SLEPc: Scalable Library for Eigenvalue Problem Computations

Jose E. Roman

High Performance Networking and Computing Group (GRyCAP)
Universidad Politécnica de Valencia, Spain

August, 2005

UNIVERSIDAD POLITECNICA DE VALENCIA

grycap
Grupo de Redes y Computación de Altas Prestaciones

SLEPc

Tutorial Outline

Eigenproblems: Motivation

Large sparse eigenvalue problems are among the most demanding calculations in scientific computing

Example application areas:

- Dynamic structural analysis (e.g. civil engineering)
- Stability analysis (e.g. control engineering)
- Eigenfunction determination (e.g. quantum chemistry)
- Bifurcation analysis (e.g. fluid dynamics)
- Statistics / information retrieval (e.g. Google's PageRank)

Motivating Example 1: Nuclear Engineering

Modal analysis of nuclear reactor cores

Objectives:

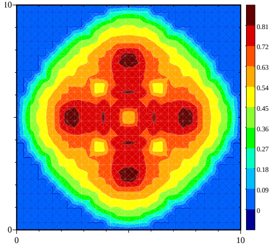
- Improve safety
- Reduce operation costs

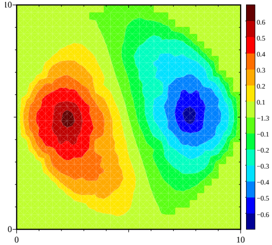
Lambda Modes Equation

$$\mathcal{L}\phi = \frac{1}{\lambda}\mathcal{M}\phi$$

Target: modes associated to largest λ

- Criticality (eigenvalues)
- Prediction of instabilities and transient analysis (eigenvectors)





Motivating Example 1: Nuclear Engineering (cont'd)

Discretized eigenproblem

$$\begin{bmatrix} L_{11} & 0 \\ -L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix} = \frac{1}{\lambda} \begin{bmatrix} M_{11} & M_{12} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix}$$

Can be restated as

$$N\psi_1 = \lambda L_{11}\psi_1 \ , \quad N = M_{11} + M_{12}L_{22}^{-1}L_{21}$$

- Generalized eigenvalue problem
- Matrix should not be computed explicitly
- In some applications, many successive problems are solved

Motivating Example 2: Computational Electromagnetics

Objective: Analysis of resonant cavities

Source-free wave equations

$$\nabla \times (\hat{\mu}_r^{-1} \nabla \times \vec{E}) - \kappa_0^2 \hat{\varepsilon}_r \vec{E} = 0$$
$$\nabla \times (\hat{\varepsilon}_r^{-1} \nabla \times \vec{H}) - \kappa_0^2 \hat{\mu}_r \vec{H} = 0$$

Target: A few smallest nonzero eigenfrequencies

Discretization: 1^{st} order edge finite elements (tetrahedral)

$$Ax = \kappa_0^2 Bx$$

Generalized Eigenvalue Problem

- A and B are large and sparse, possibly complex
- A is (complex) symmetric and semi-positive definite
- B is (complex) symmetric and positive definite

Motivating Example 2: Comp. Electromagnetics (cont'd)

Matrix A has a high-dimensional null space, $\mathcal{N}(A)$

- The problem $Ax = \kappa_0^2 Bx$ has many zero eigenvalues
- These eigenvalues should be avoided during computation

$$\underbrace{\lambda_1, \lambda_2, \ldots, \lambda_k}_{=0}, \underbrace{\lambda_{k+1}, \lambda_{k+2}}_{\text{Target}}, \ldots, \lambda_n$$

Eigenfunctions associated to 0 are irrotational electric fields, $\vec{E} = -\nabla\Phi$. This allows the computation of a basis of $\mathcal{N}(A)$

Constrained Eigenvalue Problem

$$\left.\begin{array}{c} Ax = \kappa_0^2 Bx \\ C^T Bx = 0 \end{array}\right\}$$

where the columns of C span $\mathcal{N}(A)$

SLEPc

Facts Observed from the Examples

- Many formulations
  - Not all eigenproblems are formulated as simply $Ax = \lambda x$ or $Ax = \lambda Bx$
  - We have to account for: spectral transformations, block-structured problems, constrained problems, etc.
- Wanted solutions
  - Many ways of specifying which solutions must be sought
  - We have to account for: different extreme eigenvalues as well as interior ones
- Various problem characteristics
  - Problems can be real/complex, Hermitian/non-Hermitian

Goal: provide a uniform, coherent way of addressing these problems

Background on Eigenvalue Problems

Consider the following eigenvalue problems

Standard Eigenproblem

$$Ax = \lambda x$$

Generalized Eigenproblem

$$Ax = \lambda Bx$$

where

- λ is a (complex) scalar: *eigenvalue*
- x is a (complex) vector: *eigenvector*
- Matrices A and B can be real or complex
- Matrices A and B can be symmetric (Hermitian) or not
- Typically, B is symmetric positive (semi-) definite

Solution of the Eigenvalue Problem

There are n eigenvalues (counted with their multiplicities)

Partial eigensolution: nev solutions

$$\lambda_0, \lambda_1, \ldots, \lambda_{nev-1} \in \mathbb{C}$$
$$x_0, x_1, \ldots, x_{nev-1} \in \mathbb{C}^n$$

nev = number of eigenvalues / eigenvectors (eigenpairs)

Different requirements:

- Compute a few of the dominant eigenvalues (largest magnitude)
- Compute a few λ_i's with smallest or largest real parts
- Compute all λ_i's in a certain region of the complex plane

Single-Vector Methods

The following algorithm converges to the dominant eigenpair (λ_1, x_1), where $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$

Power Method

Set $y = v_0$
For $k = 1, 2, \ldots$
  $v = y/\|y\|_2$
  $y = Av$
  $\theta = v^* y$
  Check convergence
end

Notes:

- Only needs two vectors
- Deflation schemes to find subsequent eigenpairs
- Slow convergence (proportional to $|\lambda_1/\lambda_2|$)
- Fails if $|\lambda_1| = |\lambda_2|$

Variants of the Power Method

Shifted Power Method

- Example: Markov chain problem has two dominant eigenvalues $\lambda_1 = 1$, $\lambda_2 = -1 \implies$ Power Method fails!
- Solution: Apply the Power Method to matrix $A + \sigma I$

Inverse Iteration

- Observation: The eigenvectors of A and A^{-1} are identical
- The Power Method on $(A - \sigma I)^{-1}$ will compute the eigenvalues closest to σ

Rayleigh Quotient Iteration (RQI)

- Similar to Inverse Iteration but updating σ in each iteration

Spectral Transformation

A general technique that can be used in many methods

$$Ax = \lambda x \quad \Longrightarrow \quad Tx = \theta x$$

In the transformed problem

- The eigenvectors are not altered
- The eigenvalues are modified by a simple relation
- Convergence is usually improved (better separation)

Example transformations:

Shift of Origin

$$T_S = A + \sigma I$$

Shift-and-invert

$$T_{SI} = (A - \sigma I)^{-1}$$

Drawback: T not computed explicitly, linear solves instead

Invariant Subspace

A subspace $\mathcal{S}$ is called an *invariant subspace* of A if $A\mathcal{S} \subset \mathcal{S}$

- If $A \in \mathbb{C}^{n\times n}$, $V \in \mathbb{C}^{n\times k}$, and $H \in \mathbb{C}^{k\times k}$ satisfy

$$AV = VH$$

then $\mathcal{S} \equiv \mathcal{C}(V)$ is an invariant subspace of A

Objective: build an invariant subspace to extract the eigensolutions

Partial Schur Decomposition

$$AQ = QR$$

- Q has nev columns which are orthonormal
- R is a $nev \times nev$ upper (quasi-) triangular matrix

Projection Methods

The general scheme of a projection method:

1. Build an orthonormal basis of a certain subspace
2. Project the original problem onto this subspace
3. Use the solution of the projected problem to compute an approximate invariant subspace

- Different methods use different subspaces
  - Subspace Iteration: $A^k X$
  - Arnoldi, Lanczos: $\mathcal{K}_m(A, v_1) = \mathrm{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}$
- Dimension of the subspace: ncv (number of column vectors)
- Restart & deflation necessary until nev solutions converged

SLEPc

Summary

Observations to be added to the previous ones

- The solver computes only nev eigenpairs
- Internally, it works with ncv vectors
- Single-vector methods are very limited
- Projection methods are preferred
- Internally, solvers can be quite complex (deflation, restart, ...)
- Spectral transformations can be used irrespective of the solver
- Repeated linear solves may be required

Goal: hide eigensolver complexity and separate spectral transform

Executive Summary

SLEPc: Scalable Library for Eigenvalue Problem Computations

A *general* library for solving large-scale sparse eigenproblems on parallel computers

- For standard and generalized eigenproblems
- For real and complex arithmetic
- For Hermitian or non-Hermitian problems

Current version: 2.3.0 (released July 2005)

`http://www.grycap.upv.es/slepc`

SLEPc and PETSc

SLEPc extends PETSc for solving eigenvalue problems

PETSc: Portable, Extensible Toolkit for Scientific Computation

- Software for the solution of PDE's in parallel computers
- A freely available and supported research code
- Usable from C, C++, Fortran77/90
- Focus on abstraction, portability, interoperability, ...
- Object-oriented design (encapsulation, inheritance and polymorphism)
- Current: 2.3.0 `http://www.mcs.anl.gov/petsc`

SLEPc inherits all good properties of PETSc

Structure of SLEPc

SLEPc adds two new objects: EPS and ST

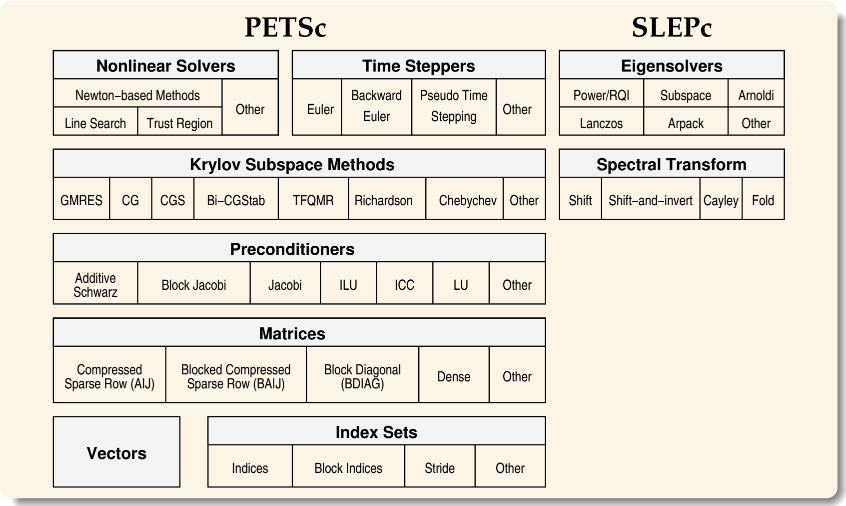
EPS: Eigenvalue Problem Solver

- The user specifies the problem via this object (entry point to SLEPc)
- Provides a collection of eigensolvers
- Allows the user to specify a number of parameters (e.g. which portion of the spectrum)

ST: Spectral Transformation

- Used to transform the original problem into $Tx = \theta x$
- Always associated to an EPS object, not used directly

SLEPc/PETSc Diagram



Basic Usage

Usual steps for solving an eigenvalue problem with SLEPc:

1. Create an EPS object
2. Define the eigenvalue problem
3. (Optionally) Specify options for the solution
4. Run the eigensolver
5. Retrieve the computed solution
6. Destroy the EPS object

All these operations are done via a generic interface, common to all the eigensolvers

Simple Example — Makefile

```
default: ex1

include ${SLEPC_DIR}/bmake/slepc_common

ex1: ex1.o chkopts
        -${CLINKER} -o ex1 ex1.o ${SLEPC_LIB}
        ${RM} ex1.o

ex1f: ex1f.o chkopts
        -${FLINKER} -o ex1f ex1f.o ${SLEPC_FORTRAN_LIB} \
                                   ${SLEPC_LIB}
        ${RM} ex1f.o
```

Simple Example

```
EPS        eps;      /* eigensolver context  */
Mat        A, B;     /* matrices of Ax=kBx   */
Vec        xr, xi;   /* eigenvector, x       */
PetscScalar kr, ki;  /* eigenvalue, k        */

EPSCreate(PETSC_COMM_WORLD, &eps);
EPSSetOperators(eps, A, B);
EPSSetProblemType(eps, EPS_GNHEP);
EPSSetFromOptions(eps);

EPSSolve(eps);

EPSGetConverged(eps, &nconv);
for (i=0; i<nconv; i++) {
  EPSGetEigenpair(eps, i, &kr, &ki, xr, xi);
}

EPSDestroy(eps);
```

Details: Object Management

EPS is managed like any other PETSc object

`EPSCreate(MPI_Comm comm,EPS *eps)`

Creates a new instance

EPS is a "parallel" object:

- Many operations are collective
- Parallel details are hidden from the programmer

`EPSDestroy(EPS eps)`

Destroys the instance

Details: Problem Definition

EPSSetOperators(EPS eps, Mat A, Mat B)

Used to pass the matrices that constitute the problem

- A generalized problem $Ax = \lambda Bx$ is specified by A and B
- For a standard problem $Ax = \lambda x$ set B=PETSC_NULL

EPSSetProblemType(EPS eps,EPSProblemType type)

Used to indicate the problem type

Problem Type	EPSProblemType	Command line key
Hermitian	EPS_HEP	-eps_hermitian
Generalized Hermitian	EPS_GHEP	-eps_gen_hermitian
Non-Hermitian	EPS_NHEP	-eps_non_hermitian
Generalized Non-Herm.	EPS_GNHEP	-eps_gen_non_hermitian

Details: Specification of Options

`EPSSetFromOptions(EPS eps)`

Looks in the command line for options related to EPS

For example, the following command line

```
% program -eps_hermitian
```

is equivalent to a call `EPSSetProblemType(eps,EPS_HEP)`

Other options have an associated function call

```
% program -eps_nev 6 -eps_tol 1e-8
```

`EPSView(EPS eps, PetscViewer viewer)`

Prints information about the object (equivalent to `-eps_view`)

Details: Viewing Current Options

Sample output of -eps_view

```
EPS Object:
  problem type: symmetric eigenvalue problem
  method: lanczos
  selected portion of spectrum: largest eigenvalues in magnitude
  number of eigenvalues (nev): 1
  number of column vectors (ncv): 16
  maximum number of iterations: 100
  tolerance: 1e-07
  orthogonalization method: classical Gram-Schmidt
  orthogonalization refinement: if needed (eta: 0.500000)
  dimension of user-provided deflation space: 0
  ST Object:
    type: shift
    shift: 0
```

SLEPc

Details: Solving the Problem

`EPSSolve(EPS eps)`

Launches the eigensolver

- Power Iteration with deflation
  - Includes Inverse Iteration and RQI
- Subspace Iteration with Rayleigh-Ritz projection and locking
- Arnoldi with explicit restart and deflation
- Lanczos with explicit restart and deflation
  - Reorthog. choices: local, full, selective, periodic, partial
- Interfaces to external software: ARPACK, etc.

SLEPc

Details: Retrieving the Solution

```
EPSGetConverged(EPS eps, int *nconv)
```

Returns the number of computed eigenpairs

The number of computed eigenpairs may differ from that requested

```
EPSGetEigenpair(EPS eps, int i, PetscScalar *kr,
                    PetscScalar *ki, Vec xr, Vec xi)
```

Returns the i-th solution of the eigenproblem

- kr Real part of the eigenvalue
- ki Imaginary part of the eigenvalue
- xr Real part of the eigenvector
- xi Imaginary part of the eigenvector

The eigenvalues are ordered according to certain criterion

Built-in Support Tools

- Plotting computed eigenvalues

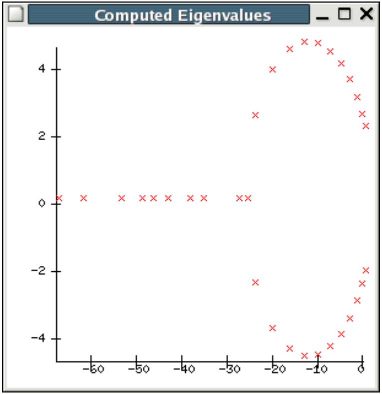
  `% program -eps_plot_eigs`

- Printing profiling information

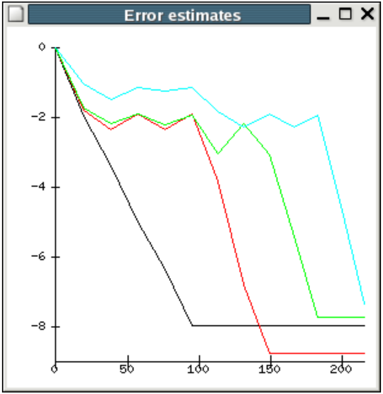
  `% program -log_summary`

- Debugging

  `% program -start_in_debugger`
  `% program -malloc_dump`



Built-in Support Tools



- Monitoring convergence (textually)

  % program -eps_monitor

- Monitoring convergence (graphically)

  % program -eps_xmonitor
  -draw_pause 1

Eigensolver Parameters

`EPSSetDimensions(EPS eps, int nev, int ncv)`

nev: Number of requested eigenvalues (`-eps_nev`)

ncv: Number of column vectors (i.e. largest dimension of the working subspace) (`-eps_ncv`)

- One may let SLEPc decide the value of ncv
- Typically, $ncv > 2 \cdot nev$, even larger if possible

`EPSSetTolerances(EPS eps, PetscReal tol, int max_it)`

tol: Tolerance for the convergence criterion (`-eps_tol`)

max_it: Maximum number of iterations (`-eps_max_it`)

Changing the Eigensolver

EPSSetType(EPS eps,EPSType type)

Used to specify the solution algorithm

Method	EPSType	-eps_type
Dense (LAPACK)	EPSLAPACK	lapack
Power / Inverse / RQI	EPSPOWER	power
Subspace Iteration	EPSSUBSPACE	subspace
Arnoldi Method	EPSARNOLDI	arnoldi
Lanczos Method	EPSLANCZOS	lanczos
Interface to ARPACK	EPSARPACK	arpack
Interface to BLZPACK	EPSBLZPACK	blzpack
Interface to PLANSO	EPSPLANSO	planso
Interface to TRLAN	EPSTRLAN	trlan
Interface to LOBPCG	EPSLOBPCG	lobpcg

Selecting the Portion of the Spectrum

EPSSetWhichEigenpairs(EPS eps, EPSWhich which)

Specifies which part of the spectrum is requested

which	Command line key	Sorting criterion
EPS_LARGEST_MAGNITUDE	-eps_largest_magnitude	Largest $\|\lambda\|$
EPS_SMALLEST_MAGNITUDE	-eps_smallest_magnitude	Smallest $\|\lambda\|$
EPS_LARGEST_REAL	-eps_largest_real	Largest $\mathrm{Re}(\lambda)$
EPS_SMALLEST_REAL	-eps_smallest_real	Smallest $\mathrm{Re}(\lambda)$
EPS_LARGEST_IMAGINARY	-eps_largest_imaginary	Largest $\mathrm{Im}(\lambda)$
EPS_SMALLEST_IMAGINARY	-eps_smallest_imaginary	Smallest $\mathrm{Im}(\lambda)$

- Eigenvalues are sought according to this criterion (not all possibilities available for all solvers)
- Computed eigenvalues are sorted according to this criterion

Run-Time Examples

```
% program -eps_monitor -eps_view

% program -eps_type lanczos -eps_nev 6 -eps_ncv 24

% program -eps_type lanczos -eps_smallest_real

% program -eps_type arnoldi -eps_tol 1e-8 -eps_max_it 2000

% program -eps_type subspace -log_summary

% program -eps_type lapack -eps_plot_eigs -draw_pause -1

% program -eps_type arpack
```

Some Utilities

`EPSSetInitialVector(EPS eps,Vec v0)`

Sets the initial vector used to build the projection subspace

- Should be rich in the directions of wanted eigenvectors
- If no initial vector is provided, a random vector is used

`EPSComputeRelativeError(EPS eps,int j,PetscReal *err)`

Returns the relative error associated to the j-th solution

$$\frac{\|Ax_j - \lambda_j Bx_j\|}{\|\lambda_j x_j\|}$$

If $\lambda_j \simeq 0$ then it is computed as $\|Ax_j\|/\|x_j\|$

Spectral Transformation in SLEPc

An ST object is always associated to any EPS object

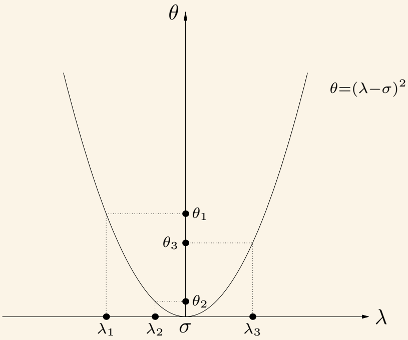
$$Ax = \lambda x \quad \Longrightarrow \quad Tx = \theta x$$

- The user need not manage the ST object directly
- Internally, the eigensolver works with the operator T
- At the end, eigenvalues are transformed back automatically

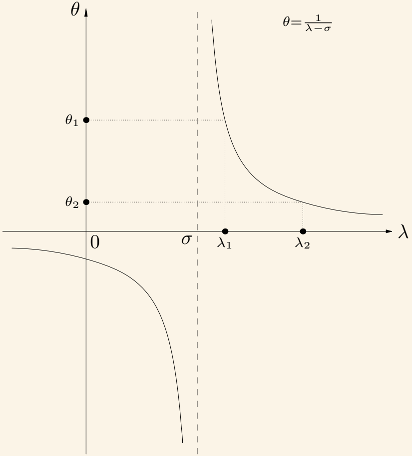
ST	Standard problem	Generalized problem
`shift`	$A + \sigma I$	$B^{-1}A + \sigma I$
`fold`	$(A + \sigma I)^2$	$(B^{-1}A + \sigma I)^2$
`sinvert`	$(A - \sigma I)^{-1}$	$(A - \sigma B)^{-1}B$
`cayley`	$(A - \sigma I)^{-1}(A + \tau I)$	$(A - \sigma B)^{-1}(A + \tau B)$

Illustration of Spectral Transformation

Spectrum folding



Shift-and-invert



Defining the Spectral Transform

STSetType(ST st,STType type)

For setting the type of spectral transformation

Spectral Transform	type	-st_type	Operator
Shift of origin	STSHIFT	shift	$B^{-1}A+\sigma I$
Spectrum folding	STFOLD	fold	$(B^{-1}A+\sigma I)^2$
Shift-and-invert	STSINV	sinvert	$(A-\sigma B)^{-1}B$
Cayley	STCAYLEY	cayley	$(A-\sigma B)^{-1}(A+\tau B)$

The default is shift of origin with a value of $\sigma = 0$

STSetShift(ST st,PetscScalar shift)

Used to provide the value of the shift σ (-st_shift)

There is an analogous function for setting the value of τ

Accessing the ST Object

The user does not create the ST object

`EPSGetST(EPS eps, ST *st)`

Gets the ST object associated to an EPS

Necessary for setting options in the source code

Linear Solves. All operators contain an inverse (except $B^{-1}A + \sigma I$ in the case of a standard problem)

- Linear solves are handled internally via a KSP object

`STGetKSP(ST st, KSP *ksp)`

Gets the KSP object associated to an ST

All KSP options are available, by prepending the `-st_` prefix

More Run-Time Examples

```
% program -eps_type power -st_type shift -st_shift 1.5

% program -eps_type power -st_type sinvert -st_shift 1.5

% program -eps_type power -st_type sinvert
          -eps_power_shift_type rayleigh

% program -eps_type arpack -eps_tol 1e-6
          -st_type sinvert -st_shift 1
          -st_ksp_type cgs -st_ksp_rtol 1e-8
          -st_pc_type sor  -st_pc_sor_omega 1.3
```

Coefficient Matrix of Linear Systems

STSetMatMode(ST st, STMatMode mode)

Allows to modify the way in which the matrix $A - \sigma B$ is created

mode	-st_matmode	Description
STMATMODE_COPY	copy	Creates a copy of A (default)
STMATMODE_INPLACE	inplace	Overwrites matrix A
STMATMODE_SHELL	shell	Uses a *shell* matrix

STSetMatStructure(ST st, MatStructure str)

To indicate whether matrices A and B have the same nonzero structure or not

Preserving the Symmetry

In the case of generalized eigenproblems in which both A and B are symmetric, symmetry is lost because none of $B^{-1}A + \sigma I$, $(A - \sigma B)^{-1}B$ or $(A - \sigma B)^{-1}(A + \tau B)$ is symmetric

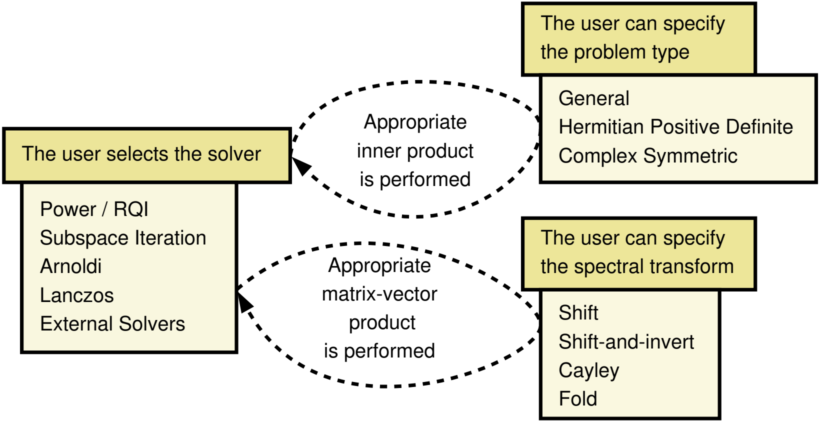
Choice of Inner Product

- Standard Hermitian inner product: $\langle x, y \rangle = x^* y$
- B-inner product: $\langle x, y \rangle_B = x^* B\, y$

Observations:

- $\langle x, y \rangle_B$ is a genuine inner product only if B is symmetric positive definite
- $\mathbb{R}^n$ with $\langle x, y \rangle_B$ is isomorphic to the Euclidean n-space $\mathbb{R}^n$ with the standard Hermitian inner product
- $B^{-1}A$ is auto-adjoint with respect to $\langle x, y \rangle_B$

SLEPc Abstraction



These operations are virtual functions: STInnerProduct and STApply

SLEPc

Deflation Subspaces

EPSAttachDeflationSpace(EPS eps,int n,Vec *ds,PetscTruth ortho)

Allows to provide a basis of a deflation subspace $\mathcal{S}$

The eigensolver operates in the restriction of the problem to the orthogonal complement of this subspace $\mathcal{S}$

Possible uses:

- When $\mathcal{S}$ is an invariant subspace, then the corresponding eigenpairs are not computed again
- If $\mathcal{S}$ is the null space of the operator, then zero eigenvalues are skipped
- In general, for constrained eigenvalue problems
- Also for singular pencils (A and B share a common null space)

Highlights

- Growing number of eigensolvers
- Seamlessly integrated spectral transformation
- Easy programming with PETSc's object-oriented style
- Data-structure neutral implementation
- Run-time flexibility, giving full control over the solution process
- Portability to a wide range of parallel platforms
- Usable from code written in C, C++ and Fortran
- Extensive documentation

Future Directions

Short Term

- Non-symmetric Lanczos
- Lanczos bi-diagonalization for SVD
- Enable computational intervals in some eigensolvers

Mid Term

- Implicitly Restarted Arnoldi method
- Davidson and Jacobi-Davidson methods
- Support for a series of closely related problems
- Block versions of some eigensolvers

Notice to Users

Help us improve SLEPc!

Want to hear about:

- New features you would like to see
- Bugs or portability problems
- Request for project collaboration

Contact us: `slepc-maint@grycap.upv.es`

Thanks!

SLEPc

http://www.grycap.upv.es/slepc

slepc-maint@grycap.upv.es