Journal of Computational Mathematics, Vol.14, No.1, 1996, 23–31.

A SMALLEST SINGULAR VALUE METHOD FOR SOLVING INVERSE EIGENVALUE PROBLEMS *[1)]

S.F. Xu
(*Department of Mathematics, Peking University, Beijing*)

Abstract

Utilizing the properties of the smallest singular value of a matrix, we propose a new, efficient and reliable algorithm for solving nonsymmetric matrix inverse eigenvalue problems, and compare it with a known method. We also present numerical experiments which illustrate our results.

1. Introduction

Consider the following inverse eigenvalue problem:

Problem G. *Let $A(x) \in \mathbf{R}^{n\times n}$ be a real analytic matrix-valued function of $x \in \mathbf{R}^n$. Find a point $x^* \in \mathbf{R}^n$ such that the matrix $A(x^*)$ has a given spectral set $L = \{\lambda_1, \cdots, \lambda_n\}$. Here $\lambda_1, \cdots, \lambda_n$ are given complex numbers and closed under complex conjugation.*

This kind of problem arises often in various areas of applications (see Freidland et al.(1987) and references contained therein). The two special cases of Problem G, which are frequently encountered, are the following problems proposed by Downing and Householder(1956):

Problem A. *Let A be a given $n \times n$ real symmetric matrix, and $\lambda_1, \cdots \lambda_n$ be n given real numbers. Find an $x^* = (x_1^*, \cdots, x_n^*)^T \in \mathbf{R}^n$ such that the matrix $A + D(x^*)$ has eigenvalue $\lambda_1, \cdots \lambda_n$. Here $D(x^*) = diag(x_1^*, \cdots, x_n^*)$.*

Problem M. *Let A be a given $n \times n$ real symmetric positive definite matrix, and $\lambda_1, \cdots \lambda_n$ be n given positive numbers. Find an $x^* = (x_1^*, \cdots, x_n^*)^T \in \mathbf{R}^n$ with $x_i^* > 0$ such that the matrix $AD(x^*)$ has eigenvalue $\lambda_1, \cdots \lambda_n$.*

Problem A and M are known as the additive inverse eigenvalue problem and the multiplicative inverse eigenvalue problem, respectively.

There are large literature on conditions for existence and uniqueness of solutions to Problem G in many special cases (see Xu(1989) and references contained therein). Here, we will assume that Problem G has a solution and concentrate on how to compute it numerically.

* Received February 28, 1994.
[1)] China State Major Key Project for Basic Researches.

Although many numerical methods for solving various special cases of Problem G have been proposed, only one, which is proposed by Biegler-König(1981), can be applied to the nonsymmetric case of $A(x)$. In this paper we propose a new, efficient and reliable algorithm for solving Problem G, which is based on the properties of the smallest singular value of a matrix, therefore, called a smallest singular value method. This algorithm also has no restriction of symmetry, and is more efficient and reliable than Biegler-König's algorithm.

Notation. Throughout this paper we use the following notation. $\mathbf{R}^{m\times n}$ is the set of all $m \times n$ real matrices, and $\mathbf{C}^{m\times n}$ the set of all $m \times n$ complex matrices. $\mathbf{R}^n = \mathbf{R}^{n\times 1}$ and $\mathbf{C}^n = \mathbf{C}^{n\times 1}$. I is the $n \times n$ identity matrix. The superscript T and H are for transpose and conjugate transpose, respectively. $\det(A)$ and $\mathrm{tr}(A)$ denote the determinant and the trace of a matrix A, respectively. $\sigma_{\min}(A)$ denotes the smallest singular value of a matrix A. The norm $\|x\|$ stands for the usual Euclidean norm of vector x, and $\|x\|_\infty$ for the max-norm of vector x.

2. Formulation of the Numerical Methods

We will now describe two methods for solving Problem G in the case where the given eigenvalues are distinct. Assume there exists a solution x^* to Problem G. In some neighborhood of x^* we will first consider the following formulation of Problem G:

Formulation I. Solve the nonlinear system

$$f(x) = (f_1(x), \cdots, f_n(x))^T = 0, \tag{2.1}$$

where

$$f_i(x) = \sigma_{\min}(A(x) - \lambda_i I) \tag{2.2}$$

for $i = 1, 2, \cdots, n$.

In order to apply Newton's method to (2.1), we need the partial derivatives of $f(x)$ with respect to the $x_1, \cdots, x_n$. To calculate these derivatives we apply Sun's Theorem [9]:

Theorem. *Let $x \in \mathbf{R}^l$ and $B(x) \in \mathbf{C}^{m\times n}$. Suppose that Re(B(x)) and Im(B(x)) are real analytic matrix-valued function of x. If σ is a simple non-zero singular value of $B(x^{(0)})$, $v \in \mathbf{C}^n$ and $u \in \mathbf{C}^m$ are associated unit right and left singular vectors, respectively, then there exists a simple singular value $\sigma(x)$ of $B(x)$ which is a real analytic function of x in some neighborhood of $x^{(0)}$, and*

$$\sigma(x^{(0)}) = \sigma, \qquad \frac{\partial \sigma(x^{(0)})}{\partial x_j} = Re\Big(u^H \frac{\partial B(x^{(0)})}{\partial x_j} v\Big).$$

Utilizing the above theorem, if $f_i(x) \neq 0$, we get

$$\frac{\partial f_i(x)}{\partial x_j} = \mathrm{Re}\Big(u_i(x)^H \frac{\partial A(x)}{\partial x_j} v_i(x)\Big), \tag{2.3}$$

where $v_i(x),\ u_i(x) \in \mathbf{C}^n$ are the unit right and left singular vectors associated with the smallest singular value $\sigma_{\min}(A(x) - \lambda_i I)$ of $A(x) - \lambda_i I$, respectively. Thus the Jacobian of f is

$$J(x) = \left[\frac{\partial f_i(x)}{\partial x_j}\right],$$

where $\frac{\partial f_i(x)}{\partial x_j}$ are defined by (2.3), and one step of Newton's method is defined by

$$J(x^{(m)})(x^{(m+1)} - x^{(m)}) = -f(x^{(m)}). \tag{2.4}$$

To apply this method, $2n$ singular vectors have to be computed per step, which is very time consuming. Therefore, instead of computing these singular vectors at each step we may consider approximating them. One possibility is to use inverse iteration with a zero shift (see[3], [4], [5], and [10]). Let

$$\begin{aligned} B &= A(x) - \lambda_i I, \qquad & \sigma &= \sigma_{\min}(A(x) - \lambda_i I), \\ v &= v_i(x), & u &= u_i(x). \end{aligned}$$

Inverse iteration for $u,\ v$ and σ. Starting with an initial guess u_0, iterates until convergence:

1. $Bw = u_i$,
2. $v_{i+1} = w / \|w\|$,
3. $B^H y = v_{i+1}$,
4. $u_{i+1} = y / \|y\|$.

Denote the converged u_i by u, compute v and σ by

$$Bw = u, \qquad v = w / \|w\|, \qquad \sigma = \frac{1}{\|w\|}. \tag{2.5}$$

By the properties of inverse iteration, generally speaking, only one iteration is typically required to reduce an adequate approximate singular value and the associated singular vectors when x is very close to x^*

Besides, in order to reduce the amount of work per iteration, we may translate $A(x)$ to Hessenberg form first. Let U is the orthogonal matrix with

$$A(x) = UHU^T, \tag{2.6}$$

where H is an upper Hessenberg matrix, then

$$A(x) - \lambda_i I = U(H - \lambda_i I)U^T. \tag{2.7}$$

Thus, instead of using $A(x) - \lambda_i I$ to perform inverse iteration we may apply inverse iteration to $H - \lambda_i I$, which can reduce the amount of work per iteration from $O(n^5)$ to $O(n^4)$.

Now, summarizing the above discussion, we obtain the following algorithm for solving Problem G:

Algorithm 2.1.

Choose a starting point $x^{(0)} \in \mathbf{R}^n$ and a starting matrix $U_0 = [u_1^{(0)}, \cdots, u_n^{(0)}] \in \mathbf{C}^{n\times n}$.

For $m = 0,\ 1, \cdots$

(1). Compute $A(x^{(m)})$ and $\frac{\partial A(x^{(m)})}{\partial x_j}$, and

$$A := A(x^{(m)}), \quad A_j := \frac{\partial A(x^{(m)})}{\partial x_j}, \quad j = 1,\ 2, \cdots, n.$$

(2). Compute the factorization

$$A = UHU^T$$

where U is an orthogonal matrix and H an upper Hessenberg matrix.

(3). For $i = 1,\ 2, \cdots n$,

(3.1). Compute the factorization

$$H - \lambda_i I = QR,$$

where Q is an unitary matrix and R an upper triangular matrix.

(3.2). Solve the linear system

$$Ry = u_i^{(0)},$$

and compute

$$v = \frac{y}{\|y\|};$$

then solve the linear system

$$R^H w = v,$$

and compute

$$u = \frac{w}{\|w\|}.$$

(3.3).

$$f_i := \frac{1}{\|w\|}, \qquad u_i^{(0)} := u,$$
$$u := UQu, \qquad v := Uv.$$

(3.4). compute

$$J_{ij} = \mathrm{Re}(u^H A_j v), \quad j = 1,\ 2, \cdots, n.$$

(4). If $\max\{|f_1|,\ \cdots, |f_n|\}$ is sufficiently small, stop.

(5). Solve the linear system

$$Jy = -f,$$

where $J = [J_{ij}]$ and $f = (f_1, \cdots, f_n)^T$, and compute

$$x^{(m+1)} = x^{(m)} + y.$$

Let us now look at a different formulation of Problem G:

Formulation II. Solve the nonlinear system

$$g(x) = (g_1(x), \cdots, g_n(x))^T = 0, \tag{2.8}$$

where

$$g_i(x) = \det(A(x) - \lambda_i I) \tag{2.9}$$

for $i = 1,\ 2, \cdots, n$.

This formulation is the most natural one and has been used by Biegler-König (1981). Applied Newton's method to (2.8) and utilized Trace-Theorem of Davidenko, he proposed the following algorithm:

Algorithm 2.2.

Choose a starting point $x^{(0)} \in \mathbf{R}^n$.

For $m = 0,\ 1, \cdots$

(1). Compute $A(x^{(m)})$ and $\frac{\partial A(x^{(m)})}{\partial x_j}$, and

$$A := A(x^{(m)}), \quad A_j := \frac{\partial A(x^{(m)})}{\partial x_j}, \quad j = 1,\ 2, \cdots, n.$$

(2). Compute the factorization

$$A = UHU^T,$$

where U is an orthogonal matrix and H an upper Hessenberg matrix, and compute

$$B_j = U^T A_j U, \qquad j = 1, \cdots, n.$$

(3). For $i = 1,\ 2, \cdots n$,

(3.1). Compute the factorization

$$P(H - \lambda_i I) = LU,$$

where P is a permutation matrix, $L = [l_{ij}]$ is an unit lower triangular matrix with $|l_{ij}| \leq 1$ and $U = [u_{ij}]$ an upper triangular matrix; and compute

$$g_i = \prod_{i=1}^{n} u_{ii}.$$

(3.2). Solve the n linear systems

$$LUX_j = PB_j, \qquad j = 1, \cdots, n.$$

(3.3). compute

$$\widetilde{J}_{ij} = g_i \times \mathrm{tr}(X_j), \quad j = 1,\ 2, \cdots, n.$$

(4). If $\max\{|g_1|,\ \cdots, |g_n|\}$ is sufficiently small, stop.

(5). Solve the linear system

$$\widetilde{J}y = -g,$$

where $\widetilde{J} = [\widetilde{J}_{ij}]$ and $g = (g_1, \cdots, g_n)^T$, and compute

$$x^{(m+1)} = x^{(m)} + y.$$

Let us now compare computational requirements of Algorithm 2.1 and 2.2. Algorithm 2.1 only differs from Algorithm 2.2 in step 3. Algorithm 2.1 requires approximately n^4 multiplications to form the Jacobi matrix J in step 3, whereas Algorithm 2.2 requires approximately n^5 multiplications to form the Jacobi matrix $\widetilde{J}$ in step 3

Besides, our numerical experience indicates that Algorithm 2.1 almost always requires fewer iterations and that Algorithm 2.2 suffers more often from ill-conditioning. It therefore seems that Algorithm 2.1 is always to be preferred over Algorithm 2.2.

3. Numerical Examples

We have tested Algorithm 2.1 and 2.2 on various types of problems. In our experience, Algorithm 2.1 is more efficient and reliable, but Algorithm 2.2 encounters difficulties more often. Here, we present four examples to illustrate the local behavior of Algorithm 2.1. The tests were made on IBM PC/XT. Double precision arithmetic was used throughout.

Example 1.[14] This is an additive inverse eigenvalue problem. Here $n = 3$,

$$A(x) = \begin{pmatrix} x_1 & 1 & 0 \\ 1 & x_2 & 1 \\ 0 & 1 & x_3 \end{pmatrix} \qquad \lambda = (-2,\ 0,\ 2)^T.$$

Using Algorithm 2.1 with the starting point $x^{(0)} = (1.2,\ 0.01,\ -1.3)^T$, after 4 iterations we get a computing solution

$$x = (1.414213562,\ -.7267619040 \times 10^{-14},\ -1.414213562)^T.$$

comparing it with the exact solution

$$x^* = (\sqrt{2},\ 0,\ -\sqrt{2})^T,$$

we know that

$$\|x - x^*\|_\infty = .2420324074 \times 10^{-7}.$$

Table 3.1 displays the values of $\|f(x^{(m)})\|_\infty$.

Table 3.1

m	$\|f(x^{(m)})\|_\infty$
0	$.1453866481D+00$
1	$.7932893273D-02$
2	$.1517640384D-03$
3	$.3565542171D-07$
4	$.2998220101D-14$

Table 3.2

m	$\|f(x^{(m)})\|_\infty$
0	$.5956225156D+01$
1	$.9078626305D+00$
2	$.4975300682D-01$
3	$.8968967906D-03$
4	$.3157841046D-06$
5	$.3767194502D-13$

Example 2.[7] This is also an additive inverse eigenvalue problem. Here $n = 8$,

$$A(x) = A + \mathrm{diag}(x_1,\ \cdots,\ x_8),$$
$$\lambda = (10,\ 20,\ 30,\ 40,\ 50,\ 60,\ 70,\ 80)^T,$$

where

$$A = \begin{pmatrix} 0.0 & 4.0 & -1.0 & 1.0 & 1.0 & 5.0 & -1.0 & 1.0 \\ 4.0 & 0.0 & -1.0 & 2.0 & 1.0 & 4.0 & -1.0 & 2.0 \\ -1.0 & -1.0 & 0.0 & 3.0 & 1.0 & 3.0 & -1.0 & 3.0 \\ 1.0 & 2.0 & 3.0 & 0.0 & 1.0 & 2.0 & -1.0 & 4.0 \\ 1.0 & 1.0 & 1.0 & 1.0 & 0.0 & 1.0 & -1.0 & 5.0 \\ 5.0 & 4.0 & 3.0 & 2.0 & 1.0 & 0.0 & -1.0 & 6.0 \\ -1.0 & -1.0 & -1.0 & -1.0 & -1.0 & -1.0 & 0.0 & 7.0 \\ 1.0 & 2.0 & 3.0 & 4.0 & 5.0 & 6.0 & 7.0 & 0.0 \end{pmatrix}$$

Using Algorithm 2.1 with the starting point $x^{(0)} = \lambda$, after 5 iterations we get a computing solution

$$x = (11.90787610,\ 19.70552151,\ 30.54549819,\ 40.06265749,$$
$$51.58714029,\ 64.70213143,\ 70.17067582,\ 71.31849917)^T.$$

The residual values are given in Table 3.2.

Example 3.[1] Here $n = 5$,

$$A(x) = A + x_1A_1 + x_2A_2 + x_3A_3 + x_4A_4 + x_5A_5,$$
$$\lambda = (0,\ 1,\ 2,\ 3,\ 4)^T.$$

where

$$A = \begin{pmatrix} 2 & -0.08 & 0 & 0 & 0 \\ -0.03 & 2 & -0.08 & 0 & 0 \\ 0 & -0.03 & 2 & -0.08 & 0 \\ 0 & 0 & -0.03 & 2 & -0.08 \\ 0 & 0 & 0 & -0.03 & 2 \end{pmatrix}$$

and $A_i = r_i e_i^T,\ i = 1,\ 2,\ 3,\ 4,\ 5$, with

$$R = \sum_{i=1}^{5} r_i e_i^T = \begin{pmatrix} 1 & 0 & 0.01 & -0.02 & 0.03 \\ -0.03 & 1 & 0 & 0.01 & -0.02 \\ 0.02 & -0.03 & 1 & 0 & 0.01 \\ -0.01 & 0.02 & -0.03 & 1 & 0 \\ 0 & -0.01 & 0.02 & -0.03 & 1 \end{pmatrix}$$

Table 3.3 displays the computational results of Algorithm 2.1 with the starting point $x^{(0)} = (-2,\ -1,\ 0,\ 1,\ 2)^T$.

Table 3.3

m	$\|f(x^{(m)})\|_\infty$
0	$.4625594388D-02$
1	$.3079773599D-06$
2	$.1767177906D-14$
$x = (-2.002401944,\ -0.9979977295,$	
$0.002364089452, 1.002706273,\ 1.995329310)^T$	

Example 4. Here $n = 3$,

$$A(x) = A + x_1 A_1 + x_2 A_2 + x_3 A_3,$$
$$\lambda = (1,\ 2,\ 3)^T.$$

where

$$A = \begin{pmatrix} 0.66 & -0.42 & -0.34 \\ 2.94 & 0.33 & 4.09 \\ 0.1 & 0.48 & 2.96 \end{pmatrix}, \qquad A_1 = \begin{pmatrix} 1 & 0.1 & 0.02 \\ 0.1 & 0 & 0.01 \\ 0.02 & 0.03 & 1 \end{pmatrix},$$
$$A_2 = \begin{pmatrix} 0 & 0.01 & 0 \\ 0 & 1 & 0 \\ 0.05 & 0.01 & 0 \end{pmatrix}, \qquad A_3 = \begin{pmatrix} 0 & 0 & 0.01 \\ 0 & 1 & 0.01 \\ 0 & 0.06 & 1 \end{pmatrix}.$$

Using Algorithm 2.1 with the starting point $x^{(0)} = (-0.5,\ -0.05,\ 2.1)^T$, after 9 iterations we get a computing solution

$$x = (0.8902087281,\ 4.035945140,\ -1.883181298)^T.$$

The residual values are given in Table 3.4.

Table 3.4

m	$\|f(x^{(m)})\|_\infty$	m	$\|f(x^{(m)})\|_\infty$
0	$.8883111302D+00$	5	$.1599815846D-01$
1	$.3411482853D+00$	6	$.6326446527D-03$
2	$.6993713739D+00$	7	$.1314660311D-05$
3	$.2497254940D+00$	8	$.2657927133D-10$
4	$.8748208974D-01$	9	$.7655396217D-15$

These examples, and our overall numerical experience with Algorithm 2.1, indicate that quadratic convergence indeed occurs in practice.

References

[1] F.W. Biegler-König, A Newton iterative process for inverse eigenvalue problems, *Numer. Math.,* 37(1981), 349–354.

[2] Z. Bohte, Numerical solution of the inverse algebraic eigenvalue problem, *Comput. J.*, 10(1968), 385–388.

[3] T.F. Chan, Defflated decomposition of solutions of nearly singular systems, *SIAM J. Numer. Anal.*, 21(1984), 738–754.

[4] T.F. Chan, Rank revealing QR factorizations, *Linear Algebra Appl.*, 88/89(1987), 67–82.

[5] J. Demmel and W. Kahan, Accurate singular values of bidiagonal matrices, *SIAM J. Sci. Stat. Comput.*, 11(1990), 873–912.

[6] A.C. Downing and A.S. Householder , Some inverse characteristic value problems, *J. Assoc. Comput. Mach.,* 3(1956), 203–207.

[7] S. Friedland, J. Nocedal and M.L. Overton, The formulation and analysis of numerical methods for inverse eigenvalue problems, *SIAM J. Numer. Anal.*, 24(1987), 634–667

[8] J.G. Sun, Eigenvalues and eigenvectors of a matrix dependent on several parameters, *J. Comp. Math.*, 3(1985), 351–364.

[9] J.G. Sun, A notes on simple non-zero singular values, *J. Comp. Math.*, 6(1988), 258–266.

[10] S. Van Huffel, J. Vandewall and A. Haegemans, An efficient and reliable algorithm for computing the singular subspace of a matrix, associated with its smallest singular values, *J. Computational and Appl. Math.*, 19(1987), 313–333.

[11] J.Y. Wang and B.S. Garbow, A numerical method for solving inverse real symmetric eigenvalue problems, *SIAM J. Sci. Stat. Comput.*, 4(1983), 45–51.

[12] J.H. Wilkinson, The Algebraic Eigenvalue Problem, Oxford Univ. Press, 1965.

[13] S.F. Xu, On the qualitative theory and numerical methods for the classical inverse algebraic eigenvalue problems, Ph.D. dissertation, Computing Center, Academia Sinica, Beijing, China, 1989.

[14] S.F. Xu, A homotopy algorithm for solving inverse eigenvalue problems, *Numer. Math. Sinica*, 14(1992), 199–206.