Rational approximation to the Fermi-Dirac function with applications in Density Functional Theory *

Yousef Saad † Roger B. Sidje ‡

February 10, 2009

Abstract

We are interested in computing the Fermi-Dirac matrix function in which the matrix argument is the Hamiltonian matrix arising from Density Function Theory (DFT) applications. More precisely, we are really interested in the diagonal of this matrix function. We discuss rational approximation methods to the problem, specifically the rational Chebyshev approximation and the continued fraction representation. These schemes are further decomposed into their partial fraction expansions, leading ultimately to computing the diagonal of the inverse of a shifted matrix over a series of shifts. We descibe Lanczos methods and sparse direct method to address these systems. Each approach has advanatges and disadvatanges that are illustrated with experiments.

Keywords: Fermi-Dirac, diagonal of the inverse of a matrix, electronic structure calculation, Density function theory

1 Introduction

The problem considered in this paper is that of computing the diagonal of the matrix function

$$P = f(H)$$

where H is the Hamiltonian and

$$f(z) = \frac{1}{1 + \exp\left(\frac{z-\mu}{k_B T}\right)}$$

is the Fermi-Dirac function, in which k_B is the Boltzmann's constant, μ is a real variable representing the chemical potential, T is the temperature, and z is a complex variable. Thus in matrix form,

$$f(H) = \left[I + \exp\left(\frac{1}{k_B T}(H - \mu I) \right) \right]^{-1},$$

*Work supported by NSF under grant 0325218, by DOE under Grant DE-FG02-03ER25585, and by the Minnesota Supercomputing Institute

†Computer Science & Engineering, University of Minnesota, Twin Cities. saad@cs.umn.edu

‡Department of Mathematics, The University of Alabama.

and therefore, if we let diag(X) denote a vector whose entries are the diagonal elements of the matrix X, the problem amounts to retrieving diag($f(H)$), i.e., literally the diagonal of the inverse of a shifted matrix exponential where the argument matrix is a sparse Halmitonian of large dimension.

It is well known that computing the matrix exponential can be a treacherous task, and if we are to compute the matrix exponential in isolation, it would result in a full matrix even though the original matrix is sparse. Furthermore in our present case, the difficulty is compounded with the subsequent inversion before retrieving the diagonal entries. Hence any approach based on first computing the matrix exponential in full would be impractical for large realistic problems. Several techniques have been suggested for this problem, the most promising of which have relied on approximating the Fermi-Dirac function by another function that is easier to compute. In [2] for example, Bekas, Kokiopoulou and Saad described how to construct a polynomial filter approximation based on a conjugate-residual type algorithm in polynomial space. They considered the particular case where $T \to 0$, in which case f reduces to the Heaviside (or step) function. Related works have also considered techniques based on a Chebyshev series expansion of the Heaviside function [1, 13]. Other recent works include [4].

In this paper, we explore rational approximation methods to the Fermi-Dirac function, namely the rational Chebyshev approximation and the continued fraction representation. The resulting rational approximations are further decomposed into their partial fraction expansions, leading ultimately to a series of shifted matrix inversions. These subproblems can then be handled using either direct methods if feasible, or iterative methods. It should however be stressed that we are really interested in the diagonal of the inverse and so further specialization comes into play. The focus of our presentation in the paper is on how sparse direct methods or the iterative Lanczos algorithm can be used with rational approximations to address the problem.

The organization of the paper is as follows. Section 2 outlines the Lanczos algorithm and its use to approximate matrix functions in general and the Fermi-Dirac function in particular. Section 3 describes the two rational approximation schemes considered in our study, with Section 4 and Section 5 respectively detail how the iterative Lanczos process and sparse direct methods can be used to evaluate each term of their partial fraction expansion. Section 6 presents some numerical results. Section 7 finally gives some concluding remarks.

2 The Lanczos algorithm

The Lanczos algorithm [14, 10, 7, 24] is the best known method for computing eigenpairs of a large sparse symmetric real (or Hermitian complex) matrix. This algorithm has also been used for a wide range of other calculations, such as solving linear systems [20, 25], computing the action of the matrix exponential on a vector, see, *e.g.* [23, 27], and even solving differential equations using exponentially-fitted methods, *e.g.*, [16, 17, 12]. In exact arithmetic, the algorithm can be recast as a simple three-term recurrence, namely,

$$\beta_{i+1} q_{i+1} = A q_i - \alpha_i q_i - \beta_i q_{i-1} \tag{1}$$

where α_i, β_{i+1} are selected at step i so that the vector q_{i+1} is of norm unity and orthogonal to q_i and q_{i-1} (when $i > 1$). This also shows that only three vectors are required in memory

at any step. What is remarkable about this recurrence is that, after m s it has computed an orthonormal basis of the m-th Krylov subspace

$$K_m(A, q_1) = \text{span}\{q_1, Aq_1, \cdots, A^{m-1}q_1\}.$$

Algorithmic details are outlined below.

ALGORITHM **2.1** *Lanczos algorithm*

1. *Set* $\beta_1 := 0$, $q_1 := 0$;
2. *For* $i := 1, \ldots, m$ *Do*
3. $\quad w := Hq_i - \beta_i q_{i-1}$;
4. $\quad \alpha_i := q_i^T w$;
5. $\quad w := w - \alpha_i q_i$;
6. $\quad \beta_{i+1} := \|w\|_2$;
7. $\quad q_{i+1} := w/\beta_{i+1}$;
8. *EndDo*

After m s of the Lanczos algorithm on the Hamiltonian H and a unit norm starting vector q_1 the following factorization holds

$$HQ_m = Q_m T_m + \beta_{m+1} q_{m+1} e_m^\top, \tag{2}$$

where $Q_m = [q_1, \ldots, q_m]$, q_{m+1} is the last vector computed by Lanczos and e_m is the m-th column of the canonical basis (thus e_m has 1 at the $m-$th entry and zeros elsewhere), $T_m = \text{tridiag}\ [\ \beta_i,\ \alpha_i,\ \beta_{i+1}\ ]$, is the tridiagonal symmetric matrix, with nonzero entries $\beta_i, \alpha_i, \beta_{i+1}$ in row i.

In practice, however, it is well known that the algorithm in its simplest form given by the recurrence in Eq. (1) is unstable and severe loss of orthogonality among the q_i's will take place after a number of s. The onset of this instability tends to coincide with the convergence of one or more eigenvalues, as Paige discovered in 1971 [19]. The simplest remedy against loss of orthogonality is to apply a full reorthogonalization step, whereby the orthogonality of the basis vector q_i is enforced against all previous vectors at each step i. This means that the vector q_i, which in theory is already orthogonal against $q_1, \ldots, q_{i-1}$, is orthogonalized (a second time) against these vectors. The total additional cost at the m-th step will be of order $O(nm^2)$. In addition, all basis vectors must be stored and accessed at each step, making this approach impractical for runs with a large number of Lanczos s.

An inexpensive alternative is the *partial reorthogonalization* scheme which performs a reorthogonalization step only when it is deemed necessary. This scheme does not guarantee that the vectors are exactly orthogonal, but ensures that they are at least nearly orthogonal. Typically, the loss of orthogonality is allowed to grow up to roughly the square root of the machine precision, before a reorthogonalization is performed. This technique relies on the existence of clever recurrences to estimate the level of orthogonality among the basis vectors [15, 28]. The cost of updating the recurrences is negligible.

An important side benefit of this procedure, is that it becomes unnecessary to store all basis vectors in main memory. We can instead use secondary storage and bring these vectors back to main memory, say a few at a time, when they are needed for reorthogonalization.

The rationale is that previous vectors will only be needed infrequently, so the cost of accessing secondary storage will not hamper overall performance significantly. The simple regular access pattern will allow to dampen the high cost of accessing secondary storage by overlapping computations with read/writes from disk and UPC is an ideal language to employ to efficiently handle such data transfers.

On the implemention side, an appealing characteristic of the Lanczos algorithm is that the matrix A is only needed in "functional" form: All that is needed is a routine to compute the product Aq_i for any given vector q_i. The matrix can be applied in stencil form, meaning that it is not stored, but its action on a given vector is implemented by working directly on the vector in the lattice. It can also be stored in a sparse format [22].

If we were to run all n s of Lanczos in exact arithmetic, then $H = Q_n T_n Q_n^\top$ and hence the matrix function $f(H)$ could be obtained as $f(H) = Q_n f(T_n) Q_n^T$ and from there the problem becomes that of computing a matrix function whose argument is a tridiagonal matrix. But it is not necessary to carry all n s to obtain useful approximations. Rather, one can use

$$f(H) \approx Q_m f(T_m) Q_m^T.$$

Bekas *et al.* [3] used the Lanczos algorithm with partial reorthogonalization and studied this approach in the case of the Fermi-Dirac function. They evaluated $f(T_m)$ by diagonalization (as opposed to using a rational approximation as done here). It proved very competitive with the standard implicitly restarted Lanczos procedure of ARPACK.

Consider as another example a linear system, where $f(z) = z^{-1}$, one can view the conjugate gradient algorithm for solving the linear system $Hx = b$ as a means of approximating the inverse of H by $Q_m T_m^{-1} Q_m^T$, which is then applied to the right-hand side (or the initial residual $r_0 = b - Hx_0$ to be more accurate). The actual algorithm works by exploiting some useful recurrences. A way to gain another insight into the derivation of the reduced size approximation is to post-multiply (2) by $Q_m^\top$, thereby obtaining

$$HQ_m Q_m^\top = Q_m T_m Q_m^\top + \beta_{m+1} q_{m+1} q_m^\top,$$

and thus if we neglect the trailing term, $\beta_{m+1} q_{m+1} q_m^\top$, which usually becomes small as $\beta_{m+1} \to 0$ when m increases, we can consider the approximation $HQ_m Q_m^\top \approx Q_m T_m Q_m^\top$. Using the Taylor series representation of f, it is not difficult to see that we are led to $f(H) Q_m Q_m^\top \approx Q_m f(T_m) Q_m^\top$. To recap therefore, $Q_m f(T_m) Q_m^\top$ is a candidate approximation to either $f(H)$ or $f(H) Q_m Q_m^\top$. Such a distinction is not significant in the case of a linear system with the starting vector $q_1 = Q_m e_1 = r_0/\beta$, or when computing the action of the matrix function $f(H)$ on the operand vector q_1. The reason is because the Q_m factor is cancelled out when the approximation is ultimately applied to q_1. However this distinction turns out to be insightful if we are to approximate $f(H)$ in general. Experiments suggest that for certain functions $Q_m f(T_m) Q_m^\top$ can be a significantly better approximation to $f(H) Q_m Q_m^\top$ than it is to $f(H)$, and that in general a good approximation to $f(H)$ is achieved only when orthogonality is maintained. We illustrate this in Figure 1.

We can clearly draw from these observations that for $Q_m f(T_m) Q_m^\top$ to be a robust enough approach, orthogonality must be preserved. We now turn our attention at investigating other approaches.

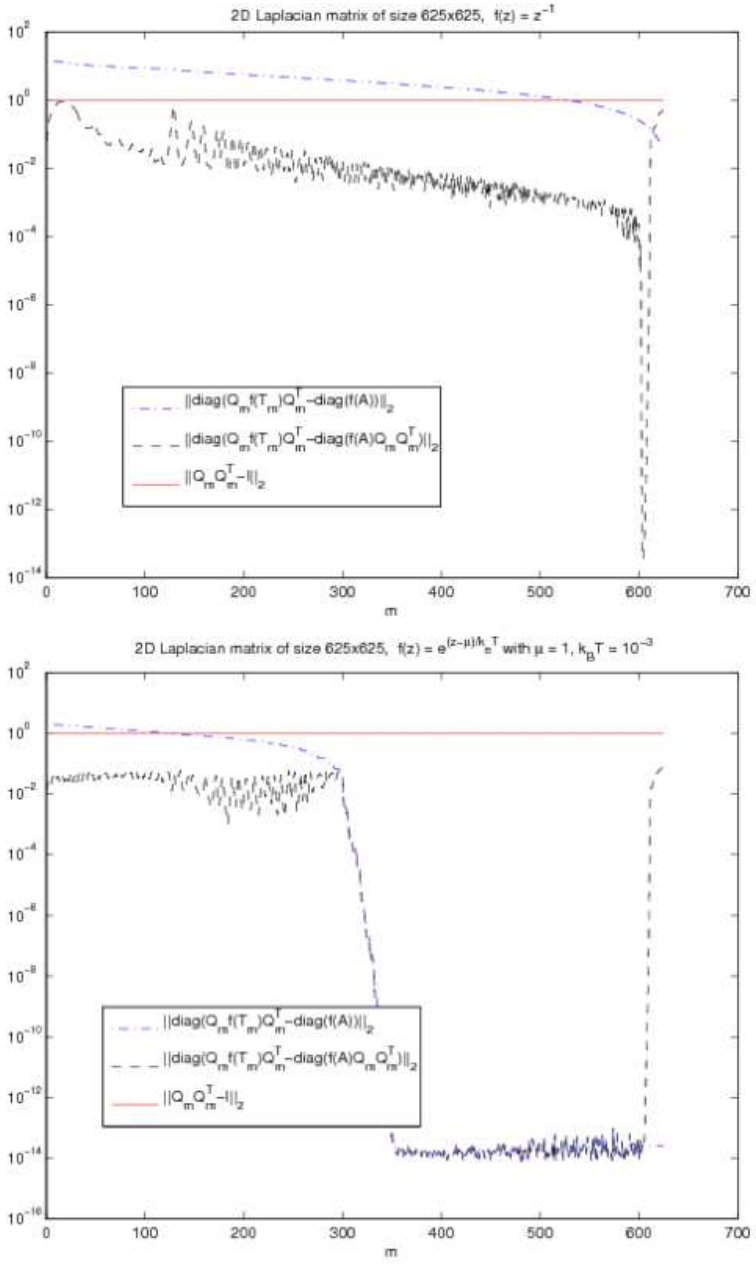

Figure 1: Approximation errors during the Lanczos iterations with full reorthogonalization. The first plot uses $f(z) = z^{-1}$ and the second plot uses $f(z) = \exp((z-\mu)/k_BT)$, both the same matrix and a random starting vector. Observe in the first plot how the error $\| \operatorname{diag}(Q_m f(T_m) Q_m^\top) - \operatorname{diag}(f(H))\|_2$ stagnates. Whereas in the second plot, accuracy is pretty good when the Lanczos basis is big enough to capture the eigenvectors corresponding to the eigenvalues $\leq \mu$ as reported in Bekas *et al.* [3]. In both cases, the approximation should be exact when $m = n$, but finite precision arithmetic introduce discrepancies.

3 Rational approximation

Since the exponential function is the main ingredient in the definition of the Fermi-Dirac function, it is natural to explore how approximation schemes that have been proposed for

the exponential function can be extended to the Fermi-Dirac function. However, not all extensions are suitable because the conversion from the scalar case to the matrix case comes with some constraints. Matrix decomposition methods or methods such as the popular Padé method with the so-called scaling-squaring technique would make it necessary to deal with the matrix in full. This section focuses on two approaches that avoid matrix-matrix operations, namely the rational Chebyshev approximation and the continued fraction representation.

3.1 Rational Chebyshev approximation

When used for the exponential function, the main strength of the rational Chebyshev approximation is its ability to provide accurate results with a relatively low and fixed degree (provided that the scheme is used in its proper domain of applicability).

The rational Chebyshev approximation problem comes from extending the minimax Chebyshev theory to rational functions, specifically: find $r_d(x) \equiv p(x)/q(x)$ such that

$$\|r_d(x) - e^{-x}\|_{L^\infty[0,+\infty)} = \min_{r \in \mathcal{R}_d} \max_{x \in [0,+\infty)} |r(x) - e^{-x}| \tag{3}$$

where $\mathcal{R}_d$ denotes the class of rational functions of type (d, d). In general, the degree of the numerator need not be the same as the degree of the denominator, but we limit our presentation to this case because it is sufficient for our purposes.

This problem does not have a closed form solution, but it has been solved numerically for $d = 1, ..., 14$, by Cody, Meinardus and Varga [6], and subsequently up to degree $d = 30$ by Carpenter, Ruttan and Varga [5]. Interestingly, the problem does have a closed form solution if it is instead formulated (in the unit disk) over the extended approximation space $\tilde{\mathcal{R}}_d \supset \mathcal{R}_d$ of rational functions such that

$$\tilde{r}_d(z) = \sum_{k=-\infty}^{d} a_k z^k \Bigg/ \sum_{k=0}^{d} b_k z^k \,. \tag{4}$$

Dropping the terms of negative degree of the numerator in (4) gives a near-best approximation (see Trefethen [29] or Trefethen and Gutknecht [30] for details)

$$r_d^{cf}(z) = \sum_{k=0}^{d} a_k z^k \Bigg/ \sum_{k=0}^{d} b_k z^k \,.$$

What is noteworthy in this approach is that there is a constructive algorithm based on an earlier result of Carathéodory and Fejér to compute the coefficients a_k and b_k on the fly for any arbritrary degree d using the singular value decomposition (SVD) of a Hankel matrix that is populated by the coeffients of the Taylor series expansion.

While we could interchangeably use this approach (and have tested that it works), we continue our presentation with the ordinary rational Chebyshev approximation for which the coefficients of the best approximants $p(x)$ and $q(x)$ have been computed and listed for $d = 1, 2, ..., 30$ in [6, 5]. Starting therefore with $e^{-x} \approx r_d(x)$, we can derive an approximation to the Fermi-Dirac function as

$$f(x) = \frac{1}{e^x + 1} \approx \frac{1}{\frac{q(x)}{p(x)} + 1} = \frac{p(x)}{p(x) + q(x)}. \tag{5}$$

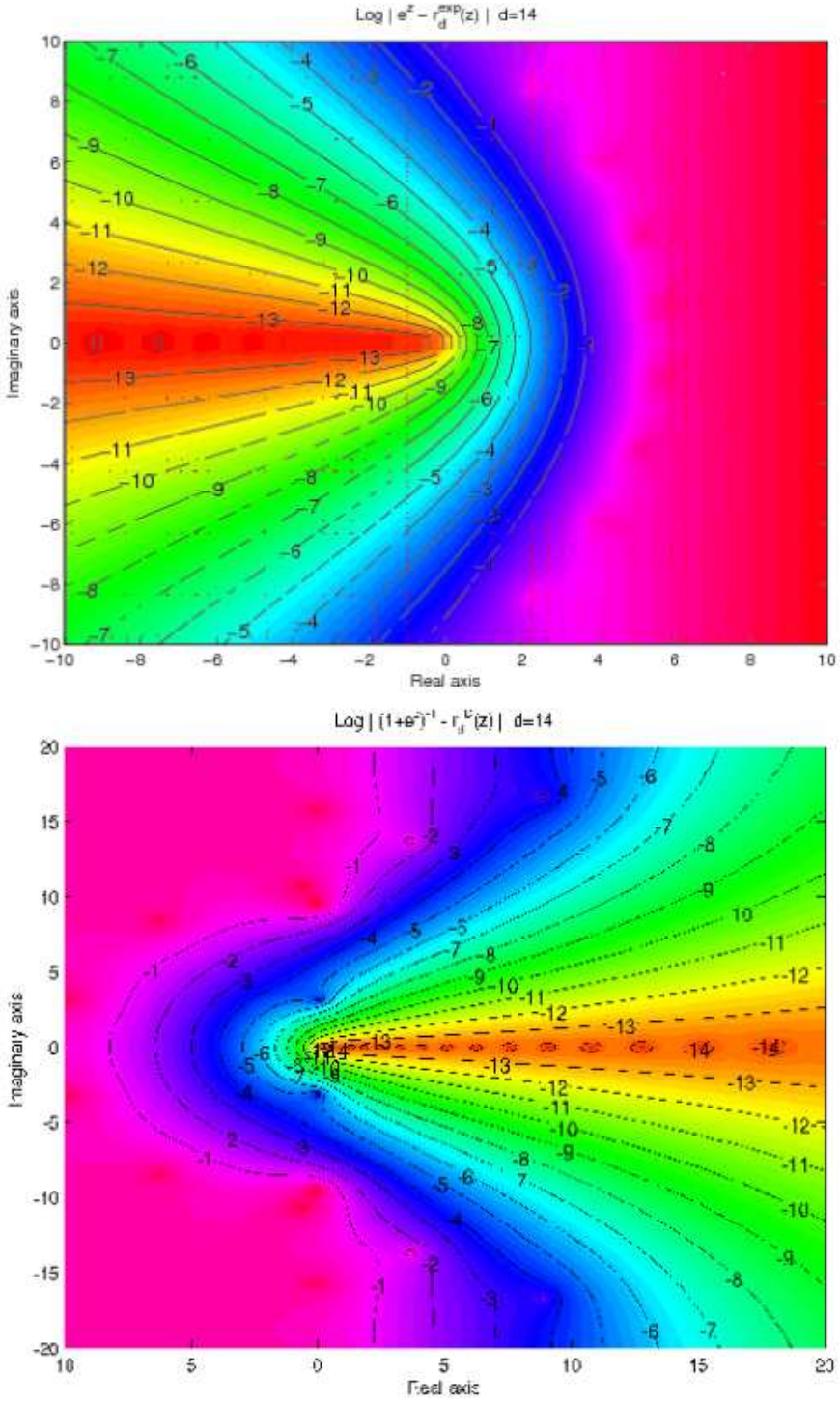

Figure 2: Contour plots of the error in the respective rational Chebyshev approximations of the exponential function and the Fermi-Dirac function.

From the rational approximation (5), we can compute the partial fraction expansion

$$f(x) \approx R_0 + \sum_{k=1}^{d} \frac{R_k}{x - z_k}$$

to obtain

$$\operatorname{diag}(f(H)) \approx R_0 I + \sum_{k=1}^{d} R_k \operatorname{diag}(H - z_k I)^{-1}. \tag{6}$$

We have therefore turned the problem into computing the diagonal of the inverse of a shifted matrix across a few number of known shifts. Because the poles are complex and H is symmetric and thus with real eigenvalues, $H - z_k I$ is guaranteed to be invertible. As we indicated at the beginning, the main caveat on the Chebyshev approach is the domain of applicability specified in (3), which makes the approach more suited for symmetric definite matrices. In fact it has been shown for the exponential that

$$\| \exp(-H) - r_d(H)_d \|_2 \leq \lambda_d,$$

where λ_d is an explicitly known constant alternatively referred to as the *uniform rational Chebyshev constant* [31] or the *Halphen constant* due to its earlier origin [11]. It is now known that $\lambda_d \approx 10^{-d}$ which means that a type (d, d)-approximation yields about d-digit accuracy. Note however that the bound only holds for a symmetric real matrix (or Hermitian complex matrix), and not for any general matrix with a complex spectrum and/or a poorly conditioned system of eigenvectors. As can be seen in Figure 2, these approximants become invalid and are subject to large errors if employed where not intended. Shifting the exponential as $e^z = e^s e^{(z-s)} \approx e^s r_d(-(z-s))$ only works for small s since e^s quickly becomes large and magnifies the error. While this straightforward adaptation of the Chebyshev approach is very efficient due to its low degree, our experiments confirmed that it can not be used in all circumstances. Its restricted domain of applicability prevents using the scheme in its basic form as a black-box, general purpose method for the Fermi-Dirac function.

Table 1: Residues and poles of the rational Chebyshev approximation of type (14,14) to the Fermi-Dirac function. They come in conjugate pairs and so only half of the set is listed.

$R_0 =$	$1.832174378254008\text{e}{-14}$
$R_1 =$	$7.153332540307382\text{e}{-05} + 1.436536356343437\text{e}{-04}i$
$R_2 =$	$-9.372540241863129\text{e}{-03} - 1.659031409384731\text{e}{-02}i$
$R_3 =$	$1.081135621732985\text{e}{+00} - 7.781250683748498\text{e}{-01}i$
$R_4 =$	$6.007249618115624\text{e}{-01} - 7.563702806572788\text{e}{-01}i$
$R_5 =$	$9.903908361025214\text{e}{-01} - 3.162473096388244\text{e}{-01}i$
$R_6 =$	$-1.662954347498257\text{e}{+00} + 2.014087581978868\text{e}{-01}i$
$R_7 =$	$-9.999960653156149\text{e}{-01} + 5.974185742799262\text{e}{-07}i$
$z_1 =$	$8.897701648364055\text{e}{+00} + 1.663083898786899\text{e}{+01}i$
$z_2 =$	$3.712681837943989\text{e}{+00} + 1.367325588180097\text{e}{+01}i$
$z_3 =$	$-6.407114925441066\text{e}{-01} + 1.074407843430750\text{e}{+01}i$
$z_4 =$	$-6.242676963830224\text{e}{+00} + 8.370140270020642\text{e}{+00}i$
$z_5 =$	$-9.787553309337540\text{e}{+00} + 3.234733921618516\text{e}{+00}i$
$z_6 =$	$-1.052768339150649\text{e}{-01} + 9.647884870240162\text{e}{+00}i$
$z_7 =$	$8.144083801508994\text{e}{-07} + 3.141591911938352\text{e}{+00}i$

3.2 Continued fraction approximation

Any rational approximation corresponds to a truncated continued fraction and vice-versa. In [18], Ozaki evaluated the Fermi-Dirac function using a continued fraction representation that can further be decomposed into a partial fraction expansion via a generalized eigenvalue problem. This continued fraction representation is equivalent to a Padé approximation, although Ozaki derived it differently using the ratio of two hypergeometric functions. We briefly summarize the key results. Writing

$$\frac{1}{1+e^x} = \frac{1}{1+\frac{1+\tanh(x/2)}{1-\tanh(x/2)}} = \frac{1}{2} - \frac{1}{2}\tanh(x/2)$$

and using the continued fraction expansion of the hyperbolic tangent function, it follows that the one of the Fermi-Dirac function is

$$\frac{1}{1+e^x} = \frac{1}{2} - \frac{1}{2}\cfrac{(x/2)}{1+\cfrac{(x/2)^2}{3+\cfrac{(x/2)^2}{5+\cfrac{(x/2)^2}{\cdots\over (2k-1)+\ddots}}}} \tag{7}$$

and, if truncated at length d, its partial fraction expansion can be written as

$$\frac{1}{1+e^x} \approx R_0 + \sum_{k=1}^{d/2} \frac{R_k}{x - iz_k} + \sum_{k=1}^{d/2} \frac{R_k}{x + iz_k}. \tag{8}$$

where the degree d is chosen to be even for convenience, and also to make it clear that only half of the evaluation is done because the poles come in conjugate pairs. It turns out that the residues R_k are real and identical for conjugate poles, and that the poles are purely imaginary. They can be computed on the fly in an elegant manner owing to the property that the $(1,1)$ element of the inverse of the tridiagonal matrix

$$C = \begin{pmatrix} c_{11} & c_{12} & & & \\ c_{21} & c_{22} & c_{23} & & \\ & c_{32} & c_{33} & \ddots & \\ & & \ddots & \ddots & c_{d-1,d} \\ & & & c_{d,d-1} & c_{dd} \end{pmatrix}$$

is given by

$$e_1^T C^{-1} e_1 = \cfrac{1}{c_{11} - \cfrac{c_{12}c_{21}}{c_{22} - \cfrac{c_{23}c_{32}}{c_{33} - \cfrac{c_{34}c_{43}}{\cdots\over c_{dd}}}}}.$$

This property has been used elsewhere, for example in Filipponi [9], and earlier in Haydock, Heine and Kelly [21] and others. Setting in particular $c_{kk} = 2k-1$ and $c_{k,k+1} = c_{k,k+1} = i\frac{x}{2}$ allows approximating the Fermi-Dirac function as done by Ozaki [18]. Specifically, with

$$C = \begin{pmatrix} 1 & i\frac{x}{2} & & & \\ i\frac{x}{2} & 3 & i\frac{x}{2} & & \\ & i\frac{x}{2} & 5 & \ddots & \\ & & \ddots & \ddots & i\frac{x}{2} \\ & & & i\frac{x}{2} & 2d-1 \end{pmatrix} = D + ixN,$$

where

$$D = \begin{pmatrix} 1 & & & & \\ & 3 & & & \\ & & 5 & & \\ & & & \ddots & \\ & & & & 2d-1 \end{pmatrix}, \quad N = \frac{1}{2}\begin{pmatrix} 0 & 1 & & & \\ 1 & 0 & 1 & & \\ & 1 & 0 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 0 \end{pmatrix},$$

we infer from (7) that

$$\begin{aligned} \frac{1}{1+e^x} \approx \frac{1}{2} - \frac{x}{4} e_1^T C^{-1} e_1 &= \frac{1}{2} - \frac{x}{4} e_1^T (D + ixN)^{-1} e_1 \\ &= \frac{1}{2} - \frac{x}{4} e_1^T D^{-1/2} (I + ixD^{-1/2} N D^{-1/2})^{-1} D^{-1/2} e_1 \\ &= \frac{1}{2} - \frac{x}{4} e_1^T U (I + ix\Lambda)^{-1} U^T e_1 \end{aligned}$$

with the observation that $D^{-1/2} e_1 = e_1$ and using the eigen decomposition

$$D^{-1/2} N D^{-1/2} = U \Lambda U^T, \quad \Lambda = \mathrm{diag}(-\lambda_{d/2}, \ldots, -\lambda_1, \lambda_1, \ldots, \lambda_{d/2}).$$

The result simplifies to (8) with

$$R_0 = \frac{1}{2}, \quad R_k = -\frac{1}{4}\left(\frac{e_1^T U e_k}{\lambda_k}\right)^2, \quad z_k = \frac{1}{\lambda_k}, \quad k = 1, \ldots, d/2.$$

Given the relevance of the method in other contexts, we include a Matlab script to perform the partial fraction decomposition for interested readers.

```
function [zk, Rk, R0] = cfracgen(halfdeg)
% Generate the zeros and residues of a continued fraction expansion for
% the Fermi-Dirac function terminated at 2*halfdeg.
d = 2*halfdeg;                          % degree of the continued fraction
v = [1:2:2*d-1];                        % setup the
w = 0.5 ./ sqrt(v(1:end-1).*v(2:end));%   underlying
T = diag(w,1) + diag(w,-1);             %     eigenvalue problem
[U,D] = eig(T);
zk = 1./diag(D);                        % poles are inverse of eigenvalues
Rk = U(1,:); Rk = Rk(:);                % unscaled residues
Rk = -(1/4)*(Rk./diag(D)).^2;           % scale them to their final values
R0 = 1/2;
[dummy, order] = sort(zk);              % sort to select the negative half
zk = zk(order); zk = i*zk(1:halfdeg); % make them purely imaginary and
Rk = Rk(order); Rk = Rk(1:halfdeg);   % collect their associated residues
```

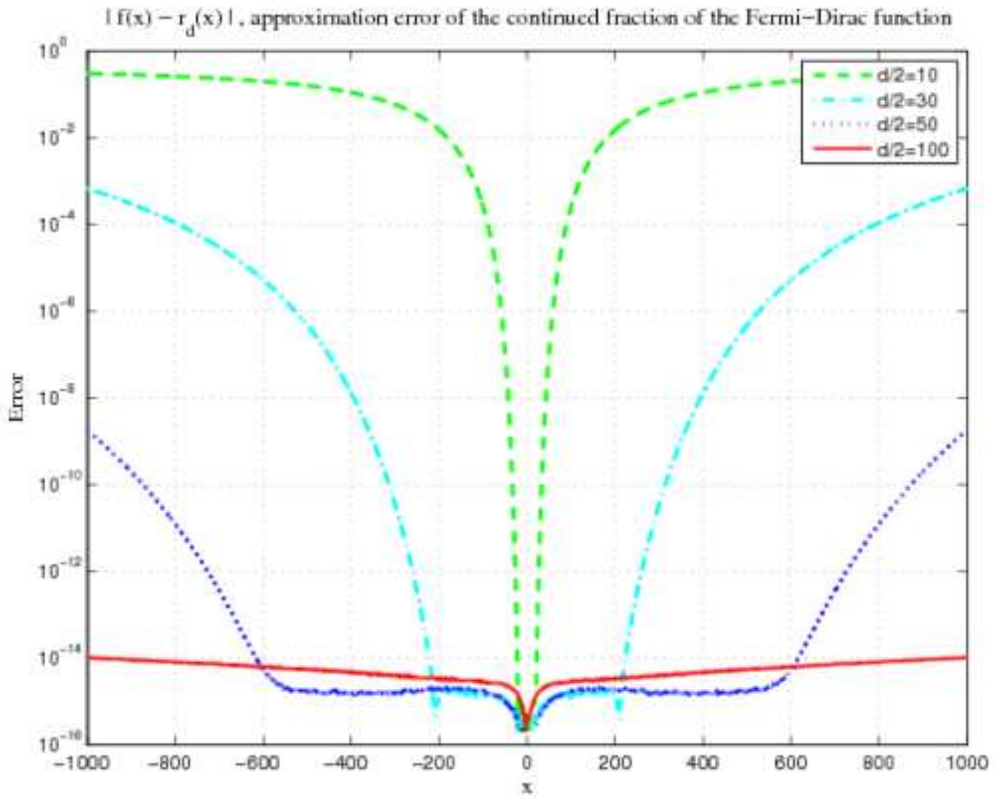

Figure 3: Approximation error of the continued fraction expansion of the Fermi-Dirac function truncated at length d. Points of the curves that dip below the machineilon $2.2 \cdot 10^{-16}$ are left out for readability. It is clear that targeting a wide range of x requires a high degree.

As in the Chebyshev approach, once the partial fraction expansion is known the problem becomes that of computing the diagonal of the inverse of a shifted matrix across a number of known shifts. This is discussed next.

4 Computing $\mathrm{diag}(H-\theta I)^{-1}$ via the Lanczos algorithm

We showed earlier how the Lanczos algorithm can in principle be used to define the basic approximation

$$\mathrm{diag}((H-\theta I)^{-1}) \approx \mathrm{diag}(Q_m(T_m-\theta I)^{-1}Q_m^T).$$

We shall now describe how the diagonal can be updated in an elegant way by using recurrences that are similar to those of the Conjugate Gradient (CG) algorithm. What this suggests is that the Lanczos algorithm can be used as in Bekas *et al.* [3], but combined with a rational approximation to the Fermi-Dirac function for $f(T_m)$ instead of diagonalization.

Lanczos algorithm for computing diag(inv($H-\theta I$))
The algorithm starts with a random vector q_1, $d_0 = p_0 = q_0 = 0$; and $\beta_1 = \eta_1 = 0$. In the algorithm, d_j represents the sequence of vectors whose entries approximate the diagonal of H^{-1}. The notation $u \odot v$ stands for the component-wise product of the vectors u and v. Thus $p_j \odot p_j$ is simply the vector of the squares of the entries of p_j.

For $j = 1, 2, \cdots, Do$:
  {Compute next Lanczos vector & scalars}
  $\beta_{j+1} q_{j+1} = Hq_j - \alpha_j q_j - \beta_j q_{j-1}$
  {Compute next direction & diagonal iterate }
  $p_j := q_j - \eta_j p_{j-1}$
  $\delta_j := \alpha_j - \theta - \beta_j \eta_j$
  $d_j := d_{j-1} + \frac{p_j \odot p_j}{\delta_j}$
  $\eta_{j+1} := \frac{\beta_{j+1}}{\delta_j}$
EndDo

The directions p_j are scaled versions of the conjugate directions of CG. Indeed to justify the algorithm, assume that we have the LDL^T decomposition of $T_m - \theta I$,

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_m \\ & & \beta_m & \alpha_m \end{pmatrix}, \quad L_m = \begin{pmatrix} 1 & & & 0 \\ \eta_2 & 1 & & \\ & \ddots & \ddots & \\ 0 & & \eta_m & 1 \end{pmatrix}, \quad D_m = \begin{pmatrix} \delta_1 & & & 0 \\ & \delta_2 & & \\ & & \ddots & \\ 0 & & & \delta_m \end{pmatrix},$$

where the coefficients of the decomposition can be shown to satisfy the following relations: $\delta_1 = \alpha_1$ and for $j = 2 : m$, $\eta_j = \beta_j/\delta_{j-1}$, $\delta_j = \alpha_j - \beta_j \eta_j$. From there,

$$\mathrm{diag}(Q_m(T_m-\theta I)^{-1}Q_m^T) = \mathrm{diag}(Q_m L_m^{-T} D_m^{-1} L_m^{-1} Q_m^T) = \sum_{j=1}^{m} \frac{p_j \odot p_j}{\delta_j},$$

where we set $Q_m L_m^{-T} = (\, p_1 \quad \ldots \quad p_m \,)$ and we have $p_j = q_j - \eta_j p_{j-1}$ given that

$$Q_j L_j^{-T} = (\, Q_{j-1} \quad q_j \,) \begin{pmatrix} L_{j-1} & 0 \\ \eta_j e_{j-1}^T & 1 \end{pmatrix}^{-T} = (\, Q_{j-1} L_{j-1}^{-T} \quad q_j - \eta_j Q_{j-1} L_{j-1}^{-T} e_{j-1} \,).$$

To summarize the overall procedure for (6), the basis Q_m and the tridiagonal matrix T_m are the same for a series of θ_k, and the extra cost comes only from a loop over different sequences, say p_j^k and d_j^k corresponding to each pole θ_k. From there the d_j^k are summed up to obtain the final approximation in the way shown in (6). Interestingly, approximating (6)

acurately may generally take fewer iterations than approximating each term in the partial sum individually. As we pointed out earlier, $Q_m f(T_m) Q_m$ behaves differently depending on f. When considering each term individualy, it is as if each term aims at the inverse function $f(z) = z^{-1}$, in which case, a good approximation needs the entire spectrum, whereas when taken collectively, we in effect take the Fermi-Dirac function $f(z) = \exp((z-\mu)/k_B T)$, and in this case only the eigenvalues less than the Fermi level μ are significant, as can be seen in Bekas *et al.* [3].

In applications such as those envisioned in this work, the Hamiltonian can be very large, requiring a large number of Lanczos s to get to a good approximation. In this situation, the cost of reorthogonalization becomes prohibitive and dominates the overall calculation. Having highlighted the strong connection with Bekas *et al.* [3], we now focus the rest of presentation on sparse direct methods.

5 Computing $\mathrm{diag}(H - \theta I)^{-1}$ via sparse direct methods

As can be seen for example in Duff, Erisman and Reid [8], the standard way of extracting the diagonal of the inverse of a matrix is through its LU decomposition, or through the square-root free Cholesky LDL^* decomposition in the Hermitian case. However, direct techniques create an extra fill-in that can make them very demanding in terms of storage. Another aspect is that, unlike the Lanczos algorithm, which allows us to extract several diagonals at once, a new factorization must be performed for every new θ. But the stability concerns that we have illustrated earlier with the Lanczos approach are serious enough to warrant giving a consideration to the more accurate sparse direct methods nowithstanding their higher cost. Moreover, we will see later that incomplete factorizations can possibly be attempted to reduce their cost.

We start by briefly outlining the approach described in Duff, Erisman and Reid [8] for computing entries of the inverse of a matrix. Let $Z = A^{-1}$, where we assume to begin with that A is general, and that we have computed a sparse factorization

$$A = LDU,$$

with L unit lower triangular, D diagonal, and U unit upper triangular. We can exploit the sparsity pattern of L and U to get the entries of $Z = A^{-1}$ in an economical way, owing to these relations due to Takahashi, Fagan, and Chin

$$\begin{aligned} Z &= U^{-1}D^{-1} + Z(I-L), \\ Z &= D^{-1}L^{-1} + (I-U)Z. \end{aligned}$$

Now, $(I-L)$ is strictly lower triangular and $(I-U)$ is strictly upper triangular, and so

$$\begin{aligned} z_{ij} &= e_i^T Z(I-L)e_j = -\sum_{k=j+1}^{n} z_{ik} l_{kj}, \quad i > j, \\ z_{ij} &= e_i^T (I-U)Z e_j = -\sum_{k=i+1}^{n} u_{ik} z_{kj}, \quad i < j, \\ z_{ii} &= d_{ii}^{-1} + e_i^T Z(I-L)e_i, \\ z_{ii} &= d_{ii}^{-1} + e_i^T (I-U)Z e_i. \end{aligned}$$

What these relations imply is that we can develop a computational sequence, starting from $z_{nn} = d_{nn}^{-1}$ and moving backwards such that the computation of any entry z_{ij} only involves the entries z_{st} $(s > i,\ t > j)$ that have already been computed, and this, importantly, while exploiting the sparsity pattern of L and U in the products $z_{ik}l_{kj}$ and $u_{ik}z_{kj}$ to economize the computations.

5.1 Using the LDL^* decomposition

Consider the case where the matrix A is Hermitian, which permits the root-free Cholesky decomposition $A = LDL^*$ where L is unit lower triangular and D is diagonal real. We show below two possible algorithms (in the MATLAB language[1]) to compute $Z = A^{-1}$, also Hermitian, by recasting the above relations in the Hermitian context. The sequence in the first algorithm is row oriented while it is column oriented in the second one. Another variant (not shown here) is also possible by organizing the sequence to compute the trailing submatrix, i.e., starting with z_{nn} and expanding to $Z(n-1\colon n, n-1\colon n)$ and so on.

```
function Z = invLDL1(L, D)
% Given A = LDL', compute Z = inv(A)
% row oriented version
[n,n] = size(L); Z = zeros(n,n);
for i = n:-1:1
  Z(i,i) = 1/D(i,i)-Z(i,i+1:n)*L(i+1:n,i);
  for j = i-1:-1:1
    Z(i,j) = -Z(i,j+1:n)*L(j+1:n,j);
    Z(j,i) = conj(Z(i,j));
  end
end
```

```
function Z = invLDL2(L, D)
% Given A = LDL', compute Z = inv(A)
% column oriented version
[n,n] = size(L); Z = zeros(n,n);
for j = n:-1:1
  Z(j,j) = 1/D(j,j)-L(j+1:n,j)'*Z(j+1:n,j);
  for i = j-1:-1:1
    Z(i,j) = -L(i+1:n,i)'*Z(i+1:n,j);
    Z(j,i) = conj(Z(i,j));
  end
end
```

The algorithms above are not optimized and they only serve to illustrate possible computational sequences as per our early discussion. Sparse data structures normally entail programming practices vastly more intricate than shown in the algorithms. Our presentation is geared toward readability, but a final implementation should make the most of sparse data structures. Since our real interest is in the diagonal of the inverse, our main goal is to tune the computations for this situation. If we partition the decomposition of $A = LDL^*$ as

$$L = \begin{pmatrix} L_{n-1} & 0 \\ l_n^* & 1 \end{pmatrix}, \quad D = \begin{pmatrix} D_{n-1} & 0 \\ 0 & d_n \end{pmatrix},$$

then

$$Z = L^{-*}D^{-1}L^{-1} = \begin{pmatrix} L_{n-1}^{-*}D_{n-1}^{-1}L_{n-1}^{-1} + d_n^{-1}L_{n-1}^{-*}l_n l_n^* L_{n-1}^{-1} & -d_n^{-1}L_{n-1}^{-*}l_n \\ -d_n^{-1}l_n^* L_{n-1}^{-1} & d_n^{-1} \end{pmatrix}.$$

[1]For readers not familiar with the Matlab notation: X' denotes the conjugate transpose, $X\backslash Y$ performs $X^{-1}Y$ whereas X/Y performs XY^{-1}, but either case relies on Gaussian elimination and the matrix inverse is never computed. We will also often use $x(:)$ in our listings to handily turn a row vector into a column vector so as to operate with consistent dimensions when appropriate.

This suggests these possible algorithmic sequences for obtaining $\mathrm{diag}(Z)$ in the Hermitian case. Note the use of the identity $\mathrm{diag}(uv^*) = u \odot \bar{v}$ in the algorithms, and that it cannot be interchanged with $\bar{u} \odot v$ unless $u = v$.

```
function z = diaginvLDL1(L, D)
% Given A = LDL', compute z = diag(inv(A)), recursive
[n,n] = size(L);
z = zeros(n,1);
z(n) = 1/D(n);
if n > 1
  v = L(n,1:n-1)/L(1:n-1,1:n-1); v = v(:);
  z(1:n-1) = z(n)*v.*conj(v) + ...
    diaginvLDL1(L(1:n-1,1:n-1),D(1:n-1));
end
```

```
function z = diaginvLDL2(L, D)
% Given A = LDL', compute z = diag(inv(A))
[n,n] = size(L);
z = zeros(n,1);
z(1) = 1/D(1);
for i = 2:n
  z(i) = 1/D(i);
  v = L(i,1:i-1)/L(1:i-1,1:i-1); v = v(:);
  z(1:i-1) = z(1:i-1) + z(i)*v.*conj(v);
end
```

While the recursive version is more readable, it has the disadvantage that it will not scale well because the recursion stack will become too deep to the point of exceeding runtime limits. Matlab for example has by default a maximum recursion limit of 100. It can be changed with `set(0,'RecursionLimit',N)`, but in general, exceeding the available stack space can crash Matlab and/or the computer system. The non-recursive version is immune to this issue. The *major* caveat in both cases, however, is that they assume that the matrix is Hermitian. Recall in our particular situation that we are dealing with $H - \theta I$, which is *not* Hermitian even if H is, because the pole θ is complex. Thus we need to handle this specific case. Although this might look like an innocuous change from the Hermitian case, it has a far reaching consequence: the more economical LDL^* decomposition cannot be used anymore. We are compelled to revert to the general case.

5.2 Using the LDU decomposition

Going back to a general matrix A, if we partition its decomposition $A = LDU$ as

$$L = \begin{pmatrix} L_{n-1} & 0 \\ l_n^* & 1 \end{pmatrix}, \quad D = \begin{pmatrix} D_{n-1} & 0 \\ 0 & d_n \end{pmatrix}, \quad U = \begin{pmatrix} U_{n-1} & u_n \\ 0 & 1 \end{pmatrix},$$

then

$$Z = U^{-1}D^{-1}L^{-1} = \begin{pmatrix} U_{n-1}^{-1}D_{n-1}^{-1}L_{n-1}^{-1} + d_n^{-1}U_{n-1}^{-1}u_n l_n^* L_{n-1}^{-1} & -d_n^{-1}U_{n-1}^{-1}u_n \\ -d_n^{-1}l_n^* L_{n-1}^{-1} & d_n^{-1} \end{pmatrix}.$$

This suggests these possible coding sequences for obtaining diag(Z) in the general case. The first variant uses recursion and so the reservation that we mentioned earlier applies here too.

```
function z = diaginvLDU1(L, D, U)
% Given A = LDU, compute z = diag(inv(A)), recursive
[n,n] = size(L);
z = zeros(n,1);
z(n) = 1/D(n);
if n > 1
  u = U(1:n-1,1:n-1)\U(1:n-1,n);
  v = L(n,1:n-1)/L(1:n-1,1:n-1); v = v(:);
  z(1:n-1) = z(n)*u.*v + ...
    diaginvLDU1(L(1:n-1,1:n-1),D(1:n-1),U(1:n-1,1:n-1));
end
```

```
function z = diaginvLDU2(L, D, U)
% Given A = LDU, compute z = diag(inv(A))
[n,n] = size(L);
z = zeros(n,1);
z(1) = 1/D(1);
for i = 2:n
  z(i) = 1/D(i);
  u = U(1:i-1,1:i-1)\U(1:i-1,i);
  v = L(i,1:i-1)/L(1:i-1,1:i-1); v = v(:);
  z(1:i-1) = z(1:i-1) + z(i)*u.*v;
end
```

Although we are compelled to use the LDU decomposition even though H is symmetric, we should point out that the poles and residues come in conjugate pairs as indicated in Table 1, and this yields substantial savings. Indeed we only need to perform half of the work because $(H - \bar{z}_k I)^{-1} = (H - z_k I)^{-*}$, thus

$$\text{diag}\left(R_k(H - z_k I)^{-1} + \bar{R}_k(H - \bar{z}_k I)^{-1}\right) = \text{diag}\left(Re[2R_k(H - z_k I)^{-1}]\right). \tag{9}$$

5.3 Using the LDU decomposition with partial pivoting

It is well known that a nonpivoting direct solver tends to suffer from rounding errors that are much more significant than the rounding errors observed when pivoting is used. Hence pivoting may be needed in certain cases to ensure robustness. We limit ourselves to partial pivoting, which is usually sufficient in general. Consider the factorization

$$PA = LDU$$

where P is a permutation matrix and so

$$Z = A^{-1} = U^{-1}D^{-1}L^{-1}P.$$

The product $L^{-1}P$ is not unit lower triangular, but L is and partitioning as before

$$L = \begin{pmatrix} L_{n-1} & 0 \\ l_n^* & 1 \end{pmatrix}, \quad D = \begin{pmatrix} D_{n-1} & 0 \\ 0 & d_n \end{pmatrix}, \quad U = \begin{pmatrix} U_{n-1} & u_n \\ 0 & 1 \end{pmatrix},$$

and

$$P = \begin{pmatrix} P_{n-1} & c_n \\ r_n^* & \delta_{nn} \end{pmatrix},$$

we can write

$$\mathrm{diag}(Z) = \begin{pmatrix} \mathrm{diag}\left[U_{n-1}^{-1}D_{n-1}^{-1}L_{n-1}P_{n-1} + d_n^{-1}U_{n-1}^{-1}u_n\left(l_n^*L_{n-1}^{-1}P_{n-1} - r_n^*\right)\right] \\ d_n^{-1}(\delta_{nn} - l_n^*L_{n-1}^{-1}c_n) \end{pmatrix}.$$

Note that the submatrix P_{n-1} is not necessarily a permutation matrix and should not be treated as such. The main difference (and extra cost) compared to the nonpivoting version is that at each step we need to account for the action of this submatrix, as well as the interference of r_n^*, c_n and δ_{nn} in the computational sequences. The above relation establishes a recurrence involving $\mathrm{diag}(U_{n-1}^{-1}D_{n-1}^{-1}L_{n-1}P_{n-1})$ and thus it can be translated into an algorithm as done earlier. However, it is inefficient to setup the permutation matrix P explicitly. It is sufficient, economical, and more efficient by far, to keep the permutation information into a vector, and this is all the more important given that we target large problems. We provide further details as to how to cast the algorithm with this in mind since such details are important to make good use of sparse data structures in practice. Let σ_r be the permutation vector that characterizes P row-wise, and let σ_c be the one that characterizes P column-wise. They are related through the relation $\sigma_c(\sigma_r) = 1{:}n = \sigma_r(\sigma_c)$, and we have

$$P = \begin{pmatrix} e_{\sigma_r(1)}^* \\ \vdots \\ e_{\sigma_r(n)}^* \end{pmatrix} = \begin{pmatrix} e_{\sigma_c(1)} & \cdots & e_{\sigma_c(n)} \end{pmatrix},$$

which shows how to easily extract rows or columns of P knowing σ_r or σ_c. In fact, the quantities r_n^*, c_n, δ_{nn} introduced earlier in the partitioning of P can all be null as the recurrence unwinds into the principal submatrices of P because P is made up of permuted rows (or columns) of the identity matrix. For this same reason, if $\delta_{nn} = 1$, then r_n^* and c_n must necessarily be null, and if either r_n^* or c_n is non null, then $\delta_{nn} = 0$. In this latter case there must only be a single non null component with value one in r_n^* and/or c_n. Computations can be tuned to only rely on the index of this single non null entry. Also, in the course of the algorithm, we need to compute v^*P_s where v is a (sparse) vector of length $s < n$ and P_s is the s-th leading principal submatrix of P. To perform this, we just write

$$v^*P_s = v^*\begin{bmatrix} I_s & 0 \end{bmatrix}P\begin{bmatrix} I_s \\ 0 \end{bmatrix} = \begin{pmatrix} \tilde{v}_{\sigma_c(1)} & \cdots & \tilde{v}_{\sigma_c(s)} \end{pmatrix}, \quad \tilde{v}^* = \begin{pmatrix} v^* & 0 \end{pmatrix}.$$

The following algorithm takes these observations into account. Of all the algorithms discussed so far, this is the most robust and general purpose, albeit the downside of partial pivoting is in general a much increased fill-in. In the listing, σ_r is represented by the variable `pr` while σ_c is represented by the variable `pc`. It is clear that most of the

compute time of the algorithm will come from the sparse triangular system solves for `u = U(1:i-1,1:i-1)\U(1:i-1,i)` and `v = L(i,1:i-1)/L(1:i-1,1:i-1)` at each step. These need to be performed with efficient sparse data structures in a production code.

```
function z = diaginvLDUpiv(L, D, U, pr)
% Given PA = LDU where the permutation matrix P is characterized
% row-wise by the vector pr, compute z = diag(inv(A)).
[n,n] = size(L);
pc(pr) = 1:n;
z = zeros(n,1);
if pr(1) == 1
  z(1) = 1/D(1);
end
for i = 2:n
  d = 1/D(i);
  u = U(1:i-1,1:i-1)\U(1:i-1,i);
  v = L(i,1:i-1)/L(1:i-1,1:i-1);
  vp = zeros(n,1);
  vp(1:i-1) = v;
  vp = vp(pc(1:i-1));
  ir = pr(i);
  ic = pc(i);
  if ir == i    % r and c are null
    z(i) = d;
  else
    if ir < i   % r = e_{ir}
      vp(ir) = vp(ir) - 1;
    end
    if ic < i   % c = e_{ic}
      z(i) = -d*v(ic);
    end
  end
  z(1:i-1) = z(1:i-1) + d*u.*vp;
end
```

6 Numerical results

When we put together all the elements that we have discussed so far, we obtain an overall algorithm for our stated problem. The following Matlab template provides a basis for developing a more advanced code in Fortran or C/C++.

```
function f = ratfermidirac(H, mu, kBT, R0, Rk, zk)
% Compute f = diag( inv(I + expm[(H - mu*I)/kBT]) ).
% This only uses half of the poles, hence length(zk) is d/2.
I = speye(size(H));
f = R0*ones(size(H,1),1);
for k = 1:length(zk)
  z = diaginv(H - (mu+kBT*zk(k))*I);
  f = f + real((2*kBT*Rk(k))*z);
end;
```

On input the poles and residues can either be the Chebyshev ones given on Table 1 (if using the Chebyshev approach is suitable for the problem), or, more generally, the ones of the continued fraction as computed in section 3.2. We have already stressed in our presentation that the core task needed for for each pole is common to either approach as (9) shows. The Chebyshev approach, if applicable, just requires fewer terms in its partial fraction expansion. The experiments reported here are performed with the continued fraction, with the understanding that the Chebyshev would be cheaper if it was applicable for the problem at the hand. Hence we will not dwell further on the difference between these two approaches. The computer system that we use for the experiments is a Dell workstation called *syphax* in our environment and running Linux. It has 16GB of RAM and 2 dual core AMD Opteron processors (thus 4 core processors total), each with a 2.2 Ghz clock, but it should be understood that our code entails sequential computations. The generic function `diaginv` shown in the script is meant to compute the diagonal of the inverse. Our Fortran code performs an LU decomposition without pivoting to limit the fill-in, and this is then used to compute the diagonal of the inverse as explained in our presentation. On the tables, z_1 and z_n represent the first and last diagonal entries of the matrix function. In addition to experimenting with a complete factorization, we also report experiments where we attempted obtaining the LU factors with an incomplete factorization using various drop tolerance thresholds. Although incomplete factorizations are generally used as preconditioners instead of one-shot solvers in their own right, Sidje and Stewart [26] observed that they can be useful as solvers in the Newton phase of implicit integrators. This motivates trying similar experiments here given the need to compute the diagonal of the inverse over potentially numerous shifts. Results of our experiments are listed in the tables below. We set in the code $k_B = 6.33327186 \cdot 10^{-6}$, the Fermi level $\mu = 7$, the Fermi temperature $T = 10^3$, and we use a continued fraction of degree $d = 200$, that is the effective number of systems to deal with is $d/2 = 100$.

If we denote by $\lambda_{\min}$ and $\lambda_{\max}$ the smallest and largest eigenvalues of the matrix, then it is important to ensure that domain of accuracy of the continued fraction is big enough to include $(\lambda_{\min} - \mu)/k_B T$ and $(\lambda_{\max} - \mu)/k_B T$. We report these quantities on Table 2, and it can be seen from Figure 3 that the range for a continued fraction of degree $d = 200$ is big enough to cover all the test problems.

Matrix	n	nz	density	$\lambda_{\min}$	$\lambda_{\max}$	$\frac{\lambda_{\min}-\mu}{k_BT}$	$\frac{\lambda_{\max}-\mu}{k_BT}$
si2	769	17801	0.030	-0.3844	41.3813	-1.1660e+03	5.4287e+03
Si_2sym	1647	69701	0.026	-0.7064	35.9969	-1.2168e+03	4.5785e+03
Si_sym	2777	102703	0.013	-0.7595	36.4485	-1.2252e+03	4.6498e+03
Na5_sym	5832	305630	0.009	-0.1638	25.6604	-1.1311e+03	2.9464e+03
benzene_sym	8219	242669	0.004	-0.7296	58.3937	-1.2205e+03	8.1149e+03
gr_30_30	900	7744	0.010	0.0615	11.9591	-1.0956e+03	7.8302e+02

Table 2: Problems characteristics. To illustrate the effect of bandness we include in the tests the banded matrix **gr_30_30** from the Harwell-Boeing collection.

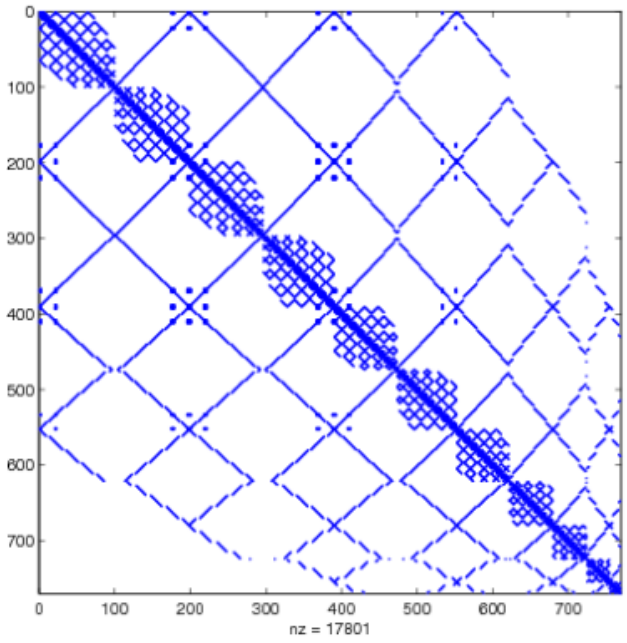

Figure 4: Sparsity partten of the matrix **si2**.

	Matrix **si2**: $n = 769$; $nz = 17801$				
	fill-in	Time	z_1	z_n	$\|z_{\rm LU} - z_{\rm ILU}\|_\infty$
LU	468343	4.3e+02	1.64567257e-01	1.04394207e-02	
ILUTH(10^{-5})	445892	3.6e+02	1.64567673e-01	1.04394258e-02	3.1e-06
ILUTH(10^{-4})	401972	3.0e+02	1.64570421e-01	1.04395151e-02	4.7e-05
ILUTH(10^{-3})	334131	2.1e+02	1.64376532e-01	1.04367885e-02	5.3e-04
ILUTH(10^{-2})	233226	1.2e+02	1.64285585e-01	1.04288876e-02	1.2e-02

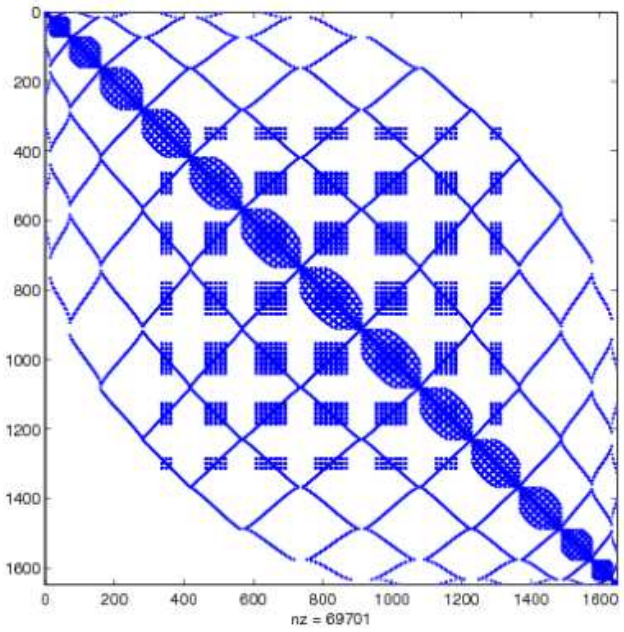

Figure 5: Sparsity partten of the matrix **Si_2sym**.

	Matrix **Si_2sym**: $n = 1647$; $nz = 69701$				
	fill-in	Time	z_1	z_n	$\|z_{\rm LU} - z_{\rm ILU}\|_\infty$
LU	1787271	3.3e+03	1.46619766e-02	2.41112218e-02	
ILUTH(10^{-5})	1677064	2.4e+03	1.46618411e-02	2.41111652e-02	1.0e-03
ILUTH(10^{-4})	1497547	1.8e+03	1.46583863e-02	2.41108016e-02	2.3e-03
ILUTH(10^{-3})	1228400	1.2e+03	1.46924738e-02	2.41128873e-02	3.2e-02
ILUTH(10^{-2})	849297	5.8e+02	1.48429178e-02	2.42608149e-02	6.0e-01

	Matrix **Si_sym**: $n = 2777$; $nz = 102703$				
	fill-in	Time	z_1	z_n	$\|z_{\rm LU} - z_{\rm ILU}\|_\infty$
LU	4656279	1.4e+04	4.79197023e-02	4.81102302e-02	
ILUTH(10^{-5})	4341197	9.5e+03	4.79193415e-02	4.81103693e-02	1.9e-04
ILUTH(10^{-4})	3824245	7.0e+03	4.79174357e-02	4.81082004e-02	4.7e-04
ILUTH(10^{-3})	3078775	4.3e+03	4.80198014e-02	4.80819374e-02	2.6e-02
ILUTH(10^{-2})	2082093	2.0e+03	4.78236221e-02	4.83636722e-02	6.2e-01

	Matrix **Na5_sym**: $n = 5832$; $nz = 305630$				
	fill-in	Time	z_1	z_n	$\|z_{\rm LU} - z_{\rm ILU}\|_\infty$
LU	17546710	1.1e+05	1.41527931e-01	1.53972605e-01	
ILUTH(10^{-5})	16122088	5.7e+04	1.41527719e-01	1.53972497e-01	1.5e-04
ILUTH(10^{-4})	14080028	3.9e+04	1.41523559e-01	1.53969451e-01	3.8e-04
ILUTH(10^{-3})	11170122	2.2e+04	1.41574902e-01	1.53960797e-01	2.6e-02
ILUTH(10^{-2})	7433217	9.2e+03	1.41680342e-01	1.54277648e-01	1.4e-01

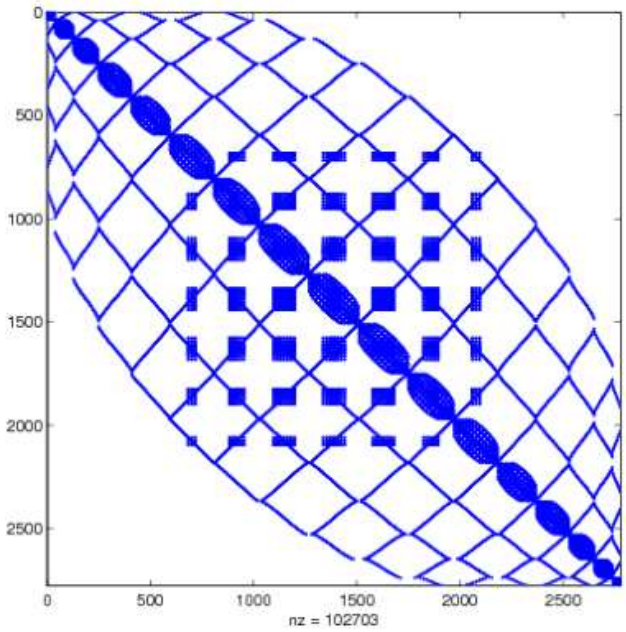

Figure 6: Sparsity partten of the matrix **Si_sym**.

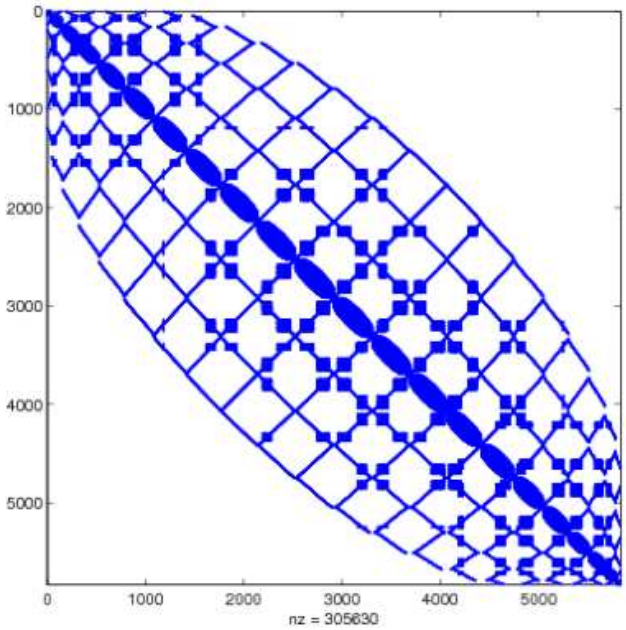

Figure 7: Sparsity partten of the matrix **Na5_sym**.

	Matrix **benzene_sym**: $n = 8219$; $nz = 242669$				
	fill-in	Time	z_1	z_n	$\|z_{\rm LU} - z_{\rm ILU}\|_\infty$
LU	32403625	3.2e+05	6.91951271e-02	9.07962363e-03	
ILUTH(10^{-5})	29278196	1.8e+05	6.91949792e-02	9.07966677e-03	6.5e-06
ILUTH(10^{-4})	24720088	1.4e+05	6.92091920e-02	9.07948568e-03	7.2e-05
ILUTH(10^{-3})	18800858	8.2e+04	6.90126254e-02	9.07945655e-03	8.2e-04
ILUTH(10^{-2})	11526582	3.3e+04	6.92235099e-02	9.18450982e-03	3.7e-02

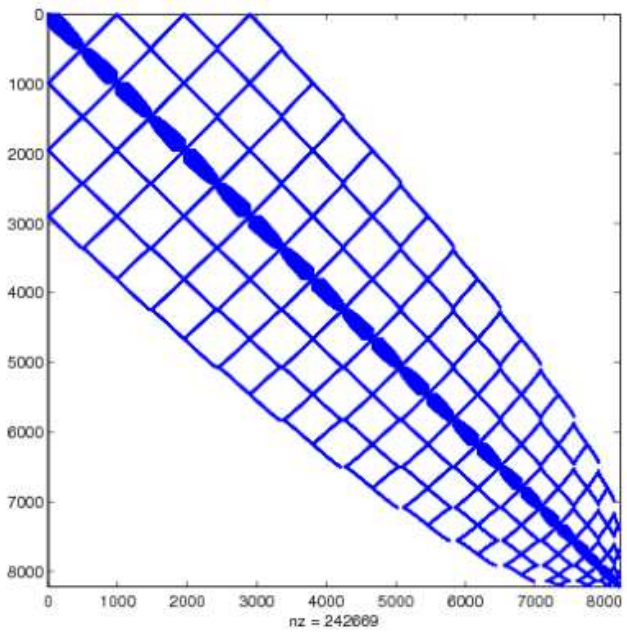

Figure 8: Sparsity partten of the matrix **benzene_sym**.

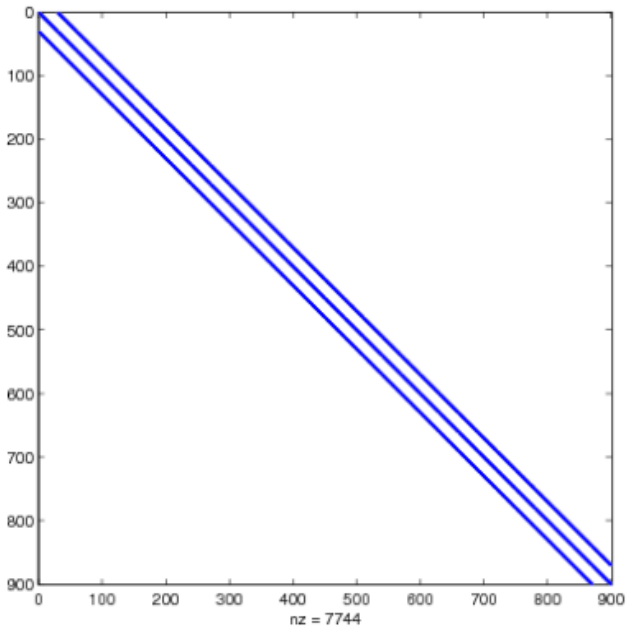

Figure 9: Sparsity partten of the matrix **gr_30_30**.

	Matrix **gr_30_30**: $n = 900$; $nz = 7744$				
	fill-in	Time	z_1	z_n	$\|z_{\mathrm{LU}} - z_{\mathrm{ILU}}\|_\infty$
LU	54840	5.6e+01	2.29625553e-01	2.29625553e-01	
ILUTH(10^{-5})	54840	4.6e+01	2.29625553e-01	2.29625553e-01	6.9e-10
ILUTH(10^{-4})	54840	4.2e+01	2.29625543e-01	2.29625555e-01	6.4e-08
ILUTH(10^{-3})	54838	3.5e+01	2.29626450e-01	2.29626340e-01	5.8e-06
ILUTH(10^{-2})	54663	2.7e+01	2.30416964e-01	2.29646851e-01	7.9e-04

7 Conclusion

We have explored how the Fermi-Dirac function can be evaluated using rational approximations. We discussed the uniform rational Chebyshev approximation, which has the advantage of being of low degree, but has the disadvantage of being restricted to only half of the real line. We also discussed a truncated continued fraction approximation, which has the advantage of being applicable to a wider class of problems, but has the disadvantage of requiring a high degree to achieve accuracy.

In terms of execution time, the impact of the degree in either rational scheme is evident by the fact that the rational scheme is ultimately converted into a partial fraction expansion that must be evaluated via shifted matrix inversions with complex shifts. The lower the degree, the fewer the number of terms to evaluate in the expansion. Because our interest is really in the diagonal of the inverse, we showed that the Lanczos method could in principle provide a very elegant mechanism for the problem in exact arithmetic. However, we observed in practice that loss of orthogonally in finite arithmetic can be detrimental, and the remedy against such loss of orthogonality involving some sort of reorthogonalization as done in Bekas et al. [3]. We also implemented sparse direct methods in Fortran and performed numerical tests that showed that while accuracy is achieved with such methods, the execution time can be high for large problems. To reduce the cost of the computations, we used incomplete factorizations that drop a certain amount of the extra fill-in created by the sparse direct methods. We observed that such incomplete methods can trade accuracy for substantial savings, albeit the extent of the lost of accuracy is not predicted.

On the whole, we can conclude that the fact that the rational approximation method converts the original Fermi-Dirac problem into computing the diagonal of a series of matrix inverses makes the approach particular suitable for special matrices (e.g., narrowly banded matrices). It also means that the information gather from one one solve could help guide th next solve, although strategies remain open issues. Finally in parallel computing environments, the systems can be solved concurrently so that the nominal cost of the method could become the cost of only one solve.

References

[1] C. Bekas, E. Kokiopoulou, and Y. Saad. An estimator for the diagonal of a matrix. *Applied Numerical Mathematics*, 2006. To appear.

[2] C. Bekas, E. Kokiopoulou, and Y. Saad. Polynomial filtered Lanczos iterations with applications in density functional theory. *SIAM Journal on Matrix Analysis and Applications*, pages –, 2008. To appear.

[3] C. Bekas, Y. Saad, M. L. Tiago, and J. R. Chelikowsky. Computing charge densities with partially reorthogonalized Lanczos. *Computer Physics Communications*, 171(3):175–186, 2005.

[4] M. Benzi and N. Razouk. Decay bounds and *O(N)* algorithms for approximating functions of sparse matrices. *ETNA*, 28:16–39, 2007.

[5] A. J. Carpenter, A. Ruttan, and R. S. Varga. Extended numerical computations on the 1/9 conjecture in rational approximation theory. In *Lecture Notes in Mathematics 1105*, pages 383–411, Berlin, 1984. Springer-Verlag.

[6] W. J. Cody, G. Meinardus, and R. S. Varga. Chebyshev rational approximation to $\exp(-x)$ in $[0, +\infty)$ and applications to heat conduction problems. *J. Approx. Theory*, 2:50–65, 1969.

[7] J. Cullum and R. A. Willoughby. *Lanczos algorithms for large symmetric eigenvalue computations*. Volumes 1 and 2. Birkhäuser, Boston, 1985.

[8] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Method for Sparse Matrices*. Clarendon Press, Oxford, 1989.

[9] A. Filipponi. Continued fraction expansion for the x-ray absorption cross section. *J. Phys.: Condens. Matter*, 3:6489–6507, 1991.

[10] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[11] A. A. Goncar and E. A. Rakhmanov. On the rate of rational approximation of analytic functions. In *Lecture Notes in Mathematics 1354*, pages 25–42, Berlin-Heidelberg, 1988. Springer-Verlag.

[12] M. Hochbruck, C. Lubich, and H. Selhofer. Exponential integrators for large systems of differential equations. *SIAM Journal on Scientific Computing*, 19:1552–1574, 1998.

[13] L. O. Jay, H. Kim, Y. Saad, and J. R. Chelikowsky. Electronic structure calculations using plane wave codes without diagonlization. *Comput. Phys. Comm.*, 118:21–30, 1999.

[14] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45:255–282, 1950.

[15] R. M. Larsen. *Efficient Algorithms for Helioseismic Inversion*. PhD thesis, Dept. Computer Science, University of Aarhus, DK-8000 Aarhus C, Denmark, October 1998.

[16] B. Nour-Omid. Applications of the Lanczos algorithm. *Computer Physics Communications*, 53, 1989.

[17] B. Nour-Omid and R. W. Clogh. Dynamic analysis of structures using Lanczos coordinates. *Earthquake Eng. and Struct. Dynamics*, 12:565–577, 1984.

[18] T. Ozaki. Continued fraction representation of the Fermi-Dirac function for large-scale electronic structure calculations. *Phys. Rev. B*, 75:035123(9), 2007.

[19] C. C. Paige. *The computation of eigenvalues and eigenvectors of very large sparse matrices*. PhD thesis, London University, Institute of Computer Science, London, England, 1971.

[20] B. N. Parlett. A new look a the Lanczos algorithm for solving symmetric systems of linear equations. *Linear Algebra and its Applications*, 29:323–346, 1980.

[21] V. Heine R. Haydock and M. J. Kelly. Electronic structure based on the local atomic environment for tight-binding bands: Ii. *J. Phys.: Solid State Phys.*, 8:2591–2605, 1975.

[22] Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations. Technical Report RIACS-90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, CA, 1990.

[23] Y. Saad. Analysis of some Krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 29:209–228, 1992.

[24] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Halstead Press, New York, 1992.

[25] Y. Saad. *Iterative Methods for Sparse Linear Systems, 2nd edition*. SIAM, Philadelpha, PA, 2003.

[26] R. B. Sidje and W. J. Stewart. A numerical study of large sparse matrix exponentials arising in Markov chains. *Computional Statistics & Data Analysis*, 29(3):345–368, 1999.

[27] R.B. Sidje. EXPOKIT: A software package for computing matrix exponentials. *ACM Transactions on Mathematical Software*, 24(1):130–156, 1998. http://www.expokit.org.

[28] H. D. Simon. The Lanczos algorithm with partial reorthogonalization. *Math. Comp.*, 42(165):115–142, 1984.

[29] L. N. Trefethen. Rational Chebyshev approximation on the unit disk. *Numer. Math.*, 37(2):297–320, 1981.

[30] L. N. Trefethen and M. H. Gutknecht. The Carathéodory–Féjer method for real rational approximation. *SIAM J. Numer. Anal.*, 20(2):420–436, 1983.

[31] R. S. Varga. *Scientific Computation on Mathematical Problems and Conjectures*. CBMS-NSF, Regional Conference Series in Applied Mathematics, Vol. 60. SIAM, Philadelphia, 1990.