数理解析研究所講究録
第 1638 巻 2009 年 169-176

# An Algorithm for the Nonlinear Eigenvalue Problem based on the Residue Theorem

Yusaku Yamamoto
Department of Computational Science and Engineering, Nagoya University

## 1 Introduction

Given an $n \times n$ matrix $A(\lambda)$ whose elements are analytical functions of a complex parameter $\lambda$, we consider the problem of finding the values of $\lambda$ for which a nonzero vector $\mathbf{x}$ that satisfies $A(\lambda)\mathbf{x} = \mathbf{0}$ exists. This is known as the nonlinear eigenvalue problem and the $\lambda$'s are called eigenvalues. The nonlinear eigenvalue problem has applications in various fields such as nonlinear elasticity, electronic structure calculation and theoretical fluid dynamics. In this paper, we focus on finding all the eigenvalues that lie in a specified region on the complex plane.

Conventional approaches for the nonlinear eigenvalue problem include multivariate Newton's method [9] and its modifications [8], Arnoldi method [4] and Jacobi-Davidson method [12]. However, Newton's method requires a good initial estimate both for the eigenvalue and the eigenvector for stable convergence. Arnoldi and Jacobi-Davidson methods are efficient for large sparse matrices, but in general, they cannot guarantee that all the eigenvalues in a specified region are obtained.

In our approach, we construct a complex function that has simple poles at the roots of the nonlinear eigenvalue problem and is analytical elsewhere. Then, by computing the complex moments through contour integration along the boundary of the specified region, we can locate the poles [7]. This method has the advantage that it can find all the eigenvalues in the region. Moreover, the computationally dominant part, the evaluation of the contour integral by numerical integration, has large-grain parallelism since function evaluation at each sample point can be done independently. Thus our algorithm is expected to achieve excellent speedup in virtually any parallel environments.

We implemented our method on the Fujitsu HPC2500 supercomputer. Numerical experiments using a model problem show that our method can actually find the eigenvalues with high accuracy. Also, our method achieved nearly linear speedup using up to 16 processors for matrices of order 500 to 2000.

## 2 The algorithm

Assume that $A(z)$ is an $n \times n$ matrix whose elements are analytical functions of a complex parameter $z$ and that we are interested in computing the (nonlinear) eigenvalues within a closed curve $\Gamma$ on the complex plane. In the following, we restrict ourselves to the case where $\Gamma$ is a circle. Now, let $f(z) = \det(A(z))$. Then $f(z)$ is an analytical function of $z$ and the eigenvalues are characterized as the zeroes of $f(z)$.

To solve $f(z) = 0$, we use the method by Kravanja et al. based on complex contour integral [7]. Assume that there are $m$ distinct roots of $f(z)$ within $\Gamma$ and define the complex moments by

$$\mu_p = \frac{1}{2\pi i}\oint_\Gamma z^p \frac{f'(z)}{f(z)} \quad (p = 0, 1, \ldots, 2m-1). \tag{1}$$

Then $\mu_p$ can be rewritten as

$$\mu_p = \sum_{k=1}^{m} \lambda_k^p \nu_k, \tag{2}$$

where $\lambda_1, \lambda_2, \ldots, \lambda_m$ are the distinct roots of $f(z) = 0$ within $\Gamma$ and $\nu_1, \nu_2, \ldots, \nu_m$ are their multiplicities [7]. To extract the information on $\{\lambda_j\}$ from $\{\mu_p\}$, we define the following matrices:

$$H_m = \begin{pmatrix} \mu_0 & \mu_1 & \cdots & \mu_{m-1} \\ \mu_1 & \mu_2 & \cdots & \mu_m \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{m-1} & \mu_m & \cdots & \mu_{2m-2} \end{pmatrix}, \quad H_m^< = \begin{pmatrix} \mu_1 & \mu_2 & \cdots & \mu_m \\ \mu_2 & \mu_3 & \cdots & \mu_{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_m & \mu_{m+1} & \cdots & \mu_{2m-1} \end{pmatrix}, \tag{3}$$

$$V_m = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \lambda_1 & \lambda_2 & \cdots & \lambda_m \\ \vdots & \vdots & \vdots & \vdots \\ \lambda_1^{m-1} & \lambda_2^{m-1} & \cdots & \lambda_m^{m-1} \end{pmatrix}, \tag{4}$$

$$D_m = \mathrm{diag}(\nu_1, \nu_2, \cdots, \nu_m), \quad \Lambda_m = \mathrm{diag}(\lambda_1, \lambda_2, \cdots, \lambda_m). \tag{5}$$

Then it is easy to see that $H_m = V_m D_m V_m^T$ and $H_m^< = V_m D_m \Lambda_m V_m^T$. Noting that $D_m$ and $V_m$ are nonsingular, we can conclude that $\lambda = \lambda_j$ for some $1 \le j \le m$ if and only if $\lambda$ is an eigenvalue of a matrix pencil $H_m^< - \lambda H_m$. Hence, by computing the complex moments by eq. (1), forming the two Hankel matrices $H_m$ and $H_m^<$ by eq. (3) and computing the eigenvalues of $H_m^< - \lambda H_m$, we can find the roots of $f(z)$ within $\Gamma$.

In our problem, $f(z) = \det(A(z))$ and it can be shown that

$$\frac{f'(z)}{f(z)} = \mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]. \tag{6}$$

Note that Sakurai and Sugiura propose a complex contour integral method for the linear eigenvalue problem $A\mathbf{x} = \lambda\mathbf{x}$ [10]. They use $\mathbf{u}(A - zI)^{-1}\mathbf{v}$, where $\mathbf{u}$ and $\mathbf{v}$ are random vectors, instead of $f'(z)/f(z)$. It is also possible to extend this approach to the nonlinear eigenvalue problem, at least when the eigenvalues of interest are simple, and active research is being conducted. See [3] and [13] for details.

In the numerical procedure, we use the trapezoidal rule with $K$ points to compute the complex moments $\{\mu_p\}$. Also, since we do not know $m$ a priori, we replace it with some estimate $M$, which hopefully satisfies $M \ge m$. In that case, the eigenvalues of $H_M^< - \lambda H_M$ contain spurious solutions of the nonlinear eigenvalue problem. To get rid of them, we compute the eigenvector $\mathbf{x}_j$ corresponding to the computed $\lambda_j$ by inverse iteration, evaluate the relative residual $\| A(\lambda_j)\mathbf{x}_j \|_\infty / (\| A(\lambda_j) \|_\infty \| \mathbf{x}_j \|_\infty)$, and discard $\lambda_j$ if the residual is large or $\lambda_j$ is outside $\Gamma$. This works well as we will see in the next section. We show the computational procedure as Algorithm 1.

```
[Algorithm1: Solution of the nonlinear eigenvalue problem]
⟨1⟩  Input $n$, $M$, $r$, $K$ and the procedure to compute $A(z)$
     for a complex number $z$.
⟨2⟩  $\omega_K = \exp\left(\frac{2\pi i}{K}\right)$
⟨3⟩  do $j = 0, K-1$
⟨4⟩      $s_j = r\omega_K^j$
⟨5⟩      $t_j = \mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]\Big|_{z=s_j}$.
⟨6⟩  end do
⟨7⟩  do $p = 0, 2M-1$
⟨8⟩      $\mu_p = \frac{r^{p+1}}{K}\sum_{j=0}^{K-1} t_j\omega_K^{(p+1)j}$
⟨9⟩  end do
⟨10⟩ Construct $H_M$ and $H_M^<$ from $\mu_0, \mu_1, \ldots, \mu_{2m-1}$.
⟨11⟩ Find the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_M$ of $H_m^< - \lambda H_m$.
⟨12⟩ Compute the eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M$ by inverse iteration.
⟨13⟩ Compute the relative residual
     $\| A(\lambda_i)\mathbf{x}_i \|_\infty / (\| A(\lambda_i) \|_\infty \| \mathbf{x}_i \|_\infty)$ for $i = 1, 2, \ldots, M$.
⟨14⟩ Output the pair $(\lambda_i, \mathbf{x}_i)$ as the true eigenvalue-eigenvector
     pair if the corresponding relative residual is small.
```

The computationally dominant part in the above algorithm is the computation of $\mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]$ for the $K$ integration points. Since the computation at each point can be done completely independently, this algorithm has large-grained parallelism. Hence, it should be able to achieve excellent speedup in virtually any parallel environments.

## 3 Numerical results

We implemented our algorithm using C and MPI and evaluated its performance on the Fujitsu HPC2500 supercomputer. In parallelizing the algorithm, we distributed the $K$ integration points evenly among $P$ processors and performed the computation of $\mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]$ at these points in parallel. Other parts of the algorithm, such as the computation of the eigenvalues of $H_M^< - \lambda H_M$, were executed on one processor. We used vendor-supplied optimized LAPACK routines to compute $(A(z))^{-1}\frac{dA(z)}{dz}$ and to find the eigenvalues of $H_M^< - \lambda H_M$. The test matrices used are of the form $A(z) = A - zI + \epsilon B(z)$, where $A$ is a real nonsymmetric matrix whose elements follow uniform random numbers in $[0, 1]$, $B(z)$ is an anti-diagonal matrix with anti-diagonal elements $e^z$, and $\epsilon$ is a parameter that determines the strength of nonlinearity. In our experiments, we set $K = 128$ and $M = 10$ and varied $n$ from 500 to 2000, $P$ from 1 to 16, and $\epsilon$ from 0 to 0.1. The center of $\Gamma$ is origin in all cases and the radius of $\Gamma$ is 0.7, 0.7 and 0.5 for the $n = 500$, 1000 and 2000 cases, respectively.

We show the results for the $n = 1000$ and $\epsilon = 0.01$ case in Fig. 1 and Table 1. In this case, it is known that the number of (nonlinear) eigenvalues in $\Gamma$ is 9. Our algorithm computed 10 candidates from the eigenvalues of $H_M^< - \lambda H_M$, discarded one of them (the open diamond) as the spurious solution since the residual was around $10^{-2}$, and output 9 solutions (the solid diamonds). The residuals for these nine solutions were all under $10^{-11}$. Hence, our algorithm succeeded in locating all the eigenvalues in $\Gamma$ with high accuracy. The situation was similar in other cases and our algorithm always found the correct number of eigenvalues in $\Gamma$ with high accuracy. We also show the execution time as a function of $n$ and $P$ in Fig. 2. It is clear that almost linear speedup is achieved for all the cases.

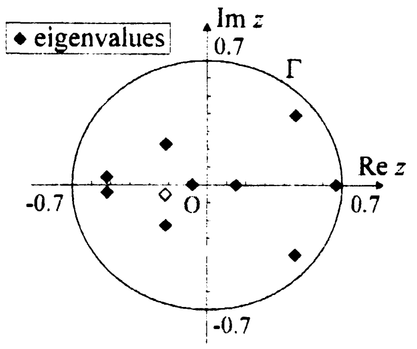

Figure 1: Computed eigenvalues for the $n = 1000$ and $\epsilon = 0.01$ case.

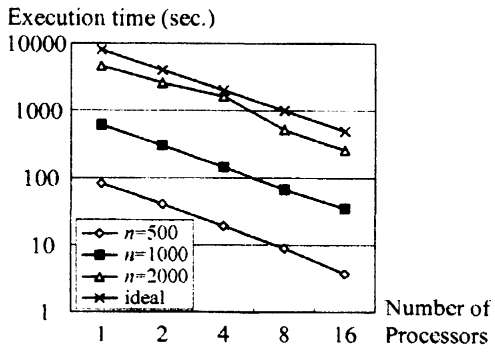

Figure 2: Execution time of our algorithm on Fujitsu HPC2500.

Table 1: Computed eigenvalues and the residuals for the $n = 1000$ and $\epsilon = 0.01$ case.

| No. | Eigenvalue | Residual |
|---|---|---|
| 1 | $+0.669065316 - 0.000000000i$ | $2.71 \times 10^{-14}$ |
| 2 | $+0.458199372 - 0.391494743i$ | $2.77 \times 10^{-14}$ |
| 3 | $+0.458199372 + 0.391494743i$ | $2.52 \times 10^{-14}$ |
| 4 | $-0.519233714 - 0.042547105i$ | $1.24 \times 10^{-12}$ |
| 5 | $-0.519233714 + 0.042547105i$ | $1.12 \times 10^{-12}$ |
| 6 | $-0.214208812 - 0.228114591i$ | $1.35 \times 10^{-12}$ |
| 7 | $-0.214208812 + 0.228114591i$ | $1.01 \times 10^{-12}$ |
| 8 | $-0.075282609 - 0.000000000i$ | $1.67 \times 10^{-11}$ |
| 9 | $+0.150003292 - 0.000000000i$ | $2.61 \times 10^{-12}$ |
| 10 | $-0.217773362 - 0.055395881i$ | $7.53 \times 10^{-3}$ |

Next we investigated the behavior of our algorithm when there are eigenvalues of multiplicity greater than one. Our test problem is a small $5 \times 5$ symmetric quadratic eigenvalue problem [11] for which $A(z)$ is given by

$$A(z) = \begin{bmatrix} -z^2-3z+1 & & & \mathit{sym.} & \\ z^2-1 & -2z^2-3z+5 & & & \\ -z^2-3z+1 & z^2-1 & -2z^2-5z+2 & & \\ -2z^2-6z+2 & 2z^2-2 & -4z & -9z^2-19z+14 & \end{bmatrix}. \quad (7)$$

This quadratic eigenvalue problem has four simple eigenvalues $-4 \pm \sqrt{18}$ and $-4 \pm \sqrt{19}$ and two eigenvalues 1 and $-2$ of multiplicity 2.

To solve this problem, we used as $\Gamma$ a circle centered at the origin and with radius 2.1. In this case, two simple eigenvalues and two multiple eigenvalues lie in $\Gamma$. In the computation, $K = 128$ and $M = 6$ and only one processor was used. The computed eigenvalues, along with the errors and residuals, are shown in Table 2. The multiplicity of each eigenvalue computed with the method of [1] is also shown in the table. From the value of the residual, eigenvalues 5 and 6 are judged to be spurious eigenvalues. So the number of computed true eigenvalues is four, which coincides with the actual number of eigenvalues in $\Gamma$. In addition, the computed multiplicity of each eigenvalue was correct. Thus we know that our algorithm can be applied to a problem with multiple eigenvalues.

Table 2: Computed eigenvalues and the residuals for the quadratic eigenvalue problem.

| No. | Eigenvalue | multiplicity | Error | Residual |
|---|---|---|---|---|
| 1 | $-2.000000000000 - 0.000000000000i$ | 2.00388725 | $0.3 \times 10^{-15}$ | $0.1 \times 10^{-15}$ |
| 2 | $+0.999999999999 - 0.000000000000i$ | 2.00000000 | $0.8 \times 10^{-13}$ | $0.6 \times 10^{-14}$ |
| 3 | $+0.242640687125 - 0.000000000002i$ | 1.00000000 | $0.7 \times 10^{-11}$ | $0.3 \times 10^{-11}$ |
| 4 | $+0.358898943542 - 0.000000000002i$ | 1.00000000 | $0.2 \times 10^{-11}$ | $0.3 \times 10^{-12}$ |
| 5 | $+0.111882174454 - 0.227062100814i$ | 0.00000000 | – | $0.1 \times 10^{-1}$ |
| 6 | $-1.349195617472 - 1.137587838458i$ | 0.00000000 | – | $0.2 \times 10^{-1}$ |

## 4 Efficient computation of $\mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]$

In our algorithm, the computationally dominant part is the evaluation of $\mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]$ at the $K$ integration points. In the implementation used in the previous section, we used LAPACK routines to compute this quantity because the test matrices were dense. More precisely, we first computed the LU decomposition of $A(z)$, computed $(A(z))^{-1}\frac{dA(z)}{dz}$ by repeating forward/backward substitution $n$ times with $\frac{dA(z)}{dz}$ as the right hand sides, and then summed up the diagonals of the resulting matrix to obtain the trace. This process requires $O(n^3)$ work for each integration point.

When the matrix is sparse, one can use sparse LU decomposition instead to reduce the computational work. However, the required work is still large. Let $\mathrm{Str}(L_{*i})$ and $\mathrm{Str}(U_{i*})$ be defined by

$$\begin{aligned} \mathrm{Str}(L_{*i}) &= \{j \mid j > i, l_{ji} \neq 0\} \quad \text{and} & (8)\\ \mathrm{Str}(U_{i*}) &= \{j \mid j > i, u_{ij} \neq 0\}, & (9) \end{aligned}$$

respectively [5]. Then the work to repeat forward/backward substitution $n$ times is

$$O\left(n\sum_{i=1}^{n}\{|\mathrm{Str}(L_{*i})| + |\mathrm{Str}(U_{i*})|\}\right). \tag{10}$$

To further reduce the work, a new efficient algorithm for computing $\mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]$ was proposed in [14]. Since this algorithm uses Erisman & Tinney's algorithm to compute partial elements of the inverse of a sparse matrix [6], we explain the latter algorithm first.

Assume that $A$ is a sparse matrix that admits LU decomposition and its $L$ and $U$ factors are given. Also, denote $A^{-1}$ by $C$. Then it can be shown that the elements of $C$

satisfy the following recursion formulae [14]:

$$c_{ip} = -\frac{1}{u_{ii}} \sum_{k \in \mathrm{Str}(U_{i*})} u_{ik} c_{kp}, \tag{11}$$

$$c_{pi} = -\frac{1}{l_{ii}} \sum_{k \in \mathrm{Str}(L_{*i})} c_{pk} l_{ki}, \tag{12}$$

$$c_{ii} = -\frac{1}{u_{ii}} \sum_{k \in \mathrm{Str}(U_{i*})} u_{ik} c_{ki} + \frac{1}{l_{ii} u_{ii}}, \tag{13}$$

where $1 \leq i \leq p \leq n$. Using these equations, Erisman & Tinney's algorithm can be described as Algorithm 2 below.

```
[Algorithm2: Erisman & Tinney's algorithm]
⟨1⟩  $c_{nn} = 1/(l_{nn} u_{nn})$
⟨2⟩  do $i = n-1, 1, -1$
⟨3⟩     Compute $c_{ip}$ for $p \in \mathrm{Str}(L_{*i})$ using eq. (11).
⟨4⟩     Compute $c_{pi}$ for $p \in \mathrm{Str}(U_{i*})$ using eq. (12).
⟨5⟩     Compute $c_{ii}$ using eq. (13).
⟨6⟩  end do
```

After the completion of this algorithm, all the elements of $C = A^{-1}$ which are situated at the nonzero positions of $L^T + U^T$ are obtained. The computational work of this algorithm is

$$4 \sum_{i=1}^{n} \{ |\mathrm{Str}(L_{*i})| |\mathrm{Str}(U_{i*})| \}. \tag{14}$$

Note that this is much smaller than the work of Eq. (10), which would be required if all the elements of $A^{-1}$ are computed by repeated forward/backward substitution.

Now we show that these partial elements are sufficient to compute $\mathrm{Tr}\left[(A(z))^{-1} \frac{dA(z)}{dz}\right]$. In the following, we denote the elements of $\frac{dA(z)}{dz}$ by $a'_{ij}$. First, notice that

$$\mathrm{Tr}\left[(A(z))^{-1} \frac{dA(z)}{dz}\right] = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ji} a'_{ij} = \sum_{\{i,j | a'_{ij} \neq 0\}} c_{ij} a'_{ij}. \tag{15}$$

Hence only those elements of $C = (A(z))^{-1}$ which correspond to the nonzero elements of $\left(\frac{dA(z)}{dz}\right)^T$ are needed. Then we use the following relationship between the nonzero positions of the related matrices:

$$\begin{aligned} &\text{Nonzero positions of } \left(\frac{dA(z)}{dz}\right)^T \\ &\subseteq \text{Nonzero positions of } (A(z))^T \\ &\subseteq \text{Nonzero positions of } L^T + U^T \\ &= \text{Nonzero positions whose value are computed by Erisman \& Tinney's} \\ &\quad \text{algorithm.} \end{aligned} \tag{16}$$

From this relationship, it is clear that the partial elements of $(A(z))^{-1}$ obtained by Erisman & Tinney's algorithm are sufficient to compute $\mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]$. Since only diagonal elements of $(A(z))^{-1}\frac{dA(z)}{dz}$ need to be computed to evaluate the trace, the computational work required in addition to Eq. (14) is $O(\sum_{i=1}^{n}\{|\mathrm{Str}(L_{*i})| + |\mathrm{Str}(U_{i*})|\})$, which is negligible compared with Eq. (14).

We did some preliminary experiments to evaluate the efficiency of the new algorithm stated above. We used four test matrices shown in Table 3 and compared the execution times of the two algorithms to compute $\mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]$ under the condition that the sparse LU decompositions are given. The sparse LU decompositions were computed using the Approximate Minimum Degree algorithm [2]. The experiments were performed on a Xeon (2.66GHz) processor running on Red Hat Enterprise Linux.

Table 3: Test matrices used in the experiments.

| Matrix | 3dp16 | bcsstk13 | bcsstk15 | bcsstk16 |
|---|---|---|---|---|
| $n$ | 4096 | 2003 | 3948 | 4884 |

The execution times of the conventional algorithm based on repeated forward/backward substitution and the new algorithm are shown in Table 4. It can be seen that the new algorithm is 20 to 90 times faster than the conventional one. Thus we can conclude that the new algorithm is very effective in reducing the work reuired to compute $\mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]$. We are now trying to incorporate this algorithm into our nonlinear eigenvalue solver.

Table 4: Execution times of the new and conventional algorithms.

| Matrix | 3dp16 | bcsstk13 | bcsstk15 | bcsstk16 |
|---|---|---|---|---|
| Conventional (sec) | 38.18 | 16.41 | 56.85 | 43.61 |
| New (sec) | 0.82 | 0.74 | 1.81 | 0.50 |
| Speedup | 46.5 | 22.1 | 31.5 | 87.2 |

## 5 Conclusion

We proposed a new algorithm for the nonlinear eigenvalue problem $A(\lambda)\mathbf{x} = \mathbf{0}$. Our algorithm has the ability to find all eigenvalues within a specified region on the complex plane. Also, it is expected to achieve excellent parallel speedup thanks to the large-grained parallelism. These advantages have been confirmed by numerical experiments. We also introduced a new algorithm for computing $\mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]$ efficiently and showed its effectiveness by preliminary numerical experiments.

More numerical results of our algorithm are found in [1]. Detailed error analysis of the algorithm, along with the effect of employing a new algorithm for computing $\mathrm{Tr}\left[(A(z))^{-1}\frac{dA(z)}{dz}\right]$, will be given in our forthcoming paper.

## References

[1] T. Amako, Y. Yamamoto, and S.-L. Zhang. A large-grained parallel algorithm for nonlinear eigenvalue problems and its implementation using OmniRPC. In *Proceedings of the 2008 IEEE International Conference on Cluster Computing*, pages 42–49, 2008.

[2] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl*, 17:886–905, 1996.

[3] J. Asakura, T. Sakurai, H. Tadano, T. Ikegami, and K. Kimura. A numerical method for nonlinear eigenvalue problems using a contour integral (in Japanese). In *Proceedings of the Annual Meeting of JSIAM*, pages 133–134, 2008.

[4] T. Betcke and H. Voss. A Jacobi-Davidson-type projection method for nonlinear eigenvalue problems. *Future Generation Comput. Syst.*, 20:363–372, 2004.

[5] T. A. Davis. *Direct Methods for Sparse Linear Systems.* SIAM, 2006.

[6] A. M. Erisman and W. F. Tinney. On computing certain elements of the inverse of a sparse matrix. *Communications of the ACM*, 18:177–179, 1975.

[7] P. Kravanja, T. Sakurai, and M. Van Barel. On locating clusters of zeros of analytic functions. *BIT*, 39:646–682, 1999.

[8] A. Neumaier. Residual inverse iteration for the nonlinear eigenvalue problem. *SIAM J. Numer. Anal.*, 22:914–923, 1985.

[9] A. Ruhe. Algorithms for the nonlinear eigenvalue problem. *SIAM J. Numer. Anal.*, 10:674–689, 1973.

[10] T. Sakurai and H. Sugiura. A projection method for generalized eigenvalue problems. *J. Comput. Appl. Math.*, 159:119–128, 2003.

[11] D. S. Scott and R. C. Ward. Solving symmetric-definite quadratic problems without factorization. *SIAM J. Sci. Stat. Comput.*, 3:58–67, 1982.

[12] H. Voss. An Arnoldi method for nonlinear eigenvalue problems. *BIT Numerical Mathematics*, 44:387–401, 2004.

[13] Y. Yamamoto. A projection method for nonlinear eigenvalue problems based on complex contour integration. submitted.

[14] Y. Yamamoto. On computing the diagonals of the inverse of a general sparse matrix based on the LU decomposition (in Japanese). In *Proceedings of the Annual Meeting of JSIAM*, pages 119–120, 2008.