

Solving Nonlinear Eigenvalue Problems with SLEPc

Jose E. Roman

D. Sistemes Informàtics i Computació
Universitat Politècnica de València, Spain

Joint work with Carmen Campos

NLA-HPC 2013, Hsinchu (Taiwan)



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Nonlinear Eigenproblems

Increasing interest in nonlinear eigenvalue problems arising in many application domains

- ▶ Structural analysis with damping effects
- ▶ Vibro-acoustics (fluid-structure interaction)
- ▶ Linear stability of fluid flows

Problem types

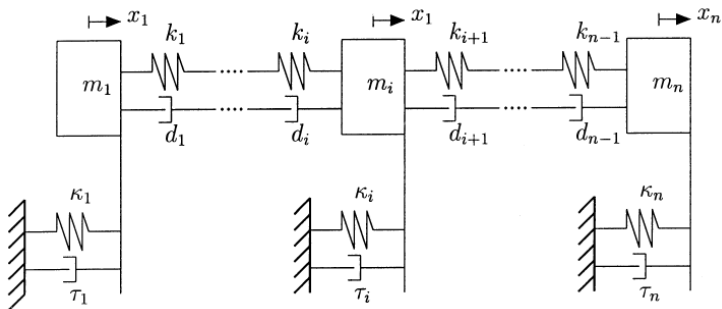
- ▶ QEP: quadratic eigenproblem, $(\lambda^2 M + \lambda C + K)x = 0$
- ▶ PEP: polynomial eigenproblem, $P(\lambda)x = 0$
- ▶ REP: rational eigenproblem, $P(\lambda)Q(\lambda)^{-1}x = 0$
- ▶ NEP: general nonlinear eigenproblem, $T(\lambda)x = 0$

Test cases available in the NLEVP collection [Betcke et al. 2013]



Example from NLEVP Collection

Connected damped mass-spring system



Second-order differential equation: $M \frac{d^2}{dt^2} x + C \frac{d}{dt} x + K x = 0$

Solution Strategy

We are interested in **large-scale** nonlinear eigenvalue problems, with methods appropriate for **sparse** matrices on **parallel** computers

- ▶ Only a small part of the spectrum

Alternatives for the PEP:

1. Projection method such as Jacobi-Davidson, $P(\theta)u \perp \mathcal{K}$
2. **Linearization**: solve via a linear eigenproblem

Linearization can leverage existing linear eigensolvers, but...

- ▶ Dimension of linearized problem is dn

→ Need memory-efficient variants capable of exploiting structure

The NEP requires completely different approaches

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 SVD, QEP, and more

$$Ax = \lambda x \quad Ax = \lambda Bx \quad Av_i = \sigma_i u_i \quad (\lambda^2 M + \lambda C + K)x = 0$$

Co-authors: C. Campos, E. Romero, A. Tomas

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

Current version: 3.4 (released July 2013)



PETSc/SLEPc Numerical Components

PETSc

Nonlinear Systems			Time Steppers			
Line Search	Trust Region	Other	Euler	Backward Euler	Pseudo Time Step	Other
Krylov Subspace Methods						
GMRES	CG	CGS	Bi-CGStab	TFQMR	Richardson	Chebyshev
Other						
Preconditioners						
Additive Schwarz	Block Jacobi	Jacobi	ILU	ICC	LU	Other
Matrices						
Compressed Sparse Row	Block CSR	Symmetric Block CSR	Dense	CUSP	Other	
Vectors		Index Sets				
Standard	CUSP	Indices	Block	Stride	Other	

PETSc/SLEPc Numerical Components

PETSc

Nonlinear Systems			Time Steppers				
Line Search	Trust Region	Other	Euler	Backward Euler	Pseudo Time Step	Other	
Krylov Subspace Methods							
GMRES	CG	CGS	Bi-CGSTab	TFQMR	Richardson	Chebyshev	Other
Preconditioners							
Additive Schwarz	Block Jacobi		Jacobi	ILU	ICC	LU	Other
Matrices							
Compressed Sparse Row		Block CSR	Symmetric Block CSR		Dense	CUSP	Other
Vectors			Index Sets				
Standard		CUSP	Indices	Block	Stride	Other	

SLEPc

Quadratic Eigensolver			Nonlinear Eigensolver		
Linear-ization	Q-Arnoldi	Q-Lanczos	SLP	RII	N-Arnoldi
SVD Solver			M. Function		
Cross Product	Cyclic Matrix	Lanczos	Thick R. Lanczos	Krylov	
Linear Eigensolver					
Krylov-Schur	GD	JD	RQCG	CISS	Other
Spectral Transformation					
Shift	Shift-and-invert	Cayley	Fold	Preconditioner	
IP		DS		FN	



Outline

- 1 Introduction
- 2 PEP: Polynomial Eigensolvers
 - Q-Arnoldi and TOAR for QEP
 - Extension to PEP
 - Usage and Numerical Results
- 3 NEP: General Nonlinear Eigensolvers
 - User Interface
 - Examples



PEP: Polynomial Eigensolvers

Quadratic Arnoldi

QEP: $(\lambda^2 M + \lambda C + K)x = 0$ linearized as

$$A - \lambda B = \begin{bmatrix} 0 & N \\ -K & -C \end{bmatrix} - \lambda \begin{bmatrix} N & 0 \\ 0 & M \end{bmatrix}$$

with either $N = I$ or $N = -K$. The eigenvector is $[x^*, \lambda x^*]^*$

Q-Arnoldi [Meerbergen 2008] builds an Arnoldi relation for $B^{-1}A$

$$\begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}C \end{bmatrix} \begin{bmatrix} V_k^0 \\ V_k^1 \end{bmatrix} = \begin{bmatrix} V_k^0 & v^0 \\ V_k^1 & v^1 \end{bmatrix} \underline{H}$$

From the 1st row block:

$$V_k^1 = [V_k^0, v^0] \underline{H}$$

Hence, only $\{V_k^0, \underline{H}, v\}$ needs to be stored

Q-Arnoldi

For each $i = 1, \dots, k$

1. Expand: $w^0 = v^1$, $w^1 = -M^{-1}(Kv^0 + Cv^1)$
2. Gram-Schmidt coefficients

$$h_j = \begin{bmatrix} V_k^0 & v^0 \\ V_k^1 & v^1 \end{bmatrix}^* w = \begin{bmatrix} V_k^{0*} w^0 + V_k^{1*} w^1 \\ u^* w \end{bmatrix}$$

$$\text{with } V_k^{1*} w^1 = \underline{H}^* [V_k^0, v^0]^* w^1$$

3. Gram-Schmidt update

$$\begin{aligned} \tilde{v}^0 &= w^0 - [V_k^0, v^0] h_j \\ \tilde{v}^1 &= w^1 - [[V_k^0, v^0] \underline{H}, v^1] h_j \end{aligned}$$

4. Normalize with $\|\tilde{v}\|_2$

Stabilized Variant

Q-Arnoldi presents instability when $\|H\|$ gets large

- Problem comes from representing the Krylov basis with matrices $[V_k^0, v^0]$ and \underline{H} , whose columns are not orthogonal

TOAR: Two-level Orthogonalization Arnoldi [Su et al. 2008]

- Main idea: use only orthogonal matrices

$$V_k = \begin{bmatrix} U_{k+1} & \\ & U_{k+1} \end{bmatrix} \begin{bmatrix} G_k^0 \\ G_k^1 \end{bmatrix}$$

Arnoldi relation:

$$\begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}C \end{bmatrix} \begin{bmatrix} U_{k+1}G_k^0 \\ U_{k+1}G_k^1 \end{bmatrix} = \begin{bmatrix} U_{k+2}G_{k+1}^0 \\ U_{k+2}G_{k+1}^1 \end{bmatrix} \underline{H}_k$$

Main Steps in TOAR

At step j , the new vector w is computed similarly to Q-Arnoldi. Then the bottom part w^1 is used to extend U_{j+1} to obtain a basis of $\text{span}([V_j^0, V_j^1, w^0, w^1])$

The new basis vector $u_{j+2} \in U_{j+1}^\perp$ is obtained via Gram-Schmidt, together with $g = \begin{bmatrix} g^0 \\ g^1 \end{bmatrix}$ so that $w = \begin{bmatrix} [U_{j+1} & u_{j+2}]g^0 \\ [U_{j+1} & u_{j+2}]g^1 \end{bmatrix}$

Gram-Schmidt coefficients:

$$h_j = V_j^* w = \begin{bmatrix} G_j^0 \\ G_j^1 \end{bmatrix}^* \begin{bmatrix} U_{j+1}^* & \\ & U_{j+1}^* \end{bmatrix} \begin{bmatrix} U_{j+1} g_j^1 \\ U_{j+1} \hat{w} + u_{j+2} \alpha \end{bmatrix} = G_j^* \begin{bmatrix} g_j^1 \\ \hat{w} \end{bmatrix}$$

The orthogonalization of w can be expressed as

$$\tilde{w} = w - V_j h_j = \begin{bmatrix} [U_{j+1} & u_{j+2}] \\ [U_{j+1} & u_{j+2}] \end{bmatrix} \left(\begin{bmatrix} g^0 \\ g^1 \end{bmatrix} - \begin{bmatrix} G_j^0 \\ G_j^1 \end{bmatrix} h_j \right)$$

Linearizations

PEP: $P(\lambda)x = 0$ $P(\lambda) = A_0 f_0(\lambda) + \dots + A_d f_d(\lambda)$

Monomial basis: $f_j(\lambda) = \lambda^j$

► Companion linearization $\mathcal{L}_0 - \lambda \mathcal{L}_1$, with

$$\mathcal{L}_0 = \begin{bmatrix} & I & & \\ & & \ddots & \\ & & & I \\ -A_0 & -A_1 & \cdots & -A_{d-1} \end{bmatrix} \quad \mathcal{L}_1 = \begin{bmatrix} I & & & \\ & \ddots & & \\ & & I & \\ & & & A_d \end{bmatrix} \quad y = \begin{bmatrix} x \\ \lambda x \\ \vdots \\ \lambda^{d-1} x \end{bmatrix}$$

Chebyshev basis: $f_j(\lambda) = \tau_j(\lambda)$ with $\tau_j(\lambda) = 2\lambda\tau_{j-1}(\lambda) - \tau_{j-2}(\lambda)$

► Linearization contains more nonzero blocks

Other bases/lineariz. [Amiraslani et al. 2009, De Terán et al. 2010]

Shift-and-Invert Transformation

For computing eigenvalues closest to a target σ

- Can be applied to the linearization or the original problem

Transformed QEP: $(\theta^2 M_\sigma + \theta C_\sigma + K_\sigma)x = 0$

$$M_\sigma = \sigma^2 M + \sigma C + K$$

$$C_\sigma = C + 2\sigma M$$

$$K_\sigma = M$$

The relation with the original eigenvalue is $\theta = (\lambda - \sigma)^{-1}$

In the polynomial case, use the reverse of the shifted polynomial $\tilde{P}(\theta)$ with coefficients

$$T_k = \sum_{j=0}^{d-k} \binom{j+k}{k} \sigma^j A_{j+k}$$

Polynomial TOAR

Arnoldi decomposition of order j for $S := \mathcal{L}_1^{-1} \mathcal{L}_0$

$$SV_j = \begin{bmatrix} V_j & v \end{bmatrix} \underline{H}_j, \quad \text{vectors split as} \quad v = \begin{bmatrix} v^0 \\ \vdots \\ v^{d-1} \end{bmatrix}$$

Due to the structure of S we have the relations

$$V_j^{i+1} = \begin{bmatrix} V_j^i & v^i \end{bmatrix} \underline{H}_j \quad i = 0, \dots, d-2$$

If U_{j+d} is an orthogonal basis of

$$\text{span}([V_j^0, \dots, V_j^{d-1}, v^0, \dots, v^{d-1}]) = \text{span}([V_j^0, v^0, \dots, v^{d-1}])$$

then it is possible to represent

$$\begin{bmatrix} V_j^i & v^i \end{bmatrix} = U_{j+d} \begin{bmatrix} G_j^i & g^i \end{bmatrix}, \quad i = 0, \dots, d-1$$

$$\text{Arnoldi: } S(I_d \otimes U_{j+d-1})G_j = (I_d \otimes U_{j+d}) \begin{bmatrix} G_j & g_{j+1} \end{bmatrix} \underline{H}_j$$

Can also be adapted to Chebyshev basis [Kressner/Roman 2013]

Scaling

Scaling the QEP can improve the **backward error** of the solutions obtained via linearization [Tisseur 2000]

The FLV scheme [Fan et al. 2004] replaces the QEP with the **scaled problem**

$$(\mu^2\gamma^2\delta M + \mu\gamma\delta C + \delta K)x = 0$$

where $\lambda = \gamma\mu$ with $\gamma = \sqrt{\|K\|/\|M\|}$, $\delta = 2/(\|K\| + \gamma\|C\|)$

Betcke [2008] extends this to polynomials of arbitrary degree as well as more general diagonal scalings $D_1P(\lambda)D_2$

- Scaling is optimal for a given λ

Extraction and Refinement

Invariant pairs generalize the concept of eigenpair to subspaces [Betcke/Kressner 2011]. From a computed invariant pair (\tilde{Y}, \tilde{S}) of $L(\lambda)$, **extract** an approximate invariant pair (\tilde{X}, \tilde{S}) for $P(\lambda)$

The structured strategy minimizes the norm $\left\| \begin{bmatrix} \tilde{X} \\ \vdots \\ \tilde{X} \tilde{S}^{d-1} \end{bmatrix} - \begin{bmatrix} \tilde{Y}_0 \\ \vdots \\ \tilde{Y}_{d-1} \end{bmatrix} \right\|_F$

In TOAR it is possible to exploit the structure of U

$$X = \left(\sum_{j=0}^{d-1} \tilde{Y}_j (\tilde{S}^j)^* \right) \left(\sum_{j=0}^{d-1} \tilde{S}^j (\tilde{S}^j)^* \right)^{-1} = U_{j+d} \left(\sum_{j=0}^{d-1} G^j (\tilde{S}^j)^* \right) \left(\sum_{j=0}^{d-1} \tilde{S}^j (\tilde{S}^j)^* \right)^{-1}$$

Extraction may not be enough and **iterative refinement** is required

- A few steps of Newton method for invariant pairs [Betcke/Kressner 2011]. Also for NEP [Kressner 2009]

Basic PEP Usage

```

PEP          pep;          /* eigensolver context */
Mat          A[5];         /* matrices of the PEP */
Vec          xr, xi;       /* eigenvector, x      */
PetscScalar  kr, ki;       /* eigenvalue, k      */
PetscInt     j, nconv;
PetscReal    error;

PEPCreate( PETSC_COMM_WORLD, &pep );
PEPSetOperators( pep, 5, A );
PEPSetFromOptions( pep );
PEPSolve( pep );
PEPGetConverged( pep, &nconv );
for (j=0; j<nconv; j++) {
    PEPGetEigenpair( pep, j, &kr, &ki, xr, xi );
    PEPComputeRelativeError( pep, j, &error );
}
PEPDestroy( pep );
    
```

Sample Command-line Usage

```
$ ./ex23 -pep_type toar -pep_nev 6 -pep_ncv 24
```

```
$ ./ex23 -pep_type toar -pep_tol 1e-8  
          -pep_max_it 2000 -pep_scale 1e3
```

```
$ ./ex23 -pep_type toar -st_type sinvert -pep_target 2  
          -st_ksp_type preonly -st_pc_type lu
```

```
$ ./ex23 -pep_type toar -st_type sinvert -pep_target 0  
          -st_ksp_type bcgs -st_pc_type bjacobi
```

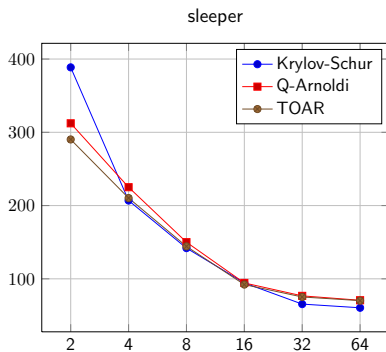
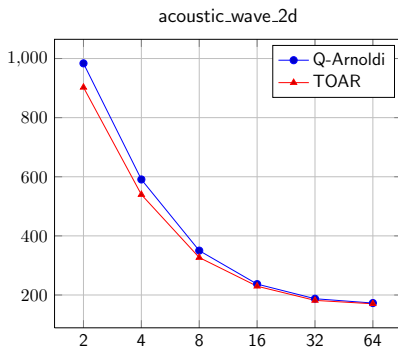
Some Experiments with SLEPc Implementation

PEP: Maximum residual error for tolerance 10^{-8} , maximum basis size $k = 2\text{nev}$ (times for 64 processes)

name	method	nev	nconv	its	$\max_i \{\ r_i\ \}$	time
<i>acoustic_wave_2d</i>	Krylov-Schur	60	-	-	-	-
$n = 6, 247, 500$	Q-Arnoldi	60	66	3	2.0e-10	172.8
$d = 2, \sigma = 0$	TOAR	60	66	3	1.4e-10	170.3
<i>sleeper</i>	Krylov-Schur	40	40	2	1.8e-12	60.5
$n = 5, 000, 000$	Q-Arnoldi	40	41	2	7.0e-09	70.8
$d = 2, \sigma = -0.9$	TOAR	40	41	2	4.2e-07	70.3
<i>plasma-drift</i>	TOAR	20	20	11	3.5e-8	-
$n = 512$						
$d = 3, \sigma = 0.01$						
<i>planar_waveguide</i>	TOAR	10	10	41	3.4e-7	-
$n = 129$						
$d = 4, \sigma = 0$						

Parallel Performance

Parallel computing times on MareNostrum III from Barcelona Supercomputing Center





NEP: General Nonlinear Eigensolvers

General Nonlinear Eigenproblems

NEP:

$$T(\lambda)x = 0, \quad x \neq 0$$

$T : \Omega \rightarrow \mathbb{C}^{n \times n}$ is a matrix-valued function analytic on $\Omega \subset \mathbb{C}$

Example 1: REP arising in the study of free vibration of plates with elastically attached masses

$$-Kx + \lambda Mx + \sum_{j=1}^k \frac{\lambda}{\sigma_j - \lambda} C_j x = 0$$

All matrices symmetric, $K > 0$, $M > 0$ and C_j have small rank

Example 2: Discretization of parabolic PDE with time delay τ

$$(-\lambda I + A + e^{-\tau\lambda} B)x = 0$$

Split Form

The NEP can always be rewritten as

$$(A_0 f_0(\lambda) + A_1 f_1(\lambda) + \dots + A_{\ell-1} f_{\ell-1}(\lambda))x = \left(\sum_{i=0}^{\ell-1} A_i f_i(\lambda) \right) x = 0,$$

with A_i $n \times n$ matrices and $f_i : \Omega \rightarrow \mathbb{C}$ analytic functions

We will call this the **split form** of the NEP

- ▶ Often, the formulation arising from applications already has this form, as in the examples
- ▶ The PEP fits this form, the f_i functions being the polynomial bases of degree i , either monomial or non-monomial
- ▶ The NEP can be approximated by a PEP via interpolation, see e.g. [Kressner/Roman 2013]

NEP Usage with Callbacks

Approach based on user-defined functions to compute $T(\lambda), T'(\lambda)$

```
NEP          nep;          /* eigensolver context */
Mat          F, J;         /* Function and Jacobian matrices */
Vec          x;           /* eigenvector */
PetscScalar  lambda;      /* eigenvalue */
AppIctx      ctx;         /* user-defined context */
PetscInt     j, nconv;
PetscReal    error;

NEPCreate( PETSC_COMM_WORLD, &nep );
/* create and preallocate F and J matrices */
NEPSetFunction( nep, F, F, FormFunction, &ctx );
NEPSetJacobian( nep, J, FormJacobian, &ctx );
NEPSetFromOptions( nep );
NEPSolve( nep );
NEPGetConverged( nep, &nconv );
for (j=0; j<nconv; j++) {
    NEPGetEigenpair( nep, j, &lambda, x );
    NEPComputeRelativeError( nep, j, &error );
}
NEPDestroy( nep );
```

FN: Mathematical Functions

The FN class provides a few predefined mathematical functions

- ▶ The user specifies the type and relevant coefficients α_i and β_j

1. Rational function (includes polynomial):

$$r(x) = \frac{p(x)}{q(x)} = \frac{\alpha_1 x^{n-1} + \dots + \alpha_{n-1} x + \alpha_n}{\beta_1 x^{m-1} + \dots + \beta_{m-1} x + \beta_m}$$

2. Exponential function with scaling:

$$g(x) = \beta_1 e^{\alpha_1 x}$$

To be done:

- ▶ Add more functions and allow for user-defined ones
- ▶ Some eigensolvers require to evaluate $f_i(X)$ on a small matrix

NEP Usage in Split Form

The user provides an array of matrices A_i and functions f_i

```
FNCreate(PETSC_COMM_WORLD,&f1);      /* f1 = -lambda */
FNSetType(f1,FNRATIONAL);
coeffs[0] = -1.0; coeffs[1] = 0.0;
FNSetParameters(f1,2,coeffs,0,NULL);

FNCreate(PETSC_COMM_WORLD,&f2);      /* f2 = 1 */
FNSetType(f2,FNRATIONAL);
coeffs[0] = 1.0;
FNSetParameters(f2,1,coeffs,0,NULL);

FNCreate(PETSC_COMM_WORLD,&f3);      /* f3 = exp(-tau*lambda) */
FNSetType(f3,FNEXP);
coeffs[0] = -tau;
FNSetParameters(f3,1,coeffs,0,NULL);

mats[0] = A;  funcs[0] = f2;
mats[1] = Id; funcs[1] = f1;
mats[2] = B;  funcs[2] = f3;
NEPSetSplitOperator(nep,3,mats,funcs,SUBSET_NONZERO_PATTERN);
```

Currently Available NEP Solvers

1. Residual inverse iteration (RII) [Neumaier 1985]
 - ▶ Eigenvector correction computed as $T(\sigma)^{-1}r$
2. Successive linear problems (SLP) [Ruhe 1973]
 - ▶ In each iteration a linear eigenvalue problem $T(\tilde{\lambda})\tilde{x} = \mu T'(\tilde{\lambda})\tilde{x}$ is solved for the eigenvalue correction μ
3. Nonlinear Arnoldi [Voss 2004]
 - ▶ Builds an orthogonal basis V_j of a subspace expanded with the vectors generated by RII
 - ▶ Then chooses approximate eigenpair $(\tilde{\lambda}, \tilde{x})$ such that $\tilde{x} = V_j y$ and $V_j^* T(\tilde{\lambda}) V_j y = 0$
 - ▶ Requires the split form

Sample Command-line Usage

```
$ ./ex20 -nep_type rii -nep_target 20  
        -nep_ksp_type preonly -nep_pc_type lu  
        -nep_lag_preconditioner 2  
  
$ ./ex20 -nep_type slp -nep_rtol 1e-8 -nep_max_it 2000  
        -nep_st_ksp_type preonly -nep_st_pc_type cholesky  
  
$ ./ex22 -nep_type narnoldi -nep_ncv 30  
        -nep_ksp_type bcgs -nep_pc_type ilu  
        -nep_const_correction_tol 1 -nep_ksp_rtol 1e-3
```

Conclusion

Ongoing work to provide parallel nonlinear eigensolvers in SLEPc

PEP:

- ▶ Extending existing QEP solvers
- ▶ Solvers based on linearization, memory-efficient variants
- ▶ Reasonable robustness via scaling, invariant pair extraction and iterative refinement
- ▶ Future: consider other bases/linearizations

NEP:

- ▶ Two user interfaces: callbacks and split form
- ▶ Current solvers are simple, compute just one eigenpair
- ▶ Future: NLEIGS, contour integral, interpolation, ...