Parallel Resolution with Newton Algorithms of the Inverse Non-symmetric Eigenvalue Problem

Pedro V. Alberti^{1,2}, Victor M. García¹, and Antonio M. Vidal¹

¹ Universidad Politécnica de Valencia,
Camino de Vera s/n, Valencia España
{palberti, vmgarcia, a Vidal}@dsic.upv.es

² Universidad de Magallanes,
Av. Bulnes 01855, Punta Arenas Chile

Abstract. In this work, we describe two sequential algorithms to solve the Inverse Non-Symmetric Eigenvalue Problem. Both approaches have been based on Newton's iteration. The second one has been parallelized by using parallel standard public domain libraries. Good performance has been obtained in a cluster of PC's.

1 Introduction

An Inverse Eigenvalue Problem (*IEP*) is the reconstruction of a matrix from its spectrum, possibly forcing some structure characteristics in the resulting matrix. The goal of the work described in this paper is to solve a kind of inverse problem, the Inverse Additive Non-Symmetric Eigenvalue Problem (*IANSEP*). This problem appears in several areas of Science and Engineering, such as Seismic Tomography, Pole Assignment Problems, Geophysics and many more [1, 4, 6, 7, 10].

The state of the art of the problem (see, for example, [6]) shows that the vast majority of algorithms proposed are restricted to the symmetric case. Some authors like [6, 8] review the *IANSEP* in a quite general form, but we have not found any proposal for resolution of this problem; the present paper intends to fill this gap.

This paper presents two iterative algorithms based on Newton's method. A parallel version of the second one is also presented.

In section 2 a description of the problem is given, discussing the existence of solutions of the *IANSEP*. In section 3 a sequential resolution algorithm is proposed. It is based on a full statement of the problem as a nonlinear problem, and gives rise to large linear and nonlinear systems; we call it a 'Brute Force' approach. In section 4, another approach is taken, in which the nonlinear system is written only with the differences between the desired eigenvalues and the 'actual' ones. We call it a Restricted Newton method. In that case, the Jacobian matrix is approximated through finite differences. This algorithm is very suitable for parallel processing, and we describe as well the parallel implementation. Finally, some conclusions are given in section 5.

2 The *IANSEP*: Description and Existence of Solution

The problem can be stated as follows:

Given $n+1$ real, $n \times n$ matrices, $A_0, A_1, \dots, A_n$ and a set of real numbers $\lambda_1^* \leq \lambda_2^* \leq \dots \leq \lambda_n^*$, find $d \in \mathbb{R}^n$ such that the eigenvalues $\lambda_1(d) \leq \lambda_2(d) \leq \dots \leq \lambda_n(d)$ of

$$A(d) := A_0 + \sum_{k=1}^n d_k A_k \quad (1)$$

fulfill $\lambda_i(d) = \lambda_i^*$ for $i = 1, \dots, n$.

Of course, the main problem that arises if A_i are non-symmetric matrices is that the matrix $A_0 + \sum_{i=1}^n d_i A_i$ can have non real eigenvalues. In our work, we shall assume that the matrices A_i , the vector d and the desired eigenvalues λ_i^* are real.

This sets a double existence problem. First, it should be determined when the matrices $A(d)$ have real eigenvalues. Second, it should be determined whether given a vector, $\rho \in \mathbb{R}^n$ (the desired spectrum), $\exists d \in \mathbb{R}^n$ such that the spectrum of $A(d)$ ($\text{eig}(A(d))$) is ρ .

As far as we know, there are no theorem which can answer to these questions. The study of the 2×2 case can help us to determine under which circumstances will the algorithms work (or fail). A similar analysis can be found in [5] for the symmetric case.

Let us consider $A_0, A_1, \dots, A_n \in \mathbb{R}^{2 \times 2}$ and $d \in \mathbb{R}^2$, and let us try to find out how are the eigenvalues of $A(d) = A_0 + \sum_{i=1}^n d_i A_i$.

In this case

$$A_0 + \sum_{i=1}^n d_i A_i = \begin{bmatrix} \alpha(d) & \beta(d) \\ \gamma(d) & \delta(d) \end{bmatrix}, \quad (2)$$

and, therefore, the eigenvalues of $A(d)$ are the solutions of the equation $\lambda^2 - S(d)\lambda + \det(d) = 0$, with $S(d) = \alpha(d) + \delta(d)$ and $\det(d) = \alpha(d)\delta(d) - \gamma(d)\beta(d)$.

The eigenvalues of $A(d)$ shall be either real or complex depending on the value of the discriminant: $\Delta(d) = (\alpha(d) - \delta(d))^2 + 4\beta(d)\gamma(d)$.

If $\Delta(d) < 0$, $\forall d$, then the matrix $A(d)$ will always have non real eigenvalues, which means that our problem (that is, to find $d \in \mathbb{R}^2$ such that $\text{eig}(A(d)) = \rho \in \mathbb{R}^2$) does not have 'real' solution. Trivial numerical examples can be formed but we have avoided them for the sake of simplicity.

If $\Delta(d) \geq 0$, $\forall d$, then the matrix $A(d)$ always have real eigenvalues, which means that there can exist a 'real' solution of the problem.

Finally, it can happen that $\Delta(d) < 0$ for some d and $\Delta(d) \geq 0$ for other values of d . This means that the problem can have solution, but during the iterative search the algorithm may enter regions where $\Delta(d) < 0$. This should be avoided or treated with special care.

If we consider now the case in which $\Delta(d) \geq 0$ (and therefore it is possible a real solution) we might ask whether given a $\rho^* \in \mathbb{R}^2$, there exists always a $d \in \mathbb{R}^2$ such that $\text{eig}(A(d)) = \rho^*$. Clearly, this will not happen always, for example, in the case where $\Delta(d) < 0$.

Let us try to express d as a function of λ^* . Let us set the system of equations:

$$\begin{cases} \alpha(d)\delta(d) = \lambda_1^* + \lambda_2^* \\ \alpha(d)\delta(d) - \gamma(d)\beta(d) = \lambda_1^* \lambda_2^* \end{cases}, \quad (3)$$

which is equivalent (in the 2×2 case) to force that $\text{eig}(A(d))$ are λ_1^* , λ_2^* , and let us solve it for d :

$$\begin{aligned} d_2 &= \epsilon_1 d_1 + \epsilon_2 \\ \theta_1 d_1^2 + \theta_2 d_1 + \theta_3 &= 0, \end{aligned} \quad (4)$$

being $\theta_1, \theta_2, \theta_3, \epsilon_1, \epsilon_2$ functions of the entries of matrices A_0, A_1, A_2 and of λ_1^*, λ_2^* which can be easily formed. Clearly, if $\theta_2^2 - 4\theta_1\theta_3 < 0$ the problem does not have 'real' solution.

This shows again that the real solution of *IANSEP* does not always exist.

The results found can be extrapolated to $n \times n$ matrices. Therefore, the situation when $A_i \in \mathbb{R}^{n \times n}$ and, for a given set of given eigenvalues $\rho^* \in \mathbb{R}^n$ it is desired to find $d \in \mathbb{R}^n$ such that $\text{eig}(A(d)) = \rho^*$, is the following:

It may happen that for all $d \in \mathbb{R}^n$ the matrix $A(d)$ has non real eigenvalues. In this case the problem will not have real solution.

It also may happen that for all $d \in \mathbb{R}^n$ all the eigenvalues of $A(d)$ are real. In this case, it might happen that for some $\rho^* \in \mathbb{R}^n$ does not exist any $d \in \mathbb{R}^n$ such that $\text{eig}(A(d)) = \rho^*$.

Finally, it can happen that the eigenvalues of $A(d)$, are real for some $d \in \mathbb{R}^n$ and complex for other values of $d \in \mathbb{R}^n$.

This leads us to consider the working hypothesis of the algorithms for *IANSEP*, in the same way in which it is considered for the resolution of nonlinear system. Let us suppose that our problem, *IANSEP*, has a solution $d^* \in \mathbb{R}^n$. From the properties of continuity of eigenvalues it can be stated that there exists a (small enough) region comprising $d^* \in \mathbb{R}^n$ in which the eigenvalues of any matrix of the form $A_0 + \sum_{i=1}^n d_i A_i$ are also real. If a Newton-like method is used to find d^* , the initial guess should be taken inside this region; since Newton methods are locally convergent, if the region is small enough, quadratic convergence can be achieved.

3 'Brute Force' Method

As a first approach, we will try to rewrite the *IANSEP* as a system of nonlinear equations.

If the system has real solution $\rho^* = (\lambda_1^*, \lambda_2^* \dots, \lambda_n^*)$ must satisfy

$$A_0 + \sum_{i=1}^n d_i A_i = Q \tilde{T} Q^T, \quad (5)$$

where the elements of main diagonal of the upper triangular matrix $\tilde{T}$ are $(\lambda_1^* \leq \lambda_2^* \leq \dots \leq \lambda_n^*)$ and

$$QQ^T = I \quad (6)$$

So, we can set up the nonlinear system $F(v) = 0$, with $n^2 + \frac{n(n+1)}{2}$ equations and unknowns:

$$F(v) = \begin{cases} q_i^T q_j - \delta_{ij} = 0 & \text{with } i = 1, 2, \dots, n \text{ and } j = i, \dots, n \\ (A_0 + \sum_{k=1}^n d_k A_k)_{ij} - q_i^T \tilde{T} q_j = 0 & \text{with } i = 1, 2, \dots, n \text{ and } j = 1, \dots, n, \end{cases}$$

whose unknowns are

$$v = [q_{1,1}, q_{2,1}, \dots, q_{n,1}, \dots, q_{n,2}, \dots, q_{n,n}, \\ t_{12}, t_{1,3}, \dots, t_{1,n}, t_{2,3}, \dots, t_{2,n}, \dots, t_{(n-1),n}, d_1, d_2, \dots, d_n].$$

This nonlinear system can be tackled with Newton's method, through the iteration $v^{(i+1)} = v^{(i)} - J^{-1}(v^{(i)})F(v^{(i)}) = v^{(i)} + s^{(i)}$, with $J(v^{(i)})s^{(i)} = -F(v^{(i)})$. The Jacobian matrix J has the form

$$J = [J_{ij}] = \left[\frac{\partial F_i(v^{(k)})}{\partial v_j^{(k)}} \right] = \begin{bmatrix} J_{1,1} & J_{1,2} & J_{1,3} \\ J_{2,1} & J_{2,2} & J_{2,3} \end{bmatrix}, \quad (7)$$

with:

$$J_{1,1} \in \mathbb{R}^{\frac{n(n+1)}{2} \times n} \text{ where } \frac{\partial F_i(v^{(k)})}{\partial v_j^{(k)}} = \delta_{i,s} q_{r,j} + \delta_{j,s} q_{r,i}.$$

$$J_{1,2} \in \mathbb{R}^{\frac{n(n+1)}{2} \times \frac{n(n-1)}{2}} \text{ where } \frac{\partial F_i(v^{(k)})}{\partial v_j^{(k)}} = 0.$$

$$J_{1,3} \in \mathbb{R}^{\frac{n(n+1)}{2} \times n} \text{ where } \frac{\partial F_i(v^{(k)})}{\partial v_j^{(k)}} = 0.$$

$$J_{2,1} \in \mathbb{R}^{n \times n} \text{ where}$$

$$\frac{\partial F_i(v^{(k)})}{\partial v_j^{(k)}} = -\lambda_s^* q_{j,s} \delta_{i,r} - \lambda_s^* q_{i,s} \delta_{j,r} - \delta_{j,r} \sum_{k=s+1}^n t_{s,k} q_{j,k} \\ - \sum_{z=1}^n q_{i,z} \left(\sum_{k=z+1}^n t_{z,k} \delta_{j,r} \delta_{k,s} \right).$$

$$J_{2,2} \in \mathbb{R}^{n \times \frac{n(n-1)}{2}} \text{ where } \frac{\partial F_i(v^{(k)})}{\partial v_j^{(k)}} = - \sum_{z=1}^n q_{i,z} \left(\sum_{k=z+1}^n q_{j,z} \delta_{r,z} \delta_{s,k} \right).$$

$$J_{2,3} \in \mathbb{R}^{n \times n} \text{ where } \frac{\partial F_i(v^{(k)})}{\partial v_j^{(k)}} = (A_r)_{i,j}.$$

The solutions of the system of equations $J(v^{(i)})s^{(i)} = -F(v^{(i)})$ has computational complexity of $O(n^6)$ which makes this algorithm very expensive. This algorithm, as other Newton-like algorithm in inverse problems, has serious problems of convergence unless the initial guess is close enough to the solution. To alleviate this problem, the algorithm has been implemented including a well-known globalization technique, Armijo's rule [9].

3.1 Experimental Results

The algorithm has been developed in C language, by using BLAS and LAPACK libraries [2]. Experimental results and validations have been carried out in a PC with Intel Xeon processor, at 2GHz. and 1GB. of RAM.

It has been tested using random matrices with known eigenvalues. The initial guess has been chosen by perturbing the solution, $v^{(0)} = v^* + \delta$, with a small $\delta = 0.1$. The execution time of the algorithm can be divided in two parts, the initialization phase and iterative phase; figure1 shows (for different problem sizes)

the time for the initial phase, the iteration time and the number of iterations to reach the solution.

The algorithm needs large memory resources, since it needs to create very large Jacobians matrices. As an example, for a problem of size of 210×210 the Jacobian matrix would have a size of 132405×132405 . As a consequence, it can be applied only to small problems

Fig. 1. Time in seconds for 'Brute Force' algorithm

4 Restricted Newton's Method

A different approach to the problem is to solve the nonlinear system

$$F_i(d^{(k)}) = \lambda_i(d^{(k)}) - \lambda_i^* = 0 \quad i = 1, \dots, n. \quad (8)$$

Newton's method can be applied as well to this system, through the iteration: $d^{(k+1)} = d^{(k)} - J(d^{(k)})^{-1} F(d^{(k)}) = d^{(k)} + s^{(k)}$ with $J(d^{(k)})s^{(k)} = -F(d^{(k)})$, where $d \in \mathbb{R}^n$, $J \in \mathbb{R}^{n \times n}$. A similar idea is used in [8] for the symmetric case. In this work, we have approximated the Jacobian matrix through Finite Differences:

$$J = \left[\frac{\partial F_i(d)}{\partial d_j} \right] \cong \left[\frac{F_i(d + e_j h) - F_i(d)}{h} \right] = \left[\frac{eig(A(d) + hA_j)_i - eig(A(d))_i}{h} \right] \quad (9)$$

The main cost of this algorithm is the computation of the Jacobian matrix, since it is needed to compute $(n+1)$ times the eigenvalues of an $n \times n$ matrix, that is, the complexity is $O(n^4)$. The resolution of the linear system $J(d^{(k)})s^{(k)} = -F(d^{(k)})$ has only a cost of $O(n^3)$. As in the former section, the algorithm was implemented using Armijo's rule.

4.1 Experimental Results: Sequential Implementation

Figure 2 is analogous to figure 1 in the former section, reflecting the costs of the start-up phase, of the iterating phase and the number of iterations.

Matrices and initial guesses were chosen as in the former section. It is quite clear that the execution times of this algorithm are lower than the Brute Force algorithm ones.

However, it can still be improved through parallel computing techniques; this is discussed in the next section.

Fig. 2. Restricted Newton: Execution times (s.) and number of iterations

4.2 Parallelization of the Restricted Newton Method

The parallelization was carried out with the standard ScaLAPACK [β] approach, that is, distributing in a 2-D processors mesh the data of the problem; in this case, the matrices $A_0, A_1, A_2, \dots, A_n$, the initial guess $d^{(0)} = [d_1, d_2, d_3, \dots, d_n]^T$ and the desired eigenvalues $\rho^* = [\lambda_1^*, \lambda_2^*, \lambda_3^*, \dots, \lambda_n^*]^T$.

A C language subroutine was written to create the Jacobian matrix:

$$J_{ij}^{(k)} = \frac{eig_i(A(d) + hA_j) - eig_i(A(d))}{h} \quad i = 1, 2, \dots, n; \quad j = 1, 2, \dots, n.$$

An obvious parallelization possibility was to assign each column of the Jacobian matrix (solution of a single eigenvalue problem) to each processor. The second option was to perform each different eigenvalue problem by using all the processors, making use of the SCALAPACK 2-D distribution. This second option was chosen to take full advantage of the parallel SCALAPACK routines.

All the other steps in the algorithm correspond to calls to BLACS, PBLAS or ScaLAPACK.

4.3 Numerical Experiments

The code was tested in a cluster of PC's with up to 7 processors. As before, the matrices have random values and have known eigenvalues, and the times of the initial part (T-ini) and of the iterative part (T-iter) have been recorded separately. So, the speed-up can be expressed as a function of the number of iterations k to reach convergence and of the number of processors, p :

$$Sp = \frac{T_{ini\ 1} + k T_{iter\ 1}}{T_{ini\ p} + k T_{iter\ p}}. \quad (10)$$

In figure 3 it is shown the execution time for several sizes, of both the initial part and the iterative part as a function of the number of processors. The speed-up as a function of the number of processors is shown in figure 4.

It can be noted the influence of the dimensions of the 2-D mesh used in the ScaLAPACK. Thus, for square logical 2-D meshes (i.e. $p = 4$) efficiency is optimum.

It can be seen that the algorithm works quite well for large problem sizes.

Fig. 3. Execution time of the initial phase (T-ini) and iterative phase (T-iter)

Fig. 4. Experimental Speed Up

5 Conclusions

In this paper, we have discussed the *IANSEP*, analyzed its peculiarities studying the 2×2 case, and proposed two solution methods. One of them has been parallelized, giving good results. The algorithms have been implemented using C language and standard libraries such as LAPACK, ScaLAPACK, so that the resulting codes are efficient and portable.

The study in section 2 shows that the Non-Symmetric problem is far more complex than the symmetric one, and that even if the matrix $A(d)$ has only real eigenvalues, the problem may not have any solution.

The 'Brute Force' approach is an attempt to state the problem directly, that for some problems (and thanks to the power of today's processors) can be successful, but which compares poorly with the other method, the restricted Newton method.

The method in section 3 is a good option for the solution of the *IANSEP*, working quite well both in its sequential and parallel versions.

Acknowledgement

This work has been supported by Spanish MYCT and FEDER under Grant TIC 2003-08238-C02-02.

References

1. Pedro V. Alberti and Antonio M. Vidal. Solución Paralela de Mínimos Cuadrados para el Problema Inverso Aditivo de Valores Propios. Technical Report DSIC-II/03/02, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022, Valencia España, 2002.
2. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. LAPACK User's Guide. *SIAM*, Second edition, 1995.
3. L. S. Blackford, J. Choi, A. Cleary, E. D. Acevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK User's Guide. *SIAM*, 1997.
4. D. L. Boyle and G. H. Golub. A survey of inverse eigenvalue problems. *Inv. Prob.*, 595-622, 1987.
5. Xuzho Chen and Moody T. Chu. On the least squares solution of inverse eigenvalue problems. *SIAM J. Numer. Anal.*, 33(6):2417-2430, 1996.
6. Moody T. Chu. Inverse eigenvalue problem. *SIAM Rev.*, 40(1):1-39, March 1998.
7. Hua Dai. A Numerical Method for Solving Inverse Eigenvalue Problem. Technical Report TR/PA/98/33, Nanjing 210016, People's Republic of China, 1998.
8. S. Friedland, J. Nocedal, and M. L. Overton. The formulation and analysis of numerical methods for inverse eigenvalue problems. *SIAM Anal. Vol 24, june, N3*, 24(5):634-667, June 1987.
9. C. T. Kelley. Iterative methods for linear and nonlinear equations. *SIAM Pub.*, 1995.
10. J. Peinado and A.M. Vidal. A new parrallel approach to the toeplitz inverse eigenproblem using the newton's method. *Lecture Notes in Computer Science. Springer-Verlag*, 1999.