JOURNAL OF COMPUTATIONAL PHYSICS 6, 124–130 (1970)

Modification of Nesbet's Algorithm for the Iterative Evaluation of Eigenvalues and Eigenvectors of Large Matrices

In the investigation of the electronic structure of atoms and molecules by the configuration interaction method, it is necessary to determine the lowest eigenvalue, and corresponding eigenvector, of quite sizable Hermitian (generally real symmetric) matrices having a dominant main diagonal. In large-scale calculations of this kind, the order of the matrices may be anywhere from a few hundred to tens of thousands, the matrices cannot normally be accommodated in their entirety in the high-speed store of a computer, and the classical methods for matrix diagonalization are ineffective or impractical. Furthermore, those methods which modify the original matrix, such as the tridiagonalization schemes of Givens and of Householder (for detailed references see Wilkinson [1]), are prone to accumulate sizable round-off errors when applied to very large matrices, while iterative techniques which leave the matrix unchanged and do not propagate errors from one iteration to the next are much more reliable. Another consideration is the considerable sparsity of the matrices that occur in the configuration interaction calculations. Most available diagonalization methods will destroy this sparsity in the intermediate stages of the calculation, while a method which does not modify the original matrix can be organized to avoid storing and handling any zero elements throughout the process.

A very effective iterative method for this problem has been devised by Nesbet [2], and has been used by the present author and his co-workers for some time with considerable success. There is, however, one feature in Nesbet's algorithm which introduces some difficulty in its application to matrices which are too large to fit into the central store of the computer and introduces complications in attempts to take advantage of the sparsity of the matrix. The algorithm requires access to all the elements of the matrix, one by one (ordered by rows), for each iteration. As symmetric matrices are most effectively generated and stored as semimatrices, with their lower triangular part ordered by rows, going through a complete row involves also going down a column for that part of the row which is in the omitted triangle. If the matrix overflows the available central storage and is fetched in blocks from an external storage device (such as a tape) for each iteration, this would necessitate prior conversion to square form, which is a rather laborious process in itself and results in approximately doubling the amount of external storage that has to be accessed in each iteration. If we are dealing with a very sparse matrix, so that it is advantageous to omit zero elements (requiring appropriate identification of the remaining elements), then even if the nonzero part of the semimatrix does fit into the central store, there is no practical algorithm for locating elements down a specified column, as would be necessary if we are to avoid conversion to full-matrix form (with the corresponding doubling of required storage).

A modification of Nesbet's algorithm is proposed here, which makes it possible to proceed through the matrix, one row at a time, in its semimatrix (lower triangular) form. Each row (up through the diagonal element only) is brought in once and scanned twice in succession during each iteration, resulting in about the same amount of arithmetic as in the original algorithm but considerably less data handling. Sparse semimatrices are handled very easily since all processes are sequential in the rows and no column access is required, allowing a storage scheme which omits all zeros. The modified algorithm has been applied successfully to a considerable number of configuration interaction matrices of various sizes (see, e.g. [3–6]), including one of order 12,077 with about 2,000,000 nonzero elements in the semimatrix, and has generally converged to the lowest eigenvalue in about seven iterations. The number of iterations required does not seem to depend appreciably on the order of the matrix (the $n = 12{,}077$ case took nine iterations), and the total work involved in the solution is merely proportional to the number of nonzero elements in the semimatrix.

The Original Algorithm

Nesbet's algorithm as originally given [2] is meant to solve the generalized eigenvalue problem

$$\mathbf{H}\mathbf{c} = E\mathbf{S}\mathbf{c}, \tag{1}$$

where **H** is the given (real symmetric) matrix, **S** is a given metric (*overlap matrix*) which is real, symmetric, and positive definite (and often just the unit matrix), E is the desired eigenvalue, and **c** is the corresponding eigenvector. Starting with some initial guess for **c** (e.g., $c_\nu = 1$ for some nonzero component of **c**, preferably the dominant one, and $c_\mu = 0$ for all $\mu \neq \nu$), we compute an estimate for E from

$$D = \sum_{\mu,\lambda=1}^{n} c_\mu S_{\mu\lambda} c_\lambda , \tag{2}$$

$$N = \sum_{\mu,\lambda=1}^{n} c_\mu H_{\mu\lambda} c_\lambda , \tag{3}$$

and

$$E = N/D \tag{4}$$

(n being the order of the matrices). Then, keeping one non-zero component of $\mathbf{c}$ (preferably the dominant one) fixed, each of the other components c_μ is adjusted in turn according to

$$f_\mu = \sum_{\lambda=1}^{n} S_{\mu\lambda} c_\lambda , \tag{5}$$

$$\sigma_\mu = \sum_{\lambda=1}^{n} H_{\mu\lambda} c_\lambda - E f_\mu , \tag{6}$$

$$\Delta c_\mu = \sigma_\mu / (E S_{\mu\mu} - H_{\mu\mu}), \tag{7}$$

$$\Delta D = (2 f_\mu + S_{\mu\mu} \Delta c_\mu) \Delta c_\mu , \tag{8}$$

$$\Delta E = \sigma_\mu \Delta c_\mu / (D + \Delta D), \tag{9}$$

with c_μ, D, and E being incremented by Δc_μ, ΔD, and ΔE, respectively, at the end of this sequence before proceeding to the next component. A complete iteration consists of the repetition of these steps for all rows μ (except the row corresponding to the fixed component c_ν), and iterations are continued until the largest value of $| \Delta c_\mu |$ in one complete iteration is less than a specified criterion C. The relative error in $E - H_{\mathrm{av}}$, where H_{av} is an average diagonal element of $\mathbf{H}$, is then roughly proportional to C^2 (assuming that $\mathbf{c}$ is approximately normalized). It is not essential in this form of the algorithm that the rows be processed in any particular order, and in fact, it is not even necessary that a complete iteration on all the rows be carried out before returning to process any particular row again, though the program structure is simpler if no advantage is taken of this flexibility.

It should be noted that the quantities f_μ, σ_μ, Δc_μ, ΔD, and ΔE need not be kept from one row to the next (except that a record of the largest $| \Delta c_\mu |$ so far in the current iteration has to be maintained and updated as required), and the only array for which storage has to be provided, in addition to buffers for the current blocks of $\mathbf{H}$ and $\mathbf{S}$ (when these are in square matrix form), is the vector $\mathbf{c}$. The published algorithm includes an additional feature, whereby an iteration is cut short after any row λ whenever $| \Delta c_\lambda |$ is greater than a variable criterion C_A which is taken as a specified fraction (usually about one half) of the largest $| \Delta c_\mu |$ of the last complete iteration, and a new iteration is then started. In the experience of the present author, this feature was generally ineffective in reducing the number of complete iterations required and was abandoned.

In the special case in which $\mathbf{S}$ is a unit matrix, Eq. (2) is replaced by

$$D = \sum_{\lambda=1}^{n} c_\lambda^2, \tag{2'}$$

Eq. (5) is omitted, f_μ is replaced by c_μ in Eqs. (6) and (8), and $S_{\mu\mu}$ is omitted from Eqs. (7) and (8). In the opinion of the present author, it is usually more advantageous in linear variational calculations to set up the eigenvalue problem in an orthogonal basis ($\mathbf{S} = \mathbf{1}$) in the first place and to use the special form of the algorithm for this case, than to solve the general case of Eq. (1) directly. In addition to a reduction in total storage requirements and computation time (and in many cases an increase in the sparsity of **H**), this may also have a beneficial effect in reducing the sensitivity of the computed eigenvalue to inaccuracies in the matrix elements (see Delves [7], but this argument does not apply of course to the case where the variational problem is first set up in nonorthogonal form and **H** is only transformed to an orthogonal basis before solving the eigenvalue equation). An exception to these comments may occur when **S** is almost a unit matrix, possessing only small nondiagonal blocks along the main diagonal, when the use of the general (nonorthogonal) algorithm may be preferable.

The Modified Algorithm

In order to eliminate the need for matrix elements beyond the diagonal (i.e., for $\lambda > \mu$) in the sums of Eqs. (5) and (6), certain partial sums can be formed during each iteration, at the point where the required elements are available, and used in the following iteration. Thus, if we write

$$\sum_{\lambda=1}^{n} S_{\mu\lambda} c_\lambda = \sum_{\lambda=1}^{\mu} S_{\mu\lambda} c_\lambda + t_\mu , \tag{10}$$

and

$$\sum_{\lambda=1}^{n} H_{\mu\lambda} c_\lambda = \sum_{\lambda=1}^{\mu} H_{\mu\lambda} c_\lambda + u_\mu , \tag{11}$$

where (using the symmetric character of the matrices)

$$t_\mu = \sum_{\lambda=\mu+1}^{n} S_{\mu\lambda} c_\lambda = \sum_{\lambda=\mu+1}^{n} S_{\lambda\mu} c_\lambda \tag{12}$$

$$u_\mu = \sum_{\lambda=\mu+1}^{n} H_{\mu\lambda} c_\lambda = \sum_{\lambda=\mu+1}^{n} H_{\lambda\mu} c_\lambda , \tag{13}$$

we can calculate the contribution of row λ to all t_μ and u_μ for $\mu < \lambda$ when c_λ is updated, in preparation for the next iteration (any t_μ and u_μ which is recalculated in this manner is no longer needed in the current iteration). Unlike the situation in the original algorithm, this requires that the rows be processed in their natural sequence. Storage is now required for the vectors **t** and **u**, as well as for (at least) one complete row of **S** and **H**. In the orthogonal case **t** and **S** are of course unnecessary.

The modified algorithm can be described by the following sequence of steps (the reverse arrow $\leftarrow$ denotes replacement):

I. *Initialization*

1. Obtain an initial approximation for **c** (e.g., $c_\nu = 1$ for the dominant component and $c_\mu = 0$ for all $\mu \neq \nu$).
2. Set $V = W = 0$.
3. For $\mu = 1, 2,..., n$ in turn,
   (a) set $t_\mu = u_\mu = 0$,
   (b) obtain row μ ($S_{\mu\lambda}$ and $H_{\mu\lambda}$ for $\lambda = 1, 2,..., \mu$),
   (c) except when $\mu = 1$, for each $\lambda = 1, 2,..., \mu - 1$, set $t_\lambda \leftarrow t_\lambda + S_{\mu\lambda} c_\mu$ and $u_\lambda \leftarrow u_\lambda + H_{\mu\lambda} c_\mu$,
   (d) set $V \leftarrow V + S_{\mu\mu} c_\mu^2$ and $W \leftarrow W + H_{\mu\mu} c_\mu^2$.
4. Compute $D = 2\Sigma t_\mu c_\mu + V$, $N = 2\Sigma u_\mu c_\mu + W$, $E = N/D$.

II. *Iteration*

1. Set $\Delta c_{\max} = 0$.
2. For each $\mu = 1, 2,..., n$ [but steps (b)–(g) should be skipped for $\mu = \nu$, the fixed component],
   (a) obtain row μ,
   (b) compute $f_\mu = t_\mu + \sum_{\lambda=1}^{\mu} S_{\mu\lambda} c_\lambda$, $g_\mu = u_\mu + \sum_{\lambda=1}^{\mu} H_{\mu\lambda} c_\lambda$,
   (c) compute $\sigma_\mu = g_\mu - E f_\mu$,
   (d) compute $\Delta c_\mu = \sigma_\mu / (E S_{\mu\mu} - H_{\mu\mu})$,
   (e) set $D \leftarrow D + (2 f_\mu + S_{\mu\mu} \Delta c_\mu) \Delta c_\mu$,
   (f) set $E \leftarrow E + \sigma_\mu \Delta c_\mu / D$,
   (g) set $c_\mu \leftarrow c_\mu + \Delta c_\mu$ and $\Delta c_{\max} \leftarrow \max(\Delta c_{\max}, |\Delta c_\mu|)$,
   (h) set $t_\mu = u_\mu = 0$,
   (i) for each $\lambda = 1, 2,..., \mu - 1$, set $t_\lambda \leftarrow t_\lambda + S_{\mu\lambda} c_\mu$ and $u_\lambda \leftarrow u_\lambda + H_{\mu\lambda} c_\mu$.
3. If $\Delta c_{\max} > C$ (at the end of a complete iteration), repeat from step II.1.

If the word-length of the computer is rather short and there is some danger of accumulation of round-off errors in the continuing updating of D and E in steps II.2(e, f), then E should be recalculated after the last iteration (or in extreme cases after each iteration) from step I.4, with steps I.2 and I.3(d) repeated in the iteration stage. On the other hand, if the word-length is more than adequate, it would be permissible to eliminate step II.2(h) and at the same time replace c_μ by Δc_μ in step II.2(i); in this case, the whole of step II.2 will be skipped for $\mu = \nu$, and Nesbet's device of cutting short an iteration whenever a particular Δc_μ is large could be incorporated in this form of the algorithm if desired. The procedure would of course be simplified somewhat if we require that the fixed component of $\mathbf{c}$ be the first one ($\nu = 1$). In the special case of $\mathbf{S} = \mathbf{1}$, all steps involving $\mathbf{t}$ and $\mathbf{S}$ are omitted, while f_μ and $S_{\mu\mu}$ are replaced by c_μ and 1, respectively, throughout.

As previously mentioned, in order to handle sparse matrices efficiently it is advantageous to record and handle the nonzero elements only. These should be ordered by rows (for the lower-triangular semimatrix), and each element $H_{\mu\lambda}$ (or $S_{\mu\lambda}$) should be identified by its column index λ—the program can keep track of the row index μ automatically by stepping it up by 1 every time a diagonal element ($\lambda = \mu$) is passed (the diagonal elements are always assumed to be present). On computers with ample word length, and with convenient access to parts of complete words, the column index can be packed into the low-order part of the computer word containing the matrix element and considered part of the number for the purpose of arithmetic operations on the element. In steps II.2(b, i), instead of using a DO-type loop control, each packed element would be picked up in turn, its low-order part would be obtained to determine λ, and the element would be multiplied by the appropriate vector component and added to the appropriate partial sum. When such packing is not practical, a separate list of indices λ, paralleling the list of non-zero matrix elements (but made up of shorter words, if possible), can be used. The non-zero elements would not necessarily occur in the same positions in $\mathbf{S}$ and $\mathbf{H}$, and these two matrices should therefore be handled independently.

The computation time would of course vary with the details of the computer word length, instruction code, and overall speed. With a program written by Pipano [5] for the CDC 6400 (in FORTRAN except for index unpacking) for the case $\mathbf{S} = \mathbf{1}$, the central processor time required was about 0.1 msec per nonzero element of $\mathbf{H}$ per iteration (including external storage access), with about seven iterations needed in most cases.

Convergence

Let E and $\mathbf{c}$ be an exact solution of Eq. (1), and let $\mathbf{c}' = \mathbf{c} + \boldsymbol{\delta}$ be an approximation to $\mathbf{c}$, such that the error vector $\boldsymbol{\delta}$ is of order $\epsilon\mathbf{c}$. The error $\mathscr{E}$ in the corresponding eigenvalue approximation, $E' = E + \mathscr{E}$, as obtained from Eqs. (2)–(4), is easily shown to be

$$\mathscr{E} = \sum_{\mu,\lambda} \delta_\mu (H_{\mu\lambda} - E' S_{\mu\lambda})\, \delta_\lambda \Big/ \sum_{\mu,\lambda} c_\mu' S_{\mu\lambda} c_\lambda', \tag{14}$$

and is of order ϵ^2, as pointed out by Nesbet [2]. The correction Δc_μ obtained for the element c_μ' of $\mathbf{c}'$ by the application of Eqs. (5)–(7) can then be expressed as

$$\Delta c_\mu = -\delta_\mu - (H_{\mu\mu} - E'S_{\mu\mu})^{-1} \left\{ \sum_{\lambda(\neq\mu)} (H_{\mu\lambda} - E'S_{\mu\lambda})\, \delta_\lambda - \mathscr{E} \sum_\lambda S_{\mu\lambda} c_\lambda \right\}. \tag{15}$$

The first term on the right side of Eq. (15) represents the true correction needed to make $c_\mu' + \Delta c_\mu$ exactly equal to c_μ, while the second term represents the residual error. In order to keep this error small, it is necessary that the first term in the curly brackets, which couples the correction Δc_μ to the errors δ_λ in all the other vector components, be small compared to $H_{\mu\mu} - E'S_{\mu\mu}$. Good convergence therefore requires that the off-diagonal elements of the matrix $\mathbf{H} - E'\mathbf{S}$ be small compared with its diagonal elements, and particularly that E' (and thus E) be reasonably well separated from the values of $H_{\mu\mu}/S_{\mu\mu}$ for all $\mu \neq \nu$ (where c_ν is the component of $\mathbf{c}$ held fixed during the iterations).

It is thus seen that the algorithm can be expected to converge particularly well for the lowest and highest eigenvalues (which are outside the range of all $H_{\mu\mu}/S_{\mu\mu}$ values), and that the best choice for the value of ν, the index of the fixed component, is that for which $H_{\nu\nu}/S_{\nu\nu}$ is closest to the desired eigenvalue.

Acknowledgment

The author gratefully acknowledges valuable discussions with Dr. A. Pipano and Dr. C. F. Bender.

References

1. J. H. Wilkinson, "The Algebraic Eigenvalue Problem," Oxford University Press, London, 1965.
2. R. K. Nesbet, *J. Chem. Phys.* **43** (1965), 311.
3. A. Pipano and I. Shavitt, *Int. J. Quantum Chem.* **2** (1968), 741.
4. M. Rubinstein and I. Shavitt, *J. Chem. Phys.* **51** (1969), 2014.
5. A. Pipano, R. R. Gilman, and I. Shavitt, to be published.
6. C. F. Bender and E. R. Davidson, *Phys. Rev.* **183** (1969), 23.
7. L. M. Delves, *J. Computational Phys.* **3** (1968), 17.

Received: October 7, 1969

Isaiah Shavitt

Battelle Memorial Institute, 505 King Avenue, Columbus, Ohio 43201
and
Department of Chemistry, The Ohio State University, Columbus, Ohio 43210