SLEPc:
Scalable Library for Eigenvalue Problem Computations

The DOE Advanced CompuTational Software Collection (ACTS)

Thirteenth
DOE ACTS Collection Workshop
Berkeley, California, August 14-17, 2012

Tony Drummond
Computational Research Division
Lawrence Berkeley National Laboratory

This presentation was prepared from slides from Jose E. Roman and SLEPc Team (UPV)

BERKELEY LAB
Lawrence Berkeley National Laboratory

ACTS COLLECTION

UNIVERSIDAD POLITECNICA DE VALENCIA

U.S. DEPARTMENT OF ENERGY | Office of Science

OUTLINE

- What is **SLEPc**?
- Computational Problems target by SLEPc
- **SLEPc**: Eigenvalue Solvers
- **SLEPc**: Spectral Transformation
- **SLEPc**: SVD Solvers
- **SLEPc**: Quadratic Eigenvalue Solvers
- Additional Features of **SLEPc**
- short DEMO

Scalable Library for Eigenvalue Problem computation

Developed at Polytechnic University of Valencia, Spain

- Lead Developer: Prof. Jose Roman
  - others: Andres Thomas, Eloy Romero and Carmen Campos
- Contact SLEPc Team:

  slepc-maint@grycap.upv.es

Scalable Library for Eigenvalue Problem computation

SLEPc

- home page

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

- Additional Material:

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

  > module load slepc/3.1_g (there are more choices)

  > cp -r $SLEPC_DIR/src/eps/examples/ .

UNIVERSIDAD POLITECNICA DE VALENCIA

Functionality in The DOE ACTS Collection

Computational Problem	Methodology	Algorithm	Library
Linear Least Squares Problems	Least Squares	$min_x \|\| b - Ax \|\|_2$	ScaLAPACK
	Minimum Norm Solution	$min_x \|\| x \|\|_2$	ScaLAPACK
	Minimum Norm Least Squares	$min_x \|\| b - Ax \|\|_2$ $min_x \|\| x \|\|_2$	ScaLAPACK
Standard Eigenvalue Problem	Symmetric Eigenvalue Problem	$Az = \lambda z$ *For* $A=A^H$ or $A=A^T$	ScaLAPACK (dense) SLEPc (sparse)
Singular Value Problem	Singular Value Decomposition	$A = U\Sigma V^T$ $A = U\Sigma V^H$	ScaLAPACK (dense) SLEPc (sparse)
Generalized Symmetric Definite Eigenproblem	Eigenproblem	$Az = \lambda Bz$ $ABz = \lambda z$ $BAz = \lambda z$	ScaLAPACK (dense) SLEPc (sparse)

Computational Problems

Computational Problem	Methodology	Algorithms	Library
• Eigenvalue Solvers • Spectral Transformations • SVD Solvers • Quadratic Eigenvalue Solvers	Availa in ACT	in ACTS	es that

[background]:Solving Eigenvalue Problems

Computational Problem

- Eigenvalue Solvers
- Spectral Transformations
- SVD Solvers
- Quadratic Eigenvalue Solvers

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 (un)symmetric (Hermitian)
- Typically B is symmetric positive (semi-) definite

[background]: Solving Eigenvalue Problems

Computational Problem

- Eigenvalue Solvers
- Spectral Transformations
- SVD Solvers
- Quadratic Eigenvalue Solvers

Solutions

$$\lambda_0, \lambda_1, \ldots \lambda_{nev-1} \in \mathbb{C}$$

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

Where,

- there are nev eigenvalues (counted with their multiplicities)

Computational requirements:

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

[background]: Spectral Transformation

Computational Problem

- Eigenvalue Solvers
- Spectral Transformations
- SVD Solvers
- Quadratic Eigenvalue Solvers

A general techniques that can be used in many methods to improve convergence (better separation)

Original System ⇒ **Transformed System**

$$Ax = \lambda x \qquad \Rightarrow \qquad Tx = \theta x$$

In the transformed systems;

- λ_i's are modified by simple relation
- x_i's are not altered

Shift of Origin

$$T_S = A + \sigma I$$

Shift-and-Invert

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

Cayley

$$T_C = (A - \sigma I)^{-1} (A + \tau I)$$

* Drawback: T not computed explicitly, linear solves

[background] Singular Value Decomposition (SVD) Problems

Computational Problem	
• Eigenvalue Solvers • Spectral Transformations • **SVD Solvers** • Quadratic Eigenvalue Solvers	Compute the SVD of a rectangular matrix $A \in \mathbb{R}^{mxn}$

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

where

- Singular Values: $\sigma_1, \sigma_2, \ldots \sigma_n$
- Left singular vectors: $u_1, u_2, \ldots u_m$
- Right singular vectors : $v_1, v_2, \ldots v_n$

[background] Solving a SVD Problems

Computational Problem

- Eigenvalue Solvers
- Spectral Transformations
- **SVD Solvers**
- Quadratic Eigenvalue Solvers

Partial solution: nsv solutions:

- Singular values: $\sigma_0, \sigma_1, \ldots \sigma_{nsv-1} \in \mathbb{R}$
- Left singular vectors: $u_0, u_1, \ldots u_{nsv-1} \in \mathbb{R}^m$
- Right singular vectors : $v_0, v_1, \ldots v_{nsv-1} \in \mathbb{R}^n$

There are nsv singular values (counted with their multiplicities)

Computational requirements:

- Compute a few smallest or largest σ_i's
- Solve the eigenproblem A^TA
- Solve the eigenproblem $H(A) = \begin{bmatrix} 0^{mxm} & A \\ A^T & 0^{nxn} \end{bmatrix}$
- Bidiagonalization

[background] Quadratic Eigenvalue Problems

Computational Problem
• Eigenvalue Solvers
• Spectral Transformations
• SVD Solvers
• **Quadratic Eigenvalue Solvers**

Quadratic Eigenvalue Problem

$$(\lambda^2 M + \lambda C + K)x = 0$$

Where,

- λ is a (complex) scalar, eigenvalue
- x is a (complex) vector: eigenvector
- Matrices M, C and K can be real or complex
- Matrices M, C and K can be (un)symmetric (Hermitian)
- Typically some matrices are also symmetric positive (semi-) definite

[background] Solving Quadratic Eigenvalue Problems

Computational Problem

- Eigenvalue Solvers
- Spectral Transformations
- SVD Solvers
- **Quadratic Eigenvalue Solvers**

$$\lambda_0, \lambda_1, \ldots \lambda_{nev-1} \in \mathbb{C}$$

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

Where,

- there are 2 X nev eigenvalues

Alternatives:

- Linearization $Az = \lambda Bz$

$$z = \begin{bmatrix} x \\ \lambda x \end{bmatrix} \qquad A = \begin{bmatrix} 0 & I \\ -K & -C \end{bmatrix} \qquad B = \begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix}$$

- Specific method (Q-Arnoldi)

SLEPc Design Considerations

- Various problem characteristics:
  - real/complex
  - Hermitian/non-hermitian
- Multiple ways to specify the solutions that are sought
- Many formulations (beyond $Ax = \lambda x$ or $Ax = \lambda Bx$)

Characteristics of the SLEPc Library

- Uniform abstract User Interfaces to address all the aforementioned problems
  - Through a simple and intuitive interphase, SLEPc provides internally solver implementations with a high-level of algorithmic complexity (deflation, restart, etc. .)
- Spectral transformations can be used irrespectively of the solver
- Recurrent linear solves may be necessary
- SVD and QEP can be solved via associated eigenproblem or specific methods (bidiagonalization/Q-Arnoldi)

Characteristics of the SLEPc Library

- General Purpose library for the solution of large-scale sparse eigenproblems on parallel computers
  - For standard, generalized and quadratic eigenproblems
  - For real and complex arithmetic
  - For Hermitian or non-Hermitian problems
  - For the partial SVD decomposition

  - Relies on PETSc Functionality
  - Current version 3.3 (released on August 2012). The major changes in this version are:
    - New EPS solvers: RQCG, GD2 and indefinite Krylov-Schur.
    - A major reorganization of code (now everything related to projected eigenproblems is encapsulated in a new auxiliary object DS).

Four Abstract Objects SLEPc

- Extends PETSc functionality with four objects
  - **EPS**: Eigenvalue Problem Solver

EPS is the abstract User Interface to:

- Describe an eigenvalue problem
- Access a collection of sparse eigensolver implementations and algorithmic parameters (e.g., eigenvalues of interest)

Four Abstract Objects SLEPc

- Extends PETSc functionality with four objects
  - **EPS**: Eigenvalue Problem Solver
  - **ST:** Spectral Transformation

- ST is abstract interface to transform the original system into $Tx = \theta x$
- ST is always associated to an EPS object and cannot be directly accessed

Four Abstract Objects SLEPc

- Extends PETSc functionality with four objects
  - **EPS**: Eigenvalue Problem Solver
  - **ST:** Spectral Transformation
  - **SVD:** Singular Value Decomposition

SVD is the abstract User Interface to:

- Describe a SVD problem
- Provides, transparently, access to eigensolvers for the associated eigenproblems or the specialized solver based on bidiagonalization

Four Abstract Objects SLEPc

- Extends PETSc functionality with four objects
  - **EPS**: Eigenvalue Problem Solver
  - **ST:** Spectral Transformation
  - **SVD:** Singular Value Decomposition
  - **QEP:** Quadratic Eigenvalue Problem

QEP is the abstract User Interface to:

- Describe a Quadratic Eigenproblem
- Provides, transparently, the linearlization to a generalized eigenproblem or the specialized solver (Q-Arnoldi)

Characteristics of the SLEPc Library

PETSc

SNES — Nonlinear Systems

Line Search	Trust Region	Other

TS — Time Steppers

Euler	Backward Euler	Time Stepping	Other

KSP — Krylov Subspace Methods

GMRES	CG	CGS	Bi-CGStab	TFQMR	Richardson	Chebychev	Other

PC — Preconditioners

Additive Schwarz	Block Jacobi	Jacobi	ILU	ICC	LU	Other

Mat — Matrices

Compressed Sparse Row	Block Compressed Sparse Row	Block Diagonal	Dense	Other

Vec — Vectors

IS — Index Sets

Indices	Block Indices	Stride	Other

SLEPc

SVD — SVD Solvers

Cross Product	Cyclic Matrix	Lanczos	Thick R. Lanczos

QEP — Quadratic

Linear-ization	Q-Arnoldi

EPS — Eigensolvers

Krylov-Schur	Arnoldi	Lanczos	GD	JD	Other

ST — Spectral Transformation

Shift	Shift-and-invert	Cayley	Fold	Preconditioner

Solving an Eigenvalue Problem with SLEPc

- Usual steps:
  - Declare a SLEPc EPS object and create the EPS object
  - Define the eigenvalue problem
  - Optionally specify algorithmic parameters for the solution
  - Invoke the eigensolver
  - Retrieve the computed solution
  - Don't forget to **Destroy** the EPS object

Solving an Eigenvalue Problem with SLEPc

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);
```

Functionality available in the EPSSolve

Currently available eigensolvers:

- Power Iteration and Rayleigh-Quatient Iteration (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**)
- Preconditioned solvers: Generalized Davison and Jacobi-Davidson (non-hermitian)
- ***new:*** Rayleigh-Quatient CG (RQCG)
- ***new:*** GD2
- ***new:*** Indefinite Krylov-Schur

Defining the Eigenproblem in SLEPc

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

Standard Eigenproblem

$$Ax = \lambda x$$

Specified through `Mat A`, while `Mat B` is set to `PETSC_NULL`

Generalized Eigenproblem

$$Ax = \lambda Bx$$

Specified through `Mat A` and `Mat B`

```
EPSSetProblemType( EPS eps, EPSProblemType type)
```

Problem Type	EPSProblemType	Command line option
Hermitian	EPS_HEP	-eps_hermitian
Generalized Hermitian	EPS_GHEP	-eps_gen_hermitian
non-Hermitian	EPS_NHEP	-eps_non_hermitian
Generalized non-Hermitian	EPS_GNHEP	-eps_gen_non_hermitian
GNHEP with B > 0	EPS_PGNHEP	-eps_pos_gen_non_hermitian

Defining the Eigenproblem in SLEPc

`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`)

Profiling in SLEPc (EPS)

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
```

Built-In Profiling/Debugging Support SLEPc

- Plotting computed eigenvalues

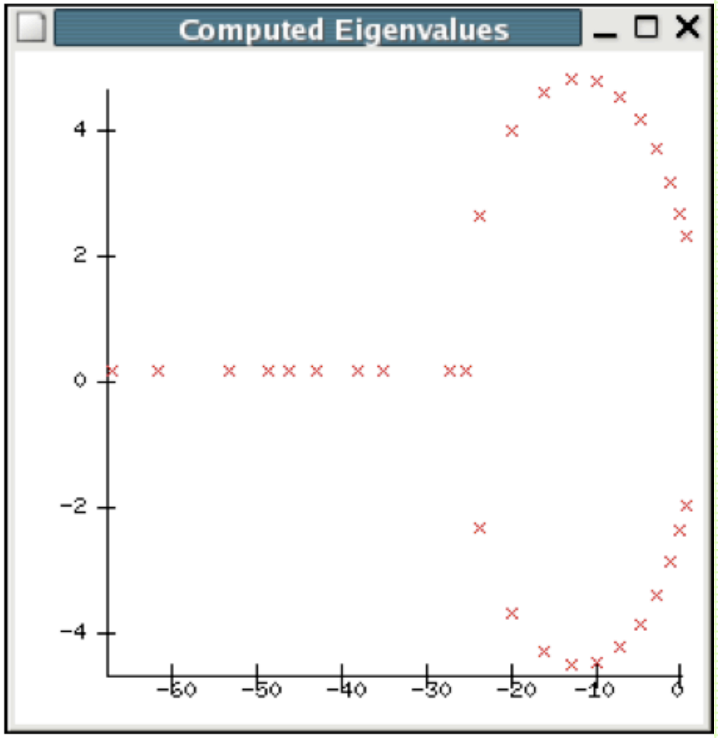
```
% program -eps_plot_eigs
```

- Printing profiling information

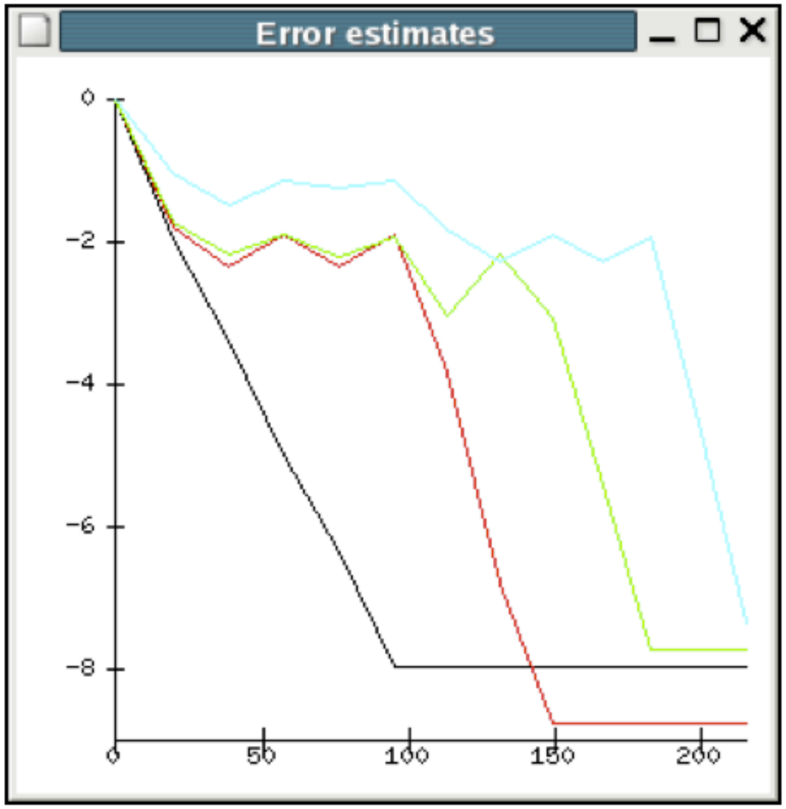
```
% program -log_summary
```

- Debugging

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



Built-In Profiling/Debugging Support SLEPc



- Monitoring convergence (textually)

  ```
  % program -eps_monitor
  ```

- Monitoring convergence (graphically)

  ```
  % program -draw_pause 1
          -eps_monitor_draw_all
  ```

Spectral Transformation in SLEPc

Original System	$\Rightarrow$	Transformed System
$Ax = \lambda x$		$Tx = \theta x$

- A **ST** object is always associated to a **EPS** object
- Internally, the eigensolver works with the operator T
- At the end, eigenvalues are transformed back automatically

ST Type	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	$(A - \sigma I)^{-1}(A + \tau I)$	$(A - \sigma B)^{-1}(A + \tau B)$
precond	$K^{-1} \approx (A - \sigma I)^{-1}$	$K^{-1} \approx (A - \sigma B)^{-1}$

Accessing SLEPc's ST Object

```
EPSGetST( EPS eps, ST *st )
```

- **ST** objects are not created by the user instead it is obtained
- Users only need *st to set options inside the code
- Linear solve are handled internally through PETSc's **KSP**

```
STGetKSP( ST st, KSP *ksp )
```

Gets the KSP object associated to an ST

All KSP options are available to the user, in the command line by prepending the **-st_** prefix

ST Run-Time Examples

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

% program -eps_type power -st_type sinvert -eps_target 1.5

% program -eps_type power -st_type sinvert
          -eps_power_shift_type rayleigh

% program -eps_type krylovschur -eps_tol 1e-6
          -st_type sinvert -eps_target 1
          -st_ksp_type cgs -st_ksp_rtol 1e-8
          -st_pc_type sor  -st_pc_sor_omega 1.3

% program -eps_type jd -eps_target 2
```

Solving a SVD Problem with SLEPc

- Usual steps:
  - Declare a SLEPc SVD object and create the SVD object
  - Define the problem
  - Optionally specify algorithmic parameters for the solution
  - Invoke the solver
  - Retrieve the computed solution
  - Don’t forget to **Destroy** the SVD object

Example of Solving a SVD Problem with SLEPc

```
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);
```

Functionality available in the SVDSolve

Currently available SVD solver:

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

Additional Parameters for the SVD in SLEPc

```
SVDSetOperators( SVD svd, Mat A)
```

Specified through `Mat A` as the operator

```
SVDSetFromOptions( SVD svd)
```

Overwrite options from command-line arguments

```
SVDView( SVD svd, PetscViewer viewer)
```

Equivalent to -svd_view

Solving a QEP with SLEPc

- Usual steps:
  - Declare a SLEPc QEP object and create the QEP object
  - Define the eigenvalue problem
  - Optionally specify algorithmic parameters for the solution
  - Invoke the solver
  - Retrieve the computed solution
  - Don't forget to **Destroy** the QPD object

Example of Solving a QEP with SLEPc

```
QEP         qep;        /*  eigensolver context   */
Mat         M, C, K;    /*  matrices of the QEP   */
Vec         xr, xi;     /*  eigenvector, x        */
PetscScalar kr, ki;     /*  eigenvalue, k         */

QEPCreate(PETSC_COMM_WORLD, &qep);
QEPSetOperators(qep, M, C, K);
QEPSetProblemType(qep, QEP_GENERAL);
QEPSetFromOptions(qep);

QEPSolve(qep);

QEPGetConverged(qep, &nconv );
for (i=0; i<nconv; i++) {
  QEPGetEigenpair(qep, i, &kr, &ki, xr, xi );
}

QEPDestroy(qep);
```

Functionality available in the QEPSolve

Currently available eigensolvers:

- Linearization with any EPS solver
  - Non-symmetric $\begin{bmatrix} 0 & I \\ -K & -C \end{bmatrix} - \lambda \begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix}$
  - Symmetric $\begin{bmatrix} 0 & -K \\ -K & -C \end{bmatrix} - \lambda \begin{bmatrix} -K & 0 \\ 0 & M \end{bmatrix}$
  - Hamiltonian $\begin{bmatrix} K & 0 \\ C & K \end{bmatrix} - \lambda \begin{bmatrix} 0 & K \\ -M & 0 \end{bmatrix}$
- Q-Arnoldi

Defining the QEP in SLEPc

```
QEPSetOperators( QEP qep, Mat M, Mat C, Mat K)
```

Define the QEP through matrices M,C, and K

```
QEPSetProblemType( QEP qep, QEPProblemType type)
```

Problem Type	EPSProblemType	Command line option
General	QEP_GENERAL	-qep_general
hermitian	QEP_HERMITIAN	-qep_hermitian
Gyroscopic	QEP_GYROSCOPIC	-qep_gyroscopic

Additional Parameters for the QEP in SLEPc

`QEPSetFromOptions(QEP qep)`

Overwrite options from command-line arguments

`QEPView(QEP qep, PetscViewer viewer)`

Equivalent to -qep_view

`QEPLinearSetCompanionForm(QEP qep, PetscInt cform)`

Selects among the different available expressions for linearization

Options for Subspace Generation in SLEPc

Initial Subspace

- Provide an initial trial subspace with **`EPSSetInitialSpace`** (e.g., from previous computations)
- Krylov solvers only support a single vector

Deflation Subspace

- Provide an initial trial subspace with **`EPSSetDeflationSpace`**
- The eigensolver operates in the restriction to the orthogonal compliment
- Useful for constraint eigenproblems or problems with a known nullspace

SLEPc Highlights

- Growing number of eigensolvers
- Seamlessly integrated spectral transformation
- Support for SVD and QEP
- PETSc style user interfaces and extensibility
- Supported run-time options to drive the solver and parameter selection
- Portability to a wide range of platforms
- Supports C, C++ and different flavors of fortran
- Extensive documentation
- ***Got PETSc?*** then, very easy to install

Acknowledgements

Special thanks to Jose E. Roman from the Polytechnic University of Valencia for SLEPc and the materials used in producing this presentation