SLEPc: Scalable Library for Eigenvalue Problem Computations

Jose E. Roman

Joint work with A. Tomas and E. Romero

Universidad Politécnica de Valencia, Spain

10th ACTS Workshop - August, 2009

UNIVERSIDAD POLITECNICA DE VALENCIA

Outline

1. Introduction
2. Overview of SLEPc
3. Basic Usage
   - Eigenvalue Solvers
   - Spectral Transformation
   - SVD Solvers
4. Advanced Features

Introduction

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)

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

Spectral Transformation

A general technique that can be used in many methods

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

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)

SLEPc

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)

Shift of Origin	Shift-and-invert	Cayley
$T_S = A + \sigma I$	$T_{SI} = (A - \sigma I)^{-1}$	$T_C = (A - \sigma I)^{-1}(A + \tau I)$

Drawback: T not computed explicitly, linear solves instead

Singular Value Problems

Consider the SVD decomposition of a rectangular matrix $A \in \mathbb{R}^{m \times n}$

Singular Value Decomposition

$$A = U \Sigma V^T = \sum_{i=1}^{n} u_i \sigma_i v_i^T$$

where

- $\sigma_1, \sigma_2, \ldots, \sigma_n$: *singular values*
- $u_1, u_2, \ldots, u_n$: *left singular vectors*
- $v_1, v_2, \ldots, v_n$: *right singular vectors*

Solution of the Singular Value Problem

There are n singular values (counted with their multiplicities)

Partial solution: nsv solutions

$$\sigma_0, \sigma_1, \ldots, \sigma_{nsv-1} \in \mathbb{R}$$
$$u_0, u_1, \ldots, u_{nsv-1} \in \mathbb{R}^m$$
$$v_0, v_1, \ldots, v_{nsv-1} \in \mathbb{R}^n$$

nsv = number of singular values / vectors (singular triplets)

- Compute a few smallest or largest σ_i's

Solution of the Singular Value Problem

There are n singular values (counted with their multiplicities)

Partial solution: nsv solutions

$$\sigma_0, \sigma_1, \ldots, \sigma_{nsv-1} \in \mathbb{R}$$
$$u_0, u_1, \ldots, u_{nsv-1} \in \mathbb{R}^m$$
$$v_0, v_1, \ldots, v_{nsv-1} \in \mathbb{R}^n$$

nsv = number of singular values / vectors (singular triplets)

- Compute a few smallest or largest σ_i's

Alternatives:

- Solve eigenproblem $A^T A$
- Solve eigenproblem $H(A) = \begin{bmatrix} 0^{m\times m} & A \\ A^T & 0^{n\times n} \end{bmatrix}$
- Bidiagonalization

Overview of SLEPc

Design Considerations

- Various problem characteristics: Problems can be real/complex, Hermitian/non-Hermitian
- Many ways of specifying which solutions must be sought
- Many formulations: not all eigenproblems are formulated as simply $Ax = \lambda x$ or $Ax = \lambda Bx$

Design Considerations

- Various problem characteristics: Problems can be real/complex, Hermitian/non-Hermitian
- Many ways of specifying which solutions must be sought
- Many formulations: not all eigenproblems are formulated as simply $Ax = \lambda x$ or $Ax = \lambda Bx$

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

Design Considerations

- Various problem characteristics: Problems can be real/complex, Hermitian/non-Hermitian
- Many ways of specifying which solutions must be sought
- Many formulations: not all eigenproblems are formulated as simply $Ax = \lambda x$ or $Ax = \lambda Bx$

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

- Internally, solvers can be quite complex (deflation, restart, ...)
- Spectral transformations can be used irrespective of the solver
- Repeated linear solves may be required
- SVD can be solved via associated eigenproblem or bidiagonalization

Design Considerations

- Various problem characteristics: Problems can be real/complex, Hermitian/non-Hermitian
- Many ways of specifying which solutions must be sought
- Many formulations: not all eigenproblems are formulated as simply $Ax = \lambda x$ or $Ax = \lambda Bx$

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

- Internally, solvers can be quite complex (deflation, restart, ...)
- Spectral transformations can be used irrespective of the solver
- Repeated linear solves may be required
- SVD can be solved via associated eigenproblem or bidiagonalization

Goal: hide eigensolver complexity and separate spectral transform

What Users Need

- Abstraction of mathematical objects: vectors and matrices
- Efficient linear solvers (direct or iterative)
- Easy programming interface
- Run-time flexibility, full control over the solution process
- Parallel computing, mostly transparent to the user

- State-of-the-art eigensolvers
- Spectral transformations
- SVD solvers

What Users Need

Provided by PETSc

- Abstraction of mathematical objects: vectors and matrices
- Efficient linear solvers (direct or iterative)
- Easy programming interface
- Run-time flexibility, full control over the solution process
- Parallel computing, mostly transparent to the user

Provided by SLEPc

- State-of-the-art eigensolvers
- Spectral transformations
- SVD solvers

Summary

PETSc: Portable, Extensible Toolkit for Scientific Computation

Software for the scalable (parallel) solution of algebraic systems arising from partial differential equation (PDE) simulations

- Developed at Argonne National Lab since 1991
- Usable from C, C++, Fortran77/90
- Focus on abstraction, portability, interoperability
- Extensive documentation and examples
- Freely available and supported through email

`http://www.mcs.anl.gov/petsc`

Current version: 3.0.0 (released Dec 2008)

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

Also support for the partial SVD decomposition

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

Current version: 3.0.0 (released Feb 2009)

Structure of SLEPc (1)

SLEPc extends PETSc with three new objects: EPS, ST, SVD

EPS: Eigenvalue Problem Solver

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

Structure of SLEPc (2)

ST: Spectral Transformation

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

Structure of SLEPc (2)

ST: Spectral Transformation

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

SVD: Singular Value Decomposition

- The user specifies the SVD problem via this object
- Transparently provides the associated eigenproblems or a specialized solver

PETSc/SLEPc Numerical Components

PETSc

Nonlinear Systems		
Line Search	Trust Region	Other

Time Steppers			
Euler	Backward Euler	Pseudo Time Step	Other

Krylov Subspace Methods							
GMRES	CG	CGS	Bi-CGStab	TFQMR	Richardson	Chebychev	Other

Preconditioners						
Additive Schwarz	Block Jacobi	Jacobi	ILU	ICC	LU	Other

Matrices				
Compressed Sparse Row	Block Compressed Sparse Row	Block Diagonal	Dense	Other

Vectors

Index Sets			
Indices	Block Indices	Stride	Other

SLEPc

PETSc/SLEPc Numerical Components

PETSc

Nonlinear Systems

Line Search	Trust Region	Other

Time Steppers

Euler	Backward Euler	Pseudo Time Step	Other

Krylov Subspace Methods

GMRES	CG	CGS	Bi-CGStab	TFQMR	Richardson	Chebychev	Other

Preconditioners

Additive Schwarz	Block Jacobi	Jacobi	ILU	ICC	LU	Other

Matrices

Compressed Sparse Row	Block Compressed Sparse Row	Block Diagonal	Dense	Other

Vectors

Index Sets

Indices	Block Indices	Stride	Other

SLEPc

SVD Solvers

Cross Product	Cyclic Matrix	Lanczos	Thick Res. Lanczos

Eigensolvers

Krylov-Schur	Arnoldi	Lanczos	Other

Spectral Transform

Shift	Shift-and-invert	Cayley	Fold

Basic Usage

EPS: 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

EPS: Simple Example

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

EPS: 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);
```

EPS: 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);
```

EPS: 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);
}
```

EPS: 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: Solving the Problem

`EPSSolve(EPS eps)`

Launches the eigensolver

Currently available eigensolvers:

- Power Iteration and RQI
- Subspace Iteration with Rayleigh-Ritz projection and locking
- Arnoldi method with explicit restart and deflation
- Lanczos method with explicit restart and deflation
  - Reorthogonalization: Local, Partial, Periodic, Selective, Full
- Krylov-Schur (default)

Also interfaces to external software: ARPACK, PRIMME, ...

Details: Problem Definition

EPSSetOperators(EPS eps, Mat A, Mat B)

Used for passing 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

Details: Problem Definition

`EPSSetOperators(EPS eps, Mat A, Mat B)`

Used for passing 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)`

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
```

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: krylovschur
  selected portion of 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-07
  dimension of user-provided deflation space: 0
  IP Object:
    orthogonalization method:  classical Gram-Schmidt
    orthogonalization refinement:  if needed (eta: 0.707100)
  ST Object:
    type: shift
    shift: 0
```

EPS: Run-Time Examples

```
% program -eps_view -eps_monitor

% program -eps_type krylovschur -eps_nev 6 -eps_ncv 24

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

% program -eps_type subspace -eps_hermitian -log_summary

% program -eps_type lapack

% program -eps_type arpack -eps_plot_eigs -draw_pause -1

% program -eps_type primme -eps_smallest_real
```

Built-in Support Tools

- Plotting computed eigenvalues

  % program -eps_plot_eigs

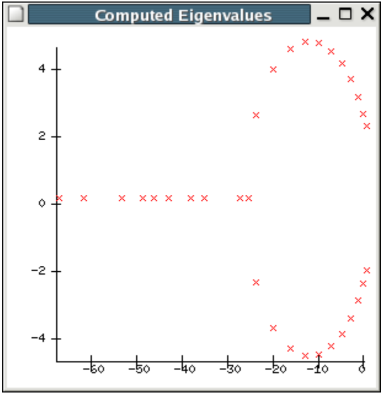
- Printing profiling information

  % program -log_summary

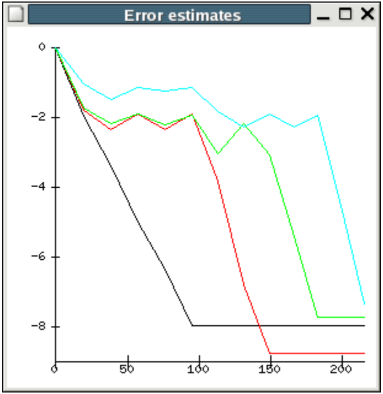
- Debugging

  % program -start_in_debugger

  % program -malloc_dump



Built-in Support Tools



- Monitoring convergence (textually)

  `% program -eps_monitor`

- Monitoring convergence (graphically)

```
% program -draw_pause 1
        -eps_monitor_draw
```

Spectral Transformation in SLEPc

An ST object is always associated to any EPS object

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

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

Spectral Transformation in SLEPc

An ST object is always associated to any EPS object

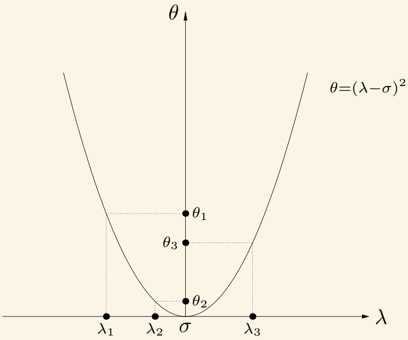
$$Ax = \lambda x \quad \Longrightarrow \quad Tx = \theta x$$

- 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

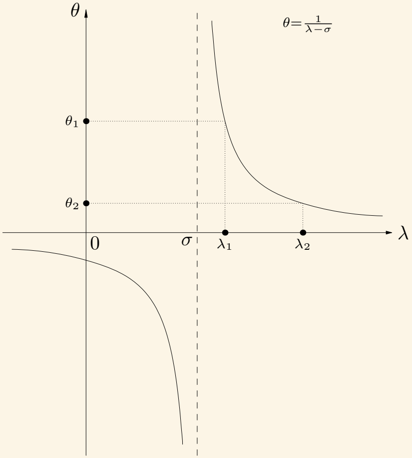
ST	Standard problem	Generalized problem
`shift`	$A + \sigma I$	$B^{-1}A + \sigma I$
`fold`	$(A + \sigma I)^2$	$(B^{-1}A + \sigma I)^2$
`sinvert`	$(A - \sigma I)^{-1}$	$(A - \sigma B)^{-1}B$
`cayley`	$(A - \sigma I)^{-1}(A + \tau I)$	$(A - \sigma B)^{-1}(A + \tau B)$

Illustration of Spectral Transformation

Spectrum folding



Shift-and-invert



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

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. Most operators contain an inverse

- 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

ST: 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
```

SVD: Basic Usage

Usual steps for solving an SVD problem with SLEPc:

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

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

SVD: Simple Example

```
SVD         svd;        /*  SVD solver context   */
Mat         A;          /*  matrix for A=USV^T   */
Vec         u,v;        /*  singular vectors     */
PetscReal   s;          /*  singular value       */
```

SVD: Simple Example

```
SVD          svd;        /*  SVD solver context    */
Mat          A;          /*  matrix for A=USV^T    */
Vec          u,v;        /*  singular vectors      */
PetscReal    s;          /*  singular value        */

SVDCreate(PETSC_COMM_WORLD, &svd);
SVDSetOperator(svd, A);
SVDSetFromOptions(svd);
```

SVD: Simple Example

```
SVD          svd;        /*  SVD solver context    */
Mat          A;          /*  matrix for A=USV^T    */
Vec          u,v;        /*  singular vectors      */
PetscReal    s;          /*  singular value        */

SVDCreate(PETSC_COMM_WORLD, &svd);
SVDSetOperator(svd, A);
SVDSetFromOptions(svd);

SVDSolve(svd);
```

SVD: Simple Example

```
SVD          svd;        /*  SVD solver context    */
Mat          A;          /*  matrix for A=USV^T    */
Vec          u,v;        /*  singular vectors      */
PetscReal    s;          /*  singular value        */

SVDCreate(PETSC_COMM_WORLD, &svd);
SVDSetOperator(svd, A);
SVDSetFromOptions(svd);

SVDSolve(svd);

SVDGetConverged(svd, &nconv);
for (i=0; i<nconv; i++) {
  SVDGetSingularTriplet(svd, i, &s, u, v);
}
```

SVD: Simple Example

```
SVD          svd;        /*  SVD solver context    */
Mat          A;          /*  matrix for A=USV^T    */
Vec          u,v;        /*  singular vectors      */
PetscReal    s;          /*  singular value        */

SVDCreate(PETSC_COMM_WORLD, &svd);
SVDSetOperator(svd, A);
SVDSetFromOptions(svd);

SVDSolve(svd);

SVDGetConverged(svd, &nconv);
for (i=0; i<nconv; i++) {
  SVDGetSingularTriplet(svd, i, &s, u, v);
}

SVDDestroy(svd);
```

Details: Solving the Problem

`SVDSolve(SVD svd)`

Launches the SVD solver

Currently available SVD solvers:

- Cross-product matrix with any EPS eigensolver
- Cyclic matrix with any EPS eigensolver
- Golub-Kahan-Lanczos bidiagonalization with explicit restart and deflation
- Golub-Kahan-Lanczos bidiagonalization with thick restart and deflation

Details: Problem Definition and Specification of Options

`SVDSetOperators(SVD svd, Mat A)`

Used for passing the matrix that constitutes the problem

Details: Problem Definition and Specification of Options

`SVDSetOperators(SVD svd, Mat A)`

Used for passing the matrix that constitutes the problem

`SVDSetFromOptions(SVD svd)`

Looks in the command line for options related to SVD

For example, the following command line

```
% program -svd_tol 1e-8 -svd_max_it 100
```

is equivalent to a call `SVDSetTolerances(eps,1e-8,100)`

Details: Problem Definition and Specification of Options

`SVDSetOperators(SVD svd, Mat A)`

Used for passing the matrix that constitutes the problem

`SVDSetFromOptions(SVD svd)`

Looks in the command line for options related to SVD

For example, the following command line

```
% program -svd_tol 1e-8 -svd_max_it 100
```

is equivalent to a call `SVDSetTolerances(eps,1e-8,100)`

`SVDView(SVD svd, PetscViewer viewer)`

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

Details: Viewing Current Options

Sample output of -svd_view

```
SVD Object:
  method: trlanczos
  transpose mode: explicit
  selected portion of the spectrum: largest
  number of singular values (nsv): 1
  number of column vectors (ncv): 10
  maximum dimension of projected problem (mpd): 10
  maximum number of iterations: 100
  tolerance: 1e-07
  Lanczos reorthogonalization: two-side
  IP Object:
    orthogonalization method:  classical Gram-Schmidt
    orthogonalization refinement:  if needed (eta: 0.707100)
```

SVD: Run-Time Examples

```
% program -svd_view -svd_monitor

% program -svd_type lanczos -svd_nsv 6 -svd_ncv 24

% program -svd_type trlanczos -svd_tol 1e-8 -svd_max_it 2000

% program -svd_type cross -svd_eps_type krylovschur

% program -svd_type lapack

% program -svd_type lanczos -svd_monitor_draw

% program -svd_type trlanczos -svd_smallest
```

Advanced Features

Options for Subspace Generation

Initial Subspace

- Provide an initial trial subspace, e.g. from a previous computation
- Current support only for a single vector (EPSSetInitialVector)

Deflation Subspace

- Provide a deflation space with EPSAttachDeflationSpace
- The eigensolver operates in the restriction to the orthogonal complement
- Useful for constrained eigenproblems or problems with a known nullspace

Subspace Extraction

In some cases, convergence of the eigensolver may be very slow

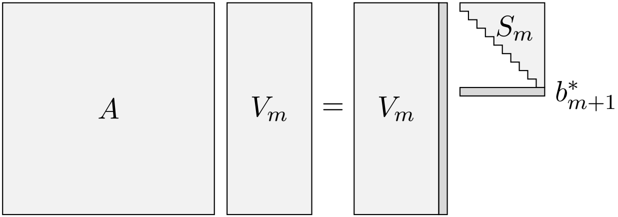
$\longrightarrow$ Enhanced subspace extraction: try to extract better approximations from the available subspace

- Harmonic extraction
  - Compute harmonic Ritz values instead of Ritz values
  - Useful for computing interior eigenvalues (alternative to the spectral transformation)
  - Currently implemented in Krylov-Schur solver
- Other: refined extraction

Computation of Many Eigenpairs

By default, a subspace of dimension $2 \cdot nev$ is used...
For large nev, this is not appropriate

- Excessive storage and inefficient computation



Strategy: compute eigenvalues in chunks - restrict the dimension of the projected problem

```
% program -eps_nev 2000 -eps_mpd 300
```

SLEPc Highlights

- Growing number of eigensolvers
- Seamlessly integrated spectral transformation
- Support for SVD
- 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

Under Development

- Generalized Davidson and Jacobi-Davidson solvers
- Enable computational intervals for symmetric problems

Mid Term

- Conjugate Gradient-type eigensolvers
- Non-symmetric Lanczos eigensolver
- Support for other types of eigenproblems: quadratic, structured, non-linear

More Information

Homepage:

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

Hands-on Exercises:

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

Contact email:

slepc-maint@grycap.upv.es