MONSTER MATRICES: THEIR EIGENVALUES AND EIGENVECTORS

Ernest R. Davidson

Department Editor: William J. Thompson

Very large matrices are increasingly used in science and engineering, so practical numerical algorithms for determining their properties are continually being developed. It is becoming common to work with matrices having a dimension N of 1 000 000 or more, and therefore possibly containing as many as 10^{12} matrix elements, in applications such as solving networks of differential equations, determining the lowest vibrational modes of mechanical structures, and finding the eigenvalues and eigenvectors of quantum systems.

In many applications, constraint conditions (such as interconnection limitations in mechanics) make the matrix diagonal and sparse. Technically speaking, in a sparse matrix the number of nonzero elements is proportional to N rather than to N^2. In quantum mechanical applications to the many-body problem, the matrix is not sparse in this sense, but typically fewer than 5% of the matrix elements will be nonzero. This mitigates—but does not resolve—such problems as holding the nonzero elements in memory; for $N = 10^6$ at a sparsity of 1% there are still 10^{10} elements.

If the matrix is symmetric and real in addition to being sparse, a saving of a factor of 4 in storage is achieved, and real (rather than complex) arithmetic produces a speedup by a similar factor. Further, if eigenvalues and their eigenvectors are needed, only the lowest 50 or so of the N possible solutions may be of interest, corresponding to the lowest vibrational frequencies of a structure or to the lowest energy levels of a molecule or nucleus. Within these simplifications, a vast array of techniques has been developed for particular patterns of sparsity, as described (for example) in Refs. 1–3. Appendix D of Ref. 2 shows many pictures of sparse matrices.

Our aim in this column is to describe how large nonsparse matrices can be handled efficiently and to show that computer hardware—especially storage and its access—is a large determinant of the practicality of implementation. The method discussed here was originally developed in the context of the quantitative solution of the many-body problem in quantum mechanics, for which the Rayleigh–Ritz variational method[4] is frequently used to obtain an approximate expansion of the wave function in an arbitrary set of functions. When large sets of expansion functions are used, the computation is reduced to finding a few eigenvalues and eigenvectors of a large matrix.

Ernest R. Davidson is a professor of chemistry at Indiana University, Bloomington, IN 47405-4001. His main research interest is solution of the electronic structure many-body problem for molecules. He is a member of the editorial boards of the International Journal for Supercomputer Applications, the Journal of Computational Physics, Theoretica Chimica Acta, and the International Journal of Quantum Chemistry.

Monster Hamiltonian matrices

Just as hydrogen is ubiquitous in the universe and its electronic structure is the paradigm for quantum mechanics of atoms, so water encompasses the earth, and the electronic structure of H_2O is the testing ground for calculations of the electronic structure of molecules. For bound states the Hamiltonian matrix elements are both real and symmetric. In 1981, matrices for up to 256 473 configurations of the ten electrons in H_2O could be handled.[5] By 1990, dimensions of more than one billion were achieved[6] by using an extension of the Davidson–Liu algorithm described below.

Typically, the matrices that are encountered have many zero matrix elements, but these zeros are randonly distributed, and their number is proportional to the square of the matrix dimension. Consequently, sparse-matrix methods that rely on the bandedness of the matrix are of limited use, whereas the algorithms to be described apply just as well when the matrix is fully nonzero, albeit at the cost of storage and execution time.

The Davidson–Liu algorithm

We now outline an algorithm first described by the author,[7] modified by Liu[8] to reduce computational bottlenecks, and subsequently improved by Murray, Racine, and Davidson.[9] The matrix eigenvalue problem may be written as

$$\mathbf{H}\mathbf{c}_k = \lambda_k \mathbf{c}_k, \quad k = 1,\dots,K, \tag{1}$$

where the Hamiltonian matrix $\mathbf{H}$ is assumed to be real and symmetric, and the eigenvalues λ_k are numbered in increasing order. The number of eigenvalues desired K is generally less than 50, while the matrix dimension N may be very large. In the Raleigh–Ritz method for solving the Schrödinger equation, the eigenvalues of the matrix are upper bounds to the corresponding exact eigenvalues, and the vectors $\mathbf{c}_k$ give the coefficients in series expansions of the wave functions in the set of expansion functions.

For matrices of low dimension (such as $N < 150$), there are efficient methods for finding all the eigenvalues and eigenvectors. For example, the Householder–QL–Wilkinson modification[10] of the Givens method is built into the EISPACK routines[11] and is routinely used. The computation time for any of these methods grows as N^3 and the memory requirement grows as N^2. There is then little advantage to limiting K to be much less than N. In the algorithm to be described for large matrices, an efficient procedure for finding all the eigenvalues and eigenvectors of small submatrices is assumed to be available.

Clearly, matrices of dimension exceeding 10^5 cannot be stored in the central memory of most computers. If the number of nonzero elements does not exceed 10^8, it is usually possible to store the matrix on disk. For even larger matrices, the matrix cannot be stored, and its elements must be recomputed as needed. In any case, relatively few matrix elements are available at any time in central memory, and usually these are most conveniently produced in an order very different from any standard pattern for storing matrices. Consequently, the only matrix-arithmetic operation that is easily performed is a matrix–vector product. For example, given the column vector $\mathbf{x}$, we may produce vector $\mathbf{y}$, having elements y_i that are given by

$$y_i = \sum_{j=1}^{N} H_{ij} x_j \,, \tag{2}$$

because its computation can be formulated as:

(A) Clear all of $\mathbf{y}$ to zero.
(B) Fetch some elements of $\mathbf{H}$ into memory along with their i,j indices.
(C) For each $\mathbf{H}$ in memory, sequentially replace y_i by $y_i + H_{ij}x_j$.
(D) Repeat from step B until all nonzero elements of $\mathbf{H}$ have been processed.

This procedure can process the matrix elements in any order. For a serial computer, the computer time is directly proportional to the number of nonzero matrix elements. Parallel processing can speed up the arithmetic involved in Eq. (2), because different blocks of $\mathbf{H}$ can be processed in different CPUs and the partial sums can be combined at the end.

All methods that have been suggested for the large matrix problem can be viewed as variants on one basic idea: An intermediate set of vectors is established in terms of which the actual eigenvector can be expanded. If this intermediate set of vectors $\{\mathbf{b}_p;\ p = 1,\ldots,P\}$ is cleverly chosen, then the expansion for $\mathbf{x}_k$ to the true eigenvector $\mathbf{c}_k$, expanded in the form

$$\mathbf{x}_k = \sum_{p=1}^{P} \mathbf{b}_p a_{pk} \tag{3}$$

will converge to $\mathbf{c}_k$ even when P is much less than N and when $\{\mathbf{b}\}$ is a very incomplete set of basis vectors. If these basis vectors are organized as the columns of a matrix $\mathbf{B}$, and the expansion coefficients are placed in a column vector $\mathbf{a}_k$, then the variational approximation to the solution of the matrix eigenvalue problem (1) in the subspace becomes

$$\mathbf{B}^T\mathbf{H}\mathbf{B}\mathbf{a}_k = \mu_k^P \mathbf{B}^T\mathbf{B}\mathbf{a}_k, \quad k = 1,\ldots,P, \tag{4}$$

in which the superscript T denotes matrix transpose. If the eigenvalues μ_k^P of this reduced problem are numbered in increasing order then, by the Hylleraas–Undheim–MacDonald variational principle[12] (which states that as the size of the basis increases, each eigenvalue tends to the true eigenvalue), the eigenvalues of the original problem (1) satisfy

$$\lambda_k \le \mu_k^P, \quad k = 1,\ldots,P. \tag{5}$$

Further, if $\{\mathbf{b}\}$ is simply augumented by an additional vector, then

$$\mu_k^P \le \mu_k^{P-1}, \quad k = 1,\ldots,P-1. \tag{6}$$

Hence, a general strategy is possible in which the eigenvalues $\{\mu_k^P; k = 1,\ldots,K; P = K, K+1,\ldots\}$ form monotonic nonincreasing sequences that converge to the desired exact eigenvalues $\{\lambda_k; k = 1,\ldots,K\}$.

If $\mathbf{x}_k$ is an approximation to $\mathbf{c}_k$, then the correction $\boldsymbol{\delta}_k = \mathbf{c}_k - \mathbf{x}_k$ is given by solving

$$\begin{aligned} \mathbf{r}_k &= (\mathbf{H} - \lambda_k \mathbf{1})\mathbf{x}_k, \\ (\mathbf{H} - \lambda_k \mathbf{1})\boldsymbol{\delta}_k &= -\mathbf{r}_k. \end{aligned} \tag{7}$$

The correction $\boldsymbol{\delta}_k$ may be approximated by replacing λ_k by the Rayleigh quotient

$$\rho_k = \mathbf{x}_k^T\mathbf{H}\mathbf{x}_k / \mathbf{x}_k^T\mathbf{x}_k \tag{8}$$

and approximating the inverse of $(\mathbf{H} - \rho_k\mathbf{1})$ by $(\mathbf{D} - \rho_k\mathbf{1})^{-1}$, where $\mathbf{D}$ is the diagonal of $\mathbf{H}$. Thus,

$$\boldsymbol{\delta}_k \approx -(\mathbf{D} - \rho_k\mathbf{1})^{-1}\mathbf{r}_k. \tag{9}$$

Simply adding $\boldsymbol{\delta}_k$ to $\mathbf{x}_k$ to get a new estimate of $\mathbf{c}_k$ is found not to work well. If, however, $\mathbf{x}_k$ is the result of a variational calculation in a subspace $\mathbf{B}$, as described above, then $\boldsymbol{\delta}_k$ is a good choice for the next addition to $\{\mathbf{b}\}$. This procedure is iterated until a suitable convergence criterion is met. In the original paper by Davidson,[7] only one vector was added to $\mathbf{B}$ at each iteration. The modification suggested by Liu[8] was to add a correction vector for each desired root on each iteration. In practice, due to limitations in computer memory, a compromise is made, and corrections to as many vectors as possible are added each time.

The basic algorithm based on these concepts is outlined in Box 1. In this algorithm, it is assumed that $\{\mathbf{b}\}$ is orthonormal, so that $\mathbf{B}^T\mathbf{B}$ is a unit matrix. Also it is assumed that the vectors in $\{\mathbf{b}\}$ and the vectors in $\{\mathbf{h}\}$ given by

$$\mathbf{h}_p = \mathbf{H}\mathbf{b}_p \tag{10}$$

from previous iterations are retained on disk, and the new ones for the current iteration are retained in memory. Computer memory is assumed to be adequate for holding J vectors plus additional small arrays. If $2K \leq J$ then the number Q of additional **b** vectors added on each iteration will be K. Otherwise, only $Q = J/2$ additional **b** vectors can be included each iteration.

For large matrices most of the computer time is used in step F of Box 1. Notice that this is the only step requiring access to the **H** matrix, and it is needed here just to form matrix–vector products. In particular, the residuals in step C are formed without reference to **H** by using the $\{\mathbf{b}\}$ and $\{\mathbf{h}\}$ sets of vectors. The truncation in step D is also formulated entirely in terms of linear transformations and scalar products involving these sets of vectors.

Step B requires computer time proportional to P^3, and step C requires time and data storage proportional to NP. Hence, it is important that a limit be set on P so that these time requirements remain small compared to the time needed in step F, and so that the data storage for $\{\mathbf{b}\}$ and $\{\mathbf{h}\}$ does not exceed the space available on disk.

The truncation procedure to $2K$ vectors in step D is almost equivalent to the conjugate gradient method[9] when the eigenvalues are nearly converged. Hence, frequent truncation does not slow convergence of the eigenvector once the eigenvalue is close.

The Lanczos algorithm

The other algorithm commonly used for large matrices is due to Lanczos,[13] also described in Chap. 13 of Ref. 1. In this method, the $\{\mathbf{b}\}$ vector space is chosen to span the same space as the set $\{\mathbf{H}^{p-1}\mathbf{g}; p = 1,...,P\}$ for some fixed arbitrary vector **g**. This sequence of vectors is known as the Krylov space (Ref. 1, Chap. 12). Sequential orthogonalization gives the set actually used by Lanczos. The major advantage of the Lanczos method is that the matrix $\mathbf{B}^T\mathbf{HB}$ is tridiagonal, which simplifies finding the eigenvalues. Also, a simple recursion relation for $\mathbf{B}^T\mathbf{HB}$ can be found, so that the eigenvalues (but not the eigenvectors) can be found without storing all of $\{\mathbf{b}\}$. Although this appears at first sight to be very different from the method discussed above, the two methods are closely related in one special case. [*For more on the Lanczos method, see* Computers in Physics, *Jul/Aug 1993, p. 400—Ed.*]

If the Davidson algorithm is initialized with the Krylov space for $p = 1,...,K$, if the denominator is omitted in step E, and if truncation is not done, then the two methods are identical. In this case, when $\{\mathbf{b}\}$ contains P vectors, all the residuals $\mathbf{r}_k$ are contained in the space spanned by $\{\mathbf{b}\}$ plus the additional vector $\mathbf{H}^P\mathbf{g}$. The dimension of $\{\mathbf{b}\}$ is therefore increased by only one at each iteration, even when $Q = K$. Further, the residuals are easily shown to be the gradients of the Rayleigh quotients (8), which are vectors composed of the derivatives of the Rayleigh quotient with respect to the components of $\mathbf{x}_k$. The Lanczos method is thus equivalent to a gradient method for all eigenvalues. By contrast, the Davidson–Liu method is a Newton–Raphson method with an approximation to the matrix of second derivatives. Whereas a gradient procedure will always proceed to an optimum, it is a first-order method. A true Newton–Raphson step, by contrast, has only second-order errors and will converge much more rapidly once the algorithm is close to a solution. The relative rates of convergence of the two methods, therefore, depend on the error at any stage and the improvement gained by use of an approximate second-derivative matrix.

There is also a variant of the Lanczos method that begins with an arbitrary set of vectors $\{\mathbf{g}_p;\ p = 1,...,K\}$ and forms the sequence $\mathbf{H}^k\{\mathbf{g}\}$. This is known as the block-Lanczos method and is similar to the Davidson–Liu method in its use of blocks. There also exists a variant of the Lanczos method that occasionally truncates and restarts with the current estimate of the eigenvectors as the initial set $\{\mathbf{g}\}$. This is usually called subspace iteration (Ref. 1, Chap. 14). Thus, the Davidson–Liu and Lanczos methods differ essentially only in the use of the energy denominators in step E.

Table I: Comparison of algorithms for very large matrices, sizes $10^6 \times 10^6$ for the first three matrices, and dimensions 4^{10} and 4^8 for the last two. Variants of the Davidson–Liu algorithm are A and B, and the Lanczos method is C. The required number of products of matrices with vectors is given in the rightmost column.

Matrix	Method[a]	Number of roots	Number of iterations	Number of products
$\sqrt{n}$	A	1	5	5
	C	1	6	5
	A	5	20	72
	B	5	20	70
n	A	5	5	21
	B	5	6	24
	C	1	>50	
n^2	A	5	4	17
	B	5	4	18
	C	1	>50	
4^{10}	B	1	12	12
4^8	B	3	26	58

[a] Methods: (A) Initial $\{\mathbf{b}_p;\ p = 1,...,K\}$ obtain by diagonalizing the 10×10 matrix containing lowest 10 diagonal elements. (B) Initial $\{\mathbf{b}_p;\ p = 1,...,K\}$ contains unit vectors corresponding to lowest K diagonal elements. In both cases $\{\mathbf{b}\}$ was truncated to $2K$ vectors when dimension reached 25 for the first 3 matrices, and 19 for the last two. (C) Lanczos algorithm.

Comparing Davidson–Liu and Lanczos methods

We now compare these two algorithms for the lowest eigenvalues and eigenvectors of some monster matrices. Table I compares for three sparse matrices the Lanczos method with variants of the Davidson–Liu method. These matrices are of dimension 10^6 and have off-diagonal elements equal to either 0 or -1. The nonzero values occur for $i \leq 10$ or $j \leq 10$, or for $|i-j| \leq 2$ or $|i-j| \leq 10^6 - 12$, inclusively. Although these test matrices are not particularly similar to those in quantum applications, they were chosen to give well-defined matrices that could be tested later by other researchers using methods yet to be invented. The nonzero elements

Box 1: The Davidson–Liu algorithm for estimating the lowest eigenvalues and eigenvectors of very large matrices.

Step A: Initialize $\{\mathbf{b}\}$ as unit vectors $\{\mathbf{e}_i(p), p = 1,\ldots,K\}$ corresponding to the K lowest diagonal elements of $\mathbf{H}$. Initialize $\{\mathbf{h}_p\}$ as $\{\mathbf{He}_i(p)\}$ which are just the corresponding columns of $\mathbf{H}$. Initialize: $(\mathbf{B}^T\mathbf{HB})pq$ as $\mathbf{H}_i(p)j(q)$.

Step B: Get eigenvalues μ and eigenvectors $\mathbf{a}$ of $\mathbf{B}^T\mathbf{HB}$.

Step C: Form residuals $\mathbf{r}_k = \Sigma(\mathbf{h}_p - \mu_k \mathbf{b}_p)\mathbf{a}_{pk}$. Test for convergence. Stop if all are converged. Else select up to Q residuals of maximum norm for improvement.

Step D: Check if addition of Q vectors to $\{\mathbf{b}\}$ would exceed practical limits on the size of this set. If so, replace the set with current $\{\mathbf{x}_k^P;\ k = 1,\ldots,K\}$. If limits on the set size permit, also retain in the new $\{\mathbf{b}\}$ unconverged members of $\{\mathbf{x}_k^{P-1};\ k = 1,\ldots,K\}$ orthogonalized to $\{\mathbf{x}_k^P\}$. Replace $\{\mathbf{h}\}$ by the corresponding $\mathbf{h}_k^{\text{new}} = \Sigma\mathbf{h}_k^{\text{old}} f_{pk}$, where the new $\{\mathbf{b}\}$ is related to the old set by the transformation coefficients f_{pk}, and replace $(\mathbf{B}^T\mathbf{HB})pq$ by the new $\mathbf{b}_p^T\mathbf{h}_q$.

Step E: For each of the Q vectors targeted for improvement form $\boldsymbol{\delta}_k$ as $\boldsymbol{\delta}_{ik} = d_{ik}^{-1}\mathbf{r}_{ik}$, where $d_{ik} = |D_{ii} - \mu_k|$ for $D_{ii} - \mu_k| > \epsilon > 0$ and $d_{ik} = \epsilon\,\text{sign}(D_{ii} - \mu_k)$ otherwise. Orthogonalize this set of $\boldsymbol{\delta}_k$ to the previous $\{\mathbf{b}\}$ and to each other and append them to $\{\mathbf{b}\}$.

Step F: Form $\mathbf{h} = \mathbf{Hb}$ for each of the added $\mathbf{b}$ and append these to $\{\mathbf{h}\}$. Border $\mathbf{B}^T\mathbf{HB}$ by the additional rows and columns formed from the added $\mathbf{b}$ and all of the $\mathbf{h}$.

Step G: Iterate to step B.

occupy only a fraction of $\sim 3 \times 10^{-5}$ of the 10^{12} elements. The diagonal element H_{nn} for the three cases were chosen to be (a) $\sqrt{n}$, (b) n, or (c) n^2. For matrix dimension N, these give second-order perturbation estimates of the lowest eigenvalue that (a) diverge as $\sqrt{N}$, (b) diverge as $\ln(N)$, and (c) converge. The last case is most favorable for the algorithm discussed here. The least favorable case for this algorithm would be constant diagonal elements, where this method (for a single root) is just an awkward implementation of the Lanczos method. Second-order perturbation in this case would diverge as N, and convergence (not shown in the table) is a little slower than in case (a).

The last two matrices in Table I were formed as direct products of 4×4 matrices to give nonsparse matrices of dimension $4^{10} = 1\,048\,576$ and $4^8 = 65\,536$, respectively. Details of these matrices, which have a much higher fraction of nonzero elements than in (a)–(c), are given in Ref. 9. All matrices were regenerated at each iteration. Matrices (a)–(c) were relatively fast to generate, while regeneration of the direct-product matrices took most of the computing time. These last two cases are more typical of many applications, where generating the matrix is relatively slow compared to a matrix–vector product.

The Lanczos method performed poorly for all matrices except case (a). It was abandoned for cases (b) and (c) after 50 iterations did not produce a single converged eigenvalue. Similarly, it was abandoned after 10 iterations for the 4^{10} dimension matrix because there was little progress toward convergence.

The algorithm described in Box 1 is labeled B in Table I. In most cases it produces the fewest accesses to the **H** matrix and is preferable when access to the matrix is time-limiting. Adding only one vector at a time to $\{\mathbf{b}\}$ produces fewer matrix–vector products, but more total iterations. Restarting $\{\mathbf{b}\}$ with only the current guesses of the eigenvectors, rather than the last two guesses, slows convergence by about 10%. Starting with an improved initial guess for $\{\mathbf{b}\}$ obtained by diagonalizing a larger submatrix of **H**, shown in the table as case A, produces only a small savings.

Several implementations of variants of this method have been published, with a recent one being Ref. 14, where vectorization of the method—with modifications for atomic structure calculations—for Cray X-MP and Alliant FX/8 computers is described. Very significant speedups are produced by using the Davidson–Liu algorithm rather than traditional full-matrix eigenvalue methods.

From this brief discussion of eigenvalue methods for very large matrices, we see that there are many ingenious methods for handling them. As affordable computer memory and speed continue to increase, and as computer systems for simultaneous processing develop, we can look forward to being able to solve conveniently and efficiently many problems involving monster matrices.

References

1. B. N. Parlett, *The Symmetric Eigenvalue Problem* (Prentice–Hall, Englewood Cliffs, NJ, 1980).
2. I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices* (Clarendon, Oxford, 1986).
3. Z. Zlatev, *Computational Methods for General Sparse Matrices* (Kluwer, Dordrecht, The Netherlands, 1991).
4. P. M. Morse and H. Feshbach, *Methods of Theoretical Physics* (McGraw–Hill, New York, 1953), pp. 1117–1119; R. Courant and D. Hilbert, *Methods of Mathematical Physics* (Springer, Berlin, 1937), Vol. I, pp. 175–176.
5. P. Saxe, H. F. Schaefer III, and N. C. Handy, Chem. Phys. Lett. **79**, 202 (1981).
6. J. Olsen, P. Jørgensen, and J. Simons, Chem. Phys. Lett. **169**, 463 (1990).
7. E. R. Davidson, J. Comput. Phys. **17**, 87 (1975); Comput. Phys. Commun. **53**, 49 (1989).
8. B. Liu, in *Numerical Algorithms in Chemistry: Algebraic Methods*, edited by C. Moler and I. Shavitt (Lawrence Berkeley Laboratory, Berkeley, CA, 1978).
9. C. W. Murray, S. C. Racine, and E. R. Davidson, J. Comput. Phys. **103**, 382 (1992).
10. J. H. Wilkinson, *The Algebraic Eigenvalue Problem* (Oxford, University Press, Oxford, 1965).
11. B. T. Smith, J. M. Boyle, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, *Matrix Eigensystem Routines–EISPACK Guide*, Lecture Notes in Computer Science, edited by G. Goos and J. Hartmanis (Springer, Berlin, 1976), Vol. 6, 2nd ed.
12. A. Hylleraas and B. Undheim, Z. Phys. **65**, 759 (1930); J. K. L. MacDonald, Phys. Rev. **43**, 830 (1933); J. C. Slater, *Quantum Theory of Atomic Structure* (McGraw–Hill, New York, 1960), Vol. I, pp. 113–119.
13. C. Lanczos, Vol. I, J. Res. Natl. Bur. Stand. **45**, 255 (1950).
14 V. M. Umar and C. F. Fischer, Int. J. Supercomput. Appl. **3**, 28 (1989); A. Stathopoulos and C. F. Fischer, Comput. Phys. Commun. (to be published).

In the next issue: The Voigt profile function: Efficient numerical algorithms.