A C++ Matrix library for computing the Gateaux derivative of the Fermi-Dirac operator

William Samuelsson
Abstract

Computing the Fermi-Dirac operator is done through recursive polynomial expansions, using the SP2 and SP2 Acc algorithms. The Gateaux derivative is computed for both schemes by mapping the zeroth and first order matrices onto a block upper triangular matrix, which is implemented in Python using Numpy arrays to store full matrices and in C++ by first constructing a basic matrix library to use as blocks in a later created block upper triangular matrix library which only directly references two blocks in a $2 \times 2$ block matrix. Computations of the Fermi-Dirac operator were carried out on artificially created Hamiltonians to verify correct implementations, as well on real life examples of Fock matrices resulting from ergo calculations on water clusters (http://ergoscf.org/). It was found that the relative error in the first order response in the density matrix was not different when using SP2 Acc compared to when using SP2.
## Contents

1 Introduction .......................... 2
   1.1 Matrix functions .......................... 2
   1.2 Density Functional Theory .......................... 3

2 Recursive Fermi Dirac operator expansion .......................... 6
   2.1 Numerical experiments for SP2 and SP2 Acc in Python .......................... 10
   2.2 C++ programming model for basic matrix library and SP2 and SP2 Acc implementations .......................... 11
   2.3 Numerical experiments in C++ .......................... 12

3 Derivatives of Functions of Matrices and Density Matrix Perturbation Theory .......................... 13
   3.1 Python programming model for computing the Gateaux derivative .......................... 16
   3.2 Python computations .......................... 16
   3.3 C++ programming model for computing the Gateaux derivative .......................... 18
   3.4 C++ computations .......................... 19

4 Discussion .......................... 22

5 Conclusion .......................... 23

Populärvetenskaplig sammanfattning .......................... 26

Appendix .......................... 27
1 Introduction

Functions of matrices have a broad field of applications, not at least in the fields of ordinary differential equations, control theory and particle physics [1]. Another interesting application of matrix functions is found in computational quantum chemistry, where the use of low order polynomials in the computation of the density matrix from an effective Hamiltonian has greatly impacted the field [2, 3, 4, 5, 6].

The connection between matrix functions and computational quantum chemistry also extends itself to the response in the density matrix due to a first order perturbation of the effective Hamiltonian, as this relates to the Gateaux derivative of the matrix function.

This article will explore the use of matrix functions and the computation of their Gateaux derivatives through examining two algorithms for expanding the Fermi-Dirac operator recursively, using low order polynomials, as well shedding some light on the construction of efficient matrix libraries (in C++) to deploy in these algorithms, which mainly carry a computational kernel of matrix-matrix multiplications.

1.1 Matrix functions

Defining scalar functions is usually done by considering something which takes a scalar value in the set $X$ and maps it to exactly one scalar value in the set $Y$. When discussing functions of matrices, if there are no demands on continuity, these functions can be defined by how they act on the spectrum of the matrices. This leads to the following useful definition.

**Definition 1.1** A function $f$ is defined on the spectrum of $A \in \mathbb{C}^{n \times n}$ if

$$\exists f^{(j)}(\lambda_i), \text{ } j = 0 : n_i - 1, \text{ } i = 1 : s$$

where $n_i$ is the size of the largest Jordan block corresponding to the eigenvalue $\lambda_i$ and $s$ is the number of distinct eigenvalues. Putting to use the previously discussed Jordan normal form and considering functions defined on the spectrum of a matrix $A \in \mathbb{C}^{n \times n}$, it is deemed appropriate to define a scalar function $f$ of a matrix such that $f(A) \in \mathbb{C}^{n \times n}$ where

$$f(A) := Z f(J) Z^{-1}$$

$Z$ the change of basis matrix to the Jordan basis corresponding to $A$ [7].
acts on the diagonal blocks in the Jordan form $J_k$ in a way that

$$f(J_k) := \begin{bmatrix}
  f(\lambda_k) & f'(\lambda_k) & \ldots & f^{m_k-1}(\lambda_k) \\
  f(\lambda_k) & \ddots & \ddots & \ddots \\
  \vdots & \ddots & f'(\lambda_k) & \ddots \\
  \vdots & \ddots & \ddots & f(\lambda_k)
\end{bmatrix}$$  \hspace{1cm} (3)$$

Here it should be noted that if $A$ is diagonalizable the Jordan blocks will just be the eigenvalues of $A$.

There are other equivalent ways of defining functions of matrix, using interpolation or Cauchy integrals [1]. The definition 1.1 uses the Jordan Normal Form, which is guaranteed to exist for all linear operators [7]. The Jordan Normal Form will reduce to a diagonal matrix (block diagonal matrix with blocks of size 1) if the linear operator is Hermitian. \footnote{Those unfamiliar with the Jordan Normal Form (which is the form provided by the Jordan basis) can consult Axler’s book for proofs of its existence [7].}

This definition makes sense since it can be obtained from Taylor series considerations [1]. Writing $J_k = \lambda_k I + N_k \in \mathbb{C}^{m_k \times m_k}$ with $N_k$ a nilpotent upper triangular matrix and considering a function $f$ which is assumed to have a convergent Taylor series expansion

$$f(x) = f(\lambda_k) + f'(\lambda_k)(x - \lambda_k) + \ldots + \frac{f^{(j)}(\lambda_k)(x - \lambda_k)^j}{j!} + \ldots$$  \hspace{1cm} (4)$$
performing $x \mapsto J_k \in \mathbb{C}^{m_k \times m_k}$ yields

$$f(J_k) = f(\lambda_k)I + f'(\lambda_k)N_k + \ldots + \frac{f^{(m_k-1)}(\lambda_k)N^{m_k-1}}{(m_k - 1)!}$$  \hspace{1cm} (5)$$
which does not include terms containing $N^j$ for $j \geq m_k$ since these are 0.

### 1.2 Density Functional Theory

An application of functions of matrices can be found in computational quantum chemistry. Often the problems come from approximating the Hamiltonian in a quantum mechanical system. In physics the Hamiltonian is an operator $\hat{H}$ which describes the total energy of a particular system, and in quantum mechanics it gives rise to the eigenvalue problem described by the time independent Schrödinger equation

$$\hat{H}\psi_i = E_i\psi_i$$  \hspace{1cm} (6)$$
where $\psi_i(r)$ is a wave function such that $\int_{-\infty}^{\infty} \psi_i^*\psi_i dr = 1$ and $\mathbf{r}(x_1(s_1), \ldots, x_{N_e}(r_{N_e}, s_{N_e}))$ a vector of $N_e$ components, which denote the...
number of electrons in the system of study and \( r_j \) the spatial position of the \( j \)th electron and \( s_j \) the spin of the \( j \)th electron. Integrating the square of \( \psi_i \) over some region \( \Omega \) gives the probability of finding the electrons in this region, that is

\[
\rho_i(r) = \int_{\Omega} \| \psi_i(x_1, \ldots, x_{N_e}) \|^2 dx_1 \ldots dx_{N_e}
\]  

(7)

The probability density \( \rho_i(r) \) completely describes the system, as per the postulates of Quantum Mechanics. However, computing the probability density exactly can only be done for a very limited subset of systems of interest; which is why approximations and efficient computations are needed.

Approximation techniques of the Hamiltonian often come from the use of Hartree-Fock or Kohn-Sham density functional theory, which construct a effective Hamiltonian matrix \( H \) with elements given by

\[
H_{ij} = \int \phi_i^*(r) H \phi_j(r) \, dr
\]  

(8)

with \( H \) the effective Hamiltonian operator (or sometimes the Kohn-Sham or Fock Hermitian operator) and \( \{ \phi_i \}_{i=0}^{N} \) a finite basis set for the Hartree-Fock or Kohn-Sham framework respectively \cite{8,9}. The molecular orbitals are the eigenstates \( \{ \phi_i \} \) which are obtained from solving the non-linear Kohn-Sham eigenvalue equation self-consistently \cite{10} (which also defines the Kohn-Sham operator \( H \))

\[
H \phi_i = \left( -\frac{1}{2} \nabla^2 + V_{KS}(R, \rho)(r) \right) \phi_i = \epsilon_i \phi_i
\]  

(9)

with \( V_{KS} \) the effective single particle Kohn-Sham potential \cite{10}. In practice the finite basis set \( \{ \varphi_i \} \) can be used to form \( \phi \) by taking linear combinations of the basis functions, such that

\[
\phi_i = \sum_i C_{i,j} \varphi_i
\]  

(10)

using the factors \( C_{i,j} \) the matrix \( C \) can be constructed, which in turn leads to the construction of the density matrix(non-orthogonalized) \( D \) by

\[
D = C \Theta C^T
\]  

(11)

with \( \Theta \) a diagonal matrix with \( \Theta_{ii} \) the occupation factors, since the probability density in Kohn-Sham is defined as \( \rho(r) = 2 \sum_i \Theta_i \| \phi_i \|^2 \) \cite{10} (the pre-factor 2 comes from each orbital obeying the Pauli Exclusion principle, such that each orbital houses a spin up and a spin down electron). Performing energy minimization leads to the generalized eigenvalue problem

\[
HC = OCE
\]  

(12)
where $O$ is the overlap matrix with elements $O_{ij} = \int \varphi_i \varphi_j d\mathbf{r}$ and $E = \text{diag}(\epsilon_1, \ldots)$. Transforming the generalized eigenvalue problem into a standard eigenvalue problem is done by defining $Z$

$$Z^T O Z = I$$

which can be achieved by, for example, taking $Z$ equal to the inverse of the Cholesky factorization of $O$. Then the orthogonal form of the eigenvalue problem is obtained by taking

$$H \mapsto Z^T H Z \quad D \mapsto Z^T D Z \quad C \mapsto Z^{-1} C$$

which leads to

$$HC = CE$$

Constructing the density matrix $D$ of the system is done by superposing the outer product of the eigenvectors of $H$ \[9\]

$$H v_i = \epsilon_i v_i$$

which correspond to the $N_{\text{occ}}$ occupied electron orbitals of the system

$$D := \sum_{i=1}^{N_{\text{occ}}} v_i v_i^T$$

for which the eigenvalues are arranged in a list of ascending order \[9\]. Throughout this report it should be assumed, unless stated otherwise, that the effective Hamiltonian and Density matrix is in the orthogonalized form. One obvious way to construct $D$ would be to just diagonalize $H$ and take the outer products. Another approach is to construct $D$ by mapping $H$ to $D$ using the scalar Fermi-Dirac-distribution \[8\]. $F(x)$ defined as this is indeed a matrix function as per definition \[1.1\]

$$F(x) = \frac{1}{e^{\alpha(x-\mu)} + 1}$$

with $\alpha = \frac{1}{k_B T}$ ($k_b$ is the Boltzmann constant and $T$ the electronic non-zero temperature) and $\mu$ the chemical potential, such that

$$D = F(H) = V F(\Lambda) V^T$$

where $V$ are the change of basis matrices to the eigenbasis of $H$ and $\Lambda$ is the block diagonal matrix (here just diagonal since $H$ is Hermitian). The problem of the above approaches is that they require an eigendecomposition of $H$, which for a system of size $N$ carries a computational cost which scales as $\mathcal{O}(N^3)$. 


2 Recursive Fermi Dirac operator expansion

One way of circumventing the $O(N^3)$ computational scaling is by the use of recursive Fermi-Dirac operator expansion methods, which use recursive applications of low order polynomials to approximate the Fermi-Dirac-distribution [8]. In particular if electronic states are studied at 0K, the Fermi-Dirac-distribution is given by a Heaviside step function centered at the chemical potential such that

$$D = \theta(\mu - H)$$ (20)

There are many methods to expand this matrix function. Some common schemes revolve around expanding the function recursively at an already known chemical potential by using the McWeeny polynomial $P_{McW} = 3X^2 - 2X^3$ with initial rescaling of the effective Hamiltonian $X_0 = \alpha(\mu I - H) + \beta I$ with $\alpha = \min\{\beta[\lambda_{\max} - \mu]^{-1}, (1 - \beta)[\mu - \lambda_{\min}]^{-1}\}$. $\beta \in [0, 1]$ is chosen to give $P_{McW}(x)$ an inflection point when $P_{McW}(x) = x$. This method is sometimes referred to McWeeny purification.

The McWeeny purification scheme has an obvious drawback in that the chemical potential must be pre-determined prior to starting the construction of the density matrix. Another purification scheme proposed by Palser-Manolopoulos (PM) [3, 5] approaches the problem differently leading to a reduction in complexity, as the chemical potential now does not have to be known prior to the expansion. The PM scheme starts with a normalizing guess for the density matrix, in the same way as the McWeeny purification scheme does, but instead using the parameters

$$\alpha = \frac{1}{N} \min \left( \frac{N_e}{\lambda_{\max} - \mu}, \frac{N - N_e}{\mu - \lambda_{\min}} \right)$$ (21)

$$\beta = \frac{N_e}{N}$$ (22)

$$\mu = \frac{\text{Tr}[H]}{N}$$ (23)

where $N_e$ denotes the total number of electrons in the system in study. The trace-conserving purification of $H$ is then done by recursively applying the polynomials

$$X_{n+1} = \begin{cases} \frac{(1+c_n)X_n^2 - X_n^3}{1-c_n} & c_n \geq \frac{1}{2} \\ \frac{c_n}{1-2c_n}X_n + (1+c_n)X_n^2 - X_n^3 & c_n \leq \frac{1}{2} \end{cases}$$ (24)

with

$$c_n = \frac{\text{Tr}[X_n^2 - X_n^3]}{\text{Tr}X_n^2 - \text{Tr}X_n^2}$$ (25)

This scheme will not map any eigenvalues out of $[0, 1]$ [5], which ensures stability, but it will suffer at low and high occupancies as the initial transformation will then get a derivative close to 1, which leads to stalling of the method [5].
Another set of polynomials which according to Niklasson [4] converges to a step function (when applied recursively) at the correct chemical potential without prior knowledge of the chemical potential, are the polynomials

\[
\begin{align*}
P^a_1 &= X^2 \\
P^b_1 &= 2X - X^2
\end{align*}
\]  
(26)

which from now on will be referred to as the SP2-polynomials, that is

\[
\theta(\mu I - H) = \lim_{n \to \infty} (nP^{a/b}_1(...(0P^{a/b}_1(H)...))
\]  
(27)

The choice of which polynomial to use in each iteration can be based on the trace condition that Tr\[D\] = \(N_c\). It can be seen that Tr\[P^a_1(X)\] < Tr\[X\], assuming that X has its eigenvalues in [0, 1] (which is normally done by rescaling the Hamiltonian) and that the opposite holds for \(P^b_1(X)\). Avoidance of unnecessary iterations in the recursive expansion can be done by utilizing the idempotency condition for the density matrix, that is \(D = D^2\) [9]. Introducing the idempotency error \(e_i = \|D - D^2\|_F\), with \(\|\cdot\|_F\) the Frobenius norm, a parameterless stopping criterion in the SP2 algorithm can be formulated [9], as can be seen in Algorithm 1.

### Algorithm 1 SP2 algorithm with stopping criterion

**Input:** \(H, \lambda_{\text{min}}, \lambda_{\text{max}}\)

\[X_0 = \frac{\lambda_{\text{max}}I - H}{\lambda_{\text{max}} - \lambda_{\text{min}}}\]

\[e_0 = \|X_0 - X_0^2\|_F\]

\[C_{sp}^2 = 6.8872\]

for \(i = 1, 2, \ldots\) do

if \(\text{Tr}[X_{i-1}] > N_{occ}\) then

\[X_i = X_i^2\]

\[p_i = 1\]

else

\[X_i = 2X_{i-1} - X_{i-1}^2\]

\[p_i = 0\]

end if

\[e_i = \|X_i - X_i^2\|_F\]

if \(i \geq 2\) and \(p_i \neq p_{i-1}\) and \(e_i > C_{sp}^2 e_{i-2}^2\) then

\[n = i\]

\[e^{(0)} = e_i\]

break

end if

end for

**Output:** \(n, X_n\)

It is also possible to deviate from the monotonically increasing polynomials suggested in [2, 3, 4] and to use non-monotonically increasing poly-
mials, with a lower total number of matrix multiplications (which is the computationally expensive part of Algorithm 1). A scale and fold technique introduced by Rubensson [6] builds on Algorithm 1 but allows for the eigenspectrum of the rescaled effective Hamiltonian to be outside of the eigenspectrum [0, 1], before applying the polynomials; which is also based on the trace of each iteration; applying $P_1^{(a)}$ for too high occupations and $P_1^{(b)}$ for too low occupations. The scaling part is the choice of the parameter $\alpha \geq 1$, and this is chosen as to stretch out the eigenspectrum below 0 before the application of $P_1^{(a)}$ and above 1 for $P_1^{(b)}$, explicitly this is done by

$$\begin{cases} P_1^{(a)}(\alpha X_{i-1} + (1 - \alpha)I) \\ P_1^{(b)}(\alpha X_{i-1}) \end{cases}$$

(28)

These polynomials were used in devolving the accelerated SP2 algorithm (SP2 Acc) [6]. To use the stopping criterion for the accelerated SP2 algorithm, the acceleration must first be turned off, which is seen in Algorithm 2. In the SP2 Acc Algorithm, the parameters $\beta$ and $\overline{\beta}$ approach 0 and 1 respectively, which makes $\alpha \to 1$. To use the stopping criterion, there should be a predefined tolerance $tol$ which turns off the acceleration by setting $\beta = 0$ and $\overline{\beta} = 1$, when $\beta < tol$ and $||\overline{\beta} - 1|| < tol$, which yields the original SP2 scheme and makes it possible to implement the stopping criterion developed in [6]. There is not really a need for ultra-fine detection of when the acceleration should be turned off, since this will only impact the number of iterations using acceleration and not the final density matrix.
Algorithm 2 SP2 Acc algorithm with stopping criterion

Input : \( H, N_{\text{occ}}, \lambda_{\text{min}}, \lambda_{\text{max}}, \lambda_{\text{lumo}}, \lambda_{\text{homo}} \)

\[
X_0 = \frac{\lambda_{\text{max}} I - H}{\lambda_{\text{max}} - \lambda_{\text{min}}}
\]

\[
\beta = \frac{\lambda_{\text{max}} - \lambda_{\text{lumo}}}{\lambda_{\text{max}} - \lambda_{\text{min}}}
\]

\[
\overline{\beta} = \frac{\lambda_{\text{max}} - \lambda_{\text{homo}}}{\lambda_{\text{max}} - \lambda_{\text{min}}}
\]

\[
e_0 = \| X_0 - X_0^2 \|_F
\]

\( C_{2}^{sp} = 6.8872 \)

\( \gamma = -2 \)

\( tol = 0.1 \)

for \( i = 1, 2, \ldots \) do

if \( \text{Tr}[X_{i-1}] > n_{\text{occ}} \) then

\[\alpha = \frac{2}{2 - \beta}\]

\[X_i = (\alpha X_{i-1} + (1 - \alpha)I)^2\]

\[\beta = (\alpha \beta + 1 - \alpha)^2\]

\[\overline{\beta} = (\alpha \overline{\beta} + 1 - \alpha)^2\]

\( p_i = 1 \)

else

\[\alpha = \frac{2}{1 + \beta}\]

\[X_i = 2\alpha X_{i-1} - \alpha^2 X_{i-1}^2\]

\[\beta = 2\alpha \beta - \alpha^2 \beta^2\]

\[\overline{\beta} = 2\alpha \overline{\beta} - \alpha^2 \overline{\beta}^2\]

\( p_i = 0 \)

end if

\[e_i = \| X_i - X_i^2 \|_F\]

if \( \gamma < 0 \) and \( \| \overline{\beta} - 1 \| < tol \) and \( \| \beta \| < tol \) then

\[\overline{\beta} = 1, \beta = 0\]

\( \gamma = i \)

end if

if \( i \geq \gamma + 2 \) and \( p_{i-1} \neq p_i \) and \( e_i > C_{2}^{sp} e_{i-2}^2 \) then

\( n = i \)

break

end if

end for

Output: \( X_n, n \)
2.1 Numerical experiments for SP2 and SP2 Acc in Python

The Python programming was centered around the following parts

- class Hamiltonian
- function implementing SP2
- function implementing SP2 Acc

In order to test the implemented algorithms in Python, the results from Rubensson [6] where used as a benchmark. This meant that there was a need for a smooth way of constructing effective Hamiltonians with different bandgaps. In practice this was done by writing the python class Hamiltonian. The class is structured so that the user initiates a class object by providing the number of electrons $N$ in the system, the occupation factor $OF$ (number of occupied orbitals are then $N_{occ} = N \cdot OF$), $\lambda_{min}, \lambda_{max}, \lambda_{HOMO}, \lambda_{LUMO}$. The class then requires the user to initialize the Hamiltonian, which will generate a symmetric dense matrix using the Python package Numpy [12]. This is done in the Hamiltonian class by first creating a list of eigenvalues corresponding to occupied orbitals, which are linearly spaced. Then a second list of eigenvalues corresponding to unoccupied orbitals are constructed and this list is then concatenated with the other list of eigenvalues. A diagonal matrix $\Lambda = diag(\lambda_0, \ldots, \lambda_N)$ is then constructed. Generating a new matrix $V$ consisting of random orthonormal vectors was done using the package Scipy [13] which then allows the Hamiltonian to be created as $H = V\Lambda V^T$. Verifying the correct implementations was then done by creating Hamiltonians with bandgaps $\xi = 10^{-6}, 10^{-5}, \ldots, 10^{-1}$, $N = 100$, $OF = 0.5$, $\lambda_{min} = 0$, $\lambda_{max} = 1$, $\lambda_{HOMO} = 0.5$ and $\lambda_{LUMO} = \lambda_{HOMO} + \xi$ and then computing $D_{approx}$ using the SP2 and SP2 Acc schemes, until an error of $\|D - D_{approx}\|_2 < 10^{-9}$ was reached and then recording the number of matrix multiplications needed to reach this error tolerance. The Hamiltonian class can generate the correct density matrix by taking the outer product as shown in equation 17.

Showing correct implementations of the algorithms in accordance to Rubensson [6] the results for the number of matrix multiplications until an error of $\|D_{approx} - D\|_2 \leq 10^{-9}$ was reached for the accelerated and non-accelerated SP2 expansions are presented in figure [1].
Figure 1: Number of matrix multiplications needed for the accelerated and non-accelerated SP2 expansions until an error $\|D_{\text{approx}} - D\|_2 \leq 10^{-9}$ was reached for systems of different bandgaps.

2.2 C++ programming model for basic matrix library and SP2 and SP2 Acc implementations

The C++ programming model revolved around the following headers

- *Matrix.h*: Contains a class implementation of a column major matrix.
- *PolynomialExpansionAlgos.h*: Contains class with class methods which are implementations of SP2 and SP2 Acc.
- *PolynomialExpansionAlgosUtilities.h*: Contains a struct for creating linearly spaced vectors, function for generating linearly spaced eigenvalues, function for generating Hamiltonians, function for generating symmetric perturbation with elements from $\mathcal{N}(0, 1)$.

The Matrix class is contained in a headerfile and is a template with template parameter the type of matrix elements that the user wants to use in their programming, be it integers, doubles for complex types. The core ideas behind the basic matrix library is that it should be optimized for dense matrix multiplications, which the software achieves by calling Basic Linear Algebra Subroutines (BLAS) \[^{[14]}\]. The Matrix class also provides functionality for other matrix operations, such as addition, scalar-matrix multiplication, taking the Schur product of matrices, transposing, taking the trace of a matrix, commutator operator implementation, as well as using LAPACK \[^{[15]}\] for finding the inverse of a lower triangular matrix and for computing the
Cholesky factorization of a positive semi-definite Hermitian matrix.

The class representing the polynomial expansion algorithms is also a template for both the type of matrix used to represent the Hamiltonian and the type of matrix elements used in the Hamiltonian.

2.3 Numerical experiments in C++

Test runs were performed using $N = 100, \lambda_{\text{min}} = 0, \lambda_{\text{max}} = 1.0, \lambda_{HOMO} = 0.495, \xi = 0.01, OF = 0.5$. Results are shown in figures 2, 3. Max iterations where 60 for both schemes.

![Figure 2](image.png)

Figure 2: Example run with SP2 C++ implementation showing stopping condition iteration for a Hamiltonian with $N = 100$ and $N_{\text{occ}} = 50$. 
Figure 3: Example run with SP2 Acc C++ implementation showing stopping condition iteration for a Hamiltonian with $N = 100$ and $N_{occ} = 50$.

3 Derivatives of Functions of Matrices and Density Matrix Perturbation Theory

As for scalar functions, it is possible to define a derivative of matrix functions; which should have some similar characteristics as the derivative for scalar functions.

Definition 3.1 The Gateaux derivative of a matrix function in the direction of a matrix $E \in \mathbb{C}^{n \times n}$ of $f : \mathbb{C}^{n \times n} \to \mathbb{C}^{n \times n}$ is a mapping between two Banach spaces $(V, \| \cdot \|)$ (in this case $V = \mathbb{C}^{n \times n}$ with $\| \cdot \|_F$ or $\| \cdot \|_2$ onto itself) such that

$$\lim_{h \to 0} \frac{f(X + hE) - f(X)}{h} = \frac{d}{dt} \bigg|_{t=0} f(X + tE)$$ \hspace{1cm} (29)

Related to the Gateaux derivative is the Fréchet derivative, which is a stronger derivative as its existence guarantees the existence of the Gateaux derivative for all vectors in $V$.

Definition 3.2 The Fréchet derivative of a matrix function $f : V \to V$ at $X \in V$ is defined as

$$\lim_{\|E\| \to 0} \frac{\|f(X + E) - f(X) - L(X, E)\|}{\|E\|} = 0 \quad \text{for } E \in V$$ \hspace{1cm} (30)

given that $\exists L(X, E)$ a bounded linear map. Then $L(X, E)$ is the Fréchet
derivative of \( f \) at \( X \) in the direction of \( E \) [1]. If \( \exists L(X, E) \) as above \( \forall E \in V \) then \( f \) is Fréchet differentiable.

It is straightforward to see that the Fréchet derivative has some properties familiar from scalar derivatives; linearity being the most prominent.

In the case of evaluating \( f \) at symmetric matrices \( X = X^T \in V \) the following theorem states an explicit formula for computing the Fréchet derivative.

**Theorem 3.1 (Daleckii-Krein [1, 16])** Letting \( A = ZDZ^{-1} \in \mathbb{C}^{n \times n} \) with \( D \) diagonal and \( f \) be continuously differentiable on the spectrum of \( A \). The Fréchet derivative of \( f(A) \) is then equal to

\[
L(A, E) = Z(F \odot (Z^{-1}EZ))Z^{-1}
\]

(31)

where \( \odot \) denotes the element wise matrix product (Schur/Hadamard product) and \( F \) is a matrix with divided difference elements defined as

\[
F_{ij} = \begin{cases} 
  \frac{f(D_{ii}) - f(D_{jj})}{D_{ii} - D_{jj}} & \text{if } D_{ii} \neq D_{jj}, \\
  f'(D_{ii}) & \text{if } D_{ii} = D_{jj}
\end{cases}
\]

(32)

A formula for computing the Gateaux derivative for non symmetric matrices was developed by Mathias in [17], as seen in the following theorem.

**Theorem 3.2 (Chain Rule for matrix functions [17])** Let \( f \) be \( 2m - 1 \) (\( m \) the size of the largest Jordan block) differentiable on an open subset \( D \subset \mathbb{R} \). Let \( A(t) \) be differentiable at \( t_0 \) and assume \( A(t) \in \mathbb{C}^{n \times n} \forall t \) in some neighborhood of \( t_0 \). Then

\[
\frac{d}{dt}f(A(t)) \bigg|_{t=t_0} = \left[ f\left( \begin{bmatrix} A(t_0) & A'(t_0) \\ 0 & A(t_0) \end{bmatrix} \right) \right]_{12}
\]

(33)

with subscript 12 meaning "take the upper right block of block matrix".

Then, as Higham discusses in [1], letting \( A(t) = A + tE \) with \( A, E \in \mathbb{C}^{n \times n} \) yields the special case of

\[
f\left( \begin{bmatrix} A & E \\ 0 & A \end{bmatrix} \right) = \begin{bmatrix} f(A) & \frac{d}{dt}f(A + tE) \\ 0 & f(A) \end{bmatrix} 
\]

(34)

This provides some important computational aspects. Given a matrix function \( f \) which is evaluated at some matrix \( H^{(0)} \), computing the derivative of \( f(H^{(0)}) \) due to some perturbation matrix \( H^{(1)} \) all that is needed is to map \( H^{(0)}, H^{(1)} \) onto a block upper triangular matrix \( H \) such that

\[
H = \begin{bmatrix} H^{(0)} & H^{(1)} \\ 0 & H^{(0)} \end{bmatrix}
\]

(35)
and then compute $f(H)$ and then take the right upper block of $f(H)$ to find 
\[
\frac{d}{dt} \bigg|_{t=0} f(H(0) + tH^{(1)}).
\]

Introducing a time independent perturbation $H^{(1)}$ to the effective Hamiltonian $H$ used to construct the density matrix $D$ through the Fermi-Dirac distribution, could in practice occur by applying an external static electric or magnetic field. In view of this, the effective Hamiltonian can be expanded to first order as
\[
H = H^{(0)} + \epsilon H^{(1)} + \ldots
\] (36)

It is then expected to see some similar perturbation in $D$ such that
\[
D = D^{(0)} + \epsilon D^{(1)} + \ldots
\] (37)

where $D^{(1)}$ can be identified as
\[
D^{(1)} = \frac{d}{d\epsilon} \bigg|_{\epsilon=0} \theta(\mu I - H^{(0)} - \epsilon H^{(1)})
\] (38)

which by definition 3.1 is the Gateaux derivative of $\theta(\mu I - H^{(0)})$ in the direction of $H^{(1)}$, which from now on is denoted by $G(\theta, H^{(1)})$. As the notation in equation 35 eludes to, one way of computing $G(\theta, H^{(1)})$ is to map $H^{(0)}, H^{(1)}$ onto a block upper triangular matrix $H$, and then use theorem 3.2 to compute $G(\theta, H^{(1)})$.

The preferred method of doing this would be to use the previous discussed SP2 and SP2 Acc algorithms to expand the Fermi-Dirac operator, and employ this in theorem 3.2. The answer to the question of when $G(\theta, H^{(1)})$ is deemed to have converged is given by the parameter free stopping criterion presented in [10], which utilizes results of [9] and theorem 3.1 to devise the stopping criterion presented in Algorithm 3.

Algorithm 3 Parameter free convergence criterion for $D^{(1)}$

if $D^{(0)}$ converged = true then
    FroErr$_n^{(1)} = \|X_{n-1}^{(1)} - (X_{n-1}^{(0)} X_{n-1}^{(1)} + X_{n-1}^{(1)} X_{n-1}^{(0)})\|_F$
    if FroErr$_n^{(1)} > 9 \times e^{(0)} \times$ FroErr$_n^{(1)}$ then
        $D^{(1)}$ converged = true
    end if
end if

if $D^{(1)}$ converged = false then
    Update $X$ according to SP2.
end if
3.1 Python programming model for computing the Gateaux derivative

The Hamiltonian class also allows the user to generate a symmetric perturbation matrix $H^{(1)}$ with standard normally distributed elements, which was used in the computation of the Gateaux derivative of the density matrix in the direction of the perturbation. Mapping the unperturbed Hamiltonian $H^{(0)}$ and the perturbation $H^{(1)}$ onto a block upper triangular matrix $H$ is also done by the class Hamiltonian, by

$$H = I_{2 \times 2} \otimes H^{(0)} + UT \mathbf{1}_{2 \times 2} \otimes H^{(1)}$$  \hspace{1cm} (39)$$

where $\otimes$ is the kronecker product, $I$ a $2 \times 2$ identity matrix and $UT \mathbf{1}_{2 \times 2} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$, by utilizing Numpy [12].

To test the computation of $D^{(1)}$ using the analytical method in theorem 3.2 a block upper triangular Hamiltonian $H$ was generated, with $H^{(0)}$ a $100 \times 100$ matrix with $N_{occ} = 50, \lambda_{max} = 1, \lambda_{min} = 0, \xi = 0.01, \lambda_{HOMO} = 0.495$ and with $H^{(1)}$ a symmetric $100 \times 100$ matrix with elements from the $\mathcal{N}(0, 1)$ distribution. These were then mapped onto a block upper triangular matrix (which is a $200 \times 200$ full dense matrix) which is then plugged into the SP2 and SP2 Acc scheme. These calculations were carried out in both double (floating point 64) and single (floating point 32) precision, in order to check their relative error.

Due to the form of the Gateaux derivative, it is also possible to approximate $D^{(1)}$ using finite difference schemes. A first order approximation is

$$D^{(1)} \approx \frac{\theta(\mu I - H^{(0)} - hH^{(1)}) - \theta(\mu I - H^{(0)})}{h}$$  \hspace{1cm} (40)$$

for some step size $h > 0$. This approximation was also used to compare to the analytical derivatives, using single precision arithmetic and step sizes $h \in \{10^{-13}, 10^{-12}, \ldots, 10^{-3}\}$ and of course approximating $\theta(\mu I - H^{(0)}), \theta(\mu I - H^{(0)} - hH^{(1)})$ by a polynomial expansion provided by SP2 and SP2 Acc.

3.2 Python computations

The relative error in the computation of the Gateaux derivative in the direction of a symmetric perturbation with elements drawn from a $\mathcal{N}(0, 1)$ distribution for the SP2 and SP2 Acc recursive expansions using the block upper triangular method in single precision as well as using a first order finite difference method in single precision, compared to the analytical result obtained by the Daleckii-Krein formula in double precision can be seen in figure 4.

They are plugged in with $N_{occ} = 2 \cdot N_{occ}$ since the block matrix will have twice as many electron orbitals occupied as $H^{(0)}$. 

16
Figure 4: Relative errors for the first order finite difference approximation and the block upper triangular method for computing the Gateaux derivative in the direction of a symmetric perturbation with elements drawn from a $\mathcal{N}(0, 1)$ distribution. The true value of $D^{(1)}$ is taken as the analytical result provided by the Daleckii-Krein formula in Theorem 3.1 using double precision.

A comparison on the calculated norms of $D^{(1)}$ for SP2 and SP2 Acc in single and double precision can be found in table 1 using the Daleckii-Krein result in double precision as reference.

Table 1: Comparison of norms for $D^{(1)}$ computed in both single and double precision to the analytical result of Daleckii-Krein computed in double precision

<table>
<thead>
<tr>
<th>Precision</th>
<th>$|D^{(1)}|$(SP2)</th>
<th>$|D^{(1)}|$(SP2 Acc)</th>
<th>$|D^{(1)}|$(Daleckii-Krein)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>288.4427100728525488</td>
<td>288.4427100728519235</td>
<td>-</td>
</tr>
<tr>
<td>Double</td>
<td>288.4427043740966496</td>
<td>288.4427043741030161</td>
<td>288.4427043740940917</td>
</tr>
</tbody>
</table>

Using the values from table 1, the relative errors in $D^{(1)}$ for SP2 and SP2 Acc were computed by the formula

$$\text{Relative Error} = \frac{\|D^{(1)} - D^{(1)}_{\text{ref}}\|_F}{\|D^{(1)}_{\text{ref}}\|_F}$$

and these are presented in table 2.
### Table 2: Relative errors for $D^{(1)}$ in SP2 and SP2 Acc for using single compared to double precision, using $D^{(1)}$ computed by the formula provided by Daleckii-Krein.

<table>
<thead>
<tr>
<th>Precision</th>
<th>Relative Error in $D^{(1)}$ (SP2)[%]</th>
<th>Relative Error in $D^{(1)}$ (SP2 Acc)[%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>0.00002112832209</td>
<td>0.00002112832214</td>
</tr>
<tr>
<td>Double</td>
<td>0.000000000005495</td>
<td>0.000000000022929</td>
</tr>
</tbody>
</table>

#### 3.3 C++ programming model for computing the Gateaux derivative

The C++ programming model for computation of the Gateaux derivative of the Fermi-Dirac operator, builds on providing a block upper triangular matrix library on the form of matrices used in theorem 3.2. The library is a template with template parameters the type of matrix to used as blocks and for the element type of the matrix blocks’ elements. utilizes the symmetry of these matrices and therefore only stores the diagonal block and the upper right block once respectively, the zero part of the block upper triangular matrix is never referenced.

$$H = \begin{bmatrix} H^{(0)} & H^{(1)} \\ 0 & H^{(0)} \end{bmatrix}$$

Storage in memory $\Rightarrow \begin{bmatrix} H^{(0)} \end{bmatrix}$

This carries with it some implications for common matrix-matrix operations such as multiplication and addition when used in this class. To exemplify this, consider three block upper triangular matrices, denoted by $A, B, C$. Performing $C = A + B$ is outlined in Algorithm 4.

#### Algorithm 4 Pseudo code for addition of two block upper triangular matrices.

* Input: $A, B, C$
* Get pointers to $A^{(0)}, B^{(0)}, C^{(0)}$.
* $\triangleright$ Denote by $*A, *B, *C$
* $*C[0] = *A[0] + *B[0]$

This algorithm assumes that the template matrix class used contains methods for re-sizing and addition. Performing $C = AB$ is outlined in Algorithm 5.

The matrix library also carries with it methods to calculate the trace of a block upper triangular matrix $A$, which simply becomes $2 \cdot \text{Tr}[A^{(0)}]$. These outlined methods lay the foundation for the SP2 and SP2 Acc algorithms.
**Algorithm 5** Pseudo code for multiplication of two block upper triangular matrices.

Input: A, B, C

Get pointers to \( A^{(0)} \), \( B^{(0)} \), \( C^{(0)} \). \( \triangleright \) Denote by \( \ast A \), \( \ast B \), \( \ast C \)

\[
\begin{align*}
\end{align*}
\]

3.4 C++ computations

Verifying the correct computation of \( D^{(1)} \) was done by comparing values to the results obtained in [10] for the computation of the isotropic polarizability of water clusters, with \( N = 1128 \) and \( N_{occ} = 235 \). The eigenvalue problem here is a little different to the previously encountered examples, which have assumed the effective Hamiltonian \( H \) to be in orthogonal form. Here the matrices provided from Ergo [18] where the effective Hamiltonian \( F \), first order perturbations \( F_x^{(1)}, F_y^{(1)}, F_z^{(1)} \), the overlap matrix \( S \) and the dipole matrices \( M_x, M_y, M_z \) but on the form for the generalized eigenvalue problem

\[
FC = OCE
\] (42)

which required \( S \) to be cholesky factorized and inverted to yield \( Z \) the inverse cholesky factorization of \( S \). The matrices where then transformed by \( Z \) such as

\[
H^{(0)} = Z^T FZ \quad H^{(1)}_{x,y,z} = Z^T F^{(1)}_{x,y,z} Z
\] (43)

and the responses \( D^{(1)}_{x,y,z} \) could be computed by mapping each orthogonalized perturbation onto a block upper triangular matrix object and then performing SP2 and SP2 Acc until convergence in \( D^{(1)}_{x,y,z} \). The isotropic polarizability \( \kappa \) is defined as

\[
\kappa = \frac{1}{3} \text{Tr} \left[ Z D_x^{(1)} Z^T M_x + Z D_y^{(1)} Z^T M_y + Z D_z^{(1)} Z^T M_z \right]
\] (44)

Results for \( \kappa \) and reference values from earlier computations performed in [10] are presented in table 3.

<table>
<thead>
<tr>
<th>Computed ( \kappa )</th>
<th>Ergo reference for ( \kappa )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-286.104477807</td>
<td>-286.104481906</td>
</tr>
</tbody>
</table>

Table 3: Computed value for the Isotropic polarizability \( \kappa \) for a water cluster with \( N = 1128 \) and \( N_{occ} = 235 \) compared to the ergo reference provided in [10].

The first order Frobenius error in each iteration in SP2 Acc and SP2 is shown in figure [5] and [6] until the stopping criterion for \( D_x^{(1)} \) was reached.
Figure 5: The first order Frobenius error as a function of iterations, when computing $D_x^{(1)}$ from the Ergo provided matrices using SP2 Acc.

Figure 6: First order Frobenius error as a function of iterations when computing $D^{(1)}$ from the Ergo provided matrices using SP2.
To test the performance of the block upper triangular matrix library compared to the basic matrix library outlined earlier, block Hamiltonians with \( H^{(0)}, H^{(1)} \) having size \( N \in \{100, 500, 1000, 1500, 2000\} \), using \( \text{Nocc} = \frac{N}{2} \), \( \lambda_{\text{min}} = 0, \lambda_{\text{max}} = 1, \lambda_{\text{HOMO}} = 0.495, \xi = 0.01 \) were generated. The CPU wall time on a Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz was then recorded for the SP2 and SP2 Acc schemes using double precision arithmetic, until the stopping criterion for \( D^{(1)} \) outlined in Algorithm 3 was reached. The results for the SP2 scheme are seen in figure 7 and the results for the SP2 Acc scheme are seen in figure 8. The BLAS version used was OpenBLAS 0.3.23 [19, 20].

![Wall time for SP2](image_url)

Figure 7: Wall time for different sizes of \( H^{(0)}, H^{(1)} \) using block upper triangular matrix class (BUTM) or the basic matrix library in SP2. Computations were carried out in double precision.
4 Discussion

The use of matrix functions in computational quantum chemistry certainly provides some interesting computational aspects. Using low order polynomials as a mean to approximate the Fermi-Dirac distribution in the SP2 and SP2 Acc algorithm as well as for computing $D^{(1)}$, shows the need for efficient software matrix representation for fast matrix-matrix multiplications.

The SP2, SP2 Acc algorithms and the stopping criterion for $D^{(1)}$ especially presents a real computational strength in the use of a user independent parameterless stopping criterions, which relives the user of having to choose an error tolerance for computing $D^{(0)}$ and $D^{(1)}$. In the SP2 Acc algorithm, a universal tolerance $tol = 0.1$ was used, since this was deemed to be a sufficient for not increasing the number of iterations in SP2 Acc and at the same time allowing for the implementation of the stopping condition in SP2. The choice of $tol$ is essentially arbitrary, but it should be chosen as not to be so course that the original SP2 scheme is redeemed and also not too fine, as this would cause unnecessary iterations when using low precision arithmetic. Choosing $tol$ to be in the higher end of this range would only result in less acceleration and, as previously mentioned, not effect the final density matrix.

The computation of the Gateaux derivative $D^{(1)}$ using the formula presented
in 3.2 compared to using a finite difference method also sheds light on the advantage of using parameterless stopping criterions as well as relieving the user of the hassle of choosing step size in the finite difference method. This is readily apparent from the figure 4, which shows that choosing the smallest step size is not the best option due to floating point arithmetic.

It is also shown that the methods, that include the basic and upper triangular matrix libraries built from scratch in this work, as well as the SP2 and SP2 Acc algorithms, work on not only artificially generated orthogonalized Hamiltonians, but also on actual non-orthogonalized matrices resulting from Hartree-Fock calculations. The advantages of the accelerated algorithm also present themselves when comparing figures 5 and 6 as well as from inspecting 1 that the accelerated algorithm requires far less matrix-matrix-multiplications than the non-accelerated.

Using the Block Upper Triangular matrix class is shown to provide faster computations for $D^{(1)}$, as seen in figures 7 and 8 as this class only has to perform a total of 3 block matrix multiplications, compared to the 8 block matrix multiplications that a straightforward full matrix class would have to perform. It is therefor expected that the wall times for the block upper triangular matrix class should be $3/8$ of the regular matrix class, which was essentially observed for the artificially created matrices with block size 2000 as seen in figures 7 and 8.

5 Conclusion

In this work earlier results from [6, 10] have been verified. Extended work in computing $D^{(1)}$ for the SP2 Acc algorithm has been presented. It was found that the relative error in $D^{(1)}$ was not significantly different when using SP2 acc compared to SP2.

The advantages of using parameterless stopping criterion as well as the advantages of using the formula presented in theorem 3.2 have been exemplified.

Matrix libraries for both a standard matrix dense matrix representation and for a block upper triangular matrix, which only directly references two blocks in a $2 \times 2$ block matrix, have also been devolved in C++ and these have been utilized in the implementations of SP2 and SP2 Acc and in the computation of $D^{(1)}$ for both artificially created and Ergo-provided effective Hamiltonians.
References


Populärvetenskaplig sammanfattning


För att få så effektiva beräkningar som möjligt, så skrevs en matrisklass för vanliga matriser och en matrisklass för blockövertriangulära matriser i C++. Implementationen av den blockövertriangulära klassen ger snabbare beräkningar av derivatan än den vanliga fulla matrisklassen.

Denna artikel bidrar till hållbarhetsarbete genom att betona vikten av optimaliserade algoritmer för att approximera Fermi-Dirac-operatorn, vilket leder till effektiva beräkningar. Genom att effektivisera beräkningarna så minskas klimatavtrycket för hela disciplinen och energibesparningar kan göras. Mjukvaruimplementationen av matrisklasserna ger även de effektivare beräkningar och denna typ av mjukvaruutveckling belyser vikten av vidare utvecklande av resurseffektiva arbetssätt och mjukvarulösningar.
Appendix

Link to Github page containing Matrix library:

Figure 9: Link to Github page containing code for matrix implementations.