Nuclear Mathematical and Computational Sciences: A Century in Review, A Century Anew
Gatlinburg, Tennessee, April 6-11, 2003, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2003)

RESOLUTION OF THE NEUTRON DIFFUSION EQUATION WITH SLEPc, THE SCALABLE LIBRARY FOR EIGENVALUE PROBLEM COMPUTATIONS

V. Hernández, J. E. Román, and V. Vidal
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera, s/n, E-46022 Valencia, Spain
vhernand@dsic.upv.es; jroman@dsic.upv.es; vvidal@dsic.upv.es

G. Verdú
Departamento de Ingeniería Química y Nuclear
Universidad Politécnica de Valencia
Camino de Vera, s/n, E-46022 Valencia, Spain
gverdu@iqn.upv.es

D. Ginestar
Departamento de Matemática Aplicada
Universidad Politécnica de Valencia
Camino de Vera, s/n, E-46022 Valencia, Spain
dginesta@mat.upv.es

ABSTRACT

The objective of this work is to obtain the dominant λ-modes of a nuclear reactor. The underlying algebraic problem is a real generalised eigenvalue problem, which can be reduced to a standard one. A parallel code has been developed in order to be able to analyse reactors with a great level of detail. The implementation has been done using SLEPc, which is a public domain library developed by the authors. This library is based on PETSc and is intended to be an easy-to-use yet efficient object-oriented parallel framework for the solution of standard and generalised eigenproblems, with either real or complex arithmetic.

Key Words: Neutron Diffusion Equation, Eigenvalue problems, Krylov Subspace Methods

1. INTRODUCTION

The analysis of Lambda Modes is of great interest for nuclear reactor safety and modal analysis of neutron dynamical processes. In order to study the steady state neutron flux distribution inside a nuclear reactor and the sub-critical modes responsible for the regional instabilities that can take place in its core, it is necessary to obtain the dominant λ-modes and their corresponding eigenfunctions. The computation of λ-modes can also be important in transient analysis, in which the problem must be solved each certain time step. In this last case, it is critical to be able to compute the λ-modes solution as fast as possible.

The discretization of the problem leads to an algebraic eigensystem which can reach considerable sizes in real applications. The main aim of this work is to solve this problem by using appropriate numerical

methods and allowing parallel computing so that response time can be reduced. Additional benefits can be obtained with this approach. For example, larger problems can be faced and a better precision in the results can be attained.

The code which computes the solution to the problem has been developed using SLEPc [2], which is a public domain package developed by the authors. SLEPc is a general parallel library for the solution of standard and generalised eigenproblems and the Lambda Modes problem is a good example which can be used to illustrate all its possibilities.

The rest of the text is organised as follows. Section 2 describes the Lambda Modes equation and the associated eigenvalue problem. In section 3 a short description of PETSc and SLEPc is provided. Section 4 gives some details of the implementation of the solver and section 5 includes some numerical results. Finally, some concluding remarks are given.

2. THE LAMBDA MODES

Reactor calculations are usually based on the multi-group neutron diffusion equation [8]. If this equation is modelled with two energy groups, then the problem to be dealt with is to find the eigenvalues and eigenfunctions of

$$\mathcal{L}\phi_i = \frac{1}{\lambda_i}\mathcal{M}\phi_i \ , \tag{1}$$

where

$$\mathcal{L} = \begin{bmatrix} -\vec{\nabla}\left(D_1\vec{\nabla}\right) + \Sigma_{a1} + \Sigma_{12} & 0 \\ -\Sigma_{12} & -\vec{\nabla}\left(D_2\vec{\nabla}\right) + \Sigma_{a2} \end{bmatrix} \ , \tag{2}$$

$$\mathcal{M} = \begin{bmatrix} \nu_1\Sigma_{f1} & \nu_2\Sigma_{f2} \\ 0 & 0 \end{bmatrix} \ , \qquad \text{and} \qquad \phi_i = \begin{bmatrix} \phi_{1_i} \\ \phi_{2_i} \end{bmatrix} \ , \tag{3}$$

with the boundary conditions $\phi_i|_\Gamma = 0$, where Γ is the reactor border.

For a numerical treatment, this equation must be discretised in space. Nodal methods are extensively used in this case. These methods are based on approximations of the solution in each node in terms of an appropriate base of functions such as Legendre polynomials [5]. It is assumed that nuclear properties are constant in every cell. Finally, appropriate continuity conditions for fluxes and currents are enforced.

This process allows the transformation of the original system of partial differential equations into an algebraic large sparse generalised eigenvalue problem

$$L\psi_i = \frac{1}{\lambda_i}M\psi_i \ \ , \tag{4}$$

where L and M are matrices of order $2N$ with the following block structure

$$\begin{bmatrix} L_{11} & 0 \\ -L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} \psi_{1_i} \\ \psi_{2_i} \end{bmatrix} = \frac{1}{\lambda_i} \begin{bmatrix} M_{11} & M_{12} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \psi_{1_i} \\ \psi_{2_i} \end{bmatrix} \ , \tag{5}$$

being L_{11} and L_{22} nonsingular sparse matrices, and M_{11}, M_{12} and L_{21} diagonal matrices. By eliminating ψ_{2_i}, we obtain the following N-dimensional non-symmetric standard algebraic eigenproblem

$$A\psi_{1_i} = \lambda_i\psi_{1_i} \ \ , \tag{6}$$

where matrix A is given by

$$A = L_{11}^{-1}\left(M_{11} + M_{12}L_{22}^{-1}L_{21}\right) . \quad (7)$$

All the eigenvalues of this equation are real [6]. We are only interested in computing a few dominant ones with the corresponding eigenvectors.

3. THE SLEPc LIBRARY

Eigenvalue problems are a very important class of linear algebra problems. The need for the numerical solution of these problems, associated with stability and vibrational analysis, arises in a wide range of situations in science and engineering. They are usually formulated as large sparse eigenproblems. There is a lack of public domain parallel software for the solution of sparse eigenvalue problems, compared to other classes of problems. The most complete and widely used library is ARPACK [3].

In order to tackle this problem, a new library called SLEPc [2], the Scalable Library for Eigenvalue Problem Computations, has been designed and implemented by the authors. It is based on PETSc [1] and extends its functionality in order to solve eigenproblems arising in real applications by using well-known techniques such as shift-and-invert and state-of-the-art methods.

PETSc is a parallel numerical framework for the solution of partial differential equations and it provides support for problems arising from discretization by means of regular meshes as well as unstructured meshes. Its approach is to encapsulate mathematical algorithms using object-oriented programming techniques in order to be able to manage the complexity of efficient numerical message-passing codes. All the PETSc software is freely available and used around the world in many application areas.

PETSc is built around a variety of data structures and algorithmic objects (see Figure 1). The application programmer works directly with these objects rather than concentrating on the underlying data structures. The three basic abstract data objects are index sets (`IS`), vectors (`Vec`) and matrices (`Mat`). Built on top of this foundation are various classes of solver objects, including linear, nonlinear and time-stepping solvers.

3.1. Structure of SLEPc

The SLEPc package provides the eigensolver functionality which is missing in PETSc. It has been designed so that it can be used for standard and generalised problems, in both real and complex arithmetic, including solvers for either Hermitian and non-Hermitian problems. The problem can be specified with any instance of matrix object provided by PETSc, including matrix-free problems. It provides a Fortran interface.

For those users acquainted with PETSc, the use of SLEPc is straightforward. With little programming effort it is possible to easily test different solution strategies for a given eigenproblem. Due to the seamless integration with PETSc, the user has at his disposal a wide range of linear and non-linear equation solvers. In particular, the user holds control over the linear solvers associated with eigenproblems as if they were stand-alone objects. In addition, SLEPc inherits all the good properties of PETSc, including portability, scalability, efficiency and flexibility.

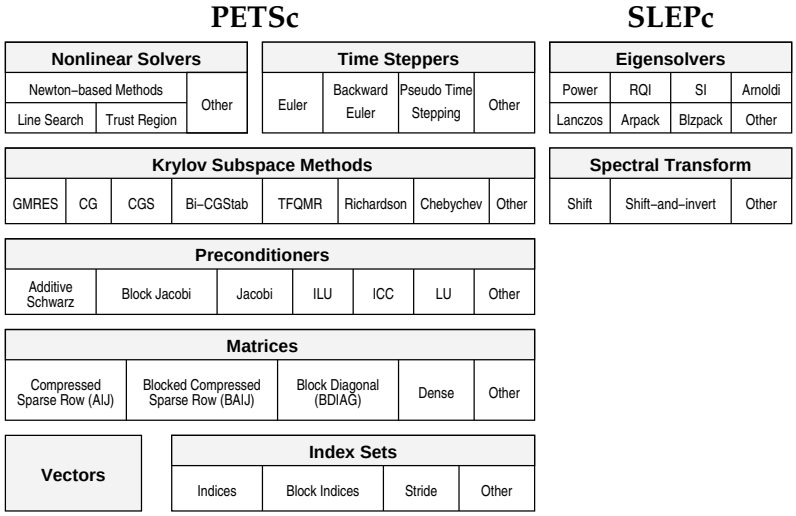

Figure 1. Structure of PETSc (on the left) and SLEPc (on the right)

The new functionality provided by the SLEPc library is organised around two objects, Eigensolvers EPS and Spectral transform ST, as shown in the diagram of Figure 1.

3.1.1. EPS: Eigenvalue Problem Solver

The Eigenvalue Problem Solver (EPS) is the main object provided by SLEPc. It is used to specify an eigenvalue problem, either in standard or generalised form, and provides uniform and efficient access to all of the package's eigensolvers.

The EPS object provides functions for setting several parameters such as the number of eigenvalues to compute, the dimension of the subspace, the requested tolerance and the maximum number of iterations allowed. The user can also specify other things such as the orthogonalization technique or the portion of the spectrum of interest.

The solution of the problem is obtained in two steps, basically. First, the matrices associated to the eigenproblem are specified via `EPSSetOperators`. Then, a call to `EPSSolve` is done which invokes the subroutine for the selected eigensolver. The solution method can be specified procedurally or via the command line. The currently available methods are the Power Iteration with deflation, the Rayleigh Quotient Iteration, the Subspace Iteration with non-Hermitian projection and locking, the Arnoldi method with explicit restart and deflation, and the Lanczos method with full reorthogonalization. In addition to these methods, there are also wrappers for external packages such as ARPACK.

3.1.2. ST: Spectral Transformation

The other main SLEPc object is the Spectral Transformation (ST), which encapsulates the functionality required for acceleration techniques based on the transformation of the spectrum. The user does not usually need to create an ST object explicitly. Instead, every EPS object internally sets up an associated ST.

One of the design cornerstones of SLEPc is to separate acceleration techniques from solution methods so

that any combination of them can be specified by the user. To achieve this, all the eigensolvers contained in EPS are implemented in a way that they are independent of which transformation has been selected by the user. That is, the solver algorithm works with a generic operator, whose actual form depends on the transformation used. With this mechanism, the shift-and-invert technique can be applied easily, as well as other similar transformations.

4. IMPLEMENTATION DETAILS

Methods implemented in SLEPc merely require vector operations and matrix-vector products. In PETSc, mathematical objects such as vectors and matrices are defined in a uniform fashion without making unnecessary assumptions about internal representation. This implies that SLEPc can be used with any of the matrix and vector representations provided by PETSc. In many applications, the matrices that define the eigenvalue problem are not available explicitly. Instead, the user knows a way of applying these matrices to a vector. These cases can be easily handled in SLEPc by means of shell matrices. These are matrices which do not require explicit storage of the component values. All matrices in PETSc are derived from a virtual matrix base class but not all the available operations need to be defined. Instead, the user must provide subroutines for all the necessary matrix operations, typically only the application of the linear operator to a vector. Once the new matrix has been defined, it can be used by SLEPc as a regular matrix.

The following C source code implements the solution of a simple standard eigenvalue problem.

```
1  #include "slepceps.h"
2  Vec        *x;           /* basis vectors */
3  Mat        A;            /* operator matrix */
4  EPS        eps;          /* eigenproblem solver context */
5  PetscReal  *error;
6  Scalar     *kr,*ki;
7  int        its, nconv;
8  EPSCreate( PETSC_COMM_WORLD, &eps );
9  EPSSetOperators( eps, A, PETSC_NULL );
10 EPSSetFromOptions( eps );
11 EPSSolve( eps, &its );
12 EPSGetConverged( eps, &nconv );
13 EPSGetSolution( eps, &kr, &ki, &x );
14 PetscMalloc( nconv*sizeof(PetscReal), &error );
15 EPSComputeError( eps, error );
16 EPSDestroy( eps );
```

All the operations of the program are done over a single EPS object, the eigenvalue problem solver. This object is created in line 8. In line 9, the operator of the eigenproblem is specified to be the object A of type Mat, which will be a user-defined shell matrix in this case. Line 11 launches the solution algorithm. The subroutine which is actually invoked depends on which solver has been selected by the user. At the end, the number of iterations carried out by the solver is returned in its and all the data associated to the solution of the eigenproblem is kept internally. Line 12 queries how many eigenpairs have converged to working precision and the solution of the eigenproblem is retrieved in line 13.

The command line for executing the program could be the following

```
$ program -eps_nev 10 -eps_ncv 24
```

where the number of eigenvalues and the dimension of the subspace have been specified. In this other example, the solution method is given explicitly and the matrix is shifted with $\sigma = 0.5$

```
$ program -eps_type subspace -st_type shift -st_shift 0.5
          -eps_mgs_orthog -eps_monitor
```

The last two keys select the modified Gram-Schmidt orthogonalization algorithm and instruct to activate the convergence monitor, respectively.

In order to use the above code to solve the Lambda Modes problem, a shell matrix must be defined which represents matrix A in equation (7). First of all, a structure is defined as a placeholder for all the relevant information:

```
1  typedef struct {
2    SLES       L11, L22;
3    Vec        w, L21, M11, M12;
4  } CTX_LAMBDA;
```

Here, L_{11} and L_{22} are represented as linear systems and the other blocks are stored as vectors since they are diagonal matrices. `w` is a work vector for intermediate results. In order to use this matrix with the EPS object, at least one operation must be defined, the matrix-vector multiplication, which can be accomplished with the following function:

```
5  int Lambda_Mult( Mat A, Vec x, Vec y )
6  {
7    CTX_LAMBDA *ctx;
8    int        its;
9    Scalar     done = 1.0;
10   MatShellGetContext( A, (void**)&ctx );
11   VecPointwiseMult( ctx->L21, x, ctx->w );
12   SLESSolve( ctx->L22, ctx->w, y, &its );
13   VecPointwiseMult( ctx->M12, y, ctx->w );
14   VecPointwiseMult( ctx->M11, x, y );
15   VecAXPY( &done, y, ctx->w );
16   SLESSolve( ctx->L11, ctx->w, y, &its );
17   PetscFunctionReturn(0);
18 }
```

Finally, the `Mat` object for matrix A can be built with the following lines of code,

```
MatCreateShell( comm, n, n, N, N, (void*)ctx, A );
MatShellSetOperation( *A, MATOP_MULT, (void(*)())Lambda_Mult );
```

which should be placed between lines 7 and 8 of the basic code shown in page 5.

5. COMPUTATIONAL RESULTS

This section provides a summary of the results obtained when using the solver code in the analysis of a commercial reactor. The experiments have been carried out in a cluster of PC's, with Pentium III 866 MHz processors, 512 Mbytes of memory each, and interconnected by a Gigabit Ethernet network.

5.1. Description of Peach Bottom-2 Power Plant

The boiling water reactor which has been taken as a reference for testing the solver is the Peach Bottom-2 BWR/4 power plant, located in Pennsylvania (USA). The information has been obtained from the reports provided by the EPRI (Electric Power Research Institute) and other additional sources such as PECo (Philadelphia Electric Company), [4].

The exercise consists in coupling the kinetics and thermohydraulics in 3D for the model of the reactor core using the input and output boundary conditions provided by the University of Pennsylvania. The chosen problem is the analysis of a firing of turbine triggered by the locking of the turbine stop valve. The pressure oscillation generated in the main steam pipes propagates inside the reactor core with a small damping. The induced pressure oscillation makes large changes in the hole distribution and the fluid flux in the core. The magnitude of the neutron flux transient which takes place in the core of a BWR is highly influenced by the initial increase of pressure and it has a high spatial variation.

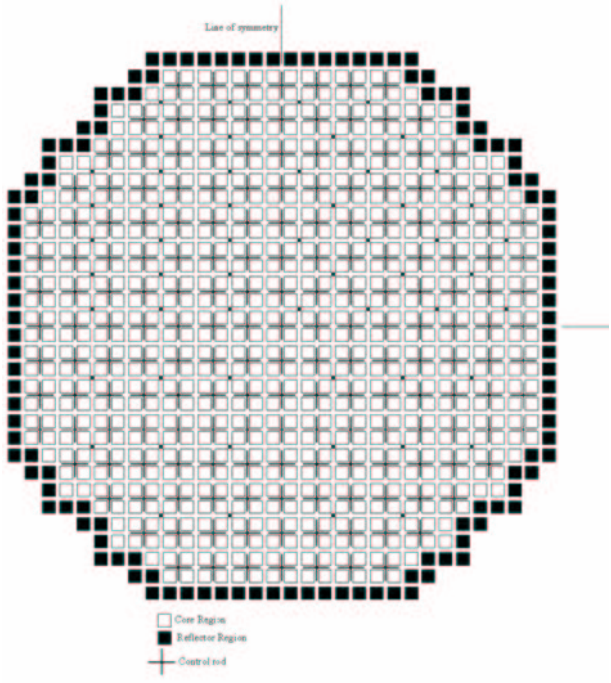

Figure 2. Nodalization of the Reactor Core.

Figure 2 shows the division in cells of the reactor core used in the analysis.

When enforcing the continuity conditions from one cell to its neighbour left and right discontinuity factors have been taken into account. This property is reflected at algebraic level in that matrices L_{11} and L_{22} of equation (7), in the general case, are non-symmetric. In this benchmark, the dimension of the associated matrices is 92352 and the number of non-zero elements is 1031424.

5.2 Analysis of Linear Systems of Equations

When performing a matrix-vector product with matrix A as defined in equation (7), two linear systems of equations must be solved. Therefore, the fastest technique for solving these systems must be sought. Table I shows some timing results for the solution of the L_{11} system for different iterative methods and preconditioners. For each case, the table displays the method used, the number of iterations carried out by the method, iter, the residual norm at the end of process, the total computation time (in seconds), ttotal, the set-up time, tsetup, and the time needed to achieve the solution, tsolve.

Table I. Evaluation of Several Combinations of Iterative Method and Preconditioner for the Solution of the Linear Systems of Equations.

precon	method	iter	norm	ttotal	tsetup	tsolve
jacobi	cg	35	2.28E-06	2.57	0.01	2.57
	bicg	35	2.28E-06	4.95	0.01	4.94
	gmres	35	2.59E-06	4.41	0.01	4.40
	bcgs	25	1.97E-06	3.68	0.02	3.67
	tfqmr	27	9.90E-07	4.39	0.03	4.37
ilu (levels=0, order=natural)	cg	13	1.38E-06	2.19	0.36	1.83
	bicg	13	1.38E-06	4.11	0.36	3.74
	gmres	13	1.59E-06	2.58	0.36	2.22
	bcgs	8	6.34E-07	2.70	0.37	2.34
	tfqmr	7	2.40E-06	2.52	0.37	2.15
ilu (levels=1, order=natural)	cg	8	2.39E-07	3.20	1.58	1.62
	bicg	8	2.39E-07	5.02	1.59	3.43
	gmres	7	4.43E-06	3.22	1.59	1.64
	bcgs	4	1.22E-06	3.35	1.59	1.76
	tfqmr	4	6.79E-07	3.40	1.60	1.80

In general, the Conjugate Gradient method (*cg*) is the fastest for this particular coefficient matrix, and therefore it has been used for subsequent tests. However, since matrices L_{11} and L_{22} are non-symmetric, in general, conjugate Gradient is not guaranteed to converge. Another conclusion is that the ILU preconditioner reduces the number of iterations considerably. It should be noted that with PETSc it is easy to organise the code so that the ILU factorisation can be reused and therefore it needs to be computed only once at the beginning of the program.

With respect to performance of parallel preconditioners, table II shows the number of processors, np, the number of iterations, iter, and some timings, the speed-up, S_p and the efficiency, E_p, for Jacobi preconditioning and Block Jacobi with ILU(0) in each block. Note that speedup and efficiency are relative to the total time. Jacobi results in better efficiency and this is due to the fact that with Block Jacobi the number of iterations of the solver depends on the number of processors.

Table II. Parallel Solution of Linear Systems.

	np	iter	ttotal	tsetup	tsolve	S_p	E_p
Jacobi	1	35	2.51	0.01	2.5	1.00	100 %
	2	35	1.43	0.00	1.43	1.76	88 %
	4	35	0.87	0.00	0.87	2.89	72 %
	8	35	0.56	0.00	0.56	4.48	56 %
block-Jacobi	1	13	2.19	0.36	1.83	1.00	100 %
	2	16	1.38	0.17	1.21	1.59	79 %
	4	19	0.88	0.09	0.8	2.49	62 %
	8	19	0.52	0.06	0.46	4.21	53 %

5.3 Performance of the Eigensolver

Several numerical tests have been carried out solving the complete problem associated to the Peach Bottom test case. Different eigensolvers have been tested, in particular the Power method, Subspace Iteration, Arnoldi and also the ARPACK wrapper. The method implemented in ARPACK is the most competitive and therefore this section only presents results corresponding to this solver.

Table III shows the number of processors, np, the number of iterations, iter, the computational time, T_p, the speed-up, S_p and the efficiency, E_p, obtained when only one dominant eigenpair is requested. The performance obtained in the general process is very similar to that obtained in the linear systems of equations, since this is the most time consuming operation. The achieved efficiency is quite acceptable. Better efficiencies could be achieved with an appropriate partitioning of the underlying graph and renumbering of the unknowns accordingly. On the other hand, a more complete scalability study would require to increase the size of the problem when increasing the number of processors.

Table III. Parallel Solution of the Complete Eigenproblem.

	np	iter	T_p	S_p	E_p
Jacobi	1	38	269,39	1,00	100 %
	2	39	163,36	1,65	82 %
	4	39	99,3	2,71	68 %
	8	38	63,52	4,24	53 %
block-jacobi	1	38	222,65	1,00	100 %
	2	39	153,77	1,45	72 %
	4	39	89,82	2,48	62 %
	8	38	52,97	4,20	53 %

6. CONCLUSIONS

This paper presents a parallel code for the solution of the Lambda Modes equation, in which it is necessary to compute the dominant eigenpairs of a large sparse generalised eigenvalue problem. This eigenproblem can be reduced to a standard one in which the matrix is not available explicitly.

The implementation of the solver has been done using SLEPc, a new library for the solution of eigenvalue problems developed by the authors. SLEPc is based on PETSc and has been designed to be able to cope with problems arising in real applications by using well-known techniques such as shift-and-invert transformations and state-of-the-art methods. It offers a growing number of solution methods as well as interfaces to well-established eigenvalue packages.

The use of shell matrices allows easy formulation of complicated block-structured eigenproblems, which is the case for the Lambda Modes, and this is one of the claims for superiority of SLEPc compared to existing alternatives. The treatment of more complicated problems as the multigroup formulation with upscattering can be treated in a similar way to the problem treated in the paper modifying only the process of creating the matrix A and the multiplication matrix-vector procedure.

The implemented solution displays quite good parallel performance.

ACKNOWLEDGEMENTS

This work has been partially supported by the Spanish Ministry of Science and Technology and the Fondo Europeo de Desarrollo Regional (FEDER) under grant DPI 2001-2766-C01-C02-02.

REFERENCES

[1] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith, “PETSc Users Manual,” Technical Report ANL-95/11 - Revision 2.1.3, Argonne National Laboratory (2002).

[2] V. Hernández, J. E. Román, and V. Vidal, “SLEPc: Scalable Library for Eigenvalue Problem Computations,” *Proceedings of VECPAR 2002, 5th International Meeting on High Performance Computing for Computational Science*, (to appear in LNCS), Porto, Portugal (2002).

[3] R. B. Lehoucq, D. C. Sorensen, and C. Yang, ARPACK *Users’ Guide, Solution of Large-Scale Eigenvalue Problems by Implicitly Restarted Arnoldi Methods*, SIAM, Philadelphia, PA (1998).

[4] J. Solís et al., *Boiling Water Reactor Turbine Trip (TT) Benchmark*, NEA/NSC/DOC (2001).

[5] G. Verdú, J. L. Muñoz-Cobo, C. Pereira, and D. Ginestar, “Lambda Modes of the Neutron-Diffusion Equation. Application to BWRs Out-of-Phase Instabilities,” *Ann. Nucl. Energy*, 7(20):477–501 (1993).

[6] A. F. Henry, “Nuclear-Reactor Analysis”. The MIT Press, (1982).

[7] V. Vidal, J. Garayoa, G. Verdú and D. Ginestar, “Optimization of the Subspace Iteration Method for the Lambda Modes Determination of a Nuclear Power Reactor,” *J. of Nuclear Science and Tech.*, 34(9):929–947 (1997).

[8] W. M. Stacey Jr., *Space-Time Nuclear Reactor Kinetics*, Academic Press, NY (1969).