Implicit restart Lanczos as an eigensolver

Reza Rajaie Khorasani and Randall S. Dumont*
Department of Chemistry, McMaster University, 1280 Main Street West, Hamilton, Ontario L8S 4M1, Canada
(Received 9 October 2008; revised manuscript received 7 January 2009; published 27 March 2009)

This paper investigates the efficiency of the implicit restart Lanczos and simple (without reorthogonalization) Lanczos algorithms, as eigensolvers for large scale computations in molecular and chemical physics. Using the cardioid billiard and the hydrogen cyanide/hydrogen isocyanide (HCN/HNC) molecule as model systems we demonstrate superior efficiency of implicit restart Lanczos compared to the simple Lanczos algorithm. A modified implementation of implicit restart Lanczos is also presented which works with a smaller Krylov space—with associated savings in memory—and can handle larger basis sets than the usual implicit restart Lanczos. It also enables getting all eigenpairs of a matrix, or all eigenvalues below a threshold (where the number of such is not known before hand), which is more difficult with the usual implicit restart algorithm.

DOI: 10.1103/PhysRevE.79.036708 PACS number(s): 02.60.Dc, 03.65.Ge

I. INTRODUCTION

The Lanczos algorithm [1–3] has become one of the most efficient and commonly used methods for large scale eigenvalue computations, with numerous applications in chemical and molecular physics—see, for example [4–12], (Ref. [11] contains a very comprehensive review on Krylov subspace methods in chemical physics). The algorithm is an orthogonal projection method for calculating a few (usually much less than matrix size) extremal eigenvalues and associated eigenvectors of large (usually) symmetric matrices. It relies on repeated matrix-vector multiplies to produce a set of basis vectors—the Lanczos vectors. The projection of a symmetric matrix onto this subspace (Krylov subspace) generates a symmetric tridiagonal matrix whose eigenvalues (called Ritz values) approximate the eigenvalues of the original matrix. There is never any need to store the full original matrix. All that is necessary is a means for computing the matrix-vector multiplies. In its simplest form the Lanczos algorithm is even more memory efficient, since only two Lanczos vectors need to be stored for computing eigenvalues. If eigenvectors are also wanted then one has to either store all the Lanczos vectors or the Lanczos iteration has to be repeated once the eigenvalues are computed, which is a considerable amount of extra work [13]. It is well known that in finite precision arithmetic Lanczos vectors lose orthogonality after a number of steps and spurious eigenvalues are generated. One solution to this problem, suggested by Cullum and Willoughby [1], is to identify and eliminate the spurious eigenvalues (this method is hereafter referred to as CWL). However, loss of orthogonality increases the size of the Krylov subspace necessary to span the space of the eigenvectors, and many more Lanczos iterations are required to converge the eigenvalues of interest. Another approach is to store the Lanczos vectors and use reorthogonalization methods to correct for the loss of orthogonality and reduce the size of the Krylov subspace. Although CWL (i.e., the simple Lanczos algorithm, without any reorthogonalization and without storing the Lanczos vectors) is more memory efficient, it is less cpu time efficient since it requires (often substantially) more matrix-vector multiplies. When Lanczos vectors are stored and reorthogonalized, eigenvectors are obtained with almost the same cost in cpu time as just computing eigenvalues. However, storing all the Lanczos vectors can have prohibitively large memory requirements. Moreover, as the iteration proceeds and the size of the Krylov subspace increases, maintaining orthogonality with respect to all previous Lanczos vectors becomes more and more costly. Restarting Lanczos methods [14–19] have been developed to address this problem. Restarting algorithms keep the size of the Krylov subspace fixed and restart the iteration after a predetermined number of steps. Implicit restart Lanczos (IRL) [3,19], as implemented in ARPACK96 [20], is one of the most widely used restart algorithms. IRL calculates k of the largest or smallest eigenvalues and associated eigenvectors using a Krylov subspace of size $m=k+p$. The algorithm contracts the Krylov subspace of size m to a Krylov subspace of size k and restarts the iterations at step $k+1$. The contractions are done so as to filter the Krylov subspace and generate a subspace richer in the wanted eigenvalues. The process is continued until all k wanted eigenvalues have converged. The storage requirement of IRL is limited by the need to store m Lanczos vectors, where m is much less than the size of the original matrix. The reorthogonalization cost is also limited to keeping m vectors orthogonal. In the simplest implementation of IRL, the number of wanted eigenvalues, k, and the total size of the Krylov subspace, m, are fixed numbers.

Due to its memory efficiency CWL has become the most commonly used of the Lanczos algorithms for large scale eigenvalue computations in the molecular and chemical physics community [4–7,9,10]. Although it is memory efficient and much appreciated, our investigations show that CWL suffers from a number of deficiencies as compared to IRL, especially for complex symmetric matrices. (1) CWL is much slower than IRL. (2) It is not as reliable as IRL. We found that when the number of wanted eigenvalues is not small, CWL often misses eigenvalues. CWL is even less reliable for complex symmetric matrices (note that complex symmetric matrices require the complex symmetric version of the Lanczos algorithm [1] and have complex eigenvalues). The reduced reliability of CWL for complex symmetric ma-

*Author to whom correspondence should be addressed. FAX: 905-522 2509; dumontr@mcmaster.ca

trices is also noted by Cullum and Willoughby [1]. (3) While for real symmetric matrices a bisection algorithm can be implemented to obtain the eigenvalues of the tridiagonal matrix in the energy range of interest, all eigenvalues of the tridiagonal matrix have to be computed in the case of complex symmetric matrices. Since without reorthogonalization the tridiagonal matrix can be very large, this step adds significantly to the required cpu time.

Although IRL, as implemented in ARPACK96, is widely used and very efficient, it also has limitations. (1) If the size of the original matrix, N, is very large (i.e., the number of the original basis vectors is very large) storing m Lanczos vectors of size N might not be possible. (2) It is not possible to calculate all eigenvalues using IRL. Since IRL keeps the Lanczos vectors orthogonal, the total size of the Krylov subspace, m, cannot exceed the size of the original space, N. So the best IRL can achieve is to calculate k eigenvalues, such that $k+p=m=N$. Decreasing p, the unwanted portion of the Krylov subspace, to make k approach N, increases the number of required restarts. Usually, p is chosen greater than or equal to k [19,21,22]. In practice IRL works best when extremal eigenvalues and eigenvectors are wanted such that $k \ll N$. (3) To implement IRL one must know the desired number of eigenvalues and eigenvectors, k. Often, this is not known. For example, all eigenvalues of a Hamiltonian matrix below a certain energy are typically desired. This number of such eigenvalues is generally not known beforehand, although it can be estimated via the Weyl formula [23].

In this paper, using the cardioid billiard model, we investigate the performance of real symmetric versions of CWL and IRL as eigensolvers. The cardioid eigenvalues are computed using Robnik's conformal mapping method [24]. CWL and IRL provide the requisite eigensolver. Even though the cardioid is just a two-dimensional system, because of the shape of its boundary—most notably its cusp, it is a challenging system for eigenvalue computations. Specifically, large basis sets are required to get converged energy eigenvalues. Furthermore, since the number of eigenvalues we seek—6000—is very large, the problem poses a special challenge for iterative methods. This is where IRL finds a niche. It is very difficult to converge the 6000 lowest eigenvalues using CWL, and be sure not to have missed any. CWL begins to miss eigenvalues after the first 500, and it is difficult to test and correct for the missing eigenvalues.

We also compared IRL to CWL in computations of rovibrational energy levels of the molecule, hydrogen cyanide/hydrogen isocyanide (HCN/HNC). Even when just 100 energy levels are desired, IRL is still faster than CWL—though by not such a large margin. The advantage of IRL is greatest for levels with $J>0$ for which a larger basis is required.

The above mentioned limitations of IRL are addressed via a variation in IRL, which we call sequential implicit restart Lanczos (SIRL). SIRL employs an alternating sequence of IRL and deflation [3,25] steps which afford a smaller Krylov space—thereby reducing memory requirements. It can be made to work with larger basis sets than is possible with IRL, and can be formulated to get all eigenvalues (and eigenvectors), or all eigenvalues below an energy threshold even when the number of such is not known. Deflation is well known, and used in combination with implicit restart Lanczos elsewhere [3]. The key advantage of our use of deflation—in SIRL—is the efficient use of writing to disk to make very large scale eigensystems accessible. This paper is organized as follows. In Sec. II, we briefly review the Lanczos and IRL algorithms and present the SIRL algorithm. In Sec. III, the algorithms are applied to the cardioid billiard model and numerical results are discussed. In the Appendix, we review a modern alternative to inverse iteration for calculating eigenvectors of a tridiagonal matrix [26], which to our knowledge has not been used before in the chemical and molecular physics communities. According to Fernando [26], the algorithm addresses significant shortcomings of inverse iteration. Our observations support these claims.

It should be mentioned that Lanczos algorithms are well suited for basis diagonalization methods, such as Robnik's, for calculating eigenpairs of billiards systems. These methods require large scale eigenvalue computations which can be done very efficiently with Lanczos, making them competitive with the Green's function matching methods [27,28].

II. LANCZOS ALGORITHMS

In this section we describe the Lanczos, implicit restart Lanczos, and sequential implicit restart Lanczos algorithms for the computation of eigenvalues of a real symmetric matrix $\mathbf{A}$. The basic Lanczos iteration starts with an arbitrary initial vector and generates a set of basis vectors $\mathbf{v}_j$ for a Krylov subspace, one vector at a time. Concurrently, it generates a symmetric tridiagonal matrix $\mathbf{T}$—the associated Lanczos representation of $\mathbf{A}$. After m steps, we have

$$\mathbf{A}\mathbf{V}_m = \mathbf{V}_m\mathbf{T}_m + \mathbf{r}_m\mathbf{e}_m^T, \tag{1}$$

where

$$T_m = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \ddots & \vdots \\ 0 & \beta_2 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_{m-1} \\ 0 & \cdots & 0 & \beta_{m-1} & \alpha_m \end{bmatrix},$$

$$\mathbf{V}_m = [\mathbf{v}_1\mathbf{v}_2 \ldots \mathbf{v}_m],$$

and the residual vector,

$$\mathbf{r}_m = \beta_m\mathbf{v}_{m+1}.$$

$\mathbf{e}_m^T$ is the mth unit row vector.

The eigenvalues of $\mathbf{T}_m$,

$$\mathbf{T}_m\mathbf{x}_i = \lambda_i\mathbf{x}_i, \tag{2}$$

approximate the eigenvalues of $\mathbf{A}$. An eigenvalue, λ_i, is considered converged if

$$\|\mathbf{A}\mathbf{y}_i - \lambda_i\mathbf{y}_i\| = \|(\mathbf{A}\mathbf{V}_m - \mathbf{V}_m\mathbf{T})\mathbf{x}_i\| = |\beta_m x_{im}| < \epsilon, \tag{3}$$

where $\mathbf{y}_i = \mathbf{V}_m\mathbf{x}_i$. The pair, $(\mathbf{y}_i, \lambda_i)$, approximates the eigenpair of $\mathbf{A}$. $\mathbf{y}_i$ is called the Ritz vector and λ_i the Ritz value of $\mathbf{A}$. Note that x_{im} can be computed without determining all remaining elements of the eigenvectors, $\mathbf{x}_i$. This is useful when one just wants eigenvalues. If eigenvectors are also desired,

after the eigenvalues have converged, the $\mathbf{x}_i$ can be computed using inverse iteration or the direct method [26] described in the Appendix. Although a direct method based on the recursion relation between the elements of $\mathbf{x}_i$,

$$\beta_{k-1}x_{ik-1} + (\alpha_k - \lambda_i)x_{ik} + \beta_k x_{ik+1} = 0,$$

has been applied before in the chemical physics literature [29,30], we found that method to be unstable, yielding catastrophic error for some eigenvector components. The method we use (see Appendix) is accurate, completely stable, and very efficient. The former attribute is critical to SIRL which performs reorthogonalizations with respect to computed Ritz vectors—see below.

Algorithm 1 is an m step Lanczos algorithm with reorthogonalization [31]. The iteration begins with an arbitrary starting vector. In the reorthogonalization step the residual vector, $\mathbf{r}$, is tested for loss of orthogonality and if necessary orthogonalized against all previous Lanczos vectors. This step can be repeated a couple of times if necessary, although usually once is enough. A larger value of ρ enforces a more stringent orthogonality condition. Lehousq and Sorensen [3] suggest $\rho=1$. If after a few times (e.g., five) a proper residual vector has not been generated, $\mathbf{r}$ is set to a random vector and orthogonalized with respect to the previous Lanczos vectors, β_j is set to zero and α_j is set to its value before reorthogonalization. This step is rarely necessary. Once the tridiagonal matrix, $\mathbf{T}_m$, is constructed, its eigenvalues are computed and checked for convergence. If necessary the iteration is continued. Setting $\rho=0$ reduces the algorithm to Lanczos without reorthogonalization.

The implicit restart Lanczos algorithm (algorithm 2; see Ref. [31]) starts with an $m=k+p$ step Lanczos iteration, constructed using algorithm 1. k is the number of wanted eigenvalues (smallest or largest algebraically or in magnitude) and p is a positive number.

$$\mathbf{A}\mathbf{V}_m = \mathbf{V}_m\mathbf{T}_m + \mathbf{r}_m\mathbf{e}_m^T. \tag{4}$$

The eigenvalues of $\mathbf{T}_m$ are computed and checked for convergence. If the k wanted eigenvalues have not converged, p shifts are selected based on the unwanted eigenvalues. A natural choice is to use the unwanted eigenvalues themselves as shifts. Using an implicit shift QR factorization, Eq. (4) is transformed to

$$\mathbf{A}\hat{\mathbf{V}}_m = \hat{\mathbf{V}}_m\hat{\mathbf{T}}_m + \mathbf{r}_m\mathbf{e}_m^T\mathbf{Q}_m, \tag{5}$$

where $\hat{\mathbf{V}}_m=\mathbf{V}_m\mathbf{Q}_m$ and $\hat{\mathbf{T}}_m=\mathbf{Q}_m^T\mathbf{T}_m\mathbf{Q}_m$. Because of the structure of the residual term, $\mathbf{r}_m\mathbf{e}_m^T\mathbf{Q}_m$, Eq. (5) is not a Lanczos factorization. However, since $\mathbf{Q}_m$ is the product of a sequence of "adjacent" $(i,i+1)$ Givens rotations, $i=1$ to $m-1$, it is tridiagonal. The product of p such tridiagonals is banded with $2p+1$ nonzero diagonals. As such, the first $k-1$ elements of $\mathbf{e}_m^T\mathbf{Q}_m$ are zero, and the first k columns of Eq. (5) therefore constitute a k step Lanczos factorization. It can be shown [21,32] that this updated Lanczos factorization corresponds to a better initial vector, $\hat{\mathbf{v}}_1$, such that $\hat{\mathbf{v}}_1$ can be written as $P(\mathbf{A})\mathbf{v}_1$, where P is a polynomial of degree p with the shifts as zeros (i.e., the updated Lanczos factorization will be richer in the wanted eigenspace).

$$\mathbf{A}\mathbf{V}_k = \mathbf{V}_k\mathbf{T}_k + \mathbf{r}_k\mathbf{e}_k^T, \tag{6}$$

where $\mathbf{V}_k=\hat{\mathbf{V}}_m(1:N,1:k)$, $\mathbf{T}_k=\hat{\mathbf{T}}_m(1:k,1:k)$, and $\mathbf{r}_k=\hat{\mathbf{v}}_{k+1}\hat{\beta}_k + \mathbf{r}_m\mathbf{Q}_m(m,k)$. Note that N is the number of original basis functions. Beginning with Eq. (6), p additional Lanczos steps are computed (step 7 of algorithm 2) to construct a new Lanczos factorization of size m. The factorization is constructed as in algorithm 1, except that it begins at step $k+1$ and the starting vector $\mathbf{r}$ is not an arbitrary vector but the updated residual from Eq. (6). Once again, the eigenvalues of $\mathbf{T}_m$ are computed and tested for convergence. This process is continued until the k wanted eigenvalues have converged. The corresponding eigenvectors can be computed in the end using the methods mentioned above. The actual implementation of IRL is more involved than described here. The reader is referred to Refs. [3,19,20] for further details.

SIRL is a simple adaptation of IRL. Yet, it enables working with a smaller Krylov subspace and can handle much larger basis sets. To calculate k eigenvalues and eigenvectors, IRL requires a Krylov subspace of size $m=k+p$, where typically $p \geq k$ [19,21,22]. In principle, p can be any positive number. However, a too small p will slow down convergence and increase the number of restarts required. The idea behind our modification is to obtain the k wanted eigenvalues and eigenvectors in successive sets of k'. Each set of k' eigenvalues is obtained using IRL. For each set we work with a Krylov subspace of size $m'=k'+p'$, where $m'<m$, $k'<k$, and $p'<p$, and for each set (except the first) the Krylov subspace is kept orthogonal to the eigenvectors computed from previous sets, $\mathbf{Y}=[\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_t]$. For example, to compute 1000 eigenvalues using IRL, with $p=k$ one requires a Krylov subspace of size 2000. Using SIRL, with $k'=p'=200$ one constructs a Krylov subspace of 400 Lanczos vectors and obtains the first 200 Ritz pairs. The 200 Ritz vectors can be stored on disk to be read when necessary. The second 200 eigenvalues are then obtained from a second Krylov subspace (of size 400) kept orthogonal to the previous set of Ritz vectors. The process is continued until all 1000 eigenpairs have been computed. So, SIRL consists of a sequence of IRL steps with the associated algorithm 1 modified to account for the previously converged Ritz vectors—i.e., they are deflated from the Krylov space. Algorithm 1′ is the modified Lanczos algorithm used in SIRL [31]. It involves orthogonalizing the arbitrary starting vector, $\mathbf{r}$, against previously converged Ritz vectors and repeating the selective orthogonalization after every reorthogonalization step. SIRL can be thought of as IRL with a systematic deflation of the previously converged eigenspace. It is different from the usual implementations of deflation, which are typically employed within a single IRL step (or related algorithm). [3] In SIRL, deflation occurs after each implementation of an IRL step, i.e., the converged eigenvalues are deflated from the next IRL step for the following k' eigenvalues. The key to SIRL is that k' can be chosen to be much smaller than k, without sacrificing performance. We typically use $k'=k/5$. As such, the storage required for Lanczos vectors is reduced to 1/5th its value with IRL. Now, this memory savings is mitigated by the requirement of saving all the converged Ritz vectors. However, because the latter can be written to and

TABLE I. cpu time (seconds) and approximate number of matrix-vector multiplies for computing k eigenvalues using IRL and CWL algorithms. N=the number of basis functions determined by maximum Bessel function order, ν_{max}.

		IRL			CWL	
ν_{max}	N	k	cpu time	Mv multiplies	cpu time	Mv multiplies
100	4001	500	113	1250	615	13000
100	4001	1000	399	2500	3116	65000
150	8945	500	378	1250	2896	14000
150	8945	1000	1143	2500	15566	71000
200	15853	500	1006	1250	9340	15000
200	15853	1000	2658	2500	46847	75000

read from disk in blocks, SIRL affords much larger scale computations than IRL.

Note that the extra selective reorthogonalization, i.e., orthogonalizing against previously converged Ritz vectors, $\mathbf{Y}=[\mathbf{y}_1, y_2, \ldots, y_t]$, does not increase the orthogonalization cost in each step compared to IRL. The number of vectors orthogonalized against (only in the last IRL step within SIRL) and the total number of vectors that have to be stored, are the same for both algorithms, except that in IRL all these vectors are Lanczos vectors whereas in SIRL they are both Lanczos vectors and converged Ritz vectors. Using the Ritz vectors enables dividing the Krylov subspace and obtaining the eigenvalues in sets. SIRL is also more flexible in memory usage in that the converged Ritz vectors can be stored on disk and read from file, in blocks, when necessary. This flexibility in memory usage together with the smaller Krylov subspace ($m<m'$) enables SIRL to handle larger basis sets than possible with IRL. For example, in Sec. III we report 6000 eigenvalues computed with SIRL. The limit for IRL, on the computer used for the study, is about 3000. We are currently approaching our goal of 10 000 cardioid eigenvalues—the estimated number needed for a semiclassical statistical analysis of the spectrum proposed by Sakr [33].

Two points should be noted. (1) It is not possible to implement IRL in sets, using the previous Lanczos vectors instead of the previous Ritz vectors to divide the Krylov subspace. In a Lanczos factorization of size m it is not known which Lanczos vectors correspond to the space of the converged eigenpairs. In fact, if more than just a few Ritz vectors are desired, the full set of Lanczos vectors is generally required to represent them. However, this set also represents unconverged Ritz vectors that have no utility in selective reorthogonalization. (2) One might be inclined to make IRL more flexible in memory usage by storing the Lanczos vectors on disk. However, IRL does not lend itself easily to such an approach. Updating the Lanczos vectors after the QR factorization (step 6 of Algorithm 2) is problematic in this case.

Algorithm 1′, in the context of SIRL, can readily be used to obtain all eigenvalues and vectors. Since the subsequent Krylov subspaces are orthogonal, each subspace generates the next set of eigenvalues. One can simply continue to compute the eigenpairs in sets until $m'(=k'+p')$ plus the number, t, of converged Ritz vectors exceeds or equals the number of original basis functions, N. At this point m' would be larger than or equal to the subspace of the remaining eigenvalues $(N-t)$, so a restart algorithm cannot (and need not) be used for the $N-t$ eigenvalues. These remaining eigenvalues are computed by a simple and relatively small orthogonal Lanczos factorization (of size $N-t$). Specifically, using algorithm

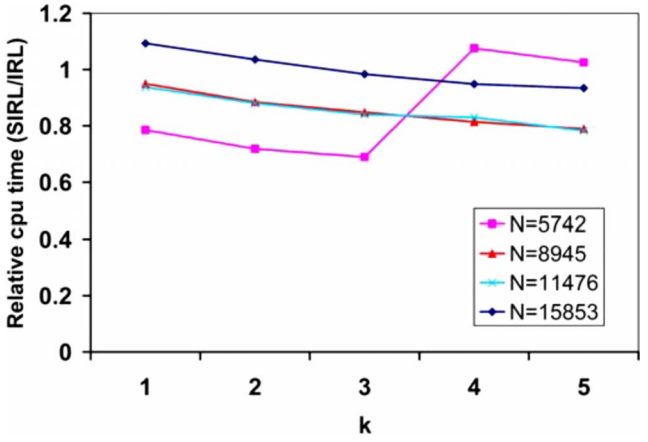

FIG. 1. (Color online) Relative cpu time (SIRL/IRL) vs k, the number of wanted eigenpairs, computed for different number of basis functions (N) and fixed $k/k'=5$.

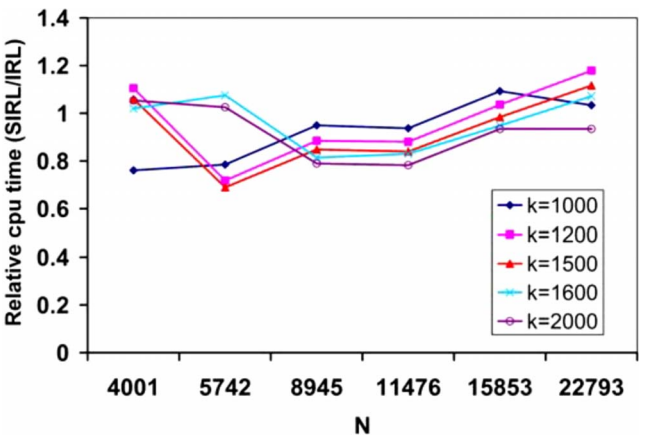

FIG. 2. (Color online) Relative cpu time (SIRL/IRL) vs N, the number of basis functions, computed for different k and k' such that $k/k'=5$.

$1'$ a small Krylov subspace of size $N-t \leq m'$ is constructed orthogonal to the previous Ritz vectors—the associated Ritz values are the remaining eigenvalues.

SIRL can also easily be adapted to situations where all eigenvalues below a certain threshold are wanted, and it is not known how many there are. Since SIRL obtains the k eigenvalues in sets of k' all we have to do is scan the eigenvalues of each set as they are generated. Once there are no further Ritz values below threshold, the process is terminated.

III. RESULTS AND DISCUSSIONS

To compare the performance of Lanczos without reorthogonalization (CWL), IRL, and SIRL, we calculated the eigenvalues and eigenvectors (even solutions only) of the cardioid billiard using Robnik's method. The method involves a conformal mapping of a family of billiards to the unit disk. The Schrödinger equation for the unit disk is discretized by invoking the basis functions (for even solutions),

$$\phi_{n,\nu} = R_{n,\nu} J_\nu(\gamma_{\nu,n} r)\cos\,\nu\theta,$$

where (r,θ) are polar coordinates in the x,y plane, J_ν are Bessel functions of order ν, and $R_{n,\nu}$ are normalization constants. $\gamma_{\nu,n}$ is the nth zero of the order ν Bessel function. The family of cardioidlike billiards studied by Robnik is characterized by two parameters, A and λ. Our particular choice, with an area of π, corresponds to $\lambda=0.5$ and $A=1/\sqrt{1+2\lambda^2}$. This is the cardioid billiard with a cusp in its boundary. In this case, a large basis set is required to converge the eigenstates in the neighborhood of the cusp, making the computation of a very large number of eigenvalues quite challenging. A finite basis was chosen such that the order of the Bessel functions ranged from $\nu=0$ to $\nu=\nu_{\max}$, while n ranged from 1 to $\frac{\sqrt{(\nu_{\max}^2-\nu^2)}}{2}+1$. All computations were done on an AMD 3 GHz processor PC with 3 GB of RAM.

Table I shows cpu times for calculating eigenvalues using IRL and CWL. The first 500 and 1000 eigenvalues were calculated using 4001 ($\nu_{\max}=100$), 8945 ($\nu_{\max}=150$), and 15853 ($\nu_{\max}=200$) basis functions. IRL computations used $p=k$ (number of wanted eigenvalues). The results clearly demonstrate that CWL is slower than IRL. As the number of basis functions and the cost of each matrix-vector multiply increase CWL becomes successively less efficient, in terms of cpu time. If eigenvectors are also required then CWL would be even less efficient since it requires repeating the Lanczos iteration. IRL was able to obtain 500 eigenvalues and eigenvectors with a total of 1250 matrix-vector multiplies, and 1000 eigenpairs with 2500 matrix-vector multiplies (for all three basis sets). CWL required many more matrix-vector multiplies.

To compare SIRL and IRL, eigenvalues and vectors were computed for different numbers of basis functions. In all cases we chose $k=p$ (for IRL) and $k'=p'$ (for the Krylov subspace divisions of SIRL). As for cpu time, SIRL could be faster or slower than IRL, depending on k, k' and the number of basis functions. Tables II and III show some typical results. We found that usually (but not always), as the number of wanted eigenpairs, k, increases for fixed number of basis functions, the relative efficiency of SIRL (i.e., in comparison to IRL) increases. Figure 1 shows the ratio of cpu times (SIRL/IRL) to calculate k eigenpairs vs k, for $k/k'=5$. Another trend we observed was that usually (again, not always) the relative efficiency of SIRL decreases with increasing size of basis set (see Fig. 2). However, in all cases, the efficiency of SIRL is very similar to that of IRL—the ratio of cpu times varies within the range 0.7 to 1.2.

TABLE II. cpu time (seconds) for computing k eigenvalues and eigenvectors using IRL and SIRL, for a basis set of 22 793 functions ($\nu_{\max}=240$).

$k=p$	$k'=p'$	SIRL	IRL
1000	100	8759	5392
1000	200	6582	5392
1000	250	6347	5392
1000	500	5575	5392
1600	100	18847	10983
1600	160	15178	10983
1600	200	13885	10983
1600	320	11771	10983
1600	400	11320	10983
1600	800	9711	10983

Note that the purpose of SIRL is not be to faster than IRL in terms of cpu time, but to be able to handle larger basis sets. SIRL allows the possibility of writing converged Ritz vectors to disk. Although this makes the algorithm slower, one can work with much larger basis sets than possible using IRL. We found that, even when writing Ritz vectors to disk, SIRL is still considerably faster than CWL. Using SIRL we calculated the first 6000 eigenvalues (Table IV) and eigenvectors (Fig. 3) of the cardioid (even solutions) using a basis set of 48 008 functions, $\phi_{n,\nu}$ ($\nu_{\max}=390$, with n ranging 1 to $\frac{2\sqrt{(\nu_{\max}^2-\nu^2)}}{5}+1$). Calculations were carried out in sets of $k'(=p')=\frac{k}{6}=1000$. The converged Ritz vectors were written to and read from disk in sets of 1000 at time. The 6000

TABLE III. cpu time (seconds) for computing eigenvalues and eigenvectors using IRL and SIRL, for a basis set of 8945 functions ($\nu_{\max}=150$).

$k=p$	$k'=p'$	SIRL	IRL
1000	100	1702	1327
1000	200	1266	1327
1000	250	1652	1327
1000	500	1606	1327
1600	100	3949	3080
1600	160	3153	3080
1600	200	2887	3080
1600	320	2508	3080
1600	400	2468	3080
1600	800	2363	3080

TABLE IV. Some eigenvalues of the cardioid (even solutions) calculated using SIRL and a basis size of 48008 functions, $\phi_{n,\nu}$, with $\nu_{\max}=390$, and n ranging 1 to $\frac{2\sqrt{(\nu_{\max}^2-\nu^2)}}{5}+1$.

i	Eigenvalue	i	Eigenvalue	i	Eigenvalue
1	6.0631286	4000	32188.944	5990	48144.968
2	16.646321	4001	32196.407	5991	48155.166
3	26.074062	4002	32202.870	5992	48158.595
4	32.931041	4003	32215.970	5993	48171.353
5	41.999007	4004	32222.640	5994	48178.853
6	53.162826	4005	32231.216	5995	48185.4845
7	61.347648	4006	32236.845	5996	48192.191
8	67.297500	4007	32242.625	5997	48195.644
9	76.097442	4008	32248.294	5998	48205.107
10	86.457570	4009	32258.747	5999	48212.331
	⋮		⋮	6000	48226.001

computed eigenvalues and eigenvectors are accurate to at least eight digits—the first ~3400 eigenvalues are accurate to ten digits. The cpu time for SIRL was approximately 6 days—23 855 Lanczos iterations (i.e., matrix-vector multiplies) were required in total. The computation time for CWL was much too long to be practical (several months).

Besides taking greater cpu time CWL also proved to be less reliable. With CWL, based on the errors estimated for the computed eigenvalues, one has to determine whether all the desired eigenvalues have been computed or not. We found this to be prone to error, particularly when many eigenvalues are required. In many cases we found that after sufficiently many iterations to yield error estimates of less than $10^{-9}-10^{-10}$ for all converged eigenvalues, a few eigenvalues were still missing. This was found even when we were looking for on the order of 500 eigenvalues, not just when we were seeking 6000 eigenvalues.

The cardioid eigenvalue computations show that when many eigenvalues are wanted, IRL is superior to CWL and the relative efficiency of IRL over CWL increases with increasing basis set size, and increasing number of wanted eigenvalues (see Table I). To further investigate the efficiency of CWL and IRL we compute rovibrational levels of the HCN/HNC molecular system (see Ref. [35] for details). Table V shows the cpu time required for computing 100 rovibrational levels (total angular momentum $J=0,1,2$) of HCN/HNC with even total parity. Table VI shows the corresponding number of matrix-vector multiplies. For a basis set size of 16 440 basis functions ($J=0$), the performance of CWL is comparable to IRL. However as the number of basis functions increases (for $J>0$), IRL becomes more efficient. The results indicate that, even when the number of wanted eigenvalues is not great, for larger matrices IRL is more efficient than CWL.

Although the computations in this paper were all done using the real symmetric Lanczos algorithm, it should be noted that, using the complex symmetric Lanczos algorithm, we also performed large scale eigenvalue computations for scattering Hamiltonians with complex absorbing potentials. We observed that the complex version of the CWL algorithm for identifying spurious eigenvalues has lower resolution than its real counterpart and it is more prone to missing eigenvalues or failing to spot spurious eigenvalues.

Some final comments are in order. (1) In IRL the only parameter that has to be chosen by the user (besides the number of wanted eigenvalues of course) is p, the unwanted portion of the Krylov subspace. Our choice of $p=k$ is the most natural choice. Choosing $p>k$ makes IRL even more efficient, but increases memory requirements. Note that the number of restarts is not set by the user. The algorithm simply restarts, when necessary, until all eigenvalues of interest are converged. (2) We also chose $k'=p'$ in SIRL. For obtaining the 6000 eigenvalues we set $k'=\frac{k}{6}$. If many eigenvalues are desired k' can be chosen based on memory restrictions. A lager k' requires writing less Ritz vectors to disk and reduces cpu time. (3) The efficiency of Lanczos depends on the distribution of eigenvalues. The less dense the area of interest in the spectrum, the faster Lanczos converges. This is true for all Lanczos algorithms—CWL, IRL, and SIRL. The relative efficiency of these algorithms is much the same for any broad class of matrices, such as molecular Hamiltonians [36]. (4) If the number of wanted eigenvalues is small, CWL will be more competitive with IRL. However, if the number of eigenvalues is large, CWL becomes increasingly less competitive—more so if eigenvectors are also wanted. Note that because so many eigenvalues are computed in this study, there is a wide range of local eigenvalue distributions represented. Consequently, the study is pretty representative of the performance one can expect for other large scale eigenvalue computations with generic molecular Hamiltonians. (5) As the number of basis functions increases IRL becomes more efficient over CWL. Increase in basis set size increases both the reorthogonalization cost in IRL and the matrix-vector multiplication cost. However it appears that the adverse effect on performance is much more severe for CWL because of the many more matrix-vector multiplies.

IV. CONCLUSION

In this paper we studied the CWL and IRL algorithms for calculating eigenvalues and eigenvectors. Although CWL is most memory efficient, our results indicate that IRL is faster

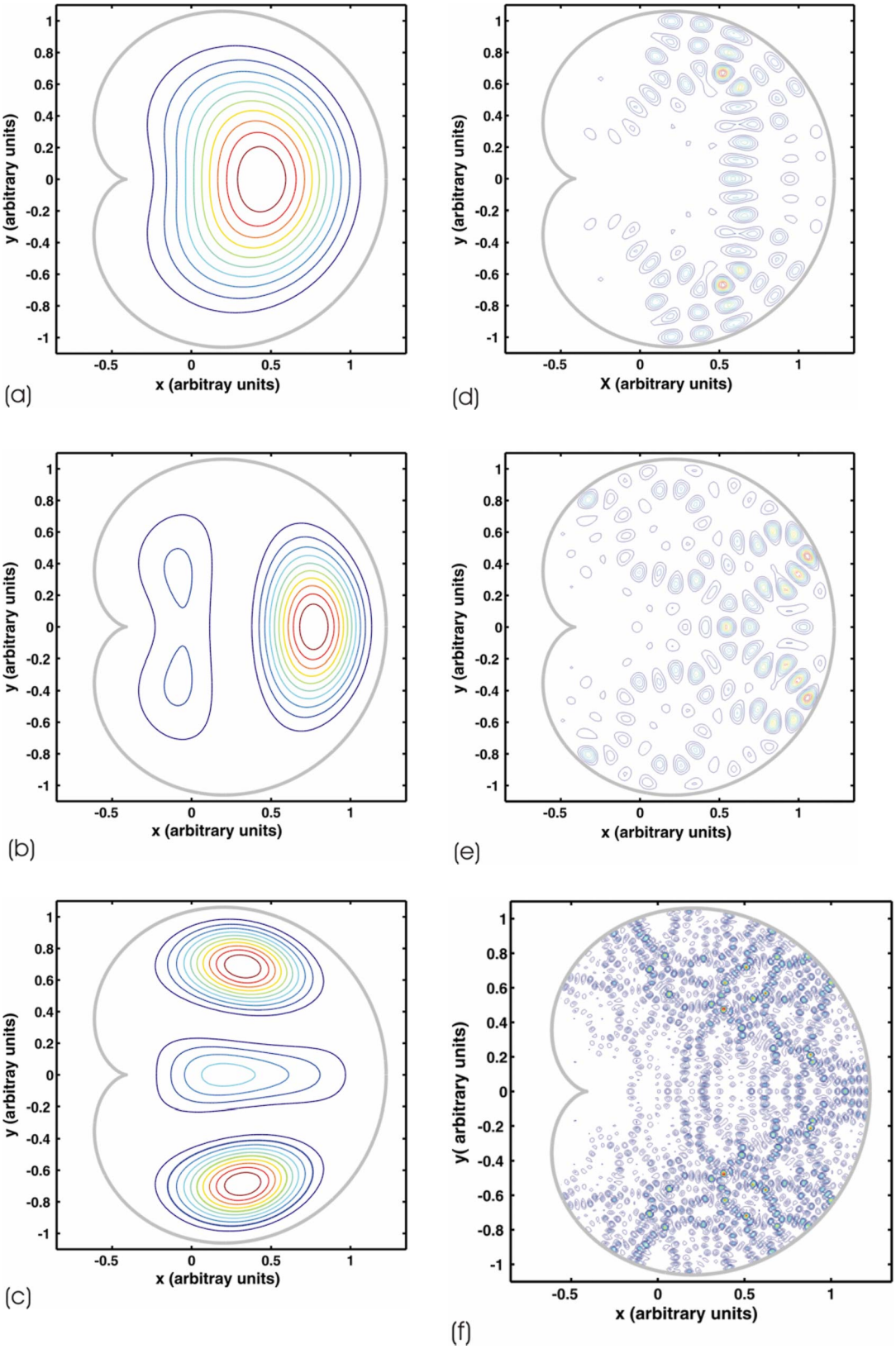

FIG. 3. (Color online) Contour plots of the even eigenvectors of the cardioid ($|\psi_i|^2$) corresponding to eigenvalues (a) $E_1=6.0631286$, (b) $E_2=16.646321$, (c) $E_3=26.074062$, and (d) $E_{101}=833.80725$, (e) $E_{105}=866.46615$, (f) $E_{1000}=8096.6728$. Panels (d) and (e) show evidence of scarring by classical periodic orbits—including diffraction orbits [34]. At high energies, eigenfunctions such as that shown in panel (f) are scarred by many orbits and are closer to the ergodic ideal.

TABLE V. cpu times (in seconds) for calculating 100 even rovibrational levels of HCN/HNC (N is the number of basis functions).

	$J=0$, $N=16440$	$J=1$, $N=32874$	$J=2$, $N=49268$
IRL	115	380	705
CWL	130	660	1200

and more reliable. We also presented an alternate implementation of IRL which we call SIRL. SIRL can be faster or slower than IRL depending on the number of wanted eigenvalues, the number of basis functions, and the subdivisions of Krylov subspace carried out in SIRL. However, the difference in speed of the two algorithms is small—SIRL was never more than 20% slower, and up to 30% faster. SIRL uses previously converged Ritz vectors in constructing the Krylov subspace and enables working with smaller Krylov subspaces. The Ritz vectors can be written to disk, at the expense of speed, allowing much larger basis sets than possible with IRL. SIRL was found to be faster than CWL even when Ritz vectors were written to and read from disk. SIRL also allows computation of all eigenvalues of a matrix, or all eigenvalues below a threshold without an *a priori* estimate of the number of such eigenvalues. Most importantly, because reading and writing from disk can be done efficiently for SIRL, it affords computations of very large numbers of eigenvalues. Such large sets of eigenvalues are very much of interest in studies of quantum chaos [37].

Based on our study of CWL, IRL, and SIRL, we find that IRL and SIRL are superior to CWL for calculating large numbers of eigenvalues and eigenvectors and should be the algorithm of choice, particularly for complex symmetric matrices. If only a small number of eigenvalues are desired, then CWL is certainly adequate. However, it should be noted that IRL and SIRL would work just as well in such cases. If memory requirements do not allow storing m Lanczos vectors, then SIRL is specifically recommended, with Ritz vectors written to disk. SIRL is also recommended if one wants to compute all eigenpairs of a matrix, or if the number of wanted eigenpairs (k) is not known beforehand. We have seen that the Lanczos algorithm without reorthogonalization (CWL) is less reliable and slower than both IRL and SIRL (not only for the cardioid problem), and should be used only as a last resort—i.e., if memory requirements are so prohibitive that both IRL and SIRL become impractical—or perhaps if the desired number of eigenvalues is rather small and eigenvectors are not required.

ACKNOWLEDGMENT

We wish to thank the Natural Sciences and Engineering Research Council of Canada for financial support.

APPENDIX: A SIMPLE METHOD FOR OBTAINING LANCZOS EIGENVECTORS

In this section we briefly review an alternative method to inverse iteration for obtaining eigenvectors of a tridiagonal matrix [26]. Consider the eigenvalue equation,

TABLE VI. Approximate number of matrix-vector multiplies for calculating 100 even rovibrational levels of HCN/HNC (N is the number of basis functions).

	$J=0$, $N=16440$	$J=1$, $N=32874$	$J=2$, $N=49268$
IRL	1230	1366	1548
CWL	2500	3500	3600

$$T_m \mathbf{x}_i - \lambda_i \mathbf{x}_i = 0, \tag{A1}$$

which constitutes a homogeneous system of m equations,

$$\beta_{k-1} x_{ik-1} + (\alpha_k - \lambda_i) x_{ik} + \beta_k x_{ik+1} = 0, \quad k = 1, \ldots, m, \tag{A2}$$

where $\beta_0 = \beta_{m+1} = 0$. Of the above m equations only $m-1$ are linearly independent and one is redundant. One way to obtain the eigenvectors of T_m is to set an element of $\mathbf{x}_i$ equal to 1, and use the above recursion relation to obtain the other elements. An obvious choice is to set $x_{i1} = 1$ and use Eq. (A2) with $k=2$ to $m-1$, thereby dropping the last equation as redundant. Another obvious choice would be to set $x_{im} = 1$ and drop the first equation. Although in exact arithmetic any of the m equations can be dropped with no consequence, Wilkinson [38,39] showed that in finite precision arithmetic the m equations are not equally redundant, the computed $\mathbf{x}_i$ is highly sensitive to the choice of which equation to drop and setting $\mathbf{x}_{i1} = 1$ or $\mathbf{x}_{im} = 1$ can lead to catastrophic results, as we also observed frequently in our computations.

Assuming that the jth equation is to be dropped we can rearrange Eq. (A1) as,

$$\begin{bmatrix} \bar{\alpha}_1 & \beta_1 & 0 & \cdots & & 0 \\ \beta_1 & \ddots & \ddots & & & \\ 0 & \ddots & \bar{\alpha}_{j-1} & 0 & \ddots & \vdots \\ & & \beta_{j-1} & 0 & \beta_j & \\ \vdots & & \ddots & 0 & \bar{\alpha}_{j+1} & \ddots \\ & & & & \ddots & \ddots \\ 0 & & \cdots & & 0 & \beta_{m-1} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \\ \vdots \\ \\ \bar{\alpha}_m \end{bmatrix} \begin{bmatrix} x_{i1} \\ \vdots \\ x_{ij-1} \\ 0 \\ x_{ij+1} \\ \vdots \\ x_{im} \end{bmatrix}$$

$$= - \begin{bmatrix} 0 & 0 & & \cdots & & & 0 \\ 0 & \ddots & \ddots & & & & \\ & \ddots & 0 & \beta_{j-1} & & & \\ \vdots & & 0 & \alpha_j & 0 & & \vdots \\ & & & \beta_j & 0 & \ddots & \\ & & & & \ddots & \ddots & 0 \\ 0 & & & \cdots & & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ x_{ij} \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ -\beta_{j-1} x_{ij} \\ -\bar{\alpha}_j x_{ij} \\ -\beta_j x_{ij} \\ \vdots \\ 0 \end{bmatrix},$$

where $\bar{\alpha}_k = \alpha_k - \lambda_i$, $k=1$ to m. The above system of equations reduces to two smaller systems of equations,

$$\begin{bmatrix} \bar{\alpha}_1 & \beta_1 & 0 & \cdots \\ \beta_1 & \ddots & \ddots & \ddots \\ 0 & \ddots & \ddots & \beta_{j-2} \\ & \ddots & \beta_{j-2} & \bar{\alpha}_{j-1} \end{bmatrix} \begin{bmatrix} x_{i1} \\ \vdots \\ \vdots \\ x_{ij-1} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ -\beta_{j-1}x_{ij} \end{bmatrix}, \quad \text{(A3)}$$

and

$$\begin{bmatrix} \bar{\alpha}_{j+1} & \beta_{j+1} & 0 & \cdots \\ \beta_{j+1} & \ddots & \ddots & \ddots \\ 0 & \ddots & \ddots & \beta_{m-1} \\ \vdots & \ddots & \beta_{m-1} & \bar{\alpha}_m \end{bmatrix} \begin{bmatrix} x_{ij+1} \\ \vdots \\ \vdots \\ x_{im} \end{bmatrix} = \begin{bmatrix} -\beta_j x_{ij} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \text{(A4)}$$

coupled via

$$\beta_{j-1}x_{ij-1} + \beta_j x_{ij+1} + \bar{\alpha}_j x_{ij} = 0. \quad \text{(A5)}$$

Setting $x_{ij}=1$, we can drop Eq. (A5) (the jth equation) and solve two inhomogeneous systems of equations for the elements of $\mathbf{x}_i$. In exact arithmetic, Eq. (A5) will be automatically satisfied by the solution to Eqs. (A3) and (A4). However, in finite precision arithmetic the eigenvalues are not known to full precision and in general Eq. (A5) will not be satisfied exactly. Instead we have

$$\beta_{j-1}x_{ij-1} + \beta_j x_{ij+1} + \bar{\alpha}_j x_{ij} = \mu_j(\lambda_i).$$

The most redundant equation, i.e., the optimum value of j, is the one which produces the smallest residual, $|\mu_j|$. Below we summarize a method for obtaining the $\mu_j(\lambda_i)$. The reader is referred to Ref. [26] for proofs and details.

The method is based on the $\mathbf{LDL}^T$ and $\mathbf{UDU}^T$ factorization of the tridiagonal matrix $\mathbf{J}=\mathbf{T}_m-\lambda_i\mathbf{I}$. The $\mathbf{L}$ and $\mathbf{U}$ matrices are lower and upper bidiagonal, respectively, with diagonal elements equal to unity. To implement the method, only the diagonal $\mathbf{D}$ matrices are required. In the $\mathbf{LDL}^T$ factorization the elements of the diagonal $\mathbf{D}$ matrix can be computed from the recursion,

$$d_k = J_{kk} - \frac{J_{kk-1}^2}{d_{k-1}} = (\alpha_k - \lambda_i) - \frac{\beta_{k-1}^2}{d_{k-1}} \quad k = 2, \ldots, m,$$

starting with

$$d_1 = J_{11} = \alpha_1 - \lambda_i.$$

The $\mathbf{UDU}^T$ factorization results in

$$\delta_k = (\alpha_k - \lambda_i) - \frac{\beta_k^2}{\delta_{k+1}} \quad k = m-1, \ldots, 1,$$

for the diagonal elements of $\mathbf{D}$, with

$$\delta_m = \alpha_m - \lambda_i.$$

Once the $\mathbf{D}$ matrices have been computed the following recursion can be used to compute the $\mu_k(\lambda_i)$,

$$\mu_{k+1} = \mu_k \frac{\delta_{k+1}}{d_k}, \quad k = 1, \ldots, m-1, \quad \text{(A6)}$$

and

$$\mu_k = \mu_{k+1} \frac{d_k}{\delta_{k+1}}, \quad k = m-1, \ldots 1, \quad \text{(A7)}$$

with

$$\mu_1 = \delta_1 \text{ and } \mu_m = d_m. \quad \text{(A8)}$$

To compute the eigenvectors of $\mathbf{T}_m$ one would

(1) compute the diagonal $\mathbf{D}$ matrices in the $\mathbf{LDL}^T$ and $\mathbf{UDU}^T$ factorization of $\mathbf{T}_m-\lambda_i\mathbf{I}$,

(2) find the optimum j (i.e., the one resulting in the smallest $|\mu_j|$),

(3) set $x_{ij}=1$, drop the jth equation and solve the remaining system of equations [Eqs. (A3) and (A4)], and

(4) normalize $\mathbf{x}_i$ in the end.

Note that, in step (1), if any of the d_k or δ_k are too small they should be set equal to an ϵ, where ϵ is a threshold of the order of the machine epsilon. Also, in step (2), the best way to solve the system of equations is to take advantage of the $\mathbf{LDL}^T$ and $\mathbf{UDU}^T$ factorizations, and use the following recursions [26].

$$x_{ik} = -\left(\frac{\beta_k}{d_k}\right) x_{ik+1} \quad k = j-1, \ldots, 1,$$

and

$$x_{ik} = -\left(\frac{\beta_{k-1}}{\delta_k}\right) x_{ik-1} \quad k = j+1, \ldots, m,$$

[1] J. K. Cullum and R. A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, Vol. 1: Theory (SIAM, Philadelphia, 2002) .

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. (Johns Hopkins University Press, Baltimore, 1996).

[3] *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, edited by Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. Van der Vorst (SIAM, Philadelphia, 2000).

[4] C. Leforestier and N. Moiseyev, J. Chem. Phys. **103**, 8468 (1995).

[5] H. G. Yu and S. C. Smith, J. Chem. Phys. **107**, 9985 (1997).

[6] X. Wang and T. Carrington, Jr., J. Chem. Phys. **114**, 1473 (2001).

[7] B. Poirier and T. Carrington, Jr., J. Chem. Phys. **116**, 1215 (2002).

[8] J. H. Skone and E. Curotto, J. Chem. Phys. **116**, 3210 (2002).

[9] R. Chen and H. Guo, Chem. Phys. Lett. **369**, 650 (2003).

[10] J. C. Tremblay and T. Carrington, Jr., J. Chem. Phys. **122**, 244107 (2005); **125**, 094311 (2006).

[11] H. Guo, Rev. Comput. Chem. **25**, 285 (2007).

[12] R. Q. Chen and H. Guo, J. Comput. Phys. **136**, 494 (1997).

[13] Note that methods have been developed which can obtain a

small number of eigenvectors without storing Lanczos vectors or redoing the Lanczos iteration. These methods combine Lanczos and the filter diagonalization method. See Ref. [12,30].
[14] W. Karush, Pac. J. Math. **1**, 233 (1951).
[15] G. H. Golub and R. Underwood, in *Mathematical Software III*, edited by J. R. Rice (Academic, New York, 1977), pp. 361–377.
[16] J. K. Cullum, BIT, Nord. Tidskr. Inf.behandl. **18**, 265 (1978).
[17] K. Wu and H. D. Simon, NERSC Lawrence Berkeley National Laboratory, Technical Report No. LBNL-42982, 1999.
[18] K. Wu, A. Canning, and H. D. Simon, NERSC Lawrence Berkeley National Laboratory, Technical Report No. LBNL-42917, 1999.
[19] D. Calvetti, I. Reichel, and D. C. Sorensen, Electron. Trans. Numer. Anal. **2**, 1 (1994).
[20] R. Lehoucq, D. Sorensen, and C. Yang, *ARPACK USERS GUIDE: Solution of Large Scale Eigenvalue Problems With Implicitly Restarted Arnoldi Methods* (SIAM, Philadelphia, 1998).
[21] R. B. Lehoucq and D. C. Sorensen, SIAM J. Matrix Anal. Appl. **17**, 789 (1996).
[22] D. C. Sorensen, Center for Research on Parallel Computation, Rice University, Technical Report No. CRPC-TR98775, 1998.
[23] M. C. Gutzwiller, *Chaos in Classical and Quantum Dynamics* (Springer, New York, 1990), p. 261.
[24] M. Robnik, J. Phys. A **17**, 1049 (1984).
[25] P. Arbenz, U. L. Hetmaniuk, R. B. Lehoucq, and R. S. Tuminaro, Int. J. Numer. Methods Eng. **64**, 204 (2005).
[26] K. V. Fernando, SIAM J. Matrix Anal. Appl. **18**, 1013 (1997).
[27] I. Kosztin and K. Schulten, Int. J. Mod. Phys. C **8**, 293 (1997).
[28] E. J. Heller, Phys. Rev. Lett. **53**, 1515 (1984).
[29] T. Koslowski and W. vonNiessen, J. Comput. Chem. **14**, 769 (1993).
[30] H. Zhang and S. Smith, Phys. Chem. Chem. Phys. **3**, 2282 (2001).
[31] See EPAPS Document No. E-PLEEE8-79-114903 for pseudocode. For more information on EPAPS, see http://www.aip.org/pubservs/epaps.html.
[32] See pp. 501–503 of Ref. [2].
[33] J. Sakr (unpublished).
[34] M. Brack and R. K. Bhaduri, *Semiclassical Physics* (Addison Wesley, New York, 1977).
[35] R. Rajaie Khorasani, Ph.D thesis, McMaster University, 2009.
[36] H.-S. Lee and J. C. Light, J. Chem. Phys. **120**, 4626 (2004).
[37] See, for example, B. Dietz, A. Heine, V. Heuveline, and A. Richter, Phys. Rev. E **71**, 026703 (2005). 3000 eigenvalues for a set of two-dimensional (2D) billiards were computed using a cluster of 64 AMD processors.
[38] J. H. Wilkinson, Comput. J. **1**, 90 (1958).
[39] J. H. Wilkinson, *The Algebraic Eigenvalue Problem* (Clarendon, Oxford, 1695).