Volume 125, Article No. 125032 (2020) https://doi.org/10.6028/jres.125.032
Journal of Research of National Institute of Standards and Technology

Parallel Generalized Real Symmetric-Definite Eigenvalue Problem

James S. Sims[1] **and María Bélen Ruiz**[2]

[1]National Institute of Standards and Technology,
Gaithersburg, MD 20899, USA

[2]Friedrich-Alexander-University Erlangen-Nürnberg
Egerlandstraße 3, D-91058, Erlangen, Germany

jim.sims@nist.gov
maria.belen.ruiz@fau.de

Software DOI: https://doi.org/10.18434/mds2-2293

A computationally fast Fortran 90+ quadruple precision portable parallel GRSDEP (generalized real symmetric-definite eigenvalue problem) package suitable for large (80,000 x 80,000 or greater) dense matrices is discussed in this paper.

Key words: dense matrix eigensolver; high precision; parallel.

Accepted: October 22, 2020

Published: November 6, 2020

https://doi.org/10.6028/jres.125.032

1. Introduction

Few-electron atomic and molecular systems continue to play an important role as a testbed for understanding fundamental physical and chemical theories. Non-relativistic ground state energies and wave functions for few-electron atoms are primary inputs for computations of atomic properties such as electron affinities and ionization potentials, as well as for computing perturbative corrections due to relativity and quantum electrodynamics [1]. Furthermore, fundamental quantum theory dictates that, in general, uncertainties in wave functions scale as the square root of uncertainties in corresponding energies. Therefore, when using convergence of energies as a metric for convergence of associated wave functions, very high accuracy in the former must be achieved to guarantee accuracy in the later.

To achieve this high accuracy (real(24) arithmetic is necessary for 2 electron systems and real(16) for more than 2 electron systems) requires explicitly correlated wave functions which in turn lead to dense matrices with sizes (N) that are large compared to most dense matrix problems. This in turn leads to linear dependency issues that are solved by going to higher precision, all of which means that calculations need to be parallelized for both computational speed and to spread the memory out across a cluster. Hence parallel packages like ScaLAPACK, which is available only in real(8), cannot be used as they stand and the packages are too big to contemplate conversion to real(16). This has led to the development of the portable parallel GRSDEP (generalized real symmetric-definite eigenvalue problem) package which is discussed in this paper.

2. Parallel GRSDEP

Computational Quantum Chemistry involves solving Schrödinger's equation for N_e electrons

$$H\Psi(\mathbf{X}_1,\mathbf{X}_2,...,\mathbf{X}_{N_e}) = E\Psi(\mathbf{X}_1,\mathbf{X}_2,...,\mathbf{X}_{N_e}), \tag{1}$$

where $\mathbf{X_i} = (\mathbf{r_i}, \xi_\mathbf{i})$ is the combined space-spin coordinate for electron i, $\Psi(\mathbf{X}_1,\mathbf{X}_2,...,\mathbf{X}_{N_e})$ represents a wave function describing a given quantum mechanical system, E is the energy of the system, and H is the Hamiltonian operator specifying components of the energy included in the model.

The wave function given by Equation (1), $\Psi(\mathbf{X}_1,\mathbf{X}_2,...,\mathbf{X}_{N_e})$, is a linear combination of terms Φ_K, where the coefficients c_K are those which minimize the total energy, E, given by

$$E = \frac{<\Psi|\mathscr{H}|\Psi>}{<\Psi|\Psi>} = \frac{\sum_{K,L} c_K c_L H_{KL}}{\sum_{K,L} c_K c_L S_{KL}}, \tag{2}$$

where

$$H_{KL} =<\Phi_K|\mathscr{H}|\Phi_L>; \quad S_{KL} =<\Phi_K|\Phi_L> . \tag{3}$$

The condition for the energy to be an extremum, $\delta E = 0$, is the well-known matrix eigenvalue (secular) equation:

$$\sum_{L=1}^{N} c_L H_{KL} = E \sum_{L=1}^{N} c_L S_{KL}. \tag{4}$$

Solving this equation is equivalent to solving the N-dimensional generalized eigenvalue problem (GEVP)

$$\mathbf{Hc} = E\mathbf{Sc}, \tag{5}$$

where $\mathbf{H}$ and $\mathbf{S}$ are both symmetric, have matrix elements H_{KL} and S_{KL} given by Equation (3), S is positive definite and N is the number of configurations Φ_K in Ψ. Equation (5) has N solutions defined by the pairs $\{(E,\mathbf{c})_i, \ i = 1,N\}$, each of them fulfilling the variational principle ([2]) for the ith quantum state of the system.

As the Hamiltonian operator H is hermitian, the matrices involved are symmetric (which makes this a generalized real symmetric-definite eigenvalue problem (GRSDEP)), so it is sufficient to compute only the upper or lower triangle of the matrix (an important CPU time and memory saving feature). For use in quantum chemical calculations[1] several iterative procedures have been devised for Configuration Interaction(CI) [3–5] and for molecular elliptic coordinate calculations [6]. As pointed out by Lüchow and Kleindienst [7], all of these show slow convergence in the more accurate explicitly correlated calculations and cannot be applied successfully if high precision, spectroscopic accuracy is desired. This slow convergence is probably due to the large off-diagonal matrix elements in the (dense) matrices. For high precision, real(16) arithmetic is necessary for molecules and N $>$ 2 electron atoms, and real(24) arithmetic is necessary for 2 electron atoms. Hence parallel packages like ScaLAPACK, which is available only in real(8), cannot be used as they stand and the packages are too big to contemplate conversion to real(16). For these reasons a portable parallel GRSDEP package was developed.

2.1 Inverse Iteration

Inverse iteration (also called shifted inverse power method in this case) is one of the fastest methods for getting selected roots of the generalized eigenvalue problem, particularly if one has a pretty good idea of what the eigenvalue should be, as is the case in our calculations [8, 9]. When a good starting approximation

[1]With applicability in other fields, such as atomic, molecular and optical physics, as well.

for an eigenvalue is not available, one uses other methods such as Givens [10] to obtain the starting approximation.

Letting $\mathbf{A} = \mathbf{H} - E_0\mathbf{S}$, inverse iteration, developed first by Wielandt [11], is the power method [12] applied to $\mathbf{A^{-1}} = (\mathbf{H} - E_0\mathbf{S})^{-1}$ in the generalized eigenvalue problem, where $\mathbf{A^{-1}}$ is the inverse of A. E_0 is some starting approximation for the eigenvalue of interest. If $\mathbf{x}$ is an eigenvector of $\mathbf{H}$ with associated eigenvalue E (so that $\mathbf{Hx} = E\mathbf{Sx}$), then subtracting $E_0\mathbf{S}$ from both sides,

$$(\mathbf{H} - E_0\mathbf{S})\mathbf{x} = (E - E_0)\mathbf{Sx}, \tag{6}$$
$$\mathbf{x} = (E - E_0)(\mathbf{H} - E_0\mathbf{S})^{-1}\mathbf{Sx}, \tag{7}$$
$$(\mathbf{H} - E_0\mathbf{S})^{-1}\mathbf{Sx} = (E - E_0)^{-1}\mathbf{x}. \tag{8}$$

If $\mathbf{v}_0$ is not an eigenvector of $\mathbf{H}$ it can be expanded in the eigenvectors of $\mathbf{H}$ as follows

$$\mathbf{v}_0 = \sum_{i=1}^{N} c_i \mathbf{x}_i, \tag{9}$$

i.e., $\mathbf{v}_1$ is the vector obtained by operating with $\mathbf{A}^{-1}\mathbf{S}$ on $\mathbf{v}_0$, $\mathbf{v}_2$ is obtained by operating with $\mathbf{A}^{-1}\mathbf{S}$ on $\mathbf{v}_1$, etc.

Letting $\mathbf{A} = \mathbf{H} - E_0\mathbf{S}$, the power method applied to $\mathbf{A^{-1}}$ (discussed in the Appendix) generates the sequence $\{\mathbf{v}_n\}$ recursively, using

$$\mathbf{v}_{n+1} = \mathbf{A}^{-1}\mathbf{S}\mathbf{v}_n. \tag{10}$$

Equation (10) is the basic iteration formula. Since it is an inverse, the eigenvalue ultimately converged upon will be a LOWER bound to the exact eigenvalue of this state, as required.

The inverse of $\mathbf{A}$ is not actually computed (which would not be efficient). Rather, as explained in the next (sub)section, L (and D) are computed and stored in a efficient form for subsequent use in an $\mathbf{L}^T\mathbf{D}^{-1}\mathbf{L}$ operation on $\mathbf{S}\mathbf{v}_n$.

The equation which is used (see Appendix) to monitor the convergence to the final eigenvalue E is

$$E = E_0 + \frac{\mathbf{v}_{n+1} \cdot \mathbf{S}\mathbf{v}_n}{\mathbf{v}_{n+1} \cdot \mathbf{S}\mathbf{v}_{n+1}}. \tag{11}$$

The convergence of $\mathbf{v}_n$ to $\mathbf{x}$ is faster the closer the trial E_0 is to E. Hence it is desirable to have a pretty good idea of what the eigenvalue is. It is generally sufficient to know the eigenvalue to 5 digits. When that is not the case, one can use NAG [13] or LAPACK [14] libraries with smaller expansions to get the starting value.[2]

2.2 Strategy for using Inverse Iteration

The (efficient) strategy for the inverse iteration without actually calculating $\mathbf{A}^{-1} = (\mathbf{H} - E_0\mathbf{S})^{-1}$ is as follows.

By construction a lower triangular matrix $\mathbf{L}$ such that $\mathbf{LA} = \mathbf{U}$, an upper triangular matrix is found. Right multiply by $\mathbf{L}^T$, where $\mathbf{L}^T$ is the transpose of $\mathbf{L}$, to get

$$\mathbf{LAL}^T = \mathbf{UL}^T = \mathbf{LU}^T = \mathbf{D}, \tag{12}$$

[2]When we have no "guess" for the desired eigenvalue, we use a truncated wave function and solve the eigenvalue problem by reduction of $\mathbf{S}$ to a unit matrix followed by solving the transformed $\mathbf{H}$ for the lowest few eigenvalues using Givens [15].

where $\mathbf{D}$ is a diagonal matrix since $\mathbf{LAL}^T$ is symmetric. Here the facts that the set of upper (lower) triangular matrices is closed under multiplication and that the intersection of the set of upper triangular matrices and lower triangular matrices is the set of diagonal matrices are used. Now solve for $\mathbf{A}$ to get

$$\mathbf{A} = \mathbf{L}^{-1}\mathbf{D}\mathbf{L}^{-T} \tag{13}$$

and

$$\mathbf{A}^{-1} = \mathbf{L}^T\mathbf{D}^{-1}\mathbf{L} \tag{14}$$

where we have used the fact that the inverse of the transpose is the transpose of the inverse. Instead of $\mathbf{A}^{-1}$ in Equation (10) $\mathbf{L}^T\mathbf{D}^{-1}\mathbf{L}$ is used.

It remains only to show how to determine $\mathbf{L}$ and $\mathbf{D}$ in the transformation of $\mathbf{A}$ to diagonal form. $\mathbf{L}$ is assembled by construction, namely,

$$\mathbf{L} = \mathbf{L}_N\mathbf{L}_{N-1}...\mathbf{L}_3\mathbf{L}_2 \tag{15}$$

and

$$\begin{aligned} \mathbf{A}_2 &= \mathbf{L}_2\mathbf{A} && (16)\\ \mathbf{A}_3 &= \mathbf{L}_3\mathbf{A}_2 && (17)\\ &... && (18)\\ \mathbf{A}_k &= \mathbf{L}_k\mathbf{A}_{k-1}. && (19) \end{aligned}$$

$\mathbf{L}_2$ is the lower triangular matrix that sets the off-diagonal lower triangular elements of the second row of $\mathbf{A}$ to zero, $\mathbf{L}_3$ operating on $\mathbf{A}_2$ does a similar thing for the third row of $\mathbf{A}_2$, and so on. In general, $\mathbf{L}_k$ is a matrix with 1s along the diagonal and zeros everywhere else except the lower triangular off-diagonal elements of row k, which are given by

$$-\mathbf{A}_{k-1}(i,k)/\mathbf{A}_{k-1}(i,i), i = 1,2,...,k-1; \tag{20}$$

that is, the elements of the k-th column of $\mathbf{A}_{k-1}$ divided by the corresponding diagonal elements.

Once $\mathbf{A}_k$ is obtained column k (above the diagonal) is no longer needed and the off-diagonal elements of row k of $\mathbf{L}_k$ can be safely stored there. Once $\mathbf{A}_N$ is obtained, the diagonal elements constitute the diagonal matrix $\mathbf{D}$. For convenience $\mathbf{D}$ is then replaced by $\mathbf{D}^{-1}$. Vector $\mathbf{A}$ is now what we call the implicit representation of $\mathbf{L}$. There is no need to multiply out the $\mathbf{L}$s (an Order N^3 operation); each operation on $\mathbf{x}$ by an $\mathbf{L}_k$ is at most an order N operation.

Note that the basic operation ($\mathbf{A}_k = \mathbf{L}_k\mathbf{A}_{k-1}$) in forming the $\mathbf{A}_k$s is the Fortran 90+ DOT_PRODUCT intrinsic function, which hopefully will be implemented efficiently. $\mathbf{L} = \mathbf{L}_N\mathbf{L}_{N-1}...\mathbf{L}_3\mathbf{L}_2$ also involves only the DOT_PRODUCT. $\mathbf{L}^T\mathbf{y}$ ($\mathbf{y} = \mathbf{D}^{-1}\mathbf{L}$) on the other hand uses the QAXPY BLAS operation, which also can be efficiently implemented for portability and extended precision.

A is not stored as an N by N array but rather as a vector of upper triangular columns, which are efficiently accessed by the DOT_PRODUCT function (and similarly for QAXPY).

One thing to note is that with the large matrices that one deals with in parallel problems, memory access patterns can be extremely important. The initial coding got very slow as large N was approached. An essentially trivial reordering of the operations involved with the $\mathbf{L}$s fixed the problem (a ROW form of $\mathbf{L}$ and a COLUMN form for $\mathbf{H}$ and $\mathbf{S}$ are now used) and led to substantial speedups for the large N case. All of which emphasizes that the construction of $\mathbf{L}$ is by no means unique and that this one is a very good one.

2.3 Parallelization

To parallelize one simply maps **A** to the several processors. A cyclic range is used to distribute elements of each row of the matrix **A** over the processors in a cluster. Each column is stored in its entirety on a particular processor. A cyclic "round robin" assignment of columns to the processors is made, which yields an approximately equal distribution of matrix elements to processors. This insures good load balancing (assuming homogeneous processors) and produces better load balancing than a block assignment (as in ScaLAPACK [16]) would.

In the parallel version of GRSDEP, MPI [17] is used to run the same program on multiple processors (on the same or different hosts) and each processor generates **H** and **S** only for the columns of **H** and **S** it owns.

The steps of the (parallel) inverse iteration are

1. Calculate $\mathbf{A} = \mathbf{H} - E\mathbf{S}$.
   On each processor, calculate the **A**s for the columns of the matrices that belong to this processor.

2. Calculate $\mathbf{Sv_0}$ using the vector $\mathbf{1}$=(1,1,...,1) as the starting vector $\mathbf{v}_0$.
   On each processor, calculate the partial vectors $\mathbf{b}_i = \mathbf{S}_i \cdot \mathbf{1}$ (a row sum, $\mathbf{1}$ is a vector of all ones) and send them to the root process (process 0). Process 0 then sums them up to get $\mathbf{b} = \mathbf{S} \cdot \mathbf{1} (= \mathbf{S}\mathbf{v}_0)$.

3. Find the transformation of **A** to the implicit representation of **L**.
   Calculate the partial **L**s of **A** with the partial **D**s stored in the diagonal elements. Set $k = 0$.

4. Begin iteration.

   (a) If $(n \neq 0)$ Calculate $\mathbf{b}_n = \mathbf{S}\mathbf{v}_n$.

   (b) Solve $\mathbf{v}_{n+1}$=$\mathbf{L}^T\mathbf{D}^{-1}\mathbf{L}\mathbf{b}_n$ (Call Ltdm1L to do $\mathbf{L}^T\mathbf{D}^{-1}\mathbf{L}\mathbf{b}_n$, returning the result as **x** which is $\mathbf{v}_{n+1}$).

5. Calculate $E = E_0 + \frac{\mathbf{v}_{n+1} \cdot \mathbf{S}\mathbf{v}_n}{\mathbf{S}\mathbf{v}_{n+1} \cdot \mathbf{v}_{n+1}}$.

6. Stop condition fulfilled?
   (E changed negligibly between last two iterations?)
   NO: set $\mathbf{v}_{n+1} = \frac{\mathbf{v}_{n+1}}{r}$ where $r = \sqrt{\mathbf{v}_{n+1} \cdot \mathbf{S}\mathbf{v}_{n+1}}$; $n \leftarrow n+1$; go to 4 (a).
   YES: stop.

One other point to discuss is how to handle the **L**s and **D**s. Do we keep them distributed during the operation with **L** on **x**, or are the distributed **L**s gathered into processor 0 and the **L** operations carried out on a single processor (that is serially; but even with multiple processors the operation by **L** is still strictly serial). The problem with keeping **L** on a single processor is that it unbalances the memory requirements. So **L** (on top of **A**) is kept distributed and the vectors are passed to the processors in a round-robin fashion. So for the **Lx** operation

1. process 0 applies its **L**s to **x** and passes the resulting vector to process 1.

2. process 1 applies its **L**s to the **x** from process 0, then passes result to process 2.

3. process 2 applies its **L**s to the vector received from process 0 and so on.

Except for root (P_0) a processor P_i receives an **x** from P_{i-1} and sends the **x** it calculates to P_{i+1}, except if $i+1$ is greater than last, in which case **x** will be sent to P_0, which will already have issued a receive. All processors except P_0 start out with a receive from processor myrank - 1 and wait until a message arrives. For $\mathbf{L^T x}$ one uses a similar logic except one starts with P_{last} and works down to P_0.

3. Summary of the Timing Tests

This section is an attempt to measure the effectiveness of the parallelization of the GRSDEP by calculating the time it takes (in seconds (s)) to compute the energy of a 4190 term H_2 wave function with a resultant total energy of -1.1744 7571 hartrees (Ha) [9] on differing numbers of processors. Table 1 gives the times (in seconds (s)). All results were obtained using real(16) precision (quadruple precision or QP, 128-bit, $\approx$ 32 digits) floating point arithmetic on the National Institute of Standards and Technology (NIST)'s 16908 processor Linux cluster. The Message Passing Interface (MPI) Standard [17] was used to parallelize the code.

Table 1. The time it takes to compute the ground state energy of the hydrogen molecule as a function of the number of processors utilized on a Linux cluster.

Number of processors	LD time (s)	Ltdm1L time (s)	Total time (s)
1	1392	26	1428
2	697	26	728
4	351	26	380
8	177	26	205
16	89	26	117
32	46	26	73

The LD step in the table refers to the time it takes to find **L** and **D** and gives a speedup of 30 on 32 processors, which is excellent! The 1 to 32 run speeds up by a factor of 20, a 23 minute job completing in a little over a minute. While performance can depend on the amount of memory available, the size of the problem (N), how one calculates $\mathbf{L^T}$, how elements of **A** are stored, etc., the speedup is ultimately limited by the two sequential steps, calculating the matrix elements and the Ltdm1L ($\mathbf{L}^T\mathbf{D}^{-1}\mathbf{L}$) step.

4. Conclusions

While the parallel GRSDEP has proven quite useful, having been utilized for investigating the following systems:

- 2-electron molecule: ground state of H_2,
- 2-,3-, 4-, and 5-electron atoms and their ions,

and producing some of the lowest (i.e. most accurate) variational energies ever reported, the lack of pivoting in this solver needs to be discussed.

The solver (shifted inverse iteration) used is one of the fastest methods for getting selected roots of the generalized eigenvalue problem. The only computational problem is the computation of the implicit representation of **L**. The addition of partial and/or complete pivoting in the use of inverse iteration complicates the code as the matrices go from triangular to square and much of the speed advantage of inverse iteration is lost. Equation (10) is also

$$\mathbf{A}\mathbf{v}_{n+1} = \mathbf{S}\mathbf{v}_n \tag{21}$$

which is the familiar linear system of the form $\mathbf{Ax} = \mathbf{b}$ for which LAPACK has specific routines with pivoting. One of these which became available in version 3.5 utilizes the bounded Bunch-Kaufman (rook) pivoting algorithm [18] which is more accurate than partial pivoting, with comparable costs. Since LAPACK is available in Fortran, it was not hard to produce real(16) versions of those routines in place of step 3 and 4.b in the inverse iteration algorithm (sequential case) outlined in Section 2.3 above, and hence test the solver-produced results reported in, for example [19]. It should be noted that the Ritz estimate

$$E = E_0 + \frac{\mathbf{v}_{n+1} \cdot \mathbf{H}\mathbf{v}_{n+1}}{\mathbf{v}_{n+1} \cdot \mathbf{S}\mathbf{v}_{n+1}}. \tag{22}$$

is not used to monitor convergence to the final eigenvector E, rather it is Equation (11), derived in the Appendix, which is used in step 2.3.5. This has the advantage that the original H matrix does not need to be saved, freeing that storage space up to be overwritten by L, which then can be used to solve for a principal minor of dimension N' with very little additional CPU time.

In all cases the results reported in [19] were reproduced to all digits reported in that work. However it took 9.5 days to reproduce the result for the largest (N = 39,381) matrix (which took only 176 minutes on 120 processors), obviously not fast enough for a production code even for matrices of this size or smaller, much less for the sizes computed (up to $N \approx 80$ K) in earlier work [20]. Public domain codes are available for inverse iteration with partial and complete pivoting, but they do not meet the need for a parallel code which is fast, efficient, numerically stable, and of high enough precision (real(16)) to enable calculations which go significantly beyond the currently investigated sizes. A real(16) parallel version of the Rook pivoting algorithm, such as the one(s) of Strazdins [21], could greatly facilitate this endeavor.

5. Software Specifications

NIST Operating Unit	NIST Information Technology, Applied and Computational Mathematics, Scientific Applications and Visualization
Category	parallel GEVP (Generalized Eigenvalue Problem)
Targeted Users	High precision atomic and molecular science
Operating Systems	All
Programming Language	Fortran 90+
Inputs/Outputs	See the HowTo included with the software: https://doi.org/10.18434/mds2-2293
Documentation	Included with the software: https://doi.org/10.18434/mds2-2293
Disclaimer	https://www.nist.gov/director/licensing

Acknowledgments

This work would not have been possible without Stanley Hagstrom, who was always willing to share his seemingly infinite knowledge of chemistry and computer science with us. One of us would like to thank William George for MPI Support, Carl Spangler and Denis Lehane for parallel systems support, and Judy Devaney Terrill and Jules Becker for encouraging the publication of this work.

6. Appendix: The power method applied to $\mathbf{A} = \mathbf{H} - E\mathbf{S}$

Letting $\mathbf{v}_1$ be the vector which results from applying $\mathbf{A}^{-1}\mathbf{S}$ to $\mathbf{v}_0$,

$$\begin{aligned} \mathbf{v}_1 &= (\mathbf{H}-E_0\mathbf{S})^{-1}\mathbf{S}\mathbf{v}_0 = (\mathbf{H}-E_0\mathbf{S})^{-1}\mathbf{S}\sum_{i=1}^{N} c_i\mathbf{x}_i && (23)\\ &= \sum_{i=1}^{N} c_i(\mathbf{H}-E_0\mathbf{S})^{-1}\mathbf{S}\mathbf{x}_i = \sum_{i=1}^{N} c_i(E_i-E_0)^{-1}\mathbf{x}_i. && (24)\end{aligned}$$

Simliarly, let $\mathbf{v}_2$ be the vector which results from applying $\mathbf{A}^{-1}\mathbf{S}\mathbf{v}$ to $\mathbf{v_1}$, and similarly for $\mathbf{v_3}$, etc.

$$\begin{aligned} \mathbf{v}_2 &= (\mathbf{H}-E_0\mathbf{S})^{-1}\mathbf{S}\mathbf{v}_1 = \sum_{i=1}^{N} c_i(E_i-E_0)^{-2}\mathbf{x}_i && (25)\\ &\dots \\ \mathbf{v}_n &= (\mathbf{H}-E_0\mathbf{S})^{-1}\mathbf{S}\mathbf{v}_{n-1} = \sum_{i=1}^{N} c_i(E_i-E_0)^{-n}\mathbf{x}_i. && (26)\end{aligned}$$

From Equation (26) it is clear that if $(E_k - E_0)$ is small, after a few iterations the right hand side will be proportional to the eigenvector $\mathbf{x}_k$ (all terms but the $i = k$ term will be small), so that

$$\mathbf{v}_n \approx c_k(E_k-E_0)^{-n}\mathbf{x}_k. \tag{27}$$

Setting $E_k = E$ and applying $\mathbf{A}^{-1}\mathbf{S}$ again to $\mathbf{v}_n$, one gets

$$\begin{aligned} \mathbf{v}_{n+1} &= \mathbf{A}^{-1}\mathbf{S}\mathbf{v}_n, && (28)\\ \mathbf{v}_{n+1} &= c_k(E-E_0)^{-(n+1)}\mathbf{x}_k, && (29)\\ \mathbf{v}_{n+1} &= (E-E_0)^{-1}c_k(E-E_0)^{-n}\mathbf{x}_k, && (30)\\ \mathbf{v}_{n+1} &= (E-E_0)^{-1}\mathbf{v}_n, && (31)\\ (E-E_0)\mathbf{v}_{n+1} &= \mathbf{v}_n. && (32)\end{aligned}$$

Applying $\mathbf{S}$ to both sides and then taking the dot product of both sides with $\mathbf{v}_{n+1}$,

$$\begin{aligned} (E-E_0)\mathbf{v}_{n+1}\cdot\mathbf{S}\mathbf{v}_{n+1} &= \mathbf{v}_{n+1}\cdot\mathbf{S}\mathbf{v}_n, && (33)\\ E-E_0 &= \frac{\mathbf{v}_{n+1}\cdot\mathbf{S}\mathbf{v}_n}{\mathbf{v}_{n+1}\cdot\mathbf{S}\mathbf{v}_{n+1}}, && (34)\\ E &= E_0 + \frac{\mathbf{v}_{n+1}\cdot\mathbf{S}\mathbf{v}_n}{\mathbf{v}_{n+1}\cdot\mathbf{S}\mathbf{v}_{n+1}}. && (35)\end{aligned}$$

Which is Equation (11).

7. References

[1] King FW (1999) High-precision calculations for the ground and excited states of the lithium atom. *Advances In Atomic, Molecular, and Optical Physics* 40:57–112.

[2] Shull H, Löwdin PO (1958) Variational Theorem for Excited States. *Physical Review* 110(6):1467.

[3] Nesbet RK (1965) Algorithms for Diagonalization of Large Matrices. *Journal of Chemical Physics* 43:311–312.

[4] Davidson ER (1975) The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *Journal of Computational Physics* 17:87–94.

[5] Shavitt I, Bender CF, Pipano A, Hosteny RP (1973) The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices. *Journal of Computational Physics* 11:90–108.

[6] Cheung LM, Bishop DM (1977) The group-coordinate relaxation method for solving the generalized eigenvalue problem for large real-symmetric matrices. *Computer Physics Communications* 13:247–250.

[7] Lüchow A, Kleindienst H (1993) Stable and efficient algorithm for selected eigenvalues and eigenvectors of the general symmetric eigenproblem. *Computers & Chemistry* 17(1):61–66.

[8] Sims JS, Padhy B, Ruiz MB (2020) Exponentially correlated Hylleraas-configuration interaction (E-Hy-CI) nonrelativistic energy for the ^1S ground state of the helium atom. *International Journal of Quantum Chemistry* 12 October 2020:e26470. https://doi.org/10.1002/qua.26470

[9] Sims JS, Hagstrom SA (2006) High precision variational calculations for the Born-Oppenheimer energies of the ground state of the hydrogen molecule. *Journal of Chemical Physics* 124:094101.

[10] Ralston and Wilf (1967) *Mathematics for digital computers* (John Wiley and Sons) Vol. 2, p 94ff.

[11] Wielandt H (1944) *Ber B4/j/37 Aerodyn Vers-Anst* .

[12] Ralston A, Rabinowitz P (1978) *A First Course in Numerical Analysis* (McGraw-Hill, New York).

[13] The Numerical Algorithms Group (2020) The NAG Library, Oxford, United Kingdom. Available at https://www.nag.com/

[14] Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D (1999) *LAPACK Users' Guide* (Society for Industrial and Applied Mathematics, Philadelphia, PA) 3rd Ed.

[15] Moler C, Spangler D (1979) Eispack-based substitute for QCPE subroutine Givens. The National Resource for Computation in Chemistry.

[16] Blackford LS, Choi J, Cleary A, D'Azevedo E, Demmel J, Dhillon I, Dongarra J, Hammarling S, Henry G, Petitet A, Stanley K, Walker D, Whaley RC (1997) *ScaLAPACK Users' Guide* (Society for Industrial and Applied Mathematics, Philadelphia, PA).

[17] Message Passing Interface Forum (1994) MPI: A Message-Passing Interface Standard. *The International Journal of Supercomputer Applications* 8(3/4):159–416.

[18] Khabou A, Demmel JW, Grigori L, Gu M (2013) LU Factorization with Panel Rank Revealing Pivoting and Its Communication Avoiding Version. *SIAM Journal on Matrix Analysis and Applications* 34(3):1401–1429.

[19] Sims JS (2020) Hylleraas-configuration interaction (Hy-CI) non-relativistic energies for the 3 ^1S, 4 ^1S, 5 ^1S, 6 ^1S, and 7 ^1S excited states of the beryllium atom. *Journal of Research of the National Institute of Standards and Technology* 125:125006. https://doi.org/10.6028/jres.125.006

[20] Sims JS, Hagstrom SA (2014) Hylleraas-configuration-interaction nonrelativistic energies for the ^1S ground states of the beryllium isoelectronic sequence. *Journal of Chemical Physics* 140:224312. https://doi.org/10.1063/1.4881639

[21] Strazdins P (2000) Accelerated methods for performing the LDLT decomposition. *ANZIAM* 42:C1328–C1355.

About the authors: *James S. Sims, a computational scientist, is a former staff member (retired) and now a Research Associate in the Scientific Applications and Visualization Group, Applied and Computational Mathematics Division of the NIST Information Technology Laboratory.*

María Belén Ruiz is a Privatdozentin (independent lecturer and researcher) in the Department of Chemistry and Pharmacy at the Friedrich-Alexander-University Erlangen-Nürnberg in Erlangen, Germany.

The National Institute of Standards and Technology is an agency of the U.S. Department of Commerce.