SLEPc: Scalable Library for Eigenvalue Problem Computations

Tony Drummond
Computational Research Division
Lawrence Berkeley National Laboratory

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

- 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/ .
## Functionality in The DOE ACTS Collection

<table>
<thead>
<tr>
<th>Computational Problem</th>
<th>Methodology</th>
<th>Algorithm</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Least Squares Problems</td>
<td>Least Squares</td>
<td>$\min_x | b - Ax |_2$</td>
<td>ScaLAPACK</td>
</tr>
<tr>
<td>Minimum Norm Solution</td>
<td></td>
<td>$\min_x | x |_2$</td>
<td>ScaLAPACK</td>
</tr>
<tr>
<td>Minimum Norm Least Squares</td>
<td></td>
<td>$\min_x | b - Ax |_2$</td>
<td>ScaLAPACK</td>
</tr>
<tr>
<td>Standard Eigenvalue Problem</td>
<td>Symmetric Eigenvalue Problem</td>
<td>$A z = \lambda z$</td>
<td>ScaLAPACK (dense)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A = U \Sigma V^T$</td>
<td>SLEPc (sparse)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A = U \Sigma V'^*$</td>
<td></td>
</tr>
<tr>
<td>Singular Value Problem</td>
<td>Singular Value Decomposition</td>
<td>$A = U \Sigma V^T$</td>
<td>ScaLAPACK (dense)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A = U \Sigma V'^*$</td>
<td>SLEPc (sparse)</td>
</tr>
<tr>
<td>Generalized Symmetric Definite Eigenproblem</td>
<td>Eigenproblem</td>
<td>$A z = \lambda B z$</td>
<td>ScaLAPACK (dense)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A B z = \lambda z$</td>
<td>SLEPc (sparse)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$B A z = \lambda z$</td>
<td></td>
</tr>
</tbody>
</table>

**SLEPc**

Thirteenth DOE ACTS Collection Workshop
Berkeley, California, August 14-17, 2012
<table>
<thead>
<tr>
<th>Computational Problem</th>
<th>Methodology</th>
<th>Algorithms</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eigenvalue Solvers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral Transformations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SVD Solvers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quadratic Eigenvalue Solvers</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Available methods and their HPC implementations in ACTS

Specific ACTS Libraries or Tools that provide that functionality

SLEPc
[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,
- \( \lambda \) 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

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

Where,

✦ there are \( \text{nev} \) eigenvalues (counted with their multiplicities)

Computational requirements:

✦ Compute a few dominant eigenvalues
✦ Compute a few \( \lambda_i \)'s with smallest or largest real parts
✦ Compute all \( \lambda_i \)'s in a given region of the complex plane
### 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)

<table>
<thead>
<tr>
<th>Original System</th>
<th>Transformed System</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Ax = \lambda x$</td>
<td>$Tx = \theta x$</td>
</tr>
</tbody>
</table>

In the transformed systems:
- $\lambda_i$’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**

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

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

where

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

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

Partial solution: \( n_{SV} \) solutions:
- Singular values: \( \sigma_0, \sigma_1, \ldots, \sigma_{n_{SV}-1} \in \mathbb{R} \)
- Left singular vectors: \( u_0, u_1, \ldots, u_{n_{SV}-1} \in \mathbb{R}^m \)
- Right singular vectors: \( v_0, v_1, \ldots, v_{n_{SV}-1} \in \mathbb{R}^n \)

There are \( n_{SV} \) singular values (counted with their multiplicities)

Computational requirements:
- Compute a few smallest or largest \( \sigma_i \)'s
- Solve the eigenproblem \( A^T A \)
- Solve the eigenproblem \( H(A) = \begin{bmatrix} 0_{m \times m} & A \\ A^T & 0_{n \times n} \end{bmatrix} \)
- Bidiagonalization
Quadratic Eigenvalue Problems

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

Where,
- \(\lambda\) 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
Solving Quadratic Eigenvalue Problems

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

Where,
✦ there are 2 X nev eigenvalues

Alternatives:
✦ Linearization \( A_\lambda = \lambda B_\lambda \)

\[ z = \begin{bmatrix} x \\ \lambda x \end{bmatrix} \quad A = \begin{bmatrix} 0 & I \\ -K & -C \end{bmatrix} \quad B = \begin{bmatrix} I & 0 \\ 0 & M \end{bmatrix} \]
✦ 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
  - **ST**: Spectral Transformation
  - **SVD**: Singular Value Decomposition
  - **QEP**: Quadratic Eigenvalue Problem

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
  - **SVD**: Singular Value Decomposition
  - **QEP**: Quadratic Eigenvalue Problem

- 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
  - **QEP**: Quadratic Eigenvalue Problem

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 linearization to a generalized eigenproblem or the specialized solver (Q-Arnoldi)
### Characteristics of the SLEPc Library

#### SNES
- Nonlinear Systems
  - Line Search
  - Trust Region
  - Other

#### PETSc
- Time Steppers
  - Euler
  - Backward Euler
  - Other
  - Time Stepping
- Krylov Subspace Methods
  - GMRES
  - CG
  - CGS
  - Bi-CGSTab
  - TFQMR
  - Richardson
  - Chebychev
  - Other
- Preconditioners
  - Additive Schwarz
  - Block Jacobi
  - Jacobi
  - ILU
  - ICC
  - LU
  - Other

#### SLEPc
- SVD
  - Cross Product
  - Cyclic Matrix
  - Lanczos
  - Thick R. Lanczos
- Quadratic
  - Linearization
  - Q-Arnoldi

#### EPS
- Eigensolvers
  - Krylov-Schur
  - Arnoldi
  - Lanczos
  - GD
  - JD
  - Other

#### QEP
- Shift
  - Shift-and-invert
  - Cayley
  - Fold
  - Preconditioner

#### Mat
- Compressed Sparse Row
  - Block Compressed Sparse Row
  - Block Diagonal
  - Dense
  - Other

#### Vec
- Vectors

#### Is
- Index Sets
  - Indices
  - Block Indices
  - Stride
  - Other
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

```c
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)**

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>EPSProblemType</th>
<th>Command line option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hermitian</td>
<td>EPS_HEP</td>
<td>-eps_hermitian</td>
</tr>
<tr>
<td>Generalized Hermitian</td>
<td>EPS_GHEP</td>
<td>-eps_gen_hermitian</td>
</tr>
<tr>
<td>non-Hermitian</td>
<td>EPS_NHEP</td>
<td>-eps_non_hermitian</td>
</tr>
<tr>
<td>Generalized non-Hermitian</td>
<td>EPS_GNHEP</td>
<td>-eps_gen_non_hermitian</td>
</tr>
<tr>
<td>GNHEP with B &gt; 0</td>
<td>EPS_PGNHEP</td>
<td>-eps_pos_gen_non_hermitian</td>
</tr>
</tbody>
</table>

SLEPc Thirteenth DOE ACTS Collection Workshop
Berkeley, California, August 14-17, 2012
Defining the Eigenproblem in SLEPc

**EPSSetFromOptions(EPS eps)**
Looks in the command line for options related to EPS

For example, the following command line

```plaintext
% program -eps_hermitian
```

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

Other options have an associated function call

```plaintext
% program -epsnev 6 -eps_tol 1e-8
```

**EPSView(EPS eps, PetscViewer viewer)**
Prints information about the object (equivalent to `-eps_view`)

---

SLEPc

Thirteenth DOE ACTS Collection Workshop
Berkeley, California, August 14-17, 2012
Profiling in SLEPc (EPS)

Sample output of \texttt{-eps_view}

EPS Object:
- problem type: symmetric eigenvalue problem
- method: krylovshur
- 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 \[ Ax = \lambda x \] \[ \implies \] Transformed System \[ Tx = \theta x \]

- 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

<table>
<thead>
<tr>
<th>ST Type</th>
<th>Standard problem</th>
<th>Generalized problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>shift</td>
<td>( A + \sigma I )</td>
<td>( B^{-1}A + \sigma I )</td>
</tr>
<tr>
<td>fold</td>
<td>( (A + \sigma I)^2 )</td>
<td>( (B^{-1}A + \sigma I)^2 )</td>
</tr>
<tr>
<td>sinvert</td>
<td>( (A - \sigma I)^{-1} )</td>
<td>( (A - \sigma B)^{-1}B )</td>
</tr>
<tr>
<td>cayley</td>
<td>( (A - \sigma I)^{-1}(A + \tau I) )</td>
<td>( (A - \sigma B)^{-1}(A + \tau B) )</td>
</tr>
<tr>
<td>precond</td>
<td>( K^{-1} \approx (A - \sigma I)^{-1} )</td>
<td>( K^{-1} \approx (A - \sigma B)^{-1} )</td>
</tr>
</tbody>
</table>
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
  -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

```c
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);
SVDSSetOperator(svd, A);
SVDSSetFromOptions(svd);

SVDSolve(svd);

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

SVDDestroy(svd);
```
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

```c
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**

\[ \text{QEPSetOperators}( \text{QEP qep, Mat M, Mat C, Mat K}) \]

Define the QEP through matrices \(M, C,\) and \(K\)

\[ \text{QEPSetProblemType}( \text{QEP qep, QEPProblemType type}) \]

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>EPSProblemType</th>
<th>Command line option</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>QEP_GENERAL</td>
<td>-qep_general</td>
</tr>
<tr>
<td>hermitian</td>
<td>QEP_HERMITIAN</td>
<td>-qep_hermitian</td>
</tr>
<tr>
<td>Gyroscopic</td>
<td>QEP_GYROSCOPIC</td>
<td>-qep_gyroscopic</td>
</tr>
</tbody>
</table>
Additional Parameters for the QEP in SLEPc

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