SLEPc
Current Achievements and Plans for the Future

Jose E. Roman

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

Celebrating 20 years of PETSc, Argonne – June, 2015

SLEPc: Scalable Library for Eigenvalue Problem Computations

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

- Linear eigenproblems (standard or generalized, real or complex, Hermitian or non-Hermitian)
- Also support for SVD, PEP, NEP and more

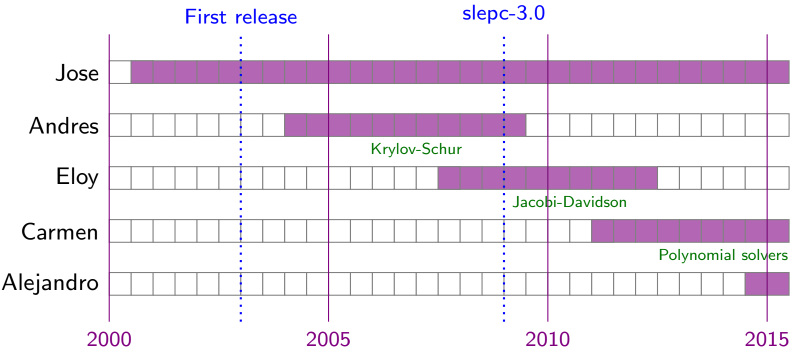
$$Ax = \lambda x \qquad Ax = \lambda Bx \qquad Av_i = \sigma_i u_i \qquad T(\lambda)x = 0$$

Authors: J. E. Roman, C. Campos, E. Romero, A. Tomas

`http://slepc.upv.es`

Current version: 3.6 (released June 2015)

Timeline




Jose Roman


Andres Tomas


Eloy Romero


Carmen Campos

Alejandro Lamas

Applications

Google Scholar: 320 citations of main paper (ACM TOMS 2005)

Nuclear Engineering	6 %
Computational Electromagnetics, Electronics, Photonics	9 %
Plasma Physics	11 %
Astrophysics	1 %
Computational Physics, Materials Science, Electronic Structure	20 %
Acoustics	4 %
Computational Fluid Dynamics	13 %
Earth Sciences, Oceanology, Hydrology, Geophysics	4 %
Bioengineering, Computational Neuroscience	2 %
Structural Analysis, Mechanical Engineering	6 %
Information Retrieval, Machine Learning, Graph Algorithms	7 %
Visualization, Computer Graphics, Image Processing	3 %
PDE's, Numerical Methods	10 %
Dynamical Systems, Model Reduction, Inverse Problems	4 %

Problem Classes

The user must choose the most appropriate solver for each problem class

Problem class	Model equation	Module
Linear eigenproblem	$Ax = \lambda x, \quad Ax = \lambda Bx$	EPS
Quadratic eigenproblem	$(K + \lambda C + \lambda^2 M)x = 0$	†
Polynomial eigenproblem	$(A_0 + \lambda A_1 + \cdots + \lambda^d A_d)x = 0$	PEP
Nonlinear eigenproblem	$T(\lambda)x = 0$	NEP
Singular value decomp.	$Av = \sigma u$	SVD
Matrix function	$y = f(A)v$	MFN

† QEP removed in version 3.5

Auxiliary classes: ST, BV DS, RG, FN

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 CSR	Symmetric Block CSR	Dense	CUSP	Other

Vectors	
Standard	CUSP

Index Sets			
Indices	Block	Stride	Other

SLEPc

Polynomial Eigensolver		
TOAR	Q-Arnoldi	Linear-ization

Nonlinear Eigensolver			
SLP	RII	N-Arnoldi	Interp.

SVD Solver			
Cross Product	Cyclic Matrix	Lanczos	Thick R. Lanczos

M. Function
Krylov

Linear Eigensolver					
Krylov-Schur	GD	JD	LOBPCG	CISS	Other

Spectral Transformation			
Shift	Shift-and-invert	Cayley	Preconditioner

BV	DS	RG	FN

Outline

EPS: Eigenvalue Problem Solver

Compute a few eigenpairs (x, λ) of

Standard Eigenproblem

$$Ax = \lambda x$$

Generalized Eigenproblem

$$Ax = \lambda Bx$$

where A, B can be real or complex, symmetric (Hermitian) or not

User can specify:

- Number of eigenpairs (`nev`), subspace dimension (`ncv`)
- Tolerance, maximum number of iterations
- The solver
- Selected part of spectrum
- Advanced: extraction type, initial guess, constraints, balancing

Available Eigensolvers

User code is independent of the selected solver

1. Basic methods
   - Single vector iteration: power iteration, inverse iteration, RQI
   - Subspace iteration with Rayleigh-Ritz projection and locking
   - Explicitly restarted Arnoldi and Lanczos
2. **Krylov-Schur**, including thick-restart Lanczos
3. Generalized Davidson, Jacobi-Davidson
4. Conjugate gradient methods: LOBPCG, RQCG
5. CISS, a contour-integral solver
6. External packages, and LAPACK for testing

. . . but some solvers are specific for a particular case:

- LOBPCG computes smallest λ_i of symmetric problems
- CISS allows computation of all λ_i within a region

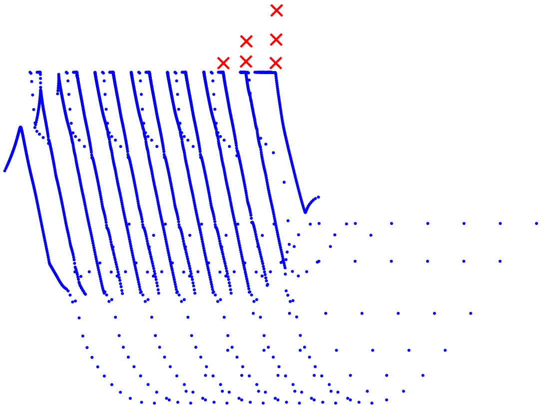
Selection of Eigenvalues (1): Basic

Largest/smallest magnitude, or real (or imaginary) part

Example: QC2534

```
-eps_nev 6
-eps_ncv 128
-eps_largest_imaginary
```

× Computed eigenvalues



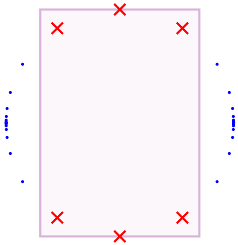
Selection of Eigenvalues (2): Region Filtering

`RG`: Region

- A region of the complex plane (interval, polygon, ellipse, ring)
- Used as an inclusion (or exclusion) region

Example: sign1 (NLEVP) $n = 225$, all λ lie at unit circle, accumulate at ± 1

```
-eps_nev 6
-rg_type interval
-rg_interval_endpoints -0.7,0.7,-1,1
```



Selection of Eigenvalues (3): Closest to Target

Shift-and-invert is used to compute interior eigenvalues

$$Ax = \lambda Bx \quad \Longrightarrow \quad (A - \sigma B)^{-1}Bx = \theta x$$

- Trivial mapping of eigenvalues: $\theta = (\lambda - \sigma)^{-1}$
- Eigenvectors are not modified
- Very fast convergence close to σ

Things to consider:

- Implicit inverse $(A - \sigma B)^{-1}$ via linear solves
- Direct linear solver for robustness
- Less effective for eigenvalues far away from σ

Selection of Eigenvalues (4): Interval (in GHEP)

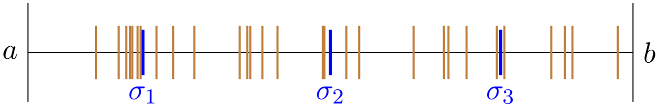
Indefinite (block-)triangular factorization: $A - \sigma B = LDL^T$

A byproduct is the number of eigenvalues on the left of σ (inertia)

$$\nu(A - \sigma B) = \nu(D)$$

Spectrum Slicing strategy:

- Multi-shift scheme that sweeps all the interval
- Compute eigenvalues by chunks
- Use inertia to validate sub-intervals



C. Campos and J. E. Roman, "Strategies for spectrum slicing based on restarted Lanczos methods", *Numer. Algorithms*, 60(2):279–295, 2012.

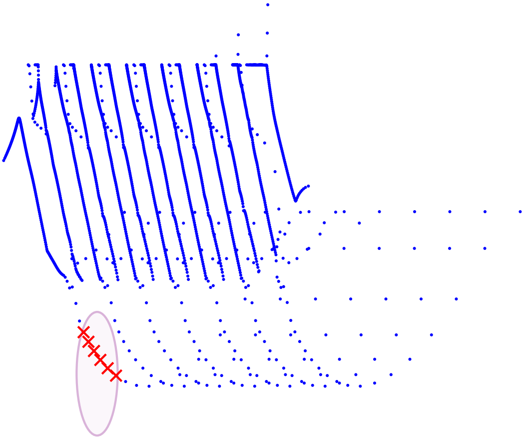
new Multi-communicator version, one subinterval per partition

Selection of Eigenvalues (5): All inside a Region

CISS solver[1]: compute all eigenvalues inside a given region

Example: QC2534

```
-eps_type ciss
-rg_type ellipse
-rg_ellipse_center -.8-.1i
-rg_ellipse_radius 0.2
-rg_ellipse_vscale 0.1
```

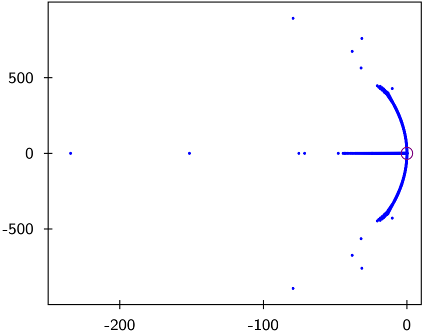


[1] Contributed by Y. Maeda, T. Sakurai

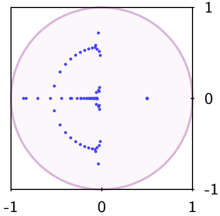
Selection of Eigenvalues (5): All inside a Region

Example: MHD1280 with CISS

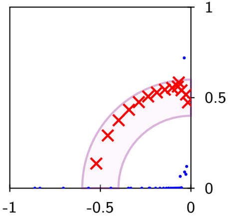
- Alfvén spectra: eigenvalues in intersection of the branches



RG=ellipse, center=0, radius=1



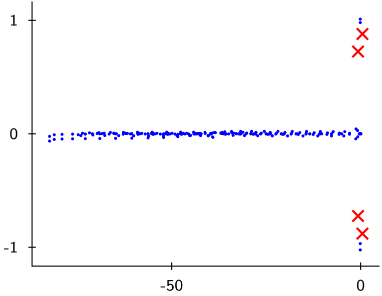
RG=ring, center=0, radius=0.5, width=0.2, angle=0.25..0.5



Selection of Eigenvalues (6): User-Defined

Selection with user-defined function for sorting eigenvalues

pdde_stability $n = 225$, wanted eigenvalues: $\|\lambda\| = 1$




```
PetscErrorCode MyEigenSort(PetscScalar ar,PetscScalar ai,
                           PetscScalar br,PetscScalar bi,PetscInt *r,void *ctx) {
  PetscReal aa,ab;
  PetscFunctionBeginUser;
  aa = PetscAbsReal(SlepcAbsEigenvalue(ar,ai)-1.0);
  ab = PetscAbsReal(SlepcAbsEigenvalue(br,bi)-1.0);
  *r = aa > ab ? 1 : (aa < ab ? -1 : 0);
  PetscFunctionReturn(0);
}
```

Arbitrary selection: apply criterion to an arbitrary user-defined function $\phi(\lambda, x)$ instead of just λ

Preconditioned Eigensolvers

Pitfalls of shift-and-invert:

- Direct solvers have high cost, limited scalability
- *Inexact* shift-and-invert (i.e., with iterative solver) not robust

Preconditioned eigensolvers try to overcome these problems

1. Davidson-type solvers
   - Jacobi-Davidson: correction equation with iterative solver
   - Generalized Davidson: simple preconditioner application

   E. Romero and J. E. Roman, "A parallel implementation of Davidson methods for large-scale eigenvalue problems in SLEPc", *ACM Trans. Math. Softw.*, 40(2):13, 2014.

2. Conjugate Gradient-type solvers (for GHEP)
   - RQCG: CG for the minimization of the Rayleigh Quotient
   - new LOBPCG: Locally Optimal Block Preconditioned CG

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]

Polynomial Eigenproblems via Linearization

PEP: $P(\lambda)x = 0$

Monomial basis: $P(\lambda) = A_0 + A_1\lambda + A_2\lambda^2 + \cdots + A_d\lambda^d$

Companion linearization: $L(\lambda) = \mathcal{L}_0 - \lambda\mathcal{L}_1$, with $L(\lambda)y = 0$ and

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

Compute an eigenpair (y, λ) of $L(\lambda)$, then extract x from y

- Pros: can leverage existing linear eigensolvers (`PEPLINEAR`)
- Cons: dimension of linearized problem is dn

SLEPc

PEP: Krylov Methods with Compact Representation

Arnoldi relation: $SV_j = \left[V_j \quad v\right] \underline{H}_j, \qquad S := \mathcal{L}_1^{-1}\mathcal{L}_0$

Write Arnoldi vectors as $v = \mathrm{vec}\left[v^0, \ldots, v^{d-1}\right]$

Block structure of S allows an implicit representation of the basis

- Q-Arnoldi: $V_j^{i+1} = \left[V_j^i \quad v^i\right] \underline{H}_j$
- TOAR: $\left[V_j^i \quad v^i\right] = U_{j+d}\left[G_j^i \quad g^i\right]$

Arnoldi relation in the compact representation:

$$S(I_d \otimes U_{j+d-1})G_j = (I_d \otimes U_{j+d})\left[G_j \quad g\right] \underline{H}_j$$

`PEPTOAR` is the default solver

- Memory-efficient (also in terms of computational cost)
- Many features: restart, locking, scaling, extraction, refinement

C. Campos and J. E. Roman, "Parallel Krylov solvers for the polynomial eigenvalue problem in SLEPc", submitted, 2015.

Shift-and-Invert on the Linearization

Set $S_\sigma := (\mathcal{L}_0 - \sigma\mathcal{L}_1)^{-1}\mathcal{L}_1$

Linear solves required to extend the Arnoldi basis $z = S_\sigma w$

$$\begin{bmatrix} -\sigma I & I & & & \\ & -\sigma I & \ddots & & \\ & & \ddots & I & \\ & & & -\sigma I & I \\ -A_0 & -A_1 & \cdots & -\tilde{A}_{d-2} & -\tilde{A}_{d-1} \end{bmatrix} \begin{bmatrix} z^0 \\ z^1 \\ \vdots \\ z^{d-2} \\ z^{d-1} \end{bmatrix} = \begin{bmatrix} w^0 \\ w^1 \\ \vdots \\ w^{d-2} \\ A_d w^{d-1} \end{bmatrix}$$

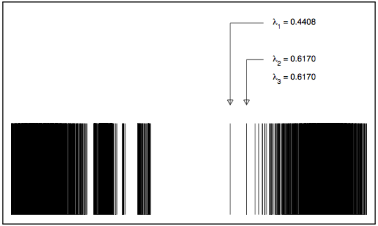
with $\tilde{A}_{d-2} = A_{d-2} + \sigma I$ and $\tilde{A}_{d-1} = A_{d-1} + \sigma A_d$

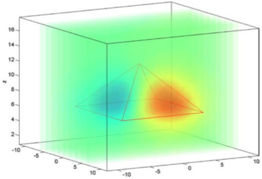
From the block LU factorization, we can derive a simple recurrence to compute $z^i \longrightarrow$ involves a linear solve with $P(\sigma)$

Quantum Dot Simulation

3D pyramidal quantum dot discretized with finite volumes

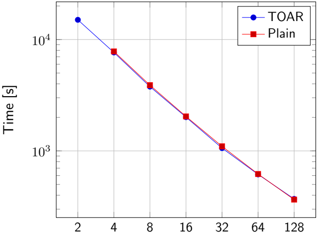
Tsung-Min Hwang et al. (2004). "Numerical Simulation of Three Dimensional Pyramid Quantum Dot," Journal of Computational Physics, 196(1): 208-232.





Quintic polynomial, $n \approx 12$ mill.

Scaling for tol$=10^{-8}$, nev=5, ncv=40 with inexact shift-and-invert (`bcgs+bjacobi`)



SLEPc

PEP: Additional Features

Non-Monomial polynomial basis

$$P(\lambda) = A_0\phi_0(\lambda) + A_1\phi_1(\lambda) + \cdots + A_d\phi_d(\lambda)$$

- Implemented for Chebyshev, Legendre, Laguerre, Hermite
- Enables polynomials of arbitrary degree

Newton iterative refinement

- Optional for ill-conditioned problems
- Implemented for single eigenpairs as well as invariant pairs

Other solvers not based on linearization

new `PEPJD` provides Jacobi-Davidson for polynomial eigenproblems

General Nonlinear Eigenproblems

NEP: $$T(\lambda)x = 0, \qquad x \neq 0$$

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

Example 1: Rational eigenproblem 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

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

NEP User Interface - Two Alternatives

Callback functions

The user provides code to compute $T(\lambda)$, $T'(\lambda)$

Split form

$T(\lambda)x = 0$ can always be rewritten as

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

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

- Often, the formulation from applications already has this form
- We need a way for the user to define f_i

FN: Mathematical Functions

The FN class provides a few predefined functions

- The user specifies the type and relevant coefficients
- Also supports evaluation of $f_i(X)$ on a small matrix

Basic functions:

1. Rational function (includes polynomial)

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

2. Other: exp, log, sqrt, φ-functions

new and a way to combine functions (with addition, multiplication, division or function composition), e.g.:

$$f(x) = (1 - x^2) \exp\left(\frac{-x}{1 + x^2}\right)$$

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;
FNRationalSetNumerator(f1,2,coeffs);

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

FNCreate(PETSC_COMM_WORLD,&f3);     /* f3 = exp(-tau*lambda) */
FNSetType(f3,FNEXP);
FNSetScale(f3,-tau,1.0);

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

Currently Available NEP Solvers

1. Single-vector iterations
   - Residual inverse iteration (RII) [Neumaier 1985]
   - Successive linear problems (SLP) [Ruhe 1973]
2. Nonlinear Arnoldi [Voss 2004]
   - Performs a projection on RII iterates, $V_j^* T(\tilde{\lambda}) V_j y = 0$
   - Requires the split form
3. Polynomial Interpolation: use PEP to solve $P(\lambda)x = 0$
   - $P(\cdot)$ is the interpolation polynomial in Chebyshev basis
4. new Contour Integral
   - Extension of the CISS method in EPS

MFN: Matrix Function

From the Taylor series expansion of e^A

$$y = e^A v = v + \frac{A}{1!}v + \frac{A^2}{2!}v + \cdots$$

so y can be approximated by an element of $\mathcal{K}_m(A, v)$

Given an Arnoldi decomposition $AV_m = V_{m+1}\underline{H}_m$

$$\tilde{y} = \beta V_{m+1} \exp(H_m) e_1$$

This extends to other functions $y = f(A)v$

What is needed:

- Efficient construction of the Krylov subspace
- Computation of $f(X)$ for a small dense matrix $\rightarrow$ `FN`

Auxiliary Classes

- `ST`: Spectral Transformation
- `FN`: Mathematical Function
  - Represent the constituent functions of the nonlinear operator in split form
  - Function to be used when computing $f(A)v$
- `RG`: Region (of the complex plane)
  - Discard eigenvalues outside the wanted region
  - Compute all eigenvalues inside a given region
- `DS`: Direct Solver (or Dense System)
  - High-level wrapper to LAPACK functions
- `BV`: Basis Vectors

BV: Basis Vectors

BV provides the concept of a block of vectors that represent the basis of a subspace; sample operations:

BVMult	$Y = \beta Y + \alpha XQ$
BVAXPY	$Y = Y + \alpha X$
BVDot	$M = Y^* X$
BVMatProject	$M = Y^* AX$
BVScale	$Y = \alpha Y$

Goal: to increase arithmetic intensity (BLAS-2 vs BLAS-1)

```
$ ./ex9 -n 8000 -eps_nev 32 -log_summary -bv_type vecs
BVMult    32563 1.0 3.2903e+01 1.0 6.61e+10 1.0 0.0e+00 0.0e+00 ...  2009
BVDot     32064 1.0 1.6213e+01 1.0 5.07e+10 1.0 0.0e+00 0.0e+00 ...  3128

$ ./ex9 -n 8000 -eps_nev 32 -log_summary -bv_type mat
BVMult    32563 1.0 2.4755e+01 1.0 8.24e+10 1.0 0.0e+00 0.0e+00 ...  3329
BVDot     32064 1.0 1.4507e+01 1.0 5.07e+10 1.0 0.0e+00 0.0e+00 ...  3497
```

Even better in block solvers (LOBPCG): BLAS-3, MatMatMult

Plans for Future Developments

Short term plans:

- More `EPS` solvers: improved LOBPCG, block Krylov methods
- More `PEP` solvers: SOAR, improved JD
- More `NEP` solvers: NLEIGS
- More `MFN` solvers: rational Krylov
- Improved GPU support in `BV`

A new solver class for Matrix equations

- Krylov methods for the continuous-time Lyapunov equation

$$AX + XA^T = C$$

- Other equations: Sylvester, Stein, Ricatti

Acknowledgements

Thanks to:

- The PETSc team
- Contributors
- Users providing feedback

Funding agencies:

GOBIERNO DE ESPAÑA — MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD

Computing resources:

BSC — Barcelona Supercomputing Center — Centro Nacional de Supercomputación