Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 194 (2004) 575–587
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Fast methods for resumming matrix polynomials and Chebyshev matrix polynomials

WanZhen Liang [a,b,1], Roi Baer [c,1], Chandra Saravanan [a,1], Yihan Shao [a], Alexis T. Bell [b], Martin Head-Gordon [a,*,1]

[a] Department of Chemistry, University of California, Berkeley, CA 94720, USA
[b] Department of Chemical Engineering, University of California, Berkeley, CA 94720, USA
[c] Department of Physical Chemistry, the Hebrew University, Jerusalem 91904, Israel

Received 26 February 2003; received in revised form 20 August 2003; accepted 28 August 2003

Abstract

Fast and effective algorithms are discussed for resumming matrix polynomials and Chebyshev matrix polynomials. These algorithms lead to a significant speed-up in computer time by reducing the number of matrix multiplications required to roughly twice the square root of the degree of the polynomial. A few numerical tests are presented, showing that evaluation of matrix functions via polynomial expansions can be preferable when the matrix is sparse and these fast resummation algorithms are employed.

1. Introduction

Functions of a matrix can be defined by their series expansions, such as the matrix exponential:

$$e^{\mathbf{X}} = 1 + \mathbf{X} + \frac{1}{2}\mathbf{X}^2 + \frac{1}{3!}\mathbf{X}^3 + \cdots. \tag{1}$$

It is relatively unusual for a matrix function to be evaluated this way because typically it is more efficient and precise to employ the eigenvectors and eigenvalues of $\mathbf{X}$, if it is diagonalizable. For example, if $\mathbf{X}$ is Hermitian, i.e., $\mathbf{X}^{\mathrm{T}} = \mathbf{X}$, we can evaluate the matrix function $e^{\mathbf{X}}$ as

$$e^{\mathbf{X}} = \mathbf{U}e^{x}\mathbf{U}^{\dagger}, \tag{2}$$

where U are the eigenvectors and x the diagonal matrix of eigenvalues of matrix $\mathbf{X}$.

* Corresponding author. Tel.: +1-510-642-5957; fax: +1-510-643-1255.
E-mail address: mhg@cchem.berkeley.edu (M. Head-Gordon).
[1] Those authors contributed equally.

0021-9991/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2003.08.027

However, some situations arise where evaluation of the series is actually preferable. The particular case we have in mind is where the dimension of **X** is very large, and in addition (or as a consequence) **X** is very sparse. Then sparse matrix multiplication methods can be used to evaluate the products entering the matrix polynomial. The resulting computational effort can then in principle increase only linearly with the dimension of the matrix, if its bandwidth is constant. By contrast, explicitly obtaining the eigenvalues and eigenvectors cannot scale linearly – indeed it is cubic scaling if standard linear algebra routines are employed. For example, this situation arises in formulating linear scaling algorithms to perform tight-binding or Kohn–Sham density functional theory calculations on very large molecules [1–7], in evaluation of time-evolution operator e^{-iHt} [8], and the calculation of Boltzmann factors and partition functions of very large molecules in computational physics [9–11].

In this paper, we report fast algorithms for summing matrix and Chebyshev matrix polynomials. The problem is to minimize the computer time required to perform the sum to a given power, N_{pol}, the degree of the polynomial. To an excellent approximation this is equivalent to minimizing the number of matrix multiplications, since the cost of matrix multiplications completely dominates the cost of matrix additions. The other degree of freedom that is available is to store multiple intermediate quantities to permit reduction of the matrix multiplication effort.

Simple term-by-term evaluation of a matrix polynomial of degree N_{pol} requires $N_{\mathrm{pol}} - 1$ matrix multiplications. However it is possible to do substantially better, as has been shown in a number of papers where fast algorithms have been reported for resumming matrix polynomials with fewer matrix multiplications [12–16]. Perhaps the most effective algorithm was suggested by Paterson and Stockmeyer [14], which requires only $O(\sqrt{N_{\mathrm{pol}}})$ nonscalar multiplications to evaluate polynomials of degree N_{pol}. These authors also presented proofs that at least $\sqrt{N_{\mathrm{pol}}}$ nonscalar multiplications are required to resum a polynomial of degree N_{pol}. The algorithm has practical application in the evaluation of matrix polynomials with scalar coefficients and offers huge improvement over simple term-by-term evaluation, at the expense of requiring more stored matrices. Storage of $O(\sqrt{N_{\mathrm{pol}}})$ matrices is required in this algorithm, where N is the dimension of the matrix **X**. Van Loan [15] later showed how this procedure could be implemented with greatly reduced storage, although significantly more matrix multiplies were required.

In the following section (Section 2), we review the algorithm of Paterson and Stockmeyer [14] and identify other two algorithms for matrix polynomial evaluation that also scale with $\sqrt{N_{\mathrm{pol}}}$, the square root of the maximum degree of the polynomial. The coefficient is approximately 2. These two relatively distinct algorithms turn out to yield very similar matrix multiplication counts but require fewer stored matrices than Paterson and Stockmeyer's algorithm.

Another purpose of the paper is to generalize these algorithms for summing simple matrix polynomials to accelerate the summation matrix series based on Chebyshev matrix polynomials of the form

$$f(\mathbf{X}) = \sum_{i=0}^{N_{\mathrm{pol}}} a_i T_i(\mathbf{X}). \tag{3}$$

$T_i(\mathbf{X})$ is the Chebyshev matrix polynomial of degree i, which is recursively defined in terms of T_{i-1} and T_{i-2}, and $a_0, a_1, \ldots, a_N$ are the scalar coefficients. If the eigenvalues of **X** lie in the range $[-1, 1]$, then Chebyshev approximations are numerically very stable and they are less susceptible to round-off errors due to limited machine precision than the equivalent power series [17]. In the context of linear scaling electronic structure methods, Chebyshev matrix polynomials have therefore been proposed as a more stable and efficient alternative to simple matrix polynomials [3,18,19]. These generalizations are described in Section 3. They yield algorithms that are broadly similar to the ones described for matrix polynomials, with differences arising from the recursion relations associated with Chebyshev matrix polynomials. Detailed application of this approach to linear scaling electronic structure calculations based on Chebyshev polynomials is described elsewhere [20].

In Section 4, we present three types of data to characterize the performance of the 3 fast summation algorithms. First, the number of matrix multiplies, as well as the number of stored intermediates, are reported for polynomials of various degrees. Second, computer time for the evaluation of fast summations is compared with the time for conventional term-by-term summation on some test matrices. Third, the computer time for evaluating matrix polynomials by fast summation is compared with evaluation by explicit diagonalization for the case where the matrix is large and sparse. Finally we conclude in Section 5.

2. Fast methods for resumming matrix polynomials

Three kinds of algorithms are presented to effectively resum matrix polynomials in this section. The central strategy of these algorithms is the hierarchical decomposition of matrix polynomials into many blocks and to reuse multiple intermediate quantities to reduce the matrix multiplications.

2.1. Algorithm I – recursive binary subdivision

Algorithm I is based on recursive binary subdivision of a polynomial. It is achieved by the following two steps.

The first step is to decompose the matrix polynomials according to the hierarchical structure shown in Fig. 1. We begin at level 0 with a matrix polynomial of the form $\sum_{i=0}^{N_{\text{pol}}} a_i \mathbf{X}^i$. At level 1, it is divided into two polynomials, where one involves only even polynomial terms and the other involves only odd polynomial terms:

$$\sum_{i=0}^{N_{\text{pol}}} a_i \mathbf{X}^i = \sum_{n=0}^{N_{\text{even}}} a_{2n} \mathbf{X}^{2n} + \sum_{n=0}^{N_{\text{odd}}} a_{2n+1} \mathbf{X}^{2n+1} = \sum_{n=0}^{N_{\text{even}}} a_{2n} \mathbf{X}_1^n + \mathbf{X} \sum_{n=0}^{N_{\text{odd}}} a_{2n+1} \mathbf{X}_1^n. \tag{4}$$

Here $N_{\text{odd}} + N_{\text{even}} = N_{\text{pol}} - 1$, $\mathbf{X}_n = (\mathbf{X}_{n-1})^2$ and $\mathbf{X}_0 = \mathbf{X}$. Thus the two subpolynomials each advance in powers of $\mathbf{X}_1 = \mathbf{X}^2$, rather than $\mathbf{X}$. This reduces the number of matrix multiplications by almost a factor of two. One may continue to divide each subpolynomial into 2 subsubpolynomials at level 2, each advancing in powers of $\mathbf{X}_2 = \mathbf{X}^4$, and so on. If the maximum level of division is ML, the polynomial is divided into 2^{ML} subpolynomials, where the stopping point ML is determined by a minimum in the number of matrix multiplications.

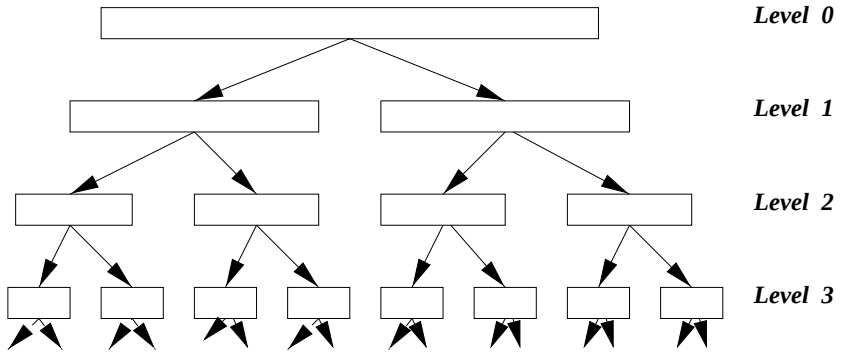

Fig. 1. Decomposition of a polynomial by Algorithm I.

To see this explicitly, we show the first few divisions for a polynomial of degree $N_{\rm pol} = n2^{ML} - 1$, where n is integer. This is a particularly simple choice since it ensures that each intermediate subpolynomial is exactly the same length. The top level, 0, has $n2^{ML}$ terms, the next level, 1, has $n2^{ML-1}$ terms, and so on until the lowest level (ML) for which each subpolynomial has n terms. Thus:

$$\begin{aligned} f(\mathbf{X}) &= \sum_{i=0}^{n2^{ML}-1} a_i \mathbf{X}^i = \sum_{i=0}^{n2^{ML-1}-1} a_{2i}\mathbf{X}_1^i + \mathbf{X}\left\{ \sum_{i=0}^{n2^{ML-1}-1} a_{2i+1}\mathbf{X}_1^i \right\} \\ &= \sum_{i=0}^{n2^{ML-2}-1} a_{4i}\mathbf{X}_2^i + \mathbf{X}_1 \sum_{i=0}^{n2^{ML-2}-1} a_{4i+2}\mathbf{X}_2^i + \mathbf{X}\left\{ \sum_{i=0}^{n2^{ML-2}-1} a_{4i+1}\mathbf{X}_2^i + \mathbf{X}_1 \sum_{i=0}^{n2^{ML-2}-1} a_{4i+3}\mathbf{X}_2^i \right\} \\ &= \sum_{i=0}^{n2^{L-3}-1} a_{8i}\mathbf{X}_3^i + \mathbf{X}_2 \sum_{i=0}^{n2^{ML-3}-1} a_{8i+4}\mathbf{X}_3^i + \mathbf{X}_1\left\{ \sum_{i=0}^{n2^{ML-3}-1} a_{8i+2}\mathbf{X}_3^i + \mathbf{X}_2 \sum_{i=0}^{n2^{ML-3}-1} a_{8i+6}\mathbf{X}_3^i \right\} \\ &\quad + \mathbf{X}\left\{ \sum_{i=0}^{n2^{ML-3}-1} a_{8i+1}\mathbf{X}_3^i + \mathbf{X}_2 \sum_{i=0}^{n2^{ML-3}-1} a_{8i+5}\mathbf{X}_3^i + \mathbf{X}_1\left\{ \sum_{i=0}^{n2^{ML-3}-1} a_{8i+3}\mathbf{X}_3^i + \mathbf{X}_2 \sum_{i=0}^{n2^{ML-3}-1} a_{8i+7}\mathbf{X}_3^i \right\}\right\} \\ &= \cdots . \end{aligned} \tag{5}$$

The second step is to evaluate the intermediate matrices, from which we can assemble the subpolynomials (with no extra multiplications for the subpolynomials), and then put back all subpolynomials together. From Eq. (5), we can infer that Algorithm I requires us to calculate and store

$$SM = \mathrm{int}\left(\frac{N_{\rm pol}+1}{2^{ML}}\right) + \mathrm{res} + ML - 2 \tag{6}$$

matrices before putting-back-together 2^{ML} subpolynomials ($\mathbf{X}$ is excluded in this count). Here res = 1, if $N_{\rm pol} + 1 > 2^{ML} \times \mathrm{int}((N_{\rm pol}+1)/2^{ML})$, otherwise res = 0. The stored matrices include a total of ML matrices X_n with $X_n = X_{n-1}^2$ ($n = 1, 2, \ldots, ML$), and a total of SM–ML matrices $\mathbf{X}_{ML}^i \mathbf{X}_{ML}$ ($i = 1, 2, \ldots, SM$–ML). A total of $2^{ML} - 1$ matrix multiplications are required in putting-back-together the 2^{ML} subpolynomials. Thus, for a polynomial of degree $N_{\rm pol}$, Algorithm I requires

$$\langle M \rangle = \mathrm{int}\left(\frac{N_{\rm pol}+1}{2^{ML}}\right) + \mathrm{res} + ML + 2^{ML} - 3 \tag{7}$$

matrix multiplications. $\langle M \rangle$ can be approximately minimized with the choice:

$$ML = \mathrm{int}\left(\frac{P}{2}\right). \tag{8}$$

However, this requires to save large number of matrices, and we find that a better compromise between $\langle M \rangle$ and SM is to use:

$$ML = \begin{cases} \mathrm{int}\left(\frac{P+1}{2}\right) & \text{if } N_{\rm pol}+1 = 2^P, \\ \mathrm{int}\left(\frac{P}{2}\right) + 1 & \text{if } N_{\rm pol}+1 > 2^P, \end{cases} \tag{9}$$

where

$$P = \mathrm{int}(\log_2(N_{\rm pol}+1)). \tag{10}$$

As an example, we resum the polynomial of degree $N_{\rm pol} = 31$. Applying our general expressions above, we see from Eq. (10) that $P = 5$, and thus that the optimum choice for the maximum level of division, ML, is 3. Accordingly, the polynomial is divided three times as

$$\sum_{i=0}^{31} a_i \mathbf{X}^i = \sum_{i=0}^{15} a_{2i}\mathbf{X}_1^i + \mathbf{X}\left\{\sum_{i=0}^{15} a_{2i+1}\mathbf{X}_1^i\right\} = \sum_{i=0}^{7} a_{4i}\mathbf{X}_2^i + \mathbf{X}_1\sum_{i=0}^{7} a_{4i+2}\mathbf{X}_2^i + \mathbf{X}\left\{\sum_{i=0}^{7} a_{4i+1}\mathbf{X}_2^i + \mathbf{X}_1\sum_{i=0}^{7} a_{4i+3}\mathbf{X}_2^i\right\}$$
$$= \sum_{i=0}^{3} a_{8i}\mathbf{X}_3^i + \mathbf{X}_2\sum_{i=0}^{3} a_{8i+4}\mathbf{X}_3^i + \mathbf{X}_1\left\{\sum_{i=0}^{3} a_{8i+2}\mathbf{X}_3^i + \mathbf{X}_2\sum_{i=0}^{3} a_{8i+6}\mathbf{X}_3^i\right\} + \mathbf{X}\left\{\sum_{i=0}^{3} a_{8i+1}\mathbf{X}_3^i + \mathbf{X}_2\sum_{i=0}^{3} a_{8i+5}\mathbf{X}_3^i + \mathbf{X}_1\left\{\sum_{i=0}^{3} a_{8i+3}\mathbf{X}_3^i + \mathbf{X}_2\sum_{i=0}^{3} a_{8i+7}\mathbf{X}_3^i\right\}\right\}. \quad (11)$$

Eq. (11) shows that the polynomial can be evaluated with a total of 12 matrix multiplications, as also predicted by Eq. (7). In detail, the matrix multiplications are:

Five intermediate matrices $\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$, $\mathbf{X}_3^2$ and $\mathbf{X}_3^3$, another 7 matrix multiplications caused by putting-back-together 8 subpolynomials:

$$F_1 = \sum_{i=0}^{3} a_{8i}\mathbf{X}_3^i + \mathbf{X}_2\sum_{i=0}^{3} a_{8i+4}\mathbf{X}_3^i,$$
$$F_2 = \sum_{i=0}^{3} a_{8i+2}\mathbf{X}_3^i + \mathbf{X}_2\sum_{i=0}^{3} a_{8i+6}\mathbf{X}_3^i,$$
$$F_3 = F_1 + \mathbf{X}_1 F_2,$$
$$F_4 = \sum_{i=0}^{3} a_{8i+1}\mathbf{X}_3^i + \mathbf{X}_2\sum_{i=0}^{3} a_{8i+5}\mathbf{X}_3^i,$$
$$F_5 = \sum_{i=0}^{3} a_{8i+3}\mathbf{X}_3^i + \mathbf{X}_2\sum_{i=0}^{3} a_{8i+7}\mathbf{X}_3^i,$$
$$F_6 = F_4 + \mathbf{X}_1 F_5,$$
$$f(\mathbf{X}) = F_3 + \mathbf{X}F_6.$$

2.2. Algorithm II – separate subdivision

It is possible that binary subdivision, while very simple and clearly effective, may not be completely optimal. An alternative that would provide additional flexibility is to initially subdivide the matrix polynomial not into 2 but instead into K separate series. These K subpolynomials could then be further treated by binary subdivision. Specifically we shall define the maximum level of subdivision of the subpolynomial as $ml = \mathrm{int}(\overline{p}/2)$ where $\overline{p} = \log_2 \mathrm{int}(N_{\mathrm{pol}}/K)$, for an initial K-fold subdivision. If K is set to equal to 2, the algorithm is same as Algorithm I. If K is set to equal to 3, we shall define the resulting approach as Algorithm II, and we shall investigate for what degrees of polynomial (if any) it improves upon Algorithm I.

Algorithm II is achieved by the following three steps. In the initial step, we divide the matrix polynomial of degree N_{pol} into three subpolynomials f_1, f_2, f_3 with N_1, N_2 and N_3 terms, respectively, i.e.,

$$f(\mathbf{X}) = f_1(\mathbf{X}) + f_2(\mathbf{X}) + f_3(\mathbf{X}) = \sum_{i=0}^{N_1-1} \overline{a}_i\mathbf{X}^i + \mathbf{X}^{N_1}\left\{\sum_{i=0}^{N_1-1} \overline{b}_i\mathbf{X}^i + \mathbf{X}^{N_1}\sum_{i=0}^{N_3-1} \overline{c}_i\mathbf{X}^i\right\}, \quad (12)$$

where $\overline{a}_i = a_i$, $\overline{b}_i = a_{N_1+i}$ and $\overline{c}_i = a_{2N_1+i}$. $N_1 = N_2 = 2^{ml} \times \mathrm{int}((N_{\mathrm{pol}}+1)/3 \times 2^{ml})$, $N_3 = N_{\mathrm{pol}} + 1 - N_1 - N_2$, $ml = \mathrm{int}(\overline{p}/2)$ and $\overline{p} = \mathrm{int}(\log_2(N_1))$.

In the second step, the three subpolynomials $\sum_{i=0}^{N_1-1} \overline{a}_i \mathbf{X}^i$, $\sum_{i=0}^{N_1-1} \overline{b}_i \mathbf{X}^i$ and $\sum_{i=0}^{N_3-1} \overline{c}_i \mathbf{X}^i$ are separately resummed by Algorithm I. The maximum level of subdivision for these three subpolynomials is ml. The third step is to add all subpolynomials together.

As for Algorithm I we evaluate and store the necessary intermediate matrices and then assemble and add all subpolynomials together. The total number of matrix multiplications for a polynomial of degree N_{pol} by Algorithm II should be

$$\langle M \rangle = \mathrm{int}(\max(N_1, N_3)/2^{ml}) + ml + 3(2^{ml} - 1). \tag{13}$$

There are a total of

$$SM = \mathrm{int}(\max(N_1, N_3)/2^{ml}) + ml - 2 \tag{14}$$

intermediate matrices that need to be stored in this approach. The last $2 + 3(2^{ml} - 1)$ matrix multiplications are required in putting back together all the subpolynomials. The value of $\langle M \rangle$ is slightly less than that required by Algorithm I when $N_{\mathrm{pol}} + 1$ lies outside the interval between $3 \times 2^{P-1}$ and $2^{2\mathrm{int}(\frac{P+1}{2})}$ (P is defined by Eq. (10)). If $N_{\mathrm{pol}} + 1$ lies in that range, then Algorithm II offers no advantage, and we employ Algorithm I to resum the matrix polynomial.

As an example, we resum the polynomial of degree 31 by Algorithm II. In the initial step involving 3-fold subdivision, the length of the first two subpolynomials, N_1, is equal to 10. Accordingly each of these subpolynomials will undergo binary subdivision once, as follows from $\overline{p} = 3$ and $ml = 1$. Explicitly, we have:

$$\begin{aligned}
\sum_{i=0}^{31} a_i \mathbf{X}^i &= \sum_{i=0}^{9} \overline{a}_i \mathbf{X}^i + \mathbf{X}^{10} \left\{ \sum_{i=0}^{9} \overline{b}_i \mathbf{X}^i \right\} + \mathbf{X}^{20} \left\{ \sum_{i=0}^{11} \overline{c}_i \mathbf{X}^i \right\} \\
&= \sum_{i=0}^{4} \overline{a}_{2i} \mathbf{X}_1^i + \mathbf{X} \sum_{i=0}^{4} \overline{a}_{2i+1} \mathbf{X}_1^i + \mathbf{X}^{10} \left\{ \sum_{i=0}^{4} \overline{b}_{2i} \mathbf{X}_1^i + \mathbf{X} \sum_{i=0}^{4} \overline{b}_{2i+1} \mathbf{X}_1^i \right. \\
&\quad \left. + \mathbf{X}^{10} \left\{ \sum_{i=0}^{5} \overline{c}_{2i} \mathbf{X}_1^i + \mathbf{X} \sum_{i=0}^{5} \overline{c}_{2i+1} \mathbf{X}_1^i \right\} \right\}.
\end{aligned} \tag{15}$$

Eq. (15) shows we only need to calculate a total of 10 matrix multiplications, which is less than that required by Algorithm I. 5 intermediate matrices are required: $\mathbf{X}_1$, $\mathbf{X}_1^2$, $\mathbf{X}_1^3$, $\mathbf{X}_1^4$, and $\mathbf{X}_1^5$.

2.3. Algorithm III – $\sqrt{N_{pol}+1} \times \sqrt{N_{pol}+1}$ scheme

Algorithm III for resumming matrix polynomials is similar to that first described in [14]. A short description of the algorithm is given here. The basic idea is instead of performing repeated subdivisions, as in Algorithms I and II, we perform just a single subdivision into a set of SP subpolynomials, which we then reassemble as efficiently as possible. It requires three steps to be performed.

At first we divide the polynomial of degree N_{pol} into SP subpolynomials ($SP = \mathrm{int}(\sqrt{N+1}) + \mathrm{res}$), i.e.,

$$f(\mathbf{X}) = \sum_{m=0}^{N} a_m \mathbf{X}^m = \sum_{i=0}^{N_1-1} \overline{a}_i^0 \mathbf{X}^i + \sum_{n=2}^{SP} \mathbf{X}^{(n-1)\times nl} \sum_{i=1}^{N_n} \overline{a}_i^{n-1} \mathbf{X}^i. \tag{16}$$

Each subpolynomial except the first and last one includes nl terms ($N_n = nl = \mathrm{int}(N + 1/\sqrt{N+1})$, $1 < n < SP$). The first one and last one include $nl + 1$ and $N - (SP-1) \times nl$ terms, respectively. Here

res = 1 if $N > nl \times \mathrm{int}(\sqrt{N+1})$, otherwise res = 0. To reduce the evaluation of matrix multiplications, Eq. (16) can be written in a more efficient way by Horner's rule as shown in Ref. [14]

$$f(\mathbf{X}) = \sum_{i=0}^{N_1-1} \overline{\overline{a}}_i^0 \mathbf{X}^i + \mathbf{X}^{nl}\left\{ \sum_{i=1}^{N_2} \overline{\overline{a}}_i^1 \mathbf{X}^i + \mathbf{X}^{nl}\left\{ \sum_{i=1}^{N_3} \overline{\overline{a}}_i^2 \mathbf{X}^i + \cdots + \mathbf{X}^{nl}\left\{ \sum_{i=1}^{N_{SP}} \overline{\overline{a}}_i^{SP-1} \mathbf{X}^i \right\} \right\} \right\}. \tag{17}$$

To evaluate this expression we first calculate $nl-1$ intermediate matrices $\mathbf{X}^2, \mathbf{X}^3, \ldots, \mathbf{X}^{nl}$ at the cost of a single matrix multiply apiece. These matrices needed to be stored, i.e.,

$$SM = nl - 1. \tag{18}$$

An additional $SP-1$ matrix multiplications are required in the process of constructing the subpolynomials which are added together to complete evaluation of Eq. (17), for a total of:

$$M = nl + SP - 2 \tag{19}$$

matrix multiplications. Minimizing the value of $\langle M \rangle$ is the same as minimizing $nl + SP$. This is achieved by the choice $SP \sim \sqrt{N_{\mathrm{pol}}+1}$.

The coefficients entering Eq. (17) can be easily obtained from the input coefficients, a_i, as $\overline{a}_i^n = a_{i+n\times nl}$, and $\overline{\overline{a}}_i^n = \overline{a}_i^n$. Van Loan [15] shows how the procedure can be implemented without storage arrays for $\mathbf{X}^2, \mathbf{X}^3, \ldots, \mathbf{X}^{nl}$. However this involves roughly 50 and therefore is only preferable if there is not enough memory to hold the intermediates.

To resum the polynomial of degree 31 by Algorithm III, we divide it into 6 subpolynomials and reorganize them as

$$\begin{aligned} f(\mathbf{X}) &= \sum_{i=0}^{31} a_i \mathbf{X}^i \\ &= \sum_{i=0}^{6} \overline{a}_i^0 \mathbf{X}^i + \mathbf{X}^{\mathbf{6}} \sum_{i=1}^{6} \overline{a}_i^1 \mathbf{X}^i + \mathbf{X}^{12} \sum_{i=1}^{6} \overline{a}_i^2 \mathbf{X}^i + \mathbf{X}^{18} \sum_{i=1}^{6} \overline{a}_i^3 \mathbf{X}^i + \mathbf{X}^{24} \sum_{i=1}^{6} \overline{a}_i^4 \mathbf{X}^i + \mathbf{X}^{30} \sum_{i=1}^{1} \overline{a}_i^5 \mathbf{X}^i \\ &= \sum_{i=0}^{6} \overline{\overline{a}}_i^0 \mathbf{X}^i + \mathbf{X}^6 \left\{ \sum_{i=1}^{6} \overline{\overline{a}}_i^1 \mathbf{X}^i + \mathbf{X}^6 \left\{ \sum_{i=1}^{6} \overline{\overline{a}}_i^2 \mathbf{X}^i + \mathbf{X}^{\mathbf{6}} \left\{ \sum_{i=1}^{6} \overline{\overline{a}}_i^3 \mathbf{X}^i \right.\right.\right. \\ &\quad \left.\left.\left. + \mathbf{X}^{\mathbf{6}} \sum_{i=1}^{6} \overline{\overline{a}}_i^4 \mathbf{X}^i + \mathbf{X}^{\mathbf{6}} \sum_{i=1}^{1} \overline{\overline{a}}_i^5 \mathbf{X}^i \right\} \right\} \right\}. \end{aligned} \tag{20}$$

It requires 10 matrix multiplications with the optimal choice of $nl = 6$ and $SP = 6$. These multiplications include forming 5 matrices $\mathbf{X}^2$, $\mathbf{X}^3$, $\mathbf{X}^4$, $\mathbf{X}^5$,$\mathbf{X}^6$ and another 5 matrix multiplications to construct and put back together all 6 subpolynomials.

3. Fast algorithms for resumming Chebyshev matrix polynomials

In many numerical applications, it is preferable to use Chebyshev matrix polynomials rather than simple matrix polynomials because of their superior numerical stability. Therefore it is of practical interest to generalize the fast methods for resumming a simple matrix polynomial discussed above, so that they can be employed to resum Chebyshev matrix polynomials. The key difference is that because higher Chebyshev polynomials are defined in terms of lower ones by a two-term recurrence relation rather than a simple matrix multiplication, the scalar coefficients in the reorganized subpolynomials will now have more complicated definitions in terms of the original coefficients modified by related recurrences.

We will illustrate how the coefficients change here for the three algorithms discussed above, one by one. The Chebyshev (matrix) polynomial of degree i is denoted as T_i and is defined by the following recursion relations [17]

$$\begin{aligned} T_0(\mathbf{X}) &= \mathbf{I} \\ T_1(\mathbf{X}) &= \mathbf{X} \\ T_2(\mathbf{X}) &= 2\mathbf{X}^2 - 1 \\ T_{n+m}(\mathbf{X}) &= 2T_{\mathrm{n}}(\mathbf{X})T_m(\mathbf{X}) - T_{|n-m|}(\mathbf{X}). \end{aligned} \tag{21}$$

To first see how the coefficients of the expansion and polynomials will be changed during the division, let us begin with the first binary subdivision of a Chebyshev matrix polynomial. The result will be the generalization of Eq. (4) from matrix polynomials to Chebyshev matrix polynomials. We obtain:

$$\sum_{i=0}^{N_{\mathrm{pol}}} a_i T_i(\mathbf{X}) = \sum_{n=0}^{N_{\mathrm{even}}} a_{2n} T_{2n}(\mathbf{X}) + \sum_{n=0}^{N_{\mathrm{odd}}} a_{2n+1} T_{2n+1}(\mathbf{X}) = \sum_{n=0}^{N_{\mathrm{even}}} a_{2n} T_{\mathrm{n}}(\mathbf{X}_1) + T_1(\mathbf{X}) \sum_{n=0}^{N_{\mathrm{odd}}} a'_n T_{\mathrm{n}}(\mathbf{X}_1). \tag{22}$$

Here we have redefined the $\mathbf{X}_n$ as:

$$\mathbf{X}_n = 2\mathbf{X}_{n-1} - 1 \tag{23}$$

where $\mathbf{X}_0 = \mathbf{X}$. Eqs. (22) and (23) follow directly from the recursion relation for Chebyshev polynomials:

$$T_{2n}(\mathbf{X}) = T_n(2\mathbf{X}^2 - 1). \tag{24}$$

The modified coefficients a'_n can also be found from those a_n by recurrence, with the result being:

$$\begin{aligned} a'_{N_{\mathrm{odd}}} &= 2a_{2N_{\mathrm{odd}}+1}, \\ a'_n &= 2a_{2n+1} - a'_{n+1}, \quad n = N_{\mathrm{odd}} - 1, \ldots, 1, \\ a'_0 &= a_1 - \frac{a'_2}{2}. \end{aligned} \tag{25}$$

These relations are employed at each level of subdivision that is used, up to the optimal level ML. There is a 1:1 mapping between the number of matrix multiplies required to implement this evaluation of the Chebyshev matrix polynomial, and the simple matrix polynomial, because evaluation of each of the redefined $\mathbf{X}_n$ still requires 1 matrix multiply to construct from Eq. (23).

It is advantageous to employ Algorithm II to resum a Chebyshev polynomial of degree N_{pol} when it lies outside the interval between $3 \times 2^{P-1}$ and $2^{2\mathrm{int}((P+1)/2)}$. The first step in Algorithm II is to divide the polynomial into three subpolynomials as

$$f(\mathbf{X}) = f_1(\mathbf{X}) + f_2(\mathbf{X}) + f_3(\mathbf{X}) = \sum_{i=0}^{N_3-1} \overline{a}_i T_i(\mathbf{X}) + T_{N_1}(\mathbf{X}) \left\{ \sum_{i=0}^{N_1-1} \overline{b}_i T_i(\mathbf{X}) + T_{N_1}(\mathbf{X}) \sum_{i=0}^{N_3-1} \overline{c}_i T_i(\mathbf{X}) \right\}. \tag{26}$$

Here

$$\overline{c}_i = \begin{cases} 4a_{2N_1+i}, & 0 < i < N_3, \\ 2a_{2N_1}, & i = 0, \end{cases}$$

$$\overline{b}_i = \begin{cases} 2a_{N_1+i} - \frac{1}{2}\overline{c}_{N_1-i}, & 0 < i < N_2, \\ a_{N_1} - \frac{1}{4}\overline{c}_{N_1}, & i = 0, \end{cases}$$

$$\overline{a}_i = \begin{cases} -2a_{N_1+N_2+i}, & N_1 \leqslant i < N_3, \\ 2a_i - \frac{1}{2}\overline{b}_{N_1-i} - \frac{1}{4}\overline{c}_{2N_1-i} - 2a_{N_1+N_2+i}, & 0 < i < N_1, \\ a_0 - \frac{1}{2}\overline{c}_{2N_1} - a_{2N_1} & i = 0 \end{cases}$$

and $\overline{c}_i = 0$ if $i > N_3 - 1$. The sequential steps for resumming three subpolynomials are similar to Algorithm I.

To resum a Chebyshev polynomial of degree N_{pol} by Algorithm III, the first step is to divide a Chebyshev polynomial into SP subpolynomials as

$$\begin{aligned} f(\mathbf{X}) &= \sum_{n=1}^{SP} f_n(\mathbf{X}) = \sum_{i=0}^{N_1-1} \overline{c}_i^0 T_i(\mathbf{X}) + \sum_{n=2}^{SP} T_{(n-1)\times nl}(\mathbf{X}) \sum_{i=1}^{N_n} \overline{c}_i^{n-1} T_i(\mathbf{X}) \\ &= \sum_{i=0}^{N_1-1} \overline{\overline{c}}_i^0 T_i(\mathbf{X}) + T_{nl}(\mathbf{X}) \left\{ \sum_{i=1}^{N_2} \overline{\overline{c}}_i^1 T_i(\mathbf{X}) + T_{nl}(\mathbf{X}) \left\{ \sum_{i=1}^{N_3} \overline{\overline{c}}_i^2 T_i(\mathbf{X}) + \cdots + T_{nl}(\mathbf{X}) \left\{ \sum_{i=1}^{N_{SP}} \overline{\overline{c}}_i^{SP-1} T_i(\mathbf{X}) \right\} \right\} \right\}. \end{aligned} \tag{27}$$

Here N_n is the length of the nth subpolynomial. The coefficients change and can be evaluated according to the following recurrences

$$\overline{c}_i^n = \begin{cases} 2c_{(SP-1)\times nl+i}, & n = SP-1 \text{ and } 1 \leqslant i \leqslant N_{SP}, \\ 2c_{n\times nl+i} - \overline{c}_{nl-i}^{n+1}, & 1 \leqslant n < SP-1 \text{ and } 1 \leqslant i < nl, \\ c_i - \frac{1}{2}\overline{c}_{nl-i}^1, & n = 0 \text{ and } 0 \leqslant i < nl, \end{cases}$$

and $\overline{c}_{nl}^n = 2c_{n\times nl+1} - \overline{c}_{nl}^{n+2}$ ($1 < n < SP-2$), $\overline{c}_{nl}^0 = c_{nl} - \frac{1}{2}\overline{c}_{nl}^2$, and $\overline{c}_i^{SP-1} = 0$ ($i > N_{SP}$). The coefficients $\overline{\overline{c}}_i^n$ can be calculated by the following relation

$$\overline{\overline{c}}_i^n = \sum_{m \geqslant n}^{SP-1} \overline{c}_i^m A_{n,m}. \tag{28}$$

Here A is an $SP \times SP$ matrix, which is evaluated according to

$$A_{m,n} = \begin{cases} 2A_{m-1,m-1} & \text{if } m = n, \\ 2A_{m-1,n-1} - A_{m,n-2} & \text{if } m \neq n \text{ and } m \leqslant n-2, \ldots, 0 \end{cases}$$

if $n \geqslant 3$, and $A_{0,0} = 1$, $A_{1,1} = 1$, $A_{2,2} = 2$, $A_{0,2} = -1$, and all other elements of the matrix A are equal to 0. The number of matrix multiplications required to resum a Chebyshev polynomial is equal to that for resumming a matrix polynomial of the same degree, as for the other two algorithms.

4. Results and discussion

As the first step in demonstrating the performance of the fast summation methods, we focus on the numbers of matrix multiplications, $\langle M \rangle$, and the number of intermediate matrices that must be saved, SM, for each algorithm, for a variety of polynomial degrees, N_{pol}. This data is presented in Table 1. We observe that the values of $\langle M \rangle$ and SM are far vastly less than $N_{\text{pol}} + 1$. Indeed it is apparent that the value of $\langle M \rangle$ in all three algorithms is approximately equal to $\sim 2\sqrt{N_{\text{pol}} + 1}$ for polynomials of high degree.

The values of SM, while smaller in magnitude, are also scaling the same way, showing that the extent of the price that is being paid for reduced numbers of matrix multiplications in terms of increased storage. Comparing the value of $\langle M \rangle$ in the three algorithms shows that Algorithm III requires the smallest number of matrix multiplications. However, it requires storing more intermediate matrices than Algorithm I or II.

Table 1
The polynomial length $N_{\mathrm{pol}}+1$, maximum level of division ML, total number of matrix multiplications $\langle M\rangle$ and the number of storing matrices SM are listed

$N_{\mathrm{pol}}+1$	Algorithm I			Algorithm II			Algorithm III		
	ML	SM	$\langle M\rangle$	ml	SM	$\langle M\rangle$	SP	SM	$\langle M\rangle$
16	2	4	7				4	3	6
128	4	10	25	2	11	22	12	10	21
180	4	14	29	2	16	27	14	12	25
256	4	18	33				16	15	30
336	5	14	45	3	16	39	19	17	35
512	5	19	50	3	23	46	24	21	44
696	5	25	56	3	31	54	27	25	51
1024	5	35	66				32	31	62
1600	6	29	93	4	36	83	40	39	78

Comparing Eqs. (7) and (13) for Algorithms I and II, we note that the value of $\langle M\rangle$ in Algorithm II is reduced from Algorithm I via decreasing the number of matrix multiplications that are due to the subdivision of polynomials. It requires $2^{ML}-1$ matrix multiplications in Algorithm I because of ML subdivisions while it requires $2+3(2^{ml}-1)$ matrix multiplications in Algorithm II. Usually, the difference between the values of ML and ml is 2 as Table 1 shows. Thus the value of $2^{ML}-1$ is larger than $2+3(2^{ml}-1)$.

Comparing Eqs. (7) and (19) for Algorithms I and III, we see that the number of matrix multiplications associated with subdivision of the polynomial is further reduced in Algorithm III. The polynomial of degree N_{pol} is divided into 2^{ML} subpolynomials in Algorithm I while it is divided into $\sim\sqrt{N_{\mathrm{pol}}+1}$ subpolynomials in Algorithm III. The value of $2^{ML}-1$ in Algorithm I is larger than $\sqrt{N_{\mathrm{pol}}+1}-1$ in Algorithm III. However, associated with the smaller number of subdivisions in Algorithm III is a larger value of SM, and thus more storage is required in Algorithm III.

We turn next to the CPU timings for evaluation of matrix polynomials using the fast summation algorithms. In this first set of tests, summarized in Table 2, we use dense matrices, and compare the resummation methods against simple term by term evaluation. Our test polynomial is the matrix function $f(\mathbf{X})=\frac{1}{1-\mathbf{X}}$ expressed as $f(\mathbf{X})=\sum_{n=0}^{N_{\mathrm{pol}}}\mathbf{X}^n$. While the details are not too important here (they are more important later when we consider sparsity), our test matrices $\mathbf{X}$ are from electronic structure theory. They are the effective Hamiltonians, $\mathbf{F}$, from converged mean theory calculations in a minimal STO-3G basis set for two alkane oligomers of size $C_{60}H_{122}$ and $C_{120}H_{242}$ This yields matrices of dimension 422 and 842 respectively. The electronic structure calculations were performed with a development version of the Q-Chem program package [21], and the CPU timings were obtained on a 375 MHZ IBM Power-3 workstation (Model 270).

Table 2
The CPU time for resumming of matrix polynomials by fast algorithms and the conventional term-by-term algorithm

N_{pol} +1	$N=422$				$N=842$			
	T_{c}	T_{n}			T_{c}	T_{n}		
		I	II	III		I	II	III
180	23.27	4.54	4.42	4.21	184.50	34.17	33.21	31.58
256	33.15	5.32		5.26	262.58	39.92		38.37
480	61.85	8.62	8.19	7.87	494.04	63.01	59.09	55.67
1024	132.69	13.0		12.81	1054.14	92.04		88.53

T_{c} and T_{n} are CPU times by conventional term-by-term algorithm and new algorithms I or II and III, respectively. CPU time is in seconds (on IBM RS/6000, 375 MHZ POWER III). N is the dimension of the matrix.

Table 2 records the CPU times for matrix polynomial evaluation, which are denoted as T_n in the fast algorithms and T_c in the simple term-by-term algorithm for these two matrices as a function of polynomial degree. We note that the ratio (T_c/T_n) grows with the degree of the polynomial as approximately proportional to the ratio $N_{pol} - 1/\langle M \rangle$. Clearly the CPU time for resumming these matrix polynomials is dominated by the (cubic scaling) matrix multiply time, as the number of (quadratic scaling) matrix additions has not been altered. Overall, large computational savings are achieved for polynomials of high degree. However, the scaling of CPU time with the matrix dimension in the new algorithms is the same as that in the conventional term-by-term resumming method, and so generally explicit evaluation of the matrix function in terms of the eigenvectors and eigenvalues would still be computationally preferable.

This situation potentially changes when the matrix (which the function or polynomial depends on) exhibits a high degree of sparsity. This happens in the electronic structure examples mentioned above [22]. The effective Hamiltonian matrix **F** is zero between basis functions on sites that are far away and thus becomes sparse for large molecules. Thus the next step in our testing, summarized in Table 3, is to employ a sparse matrix multiplication scheme to evaluate the fast resummation algorithms for the same sort of test matrices we just explored. This is a very interesting test case for two reasons. First, it should be possible to obtain speedups relative to function evaluation via diagonalization because diagonalization scales cubically with matrix dimension while polynomial evaluation can scale linearly once a high degree of sparsity is obtained. Second, both sparse matrix multiplication and sparse addition scale linearly so it will be interesting to see what speedups are observed, since fast summation algorithms reduce the number of costlier multiplications but do not affect additions.

Table 3 shows these timings for two classes of model electronic structure systems that exhibit sparsity: one-dimensional linear alkane oligomers (again) and 2-dimensional water clusters. Again, the matrix **X** comes from converged mean-field calculations in the minimal STO-3G basis set on an IBM Power-3 375MHz CPU. A blocked sparse matrix multiplication scheme is employed for the matrix multiplications [23]. In this scheme non-zero submatrices (of size roughly 30 to 60) are obtained by forming many-atom blocks. These blocks are obtained by a hierarchical boxing scheme, where the system is spatially partitioned into many boxes with each box containing many atoms. While the fraction of negligible submatrices is lower than the actual elemental sparsity, the blocking scheme benefits from the use of highly-optimized level-3 basic linear algebra subroutines (BLAS) for the submatrix multiplications, and eliminates the overhead of tracking element by element sparsity.

Table 3
The CPU time for resumming the matrix polynomial is shown for a series of linear alkanes and water clusters

Molecule	N	N_{pol}	$\langle M \rangle$			CPU time (s)				
			I	II	III	T_D	T_c	I	II	III
$C_{60}H_{122}$	422	180	29	27	25	1.35	11.92	2.91	2.75	2.65
$C_{120}H_{242}$	842	180	29	27	25	12.47	32.63	8.10	7.95	7.84
$C_{240}H_{482}$	1682	180	29	27	25	124.29	75.65	22.31	21.01	19.12
3×3	504	180	29	27	25	2.50	27.92	6.60	6.22	5.86
4×4	896	180	29	27	25	18.61	82.53	22.82	20.37	19.70
5×5	1400	180	29	27	25	72.62	166.30	49.18	46.72	45.13
6×6	2016	180	29	27	25	161.33	282.99	84.92	82.32	80.81

The geometry is ideal with C–C bond lengths of 1.54 Å, C–H bond lengths of 1.10 Å and C–C–C bond angles of 109.5°, respectively. The sparse block size includes 10 carbon atoms in these calculations for linear alkanes and 8 water molecules for water clusters. The bandwidth of these band diagonal matrices for linear alkanes is about 3 blocks long (i.e., about 200 basis functions). T_D is the CPU time by the matrix diagonalization method.

Our results shed light on both the issues mentioned two paragraphs above. First, the CPU time to either sum or resum matrix polynomials is significantly reduced by the sparse matrix multiplication scheme, and is increasing near-linearly for the largest test cases shown. By contrast, the CPU time for evaluating the matrix function by diagonalizing the matrix $\mathbf{X}$ scales as $O(N^3)$ (where N is the dimension of the matrix), and thus the two approaches cross over for the linear alkane oligomers between 60 and 120 for the fast resummation algorithms, while a later cross-over between 120 and 180 is found for conventional term-by-term matrix polynomial evaluation. Second, we see a reduced speedup for the fast summation methods in Table 3 relative to Table 2. For example, for the alkane corresponding to a matrix size of 842, the speedup decreases from a factor of almost 6 to a factor of just over 4. This reflects the increased importance of the matrix addition. However, there is still great value in applying the fast resummation methods as the cost of matrix multiplications is still significantly larger (scaling as approximately the square of the number of significant neighbors versus linear in the number of significant neighbors).

Table 3 also shows the computational time for a second model system (planar water clusters) which is approximately 2-dimensional in real space as opposed to the approximately 1-dimensional alkane oligomers. This differing topology gives more significant neighbors for a given number of atoms (i.e., for a given matrix dimension). We can anticipate that it will be slightly more difficult to obtain a cross-over between evaluating the matrix function via diagonalization versus matrix polynomial evaluation because one must go to larger systems to get comparable sparsity. This is observed in that the cross-over for fast resummation occur for a larger matrix dimension (i.e., greater than 900 as opposed to less than 840).

Other physical systems or model matrices could have been chosen with still more significant neighbors, that would exhibit this effect even more strongly (i.e., later cross-overs between diagonalization and summation-based matrix function evaluation). Additionally this would lead to larger speedups from using fast resummation relative to term-by-term summation because the increased number of significant neighbors increases the ratio between matrix multiplication and addition times. By contrast, systems with fewer significant neighbors would exhibit the opposite trends with respect to both of these considerations. The tradeoff between the sparsity pattern, the precision required (i.e., polynomial length and thus $\langle M \rangle$, and the size of the matrices determines the usefulness (or lack of usefulness) of these fast summation methods in a given application. In addition, of course, the amount of memory available is the final determinant of whether or not the additional storage of intermediate matrices in the fast resummation methods can be tolerated in a particular application. If not, the approach of Van Loan [15] is a good alternative.

5. Conclusions

Fast summation algorithms for evaluating matrix polynomials and matrix functions have been revisited in this paper. Three algorithms have been presented to reduce the number of matrix multiplications in the evaluation of matrix polynomials and Chebyshev matrix polynomials. They significantly reduce the total number of matrix multiplications and thus lead to speed-ups in CPU time relative to simple term-by-term summation.

These methods can be very useful when the matrices in question are large and sparse. In this case the usual direct evaluation of the matrix function through all the eigenvalues and eigenvectors of the matrix $\mathbf{X}$ inherently requires a calculation of $O(N^3)$ complexity. By contrast evaluation of the matrix polynomial by fast summation involves only sparse matrix multiplications and additions which both can be evaluated in only $O(N)$ effort if $\mathbf{X}$ is sparse. Furthermore, parallelization is fairly straightforward.

We present matrix multiplication counts and computer timings on some model systems to test the usefulness of the fast summation methods, and to explore the role of matrix sparsity on the viability of these approaches. A practical application to linear scaling electronic structure calculations is presented elsewhere [20].

Acknowledgements

WZL would like to express her deep gratitude to Prof. Arup K. Chakraborty and Mr. Baron Peters for many stimulating discussions relevant to this work. Financial support from BP (WZL) and the Israel-US Binational Science Foundation (Baer & MHG) as well as support (MHG) from the National Science Foundation (CHE-9981997) are gratefully acknowledged. This work was also supported by funding from Q-Chem Inc via an SBIR subcontract from the National Institutes of Health (R43GM069255-01). MHG is a part owner of Q-Chem Inc.

References

[1] X.P. Li, R.W. Nunes, D. Vanderbilt, Phys. Rev. B 47 (1993) 10891.
[2] R.N. Silver, H. Roeder, A.F. Voter, J.D. Kress, J. Comp. Phys. 124 (1996) 115.
[3] S. Goedecker, L. Colombo, Phys. Rev. Lett. 73 (1994) 122.
[4] T. Helgaker, P. Jørgensen, J. Olsen, Molecular Electronic-Structure Theory, Wiley, Chichester, 2000.
[5] T. Helgaker, H. Larsen, J. Olsen, P. Jørgensen, Chem. Phys. Lett. 327 (2000) 397.
[6] A.M. Niklasson, Phys. Rev. B 66 (2002) 155115.
[7] A.H.R. Palser, D. Manolopoulos, Phys. Rev B 58 (1998) 12704.
[8] H. Tal-ezer, R. Kosloff, J. Chem. Phys. 81 (1984) 3967.
[9] R. Kosloff, H. Tal-ezer, Chem. Phys. Lett. 127 (1986) 233.
[10] R.N. Silver, H. Röder, Int. J. Mol. Phys. C 5 (1994) 735.
[11] Y. Motome, N. Furukawa, J. Phys. Soc. Jpn. 68 (1999) 3853.
[12] T.S. Motzkin, Bull. Amer. Math. Soc. 61 (1955) 163.
[13] S. Winograd, Comm. Pure Appl. Math. 23 (1970) 165.
[14] M.S. Paterson, L.J. Stockmeyer, SIAM J. Comp. 2 (1973) 60.
[15] C.F. Van Loan, IEEE Trans. Auto. Cont. AC-24 (1979) 320.
[16] G.H. Golub, C.F. Van Loan, Matrix Computations, third ed., Johns Hopkins University Press, 1996.
[17] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in Fortrain 77, Cambridge University Press, New York, 1996.
[18] R. Kosloff, Ann. Rev. Phys. Chem. 45 (1994) 145.
[19] R. Baer, M. Head-Gordon, J. Chem. Phys. 107 (1997) 10003;
Phys. Rev. Lett. 79 (1997) 3962;
J. Chem. Phys. 109 (1998) 10159.
[20] W.Z. Liang, C. Saravanan, Y. Shao, R. Baer, A.T. Bell, M. Head-Gordon, J. Chem. Phys. 119 (2003) 4117.
[21] J. Kong, C.A. White, A.I. Krylov, C.D. Sherrill, R.D. Adamson, T.R. Furlani, M.S. Lee, A.M. Lee, S.R. Gwaltney, T.R. Adams, C. Ochsenfeld, A.T.B. Gilbert, G.S. Kedziora, V.A. Rassolov, D.R. Maurice, N. Nair, Y. Shao, N.A. Besley, P.E. Maslen, J.P. Dombroski, H. Daschel, W. Zhang, P.P. Korambath, J. Baker, E.F.C. Byrd, T. Van Voorhis, M. Oumi, S. Hirata, C.-P. Hsu, N. Ishikawa, J. Florian, A. Warshel, B.G. Johnson, P.M.W. Gill, M. Head-Gordon, J.A. Pople, J. Comput. Chem. 21 (2000) 1532.
[22] P.E. Maslen, C. Ochsenfeld, C.A. White, M.S. Lee, M. Head-Gordon, J. Phys. Chem. A 102 (1998) 2215.
[23] C. Saravanan, Y. Shao, R. Baer, P.N. Ross, M. Head-Gordon, J. Comp. Chem. 24 (2003) 618.