – The scaling factor.
- **Dl** (*petsc4py.PETSc.Vec* | *None*) – The left diagonal matrix.
- **Dr** (*petsc4py.PETSc.Vec* | *None*) – The right diagonal matrix.
- **its** (*int* | *None*) – The number of iteration of diagonal scaling.
- **lbda** (*float* | *None*) – Approximation of the wanted eigenvalues (modulus).

Return type

None

:sources: ``Source code at slepc4py/SLEPc/PEP.pyx:909 <slepc4py/SLEPc/PEP.pyx#L909>``

setStoppingTest(*stopping*, *args*=*None*, *kargs*=*None*)

Set a function to decide when to stop the outer iteration of the eigensolver.

Logically collective.

:sources: ``Source code at slepc4py/SLEPc/PEP.pyx:1103 <slepc4py/SLEPc/PEP.pyx#L1103>``

Parameters

- **stopping** (*PEPStoppingFunction* | *None*)
- **args** (*tuple*[*Any*, ...] | *None*)
- **kargs** (*dict*[*str*, *Any*] | *None*)

Return type

None

setTOARLocking(*lock*)

Toggle between locking and non-locking variants of the TOAR method.

Logically collective.

Parameters

lock (*bool*) – True if the locking variant must be selected.

Return type

None

Notes

The default is to lock converged eigenpairs when the method restarts. This behavior can be changed so that all directions are kept in the working subspace even if already converged to working accuracy (the non-locking variant).

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1604 <slepc4py/SLEPc/PEP.pyx#L1604>](#)

setTOARRestart(*keep*)

Set the restart parameter for the TOAR method.

Logically collective.

Set the restart parameter for the TOAR method, in particular the proportion of basis vectors that must be kept after restart.

Parameters

keep (*float*) – The number of vectors to be kept at restart.

Return type

None

Notes

Allowed values are in the range [0.1,0.9]. The default is 0.5.

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:1567 <slepc4py/SLEPc/PEP.pyx#L1567>](#)

setTarget(*target*)

Set the value of the target.

Logically collective.

Parameters

target (*Scalar*) – The value of the target.

Return type

None

Notes

The target is a scalar value used to determine the portion of the spectrum of interest. It is used in combination with [setWhichEigenpairs\(\)](#).

:sources: [Source code at slepc4py/SLEPc/PEP.pyx:491 <slepc4py/SLEPc/PEP.pyx#L491>](#)

setTolerances(*tol=None, max_it=None*)

Set the tolerance and maximum iteration count.

Logically collective.

Set the tolerance and maximum iteration count used by the default PEP convergence tests.

Parameters

- **tol** (*float* / *None*) – The convergence tolerance.
- **max_it** (*int* / *None*) – The maximum number of iterations

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:532 <slepc4py/SLEPc/PEP.pyx#L532>`

setTrackAll(*trackall*)

Set flag to compute the residual of all approximate eigenpairs.

Logically collective.

Set if the solver must compute the residual of all approximate eigenpairs or not.

Parameters

- **trackall** (*bool*) – Whether compute all residuals or not.

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:762 <slepc4py/SLEPc/PEP.pyx#L762>`

setType(*pep_type*)

Set the particular solver to be used in the PEP object.

Logically collective.

Parameters

- **pep_type** (*Type* / *str*) – The solver to be used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:293 <slepc4py/SLEPc/PEP.pyx#L293>`

setUp()

Set up all the necessary internal data structures.

Collective.

Set up all the internal data structures necessary for the execution of the eigensolver.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1173 <slepc4py/SLEPc/PEP.pyx#L1173>`

Return type

None

setWhichEigenpairs(*which*)

Set which portion of the spectrum is to be sought.

Logically collective.

Parameters

- **which** (*Which*) – The portion of the spectrum to be sought by the solver.

Return type

None

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:458 <slepc4py/SLEPc/PEP.pyx#L458>`

solve()

Solve the eigensystem.

Collective.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1184 <slepc4py/SLEPc/PEP.pyx#L1184>`

Return type

`None`

valuesView(viewer=None)

Display the computed eigenvalues in a viewer.

Collective.

Parameters

viewer (`Viewer` / `None`) – Visualization context; if not provided, the standard output is used.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1353 <slepc4py/SLEPc/PEP.pyx#L1353>`

vectorsView(viewer=None)

Output computed eigenvectors to a viewer.

Collective.

Parameters

viewer (`Viewer` / `None`) – Visualization context; if not provided, the standard output is used.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:1368 <slepc4py/SLEPc/PEP.pyx#L1368>`

view(viewer=None)

Print the PEP data structure.

Collective.

Parameters

viewer (`Viewer` / `None`) – Visualization context; if not provided, the standard output is used.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:243 <slepc4py/SLEPc/PEP.pyx#L243>`

Attributes Documentation

bv

The basis vectors (BV) object associated.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2299 <slepc4py/SLEPc/PEP.pyx#L2299>`

ds

The direct solver (DS) object associated.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2313 <slepc4py/SLEPc/PEP.pyx#L2313>`

extract

The type of extraction technique to be employed.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2264 <slepc4py/SLEPc/PEP.pyx#L2264>`

max_it

The maximum iteration count.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2278 <slepc4py/SLEPc/PEP.pyx#L2278>`

problem_type

The type of the eigenvalue problem.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2243 <slepc4py/SLEPc/PEP.pyx#L2243>`

rg

The region (RG) object associated.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2306 <slepc4py/SLEPc/PEP.pyx#L2306>`

st

The spectral transformation (ST) object associated.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2292 <slepc4py/SLEPc/PEP.pyx#L2292>`

target

The value of the target.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2257 <slepc4py/SLEPc/PEP.pyx#L2257>`

tol

The tolerance.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2271 <slepc4py/SLEPc/PEP.pyx#L2271>`

track_all

Compute the residual norm of all approximate eigenpairs.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2285 <slepc4py/SLEPc/PEP.pyx#L2285>`

which

The portion of the spectrum to be sought.

:sources: `Source code at slepc4py/SLEPc/PEP.pyx:2250 <slepc4py/SLEPc/PEP.pyx#L2250>`

slepc4py.SLEPc.RG

class slepc4py.SLEPc.RG

Bases: `Object`

RG.

Enumerations

<i>QuadRule</i>	RG quadrature rule for contour integral methods.
<i>Type</i>	RG type.

slepc4py.SLEPc.RG.QuadRule

class slepc4py.SLEPc.RG.QuadRule

Bases: `object`

RG quadrature rule for contour integral methods.

- *TRAPEZOIDAL*: Trapezoidal rule.
- *CHEBYSHEV*: Chebyshev points.

Attributes Summary

<i>CHEBYSHEV</i>	Constant CHEBYSHEV of type <code>int</code>
<i>TRAPEZOIDAL</i>	Constant TRAPEZOIDAL of type <code>int</code>

Attributes Documentation

CHEBYSHEV: `int` = CHEBYSHEV

Constant CHEBYSHEV of type `int`

TRAPEZOIDAL: `int` = TRAPEZOIDAL

Constant TRAPEZOIDAL of type `int`

slepc4py.SLEPc.RG.Type

class slepc4py.SLEPc.RG.Type

Bases: `object`

RG type.

Attributes Summary

<i>ELLIPSE</i>	Object ELLIPSE of type <code>str</code>
<i>INTERVAL</i>	Object INTERVAL of type <code>str</code>
<i>POLYGON</i>	Object POLYGON of type <code>str</code>
<i>RING</i>	Object RING of type <code>str</code>

Attributes Documentation

ELLIPSE: `str` = ELLIPSE

Object ELLIPSE of type `str`

INTERVAL: `str` = INTERVAL

Object INTERVAL of type `str`

POLYGON: `str` = POLYGON

Object POLYGON of type `str`

RING: `str` = RING

Object RING of type `str`

Methods Summary

<code>appendOptionsPrefix([prefix])</code>	Append to the prefix used for searching for all RG options in the database.
<code>canUseConjugates([realmats])</code>	Half of integration points can be avoided (use their conjugates).
<code>checkInside(a)</code>	Determine if a set of given points are inside the region or not.
<code>computeBoundingBox()</code>	Endpoints of a rectangle in the complex plane containing the region.
<code>computeContour(n)</code>	Compute the coordinates of several points of the contour on the region.
<code>computeQuadrature(quad, n)</code>	Compute the values of the parameters used in a quadrature rule.
<code>create([comm])</code>	Create the RG object.
<code>destroy()</code>	Destroy the RG object.
<code>getComplement()</code>	Get the flag indicating whether the region is complemented or not.
<code>getEllipseParameters()</code>	Get the parameters that define the ellipse region.
<code>getIntervalEndpoints()</code>	Get the parameters that define the interval region.
<code>getOptionsPrefix()</code>	Get the prefix used for searching for all RG options in the database.
<code>getPolygonVertices()</code>	Get the parameters that define the interval region.
<code>getRingParameters()</code>	Get the parameters that define the ring region.
<code>getScale()</code>	Get the scaling factor.
<code>getType()</code>	Get the RG type of this object.
<code>isAxisymmetric([vertical])</code>	Determine if the region is symmetric wrt.
<code>isTrivial()</code>	Tell whether it is the trivial region (whole complex plane).
<code>setComplement([comp])</code>	Set a flag to indicate that the region is the complement of the specified one.
<code>setEllipseParameters(center, radius[, vscale])</code>	Set the parameters defining the ellipse region.
<code>setFromOptions()</code>	Set RG options from the options database.
<code>setIntervalEndpoints(a, b, c, d)</code>	Set the parameters defining the interval region.
<code>setOptionsPrefix([prefix])</code>	Set the prefix used for searching for all RG options in the database.
<code>setPolygonVertices(v)</code>	Set the vertices that define the polygon region.
<code>setRingParameters(center, radius, vscale, ...)</code>	Set the parameters defining the ring region.
<code>setScale([sfactor])</code>	Set the scaling factor to be used.
<code>setType(rg_type)</code>	Set the type for the RG object.
<code>view([viewer])</code>	Print the RG data structure.

Attributes Summary

<code>complement</code>	If the region is the complement of the specified one.
<code>scale</code>	The scaling factor to be used.

Methods Documentation

appendOptionsPrefix(*prefix=None*)

Append to the prefix used for searching for all RG options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all RG option requests.

Return type

None

:sources: [Source code at slepc4py/SLEPc/RG.pyx:141 <slepc4py/SLEPc/RG.pyx#L141>](#)

canUseConjugates(*realmats=True*)

Half of integration points can be avoided (use their conjugates).

Not collective.

Used in contour integral methods to determine whether half of integration points can be avoided (use their conjugates).

Parameters

realmats (*bool*) – True if the problem matrices are real.

Returns

Whether it is possible to use conjugates.

Return type

bool

:sources: [Source code at slepc4py/SLEPc/RG.pyx:361 <slepc4py/SLEPc/RG.pyx#L361>](#)

checkInside(*a*)

Determine if a set of given points are inside the region or not.

Not collective.

Parameters

a (*Sequence[complex]*) – The coordinates of the points.

Returns

Computed result for each point (1=inside, 0=on the contour, -1=outside).

Return type

ArrayInt

:sources: [Source code at slepc4py/SLEPc/RG.pyx:274 <slepc4py/SLEPc/RG.pyx#L274>](#)

computeBoundingBox()

Endpoints of a rectangle in the complex plane containing the region.

Not collective.

Determine the endpoints of a rectangle in the complex plane that contains the region.

Returns

- **a** (*float*) – The left endpoint of the bounding box in the real axis
- **b** (*float*) – The right endpoint of the bounding box in the real axis
- **c** (*float*) – The left endpoint of the bounding box in the imaginary axis
- **d** (*float*) – The right endpoint of the bounding box in the imaginary axis

Return type`tuple[float, float, float, float]`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:337 <slepc4py/SLEPc/RG.pyx#L337>`

computeContour(*n*)

Compute the coordinates of several points of the contour on the region.

Not collective.

Compute the coordinates of several points lying on the contour of the region.

Parameters

n (*int*) – The number of points to compute.

Returns

Computed points.

Return type`list of complex`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:306 <slepc4py/SLEPc/RG.pyx#L306>`

computeQuadrature(*quad, n*)

Compute the values of the parameters used in a quadrature rule.

Not collective.

Compute the values of the parameters used in a quadrature rule for a contour integral around the boundary of the region.

Parameters

- **quad** (*QuadRule*) – The type of quadrature.
- **n** (*int*) – The number of quadrature points to compute.

Returns

- **z** (*ArrayScalar*) – Quadrature points.
- **zn** (*ArrayScalar*) – Normalized quadrature points.
- **w** (*ArrayScalar*) – Quadrature weights.

Return type`tuple[ArrayScalar, ArrayScalar, ArrayScalar]`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:385 <slepc4py/SLEPc/RG.pyx#L385>`

create(*comm=None*)

Create the RG object.

Collective.

Parameters

comm (*Comm* | *None*) – MPI communicator; if not provided, it defaults to all processes.

Return type`Self`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:58 <slepc4py/SLEPc/RG.pyx#L58>`

destroy()

Destroy the RG object.

Collective.

:sources: `Source code at slepc4py/SLEPc/RG.pyx:48 <slepc4py/SLEPc/RG.pyx#L48>`

Return type

Self

getComplement()

Get the flag indicating whether the region is complemented or not.

Not collective.

Returns

Whether the region is complemented or not.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:212 <slepc4py/SLEPc/RG.pyx#L212>`

getEllipseParameters()

Get the parameters that define the ellipse region.

Not collective.

Returns

- **center** (*Scalar*) – The center.
- **radius** (`float`) – The radius.
- **vscale** (`float`) – The vertical scale.

Return type

`tuple[Scalar, float, float]`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:442 <slepc4py/SLEPc/RG.pyx#L442>`

getIntervalEndpoints()

Get the parameters that define the interval region.

Not collective.

Returns

- **a** (`float`) – The left endpoint in the real axis.
- **b** (`float`) – The right endpoint in the real axis.
- **c** (`float`) – The upper endpoint in the imaginary axis.
- **d** (`float`) – The lower endpoint in the imaginary axis.

Return type

`tuple[float, float, float, float]`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:486 <slepc4py/SLEPc/RG.pyx#L486>`

getOptionsPrefix()

Get the prefix used for searching for all RG options in the database.

Not collective.

Returns

The prefix string set for this RG object.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:126 <slepc4py/SLEPc/RG.pyx#L126>`

getPolygonVertices()

Get the parameters that define the interval region.

Not collective.

Returns

The vertices.

Return type

`ArrayComplex`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:534 <slepc4py/SLEPc/RG.pyx#L534>`

getRingParameters()

Get the parameters that define the ring region.

Not collective.

Returns

- **center** (`Scalar`) – The center.
- **radius** (`float`) – The radius.
- **vscale** (`float`) – The vertical scale.
- **start_ang** (`float`) – The right-hand side angle.
- **end_ang** (`float`) – The left-hand side angle.
- **width** (`float`) – The width of the ring.

Return type

`tuple[Scalar, float, float, float, float, float]`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:593 <slepc4py/SLEPc/RG.pyx#L593>`

getScale()

Get the scaling factor.

Not collective.

Returns

The scaling factor.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:259 <slepc4py/SLEPc/RG.pyx#L259>`

getType()

Get the RG type of this object.

Not collective.

Returns

The inner product type currently being used.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:90 <slepc4py/SLEPc/RG.pyx#L90>`

isAxisymmetric(*vertical=False*)

Determine if the region is symmetric wrt. the real or imaginary axis.

Not collective.

Determine if the region is symmetric with respect to the real or imaginary axis.

Parameters

vertical (*bool*) – True if symmetry must be checked against the vertical axis.

Returns

True if the region is axisymmetric.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:188 <slepc4py/SLEPc/RG.pyx#L188>`

isTrivial()

Tell whether it is the trivial region (whole complex plane).

Not collective.

Returns

True if the region is equal to the whole complex plane, e.g., an interval region with all four endpoints unbounded or an ellipse with infinite radius.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:171 <slepc4py/SLEPc/RG.pyx#L171>`

setComplement(*comp=True*)

Set a flag to indicate that the region is the complement of the specified one.

Logically collective.

Parameters

comp (*bool*) – Activate/deactivate the complementation of the region.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:227 <slepc4py/SLEPc/RG.pyx#L227>`

setEllipseParameters(*center, radius, vscale=None*)

Set the parameters defining the ellipse region.

Logically collective.

Parameters

- **center** (*Scalar*) – The center.
- **radius** (*float*) – The radius.
- **vscale** (*float* / *None*) – The vertical scale.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:421 <slepc4py/SLEPc/RG.pyx#L421>`

setFromOptions()

Set RG options from the options database.

Collective.

Notes

To see all options, run your program with the `-help` option.

:sources: `Source code at slepc4py/SLEPc/RG.pyx:156 <slepc4py/SLEPc/RG.pyx#L156>`

Return type

`None`

setIntervalEndpoints(*a*, *b*, *c*, *d*)

Set the parameters defining the interval region.

Logically collective.

Parameters

- **a** (`float`) – The left endpoint in the real axis.
- **b** (`float`) – The right endpoint in the real axis.
- **c** (`float`) – The upper endpoint in the imaginary axis.
- **d** (`float`) – The lower endpoint in the imaginary axis.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:463 <slepc4py/SLEPc/RG.pyx#L463>`

setOptionsPrefix(*prefix=None*)

Set the prefix used for searching for all RG options in the database.

Logically collective.

Parameters

prefix (`str` / `None`) – The prefix string to prepend to all RG option requests.

Return type

`None`

Notes

A hyphen (-) must NOT be given at the beginning of the prefix name. The first character of all runtime options is AUTOMATICALLY the hyphen.

:sources: `Source code at slepc4py/SLEPc/RG.pyx:105 <slepc4py/SLEPc/RG.pyx#L105>`

setPolygonVertices(*v*)

Set the vertices that define the polygon region.

Logically collective.

Parameters

v (`Sequence[float]` / `Sequence[Scalar]`) – The vertices.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/RG.pyx:510 <slepc4py/SLEPc/RG.pyx#L510>`

setRingParameters(*center, radius, vscale, start_ang, end_ang, width*)

Set the parameters defining the ring region.

Logically collective.

Parameters

- **center** (*Scalar*) – The center.
- **radius** (*float*) – The radius.
- **vscale** (*float*) – The vertical scale.
- **start_ang** (*float*) – The right-hand side angle.
- **end_ang** (*float*) – The left-hand side angle.
- **width** (*float*) – The width of the ring.

Return type

None

:sources: `Source code at slepc4py/SLEPc/RG.pyx:556 <slepc4py/SLEPc/RG.pyx#L556>`

setScale(*sfactor=None*)

Set the scaling factor to be used.

Logically collective.

Set the scaling factor to be used when checking that a point is inside the region and when computing the contour.

Parameters

sfactor (*float*) – The scaling factor (default=1).

Return type

None

:sources: `Source code at slepc4py/SLEPc/RG.pyx:241 <slepc4py/SLEPc/RG.pyx#L241>`

setType(*rg_type*)

Set the type for the RG object.

Logically collective.

Parameters

rg_type (*Type / str*) – The inner product type to be used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/RG.pyx:75 <slepc4py/SLEPc/RG.pyx#L75>`

view(*viewer=None*)

Print the RG data structure.

Collective.

Parameters

viewer (*Viewer / None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/RG.pyx:33 <slepc4py/SLEPc/RG.pyx#L33>`

Attributes Documentation

complement

If the region is the complement of the specified one.

:sources: `Source code at slepc4py/SLEPc/RG.pyx:625 <slepc4py/SLEPc/RG.pyx#L625>`

scale

The scaling factor to be used.

:sources: `Source code at slepc4py/SLEPc/RG.pyx:632 <slepc4py/SLEPc/RG.pyx#L632>`

slepc4py.SLEPc.ST

class slepc4py.SLEPc.ST

Bases: `Object`

ST.

Enumerations

<i>FilterDamping</i>	ST filter damping.
<i>FilterType</i>	ST filter type.
<i>MatMode</i>	ST matrix mode.
<i>Type</i>	ST types.

slepc4py.SLEPc.ST.FilterDamping

class slepc4py.SLEPc.ST.FilterDamping

Bases: `object`

ST filter damping.

- *NONE*: No damping
- *JACKSON*: Jackson damping
- *LANCZOS*: Lanczos damping
- *FEJER*: Fejer damping

Attributes Summary

<i>FEJER</i>	Constant FEJER of type <code>int</code>
<i>JACKSON</i>	Constant JACKSON of type <code>int</code>
<i>LANCZOS</i>	Constant LANCZOS of type <code>int</code>
<i>NONE</i>	Constant NONE of type <code>int</code>

Attributes Documentation

FEJER: `int` = FEJER

Constant FEJER of type `int`

JACKSON: `int = JACKSON`
 Constant JACKSON of type `int`

LANCZOS: `int = LANCZOS`
 Constant LANCZOS of type `int`

NONE: `int = NONE`
 Constant NONE of type `int`

slepc4py.SLEPc.ST.FilterType

class slepc4py.SLEPc.ST.**FilterType**

Bases: `object`

ST filter type.

- **FILTLAN:** An adapted implementation of the Filtered Lanczos Package.
- **CHEBYSHEV:** A polynomial filter based on a truncated Chebyshev series.

Attributes Summary

<i>CHEBYSHEV</i>	Constant CHEBYSHEV of type <code>int</code>
<i>FILTLAN</i>	Constant FILTLAN of type <code>int</code>

Attributes Documentation

CHEBYSHEV: `int = CHEBYSHEV`
 Constant CHEBYSHEV of type `int`

FILTLAN: `int = FILTLAN`
 Constant FILTLAN of type `int`

slepc4py.SLEPc.ST.MatMode

class slepc4py.SLEPc.ST.**MatMode**

Bases: `object`

ST matrix mode.

- **COPY:** A working copy of the matrix is created.
- **INPLACE:** The operation is computed in-place.
- **SHELL:** The matrix $A - \sigma B$ is handled as an implicit matrix.

Attributes Summary

<i>COPY</i>	Constant COPY of type <code>int</code>
<i>INPLACE</i>	Constant INPLACE of type <code>int</code>
<i>SHELL</i>	Constant SHELL of type <code>int</code>

Attributes Documentation

COPY: `int` = `COPY`

Constant COPY of type `int`

INPLACE: `int` = `INPLACE`

Constant INPLACE of type `int`

SHELL: `int` = `SHELL`

Constant SHELL of type `int`

slepc4py.SLEPc.ST.Type

class slepc4py.SLEPc.ST.Type

Bases: `object`

ST types.

- *SHELL*: User-defined.
- *SHIFT*: Shift from origin.
- *SINVERT*: Shift-and-invert.
- *CAYLEY*: Cayley transform.
- *PRECOND*: Preconditioner.
- *FILTER*: Polynomial filter.

Attributes Summary

<i>CAYLEY</i>	Object CAYLEY of type <code>str</code>
<i>FILTER</i>	Object FILTER of type <code>str</code>
<i>PRECOND</i>	Object PRECOND of type <code>str</code>
<i>SHELL</i>	Object SHELL of type <code>str</code>
<i>SHIFT</i>	Object SHIFT of type <code>str</code>
<i>SINVERT</i>	Object SINVERT of type <code>str</code>

Attributes Documentation

CAYLEY: `str` = `CAYLEY`

Object CAYLEY of type `str`

FILTER: `str` = `FILTER`

Object FILTER of type `str`

PRECOND: `str` = `PRECOND`

Object PRECOND of type `str`

SHELL: `str` = `SHELL`

Object SHELL of type `str`

SHIFT: `str` = `SHIFT`

Object SHIFT of type `str`

SINVERT: `str` = `SINVERT`

Object SINVERT of type `str`

Methods Summary

<code>appendOptionsPrefix([prefix])</code>	Append to the prefix used for searching for all ST options in the database.
<code>apply(x, y)</code>	Apply the spectral transformation operator to a vector.
<code>applyHermitianTranspose(x, y)</code>	Apply the hermitian-transpose of the operator to a vector.
<code>applyMat(x, y)</code>	Apply the spectral transformation operator to a matrix.
<code>applyTranspose(x, y)</code>	Apply the transpose of the operator to a vector.
<code>create([comm])</code>	Create the ST object.
<code>destroy()</code>	Destroy the ST object.
<code>getCayleyAntishift()</code>	Get the value of the anti-shift for the Cayley spectral transformation.
<code>getFilterDamping()</code>	Get the type of damping used in the polynomial filter.
<code>getFilterDegree()</code>	Get the degree of the filter polynomial.
<code>getFilterInterval()</code>	Get the interval containing the desired eigenvalues.
<code>getFilterRange()</code>	Get the interval containing all eigenvalues.
<code>getFilterType()</code>	Get the method to be used to build the polynomial filter.
<code>getKSP()</code>	Get the KSP object associated with the spectral transformation.
<code>getMatMode()</code>	Get a flag that indicates how the matrix is being shifted.
<code>getMatStructure()</code>	Get the internal Mat.Structure attribute.
<code>getMatrices()</code>	Get the matrices associated with the eigenvalue problem.
<code>getOperator()</code>	Get a shell matrix that represents the operator of the spectral transformation.
<code>getOptionsPrefix()</code>	Get the prefix used for searching for all ST options in the database.
<code>getPreconditionerMat()</code>	Get the matrix previously set by <code>setPreconditionerMat()</code> .
<code>getShift()</code>	Get the shift associated with the spectral transformation.
<code>getSplitPreconditioner()</code>	Get the matrices to be used to build the preconditioner.
<code>getTransform()</code>	Get the flag indicating whether the transformed matrices are computed or not.
<code>getType()</code>	Get the ST type of this object.
<code>reset()</code>	Reset the ST object.
<code>restoreOperator(op)</code>	Restore the previously seized operator matrix.
<code>setCayleyAntishift(tau)</code>	Set the value of the anti-shift for the Cayley spectral transformation.
<code>setFilterDamping(damping)</code>	Set the type of damping to be used in the polynomial filter.
<code>setFilterDegree(deg)</code>	Set the degree of the filter polynomial.
<code>setFilterInterval(inta, intb)</code>	Set the interval containing the desired eigenvalues.
<code>setFilterRange(left, right)</code>	Set the numerical range (or field of values) of the matrix.
<code>setFilterType(filter_type)</code>	Set the method to be used to build the polynomial filter.

continues on next page

Table 96 – continued from previous page

<code>setFromOptions()</code>	Set ST options from the options database.
<code>setKSP(ksp)</code>	Set the KSP object associated with the spectral transformation.
<code>setMatMode(mode)</code>	Set a flag to indicate how the matrix is being shifted.
<code>setMatStructure(structure)</code>	Set an internal Mat.Structure attribute.
<code>setMatrices(operators)</code>	Set the matrices associated with the eigenvalue problem.
<code>setOptionsPrefix([prefix])</code>	Set the prefix used for searching for all ST options in the database.
<code>setPreconditionerMat([P])</code>	Set the matrix to be used to build the preconditioner.
<code>setShift(shift)</code>	Set the shift associated with the spectral transformation.
<code>setSplitPreconditioner(operators[, structure])</code>	Set the matrices to be used to build the preconditioner.
<code>setTransform([flag])</code>	Set a flag to indicate whether the transformed matrices are computed or not.
<code>setType(st_type)</code>	Set the particular spectral transformation to be used.
<code>setUp()</code>	Prepare for the use of a spectral transformation.
<code>view([viewer])</code>	Print the ST data structure.

Attributes Summary

<code>ksp</code>	KSP object associated with the spectral transformation.
<code>mat_mode</code>	How the transformed matrices are being stored in the ST.
<code>mat_structure</code>	Relation of the sparsity pattern of all ST matrices.
<code>shift</code>	Value of the shift.
<code>transform</code>	If the transformed matrices are computed.

Methods Documentation

`appendOptionsPrefix(prefix=None)`

Append to the prefix used for searching for all ST options in the database.

Logically collective.

Parameters

prefix (`str` / `None`) – The prefix string to prepend to all ST option requests.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:198 <slepc4py/SLEPc/ST.pyx#L198>`

`apply(x, y)`

Apply the spectral transformation operator to a vector.

Collective.

Apply the spectral transformation operator to a vector, for instance $(A - sB)^{-1}B$ in the case of the shift-and-invert transformation and generalized eigenproblem.

Parameters

- \mathbf{x} (*Vec*) – The input vector.
- \mathbf{y} (*Vec*) – The result vector.

Return type

None

:sources: `Source code at slepc4py/SLEPc/ST.pyx:563 <slepc4py/SLEPc/ST.pyx#L563>`

applyHermitianTranspose(x, y)

Apply the hermitian-transpose of the operator to a vector.

Collective.

Apply the hermitian-transpose of the operator to a vector, for instance $B^H(A - sB)^{-H}$ in the case of the shift-and-invert transformation and generalized eigenproblem.

Parameters

- \mathbf{x} (*Vec*) – The input vector.
- \mathbf{y} (*Vec*) – The result vector.

Return type

None

:sources: `Source code at slepc4py/SLEPc/ST.pyx:601 <slepc4py/SLEPc/ST.pyx#L601>`

applyMat(x, y)

Apply the spectral transformation operator to a matrix.

Collective.

Apply the spectral transformation operator to a matrix, for instance $(A - sB)^{-1}B$ in the case of the shift-and-invert transformation and generalized eigenproblem.

Parameters

- \mathbf{x} (*Mat*) – The input matrix.
- \mathbf{y} (*Mat*) – The result matrix.

Return type

None

:sources: `Source code at slepc4py/SLEPc/ST.pyx:620 <slepc4py/SLEPc/ST.pyx#L620>`

applyTranspose(x, y)

Apply the transpose of the operator to a vector.

Collective.

Apply the transpose of the operator to a vector, for instance $B^T(A - sB)^{-T}$ in the case of the shift-and-invert transformation and generalized eigenproblem.

Parameters

- \mathbf{x} (*Vec*) – The input vector.
- \mathbf{y} (*Vec*) – The result vector.

Return type

None

:sources: `Source code at slepc4py/SLEPc/ST.pyx:582 <slepc4py/SLEPc/ST.pyx#L582>`

create(*comm=None*)

Create the ST object.

Collective.

Parameters

comm (*Comm* / *None*) – MPI communicator; if not provided, it defaults to all processes.

Return type

Self

:sources: `Source code at slepc4py/SLEPc/ST.pyx:106 <slepc4py/SLEPc/ST.pyx#L106>`

destroy()

Destroy the ST object.

Collective.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:88 <slepc4py/SLEPc/ST.pyx#L88>`

Return type

Self

getCayleyAntishift()

Get the value of the anti-shift for the Cayley spectral transformation.

Not collective.

Returns

The anti-shift.

Return type

Scalar

:sources: `Source code at slepc4py/SLEPc/ST.pyx:692 <slepc4py/SLEPc/ST.pyx#L692>`

getFilterDamping()

Get the type of damping used in the polynomial filter.

Not collective.

Returns

The type of damping.

Return type

FilterDamping

:sources: `Source code at slepc4py/SLEPc/ST.pyx:870 <slepc4py/SLEPc/ST.pyx#L870>`

getFilterDegree()

Get the degree of the filter polynomial.

Not collective.

Returns

The polynomial degree.

Return type

int

:sources: `Source code at slepc4py/SLEPc/ST.pyx:841 <slepc4py/SLEPc/ST.pyx#L841>`

getFilterInterval()

Get the interval containing the desired eigenvalues.

Not collective.

Returns

- **inta** (`float`) – The left end of the interval.
- **intb** (`float`) – The right end of the interval.

Return type

`tuple[float, float]`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:765 <slepc4py/SLEPc/ST.pyx#L765>`

getFilterRange()

Get the interval containing all eigenvalues.

Not collective.

Returns

- **left** (`float`) – The left end of the interval.
- **right** (`float`) – The right end of the interval.

Return type

`tuple[float, float]`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:809 <slepc4py/SLEPc/ST.pyx#L809>`

getFilterType()

Get the method to be used to build the polynomial filter.

Not collective.

Returns

The type of filter.

Return type

`FilterType`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:721 <slepc4py/SLEPc/ST.pyx#L721>`

getKSP()

Get the KSP object associated with the spectral transformation.

Collective.

Returns

The linear solver object.

Return type

`petsc4py.PETSc.KSP`

Notes

On output, the internal value of `petsc4py.PETSc.KSP` can be NULL if the combination of eigenproblem type and selected transformation does not require to solve a linear system of equations.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:458 <slepc4py/SLEPc/ST.pyx#L458>`

getMatMode()

Get a flag that indicates how the matrix is being shifted.

Not collective.

Get a flag that indicates how the matrix is being shifted in the shift-and-invert and Cayley spectral transformations.

Returns

The mode flag.

Return type

MatMode

:sources: `Source code at slepc4py/SLEPc/ST.pyx:342 <slepc4py/SLEPc/ST.pyx#L342>`

getMatStructure()

Get the internal Mat.Structure attribute.

Not collective.

Get the internal Mat.Structure attribute to indicate which is the relation of the sparsity pattern of the matrices.

Returns

The structure flag.

Return type

`petsc4py.PETSc.Mat.Structure`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:427 <slepc4py/SLEPc/ST.pyx#L427>`

getMatrices()

Get the matrices associated with the eigenvalue problem.

Collective.

Returns

The matrices associated with the eigensystem.

Return type

`list of petsc4py.PETSc.Mat`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:378 <slepc4py/SLEPc/ST.pyx#L378>`

getOperator()

Get a shell matrix that represents the operator of the spectral transformation.

Collective.

Returns

Operator matrix.

Return type

`petsc4py.PETSc.Mat`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:639 <slepc4py/SLEPc/ST.pyx#L639>`

getOptionsPrefix()

Get the prefix used for searching for all ST options in the database.

Not collective.

Returns

The prefix string set for this ST object.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:183 <slepc4py/SLEPc/ST.pyx#L183>`

getPreconditionerMat()

Get the matrix previously set by setPreconditionerMat().

Not collective.

Returns

The matrix that will be used in constructing the preconditioner.

Return type

`petsc4py.PETSc.Mat`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:494 <slepc4py/SLEPc/ST.pyx#L494>`

getShift()

Get the shift associated with the spectral transformation.

Not collective.

Returns

The value of the shift.

Return type

`Scalar`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:250 <slepc4py/SLEPc/ST.pyx#L250>`

getSplitPreconditioner()

Get the matrices to be used to build the preconditioner.

Not collective.

Returns

- `list` of `petsc4py.PETSc.Mat` – The list of matrices associated with the preconditioner.
- `petsc4py.PETSc.Mat.Structure` – The structure flag.

Return type

`tuple[list[petsc4py.PETSc.Mat], petsc4py.PETSc.Mat.Structure]`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:529 <slepc4py/SLEPc/ST.pyx#L529>`

getTransform()

Get the flag indicating whether the transformed matrices are computed or not.

Not collective.

Returns

This flag is intended for the case of polynomial eigenproblems solved via linearization. If this flag is `False` (default) the spectral transformation is applied to the linearization (handled by the eigensolver), otherwise it is applied to the original problem.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:283 <slepc4py/SLEPc/ST.pyx#L283>`

getType()

Get the ST type of this object.

Not collective.

Returns

The spectral transformation currently being used.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:147 <slepc4py/SLEPc/ST.pyx#L147>`

reset()

Reset the ST object.

Collective.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:98 <slepc4py/SLEPc/ST.pyx#L98>`

Return type

`None`

restoreOperator(*op*)

Restore the previously seized operator matrix.

Logically collective.

Parameters

op (*Mat*) – Operator matrix previously obtained with `getOperator()`.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/ST.pyx:655 <slepc4py/SLEPc/ST.pyx#L655>`

setCayleyAntishift(*tau*)

Set the value of the anti-shift for the Cayley spectral transformation.

Logically collective.

Parameters

tau (*Scalar*) – The anti-shift.

Return type

`None`

Notes

In the generalized Cayley transform, the operator can be expressed as $OP = inv(A - \sigma B)(A + tau B)$. This function sets the value of *tau*. Use `setShift()` for setting σ .

:sources: `Source code at slepc4py/SLEPc/ST.pyx:671 <slepc4py/SLEPc/ST.pyx#L671>`

setFilterDamping(*damping*)

Set the type of damping to be used in the polynomial filter.

Logically collective.

Parameter

damping

The type of damping.

:sources: [Source code at slepc4py/SLEPc/ST.pyx:856 <slepc4py/SLEPc/ST.pyx#L856>](#)

Parameters

damping (*FilterDamping*)

Return type

None

setFilterDegree(*deg*)

Set the degree of the filter polynomial.

Logically collective.

Parameters

deg (*int*) – The polynomial degree.

Return type

None

:sources: [Source code at slepc4py/SLEPc/ST.pyx:827 <slepc4py/SLEPc/ST.pyx#L827>](#)

setFilterInterval(*inta*, *intb*)

Set the interval containing the desired eigenvalues.

Logically collective.

Parameters

- **inta** (*float*) – The left end of the interval.
- **intb** (*float*) – The right end of the interval.

Return type

None

Notes

The filter will be configured to emphasize eigenvalues contained in the given interval, and damp out eigenvalues outside it. If the interval is open, then the filter is low- or high-pass, otherwise it is mid-pass.

Common usage is to set the interval in *EPS* with *EPS.setInterval()*.

The interval must be contained within the numerical range of the matrix, see *ST.setFilterRange()*.

:sources: [Source code at slepc4py/SLEPc/ST.pyx:736 <slepc4py/SLEPc/ST.pyx#L736>](#)

setFilterRange(*left*, *right*)

Set the numerical range (or field of values) of the matrix.

Logically collective.

Set the numerical range (or field of values) of the matrix, that is, the interval containing all eigenvalues.

Parameters

- **left** (*float*) – The left end of the interval.
- **right** (*float*) – The right end of the interval.

Return type

None

Notes

The filter will be most effective if the numerical range is tight, that is, left and right are good approximations to the leftmost and rightmost eigenvalues, respectively.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:783 <slepc4py/SLEPc/ST.pyx#L783>`

setFilterType(*filter_type*)

Set the method to be used to build the polynomial filter.

Logically collective.

Parameter**filter_type**

The type of filter.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:707 <slepc4py/SLEPc/ST.pyx#L707>`

Parameters

filter_type (*FilterType*)

Return type

None

setFromOptions()

Set ST options from the options database.

Collective.

This routine must be called before `setUp()` if the user is to be allowed to set the solver type.

Notes

To see all options, run your program with the -help option.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:213 <slepc4py/SLEPc/ST.pyx#L213>`

Return type

None

setKSP(*ksp*)

Set the KSP object associated with the spectral transformation.

Collective.

Parameters

- `petsc4py.PETSc.KSP` – The linear solver object.
- **ksp** (*petsc4py.PETSc.KSP*)

Return type

None

:sources: `Source code at slepc4py/SLEPc/ST.pyx:445 <slepc4py/SLEPc/ST.pyx#L445>`

setMatMode(mode)

Set a flag to indicate how the matrix is being shifted.

Logically collective.

Set a flag to indicate how the matrix is being shifted in the shift-and-invert and Cayley spectral transformations.

Parameters

mode (`MatMode`) – The mode flag.

Return type

`None`

Notes

By default (`ST.MatMode.COPY`), a copy of matrix A is made and then this copy is shifted explicitly, e.g. $A \leftarrow (A - sB)$.

With `ST.MatMode.INPLACE`, the original matrix A is shifted at `setUp()` and unshifted at the end of the computations. With respect to the previous one, this mode avoids a copy of matrix A . However, a drawback is that the recovered matrix might be slightly different from the original one (due to roundoff).

With `ST.MatMode.SHELL`, the solver works with an implicit shell matrix that represents the shifted matrix. This mode is the most efficient in creating the shifted matrix but it places serious limitations to the linear solves performed in each iteration of the eigensolver (typically, only iterative solvers with Jacobi preconditioning can be used).

In the case of generalized problems, in the two first modes the matrix $A - sB$ has to be computed explicitly. The efficiency of this computation can be controlled with `setMatStructure()`.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:302 <slepc4py/SLEPc/ST.pyx#L302>`

setMatStructure(structure)

Set an internal `Mat.Structure` attribute.

Logically collective.

Set an internal `Mat.Structure` attribute to indicate which is the relation of the sparsity pattern of the two matrices A and B constituting the generalized eigenvalue problem. This function has no effect in the case of standard eigenproblems.

Parameters

structure (`petsc4py.PETSc.Mat.Structure`) – Either same, different, or a subset of the non-zero sparsity pattern.

Return type

`None`

Notes

By default, the sparsity patterns are assumed to be different. If the patterns are equal or a subset then it is recommended to set this attribute for efficiency reasons (in particular, for internal `AXPY()` matrix operations).

:sources: `Source code at slepc4py/SLEPc/ST.pyx:400 <slepc4py/SLEPc/ST.pyx#L400>`

setMatrices(operators)

Set the matrices associated with the eigenvalue problem.

Collective.

Parameters

operators (*list*[*Mat*]) – The matrices associated with the eigensystem.

Return type

None

:sources: [Source code at slepc4py/SLEPc/ST.pyx:360 <slepc4py/SLEPc/ST.pyx#L360>](#)

setOptionsPrefix(*prefix=None*)

Set the prefix used for searching for all ST options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all ST option requests.

Return type

None

Notes

A hyphen (-) must NOT be given at the beginning of the prefix name. The first character of all runtime options is AUTOMATICALLY the hyphen.

:sources: [Source code at slepc4py/SLEPc/ST.pyx:162 <slepc4py/SLEPc/ST.pyx#L162>](#)

setPreconditionerMat(*P=None*)

Set the matrix to be used to build the preconditioner.

Collective.

Parameters

P (*Mat* / *None*) – The matrix that will be used in constructing the preconditioner.

Return type

None

:sources: [Source code at slepc4py/SLEPc/ST.pyx:480 <slepc4py/SLEPc/ST.pyx#L480>](#)

setShift(*shift*)

Set the shift associated with the spectral transformation.

Collective.

Parameters

shift (*Scalar*) – The value of the shift.

Return type

None

Notes

In some spectral transformations, changing the shift may have associated a lot of work, for example recomputing a factorization.

:sources: [Source code at slepc4py/SLEPc/ST.pyx:230 <slepc4py/SLEPc/ST.pyx#L230>](#)

setSplitPreconditioner(*operators, structure=None*)

Set the matrices to be used to build the preconditioner.

Collective.

Parameters

- **operators** (*list*[*petsc4py.PETSc.Mat*]) – The matrices associated with the preconditioner.
- **structure** (*petsc4py.PETSc.Mat.Structure* / *None*)

Return type

None

:sources: [Source code at slepc4py/SLEPc/ST.pyx:510 <slepc4py/SLEPc/ST.pyx#L510>](#)

setTransform(flag=True)

Set a flag to indicate whether the transformed matrices are computed or not.

Logically collective.

Parameters

flag (*bool*) – This flag is intended for the case of polynomial eigenproblems solved via linearization. If this flag is False (default) the spectral transformation is applied to the linearization (handled by the eigensolver), otherwise it is applied to the original problem.

Return type

None

:sources: [Source code at slepc4py/SLEPc/ST.pyx:265 <slepc4py/SLEPc/ST.pyx#L265>](#)

setType(st_type)

Set the particular spectral transformation to be used.

Logically collective.

Parameters

st_type (*Type* / *str*) – The spectral transformation to be used.

Return type

None

Notes

See *ST.Type* for available methods. The default is *ST.Type.SHIFT* with a zero shift. Normally, it is best to use *setFromOptions()* and then set the ST type from the options database rather than by using this routine. Using the options database provides the user with maximum flexibility in evaluating the different available methods.

:sources: [Source code at slepc4py/SLEPc/ST.pyx:123 <slepc4py/SLEPc/ST.pyx#L123>](#)

setUp()

Prepare for the use of a spectral transformation.

Collective.

:sources: [Source code at slepc4py/SLEPc/ST.pyx:555 <slepc4py/SLEPc/ST.pyx#L555>](#)

Return type

None

view(viewer=None)

Print the ST data structure.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/ST.pyx:73 <slepc4py/SLEPc/ST.pyx#L73>`

Attributes Documentation

ksp

KSP object associated with the spectral transformation.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:915 <slepc4py/SLEPc/ST.pyx#L915>`

mat_mode

How the transformed matrices are being stored in the ST.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:901 <slepc4py/SLEPc/ST.pyx#L901>`

mat_structure

Relation of the sparsity pattern of all ST matrices.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:908 <slepc4py/SLEPc/ST.pyx#L908>`

shift

Value of the shift.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:887 <slepc4py/SLEPc/ST.pyx#L887>`

transform

If the transformed matrices are computed.

:sources: `Source code at slepc4py/SLEPc/ST.pyx:894 <slepc4py/SLEPc/ST.pyx#L894>`

slepc4py.SLEPc.STFilterDamping

class slepc4py.SLEPc.STFilterDamping

Bases: `object`

ST filter damping.

- *NONE*: No damping
- *JACKSON*: Jackson damping
- *LANCZOS*: Lanczos damping
- *FEJER*: Fejer damping

Attributes Summary

<i>FEJER</i>	Constant FEJER of type <code>int</code>
<i>JACKSON</i>	Constant JACKSON of type <code>int</code>
<i>LANCZOS</i>	Constant LANCZOS of type <code>int</code>
<i>NONE</i>	Constant NONE of type <code>int</code>

Attributes Documentation

FEJER: `int` = FEJER

Constant FEJER of type `int`

JACKSON: `int` = JACKSON
 Constant JACKSON of type `int`

LANCZOS: `int` = LANCZOS
 Constant LANCZOS of type `int`

NONE: `int` = NONE
 Constant NONE of type `int`

slepc4py.SLEPc.STFilterType

class slepc4py.SLEPc.STFilterType

Bases: `object`

ST filter type.

- FILTLAN: An adapted implementation of the Filtered Lanczos Package.
- CHEBYSHEV: A polynomial filter based on a truncated Chebyshev series.

Attributes Summary

<i>CHEBYSHEV</i>	Constant CHEBYSHEV of type <code>int</code>
<i>FILTLAN</i>	Constant FILTLAN of type <code>int</code>

Attributes Documentation

CHEBYSHEV: `int` = CHEBYSHEV
 Constant CHEBYSHEV of type `int`

FILTLAN: `int` = FILTLAN
 Constant FILTLAN of type `int`

slepc4py.SLEPc.SVD

class slepc4py.SLEPc.SVD

Bases: `Object`

SVD.

Enumerations

<i>Conv</i>	SVD convergence test.
<i>ConvergedReason</i>	SVD convergence reasons.
<i>ErrorType</i>	SVD error type to assess accuracy of computed solutions.
<i>ProblemType</i>	SVD problem type.
<i>Stop</i>	SVD stopping test.
<i>TRLanczosGBidiag</i>	SVD TRLanczos bidiagonalization choices for the GSVD case.
<i>Type</i>	SVD types.
<i>Which</i>	SVD desired part of spectrum.

slepc4py.SLEPc.SVD.Conv

class slepc4py.SLEPc.SVD.Conv

Bases: `object`

SVD convergence test.

- *ABS*: Absolute convergence test.
- *REL*: Convergence test relative to the singular value.
- *NORM*: Convergence test relative to the matrix norms.
- *MAXIT*: No convergence until maximum number of iterations has been reached.
- *USER*: User-defined convergence test.

Attributes Summary

<i>ABS</i>	Constant ABS of type <code>int</code>
<i>MAXIT</i>	Constant MAXIT of type <code>int</code>
<i>NORM</i>	Constant NORM of type <code>int</code>
<i>REL</i>	Constant REL of type <code>int</code>
<i>USER</i>	Constant USER of type <code>int</code>

Attributes Documentation

ABS: `int` = **ABS**

Constant ABS of type `int`

MAXIT: `int` = **MAXIT**

Constant MAXIT of type `int`

NORM: `int` = **NORM**

Constant NORM of type `int`

REL: `int` = **REL**

Constant REL of type `int`

USER: `int` = **USER**

Constant USER of type `int`

slepc4py.SLEPc.SVD.ConvergedReason

class slepc4py.SLEPc.SVD.ConvergedReason

Bases: `object`

SVD convergence reasons.

- *CONVERGED_TOL*: All eigenpairs converged to requested tolerance.
- *CONVERGED_USER*: User-defined convergence criterion satisfied.
- *CONVERGED_MAXIT*: Maximum iterations completed in case MAXIT convergence criterion.
- *DIVERGED_ITS*: Maximum number of iterations exceeded.
- *DIVERGED_BREAKDOWN*: Solver failed due to breakdown.

- **DIVERGED_SYMMETRY_LOST**: Underlying indefinite eigensolver was not able to keep symmetry.
- **CONVERGED_ITERATING**: Iteration not finished yet.

Attributes Summary

<i>CONVERGED_ITERATING</i>	Constant CONVERGED_ITERATING of type <i>int</i>
<i>CONVERGED_MAXIT</i>	Constant CONVERGED_MAXIT of type <i>int</i>
<i>CONVERGED_TOL</i>	Constant CONVERGED_TOL of type <i>int</i>
<i>CONVERGED_USER</i>	Constant CONVERGED_USER of type <i>int</i>
<i>DIVERGED_BREAKDOWN</i>	Constant DIVERGED_BREAKDOWN of type <i>int</i>
<i>DIVERGED_ITS</i>	Constant DIVERGED_ITS of type <i>int</i>
<i>DIVERGED_SYMMETRY_LOST</i>	Constant DIVERGED_SYMMETRY_LOST of type <i>int</i>
<i>ITERATING</i>	Constant ITERATING of type <i>int</i>

Attributes Documentation

CONVERGED_ITERATING: *int* = CONVERGED_ITERATING

Constant CONVERGED_ITERATING of type *int*

CONVERGED_MAXIT: *int* = CONVERGED_MAXIT

Constant CONVERGED_MAXIT of type *int*

CONVERGED_TOL: *int* = CONVERGED_TOL

Constant CONVERGED_TOL of type *int*

CONVERGED_USER: *int* = CONVERGED_USER

Constant CONVERGED_USER of type *int*

DIVERGED_BREAKDOWN: *int* = DIVERGED_BREAKDOWN

Constant DIVERGED_BREAKDOWN of type *int*

DIVERGED_ITS: *int* = DIVERGED_ITS

Constant DIVERGED_ITS of type *int*

DIVERGED_SYMMETRY_LOST: *int* = DIVERGED_SYMMETRY_LOST

Constant DIVERGED_SYMMETRY_LOST of type *int*

ITERATING: *int* = ITERATING

Constant ITERATING of type *int*

slepc4py.SLEPc.SVD.ErrorType

class slepc4py.SLEPc.SVD.ErrorType

Bases: *object*

SVD error type to assess accuracy of computed solutions.

- **ABSOLUTE**: Absolute error.
- **RELATIVE**: Relative error.
- **NORM**: Error relative to the matrix norm.

Attributes Summary

<i>ABSOLUTE</i>	Constant ABSOLUTE of type <code>int</code>
<i>NORM</i>	Constant NORM of type <code>int</code>
<i>RELATIVE</i>	Constant RELATIVE of type <code>int</code>

Attributes Documentation

ABSOLUTE: `int` = **ABSOLUTE**

Constant ABSOLUTE of type `int`

NORM: `int` = **NORM**

Constant NORM of type `int`

RELATIVE: `int` = **RELATIVE**

Constant RELATIVE of type `int`

`slepc4py.SLEPc.SVD.ProblemType`

class `slepc4py.SLEPc.SVD.ProblemType`

Bases: `object`

SVD problem type.

- *STANDARD*: Standard SVD.
- *GENERALIZED*: Generalized singular value decomposition (GSVD).
- *HYPERBOLIC*: Hyperbolic singular value decomposition (HSVD).

Attributes Summary

<i>GENERALIZED</i>	Constant GENERALIZED of type <code>int</code>
<i>HYPERBOLIC</i>	Constant HYPERBOLIC of type <code>int</code>
<i>STANDARD</i>	Constant STANDARD of type <code>int</code>

Attributes Documentation

GENERALIZED: `int` = **GENERALIZED**

Constant GENERALIZED of type `int`

HYPERBOLIC: `int` = **HYPERBOLIC**

Constant HYPERBOLIC of type `int`

STANDARD: `int` = **STANDARD**

Constant STANDARD of type `int`

`slepc4py.SLEPc.SVD.Stop`

class `slepc4py.SLEPc.SVD.Stop`

Bases: `object`

SVD stopping test.

- *BASIC*: Default stopping test.

- *USER*: User-defined stopping test.
- *THRESHOLD*: Threshold stopping test.

Attributes Summary

<i>BASIC</i>	Constant BASIC of type <i>int</i>
<i>THRESHOLD</i>	Constant THRESHOLD of type <i>int</i>
<i>USER</i>	Constant USER of type <i>int</i>

Attributes Documentation

BASIC: *int* = BASIC

Constant BASIC of type *int*

THRESHOLD: *int* = THRESHOLD

Constant THRESHOLD of type *int*

USER: *int* = USER

Constant USER of type *int*

slepc4py.SLEPc.SVD.TRLanczosGBidiag

class slepc4py.SLEPc.SVD.TRLanczosGBidiag

Bases: *object*

SVD TRLanczos bidiagonalization choices for the GSVD case.

- *SINGLE*: Single bidiagonalization (Qa).
- *UPPER*: Joint bidiagonalization, both Qa and Qb in upper bidiagonal form.
- *LOWER*: Joint bidiagonalization, Qa lower bidiagonal, Qb upper bidiagonal.

Attributes Summary

<i>LOWER</i>	Constant LOWER of type <i>int</i>
<i>SINGLE</i>	Constant SINGLE of type <i>int</i>
<i>UPPER</i>	Constant UPPER of type <i>int</i>

Attributes Documentation

LOWER: *int* = LOWER

Constant LOWER of type *int*

SINGLE: *int* = SINGLE

Constant SINGLE of type *int*

UPPER: *int* = UPPER

Constant UPPER of type *int*

slepc4py.SLEPc.SVD.Type

class slepc4py.SLEPc.SVD.Type

Bases: `object`

SVD types.

- *CROSS*: Eigenproblem with the cross-product matrix.
- *CYCLIC*: Eigenproblem with the cyclic matrix.
- *LAPACK*: Wrappers to dense SVD solvers in Lapack.
- *LANCZOS*: Lanczos.
- *TRLANCZOS*: Thick-restart Lanczos.
- *RANDOMIZED*: Iterative RSVD for low-rank matrices.

Wrappers to external SVD solvers (should be enabled during installation of SLEPc)

- *SCALAPACK*:
- *KSVD*:
- *ELEMENTAL*:
- *PRIMME*:

Attributes Summary

<i>CROSS</i>	Object CROSS of type <code>str</code>
<i>CYCLIC</i>	Object CYCLIC of type <code>str</code>
<i>ELEMENTAL</i>	Object ELEMENTAL of type <code>str</code>
<i>KSVD</i>	Object KSVD of type <code>str</code>
<i>LANCZOS</i>	Object LANCZOS of type <code>str</code>
<i>LAPACK</i>	Object LAPACK of type <code>str</code>
<i>PRIMME</i>	Object PRIMME of type <code>str</code>
<i>RANDOMIZED</i>	Object RANDOMIZED of type <code>str</code>
<i>SCALAPACK</i>	Object SCALAPACK of type <code>str</code>
<i>TRLANCZOS</i>	Object TRLANCZOS of type <code>str</code>

Attributes Documentation

CROSS: `str` = **CROSS**

Object CROSS of type `str`

CYCLIC: `str` = **CYCLIC**

Object CYCLIC of type `str`

ELEMENTAL: `str` = **ELEMENTAL**

Object ELEMENTAL of type `str`

KSVD: `str` = **KSVD**

Object KSVD of type `str`

LANCZOS: `str` = **LANCZOS**

Object LANCZOS of type `str`

LAPACK: `str = LAPACK`

Object LAPACK of type `str`

PRIMME: `str = PRIMME`

Object PRIMME of type `str`

RANDOMIZED: `str = RANDOMIZED`

Object RANDOMIZED of type `str`

SCALAPACK: `str = SCALAPACK`

Object SCALAPACK of type `str`

TRLANCZOS: `str = TRLANCZOS`

Object TRLANCZOS of type `str`

slepc4py.SLEPc.SVD.Which

class slepc4py.SLEPc.SVD.Which

Bases: `object`

SVD desired part of spectrum.

- `LARGEST`: Largest singular values.
- `SMALLEST`: Smallest singular values.

Attributes Summary

<code>LARGEST</code>	Constant LARGEST of type <code>int</code>
<code>SMALLEST</code>	Constant SMALLEST of type <code>int</code>

Attributes Documentation

LARGEST: `int = LARGEST`

Constant LARGEST of type `int`

SMALLEST: `int = SMALLEST`

Constant SMALLEST of type `int`

Methods Summary

<code>appendOptionsPrefix([prefix])</code>	Append to the prefix used for searching for all SVD options in the database.
<code>cancelMonitor()</code>	Clear all monitors for an <code>SVD</code> object.
<code>computeError(i[, etype])</code>	Compute the error associated with the i-th singular triplet.
<code>create([comm])</code>	Create the SVD object.
<code>destroy()</code>	Destroy the SVD object.
<code>errorView([etype, viewer])</code>	Display the errors associated with the computed solution.
<code>getBV()</code>	Get the basis vectors objects associated to the SVD object.
<code>getConverged()</code>	Get the number of converged singular triplets.

continues on next page

Table 109 – continued from previous page

<code>getConvergedReason()</code>	Get the reason why the <code>solve()</code> iteration was stopped.
<code>getConvergenceTest()</code>	Get the method used to compute the error estimate used in the convergence test.
<code>getCrossEPS()</code>	Get the eigensolver object associated to the singular value solver.
<code>getCrossExplicitMatrix()</code>	Get the flag indicating if $A^T A$ is built explicitly.
<code>getCyclicEPS()</code>	Get the eigensolver object associated to the singular value solver.
<code>getCyclicExplicitMatrix()</code>	Get the flag indicating if $H(A)$ is built explicitly.
<code>getDS()</code>	Get the direct solver associated to the singular value solver.
<code>getDimensions()</code>	Get the number of singular values to compute and the dimension of the subspace.
<code>getImplicitTranspose()</code>	Get the mode used to handle the transpose of the matrix associated.
<code>getIterationNumber()</code>	Get the current iteration number.
<code>getLanczosOneSide()</code>	Get if the variant of the Lanczos method to be used is one-sided or two-sided.
<code>getMonitor()</code>	Get the list of monitor functions.
<code>getOperators()</code>	Get the matrices associated with the singular value problem.
<code>getOptionsPrefix()</code>	Get the prefix used for searching for all SVD options in the database.
<code>getProblemType()</code>	Get the problem type from the SVD object.
<code>getSignature([omega])</code>	Get the signature matrix defining a hyperbolic singular value problem.
<code>getSingularTriplet(i[, U, V])</code>	Get the i-th triplet of the singular value decomposition.
<code>getStoppingTest()</code>	Get the stopping function.
<code>getTRLanczosExplicitMatrix()</code>	Get the flag indicating if $Z = [A; B]$ is built explicitly.
<code>getTRLanczosGBidiag()</code>	Get bidiagonalization choice used in the GSVD TR-Lanczos solver.
<code>getTRLanczosKSP()</code>	Get the linear solver object associated with the SVD solver.
<code>getTRLanczosLocking()</code>	Get the locking flag used in the thick-restart Lanczos method.
<code>getTRLanczosOneSide()</code>	Get if the variant of the method to be used is one-sided or two-sided.
<code>getTRLanczosRestart()</code>	Get the restart parameter used in the thick-restart Lanczos method.
<code>getThreshold()</code>	Get the threshold used in the threshold stopping test.
<code>getTolerances()</code>	Get the tolerance and maximum iteration count.
<code>getTrackAll()</code>	Get the flag indicating if all residual norms must be computed or not.
<code>getType()</code>	Get the SVD type of this object.
<code>getValue(i)</code>	Get the i-th singular value as computed by <code>solve()</code> .
<code>getVectors(i, U, V)</code>	Get the i-th left and right singular vectors as computed by <code>solve()</code> .
<code>getWhichSingularTriplets()</code>	Get which singular triplets are to be sought.

continues on next page

Table 109 – continued from previous page

<i>isGeneralized()</i>	Tell if the SVD corresponds to a generalized singular value problem.
<i>isHyperbolic()</i>	Tell whether the SVD object corresponds to a hyperbolic singular value problem.
<i>reset()</i>	Reset the SVD object.
<i>setBV(V[, U])</i>	Set basis vectors objects associated to the SVD solver.
<i>setConvergenceTest(conv)</i>	Set how to compute the error estimate used in the convergence test.
<i>setCrossEPS(eps)</i>	Set an eigensolver object associated to the singular value solver.
<i>setCrossExplicitMatrix([flag])</i>	Set if the eigensolver operator $A^T A$ must be computed.
<i>setCyclicEPS(eps)</i>	Set an eigensolver object associated to the singular value solver.
<i>setCyclicExplicitMatrix([flag])</i>	Set if the eigensolver operator $H(A)$ must be computed explicitly.
<i>setDS(ds)</i>	Set a direct solver object associated to the singular value solver.
<i>setDimensions([nsv, ncv, mpd])</i>	Set the number of singular values to compute and the dimension of the subspace.
<i>setFromOptions()</i>	Set SVD options from the options database.
<i>setImplicitTranspose(mode)</i>	Set how to handle the transpose of the matrix associated.
<i>setInitialSpace([spaceright, spaceleft])</i>	Set the initial spaces from which the SVD solver starts to iterate.
<i>setLanczosOneSide([flag])</i>	Set if the variant of the Lanczos method to be used is one-sided or two-sided.
<i>setMonitor(monitor[, args, kargs])</i>	Append a monitor function to the list of monitors.
<i>setOperators(A[, B])</i>	Set the matrices associated with the singular value problem.
<i>setOptionsPrefix([prefix])</i>	Set the prefix used for searching for all SVD options in the database.
<i>setProblemType(problem_type)</i>	Set the type of the singular value problem.
<i>setSignature([omega])</i>	Set the signature matrix defining a hyperbolic singular value problem.
<i>setStoppingTest(stopping[, args, kargs])</i>	Set a function to decide when to stop the outer iteration of the eigensolver.
<i>setTRLanczosExplicitMatrix([flag])</i>	Set if the matrix $Z = [A; B]$ must be built explicitly.
<i>setTRLanczosGBidiag(bidiag)</i>	Set the bidiagonalization choice to use in the GSVD TRLanczos solver.
<i>setTRLanczosKSP(ksp)</i>	Set a linear solver object associated to the SVD solver.
<i>setTRLanczosLocking(lock)</i>	Toggle between locking and non-locking variants of the method.
<i>setTRLanczosOneSide([flag])</i>	Set if the variant of the method to be used is one-sided or two-sided.
<i>setTRLanczosRestart(keep)</i>	Set the restart parameter for the thick-restart Lanczos method.
<i>setThreshold(thres[, rel])</i>	Set the threshold used in the threshold stopping test.
<i>setTolerances([tol, max_it])</i>	Set the tolerance and maximum iteration count used.
<i>setTrackAll(trackall)</i>	Set flag to compute the residual of all singular triplets.
<i>setType(svd_type)</i>	Set the particular solver to be used in the SVD object.
<i>setUp()</i>	Set up all the necessary internal data structures.

continues on next page

Table 109 – continued from previous page

<code>setWhichSingularTriplets(which)</code>	Set which singular triplets are to be sought.
<code>solve()</code>	Solve the singular value problem.
<code>valuesView([viewer])</code>	Display the computed singular values in a viewer.
<code>vectorsView([viewer])</code>	Output computed singular vectors to a viewer.
<code>view([viewer])</code>	Print the SVD data structure.

Attributes Summary

<code>ds</code>	The direct solver (DS) object associated.
<code>max_it</code>	The maximum iteration count.
<code>problem_type</code>	The type of the eigenvalue problem.
<code>tol</code>	The tolerance.
<code>track_all</code>	Compute the residual norm of all approximate eigenpairs.
<code>transpose_mode</code>	How to handle the transpose of the matrix.
<code>which</code>	The portion of the spectrum to be sought.

Methods Documentation

`appendOptionsPrefix(prefix=None)`

Append to the prefix used for searching for all SVD options in the database.

Logically collective.

Parameters

prefix (*str* / *None*) – The prefix string to prepend to all SVD option requests.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:284 <slepc4py/SLEPc/SVD.pyx#L284>`

`cancelMonitor()`

Clear all monitors for an *SVD* object.

Logically collective.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:912 <slepc4py/SLEPc/SVD.pyx#L912>`

Return type

None

`computeError(i, etype=None)`

Compute the error associated with the *i*-th singular triplet.

Collective.

Compute the error (based on the residual norm) associated with the *i*-th singular triplet.

Parameters

- **i** (*int*) – Index of the solution to be considered.
- **etype** (*ErrorType* / *None*) – The error type to compute.

Returns

The relative error bound, computed in various ways from the residual norm $\sqrt{n_1^2 + n_2^2}$ where

$n_1 = \|Av - \sigma u\|_2$, $n_2 = \|A^T u - \sigma v\|_2$, σ is the singular value, u and v are the left and right singular vectors.

Return type

`float`

Notes

The index `i` should be a value between 0 and `nconv-1` (see `getConverged()`).

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1091 <slepc4py/SLEPc/SVD.pyx#L1091>`

create(*comm=None*)

Create the SVD object.

Collective.

Parameters

comm (*Comm* / *None*) – MPI communicator; if not provided, it defaults to all processes.

Return type

Self

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:186 <slepc4py/SLEPc/SVD.pyx#L186>`

destroy()

Destroy the SVD object.

Collective.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:168 <slepc4py/SLEPc/SVD.pyx#L168>`

Return type

Self

errorView(*etype=None*, *viewer=None*)

Display the errors associated with the computed solution.

Collective.

Display the errors and the eigenvalues.

Parameters

- **etype** (*ErrorType* / *None*) – The error type to compute.
- **viewer** (*petsc4py.PETSc.Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

`None`

Notes

By default, this function checks the error of all eigenpairs and prints the eigenvalues if all of them are below the requested tolerance. If the viewer has format `ASCII_INFO_DETAIL` then a table with eigenvalues and corresponding errors is printed.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1128 <slepc4py/SLEPc/SVD.pyx#L1128>`

getBV()

Get the basis vectors objects associated to the SVD object.

Not collective.

Returns

- **V** (*BV*) – The basis vectors context for right singular vectors.
- **U** (*BV*) – The basis vectors context for left singular vectors.

Return type

tuple[*BV*, *BV*]

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:676 <slepc4py/SLEPc/SVD.pyx#L676>`

getConverged()

Get the number of converged singular triplets.

Not collective.

Returns

Number of converged singular triplets.

Return type

int

Notes

This function should be called after *solve()* has finished.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:981 <slepc4py/SLEPc/SVD.pyx#L981>`

getConvergedReason()

Get the reason why the *solve()* iteration was stopped.

Not collective.

Returns

Negative value indicates diverged, positive value converged.

Return type

ConvergedReason

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:966 <slepc4py/SLEPc/SVD.pyx#L966>`

getConvergenceTest()

Get the method used to compute the error estimate used in the convergence test.

Not collective.

Returns

The method used to compute the error estimate used in the convergence test.

Return type

Conv

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:546 <slepc4py/SLEPc/SVD.pyx#L546>`

getCrossEPS()

Get the eigensolver object associated to the singular value solver.

Collective.

Returns

The eigensolver object.

Return type

EPS

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1202 <slepc4py/SLEPc/SVD.pyx#L1202>`

getCrossExplicitMatrix()

Get the flag indicating if $A^T A$ is built explicitly.

Not collective.

Returns

True if $A^T A$ is built explicitly.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1232 <slepc4py/SLEPc/SVD.pyx#L1232>`

getCyclicEPS()

Get the eigensolver object associated to the singular value solver.

Collective.

Returns

The eigensolver object.

Return type

`EPS`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1260 <slepc4py/SLEPc/SVD.pyx#L1260>`

getCyclicExplicitMatrix()

Get the flag indicating if $H(A)$ is built explicitly.

Not collective.

Get the flag indicating if $H(A) = [0 \ A; A^T \ 0]$ is built explicitly.

Returns

True if $H(A)$ is built explicitly.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1293 <slepc4py/SLEPc/SVD.pyx#L1293>`

getDS()

Get the direct solver associated to the singular value solver.

Not collective.

Returns

The direct solver context.

Return type

`DS`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:713 <slepc4py/SLEPc/SVD.pyx#L713>`

getDimensions()

Get the number of singular values to compute and the dimension of the subspace.

Not collective.

Returns

- `nsv (int)` – Number of singular values to compute.
- `ncv (int)` – Maximum dimension of the subspace to be used by the solver.

- **mpd** (**int**) – Maximum dimension allowed for the projected problem.

Return type

`tuple[int, int, int]`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:609 <slepc4py/SLEPc/SVD.pyx#L609>`

getImplicitTranspose()

Get the mode used to handle the transpose of the matrix associated.

Not collective.

Get the mode used to handle the transpose of the matrix associated with the singular value problem.

Returns

How to handle the transpose (implicitly or not).

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:379 <slepc4py/SLEPc/SVD.pyx#L379>`

getIterationNumber()

Get the current iteration number.

Not collective.

If the call to `solve()` is complete, then it returns the number of iterations carried out by the solution method.

Returns

Iteration number.

Return type

`int`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:948 <slepc4py/SLEPc/SVD.pyx#L948>`

getLanczosOneSide()

Get if the variant of the Lanczos method to be used is one-sided or two-sided.

Not collective.

Returns

True if the method is one-sided.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1333 <slepc4py/SLEPc/SVD.pyx#L1333>`

getMonitor()

Get the list of monitor functions.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:908 <slepc4py/SLEPc/SVD.pyx#L908>`

Return type

`SVDMonitorFunction`

getOperators()

Get the matrices associated with the singular value problem.

Collective.

Returns

- **A** (`petsc4py.PETSc.Mat`) – The matrix associated with the singular value problem.
- **B** (`petsc4py.PETSc.Mat`) – The second matrix in the case of GSVD.

Return type

`tuple[petsc4py.PETSc.Mat, petsc4py.PETSc.Mat] | tuple[petsc4py.PETSc.Mat, None]`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:742 <slepc4py/SLEPc/SVD.pyx#L742>`

getOptionsPrefix()

Get the prefix used for searching for all SVD options in the database.

Not collective.

Returns

The prefix string set for this SVD object.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:242 <slepc4py/SLEPc/SVD.pyx#L242>`

getProblemType()

Get the problem type from the SVD object.

Not collective.

Returns

The problem type that was previously set.

Return type

`ProblemType`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:315 <slepc4py/SLEPc/SVD.pyx#L315>`

getSignature(*omega=None*)

Get the signature matrix defining a hyperbolic singular value problem.

Collective.

Parameters

omega (`petsc4py.PETSc.Vec` | `None`) – Optional vector to store the diagonal elements of the signature matrix.

Returns

A vector containing the diagonal elements of the signature matrix.

Return type

`petsc4py.PETSc.Vec`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:782 <slepc4py/SLEPc/SVD.pyx#L782>`

getSingularTriplet(*i, U=None, V=None*)

Get the *i*-th triplet of the singular value decomposition.

Collective.

Get the *i*-th triplet of the singular value decomposition as computed by `solve()`. The solution consists of the singular value and its left and right singular vectors.

Parameters

- **i** (`int`) – Index of the solution to be obtained.
- **U** (`Vec` | `None`) – Placeholder for the returned left singular vector.

- **V** (*Vec* / *None*) – Placeholder for the returned right singular vector.

Returns

The computed singular value.

Return type

`float`

Notes

The index `i` should be a value between 0 and `nconv-1` (see `getConverged()`). Singular triplets are indexed according to the ordering criterion established with `setWhichSingularTriplets()`.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1052 <slepc4py/SLEPc/SVD.pyx#L1052>`

`getStoppingTest()`

Get the stopping function.

Not collective.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:877 <slepc4py/SLEPc/SVD.pyx#L877>`

Return type

`SVDStoppingFunction`

`getTRLanczosExplicitMatrix()`

Get the flag indicating if $Z = [A; B]$ is built explicitly.

Not collective.

Returns

True if $Z = [A; B]$ is built explicitly.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1538 <slepc4py/SLEPc/SVD.pyx#L1538>`

`getTRLanczosGBidiag()`

Get bidiagonalization choice used in the GSVD TRLanczos solver.

Not collective.

Returns

The bidiagonalization choice.

Return type

`TRLanczosGBidiag`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1404 <slepc4py/SLEPc/SVD.pyx#L1404>`

`getTRLanczosKSP()`

Get the linear solver object associated with the SVD solver.

Collective.

Returns

The linear solver object.

Return type

`petsc4py.PETSc.KSP`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1508 <slepc4py/SLEPc/SVD.pyx#L1508>`

getTRLanczosLocking()

Get the locking flag used in the thick-restart Lanczos method.

Not collective.

Returns

The locking flag.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1480 <slepc4py/SLEPc/SVD.pyx#L1480>`

getTRLanczosOneSide()

Get if the variant of the method to be used is one-sided or two-sided.

Not collective.

Get if the variant of the thick-restart Lanczos method to be used is one-sided or two-sided.

Returns

True if the method is one-sided.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1372 <slepc4py/SLEPc/SVD.pyx#L1372>`

getTRLanczosRestart()

Get the restart parameter used in the thick-restart Lanczos method.

Not collective.

Returns

The number of vectors to be kept at restart.

Return type

`float`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1441 <slepc4py/SLEPc/SVD.pyx#L1441>`

getThreshold()

Get the threshold used in the threshold stopping test.

Not collective.

Returns

- **thres** (`float`) – The threshold.
- **rel** (`bool`) – Whether the threshold is relative or not.

Return type

`tuple[float, bool]`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:451 <slepc4py/SLEPc/SVD.pyx#L451>`

getTolerances()

Get the tolerance and maximum iteration count.

Not collective.

Get the tolerance and maximum iteration count used by the default SVD convergence tests.

Returns

- `tol` (`float`) – The convergence tolerance.
- `max_it` (`int`) – The maximum number of iterations

Return type

`tuple[float, int]`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:498 <slepc4py/SLEPc/SVD.pyx#L498>`

getTrackAll()

Get the flag indicating if all residual norms must be computed or not.

Not collective.

Returns

Whether the solver compute all residuals or not.

Return type

`bool`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:577 <slepc4py/SLEPc/SVD.pyx#L577>`

getType()

Get the SVD type of this object.

Not collective.

Returns

The solver currently being used.

Return type

`str`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:227 <slepc4py/SLEPc/SVD.pyx#L227>`

getValue(i)

Get the i-th singular value as computed by `solve()`.

Collective.

Parameters

`i` (`int`) – Index of the solution to be obtained.

Returns

The computed singular value.

Return type

`float`

Notes

The index `i` should be a value between 0 and `nconv-1` (see `getConverged()`). Singular triplets are indexed according to the ordering criterion established with `setWhichSingularTriplets()`.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1000 <slepc4py/SLEPc/SVD.pyx#L1000>`

getVectors(i, U, V)

Get the i-th left and right singular vectors as computed by `solve()`.

Collective.

Parameters

- `i` (`int`) – Index of the solution to be obtained.

- **U** (*Vec*) – Placeholder for the returned left singular vector.
- **V** (*Vec*) – Placeholder for the returned right singular vector.

Return type

None

Notes

The index *i* should be a value between 0 and *nconv* - 1 (see *getConverged()*). Singular triplets are indexed according to the ordering criterion established with *setWhichSingularTriplets()*.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1027 <slepc4py/SLEPc/SVD.pyx#L1027>`

getWhichSingularTriplets()

Get which singular triplets are to be sought.

Not collective.

Returns

The singular values to be sought (either largest or smallest).

Return type

Which

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:422 <slepc4py/SLEPc/SVD.pyx#L422>`

isGeneralized()

Tell if the SVD corresponds to a generalized singular value problem.

Not collective.

Tell whether the SVD object corresponds to a generalized singular value problem.

Returns

True if two matrices were set with *setOperators()*.

Return type

bool

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:344 <slepc4py/SLEPc/SVD.pyx#L344>`

isHyperbolic()

Tell whether the SVD object corresponds to a hyperbolic singular value problem.

Not collective.

Returns

True if the problem was specified as hyperbolic.

Return type

bool

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:362 <slepc4py/SLEPc/SVD.pyx#L362>`

reset()

Reset the SVD object.

Collective.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:178 <slepc4py/SLEPc/SVD.pyx#L178>`

Return type

None

setBV(*V*, *U=None*)

Set basis vectors objects associated to the SVD solver.

Collective.

Parameters

- **V** (*BV*) – The basis vectors context for right singular vectors.
- **U** (*BV* / *None*) – The basis vectors context for left singular vectors.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:696 <slepc4py/SLEPc/SVD.pyx#L696>`

setConvergenceTest(*conv*)

Set how to compute the error estimate used in the convergence test.

Logically collective.

Parameters

conv (*Conv*) – The method used to compute the error estimate used in the convergence test.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:562 <slepc4py/SLEPc/SVD.pyx#L562>`

setCrossEPS(*eps*)

Set an eigensolver object associated to the singular value solver.

Collective.

Parameters

eps (*EPS*) – The eigensolver object.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1189 <slepc4py/SLEPc/SVD.pyx#L1189>`

setCrossExplicitMatrix(*flag=True*)

Set if the eigensolver operator $A^T A$ must be computed.

Logically collective.

Parameters

flag (*bool*) – True to build $A^T A$ explicitly.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1218 <slepc4py/SLEPc/SVD.pyx#L1218>`

setCyclicEPS(*eps*)

Set an eigensolver object associated to the singular value solver.

Collective.

Parameters

eps (*EPS*) – The eigensolver object.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1247 <slepc4py/SLEPc/SVD.pyx#L1247>`

setCyclicExplicitMatrix(*flag=True*)

Set if the eigensolver operator $H(A)$ must be computed explicitly.

Logically collective.

Set if the eigensolver operator $H(A) = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}$ must be computed explicitly.

Parameters

flag (*bool*) – True if $H(A)$ is built explicitly.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1276 <slepc4py/SLEPc/SVD.pyx#L1276>`

setDS(*ds*)

Set a direct solver object associated to the singular value solver.

Collective.

Parameters

ds (*DS*) – The direct solver context.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:729 <slepc4py/SLEPc/SVD.pyx#L729>`

setDimensions(*nsv=None, ncv=None, mpd=None*)

Set the number of singular values to compute and the dimension of the subspace.

Logically collective.

Parameters

- **nsv** (*int* / *None*) – Number of singular values to compute.
- **ncv** (*int* / *None*) – Maximum dimension of the subspace to be used by the solver.
- **mpd** (*int* / *None*) – Maximum dimension allowed for the projected problem.

Return type

None

Notes

Use *DECIDE* for *ncv* and *mpd* to assign a reasonably good value, which is dependent on the solution method.

The parameters *ncv* and *mpd* are intimately related, so that the user is advised to set one of them at most. Normal usage is the following:

- In cases where *nsv* is small, the user sets *ncv* (a reasonable default is $2 * nsv$).
- In cases where *nsv* is large, the user sets *mpd*.

The value of *ncv* should always be between *nsv* and (*nsv* + *mpd*), typically $ncv = nsv + mpd$. If *nsv* is not too large, $mpd = nsv$ is a reasonable choice, otherwise a smaller value should be used.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:630 <slepc4py/SLEPc/SVD.pyx#L630>`

setFromOptions()

Set SVD options from the options database.

Collective.

This routine must be called before `setUp()` if the user is to be allowed to set the solver type.

Notes

To see all options, run your program with the `-help` option.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:299 <slepc4py/SLEPc/SVD.pyx#L299>`

Return type

`None`

setImplicitTranspose(mode)

Set how to handle the transpose of the matrix associated.

Logically collective.

Set how to handle the transpose of the matrix associated with the singular value problem.

Parameters

- **impl** – How to handle the transpose (implicitly or not).
- **mode** (`bool`)

Return type

`None`

Notes

By default, the transpose of the matrix is explicitly built (if the matrix has defined the `MatTranspose` operation).

If this flag is set to true, the solver does not build the transpose, but handles it implicitly via `MatMultTranspose()`.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:397 <slepc4py/SLEPc/SVD.pyx#L397>`

setInitialSpace(spaceright=None, spaceleft=None)

Set the initial spaces from which the SVD solver starts to iterate.

Collective.

Parameters

- **spaceright** (`list[Vec]` | `None`) – The right initial space.
- **spaceleft** (`list[Vec]` | `None`) – The left initial space.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:823 <slepc4py/SLEPc/SVD.pyx#L823>`

setLanczosOneSide(flag=True)

Set if the variant of the Lanczos method to be used is one-sided or two-sided.

Logically collective.

Parameters

flag (`bool`) – True if the method is one-sided.

Return type

None

Notes

By default, a two-sided variant is selected, which is sometimes slightly more robust. However, the one-sided variant is faster because it avoids the orthogonalization associated to left singular vectors. It also saves the memory required for storing such vectors.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1311 <slepc4py/SLEPc/SVD.pyx#L1311>`

setMonitor(*monitor*, *args*=None, *kargs*=None)

Append a monitor function to the list of monitors.

Logically collective.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:887 <slepc4py/SLEPc/SVD.pyx#L887>`

Parameters

- **monitor** (*SVDMonitorFunction* | *None*)
- **args** (*tuple*[*Any*, ...] | *None*)
- **kargs** (*dict*[*str*, *Any*] | *None*)

Return type

None

setOperators(*A*, *B*=None)

Set the matrices associated with the singular value problem.

Collective.

Parameters

- **A** (*Mat*) – The matrix associated with the singular value problem.
- **B** (*Mat* | *None*) – The second matrix in the case of GSVD; if not provided, a usual SVD is assumed.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:765 <slepc4py/SLEPc/SVD.pyx#L765>`

setOptionsPrefix(*prefix*=None)

Set the prefix used for searching for all SVD options in the database.

Logically collective.

Parameters

prefix (*str* | *None*) – The prefix string to prepend to all SVD option requests.

Return type

None

Notes

A hyphen (-) must NOT be given at the beginning of the prefix name. The first character of all runtime options is AUTOMATICALLY the hyphen.

For example, to distinguish between the runtime options for two different SVD contexts, one could call:

```
S1.setOptionsPrefix("svd1_")
S2.setOptionsPrefix("svd2_")
```

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:257 <slepc4py/SLEPc/SVD.pyx#L257>`

setProblemType(*problem_type*)

Set the type of the singular value problem.

Logically collective.

Parameters

problem_type (`ProblemType`) – The problem type to be set.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:330 <slepc4py/SLEPc/SVD.pyx#L330>`

setSignature(*omega=None*)

Set the signature matrix defining a hyperbolic singular value problem.

Collective.

Parameters

omega (`Vec` / `None`) – A vector containing the diagonal elements of the signature matrix.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:807 <slepc4py/SLEPc/SVD.pyx#L807>`

setStoppingTest(*stopping, args=None, kargs=None*)

Set a function to decide when to stop the outer iteration of the eigensolver.

Logically collective.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:857 <slepc4py/SLEPc/SVD.pyx#L857>`

Parameters

- **stopping** (`SVDStoppingFunction` / `None`)
- **args** (`tuple`[`Any`, ...] / `None`)
- **kargs** (`dict`[`str`, `Any`] / `None`)

Return type

`None`

setTRLanczosExplicitMatrix(*flag=True*)

Set if the matrix $Z = [A; B]$ must be built explicitly.

Logically collective.

Parameters

flag (`bool`) – True if $Z = [A; B]$ is built explicitly.

Return type

`None`

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1524 <slepc4py/SLEPc/SVD.pyx#L1524>`

setTRLanczosGBidiag(*bidiag*)

Set the bidiagonalization choice to use in the GSVD TRLanczos solver.

Logically collective.

Parameters

bidiag (TRLanczosGBidiag) – The bidiagonalization choice.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1390 <slepc4py/SLEPc/SVD.pyx#L1390>`

setTRLanczosKSP(*ksp*)

Set a linear solver object associated to the SVD solver.

Collective.

Parameters

ksp (*petsc4py.PETSc.KSP*) – The linear solver object.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1495 <slepc4py/SLEPc/SVD.pyx#L1495>`

setTRLanczosLocking(*lock*)

Toggle between locking and non-locking variants of the method.

Logically collective.

Toggle between locking and non-locking variants of the thick-restart Lanczos method.

Parameters

lock (*bool*) – True if the locking variant must be selected.

Return type

None

Notes

The default is to lock converged singular triplets when the method restarts. This behavior can be changed so that all directions are kept in the working subspace even if already converged to working accuracy (the non-locking variant).

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1456 <slepc4py/SLEPc/SVD.pyx#L1456>`

setTRLanczosOneSide(*flag=True*)

Set if the variant of the method to be used is one-sided or two-sided.

Logically collective.

Set if the variant of the thick-restart Lanczos method to be used is one-sided or two-sided.

Parameters

flag (*bool*) – True if the method is one-sided.

Return type

None

Notes

By default, a two-sided variant is selected, which is sometimes slightly more robust. However, the one-sided variant is faster because it avoids the orthogonalization associated to left singular vectors.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1348 <slepc4py/SLEPc/SVD.pyx#L1348>`

setTRLanczosRestart(*keep*)

Set the restart parameter for the thick-restart Lanczos method.

Logically collective.

Set the restart parameter for the thick-restart Lanczos method, in particular the proportion of basis vectors that must be kept after restart.

Parameters

keep (*float*) – The number of vectors to be kept at restart.

Return type

None

Notes

Allowed values are in the range [0.1,0.9]. The default is 0.5.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1419 <slepc4py/SLEPc/SVD.pyx#L1419>`

setThreshold(*thres*, *rel=False*)

Set the threshold used in the threshold stopping test.

Logically collective.

Parameters

- **thres** (*float*) – The threshold.
- **rel** (*bool*) – Whether the threshold is relative or not.

Return type

None

Notes

This function internally sets a special stopping test based on the threshold, where singular values are computed in sequence until one of the computed singular values is below/above the threshold (depending on whether largest or smallest singular values are computed).

In the case of largest singular values, the threshold can be made relative with respect to the largest singular value (i.e., the matrix norm).

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:469 <slepc4py/SLEPc/SVD.pyx#L469>`

setTolerances(*tol=None*, *max_it=None*)

Set the tolerance and maximum iteration count used.

Logically collective.

Set the tolerance and maximum iteration count used by the default SVD convergence tests.

Parameters

- **tol** (*float* / *None*) – The convergence tolerance.
- **max_it** (*int* / *None*) – The maximum number of iterations

Return type

None

Notes

Use *DECIDE* for *max_it* to assign a reasonably good value, which is dependent on the solution method.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:519 <slepc4py/SLEPc/SVD.pyx#L519>`

setTrackAll(*trackall*)

Set flag to compute the residual of all singular triplets.

Logically collective.

Set if the solver must compute the residual of all approximate singular triplets or not.

Parameters

trackall (*bool*) – Whether compute all residuals or not.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:592 <slepc4py/SLEPc/SVD.pyx#L592>`

setType(*svd_type*)

Set the particular solver to be used in the SVD object.

Logically collective.

Parameters

svd_type (*Type* / *str*) – The solver to be used.

Return type

None

Notes

See *SVD.Type* for available methods. The default is CROSS. Normally, it is best to use *setFromOptions()* and then set the SVD type from the options database rather than by using this routine. Using the options database provides the user with maximum flexibility in evaluating the different available methods.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:203 <slepc4py/SLEPc/SVD.pyx#L203>`

setUp()

Set up all the necessary internal data structures.

Collective.

Set up all the internal data structures necessary for the execution of the singular value solver.

Notes

This function need not be called explicitly in most cases, since *solve()* calls it. It can be useful when one wants to measure the set-up time separately from the solve time.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:923 <slepc4py/SLEPc/SVD.pyx#L923>`

Return type

None

setWhichSingularTriplets(*which*)

Set which singular triplets are to be sought.

Logically collective.

Parameters

which (*Which*) – The singular values to be sought (either largest or smallest).

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:437 <slepc4py/SLEPc/SVD.pyx#L437>`

solve()

Solve the singular value problem.

Collective.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:940 <slepc4py/SLEPc/SVD.pyx#L940>`

Return type

None

valuesView(*viewer=None*)

Display the computed singular values in a viewer.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1157 <slepc4py/SLEPc/SVD.pyx#L1157>`

vectorsView(*viewer=None*)

Output computed singular vectors to a viewer.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1172 <slepc4py/SLEPc/SVD.pyx#L1172>`

view(*viewer=None*)

Print the SVD data structure.

Collective.

Parameters

viewer (*Viewer* / *None*) – Visualization context; if not provided, the standard output is used.

Return type

None

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:153 <slepc4py/SLEPc/SVD.pyx#L153>`

Attributes Documentation

ds

The direct solver (DS) object associated.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1599 <slepc4py/SLEPc/SVD.pyx#L1599>`

max_it

The maximum iteration count.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1585 <slepc4py/SLEPc/SVD.pyx#L1585>`

problem_type

The type of the eigenvalue problem.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1557 <slepc4py/SLEPc/SVD.pyx#L1557>`

tol

The tolerance.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1578 <slepc4py/SLEPc/SVD.pyx#L1578>`

track_all

Compute the residual norm of all approximate eigenpairs.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1592 <slepc4py/SLEPc/SVD.pyx#L1592>`

transpose_mode

How to handle the transpose of the matrix.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1564 <slepc4py/SLEPc/SVD.pyx#L1564>`

which

The portion of the spectrum to be sought.

:sources: `Source code at slepc4py/SLEPc/SVD.pyx:1571 <slepc4py/SLEPc/SVD.pyx#L1571>`

slepc4py.SLEPc.Sys

class slepc4py.SLEPc.Sys

Bases: `object`

Sys.

Methods Summary

<code>getVersion([devel, date, author])</code>	Return SLEPc version information.
<code>getVersionInfo()</code>	Return SLEPc version information.
<code>hasExternalPackage(package)</code>	Return whether SLEPc has support for external package.
<code>isFinalized()</code>	Return whether SLEPc has been finalized.
<code>isInitialized()</code>	Return whether SLEPc has been initialized.

Methods Documentation

classmethod getVersion(*devel=False, date=False, author=False*)

Return SLEPc version information.

Not collective.

Parameters

- **devel** (*bool*) – Additionally, return whether using an in-development version.
- **date** (*bool*) – Additionally, return date information.
- **author** (*bool*) – Additionally, return author information.

Returns

- **major** (*int*) – Major version number.
- **minor** (*int*) – Minor version number.
- **micro** (*int*) – Micro (or patch) version number.

Return type

`tuple[int, int, int]`

See also

`SlepcGetVersion`, `SlepcGetVersionNumber`

:sources: `Source code at slepc4py/SLEPc/Sys.pyx:6 <slepc4py/SLEPc/Sys.pyx#L6>`

`classmethod getVersionInfo()`

Return SLEPc version information.

Not collective.

Returns

info – Dictionary with version information.

Return type

`dict`

See also

`SlepcGetVersion`, `SlepcGetVersionNumber`

:sources: `Source code at slepc4py/SLEPc/Sys.pyx:62 <slepc4py/SLEPc/Sys.pyx#L62>`

`classmethod hasExternalPackage(package)`

Return whether SLEPc has support for external package.

Not collective.

Parameters

package (*str*) – The external package name.

Return type

`bool`

See also

`SlepcHasExternalPackage`

:sources: `Source code at slepc4py/SLEPc/Sys.pyx:116 <slepc4py/SLEPc/Sys.pyx#L116>`

classmethod `isFinalized()`

Return whether SLEPc has been finalized.

Not collective.

See also

`isInitialized`

:sources: `Source code at slepc4py/SLEPc/Sys.pyx:101 <slepc4py/SLEPc/Sys.pyx#L101>`

Return type

`bool`

classmethod `isInitialized()`

Return whether SLEPc has been initialized.

Not collective.

See also

`isFinalized`

:sources: `Source code at slepc4py/SLEPc/Sys.pyx:88 <slepc4py/SLEPc/Sys.pyx#L88>`

Return type

`bool`

slepc4py.SLEPc.Util

class `slepc4py.SLEPc.Util`

Bases: `object`

Util.

Methods Summary

<i><code>createMatBSE</code></i> (<code>R</code> , <code>C</code>)	Create a matrix that can be used to define a BSE type problem.
<i><code>createMatHamiltonian</code></i> (<code>A</code> , <code>B</code> , <code>C</code>)	Create matrix to be used for a structured Hamiltonian eigenproblem.

Methods Documentation

classmethod `createMatBSE(R, C)`

Create a matrix that can be used to define a BSE type problem.

Collective.

Create a matrix that can be used to define a structured eigenvalue problem of type BSE (Bethe-Salpeter Equation).

Parameters

- `R` (*`petsc4py.PETSc.Mat`*) – The matrix for the diagonal block (resonant).

- **C** (*petsc4py.PETSc.Mat*) – The matrix for the off-diagonal block (coupling).

Returns

The matrix with the block form $H = [R \ C; -C^H \ -R^T]$.

Return type

petsc4py.PETSc.Mat

:sources: [Source code at slepc4py/SLEPc/Util.pyx:6 <slepc4py/SLEPc/Util.pyx#L6>](#)

classmethod **createMatHamiltonian**(*A, B, C*)

Create matrix to be used for a structured Hamiltonian eigenproblem.

Collective.

Parameters

- **A** (*petsc4py.PETSc.Mat*) – The matrix for (0,0) block.
- **B** (*petsc4py.PETSc.Mat*) – The matrix for (0,1) block, must be real symmetric or Hermitian.
- **C** (*petsc4py.PETSc.Mat*) – The matrix for (1,0) block, must be real symmetric or Hermitian.

Returns

The matrix with the block form $H = [AB; C - A^*]$.

Return type

petsc4py.PETSc.Mat

:sources: [Source code at slepc4py/SLEPc/Util.pyx:32 <slepc4py/SLEPc/Util.pyx#L32>](#)

Attributes

<i>DECIDE</i>	Constant DECIDE of type <i>int</i>
<i>DEFAULT</i>	Constant DEFAULT of type <i>int</i>
<i>DETERMINE</i>	Constant DETERMINE of type <i>int</i>
<i>CURRENT</i>	Constant CURRENT of type <i>int</i>

slepc4py.SLEPc.DECIDE

slepc4py.SLEPc.DECIDE: *int* = DECIDE

Constant DECIDE of type *int*

slepc4py.SLEPc.DEFAULT

slepc4py.SLEPc.DEFAULT: *int* = DEFAULT

Constant DEFAULT of type *int*

slepc4py.SLEPc.DETERMINE

slepc4py.SLEPc.DETERMINE: *int* = DETERMINE

Constant DETERMINE of type *int*

slepc4py.SLEPc.CURRENT

slepc4py.SLEPc.CURRENT: `int` = CURRENT

Constant CURRENT of type `int`

1.6 slepc4py demos

Standard symmetric eigenproblem for the Laplacian operator in 1-D

This tutorial is intended for basic use of slepc4py. For more advanced use, the reader is referred to SLEPc tutorials as well as to slepc4py reference documentation.

The source code for this demo can be [downloaded here](#)

The first thing to do is initialize the libraries. This is normally not required, as it is done automatically at import time. However, if you want to gain access to the facilities for accessing command-line options, the following lines must be executed by the main script prior to any petsc4py or slepc4py calls:

```
import sys, slepc4py
slepc4py.init(sys.argv)
```

Next, we have to import the relevant modules. Normally, both PETSc and SLEPc modules have to be imported in all slepc4py programs. It may be useful to import NumPy as well:

```
from petsc4py import PETSc
from slepc4py import SLEPc
import numpy
```

At this point, we can use any petsc4py and slepc4py operations. For instance, the following lines allow the user to specify an integer command-line argument `n` with a default value of 30 (see the next section for example usage of command-line options):

```
opts = PETSc.Options()
n = opts.getInt('n', 30)
```

It is necessary to build a matrix to define an eigenproblem (or two in the case of generalized eigenproblems). The following fragment of code creates the matrix object and then fills the non-zero elements one by one. The matrix of this particular example is tridiagonal, with value 2 in the diagonal, and -1 in off-diagonal positions. See petsc4py documentation for details about matrix objects:

```
A = PETSc.Mat(); A.create()
A.setSizes([n, n])
A.setFromOptions()

rstart, rend = A.getOwnershipRange()

# first row
if rstart == 0:
    A[0, :2] = [2, -1]
    rstart += 1
# last row
if rend == n:
    A[n-1, -2:] = [-1, 2]
    rend -= 1
# other rows
```

(continues on next page)

(continued from previous page)

```
for i in range(rstart, rend):
    A[i, i-1:i+2] = [-1, 2, -1]

A.assemble()
```

The solver object is created in a similar way as other objects in `petsc4py`:

```
E = SLEPc.EPS(); E.create()
```

Once the object is created, the eigenvalue problem must be specified. At least one matrix must be provided. The problem type must be indicated as well, in this case it is HEP (Hermitian eigenvalue problem). Apart from these, other settings could be provided here (for instance, the tolerance for the computation). After all options have been set, the user should call the `setFromOptions()` operation, so that any options specified at run time in the command line are passed to the solver object:

```
E.setOperators(A)
E.setProblemType(SLEPc.EPS.ProblemType.HEP)

history = []
def monitor(eps, its, nconv, eig, err):
    if nconv < len(err): history.append(err[nconv])
E.setMonitor(monitor)

E.setFromOptions()
```

After that, the `solve()` method will run the selected eigensolver, keeping the solution stored internally:

```
E.solve()
```

Once the computation has finished, we are ready to print the results. First, some informative data can be retrieved from the solver object:

```
Print = PETSc.Sys.Print

Print()
Print("*****")
Print("*** SLEPc Solution Results ***")
Print("*****")
Print()

its = E.getIterationNumber()
Print( "Number of iterations of the method: %d" % its )

eps_type = E.getType()
Print( "Solution method: %s" % eps_type )

nev, ncv, mpd = E.getDimensions()
Print( "Number of requested eigenvalues: %d" % nev )

tol, maxit = E.getTolerances()
Print( "Stopping condition: tol=%.4g, maxit=%d" % (tol, maxit) )
```

For retrieving the solution, it is necessary to find out how many eigenpairs have converged to the requested precision:

```
nconv = E.getConverged()
Print( "Number of converged eigenpairs %d" % nconv )
```

For each of the `nconv` eigenpairs, we can retrieve the eigenvalue `k`, and the eigenvector, which is represented by means of two `petsc4py` vectors `vr` and `vi` (the real and imaginary part of the eigenvector, since for real matrices the eigenvalue and eigenvector may be complex). We also compute the corresponding relative errors in order to make sure that the computed solution is indeed correct:

```
if nconv > 0:
    # Create the results vectors
    v, _ = A.createVecs()
    #
    Print()
    Print("          k          ||Ax-kx||/||kx|| ")
    Print("-----")
    for i in range(nconv):
        k = E.getEigenpair(i, v)
        error = E.computeError(i)
        Print( " %12f          %12g" % (k, error) )
    Print()
```

Example of command-line usage

Now we illustrate how to specify command-line options in order to extract the full potential of `slepc4py`.

A simple execution of the `demo/ex1.py` script will result in the following output:

```
$ python demo/ex1.py

*****
*** SLEPc Solution Results ***
*****

Number of iterations of the method: 4
Solution method: krylovschur
Number of requested eigenvalues: 1
Stopping condition: tol=1e-07, maxit=100
Number of converged eigenpairs 4

          k          ||Ax-kx||/||kx||
-----
    3.989739    5.76012e-09
    3.959060    1.41957e-08
    3.908279    6.74118e-08
    3.837916    8.34269e-08
```

For specifying different setting for the solver parameters, we can use SLEPc command-line options with the `-eps` prefix. For instance, to change the number of requested eigenvalues and the tolerance:

```
$ python demo/ex1.py -eps_nev 10 -eps_tol 1e-11
```

The method used by the solver object can also be set at run time:

```
$ python demo/ex1.py -eps_type subspace
```

All the above settings can also be changed within the source code by making use of the appropriate `slepc4py` method. Since options can be set from within the code and the command-line, it is often useful to view the particular settings that are currently being used:

```
$ python demo/ex1.py -eps_view
```

EPS Object: 1 MPI process

type: krylovschur

*50% of basis vectors kept after restart
using the locking variant*

problem type: symmetric eigenvalue problem

selected portion of the spectrum: largest eigenvalues in magnitude

number of eigenvalues (nev): 1

number of column vectors (ncv): 16

maximum dimension of projected problem (mpd): 16

maximum number of iterations: 100

tolerance: 1e-08

convergence test: relative to the eigenvalue

BV Object: 1 MPI process

type: mat

17 columns of global length 30

orthogonalization method: classical Gram-Schmidt

orthogonalization refinement: if needed (eta: 0.7071)

block orthogonalization method: GS

doing matmult as a single matrix-matrix product

DS Object: 1 MPI process

type: hep

solving the problem with: Implicit QR method (_steqr)

ST Object: 1 MPI process

type: shift

shift: 0

number of matrices: 1

Note that for computing eigenvalues of smallest magnitude we can use the option `-eps_smallest_magnitude`, but for interior eigenvalues things are not so straightforward. One possibility is to try with harmonic extraction, for instance to get the eigenvalues closest to 0.6:

```
$ python demo/ex1.py -eps_harmonic -eps_target 0.6
```

Depending on the problem, harmonic extraction may fail to converge. In those cases, it is necessary to specify a spectral transformation other than the default. In the command-line, this is indicated with the `-st_` prefix. For example, shift-and-invert with a value of the shift equal to 0.6 would be:

```
$ python demo/ex1.py -st_type sinvert -eps_target 0.6
```

Python Module Index

S

`slepc4py`, [10](#)
`slepc4py.SLEPc`, [15](#)
`slepc4py.typing`, [11](#)

Index

A

A (*slepc4py.SLEPc.DS.MatType attribute*), 42
ABS (*slepc4py.SLEPc.EPS.Conv attribute*), 60
ABS (*slepc4py.SLEPc.NEP.Conv attribute*), 148
ABS (*slepc4py.SLEPc.PEP.Conv attribute*), 187
ABS (*slepc4py.SLEPc.SVD.Conv attribute*), 253
ABSOLUTE (*slepc4py.SLEPc.EPS.ErrorType attribute*), 62
ABSOLUTE (*slepc4py.SLEPc.NEP.ErrorType attribute*), 150
ABSOLUTE (*slepc4py.SLEPc.PEP.ErrorType attribute*), 189
ABSOLUTE (*slepc4py.SLEPc.SVD.ErrorType attribute*), 255
ADD (*slepc4py.SLEPc.FN.CombineType attribute*), 117
ALL (*slepc4py.SLEPc.EPS.Which attribute*), 69
ALL (*slepc4py.SLEPc.NEP.Which attribute*), 154
ALL (*slepc4py.SLEPc.PEP.Which attribute*), 195
allocate() (*slepc4py.SLEPc.DS method*), 47
ALWAYS (*slepc4py.SLEPc.BV.OrthogRefineType attribute*), 17
appendOptionsPrefix() (*slepc4py.SLEPc.BV method*), 21
appendOptionsPrefix() (*slepc4py.SLEPc.DS method*), 47
appendOptionsPrefix() (*slepc4py.SLEPc.EPS method*), 75
appendOptionsPrefix() (*slepc4py.SLEPc.FN method*), 120
appendOptionsPrefix() (*slepc4py.SLEPc.LME method*), 130
appendOptionsPrefix() (*slepc4py.SLEPc.MFN method*), 140
appendOptionsPrefix() (*slepc4py.SLEPc.NEP method*), 158
appendOptionsPrefix() (*slepc4py.SLEPc.PEP method*), 199
appendOptionsPrefix() (*slepc4py.SLEPc.RG method*), 228
appendOptionsPrefix() (*slepc4py.SLEPc.ST method*), 239
appendOptionsPrefix() (*slepc4py.SLEPc.SVD method*), 261
apply() (*slepc4py.SLEPc.ST method*), 239
applyHermitianTranspose() (*slepc4py.SLEPc.ST method*), 240
applyMat() (*slepc4py.SLEPc.ST method*), 240
applyMatrix() (*slepc4py.SLEPc.BV method*), 21
applyResolvent() (*slepc4py.SLEPc.NEP method*), 158
applyTranspose() (*slepc4py.SLEPc.ST method*), 240
ARCHFLAGS, 9

ARNOLDI (*slepc4py.SLEPc.EPS.Type attribute*), 68
ARPACK (*slepc4py.SLEPc.EPS.Type attribute*), 68
ArrayComplex (*in module slepc4py.typing*), 12
ArrayInt (*in module slepc4py.typing*), 12
ArrayReal (*in module slepc4py.typing*), 12
ArrayScalar (*in module slepc4py.typing*), 12

B

B (*slepc4py.SLEPc.DS.MatType attribute*), 42
BACKWARD (*slepc4py.SLEPc.EPS.ErrorType attribute*), 62
BACKWARD (*slepc4py.SLEPc.NEP.ErrorType attribute*), 150
BACKWARD (*slepc4py.SLEPc.PEP.ErrorType attribute*), 189
Balance (*class in slepc4py.SLEPc.EPS*), 58
BASIC (*slepc4py.SLEPc.EPS.Stop attribute*), 66
BASIC (*slepc4py.SLEPc.NEP.Stop attribute*), 152
BASIC (*slepc4py.SLEPc.PEP.Stop attribute*), 193
BASIC (*slepc4py.SLEPc.SVD.Stop attribute*), 256
Basis (*class in slepc4py.SLEPc.PEP*), 186
block_size (*slepc4py.SLEPc.DS attribute*), 57
BLOPEX (*slepc4py.SLEPc.EPS.Type attribute*), 68
BOTH (*slepc4py.SLEPc.PEP.Scale attribute*), 192
BSE (*slepc4py.SLEPc.EPS.ProblemType attribute*), 65
BV (*class in slepc4py.SLEPc*), 15
bv (*slepc4py.SLEPc.EPS attribute*), 115
bv (*slepc4py.SLEPc.LME attribute*), 137
bv (*slepc4py.SLEPc.MFN attribute*), 146
bv (*slepc4py.SLEPc.NEP attribute*), 184
bv (*slepc4py.SLEPc.PEP attribute*), 224
BVSVDMethod (*class in slepc4py.SLEPc*), 41

C

C (*slepc4py.SLEPc.DS.MatType attribute*), 42
CAA (*slepc4py.SLEPc.NEP.CISSExtraction attribute*), 148
CAA (*slepc4py.SLEPc.PEP.CISSExtraction attribute*), 187
cancelMonitor() (*slepc4py.SLEPc.EPS method*), 75
cancelMonitor() (*slepc4py.SLEPc.LME method*), 130
cancelMonitor() (*slepc4py.SLEPc.MFN method*), 140
cancelMonitor() (*slepc4py.SLEPc.NEP method*), 158
cancelMonitor() (*slepc4py.SLEPc.PEP method*), 199
cancelMonitor() (*slepc4py.SLEPc.SVD method*), 261
canUseConjugates() (*slepc4py.SLEPc.RG method*), 228
CAYLEY (*slepc4py.SLEPc.ST.Type attribute*), 237
CGS (*slepc4py.SLEPc.BV.OrthogType attribute*), 17
CHASE (*slepc4py.SLEPc.EPS.Type attribute*), 68
CHEBYSHEV (*slepc4py.SLEPc.EPS.CISSQuadRule attribute*), 60

CHEBYSHEV (*slepc4py.SLEPc.RG.QuadRule* attribute), 226

CHEBYSHEV (*slepc4py.SLEPc.ST.FilterType* attribute), 236

CHEBYSHEV (*slepc4py.SLEPc.STFilterType* attribute), 252

CHEBYSHEV1 (*slepc4py.SLEPc.PEP.Basis* attribute), 186

CHEBYSHEV2 (*slepc4py.SLEPc.PEP.Basis* attribute), 186

checkInside() (*slepc4py.SLEPc.RG* method), 228

CHOL (*slepc4py.SLEPc.BV.OrthogBlockType* attribute), 16

CISS (*slepc4py.SLEPc.EPS.Type* attribute), 68

CISS (*slepc4py.SLEPc.NEP.Type* attribute), 153

CISS (*slepc4py.SLEPc.PEP.Type* attribute), 194

CISSExtraction (class in *slepc4py.SLEPc.EPS*), 59

CISSExtraction (class in *slepc4py.SLEPc.NEP*), 147

CISSExtraction (class in *slepc4py.SLEPc.PEP*), 186

CISSQuadRule (class in *slepc4py.SLEPc.EPS*), 59

column_size (*slepc4py.SLEPc.BV* attribute), 40

COMBINE (*slepc4py.SLEPc.FN.Type* attribute), 118

CombineType (class in *slepc4py.SLEPc.FN*), 117

compact (*slepc4py.SLEPc.DS* attribute), 57

complement (*slepc4py.SLEPc.RG* attribute), 235

COMPOSE (*slepc4py.SLEPc.FN.CombineType* attribute), 117

computeBoundingBox() (*slepc4py.SLEPc.RG* method), 228

computeContour() (*slepc4py.SLEPc.RG* method), 229

computeError() (*slepc4py.SLEPc.EPS* method), 75

computeError() (*slepc4py.SLEPc.LME* method), 130

computeError() (*slepc4py.SLEPc.NEP* method), 159

computeError() (*slepc4py.SLEPc.PEP* method), 199

computeError() (*slepc4py.SLEPc.SVD* method), 261

computeQuadrature() (*slepc4py.SLEPc.RG* method), 229

cond() (*slepc4py.SLEPc.DS* method), 47

CONDENSED (*slepc4py.SLEPc.DS.StateType* attribute), 44

CONSTANT (*slepc4py.SLEPc.EPS.PowerShiftType* attribute), 65

CONTIGUOUS (*slepc4py.SLEPc.BV.Type* attribute), 18

Conv (class in *slepc4py.SLEPc.EPS*), 60

Conv (class in *slepc4py.SLEPc.NEP*), 148

Conv (class in *slepc4py.SLEPc.PEP*), 187

Conv (class in *slepc4py.SLEPc.SVD*), 253

CONVERGED_ITERATING (*slepc4py.SLEPc.EPS.ConvergedReason* attribute), 61

CONVERGED_ITERATING (*slepc4py.SLEPc.LME.ConvergedReason* attribute), 127

CONVERGED_ITERATING (*slepc4py.SLEPc.MFN.ConvergedReason* attribute), 138

CONVERGED_ITERATING (*slepc4py.SLEPc.NEP.ConvergedReason* attribute), 149

CONVERGED_ITERATING (*slepc4py.SLEPc.PEP.ConvergedReason* attribute), 188

CONVERGED_ITERATING (*slepc4py.SLEPc.SVD.ConvergedReason* attribute), 254

CONVERGED_ITERATING (*slepc4py.SLEPc.EPS.ConvergedReason* attribute), 61

CONVERGED_ITERATING (*slepc4py.SLEPc.NEP.ConvergedReason* attribute), 149

CONVERGED_ITERATING (*slepc4py.SLEPc.PEP.ConvergedReason* attribute), 188

CONVERGED_ITERATING (*slepc4py.SLEPc.SVD.ConvergedReason* attribute), 254

ConvergedReason (class in *slepc4py.SLEPc.EPS*), 60

ConvergedReason (class in *slepc4py.SLEPc.LME*), 127

ConvergedReason (class in *slepc4py.SLEPc.MFN*), 138

ConvergedReason (class in *slepc4py.SLEPc.NEP*), 148

ConvergedReason (class in *slepc4py.SLEPc.PEP*), 188

ConvergedReason (class in *slepc4py.SLEPc.SVD*), 253

COPY (*slepc4py.SLEPc.ST.MatMode* attribute), 237

copy() (*slepc4py.SLEPc.BV* method), 21

copyColumn() (*slepc4py.SLEPc.BV* method), 21

copyVec() (*slepc4py.SLEPc.BV* method), 21

create() (*slepc4py.SLEPc.BV* method), 22

create() (*slepc4py.SLEPc.DS* method), 47

create() (*slepc4py.SLEPc.EPS* method), 76

create() (*slepc4py.SLEPc.FN* method), 120

create() (*slepc4py.SLEPc.LME* method), 130

create() (*slepc4py.SLEPc.MFN* method), 140

create() (*slepc4py.SLEPc.NEP* method), 159

create() (*slepc4py.SLEPc.PEP* method), 199

create() (*slepc4py.SLEPc.RG* method), 229

create() (*slepc4py.SLEPc.ST* method), 240

create() (*slepc4py.SLEPc.SVD* method), 262

createFromMat() (*slepc4py.SLEPc.BV* method), 22

createMat() (*slepc4py.SLEPc.BV* method), 22

`createMatBSE()` (*slepc4py.SLEPc.Util class method*), 282
`createMatHamiltonian()` (*slepc4py.SLEPc.Util class method*), 283
`createVec()` (*slepc4py.SLEPc.BV method*), 22
`CROSS` (*slepc4py.SLEPc.SVD.Type attribute*), 257
`CURRENT` (*in module slepc4py.SLEPc*), 284
`CYCLIC` (*slepc4py.SLEPc.SVD.Type attribute*), 257

D

`D` (*slepc4py.SLEPc.DS.MatType attribute*), 43
`DECIDE` (*in module slepc4py.SLEPc*), 283
`DEFAULT` (*in module slepc4py.SLEPc*), 283
`DELAYED` (*slepc4py.SLEPc.EPS.LanczosReorthogType attribute*), 64
`destroy()` (*slepc4py.SLEPc.BV method*), 23
`destroy()` (*slepc4py.SLEPc.DS method*), 48
`destroy()` (*slepc4py.SLEPc.EPS method*), 76
`destroy()` (*slepc4py.SLEPc.FN method*), 120
`destroy()` (*slepc4py.SLEPc.LME method*), 131
`destroy()` (*slepc4py.SLEPc.MFN method*), 141
`destroy()` (*slepc4py.SLEPc.NEP method*), 159
`destroy()` (*slepc4py.SLEPc.PEP method*), 200
`destroy()` (*slepc4py.SLEPc.RG method*), 229
`destroy()` (*slepc4py.SLEPc.ST method*), 241
`destroy()` (*slepc4py.SLEPc.SVD method*), 262
`DETERMINE` (*in module slepc4py.SLEPc*), 283
`DIAGONAL` (*slepc4py.SLEPc.PEP.Scale attribute*), 192
`DISTRIBUTED` (*slepc4py.SLEPc.DS.ParallelType attribute*), 43
`DIVERGED_BREAKDOWN` (*slepc4py.SLEPc.EPS.ConvergedReason attribute*), 61
`DIVERGED_BREAKDOWN` (*slepc4py.SLEPc.LME.ConvergedReason attribute*), 127
`DIVERGED_BREAKDOWN` (*slepc4py.SLEPc.MFN.ConvergedReason attribute*), 138
`DIVERGED_BREAKDOWN` (*slepc4py.SLEPc.NEP.ConvergedReason attribute*), 149
`DIVERGED_BREAKDOWN` (*slepc4py.SLEPc.PEP.ConvergedReason attribute*), 188
`DIVERGED_BREAKDOWN` (*slepc4py.SLEPc.SVD.ConvergedReason attribute*), 254
`DIVERGED_ITS` (*slepc4py.SLEPc.EPS.ConvergedReason attribute*), 61
`DIVERGED_ITS` (*slepc4py.SLEPc.LME.ConvergedReason attribute*), 127
`DIVERGED_ITS` (*slepc4py.SLEPc.MFN.ConvergedReason attribute*), 138
`DIVERGED_ITS` (*slepc4py.SLEPc.NEP.ConvergedReason attribute*), 149
`DIVERGED_ITS` (*slepc4py.SLEPc.PEP.ConvergedReason attribute*), 188
`DIVERGED_ITS` (*slepc4py.SLEPc.SVD.ConvergedReason attribute*), 254
`DIVERGED_LINEAR_SOLVE` (*slepc4py.SLEPc.NEP.ConvergedReason attribute*), 149
`DIVERGED_SUBSPACE_EXHAUSTED` (*slepc4py.SLEPc.NEP.ConvergedReason attribute*), 149
`DIVERGED_SYMMETRY_LOST` (*slepc4py.SLEPc.EPS.ConvergedReason attribute*), 61
`DIVERGED_SYMMETRY_LOST` (*slepc4py.SLEPc.PEP.ConvergedReason attribute*), 188
`DIVERGED_SYMMETRY_LOST` (*slepc4py.SLEPc.SVD.ConvergedReason attribute*), 254
`DIVIDE` (*slepc4py.SLEPc.FN.CombineType attribute*), 117
`dot()` (*slepc4py.SLEPc.BV method*), 23
`dotColumn()` (*slepc4py.SLEPc.BV method*), 23
`dotVec()` (*slepc4py.SLEPc.BV method*), 23
`DS` (*class in slepc4py.SLEPc*), 41
`ds` (*slepc4py.SLEPc.EPS attribute*), 115
`ds` (*slepc4py.SLEPc.NEP attribute*), 184
`ds` (*slepc4py.SLEPc.PEP attribute*), 224
`ds` (*slepc4py.SLEPc.SVD attribute*), 280
`DT_LYAPUNOV` (*slepc4py.SLEPc.LME.ProblemType attribute*), 128
`duplicate()` (*slepc4py.SLEPc.BV method*), 24
`duplicate()` (*slepc4py.SLEPc.DS method*), 48
`duplicate()` (*slepc4py.SLEPc.FN method*), 120
`duplicateResize()` (*slepc4py.SLEPc.BV method*), 24

E

`ELEMENTAL` (*slepc4py.SLEPc.EPS.Type attribute*), 68
`ELEMENTAL` (*slepc4py.SLEPc.SVD.Type attribute*), 257
`ELLIPSE` (*slepc4py.SLEPc.RG.Type attribute*), 226
`ELPA` (*slepc4py.SLEPc.EPS.Type attribute*), 68
`environment variable`
`ARCHFLAGS`, 9
`MACOSX_DEPLOYMENT_TARGET`, 9
`PETSC_ARCH`, 8, 9
`PETSC_DIR`, 8, 9
`SDKROOT`, 9
`SLEPC_DIR`, 8, 9
`EPS` (*class in slepc4py.SLEPc*), 58
`EPSSArbitraryFunction` (*in module slepc4py.typing*), 13
`EPSEigenvalueComparison` (*in module slepc4py.typing*), 13
`EPSKrylovSchurBSEType` (*class in slepc4py.SLEPc*), 116
`EPSSMonitorFunction` (*in module slepc4py.typing*), 13
`EPSSStoppingFunction` (*in module slepc4py.typing*), 13
`ErrorType` (*class in slepc4py.SLEPc.EPS*), 61

ErrorType (*class in slepc4py.SLEPc.NEP*), 150
ErrorType (*class in slepc4py.SLEPc.PEP*), 188
ErrorType (*class in slepc4py.SLEPc.SVD*), 254
errorView() (*slepc4py.SLEPc.EPS method*), 76
errorView() (*slepc4py.SLEPc.NEP method*), 159
errorView() (*slepc4py.SLEPc.PEP method*), 200
errorView() (*slepc4py.SLEPc.SVD method*), 262
evaluateDerivative() (*slepc4py.SLEPc.FN method*), 120
evaluateFunction() (*slepc4py.SLEPc.FN method*), 121
evaluateFunctionMat() (*slepc4py.SLEPc.FN method*), 121
evaluateFunctionMatVec() (*slepc4py.SLEPc.FN method*), 121
EVSL (*slepc4py.SLEPc.EPS.Type attribute*), 68
EXP (*slepc4py.SLEPc.FN.Type attribute*), 118
EXPLICIT (*slepc4py.SLEPc.NEP.RefineScheme attribute*), 151
EXPLICIT (*slepc4py.SLEPc.PEP.RefineScheme attribute*), 192
EXPOKIT (*slepc4py.SLEPc.MFN.Type attribute*), 139
extra_row (*slepc4py.SLEPc.DS attribute*), 57
Extract (*class in slepc4py.SLEPc.PEP*), 189
extract (*slepc4py.SLEPc.PEP attribute*), 224
Extraction (*class in slepc4py.SLEPc.EPS*), 62
extraction (*slepc4py.SLEPc.EPS attribute*), 115

F

- FEAST (*slepc4py.SLEPc.EPS.Type attribute*), 68
- FEJER (*slepc4py.SLEPc.ST.FilterDamping attribute*), 235
- FEJER (*slepc4py.SLEPc.STFilterDamping attribute*), 251
- FILTER (*slepc4py.SLEPc.ST.Type attribute*), 237
- FilterDamping (*class in slepc4py.SLEPc.ST*), 235
- FilterType (*class in slepc4py.SLEPc.ST*), 236
- FILTLAN (*slepc4py.SLEPc.ST.FilterType attribute*), 236
- FILTLAN (*slepc4py.SLEPc.STFilterType attribute*), 252
- FN (*class in slepc4py.SLEPc*), 117
- fn (*slepc4py.SLEPc.LME attribute*), 137
- fn (*slepc4py.SLEPc.MFN attribute*), 146
- FULL (*slepc4py.SLEPc.EPS.LanczosReorthogType attribute*), 64

G

GD (*slepc4py.SLEPc.EPS.Type attribute*), 68

GEN_LYAPUNOV (*slepc4py.SLEPc.LME.ProblemType attribute*), 128

GEN_SYLVESTER (*slepc4py.SLEPc.LME.ProblemType attribute*), 128

GENERAL (*slepc4py.SLEPc.NEP.ProblemType attribute*), 150

GENERAL (*slepc4py.SLEPc.PEP.ProblemType attribute*), 191

getConverged() (*slepc4py.SLEPc.SVD method*), 263
 getConvergedReason() (*slepc4py.SLEPc.EPS method*), 79
 getConvergedReason() (*slepc4py.SLEPc.LME method*), 131
 getConvergedReason() (*slepc4py.SLEPc.MFN method*), 141
 getConvergedReason() (*slepc4py.SLEPc.NEP method*), 161
 getConvergedReason() (*slepc4py.SLEPc.PEP method*), 202
 getConvergedReason() (*slepc4py.SLEPc.SVD method*), 263
 getConvergenceTest() (*slepc4py.SLEPc.EPS method*), 79
 getConvergenceTest() (*slepc4py.SLEPc.NEP method*), 162
 getConvergenceTest() (*slepc4py.SLEPc.PEP method*), 202
 getConvergenceTest() (*slepc4py.SLEPc.SVD method*), 263
 getCrosEPS() (*slepc4py.SLEPc.SVD method*), 263
 getCrossExplicitMatrix() (*slepc4py.SLEPc.SVD method*), 264
 getCyclicEPS() (*slepc4py.SLEPc.SVD method*), 264
 getCyclicExplicitMatrix() (*slepc4py.SLEPc.SVD method*), 264
 getDefiniteTolerance() (*slepc4py.SLEPc.BV method*), 25
 getDimensions() (*slepc4py.SLEPc.DS method*), 48
 getDimensions() (*slepc4py.SLEPc.EPS method*), 80
 getDimensions() (*slepc4py.SLEPc.LME method*), 131
 getDimensions() (*slepc4py.SLEPc.MFN method*), 141
 getDimensions() (*slepc4py.SLEPc.NEP method*), 162
 getDimensions() (*slepc4py.SLEPc.PEP method*), 203
 getDimensions() (*slepc4py.SLEPc.SVD method*), 264
 getDS() (*slepc4py.SLEPc.EPS method*), 80
 getDS() (*slepc4py.SLEPc.NEP method*), 162
 getDS() (*slepc4py.SLEPc.PEP method*), 203
 getDS() (*slepc4py.SLEPc.SVD method*), 264
 getEigenpair() (*slepc4py.SLEPc.EPS method*), 80
 getEigenpair() (*slepc4py.SLEPc.NEP method*), 162
 getEigenpair() (*slepc4py.SLEPc.PEP method*), 203
 getEigenvalue() (*slepc4py.SLEPc.EPS method*), 81
 getEigenvector() (*slepc4py.SLEPc.EPS method*), 81
 getEllipseParameters() (*slepc4py.SLEPc.RG method*), 230
 getErrorEstimate() (*slepc4py.SLEPc.EPS method*), 81
 getErrorEstimate() (*slepc4py.SLEPc.LME method*), 132
 getErrorEstimate() (*slepc4py.SLEPc.NEP method*), 163
 getErrorEstimate() (*slepc4py.SLEPc.PEP method*), 203
 getErrorIfNotConverged() (*slepc4py.SLEPc.LME method*), 132
 getErrorIfNotConverged() (*slepc4py.SLEPc.MFN method*), 141
 getExtract() (*slepc4py.SLEPc.PEP method*), 204
 getExtraction() (*slepc4py.SLEPc.EPS method*), 82
 getExtraRow() (*slepc4py.SLEPc.DS method*), 49
 getFilterDamping() (*slepc4py.SLEPc.ST method*), 241
 getFilterDegree() (*slepc4py.SLEPc.ST method*), 241
 getFilterInterval() (*slepc4py.SLEPc.ST method*), 241
 getFilterRange() (*slepc4py.SLEPc.ST method*), 242
 getFilterType() (*slepc4py.SLEPc.ST method*), 242
 getFN() (*slepc4py.SLEPc.MFN method*), 142
 getFunction() (*slepc4py.SLEPc.NEP method*), 163
 getGDBlockSize() (*slepc4py.SLEPc.EPS method*), 82
 getGDBOrth() (*slepc4py.SLEPc.EPS method*), 82
 getGDDoubleExpansion() (*slepc4py.SLEPc.EPS method*), 82
 getGDInitialSize() (*slepc4py.SLEPc.EPS method*), 83
 getGDKrylovStart() (*slepc4py.SLEPc.EPS method*), 83
 getGDRestart() (*slepc4py.SLEPc.EPS method*), 83
 getGSVDDimensions() (*slepc4py.SLEPc.DS method*), 49
 getHSVDDimensions() (*slepc4py.SLEPc.DS method*), 49
 getImplicitTranspose() (*slepc4py.SLEPc.SVD method*), 265
 getInterpolInterpolation() (*slepc4py.SLEPc.NEP method*), 163
 getInterpolPEP() (*slepc4py.SLEPc.NEP method*), 163
 getInterval() (*slepc4py.SLEPc.EPS method*), 83
 getInterval() (*slepc4py.SLEPc.PEP method*), 204
 getIntervalEndpoints() (*slepc4py.SLEPc.RG method*), 230
 getInvariantSubspace() (*slepc4py.SLEPc.EPS method*), 84
 getIterationNumber() (*slepc4py.SLEPc.EPS method*), 84
 getIterationNumber() (*slepc4py.SLEPc.LME method*), 132
 getIterationNumber() (*slepc4py.SLEPc.MFN method*), 142
 getIterationNumber() (*slepc4py.SLEPc.NEP method*), 164
 getIterationNumber() (*slepc4py.SLEPc.PEP method*), 204
 getIterationNumber() (*slepc4py.SLEPc.SVD method*), 265
 getJacobian() (*slepc4py.SLEPc.NEP method*), 164

getJDBlockSize() (*slepc4py.SLEPc.EPS method*), 84
 getJDBOrth() (*slepc4py.SLEPc.EPS method*), 84
 getJDConstCorrectionTol() (*slepc4py.SLEPc.EPS method*), 85
 getJDFix() (*slepc4py.SLEPc.EPS method*), 85
 getJDFix() (*slepc4py.SLEPc.PEP method*), 204
 getJDInitialSize() (*slepc4py.SLEPc.EPS method*), 85
 getJDKrylovStart() (*slepc4py.SLEPc.EPS method*), 85
 getJDMinimalityIndex() (*slepc4py.SLEPc.PEP method*), 205
 getJDProjection() (*slepc4py.SLEPc.PEP method*), 205
 getJDRestart() (*slepc4py.SLEPc.EPS method*), 85
 getJDRestart() (*slepc4py.SLEPc.PEP method*), 205
 getJDReusePreconditioner() (*slepc4py.SLEPc.PEP method*), 205
 getKrylovSchurBSEType() (*slepc4py.SLEPc.EPS method*), 86
 getKrylovSchurDetectZeros() (*slepc4py.SLEPc.EPS method*), 86
 getKrylovSchurDimensions() (*slepc4py.SLEPc.EPS method*), 86
 getKrylovSchurInertias() (*slepc4py.SLEPc.EPS method*), 86
 getKrylovSchurKSP() (*slepc4py.SLEPc.EPS method*), 87
 getKrylovSchurLocking() (*slepc4py.SLEPc.EPS method*), 87
 getKrylovSchurPartitions() (*slepc4py.SLEPc.EPS method*), 87
 getKrylovSchurRestart() (*slepc4py.SLEPc.EPS method*), 87
 getKrylovSchurSubcommInfo() (*slepc4py.SLEPc.EPS method*), 88
 getKrylovSchurSubcommMats() (*slepc4py.SLEPc.EPS method*), 88
 getKrylovSchurSubcommPairs() (*slepc4py.SLEPc.EPS method*), 88
 getKrylovSchurSubintervals() (*slepc4py.SLEPc.EPS method*), 89
 getKSP() (*slepc4py.SLEPc.ST method*), 242
 getLanczosOneSide() (*slepc4py.SLEPc.SVD method*), 265
 getLanczosReorthogType() (*slepc4py.SLEPc.EPS method*), 90
 getLeadingDimension() (*slepc4py.SLEPc.BV method*), 25
 getLeadingDimension() (*slepc4py.SLEPc.DS method*), 49
 getLeftEigenvector() (*slepc4py.SLEPc.EPS method*), 90
 getLeftEigenvector() (*slepc4py.SLEPc.NEP method*), 164
 getLinearEPS() (*slepc4py.SLEPc.PEP method*), 205
 getLinearExplicitMatrix() (*slepc4py.SLEPc.PEP method*), 206
 getLinearLinearization() (*slepc4py.SLEPc.PEP method*), 206
 getLOBPCGBlockSize() (*slepc4py.SLEPc.EPS method*), 89
 getLOBPCGLocking() (*slepc4py.SLEPc.EPS method*), 89
 getLOBPCGRestart() (*slepc4py.SLEPc.EPS method*), 90
 getLyapIIRanks() (*slepc4py.SLEPc.EPS method*), 90
 getMat() (*slepc4py.SLEPc.BV method*), 25
 getMat() (*slepc4py.SLEPc.DS method*), 49
 getMatMode() (*slepc4py.SLEPc.ST method*), 242
 getMatMultMethod() (*slepc4py.SLEPc.BV method*), 25
 getMatrices() (*slepc4py.SLEPc.ST method*), 243
 getMatrix() (*slepc4py.SLEPc.BV method*), 26
 getMatStructure() (*slepc4py.SLEPc.ST method*), 243
 getMethod() (*slepc4py.SLEPc.DS method*), 50
 getMethod() (*slepc4py.SLEPc.FN method*), 122
 getMonitor() (*slepc4py.SLEPc.EPS method*), 91
 getMonitor() (*slepc4py.SLEPc.LME method*), 132
 getMonitor() (*slepc4py.SLEPc.MFN method*), 142
 getMonitor() (*slepc4py.SLEPc.NEP method*), 164
 getMonitor() (*slepc4py.SLEPc.PEP method*), 206
 getMonitor() (*slepc4py.SLEPc.SVD method*), 265
 getNArnoldiKSP() (*slepc4py.SLEPc.NEP method*), 165
 getNArnoldiLagPreconditioner() (*slepc4py.SLEPc.NEP method*), 165
 getNLEIGSEPS() (*slepc4py.SLEPc.NEP method*), 165
 getNLEIGSFullBasis() (*slepc4py.SLEPc.NEP method*), 165
 getNLEIGSInterpolation() (*slepc4py.SLEPc.NEP method*), 165
 getNLEIGSKSPs() (*slepc4py.SLEPc.NEP method*), 166
 getNLEIGSLocking() (*slepc4py.SLEPc.NEP method*), 166
 getNLEIGSRestart() (*slepc4py.SLEPc.NEP method*), 166
 getNLEIGSRKShifts() (*slepc4py.SLEPc.NEP method*), 166
 getNumConstraints() (*slepc4py.SLEPc.BV method*), 26
 getOperator() (*slepc4py.SLEPc.MFN method*), 142
 getOperator() (*slepc4py.SLEPc.ST method*), 243
 getOperators() (*slepc4py.SLEPc.EPS method*), 91
 getOperators() (*slepc4py.SLEPc.PEP method*), 206
 getOperators() (*slepc4py.SLEPc.SVD method*), 265
 getOptionsPrefix() (*slepc4py.SLEPc.BV method*), 26
 getOptionsPrefix() (*slepc4py.SLEPc.DS method*), 50
 getOptionsPrefix() (*slepc4py.SLEPc.EPS method*), 91

getOptionsPrefix() (*slepc4py.SLEPc.FN method*), 122
 getOptionsPrefix() (*slepc4py.SLEPc.LME method*), 132
 getOptionsPrefix() (*slepc4py.SLEPc.MFN method*), 142
 getOptionsPrefix() (*slepc4py.SLEPc.NEP method*), 167
 getOptionsPrefix() (*slepc4py.SLEPc.PEP method*), 207
 getOptionsPrefix() (*slepc4py.SLEPc.RG method*), 230
 getOptionsPrefix() (*slepc4py.SLEPc.ST method*), 243
 getOptionsPrefix() (*slepc4py.SLEPc.SVD method*), 266
 getOrthogonalization() (*slepc4py.SLEPc.BV method*), 26
 getParallel() (*slepc4py.SLEPc.DS method*), 50
 getParallel() (*slepc4py.SLEPc.FN method*), 122
 getPEPCoefficients() (*slepc4py.SLEPc.DS method*), 50
 getPEPDegree() (*slepc4py.SLEPc.DS method*), 50
 getPhiIndex() (*slepc4py.SLEPc.FN method*), 122
 getPolygonVertices() (*slepc4py.SLEPc.RG method*), 231
 getPowerShiftType() (*slepc4py.SLEPc.EPS method*), 91
 getPreconditionerMat() (*slepc4py.SLEPc.ST method*), 244
 getProblemType() (*slepc4py.SLEPc.EPS method*), 91
 getProblemType() (*slepc4py.SLEPc.LME method*), 133
 getProblemType() (*slepc4py.SLEPc.NEP method*), 167
 getProblemType() (*slepc4py.SLEPc.PEP method*), 207
 getProblemType() (*slepc4py.SLEPc.SVD method*), 266
 getPurify() (*slepc4py.SLEPc.EPS method*), 91
 getQArnoldiLocking() (*slepc4py.SLEPc.PEP method*), 207
 getQArnoldiRestart() (*slepc4py.SLEPc.PEP method*), 207
 getRandomContext() (*slepc4py.SLEPc.BV method*), 27
 getRationalDenominator() (*slepc4py.SLEPc.FN method*), 123
 getRationalNumerator() (*slepc4py.SLEPc.FN method*), 123
 getRefine() (*slepc4py.SLEPc.NEP method*), 169
 getRefine() (*slepc4py.SLEPc.PEP method*), 208
 getRefined() (*slepc4py.SLEPc.DS method*), 51
 getRefineKSP() (*slepc4py.SLEPc.NEP method*), 169
 getRefineKSP() (*slepc4py.SLEPc.PEP method*), 208
 getRG() (*slepc4py.SLEPc.EPS method*), 92
 getRG() (*slepc4py.SLEPc.NEP method*), 167
 getRG() (*slepc4py.SLEPc.PEP method*), 207
 getRHS() (*slepc4py.SLEPc.LME method*), 133
 getRIIConstCorrectionTol() (*slepc4py.SLEPc.NEP method*), 167
 getRIIDeflationThreshold() (*slepc4py.SLEPc.NEP method*), 167
 getRIIHermitian() (*slepc4py.SLEPc.NEP method*), 168
 getRIIKSP() (*slepc4py.SLEPc.NEP method*), 168
 getRIILagPreconditioner() (*slepc4py.SLEPc.NEP method*), 168
 getRIIMaximumIterations() (*slepc4py.SLEPc.NEP method*), 168
 getRingParameters() (*slepc4py.SLEPc.RG method*), 231
 getRQCGReset() (*slepc4py.SLEPc.EPS method*), 92
 getScale() (*slepc4py.SLEPc.FN method*), 123
 getScale() (*slepc4py.SLEPc.PEP method*), 210
 getScale() (*slepc4py.SLEPc.RG method*), 231
 getShift() (*slepc4py.SLEPc.ST method*), 244
 getSignature() (*slepc4py.SLEPc.SVD method*), 266
 getSingularTriplet() (*slepc4py.SLEPc.SVD method*), 266
 getSizes() (*slepc4py.SLEPc.BV method*), 27
 getSLPDeflationThreshold() (*slepc4py.SLEPc.NEP method*), 169
 getSLPEPS() (*slepc4py.SLEPc.NEP method*), 169
 getSLPEPSLeft() (*slepc4py.SLEPc.NEP method*), 169
 getSLPKSP() (*slepc4py.SLEPc.NEP method*), 170
 getSolution() (*slepc4py.SLEPc.LME method*), 133
 getSplitOperator() (*slepc4py.SLEPc.NEP method*), 170
 getSplitPreconditioner() (*slepc4py.SLEPc.NEP method*), 170
 getSplitPreconditioner() (*slepc4py.SLEPc.ST method*), 244
 getST() (*slepc4py.SLEPc.EPS method*), 92
 getST() (*slepc4py.SLEPc.PEP method*), 208
 getState() (*slepc4py.SLEPc.DS method*), 51
 getSTOARCheckEigenvalueType() (*slepc4py.SLEPc.PEP method*), 208
 getSTOARDetectZeros() (*slepc4py.SLEPc.PEP method*), 208
 getSTOARDimensions() (*slepc4py.SLEPc.PEP method*), 209
 getSTOARInertias() (*slepc4py.SLEPc.PEP method*), 209
 getSTOARLinearization() (*slepc4py.SLEPc.PEP method*), 209
 getSTOARLocking() (*slepc4py.SLEPc.PEP method*), 210
 getStoppingTest() (*slepc4py.SLEPc.EPS method*), 92
 getStoppingTest() (*slepc4py.SLEPc.NEP method*), 170
 getStoppingTest() (*slepc4py.SLEPc.PEP method*),

210
getStoppingTest() (slepc4py.SLEPc.SVD method), 267
getSVDDimensions() (slepc4py.SLEPc.DS method), 51
getTarget() (slepc4py.SLEPc.EPS method), 92
getTarget() (slepc4py.SLEPc.NEP method), 171
getTarget() (slepc4py.SLEPc.PEP method), 211
getThreshold() (slepc4py.SLEPc.EPS method), 93
getThreshold() (slepc4py.SLEPc.SVD method), 268
getTOARLocking() (slepc4py.SLEPc.PEP method), 210
getTOARRestart() (slepc4py.SLEPc.PEP method), 210
getTolerances() (slepc4py.SLEPc.EPS method), 93
getTolerances() (slepc4py.SLEPc.LME method), 133
getTolerances() (slepc4py.SLEPc.MFN method), 143
getTolerances() (slepc4py.SLEPc.NEP method), 171
getTolerances() (slepc4py.SLEPc.PEP method), 211
getTolerances() (slepc4py.SLEPc.SVD method), 268
getTrackAll() (slepc4py.SLEPc.EPS method), 93
getTrackAll() (slepc4py.SLEPc.NEP method), 171
getTrackAll() (slepc4py.SLEPc.PEP method), 211
getTrackAll() (slepc4py.SLEPc.SVD method), 269
getTransform() (slepc4py.SLEPc.ST method), 244
getTRLanczosExplicitMatrix() (slepc4py.SLEPc.SVD method), 267
getTRLanczosGBidiag() (slepc4py.SLEPc.SVD method), 267
getTRLanczosKSP() (slepc4py.SLEPc.SVD method), 267
getTRLanczosLocking() (slepc4py.SLEPc.SVD method), 267
getTRLanczosOneSide() (slepc4py.SLEPc.SVD method), 268
getTRLanczosRestart() (slepc4py.SLEPc.SVD method), 268
getTrueResidual() (slepc4py.SLEPc.EPS method), 93
getTwoSided() (slepc4py.SLEPc.EPS method), 94
getTwoSided() (slepc4py.SLEPc.NEP method), 171
getType() (slepc4py.SLEPc.BV method), 27
getType() (slepc4py.SLEPc.DS method), 51
getType() (slepc4py.SLEPc.EPS method), 94
getType() (slepc4py.SLEPc.FN method), 123
getType() (slepc4py.SLEPc.LME method), 133
getType() (slepc4py.SLEPc.MFN method), 143
getType() (slepc4py.SLEPc.NEP method), 171
getType() (slepc4py.SLEPc.PEP method), 211
getType() (slepc4py.SLEPc.RG method), 231
getType() (slepc4py.SLEPc.ST method), 244
getType() (slepc4py.SLEPc.SVD method), 269
getValue() (slepc4py.SLEPc.SVD method), 269
getVectors() (slepc4py.SLEPc.SVD method), 269
getVecType() (slepc4py.SLEPc.BV method), 27
getVersion() (slepc4py.SLEPc.Sys class method), 280
getVersionInfo() (slepc4py.SLEPc.Sys class method), 281

getWhichEigenpairs() (slepc4py.SLEPc.EPS method), 94
getWhichEigenpairs() (slepc4py.SLEPc.NEP method), 172
getWhichEigenpairs() (slepc4py.SLEPc.PEP method), 212
getWhichSingularTriplets() (slepc4py.SLEPc.SVD method), 270
GHEP (slepc4py.SLEPc.DS.Type attribute), 45
GHEP (slepc4py.SLEPc.EPS.ProblemType attribute), 65
GHIEP (slepc4py.SLEPc.DS.Type attribute), 45
GHIEP (slepc4py.SLEPc.EPS.ProblemType attribute), 65
GNHEP (slepc4py.SLEPc.DS.Type attribute), 45
GNHEP (slepc4py.SLEPc.EPS.ProblemType attribute), 65
GRUNING (slepc4py.SLEPc.EPS.KrylovSchurBSEType attribute), 63
GRUNING (slepc4py.SLEPc.EPSKrylovSchurBSEType attribute), 116
GS (slepc4py.SLEPc.BV.OrthogBlockType attribute), 16
GSVD (slepc4py.SLEPc.DS.Type attribute), 45
GYROSCOPIC (slepc4py.SLEPc.PEP.ProblemType attribute), 191

H

HAMILT (slepc4py.SLEPc.EPS.ProblemType attribute), 66
HANKEL (slepc4py.SLEPc.EPS.CISSExtraction attribute), 59
HANKEL (slepc4py.SLEPc.NEP.CISSExtraction attribute), 148
HANKEL (slepc4py.SLEPc.PEP.CISSExtraction attribute), 187
HARMONIC (slepc4py.SLEPc.EPS.Extraction attribute), 62
HARMONIC (slepc4py.SLEPc.PEP.JDProjection attribute), 190
HARMONIC_LARGEST (slepc4py.SLEPc.EPS.Extraction attribute), 62
HARMONIC_RELATIVE (slepc4py.SLEPc.EPS.Extraction attribute), 62
HARMONIC_RIGHT (slepc4py.SLEPc.EPS.Extraction attribute), 63
hasExternalPackage() (slepc4py.SLEPc.Sys class method), 281
HEP (slepc4py.SLEPc.DS.Type attribute), 45
HEP (slepc4py.SLEPc.EPS.ProblemType attribute), 66
HERMITE (slepc4py.SLEPc.PEP.Basis attribute), 186
HERMITIAN (slepc4py.SLEPc.PEP.ProblemType attribute), 191
HSVD (slepc4py.SLEPc.DS.Type attribute), 45
HYPERBOLIC (slepc4py.SLEPc.PEP.ProblemType attribute), 191
HYPERBOLIC (slepc4py.SLEPc.SVD.ProblemType attribute), 255

I

IFNEEDED (*slepc4py.SLEPc.BV.OrthogRefineType attribute*), 17

init() (*in module slepc4py*), 11

INPLACE (*slepc4py.SLEPc.ST.MatMode attribute*), 237

insertConstraints() (*slepc4py.SLEPc.BV method*), 27

insertVec() (*slepc4py.SLEPc.BV method*), 28

insertVecs() (*slepc4py.SLEPc.BV method*), 28

INTERMEDIATE (*slepc4py.SLEPc.DS.StateType attribute*), 44

INTERPOL (*slepc4py.SLEPc.NEP.Type attribute*), 153

INTERVAL (*slepc4py.SLEPc.RG.Type attribute*), 226

INVSQRT (*slepc4py.SLEPc.FN.Type attribute*), 118

isAxisymmetric() (*slepc4py.SLEPc.RG method*), 232

isFinalized() (*slepc4py.SLEPc.Sys class method*), 281

isGeneralized() (*slepc4py.SLEPc.EPS method*), 94

isGeneralized() (*slepc4py.SLEPc.SVD method*), 270

isHermitian() (*slepc4py.SLEPc.EPS method*), 94

isHyperbolic() (*slepc4py.SLEPc.SVD method*), 270

isInitialized() (*slepc4py.SLEPc.Sys class method*), 282

isPositive() (*slepc4py.SLEPc.EPS method*), 95

isStructured() (*slepc4py.SLEPc.EPS method*), 95

isTrivial() (*slepc4py.SLEPc.RG method*), 232

ITERATING (*slepc4py.SLEPc.EPS.ConvergedReason attribute*), 61

ITERATING (*slepc4py.SLEPc.LME.ConvergedReason attribute*), 127

ITERATING (*slepc4py.SLEPc.MFN.ConvergedReason attribute*), 138

ITERATING (*slepc4py.SLEPc.NEP.ConvergedReason attribute*), 149

ITERATING (*slepc4py.SLEPc.PEP.ConvergedReason attribute*), 188

ITERATING (*slepc4py.SLEPc.SVD.ConvergedReason attribute*), 254

J

JACKSON (*slepc4py.SLEPc.ST.FilterDamping attribute*), 235

JACKSON (*slepc4py.SLEPc.STFilterDamping attribute*), 251

JD (*slepc4py.SLEPc.EPS.Type attribute*), 68

JD (*slepc4py.SLEPc.PEP.Type attribute*), 194

JDProjection (*class in slepc4py.SLEPc.PEP*), 190

K

KRYLOV (*slepc4py.SLEPc.LME.Type attribute*), 128

KRYLOV (*slepc4py.SLEPc.MFN.Type attribute*), 139

KRYLOVSCHUR (*slepc4py.SLEPc.EPS.Type attribute*), 68

KrylovSchurBSEType (*class in slepc4py.SLEPc.EPS*), 63

ksp (*slepc4py.SLEPc.ST attribute*), 251

KSVD (*slepc4py.SLEPc.SVD.Type attribute*), 257

L

LAGUERRE (*slepc4py.SLEPc.PEP.Basis attribute*), 186

LANCZOS (*slepc4py.SLEPc.EPS.Type attribute*), 68

LANCZOS (*slepc4py.SLEPc.ST.FilterDamping attribute*), 236

LANCZOS (*slepc4py.SLEPc.STFilterDamping attribute*), 252

LANCZOS (*slepc4py.SLEPc.SVD.Type attribute*), 257

LanczosReorthogType (*class in slepc4py.SLEPc.EPS*), 63

LAPACK (*slepc4py.SLEPc.EPS.Type attribute*), 68

LAPACK (*slepc4py.SLEPc.SVD.Type attribute*), 257

LARGEST (*slepc4py.SLEPc.SVD.Which attribute*), 258

LARGEST_IMAGINARY (*slepc4py.SLEPc.EPS.Which attribute*), 69

LARGEST_IMAGINARY (*slepc4py.SLEPc.NEP.Which attribute*), 154

LARGEST_IMAGINARY (*slepc4py.SLEPc.PEP.Which attribute*), 195

LARGEST_MAGNITUDE (*slepc4py.SLEPc.EPS.Which attribute*), 69

LARGEST_MAGNITUDE (*slepc4py.SLEPc.NEP.Which attribute*), 154

LARGEST_MAGNITUDE (*slepc4py.SLEPc.PEP.Which attribute*), 195

LARGEST_REAL (*slepc4py.SLEPc.EPS.Which attribute*), 70

LARGEST_REAL (*slepc4py.SLEPc.NEP.Which attribute*), 154

LARGEST_REAL (*slepc4py.SLEPc.PEP.Which attribute*), 195

LayoutSizeSpec (*in module slepc4py.typing*), 12

LEGENDRE (*slepc4py.SLEPc.PEP.Basis attribute*), 186

LINEAR (*slepc4py.SLEPc.PEP.Type attribute*), 194

LME (*class in slepc4py.SLEPc*), 126

LMEMonitorFunction (*in module slepc4py.typing*), 14

LOBPCG (*slepc4py.SLEPc.EPS.Type attribute*), 68

LOCAL (*slepc4py.SLEPc.EPS.LanczosReorthogType attribute*), 64

local_size (*slepc4py.SLEPc.BV attribute*), 40

LOG (*slepc4py.SLEPc.FN.Type attribute*), 118

LOWER (*slepc4py.SLEPc.SVD.TRLanczosGBidiag attribute*), 256

LYAPII (*slepc4py.SLEPc.EPS.Type attribute*), 68

LYAPUNOV (*slepc4py.SLEPc.LME.ProblemType attribute*), 128

M

MACOSX_DEPLOYMENT_TARGET, 9

MAT (*slepc4py.SLEPc.BV.MatMultType attribute*), 16

MAT (*slepc4py.SLEPc.BV.Type attribute*), 18

`mat_mode` (*slepc4py.SLEPc.ST attribute*), 251
`mat_structure` (*slepc4py.SLEPc.ST attribute*), 251
`MatMode` (class in *slepc4py.SLEPc.ST*), 236
`matMult()` (*slepc4py.SLEPc.BV method*), 28
`matMultColumn()` (*slepc4py.SLEPc.BV method*), 29
`matMultHermitianTranspose()` (*slepc4py.SLEPc.BV method*), 29
`matMultHermitianTransposeColumn()`
 (*slepc4py.SLEPc.BV method*), 30
`matMultTransposeColumn()` (*slepc4py.SLEPc.BV method*), 30
`MatMultType` (class in *slepc4py.SLEPc.BV*), 15
`matProject()` (*slepc4py.SLEPc.BV method*), 30
`MatType` (class in *slepc4py.SLEPc.DS*), 42
`max_it` (*slepc4py.SLEPc.EPS attribute*), 115
`max_it` (*slepc4py.SLEPc.LME attribute*), 137
`max_it` (*slepc4py.SLEPc.MFN attribute*), 147
`max_it` (*slepc4py.SLEPc.NEP attribute*), 184
`max_it` (*slepc4py.SLEPc.PEP attribute*), 225
`max_it` (*slepc4py.SLEPc.SVD attribute*), 280
`MAXIT` (*slepc4py.SLEPc.SVD.Conv attribute*), 253
`MBE` (*slepc4py.SLEPc.NEP.RefineScheme attribute*), 151
`MBE` (*slepc4py.SLEPc.PEP.RefineScheme attribute*), 192
`method` (*slepc4py.SLEPc.DS attribute*), 57
`method` (*slepc4py.SLEPc.FN attribute*), 126
`MFN` (class in *slepc4py.SLEPc*), 138
`MFNMonitorFunction` (in module *slepc4py.typing*), 14
`MGS` (*slepc4py.SLEPc.BV.OrthogType attribute*), 17
module
 slepc4py, 10
 slepc4py.SLEPc, 15
 slepc4py.typing, 11
`MONOMIAL` (*slepc4py.SLEPc.PEP.Basis attribute*), 186
`mult()` (*slepc4py.SLEPc.BV method*), 30
`multColumn()` (*slepc4py.SLEPc.BV method*), 31
`multInPlace()` (*slepc4py.SLEPc.BV method*), 31
`MULTIPLE` (*slepc4py.SLEPc.NEP.Refine attribute*), 151
`MULTIPLE` (*slepc4py.SLEPc.PEP.Refine attribute*), 191
`MULTIPLY` (*slepc4py.SLEPc.FN.CombineType attribute*), 117
`multVec()` (*slepc4py.SLEPc.BV method*), 31

N

`NARNOLDI` (*slepc4py.SLEPc.NEP.Type attribute*), 153
`NEP` (class in *slepc4py.SLEPc*), 147
`NEP` (*slepc4py.SLEPc.DS.Type attribute*), 45
`NEPFunction` (in module *slepc4py.typing*), 14
`NEPJacobian` (in module *slepc4py.typing*), 14
`NEPMonitorFunction` (in module *slepc4py.typing*), 14
`NEPStoppingFunction` (in module *slepc4py.typing*), 13
`NEVER` (*slepc4py.SLEPc.BV.OrthogRefineType attribute*), 17
`NHEP` (*slepc4py.SLEPc.DS.Type attribute*), 45
`NHEP` (*slepc4py.SLEPc.EPS.ProblemType attribute*), 66

`NHEPTS` (*slepc4py.SLEPc.DS.Type attribute*), 45
`NLEIGS` (*slepc4py.SLEPc.NEP.Type attribute*), 153
`NONE` (*slepc4py.SLEPc.EPS.Balance attribute*), 59
`NONE` (*slepc4py.SLEPc.NEP.Refine attribute*), 151
`NONE` (*slepc4py.SLEPc.PEP.Extract attribute*), 189
`NONE` (*slepc4py.SLEPc.PEP.Refine attribute*), 191
`NONE` (*slepc4py.SLEPc.PEP.Scale attribute*), 192
`NONE` (*slepc4py.SLEPc.ST.FilterDamping attribute*), 236
`NONE` (*slepc4py.SLEPc.STFilterDamping attribute*), 252
`NORM` (*slepc4py.SLEPc.EPS.Conv attribute*), 60
`NORM` (*slepc4py.SLEPc.NEP.Conv attribute*), 148
`NORM` (*slepc4py.SLEPc.PEP.Conv attribute*), 187
`NORM` (*slepc4py.SLEPc.PEP.Extract attribute*), 189
`NORM` (*slepc4py.SLEPc.SVD.Conv attribute*), 253
`NORM` (*slepc4py.SLEPc.SVD.ErrorType attribute*), 255
`norm()` (*slepc4py.SLEPc.BV method*), 32
`normColumn()` (*slepc4py.SLEPc.BV method*), 32

O

`ONESIDE` (*slepc4py.SLEPc.EPS.Balance attribute*), 59
`OrthogBlockType` (class in *slepc4py.SLEPc.BV*), 16
`ORTHOGONAL` (*slepc4py.SLEPc.PEP.JDProjection attribute*), 190
`orthogonalize()` (*slepc4py.SLEPc.BV method*), 32
`orthogonalizeColumn()` (*slepc4py.SLEPc.BV method*), 33
`orthogonalizeVec()` (*slepc4py.SLEPc.BV method*), 33
`OrthogRefineType` (class in *slepc4py.SLEPc.BV*), 17
`OrthogType` (class in *slepc4py.SLEPc.BV*), 17
`orthonormalizeColumn()` (*slepc4py.SLEPc.BV method*), 34

P

`parallel` (*slepc4py.SLEPc.DS attribute*), 57
`parallel` (*slepc4py.SLEPc.FN attribute*), 126
`ParallelType` (class in *slepc4py.SLEPc.DS*), 43
`ParallelType` (class in *slepc4py.SLEPc.FN*), 117
`PARTIAL` (*slepc4py.SLEPc.EPS.LanczosReorthogType attribute*), 64
`PEP` (class in *slepc4py.SLEPc*), 185
`PEP` (*slepc4py.SLEPc.DS.Type attribute*), 45
`PEPMonitorFunction` (in module *slepc4py.typing*), 13
`PEPStoppingFunction` (in module *slepc4py.typing*), 13
`PERIODIC` (*slepc4py.SLEPc.EPS.LanczosReorthogType attribute*), 64
`PETSC_ARCH`, 8, 9
`PETSC_DIR`, 8, 9
`PGNHEP` (*slepc4py.SLEPc.EPS.ProblemType attribute*), 66
`PHI` (*slepc4py.SLEPc.FN.Type attribute*), 118
`POLYGON` (*slepc4py.SLEPc.RG.Type attribute*), 226
`POWER` (*slepc4py.SLEPc.EPS.Type attribute*), 68
`PowerShiftType` (class in *slepc4py.SLEPc.EPS*), 64
`PRECOND` (*slepc4py.SLEPc.ST.Type attribute*), 237
`PRIMME` (*slepc4py.SLEPc.EPS.Type attribute*), 68

PRIMME (*slepc4py.SLEPc.SVD.Type attribute*), 258
 problem_type (*slepc4py.SLEPc.EPS attribute*), 115
 problem_type (*slepc4py.SLEPc.NEP attribute*), 184
 problem_type (*slepc4py.SLEPc.PEP attribute*), 225
 problem_type (*slepc4py.SLEPc.SVD attribute*), 280
 ProblemType (*class in slepc4py.SLEPc.EPS*), 65
 ProblemType (*class in slepc4py.SLEPc.LME*), 127
 ProblemType (*class in slepc4py.SLEPc.NEP*), 150
 ProblemType (*class in slepc4py.SLEPc.PEP*), 190
 ProblemType (*class in slepc4py.SLEPc.SVD*), 255
 PROJECTEDBSE (*slepc4py.SLEPc.EPS.KrylovSchurBSEType attribute*), 63
 PROJECTEDBSE (*slepc4py.SLEPc.EPSKrylovSchurBSEType attribute*), 116
 purify (*slepc4py.SLEPc.EPS attribute*), 115

Q

Q (*slepc4py.SLEPc.DS.MatType attribute*), 43
 QARNOLDI (*slepc4py.SLEPc.PEP.Type attribute*), 194
 QR (*slepc4py.SLEPc.BVSVDMethod attribute*), 41
 QR_CAA (*slepc4py.SLEPc.BVSVDMethod attribute*), 41
 QuadRule (*class in slepc4py.SLEPc.RG*), 226

R

RANDOMIZED (*slepc4py.SLEPc.SVD.Type attribute*), 258
 RATIONAL (*slepc4py.SLEPc.FN.Type attribute*), 118
 RATIONAL (*slepc4py.SLEPc.NEP.ProblemType attribute*), 150
 RAW (*slepc4py.SLEPc.DS.StateType attribute*), 44
 RAYLEIGH (*slepc4py.SLEPc.EPS.PowerShiftType attribute*), 65
 REDUNDANT (*slepc4py.SLEPc.DS.ParallelType attribute*), 43
 REDUNDANT (*slepc4py.SLEPc.FN.ParallelType attribute*), 118
 Refine (*class in slepc4py.SLEPc.NEP*), 151
 Refine (*class in slepc4py.SLEPc.PEP*), 191
 REFINE (*slepc4py.SLEPc.BVSVDMethod attribute*), 41
 refined (*slepc4py.SLEPc.DS attribute*), 58
 REFINED (*slepc4py.SLEPc.EPS.Extraction attribute*), 63
 REFINED_HARMONIC (*slepc4py.SLEPc.EPS.Extraction attribute*), 63
 RefineScheme (*class in slepc4py.SLEPc.NEP*), 151
 RefineScheme (*class in slepc4py.SLEPc.PEP*), 191
 REL (*slepc4py.SLEPc.EPS.Conv attribute*), 60
 REL (*slepc4py.SLEPc.NEP.Conv attribute*), 148
 REL (*slepc4py.SLEPc.PEP.Conv attribute*), 187
 REL (*slepc4py.SLEPc.SVD.Conv attribute*), 253
 RELATIVE (*slepc4py.SLEPc.EPS.ErrorType attribute*), 62
 RELATIVE (*slepc4py.SLEPc.NEP.ErrorType attribute*), 150
 RELATIVE (*slepc4py.SLEPc.PEP.ErrorType attribute*), 189

RELATIVE (*slepc4py.SLEPc.SVD.ErrorType attribute*), 255
 reset() (*slepc4py.SLEPc.DS method*), 51
 reset() (*slepc4py.SLEPc.EPS method*), 95
 reset() (*slepc4py.SLEPc.LME method*), 134
 reset() (*slepc4py.SLEPc.MFN method*), 143
 reset() (*slepc4py.SLEPc.NEP method*), 172
 reset() (*slepc4py.SLEPc.PEP method*), 212
 reset() (*slepc4py.SLEPc.ST method*), 245
 reset() (*slepc4py.SLEPc.SVD method*), 270
 RESIDUAL (*slepc4py.SLEPc.PEP.Extract attribute*), 189
 resize() (*slepc4py.SLEPc.BV method*), 34
 restoreColumn() (*slepc4py.SLEPc.BV method*), 34
 restoreMat() (*slepc4py.SLEPc.BV method*), 35
 restoreMat() (*slepc4py.SLEPc.DS method*), 52
 restoreOperator() (*slepc4py.SLEPc.ST method*), 245
 RG (*class in slepc4py.SLEPc*), 225
 rg (*slepc4py.SLEPc.EPS attribute*), 115
 rg (*slepc4py.SLEPc.NEP attribute*), 184
 rg (*slepc4py.SLEPc.PEP attribute*), 225
 RII (*slepc4py.SLEPc.NEP.Type attribute*), 153
 RING (*slepc4py.SLEPc.RG.Type attribute*), 226
 RITZ (*slepc4py.SLEPc.EPS.CISSExtraction attribute*), 59
 RITZ (*slepc4py.SLEPc.EPS.Extraction attribute*), 63
 RITZ (*slepc4py.SLEPc.NEP.CISSExtraction attribute*), 148
 RITZ (*slepc4py.SLEPc.PEP.CISSExtraction attribute*), 187
 RQCG (*slepc4py.SLEPc.EPS.Type attribute*), 68

S

SCALAPACK (*slepc4py.SLEPc.EPS.Type attribute*), 69
 SCALAPACK (*slepc4py.SLEPc.SVD.Type attribute*), 258
 Scalar (*in module slepc4py.typing*), 12
 SCALAR (*slepc4py.SLEPc.PEP.Scale attribute*), 192
 Scale (*class in slepc4py.SLEPc.PEP*), 192
 scale (*slepc4py.SLEPc.RG attribute*), 235
 scale() (*slepc4py.SLEPc.BV method*), 35
 scaleColumn() (*slepc4py.SLEPc.BV method*), 35
 SCHUR (*slepc4py.SLEPc.NEP.RefineScheme attribute*), 151
 SCHUR (*slepc4py.SLEPc.PEP.RefineScheme attribute*), 192
 SDKROOT, 9
 SELECTIVE (*slepc4py.SLEPc.EPS.LanczosReorthogType attribute*), 64
 setActiveColumns() (*slepc4py.SLEPc.BV method*), 35
 setArbitrarySelection() (*slepc4py.SLEPc.EPS method*), 95
 setArnoldiDelayed() (*slepc4py.SLEPc.EPS method*), 96
 setBalance() (*slepc4py.SLEPc.EPS method*), 96
 setBasis() (*slepc4py.SLEPc.PEP method*), 212
 setBlockSize() (*slepc4py.SLEPc.DS method*), 52

setBV() (*slepc4py.SLEPc.EPS method*), 96
 setBV() (*slepc4py.SLEPc.LME method*), 134
 setBV() (*slepc4py.SLEPc.MFN method*), 143
 setBV() (*slepc4py.SLEPc.NEP method*), 172
 setBV() (*slepc4py.SLEPc.PEP method*), 212
 setBV() (*slepc4py.SLEPc.SVD method*), 270
 setCayleyAntishift() (*slepc4py.SLEPc.ST method*),
 245
 setCISSExttraction() (*slepc4py.SLEPc.EPS method*),
 96
 setCISSExttraction() (*slepc4py.SLEPc.NEP method*),
 172
 setCISSExttraction() (*slepc4py.SLEPc.PEP method*),
 212
 setCISSQuadRule() (*slepc4py.SLEPc.EPS method*), 97
 setCISSRefinement() (*slepc4py.SLEPc.EPS method*),
 97
 setCISSRefinement() (*slepc4py.SLEPc.NEP method*),
 172
 setCISSRefinement() (*slepc4py.SLEPc.PEP method*),
 213
 setCISSSizes() (*slepc4py.SLEPc.EPS method*), 97
 setCISSSizes() (*slepc4py.SLEPc.NEP method*), 173
 setCISSSizes() (*slepc4py.SLEPc.PEP method*), 213
 setCISSThreshold() (*slepc4py.SLEPc.EPS method*),
 97
 setCISSThreshold() (*slepc4py.SLEPc.NEP method*),
 173
 setCISSThreshold() (*slepc4py.SLEPc.PEP method*),
 213
 setCISSUseST() (*slepc4py.SLEPc.EPS method*), 98
 setCoefficients() (*slepc4py.SLEPc.LME method*),
 134
 setCombineChildren() (*slepc4py.SLEPc.FN method*),
 124
 setCompact() (*slepc4py.SLEPc.DS method*), 52
 setComplement() (*slepc4py.SLEPc.RG method*), 232
 setConvergenceTest() (*slepc4py.SLEPc.EPS method*), 98
 setConvergenceTest() (*slepc4py.SLEPc.NEP method*), 173
 setConvergenceTest() (*slepc4py.SLEPc.PEP method*), 214
 setConvergenceTest() (*slepc4py.SLEPc.SVD method*), 271
 setCrossEPS() (*slepc4py.SLEPc.SVD method*), 271
 setCrossExplicitMatrix() (*slepc4py.SLEPc.SVD method*), 271
 setCyclicEPS() (*slepc4py.SLEPc.SVD method*), 271
 setCyclicExplicitMatrix() (*slepc4py.SLEPc.SVD method*), 272
 setDefiniteTolerance() (*slepc4py.SLEPc.BV method*), 36
 setDeflationSpace() (*slepc4py.SLEPc.EPS method*),

setDimensions() (*slepc4py.SLEPc.DS method*), 52
 setDimensions() (*slepc4py.SLEPc.EPS method*), 99
 setDimensions() (*slepc4py.SLEPc.LME method*), 134
 setDimensions() (*slepc4py.SLEPc.MFN method*), 143
 setDimensions() (*slepc4py.SLEPc.NEP method*), 174
 setDimensions() (*slepc4py.SLEPc.PEP method*), 214
 setDimensions() (*slepc4py.SLEPc.SVD method*), 272
 setDS() (*slepc4py.SLEPc.EPS method*), 98
 setDS() (*slepc4py.SLEPc.NEP method*), 174
 setDS() (*slepc4py.SLEPc.PEP method*), 214
 setDS() (*slepc4py.SLEPc.SVD method*), 272
 setEigenvalueComparison() (*slepc4py.SLEPc.EPS method*), 99
 setEllipseParameters() (*slepc4py.SLEPc.RG method*), 232
 setErrorIfNotConverged() (*slepc4py.SLEPc.LME method*), 134
 setErrorIfNotConverged() (*slepc4py.SLEPc.MFN method*), 144
 setExtract() (*slepc4py.SLEPc.PEP method*), 214
 setExtraction() (*slepc4py.SLEPc.EPS method*), 100
 setExtraRow() (*slepc4py.SLEPc.DS method*), 53
 setFilterDamping() (*slepc4py.SLEPc.ST method*),
 245
 setFilterDegree() (*slepc4py.SLEPc.ST method*), 246
 setFilterInterval() (*slepc4py.SLEPc.ST method*),
 246
 setFilterRange() (*slepc4py.SLEPc.ST method*), 246
 setFilterType() (*slepc4py.SLEPc.ST method*), 247
 setFN() (*slepc4py.SLEPc.MFN method*), 144
 setFromOptions() (*slepc4py.SLEPc.BV method*), 36
 setFromOptions() (*slepc4py.SLEPc.DS method*), 53
 setFromOptions() (*slepc4py.SLEPc.EPS method*), 100
 setFromOptions() (*slepc4py.SLEPc.FN method*), 124
 setFromOptions() (*slepc4py.SLEPc.LME method*),
 135
 setFromOptions() (*slepc4py.SLEPc.MFN method*),
 144
 setFromOptions() (*slepc4py.SLEPc.NEP method*), 174
 setFromOptions() (*slepc4py.SLEPc.PEP method*), 214
 setFromOptions() (*slepc4py.SLEPc.RG method*), 233
 setFromOptions() (*slepc4py.SLEPc.ST method*), 247
 setFromOptions() (*slepc4py.SLEPc.SVD method*), 272
 setFunction() (*slepc4py.SLEPc.NEP method*), 174
 setGDBlockSize() (*slepc4py.SLEPc.EPS method*), 100
 setGDBOrth() (*slepc4py.SLEPc.EPS method*), 100
 setGDDoubleExpansion() (*slepc4py.SLEPc.EPS method*), 101
 setGDInitialSize() (*slepc4py.SLEPc.EPS method*),
 101
 setGDKrylovStart() (*slepc4py.SLEPc.EPS method*),
 101
 setGDRestart() (*slepc4py.SLEPc.EPS method*), 101

setGSVDDimensions() (*slepc4py.SLEPc.DS method*), 53
 setHSVDDimensions() (*slepc4py.SLEPc.DS method*), 54
 setIdentity() (*slepc4py.SLEPc.DS method*), 54
 setImplicitTranspose() (*slepc4py.SLEPc.SVD method*), 273
 setInitialSpace() (*slepc4py.SLEPc.EPS method*), 102
 setInitialSpace() (*slepc4py.SLEPc.NEP method*), 175
 setInitialSpace() (*slepc4py.SLEPc.PEP method*), 215
 setInitialSpace() (*slepc4py.SLEPc.SVD method*), 273
 setInterpolInterpolation() (*slepc4py.SLEPc.NEP method*), 175
 setInterpolPEP() (*slepc4py.SLEPc.NEP method*), 175
 setInterval() (*slepc4py.SLEPc.EPS method*), 102
 setInterval() (*slepc4py.SLEPc.PEP method*), 215
 setIntervalEndpoints() (*slepc4py.SLEPc.RG method*), 233
 setJacobian() (*slepc4py.SLEPc.NEP method*), 175
 setJDBlockSize() (*slepc4py.SLEPc.EPS method*), 102
 setJDBOrth() (*slepc4py.SLEPc.EPS method*), 102
 setJDConstCorrectionTol() (*slepc4py.SLEPc.EPS method*), 103
 setJDFix() (*slepc4py.SLEPc.EPS method*), 103
 setJDFix() (*slepc4py.SLEPc.PEP method*), 215
 setJDInitialSize() (*slepc4py.SLEPc.EPS method*), 103
 setJDKrylovStart() (*slepc4py.SLEPc.EPS method*), 103
 setJDMinimalityIndex() (*slepc4py.SLEPc.PEP method*), 215
 setJDProjection() (*slepc4py.SLEPc.PEP method*), 216
 setJDRestart() (*slepc4py.SLEPc.EPS method*), 104
 setJDRestart() (*slepc4py.SLEPc.PEP method*), 216
 setJDReusePreconditioner() (*slepc4py.SLEPc.PEP method*), 216
 setKrylovSchurBSEType() (*slepc4py.SLEPc.EPS method*), 104
 setKrylovSchurDetectZeros() (*slepc4py.SLEPc.EPS method*), 104
 setKrylovSchurDimensions() (*slepc4py.SLEPc.EPS method*), 105
 setKrylovSchurLocking() (*slepc4py.SLEPc.EPS method*), 105
 setKrylovSchurPartitions() (*slepc4py.SLEPc.EPS method*), 105
 setKrylovSchurRestart() (*slepc4py.SLEPc.EPS method*), 106
 setKrylovSchurSubintervals() (*slepc4py.SLEPc.EPS method*), 106
 setKSP() (*slepc4py.SLEPc.ST method*), 247
 setLanczosOneSide() (*slepc4py.SLEPc.SVD method*), 273
 setLanczosReorthogType() (*slepc4py.SLEPc.EPS method*), 107
 setLeadingDimension() (*slepc4py.SLEPc.BV method*), 36
 setLeftInitialSpace() (*slepc4py.SLEPc.EPS method*), 107
 setLinearEPS() (*slepc4py.SLEPc.PEP method*), 216
 setLinearExplicitMatrix() (*slepc4py.SLEPc.PEP method*), 217
 setLinearLinearization() (*slepc4py.SLEPc.PEP method*), 217
 setLOBPCGBlockSize() (*slepc4py.SLEPc.EPS method*), 106
 setLOBPCGLocking() (*slepc4py.SLEPc.EPS method*), 106
 setLOBPCGRestart() (*slepc4py.SLEPc.EPS method*), 107
 setLyapIIRanks() (*slepc4py.SLEPc.EPS method*), 108
 setMatMode() (*slepc4py.SLEPc.ST method*), 247
 setMatMultMethod() (*slepc4py.SLEPc.BV method*), 36
 setMatrices() (*slepc4py.SLEPc.ST method*), 248
 setMatrix() (*slepc4py.SLEPc.BV method*), 36
 setMatStructure() (*slepc4py.SLEPc.ST method*), 248
 setMethod() (*slepc4py.SLEPc.DS method*), 54
 setMethod() (*slepc4py.SLEPc.FN method*), 124
 setMonitor() (*slepc4py.SLEPc.EPS method*), 108
 setMonitor() (*slepc4py.SLEPc.LME method*), 135
 setMonitor() (*slepc4py.SLEPc.MFN method*), 144
 setMonitor() (*slepc4py.SLEPc.NEP method*), 176
 setMonitor() (*slepc4py.SLEPc.PEP method*), 217
 setMonitor() (*slepc4py.SLEPc.SVD method*), 274
 setNArnoldiKSP() (*slepc4py.SLEPc.NEP method*), 176
 setNArnoldiLagPreconditioner() (*slepc4py.SLEPc.NEP method*), 176
 setNLEIGSEPS() (*slepc4py.SLEPc.NEP method*), 176
 setNLEIGSFullBasis() (*slepc4py.SLEPc.NEP method*), 177
 setNLEIGSInterpolation() (*slepc4py.SLEPc.NEP method*), 177
 setNLEIGSLocking() (*slepc4py.SLEPc.NEP method*), 177
 setNLEIGSRestart() (*slepc4py.SLEPc.NEP method*), 178
 setNLEIGSRKShifts() (*slepc4py.SLEPc.NEP method*), 177
 setNumConstraints() (*slepc4py.SLEPc.BV method*), 37
 setOperator() (*slepc4py.SLEPc.MFN method*), 145
 setOperators() (*slepc4py.SLEPc.EPS method*), 108
 setOperators() (*slepc4py.SLEPc.PEP method*), 217

setOperators() (*slepc4py.SLEPc.SVD method*), 274
 setOptionsPrefix() (*slepc4py.SLEPc.BV method*), 37
 setOptionsPrefix() (*slepc4py.SLEPc.DS method*), 54
 setOptionsPrefix() (*slepc4py.SLEPc.EPS method*), 108
 setOptionsPrefix() (*slepc4py.SLEPc.FN method*), 124
 setOptionsPrefix() (*slepc4py.SLEPc.LME method*), 135
 setOptionsPrefix() (*slepc4py.SLEPc.MFN method*), 145
 setOptionsPrefix() (*slepc4py.SLEPc.NEP method*), 178
 setOptionsPrefix() (*slepc4py.SLEPc.PEP method*), 217
 setOptionsPrefix() (*slepc4py.SLEPc.RG method*), 233
 setOptionsPrefix() (*slepc4py.SLEPc.ST method*), 249
 setOptionsPrefix() (*slepc4py.SLEPc.SVD method*), 274
 setOrthogonalization() (*slepc4py.SLEPc.BV method*), 37
 setParallel() (*slepc4py.SLEPc.DS method*), 55
 setParallel() (*slepc4py.SLEPc.FN method*), 125
 setPEPCoefficients() (*slepc4py.SLEPc.DS method*), 54
 setPEPDegree() (*slepc4py.SLEPc.DS method*), 55
 setPhiIndex() (*slepc4py.SLEPc.FN method*), 125
 setPolygonVertices() (*slepc4py.SLEPc.RG method*), 233
 setPowerShiftType() (*slepc4py.SLEPc.EPS method*), 109
 setPreconditionerMat() (*slepc4py.SLEPc.ST method*), 249
 setProblemType() (*slepc4py.SLEPc.EPS method*), 109
 setProblemType() (*slepc4py.SLEPc.LME method*), 135
 setProblemType() (*slepc4py.SLEPc.NEP method*), 178
 setProblemType() (*slepc4py.SLEPc.PEP method*), 218
 setProblemType() (*slepc4py.SLEPc.SVD method*), 275
 setPurify() (*slepc4py.SLEPc.EPS method*), 110
 setQArnoldiLocking() (*slepc4py.SLEPc.PEP method*), 218
 setQArnoldiRestart() (*slepc4py.SLEPc.PEP method*), 218
 setRandom() (*slepc4py.SLEPc.BV method*), 38
 setRandomColumn() (*slepc4py.SLEPc.BV method*), 38
 setRandomCond() (*slepc4py.SLEPc.BV method*), 38
 setRandomContext() (*slepc4py.SLEPc.BV method*), 38
 setRandomNormal() (*slepc4py.SLEPc.BV method*), 39
 setRandomSign() (*slepc4py.SLEPc.BV method*), 39
 setRationalDenominator() (*slepc4py.SLEPc.FN method*), 125
 setRationalNumerator() (*slepc4py.SLEPc.FN method*), 125
 setRefine() (*slepc4py.SLEPc.NEP method*), 180
 setRefine() (*slepc4py.SLEPc.PEP method*), 219
 setRefined() (*slepc4py.SLEPc.DS method*), 55
 setRG() (*slepc4py.SLEPc.EPS method*), 110
 setRG() (*slepc4py.SLEPc.NEP method*), 178
 setRG() (*slepc4py.SLEPc.PEP method*), 219
 setRHS() (*slepc4py.SLEPc.LME method*), 136
 setRIIConstCorrectionTol() (*slepc4py.SLEPc.NEP method*), 178
 setRIIDeflationThreshold() (*slepc4py.SLEPc.NEP method*), 179
 setRIIHermitian() (*slepc4py.SLEPc.NEP method*), 179
 setRIIKSP() (*slepc4py.SLEPc.NEP method*), 179
 setRIILagPreconditioner() (*slepc4py.SLEPc.NEP method*), 179
 setRIIMaximumIterations() (*slepc4py.SLEPc.NEP method*), 180
 setRingParameters() (*slepc4py.SLEPc.RG method*), 234
 setRQCGReset() (*slepc4py.SLEPc.EPS method*), 110
 setScale() (*slepc4py.SLEPc.FN method*), 125
 setScale() (*slepc4py.SLEPc.PEP method*), 221
 setScale() (*slepc4py.SLEPc.RG method*), 234
 setShift() (*slepc4py.SLEPc.ST method*), 249
 setSignature() (*slepc4py.SLEPc.SVD method*), 275
 setSizes() (*slepc4py.SLEPc.BV method*), 39
 setSizesFromVec() (*slepc4py.SLEPc.BV method*), 39
 setSLPDeflationThreshold() (*slepc4py.SLEPc.NEP method*), 180
 setSLPEPS() (*slepc4py.SLEPc.NEP method*), 180
 setSLPEPSLeft() (*slepc4py.SLEPc.NEP method*), 181
 setSLPKSP() (*slepc4py.SLEPc.NEP method*), 181
 setSolution() (*slepc4py.SLEPc.LME method*), 136
 setSplitOperator() (*slepc4py.SLEPc.NEP method*), 181
 setSplitPreconditioner() (*slepc4py.SLEPc.NEP method*), 181
 setSplitPreconditioner() (*slepc4py.SLEPc.ST method*), 249
 setST() (*slepc4py.SLEPc.EPS method*), 110
 setST() (*slepc4py.SLEPc.PEP method*), 219
 setState() (*slepc4py.SLEPc.DS method*), 56
 setSTOARCheckEigenvalueType() (*slepc4py.SLEPc.PEP method*), 219
 setSTOARDetectZeros() (*slepc4py.SLEPc.PEP method*), 220
 setSTOARDimensions() (*slepc4py.SLEPc.PEP method*), 220
 setSTOARLinearization() (*slepc4py.SLEPc.PEP method*), 220
 setSTOARLocking() (*slepc4py.SLEPc.PEP method*),

220
 setStoppingTest() (*slepc4py.SLEPc.EPS method*), 110
 setStoppingTest() (*slepc4py.SLEPc.NEP method*), 182
 setStoppingTest() (*slepc4py.SLEPc.PEP method*), 221
 setStoppingTest() (*slepc4py.SLEPc.SVD method*), 275
 setSVDDimensions() (*slepc4py.SLEPc.DS method*), 55
 setTarget() (*slepc4py.SLEPc.EPS method*), 111
 setTarget() (*slepc4py.SLEPc.NEP method*), 182
 setTarget() (*slepc4py.SLEPc.PEP method*), 222
 setThreshold() (*slepc4py.SLEPc.EPS method*), 111
 setThreshold() (*slepc4py.SLEPc.SVD method*), 277
 setTOARLocking() (*slepc4py.SLEPc.PEP method*), 221
 setTOARRestart() (*slepc4py.SLEPc.PEP method*), 222
 setTolerances() (*slepc4py.SLEPc.EPS method*), 111
 setTolerances() (*slepc4py.SLEPc.LME method*), 136
 setTolerances() (*slepc4py.SLEPc.MFN method*), 145
 setTolerances() (*slepc4py.SLEPc.NEP method*), 182
 setTolerances() (*slepc4py.SLEPc.PEP method*), 222
 setTolerances() (*slepc4py.SLEPc.SVD method*), 277
 setTrackAll() (*slepc4py.SLEPc.EPS method*), 112
 setTrackAll() (*slepc4py.SLEPc.NEP method*), 182
 setTrackAll() (*slepc4py.SLEPc.PEP method*), 223
 setTrackAll() (*slepc4py.SLEPc.SVD method*), 278
 setTransform() (*slepc4py.SLEPc.ST method*), 250
 setTRLanczosExplicitMatrix() (*slepc4py.SLEPc.SVD method*), 275
 setTRLanczosGBidiag() (*slepc4py.SLEPc.SVD method*), 275
 setTRLanczosKSP() (*slepc4py.SLEPc.SVD method*), 276
 setTRLanczosLocking() (*slepc4py.SLEPc.SVD method*), 276
 setTRLanczosOneSide() (*slepc4py.SLEPc.SVD method*), 276
 setTRLanczosRestart() (*slepc4py.SLEPc.SVD method*), 277
 setTrueResidual() (*slepc4py.SLEPc.EPS method*), 112
 setTwoSided() (*slepc4py.SLEPc.EPS method*), 112
 setTwoSided() (*slepc4py.SLEPc.NEP method*), 182
 setType() (*slepc4py.SLEPc.BV method*), 40
 setType() (*slepc4py.SLEPc.DS method*), 56
 setType() (*slepc4py.SLEPc.EPS method*), 112
 setType() (*slepc4py.SLEPc.FN method*), 126
 setType() (*slepc4py.SLEPc.LME method*), 136
 setType() (*slepc4py.SLEPc.MFN method*), 145
 setType() (*slepc4py.SLEPc.NEP method*), 183
 setType() (*slepc4py.SLEPc.PEP method*), 223
 setType() (*slepc4py.SLEPc.RG method*), 234
 setType() (*slepc4py.SLEPc.ST method*), 250
 setType() (*slepc4py.SLEPc.SVD method*), 278
 setUp() (*slepc4py.SLEPc.EPS method*), 113
 setUp() (*slepc4py.SLEPc.LME method*), 136
 setUp() (*slepc4py.SLEPc.MFN method*), 145
 setUp() (*slepc4py.SLEPc.NEP method*), 183
 setUp() (*slepc4py.SLEPc.PEP method*), 223
 setUp() (*slepc4py.SLEPc.ST method*), 250
 setUp() (*slepc4py.SLEPc.SVD method*), 278
 setVecType() (*slepc4py.SLEPc.BV method*), 40
 setWhichEigenpairs() (*slepc4py.SLEPc.EPS method*), 113
 setWhichEigenpairs() (*slepc4py.SLEPc.NEP method*), 183
 setWhichEigenpairs() (*slepc4py.SLEPc.PEP method*), 223
 setWhichSingularTriplets() (*slepc4py.SLEPc.SVD method*), 278
 SHAO (*slepc4py.SLEPc.EPS.KrylovSchurBSEType attribute*), 63
 SHAO (*slepc4py.SLEPc.EPSKrylovSchurBSEType attribute*), 116
 SHELL (*slepc4py.SLEPc.ST.MatMode attribute*), 237
 SHELL (*slepc4py.SLEPc.ST.Type attribute*), 237
 shift (*slepc4py.SLEPc.ST attribute*), 251
 SHIFT (*slepc4py.SLEPc.ST.Type attribute*), 237
 SIMPLE (*slepc4py.SLEPc.NEP.Refine attribute*), 151
 SIMPLE (*slepc4py.SLEPc.PEP.Refine attribute*), 191
 SINGLE (*slepc4py.SLEPc.SVD.TRLanczosGBidiag attribute*), 256
 SINVERT (*slepc4py.SLEPc.ST.Type attribute*), 237
 size (*slepc4py.SLEPc.BV attribute*), 40
 sizes (*slepc4py.SLEPc.BV attribute*), 41
 slepc4py
 module, 10
 slepc4py.SLEPc
 module, 15
 slepc4py.typing
 module, 11
 SLEPC_DIR, 8, 9
 SLP (*slepc4py.SLEPc.NEP.Type attribute*), 153
 SMALLEST (*slepc4py.SLEPc.SVD.Which attribute*), 258
 SMALLEST_IMAGINARY (*slepc4py.SLEPc.EPS.Which attribute*), 70
 SMALLEST_IMAGINARY (*slepc4py.SLEPc.NEP.Which attribute*), 154
 SMALLEST_IMAGINARY (*slepc4py.SLEPc.PEP.Which attribute*), 195
 SMALLEST_MAGNITUDE (*slepc4py.SLEPc.EPS.Which attribute*), 70
 SMALLEST_MAGNITUDE (*slepc4py.SLEPc.NEP.Which attribute*), 154
 SMALLEST_MAGNITUDE (*slepc4py.SLEPc.PEP.Which attribute*), 195

SMALLEST_REAL (*slepc4py.SLEPc.EPS.Which attribute*), 70
SMALLEST_REAL (*slepc4py.SLEPc.NEP.Which attribute*), 154
SMALLEST_REAL (*slepc4py.SLEPc.PEP.Which attribute*), 195
solve() (*slepc4py.SLEPc.DS method*), 56
solve() (*slepc4py.SLEPc.EPS method*), 113
solve() (*slepc4py.SLEPc.LME method*), 137
solve() (*slepc4py.SLEPc.MFN method*), 146
solve() (*slepc4py.SLEPc.NEP method*), 183
solve() (*slepc4py.SLEPc.PEP method*), 223
solve() (*slepc4py.SLEPc.SVD method*), 279
solveTranspose() (*slepc4py.SLEPc.MFN method*), 146
SQRT (*slepc4py.SLEPc.FN.Type attribute*), 118
ST (*class in slepc4py.SLEPc*), 235
st (*slepc4py.SLEPc.EPS attribute*), 115
st (*slepc4py.SLEPc.PEP attribute*), 225
STANDARD (*slepc4py.SLEPc.SVD.ProblemType attribute*), 255
state (*slepc4py.SLEPc.DS attribute*), 58
StateType (*class in slepc4py.SLEPc.DS*), 44
STEIN (*slepc4py.SLEPc.LME.ProblemType attribute*), 128
STFilterDamping (*class in slepc4py.SLEPc*), 251
STFilterType (*class in slepc4py.SLEPc*), 252
STOAR (*slepc4py.SLEPc.PEP.Type attribute*), 194
Stop (*class in slepc4py.SLEPc.EPS*), 66
Stop (*class in slepc4py.SLEPc.NEP*), 152
Stop (*class in slepc4py.SLEPc.PEP*), 193
Stop (*class in slepc4py.SLEPc.SVD*), 255
STRUCTURED (*slepc4py.SLEPc.PEP.Extract attribute*), 190
SUBSPACE (*slepc4py.SLEPc.EPS.Type attribute*), 69
SVD (*class in slepc4py.SLEPc*), 252
SVD (*slepc4py.SLEPc.DS.Type attribute*), 45
SVDMonitorFunction (*in module slepc4py.typing*), 14
SVDStoppingFunction (*in module slepc4py.typing*), 14
SVEC (*slepc4py.SLEPc.BV.Type attribute*), 18
SVQB (*slepc4py.SLEPc.BV.OrthogBlockType attribute*), 16
SYLVESTER (*slepc4py.SLEPc.LME.ProblemType attribute*), 128
SYNCHRONIZED (*slepc4py.SLEPc.DS.ParallelType attribute*), 43
SYNCHRONIZED (*slepc4py.SLEPc.FN.ParallelType attribute*), 118
Sys (*class in slepc4py.SLEPc*), 280

T

T (*slepc4py.SLEPc.DS.MatType attribute*), 43
target (*slepc4py.SLEPc.EPS attribute*), 115
target (*slepc4py.SLEPc.NEP attribute*), 185
target (*slepc4py.SLEPc.PEP attribute*), 225
TARGET_IMAGINARY (*slepc4py.SLEPc.EPS.Which attribute*), 70
TARGET_IMAGINARY (*slepc4py.SLEPc.NEP.Which attribute*), 154
TARGET_IMAGINARY (*slepc4py.SLEPc.PEP.Which attribute*), 195
TARGET_MAGNITUDE (*slepc4py.SLEPc.EPS.Which attribute*), 70
TARGET_MAGNITUDE (*slepc4py.SLEPc.NEP.Which attribute*), 154
TARGET_MAGNITUDE (*slepc4py.SLEPc.PEP.Which attribute*), 195
TARGET_REAL (*slepc4py.SLEPc.EPS.Which attribute*), 70
TARGET_REAL (*slepc4py.SLEPc.NEP.Which attribute*), 154
TARGET_REAL (*slepc4py.SLEPc.PEP.Which attribute*), 195
TENSOR (*slepc4py.SLEPc.BV.Type attribute*), 18
THRESHOLD (*slepc4py.SLEPc.EPS.Stop attribute*), 66
THRESHOLD (*slepc4py.SLEPc.SVD.Stop attribute*), 256
TOAR (*slepc4py.SLEPc.PEP.Type attribute*), 194
tol (*slepc4py.SLEPc.EPS attribute*), 115
tol (*slepc4py.SLEPc.LME attribute*), 137
tol (*slepc4py.SLEPc.MFN attribute*), 147
tol (*slepc4py.SLEPc.NEP attribute*), 185
tol (*slepc4py.SLEPc.PEP attribute*), 225
tol (*slepc4py.SLEPc.SVD attribute*), 280
track_all (*slepc4py.SLEPc.EPS attribute*), 116
track_all (*slepc4py.SLEPc.NEP attribute*), 185
track_all (*slepc4py.SLEPc.PEP attribute*), 225
track_all (*slepc4py.SLEPc.SVD attribute*), 280
transform (*slepc4py.SLEPc.ST attribute*), 251
transpose_mode (*slepc4py.SLEPc.SVD attribute*), 280
TRAPEZOIDAL (*slepc4py.SLEPc.EPS.CISSQuadRule attribute*), 60
TRAPEZOIDAL (*slepc4py.SLEPc.RG.QuadRule attribute*), 226
TRLANCZOS (*slepc4py.SLEPc.SVD.Type attribute*), 258
TRLanczosGBidiag (*class in slepc4py.SLEPc.SVD*), 256
true_residual (*slepc4py.SLEPc.EPS attribute*), 116
truncate() (*slepc4py.SLEPc.DS method*), 56
TRUNCATED (*slepc4py.SLEPc.DS.StateType attribute*), 44
TSQR (*slepc4py.SLEPc.BV.OrthogBlockType attribute*), 16
TSQRCHOL (*slepc4py.SLEPc.BV.OrthogBlockType attribute*), 16
two_sided (*slepc4py.SLEPc.EPS attribute*), 116
TWOSIDE (*slepc4py.SLEPc.EPS.Balance attribute*), 59
Type (*class in slepc4py.SLEPc.BV*), 18
Type (*class in slepc4py.SLEPc.DS*), 44
Type (*class in slepc4py.SLEPc.EPS*), 66
Type (*class in slepc4py.SLEPc.FN*), 118
Type (*class in slepc4py.SLEPc.LME*), 128

Type (class in *slepc4py.SLEPc.MFN*), 139
 Type (class in *slepc4py.SLEPc.NEP*), 152
 Type (class in *slepc4py.SLEPc.PEP*), 193
 Type (class in *slepc4py.SLEPc.RG*), 226
 Type (class in *slepc4py.SLEPc.ST*), 237
 Type (class in *slepc4py.SLEPc.SVD*), 257

U

U (*slepc4py.SLEPc.DS.MatType* attribute), 43
 updateExtraRow() (*slepc4py.SLEPc.DS* method), 57
 updateKrylovSchurSubcommMats()
 (*slepc4py.SLEPc.EPS* method), 113
 UPPER (*slepc4py.SLEPc.SVD.TRLanczosGBidiag* at-
 tribute), 256
 USER (*slepc4py.SLEPc.EPS.Balance* attribute), 59
 USER (*slepc4py.SLEPc.EPS.Conv* attribute), 60
 USER (*slepc4py.SLEPc.EPS.Stop* attribute), 66
 USER (*slepc4py.SLEPc.EPS.Which* attribute), 70
 USER (*slepc4py.SLEPc.NEP.Conv* attribute), 148
 USER (*slepc4py.SLEPc.NEP.Stop* attribute), 152
 USER (*slepc4py.SLEPc.NEP.Which* attribute), 154
 USER (*slepc4py.SLEPc.PEP.Conv* attribute), 187
 USER (*slepc4py.SLEPc.PEP.Stop* attribute), 193
 USER (*slepc4py.SLEPc.PEP.Which* attribute), 195
 USER (*slepc4py.SLEPc.SVD.Conv* attribute), 253
 USER (*slepc4py.SLEPc.SVD.Stop* attribute), 256
 Util (class in *slepc4py.SLEPc*), 282

V

V (*slepc4py.SLEPc.DS.MatType* attribute), 43
 valuesView() (*slepc4py.SLEPc.EPS* method), 114
 valuesView() (*slepc4py.SLEPc.NEP* method), 183
 valuesView() (*slepc4py.SLEPc.PEP* method), 224
 valuesView() (*slepc4py.SLEPc.SVD* method), 279
 VECS (*slepc4py.SLEPc.BV.MatMultType* attribute), 16
 VECS (*slepc4py.SLEPc.BV.Type* attribute), 18
 vectors() (*slepc4py.SLEPc.DS* method), 57
 vectorsView() (*slepc4py.SLEPc.EPS* method), 114
 vectorsView() (*slepc4py.SLEPc.NEP* method), 184
 vectorsView() (*slepc4py.SLEPc.PEP* method), 224
 vectorsView() (*slepc4py.SLEPc.SVD* method), 279
 view() (*slepc4py.SLEPc.BV* method), 40
 view() (*slepc4py.SLEPc.DS* method), 57
 view() (*slepc4py.SLEPc.EPS* method), 115
 view() (*slepc4py.SLEPc.FN* method), 126
 view() (*slepc4py.SLEPc.LME* method), 137
 view() (*slepc4py.SLEPc.MFN* method), 146
 view() (*slepc4py.SLEPc.NEP* method), 184
 view() (*slepc4py.SLEPc.PEP* method), 224
 view() (*slepc4py.SLEPc.RG* method), 234
 view() (*slepc4py.SLEPc.ST* method), 250
 view() (*slepc4py.SLEPc.SVD* method), 279

W

W (*slepc4py.SLEPc.DS.MatType* attribute), 43
 Which (class in *slepc4py.SLEPc.EPS*), 69
 Which (class in *slepc4py.SLEPc.NEP*), 153
 Which (class in *slepc4py.SLEPc.PEP*), 194
 Which (class in *slepc4py.SLEPc.SVD*), 258
 which (*slepc4py.SLEPc.EPS* attribute), 116
 which (*slepc4py.SLEPc.NEP* attribute), 185
 which (*slepc4py.SLEPc.PEP* attribute), 225
 which (*slepc4py.SLEPc.SVD* attribute), 280
 WILKINSON (*slepc4py.SLEPc.EPS.PowerShiftType*
 attribute), 65

X

X (*slepc4py.SLEPc.DS.MatType* attribute), 43

Y

Y (*slepc4py.SLEPc.DS.MatType* attribute), 43

Z

Z (*slepc4py.SLEPc.DS.MatType* attribute), 43