

多重解像度多重ウェーブレット基底の量子化学計算への応用

2013年1月

博士（工学）

加藤 哲也

豊橋技術科学大学

2013年 1月

機能材料 工学専攻	学籍番号	D053708	指導教員	関野 秀男
申請者 氏名	加藤 哲也			栗田 典之

論文要旨（博士）

論文題目	多重解像度多重ウェーブレット基底の量子化学計算への応用
------	-----------------------------

(要旨 1,200字程度)

量子力学に基づく計算は、基礎理論は20世紀初頭に整ったが、当時は現実的に不可能な量の計算が必要であった。電子コンピュータの発明以降は応用的な計算が可能となり、化学の分野でも、Schrödinger方程式の平均場近似によるHartree-Fock方程式を行列方程式として表した、Roothaan方程式を基盤として、Hartree-Fock, post-Hartree-Fockの各手法、そして密度汎関数理論とKohn-Sham方程式による各種の解法が実装されてきた。基底状態の電子状態を解くだけにとどまらず、エネルギーの核座標微分の計算により、第一原理的な分子構造計算が可能となり、これは特に、理論の専門家ではない、実験家達にもカジュアルに用いられるほど普及した。

しかしながら、伝統的なLCAO型の基底では、大規模系の計算で必要となる巨大な固有値計算を実行することが、現代的な超並列計算機を用いても簡単ではないことや、特に応答関数を必要とする計算で物性の定量的な評価が難しいことが知られるようになってきた。

MADNESSは、離散ウェーブレット変換を基礎にした3次元直交空間上の基底である多重解像度多重ウェーブレット(Multi-Resolution Multi-Wavelet; MRMW)基底の技術をベースに、超並列計算を前提に設計された科学計算ライブラリである。MADNESSはC++で記述され、アプリケーション開発者はほとんど並列化を意識することなく、数式に近い形でアルゴリズムを記述できる。MADNESS上では基本的なHartree-Fock、及びLDA(Local Density Approximation)によるDFT計算が実装されている。

本研究では、既存の量子化学計算に利用される技術を、MADNESSを利用して動作するプログラムへ実装することにより、既存技術のLCAO依存性を明らかにし、3次元直交基底上での応用可能性について調べた。

第1に、擬ポテンシャル法の一つであるModel Core Potentialを実装した。擬ポテンシャル法は内殻電子群を局所ポテンシャルと非局所ポテンシャルで近似し、変分空間を削減して、計算負荷を軽減する手法であり、LCAO基底では広く用いられている。本研究では新たなパラメータフィッティング手法を提案し、その精度を確かめた。

第2に、CPHF/KS(Coupled Perturbed Hartree-Fock/Kohn-Sham)法を実装し、水素分子共鳴領域の周波数依存動的分極率について、LCAO基底との比較を行った。水素分子のような最も単純な分子であっても、LCAOでは精度を上げるために恣意的な基底選択を行う必要があるが、MRMW基底ではそのようなことはないことが確認できた。

これらの実装、計算を通じ、LCAO基底で用いられた種々の手法の直交空間上の基底への移植性と、各々の基底の特徴について考察した。

year month day
2013 1

Department	Functional Materials Engineering	ID	D053708	Supervisor	SEKINO Hideo
Name	KATO Tetsuya				KURITA Noriyuki

A b s t r a c t

Title	Application of Multi-resolution Multi-wavelet for the quantum chemistry
-------	---

(800 words)

In early 20th century, basic theories are prepared for quantum dynamics calculation. However, it was impossible then to execute the calculation because it requires large amount of calculation. After invention of electronic computer, it becomes possible even for applicative purpose.

In chemistry, many methods are implemented such as Hartree-Fock, post-Hartree-Fock methods, the Kohn-Sham equation by Density Functional Theory based on the Roothaan equations that represent the Hartree-Fock equation which approximates the Schrödinger equation with mean field approximation as a matrix form. It is not only for electronic state of ground state but also *ab initio* geometry optimization using energy gradient by nuclear coordinates that becomes common to be used by experimentalists who are not a specialist of theory.

Nevertheless, using traditional LCAO (Linear Combination of Atomic Orbitals) type basis functions, it is difficult to solve large eigenvalue problem needed for a calculation of large molecular system or to estimate response properties quantitatively.

MADNESS is a scientific calculation library that targets massively parallel computing and using MRMW (Multi-Resolution Multi-Wavelet) basis that is constructed based on Discrete Wavelet Transform technique. Just enough number of MRMW basis functions are spread over 3D bounding box to represent target function. It is guaranteed that a function represented with the MRMW basis is close to the target function within a given threshold.

Basic operations for 3D real/complex-valued functions are implemented and Hartree-Fock and DFT using LDA (Local Density Approximation) functionalities are also implemented. During its execution, MRMW basis coefficients are transformed, and MRMW basis functions are deleted or spawned as necessary by function operations automatically. MRMW basis functions are spawned depending not on nuclear geometries but on the shape of target function.

MADNESS solves the SCF problem using a Green's function for the Bound State Helmholtz equation. Molecular orbitals are solved as a set of orbitals under Hartree-Fock/DFT potentials generated by other electrons one by one. Therefore, there are no need to solve eigenvalue problem. Orbital updating procedure spends only O(N). If using LDA-DFT, all of calculations are done in O(N). To say fairly, transformation of MRMW coefficients tree consumes O(log M) which comes from target function complexity. Total calculation cost in order is still O(N log M) < O(N²) << O(N⁴). Multi-wavelet has shallower tree than Single-wavelet, so M is usually small.

MADNESS is written in C++ so that developers write applications like mathematical formulas, without need to concern about parallel techniques over MADNESS libraries.

In this study, we implement existing techniques used for calculations of traditional quantum chemistry to MADNESS to show capabilities to apply them to 3D orthogonal basis despite of LCAO dependencies of the techniques.

First, we implement a Model Core Potential which is one of pseudo-potentials. The pseudo-potential is a method commonly used in LCAO basis to reduce amount of calculation by replacing inner shell electrons with local and non-local potentials. It is also useful to calculate very heavy atoms which need relativistic treatment. Inner shell electrons of heavy atoms have very high velocity near the light speed but valence electrons move slowly. Hence we can calculate heavy atoms with non-relativistic equations if inner shell electrons are replaced to potentials including the relativistic effects. We find that MCP depends on basis set strongly and its consistency is easily broken by adding basis function even in LCAO fashion. We propose new method to fit potential parameters to make MCP consistent in MRMW and confirm its accuracy.

Second, we implement a CPHF/KS (Coupled Perturbed Hartree-Fock/Kohn-Sham) method to MADNESS and compare results against LCAO basis using dynamic polarizability depends on frequency of the external field with resonance region of H₂. Quantitative estimation of dynamic polarizability α in resonance region is hard because α can be represented by a equation with denominator with a different between external field frequency minus excitation energy. Therefore α quickly diverges for the frequency close to excitation energy. It is known that in LCAO basis, dynamic polarizability estimation requires diffuse type function in basis set. MRMW basis can represent diffuse type function and a function far from nuclei. LCAO basis are designed for representing ground state orbitals. Response functions which are needed for calculate α can have different characteristics against ground state orbitals.

We confirm that it is difficult to select a basis set automatically even with the simplest molecule H₂ to calculate accurately using LCAO. In contrast, it is done automatically with the MRMW basis. MRMW always reproduce target function with guaranteed accuracy even if we don't have any information of target function in advance. LCAO basis sets, or any basis function sets which have specific function type, needs information of target function shape.

We discuss interchangeabilities of methods developed in LCAO basis with the MRMW basis set and characteristics of both basis sets through these implementations and calculations.

目次

1	多重解像度多重ウェーブレット (MRMW) 基底	2
1.1	離散ウェーブレット変換と基底	2
1.2	MRMW 基底	3
2	分子分極率	6
3	分子軌道法における基底	8
4	Coupled Perturbed Hartree-Fock/Kohn-Sham	10
4.1	摂動論	10
4.2	LCAO 基底での CPHF/KS	11
4.3	任意完全基底での CPHF/KS	14
5	擬ポテンシャル法	20
5.1	Effective Core Potential	22
5.2	Pseudo Potential	23
5.3	Model Core Potential	24
6	総括	37

1 多重解像度多重ウェーブレット (MRMW) 基底

多重解像度多重ウェーブレット基底 (Multi-resolution Multi-wavelet basis; MRMW basis) は、米 Oak Ridge 国立研究所で開発された、新たな基底関数系である。

この基底関数系は、離散ウェーブレット変換 (Discrete Wavelet Transform; DWT) を基礎として設計されている。

1.1 離散ウェーブレット変換と基底

ウェーブレット変換とは、信号処理の分野で、時間-周波数空間における対象関数の特徴を捉えるために開発された、関数解析・変換手法である [1][2]。最初に対象領域を連続的に扱う連続ウェーブレット変換 (Continuous Wavelet Transform; CWT) が開発され、次いで、対象領域を 2^n 個に分割することにより、逆変換を保証した離散ウェーブレット変換が開発された。逆変換の保証により、これを、連続関数を計算機上で扱うための基底として利用することができる。

ウェーブレット変換以前に用いられていた、短時間フーリエ変換 (Short-Time Fourier Transform; STFT) では、対象の関数形状に関係なく、使用する窓関数によって時間分解能・周波数分解能が決まる。ウェーブレット変換では、不確定性原理 $\Delta x \Delta \omega \geq \frac{1}{2}$ を満たす範囲で、関数形状に最適な分解能を得ることができる。

離散ウェーブレット変換では、次のような scaling 関数 $\phi(x)$ 、および wavelet 関数 $\psi(x)$ を基底として関数を表現する。

$$\phi(x) = \sqrt{2} \sum_{n=0}^{N-1} P_n \phi(2x - n) \quad (1.1)$$

$$\psi(x) = \sqrt{2} \sum_{n=0}^{N-1} Q_n \phi(2x - n) \quad (1.2)$$

ここで、 $\{P_n, Q_n\}$ は $\phi(x), \psi(x)$ の形状を指定するパラメータであり、Daubechies や B-Spline 等、様々なものが提案されている。最も単純な Haar のウェーブレットでは、 $\{P_n\} = \{1, 1\}, \{Q_n\} = \{1, -1\}$ である。

離散ウェーブレット変換は信号処理の分野で発展してきたため、基本的には、ある周波数でサンプリングされた信号列を、段階的に粗い近似にすることで行われ、結果として二分木構造のデータが得られる。

まず、信号列を scaling 関数で補完して関数とみなす。式 (1.1) は再帰的な構造になつており、これをを利用して粗い近似の scaling 関数表現を得ることができる。サンプリング点数が 2^N の場合、最初に補完した scaling 関数は 2^N 個あるが、1 レベル上では 2^{N-1} 、その更に上では $2^{N-2} \dots$ となり、これが二分木構造となる。この表現では各レベルの scaling 関数群が、それぞれのレベル内で基底を成しているが、レベル間では重なりを持つ。

wavelet 関数は、レベル間の scaling 関数の差分として定義される。これにより、1 個の scaling 関数と、 $2^{N-1} - 1$ 個の wavelet 関数で、木構造を成す基底で表現された、係数列ができる。これを得るのが離散ウェーブレット変換である。

この木構造を、根の方から考えると、対象の関数によっては、区間内のあるレベルより深い wavelet 係数がすべて、ほぼ 0 になる場合がある。一定の閾値を設け、閾値を下回った wavelet 係数を持つノードを削除することにより、閾値によって保証された精度を保ったまま、対象関数の、より少ない係数による表現を得ることができる。

一般に、異なる 2 つの関数は異なる木構造を持つが、各区間でノードが多い方に合わせた木構造を作りなおすことで、関数同士の加減算ができる。また、wavelet 表現から scaling 表現に戻すことで、関数同士の乗算も可能である。このように、関数に対する種々の演算をこの基底の上で実装することが可能である。

1 次元の離散ウェーブレット基底を直交させることで、一般に N 次元の離散ウェーブレット基底を構築できる。この場合、区間は N 次元の箱状領域となり、2 分木構造は 2^N 分木構造となる。単純に直交させると、一般には軸ごとにレベルの異なる基底が生成されるが、すべての軸について同一のレベルの基底の積のみで構成された基底へ、変換することができる。

1.2 MRMW 基底

Multi-Resolution Multi-Wavelet(MRMW) 基底は、scaling 関数を $k+1$ 個の Legendre 多項式で分割した、Alpert[3] 級離散ウェーブレット基底である。Legendre 多項式は、区間 $[-1, 1]$ で定義された多項式関数であり、球面調和関数の導出などに用いられる。

0 次から $k = 4$ 次までの Legendre 多項式 $P_n(x)$ を式 1.3 および図 1 に示す。これらは関数の組 $\{x^n\}_k$ に対し、 $[-1, 1]$ で Gram-Schmidt の直交化を施すことで簡単に得られる。従って区間内で規格直交であり、区間外では厳密に 0 となるため、基底の重なりの問題は一切生じない。

$$\begin{aligned}
P_0(x) &= 1 \\
P_1(x) &= x \\
P_2(x) &= \frac{1}{2}(3x^2 - 1) \\
P_3(x) &= \frac{1}{2}(5x^3 - 3x) \\
P_4(x) &= \frac{1}{8}(35x^4 - 30x^2 + 3)
\end{aligned} \tag{1.3}$$

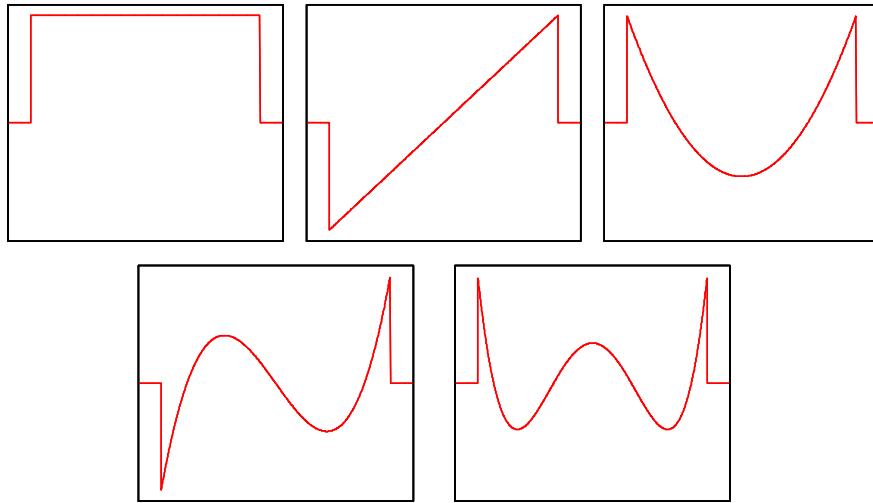


図 1 Legendre 多項式 ($k=4$)

基底を多重化することの利点は、ウェーブレットの階層構造を浅くできる点にある。1つのレベルあたりの表現力が高いので、深いレベルまで展開しなくても、誤差が閾値を下回るのである。一般に、3次元空間上に定義された関数の基底による展開では、1次元や2次元と比べて基底の数が膨大となるが、これにより、レベルの異なる関数をあまり増やさずに済む。前述の、同一レベル基底の積のみに絞る手法と合わせて、より効率的な表現が可能となる。

例として、図2に関数 $f(x) = \sin(\tan(6x))$ をウェーブレット分解した様子を示す。 \mathbf{V}_n^k は k 個の多項式で多重化した、レベル n のスケーリング基底表現、 \mathbf{W}_n^k は同様にレベル n のウェーブレット基底表現である。このようにツリー構造を取り、レベルが深くなるにつれて不要な区間が増えているのがわかる。

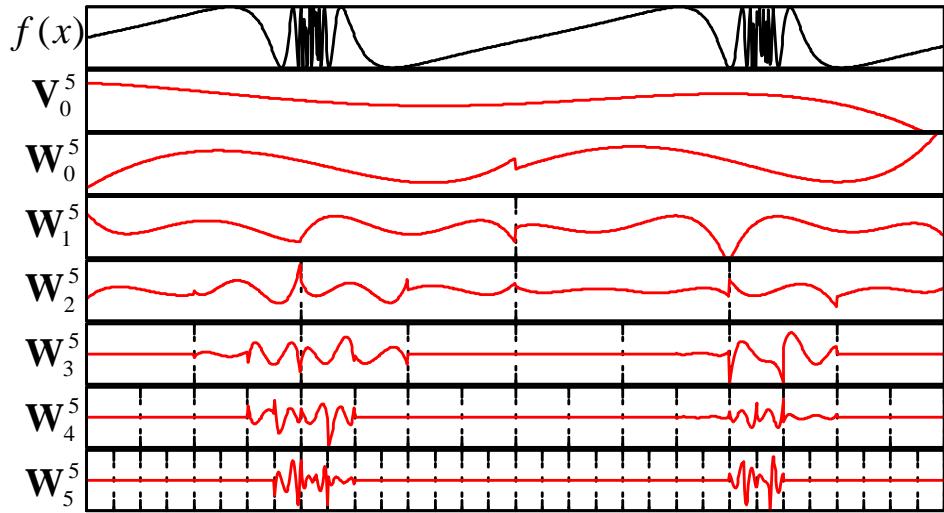


図 2 MRMW 基底による関数の表現 ($f(x) = \sin(\tan(6x))$)

図 3 に、ベンゼン上の分子軌道による例を示す。原子核中心では、カスプ領域を表現するための小さなボックスが見て取れる。また、結合領域では分子軌道がなめらかに変化するため、原子核中心よりは大きなボックスで分割されている。結合とは関係のない、ベンゼン環中心部や、外部の領域では、電子の存在確率がほぼゼロとなり、非常に大きなボックスによる近似のみで表現されている。

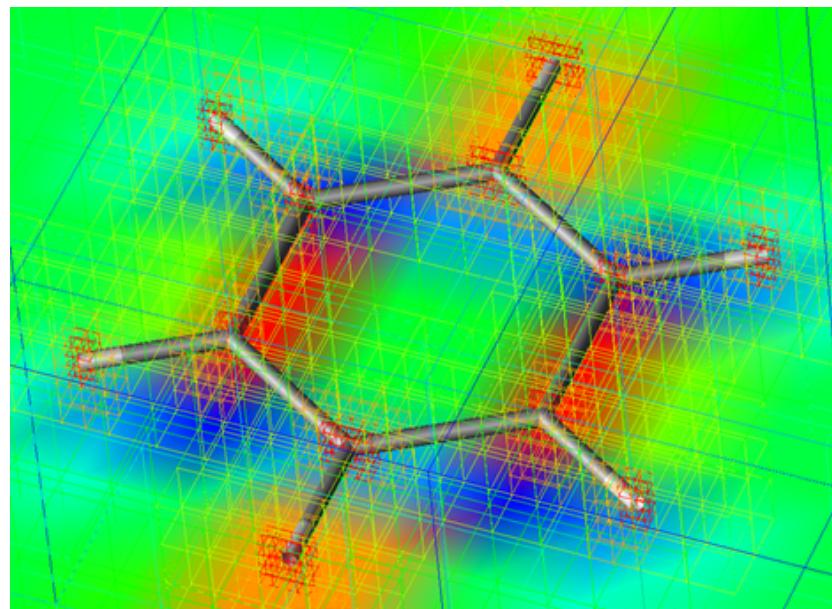


図 3 ベンゼン上の分子軌道の空間分割

MADNESS[4][5][6][7] ではこの MRMW 基底を利用している。要求されるエネルギー精度に応じて、 $k = 4, 6, 8$ 次までの Legendre 多項式で展開された基底を生成する。

また、並列動作を標準としており、各基底に対して *key* を付け、各計算ノードに分配する。関数に対する演算の際には、全ノードに対してタスクを発行し、各ノードはタスクに従って、自分が保持していない *key* のデータを別ノードに要求して、得られたデータで演算を行う。データの分配や、計算負荷はロードバランシング機構により、自動的に平坦化される。

2 分子分極率

物質は、正電荷を持つ原子核と負電荷を持つ電子とで構成されており、その構造によって電荷分布に偏りが生じ、これが電気双極子 \mathbf{p} となる。物質に外部から電場 \mathbf{E} を与えた場合、電荷分布が電場に従って変化するため、 \mathbf{p} も変化する。外場によって変化する \mathbf{p} は、次式で近似できる。

$$\mathbf{p} = \alpha \mathbf{E} \quad (2.1)$$

つまり、 \mathbf{E} に比例して \mathbf{p} が変化する。この変化率 α を分極率と呼ぶ。

物質は分子の集合であり、分極 \mathbf{p} も、分子分極 μ の集合として捉えることができる。分子自身が持つ構造により、常に電気双極子モーメントを持つ。これを永久双極子モーメント (permanent electric dipole) といい、 μ^0 で表す。永久双極子モーメントがゼロである分子は無極性分子、ゼロでない永久双極子モーメントを持つ分子は極性分子と呼ばれる。

分子の双極子モーメントは、外場 ϵ によって変化する。この時の変化量は、 α を拡張した 2 階のテンソルで表現できる。

$$\mu_i(\epsilon) = \mu_i^0 + \sum_{j=x,y,z} \alpha_{ij} \epsilon_j \quad (i = x, y, z) \quad (2.2)$$

この式は、低エネルギー領域では実測をよく反映しているが、高エネルギー領域ではそれが生じてくる。

そこで、 $E(\epsilon)$ を外場 ϵ の下での分子の全エネルギーとし、外場 ϵ についてテイラー展

開すると、次式が得られる。

$$E(\epsilon) = E_0 + \sum_{i=x,y,z} \mu_i^0 \epsilon_i + \sum_{i,j=x,y,z} \frac{1}{2!} \alpha_{ij} \epsilon_i \epsilon_j + \sum_{i,j,k=x,y,z} \frac{1}{3!} \beta_{ijk} \epsilon_i \epsilon_j \epsilon_k \\ + \sum_{i,j,k,l=x,y,z} \frac{1}{4!} \gamma_{ijkl} \epsilon_i \epsilon_j \epsilon_k \epsilon_l + \dots \quad (2.3)$$

ここで、 E_0 は外場 $\epsilon = 0$ の下での基底状態の分子の全エネルギー、 α_{ij} は分子分極率を示す 2 階のテンソル、 $\beta_{ijk}, \gamma_{ijkl}$ は分子超分極率を示す 3 階、4 階のテンソルである。3 次元であるので、それぞれ 27 要素、81 要素となるが、分子の持つ構造の対称性により、実際の自由度はもう少し下がる。例えば、z 軸上に直線状の分子では x 軸と y 軸は区別できない。

また、実際の観測では、アボガドロ数個集まった分子が、相互作用の結果、それぞれ自由な方向を向いているため、 α の各要素を実際に観測することは出来ない。そのため、 α の各要素を重み付けして平均化した量が観測される。

外場 ϵ の 2 乗以上の項は、弱い電場では全エネルギーに対する寄与が小さいため、 α による線形近似が十分成立つ。しかし、高出力レーザーを照射する場合のように、 ϵ が強くなると、これらの項が無視できなくなる。このような非線形項の寄与による光学現象を扱うのが非線形光学 (Nonlinear optics: NLO) である。

代表的な非線形光学現象には以下のようなものがある。

1. レーザーの波長変換
 - (i) 光高調波発生 (周波数の N 倍化)
 - (ii) 光混合 (周波数の加減算)
2. 光パラメトリック効果 (入射光の周波数を分解して異なる周波数を持つ複数の光を発生・增幅)
 - (i) 分光法
 - (ii) エンタングル光発生
3. 非線形屈折率変化
 - (i) 自己収束・自己位相変調 (光カーブ効果)
 - (ii) 光双安定性

レーザーの波長変換は、光通信でよく使われている他、レーザー冷却や原子核物理などの実験で有用である。光子のエネルギーは波長に依存するため、波長変換ができるということは、任意のエネルギーを与えられるということになるからである。

分光法には、試料の分析から宇宙物理学まで、幅広い応用がある。エンタングル光発生は、量子テレポーテーションに利用される。

高出力レーザーにおいては、レーザーを通すレンズ内で自己収束を起こしてレンズが破壊される場合があり、自己収束性の予測は重要である。光双安定性は光コンピューティングの開発に利用されている性質である。

このように、産業・先端問わず多くの応用があり、分子超分極率の機械計算による予測は、新たな光学デバイスの設計上、有用である。

3 分子軌道法における基底

Hartree-Fock 法では、非時間依存 Schrödinger 方程式

$$\mathcal{H} |\psi(\mathbf{x})\rangle = \mathcal{E} |\psi(\mathbf{x})\rangle \quad (3.1)$$

の分子軌道 ψ を Slater 行列式で展開し、平均場近似により、Hartree-Fock 方程式

$$\hat{f}_i \psi(\mathbf{x}) = \varepsilon_i \psi(\mathbf{x}) \quad (3.2)$$

を得て、自己無撞着場計算により、平均場近似下での系の電子状態を求める。

これを数値的に解くには、 $\psi(\mathbf{x})$ を何らかの基底で展開し、ベクトル表現する必要がある。

$$\psi(\mathbf{x}) = \sum_k^{N^{basis}} C_k^{MO} \phi_k(\mathbf{x}) \quad (3.3)$$

現在広く用いられている分子軌道法では、まず原子軌道 ϕ_k を原子核座標 \mathbf{R}_A を中心とする原子軌道 $\chi_k(r)$ で表現する。

$$\phi_k(\mathbf{x}) = Y_{lm}(\mathbf{x} - \mathbf{R}_A) \chi_k(|\mathbf{x} - \mathbf{R}_A|) \quad (3.4)$$

但し Y_{lm} は、方位量子数 l 、磁気量子数 m を持つ球面調和関数である。これを LCAO(Linear Combination of Atomic Orbitals) と呼ぶ。

原子軌道として、Gaussian 関数 $\exp(-\alpha x^2)$ の線形結合を用いる [8][9]。

$$\chi_k(r) = \sum_{\xi}^{N^{contract}} C_{\xi k}^{contract} \left(\frac{2\alpha_{\xi k}}{\pi} \right)^{\frac{3}{4}} \exp(-\alpha_{\xi k} r^2) \quad (3.5)$$

ここで Gaussian 関数 $\exp(-\alpha x^2)$ を primitive function と呼ぶ。

この Gaussian 関数による基底は、積分を高速に計算でき、また primitive function の数を増やすことにより、Slater 型関数 $\exp(-\zeta|x|)$ をよく近似できるという 2 つの性質を持つ^{*1}。基底状態の分子軌道計算や、分子の構造最適化に有利であったために、広く普及した。

一方で、Gaussian 型基底には問題もある。一つは、2 電子積分の計算量問題である。多電子系計算では 2 電子間相互作用の計算が必須となるが、Gaussian 型基底による 2 電子間相互作用の展開では、行列要素の計算量は基底関数の数の 4 乗に比例する。従って、系のサイズを大きくするほど、あるいは精度向上のために基底を大きくするほど、負荷が非常に大きくなってしまう。

この、2 電子積分の計算は、系の原子配置が確定して基底が生成された段階で 1 度だけ行えばよく、繰り返し計算の中でその都度計算する必要はない。しかし、数が膨大であるためにメモリ上にすべて格納しておくことはできず、何らかのファイルに書きだしておいて、その都度読み込む手法が取られる。ディスクへのアクセス速度はメモリに対して圧倒的に遅いため、計算時間の増大を招く。これを回避するため、繰り返し計算の中で、常にその都度計算する手法を取る場合もあるが、いずれにせよ、CPU 時間と記憶空間のどちらかで必ずボトルネックとなる。

もう一つは並列計算における問題である。系のサイズが大きくなるほど、Fock 行列が巨大化し、その固有値計算が高負荷となるのに対し、行列の固有値計算は並列化しづらい問題として知られている。

これらの問題に対する方策としては、分子を小領域に分けて並列計算させるものがあり、Fragment-MO(FMO) 法や、Divide and Conquer(DC) 法がよく知られている。これらの分割アルゴリズムには、分割方法の任意性や長距離相互作用の取り扱いなど、さまざまな問題がある。

最後に、より根源的な問題として、LCAO は、原子核配置とは異なる特徴を持つ関数を表現するには不向きであることが挙げられる。

一般論として、異なる軌道指数 α を持つ primitive function を基底に追加すれば、Hilbert 空間全体に占める「考慮している空間」の割合が上がる所以、精度が向上すると考えられる。しかし、どのような α を持たせればよいか決定するための、系統だった指針は存在しない。単に関数を増やすだけでは、基底の中で線形従属性が強くなるだけで、実質的な基底の大きさは増えない場合もある。更に、線形従属性を取り除き切れず、Fock 行列の対角化の際に問題が生じる場合もあり、overcompleteness の問題としてよく知られ

^{*1} 水素原子の Schrödinger 方程式における動径成分の解析解は、Slater 型になることが知られている。

ている。これらの問題は LCAO による精密な分子計算を行う上では事実上不可避である。波動関数をある基底の上で展開する場合、一般には、その基底で十分な精度が得られるという保証はない。分子軌道法の枠組みでは、Hartree-Fock から MPn や CCSDT のような post-Hartree-Fock への近似理論の拡張と、基底の完全性の高まりとが、直交した概念で捉えられる。もし、基底の完全性が、あるエネルギー精度の下で常に保証されていれば、常に Hartree-Fock limit に近い領域で、理論の拡張に注意を向けることができる。

4 Coupled Perturbed Hartree-Fock/Kohn-Sham

一般に、基底状態の系に対する何らかの外場による摂動が加えられた時、その response function を計算するのが Coupled Perturbed Hartree-Fock/Kohn-Sham (CPHF/KS) 法である。

Hartree-Fock は平均場近似で、Kohn-Sham は密度汎関数理論 (Density Functional Theory; DFT) による近似で、それぞれ用いられる Schrödinger 方程式の近似方程式であるが、ポテンシャルの構成法が異なるのを除いて、同じ形式でアルゴリズムを記述できるため、しばしば同一視され、実装上も同じプログラムで扱われる。

4.1 摂動論

CPHF/KS の前段階として、摂動論について触れる。摂動論は、正確に、あるいは近似的に解かれた系があり、そこに小さな摂動が加わった場合についての方法論である。

いま、ハミルトニアン \hat{H}_0 のもとで Schrödinger 方程式が解かれ、波動関数 $\{\Psi_i\}$ と固有値 $\{E_i\}$ が得られているとする。

$$\hat{H}_0\Psi_i = E_i\Psi_i \quad (i = 0, 1, \dots) \quad (4.1)$$

ここに、摂動項 \hat{H}' を加える。

$$(\hat{H}_0 + \lambda\hat{H}')\Psi = W\Psi \quad (4.2)$$

λ は摂動の強さを決めるパラメータである。

このとき、摂動が加わった Ψ および W を、次のように、摂動パラメータ λ に関するテイラー展開で表せる。

$$\Psi = \lambda^0 \Psi_0 + \lambda^1 \Psi_1 + \lambda^2 \Psi_2 + \lambda^3 \Psi_3 + \dots \quad (4.3)$$

$$W = \lambda^0 W_0 + \lambda^1 W_1 + \lambda^2 W_2 + \lambda^3 W_3 + \dots \quad (4.4)$$

ここで、 Ψ_1, Ψ_2, \dots および W_1, W_2, \dots は、それぞれ 1 次、2 次の波動関数、及びエネルギーに対する補正項である。

式 (4.3)(4.4) を式 (4.2) へ代入することで、エネルギーに対する補正項を得ることができる。

$$W_1 = \langle \Psi_0 | \hat{H}' | \Psi_0 \rangle \quad (4.5)$$

$$W_2 = \sum_{i \neq 0} \frac{\langle \Psi_0 | \hat{H}' | \Psi_i \rangle \langle \Psi_i | \hat{H}' | \Psi_0 \rangle}{E_0 - E_i} \quad (4.6)$$

1 次の補正項は、ハミルトニアンの摂動項に対する、無摂動波動関数による期待値であり、簡単に計算できる。

2 次の補正項は、すべての励起状態に対する総和を要求しており、簡単には計算できない。しかし物理的には次のような解釈を与える。すなわち、2 次の補正項の計算のためには、摂動によって基底状態にある電子が励起状態へと遷移し、エネルギーを放出しながらもとの基底状態へと戻る、一連の過程を、すべての励起状態について考える必要があることを示している。

従って、占有軌道の他に、非占有軌道についての情報を得なければ、正確な算定はできないのである。

4.2 LCAO 基底での CPHF/KS

ここでは LCAO 基底での、CPHF による分極率の算出法を示す。2 電子積分項を変更することで、容易に CPKS 法となる。簡単のため、ここでは閉殻制限軌道とする。

無摂動 Hartree-Fock-Roothaan 方程式は、次式で表せる。

$$\mathbf{FC} = \mathbf{SC}\epsilon \quad (4.7)$$

$$\mathbf{C}^\dagger \mathbf{SC} = \mathbf{1} \quad (4.8)$$

ここで、 \mathbf{F} は Fock 行列、 \mathbf{C} は分子軌道係数行列、 \mathbf{S} は基底の重なり行列、 ϵ は固有値行列である。式 (4.8) は規格直交条件である。

各行列を摂動パラメータ λ で展開し、1次の項についてまとめると、次式が得られる。

$$\mathbf{F}'\mathbf{C} + \mathbf{F}\mathbf{C}' = \mathbf{S}'\mathbf{C}\epsilon + \mathbf{S}\mathbf{C}'\epsilon + \mathbf{S}\mathbf{C}\epsilon' \quad (4.9)$$

$$\mathbf{C}^{\dagger'}\mathbf{S}\mathbf{C} + \mathbf{C}^{\dagger}\mathbf{S}'\mathbf{C} + \mathbf{C}^{\dagger}\mathbf{S}\mathbf{C}' = 0 \quad (4.10)$$

$\mathbf{F}', \mathbf{C}', \mathbf{S}', \epsilon'$ はそれぞれ $\mathbf{F}, \mathbf{C}, \mathbf{S}, \epsilon$ の1次の補正項である。

式(4.9)が1次のCoupled Perturbed Hartree-Fock方程式であり、 $\mathbf{F}, \mathbf{F}', \mathbf{C}, \mathbf{S}, \mathbf{S}', \epsilon, \epsilon'$ により、 \mathbf{C}' を求める。

1次のFock行列は次式で表せる。

$$\mathbf{F}' = \mathbf{H}' + \mathbf{G}'\mathbf{D} + \mathbf{G}\mathbf{D}' \quad (4.11)$$

但し、 \mathbf{G} は2電子積分を求める4階のテンソル量であり、基底によって定まる。

ここで、 $\mathbf{H}', \mathbf{G}', \mathbf{S}'$ はそれぞれ摂動を受けた1次の1電子行列、2電子積分のテンソル量、重なり行列の補正項であり、次式で表せる。

$$H'_{\mu\nu} = \langle \phi_{\mu} | H | \phi_{\nu} \rangle' = \frac{\partial}{\partial \lambda} \langle \phi_{\mu} | \mathbf{H} | \phi_{\nu} \rangle \quad (4.12)$$

$$G'_{\mu\nu\xi o} = \langle \phi_{\mu} \phi_{\nu} | G | \phi_{\xi} \phi_{o} \rangle' = \frac{\partial}{\partial \lambda} \langle \phi_{\mu} \phi_{\nu} | \mathbf{G} | \phi_{\xi} \phi_{o} \rangle \quad (4.13)$$

$$S'_{\mu\nu} = \langle \phi_{\mu} | \phi_{\nu} \rangle' = \frac{\partial}{\partial \lambda} \langle \phi_{\mu} | \phi_{\nu} \rangle \quad (4.14)$$

電子密度行列は分子軌道係数行列の積で表せる。

$$\mathbf{D} = \mathbf{C}^{\dagger} \mathbf{n} \mathbf{C} \quad (4.15)$$

$$\mathbf{D}' = \mathbf{C}^{\dagger'} \mathbf{n} \mathbf{C} + \mathbf{C}^{\dagger} \mathbf{n} \mathbf{C}' \quad (4.16)$$

基底が摂動に影響されないと仮定すると、次式が成り立つ。

$$\mathbf{S}' = \mathbf{S} \quad (4.17)$$

$$\mathbf{G}' = \mathbf{G} \quad (4.18)$$

これにより、式(4.9)を簡略化し、左から無摂動分子軌道係数をかけると、次式が得られる。

$$\mathbf{F}'\mathbf{C} + \mathbf{F}\mathbf{C}' = \mathbf{S}\mathbf{C}'\boldsymbol{\epsilon} + \mathbf{S}\mathbf{C}\boldsymbol{\epsilon}' \quad (4.19)$$

$$\mathbf{C}\mathbf{F}'\mathbf{C} + \mathbf{C}\mathbf{F}\mathbf{C}' = \mathbf{C}\mathbf{S}\mathbf{C}'\boldsymbol{\epsilon} + \mathbf{C}\mathbf{S}\mathbf{C}\boldsymbol{\epsilon}' \quad (4.20)$$

変分原理により、無摂動な系で最適な分子軌道は最小のエネルギーを与える。つまり、エネルギーの、分子軌道の変化に対する微分値はゼロとなる。このことは、Fock 行列の占有-非占有要素がゼロとなることと等価である。

$$\langle \psi_i | \mathbf{F} | \psi_a \rangle = 0 \quad (i : \text{occupied}, a : \text{unoccupied}) \quad (4.21)$$

いま、LCAO 基底を用いているため、基底関数系は原子配置が決定された段階で完全に特定されており、これが考えうる全空間となる。従って、摂動が加えられた場合の、ハミルトニアン全体に対して最適な軌道は、無摂動系における最適軌道のユニタリ変換によって与えられる。

$$\psi'_i = \sum_{j=1}^M U_{ji} \psi_j \quad (4.22)$$

このユニタリ変換行列 \mathbf{U} は、MO 係数の変化量を決定するものであるので、MO 係数の微分の情報を含んでいる。

ここで、 \mathbf{U} が 1 次の補正項のみを含んでいるとする。

$$|\psi_i\rangle = |\psi_i\rangle + \lambda \sum_{j=1}^M U'_{ji} |\psi_i\rangle + \dots \quad (4.23)$$

行列表現を用いると、次式が得られる。

$$\mathbf{C}' = \mathbf{U}'\mathbf{C} \quad (4.24)$$

\mathbf{U}' の行列要素は、Fock 行列の各項を 1 次まで展開することで得られる。

$$\langle \psi_\mu | \mathbf{H} | \psi_\nu \rangle \rightarrow \langle \psi_\mu | \mathbf{H} | \psi_\nu \rangle + \langle \psi_\mu | \mathbf{H} | \psi_\nu \rangle' \quad (4.25)$$

$$\langle \psi_\mu \psi_\nu | \mathbf{G} | \psi_\xi \psi_o \rangle \rightarrow \langle \psi_\mu \psi_\nu | \mathbf{G} | \psi_\xi \psi_o \rangle + \langle \psi_\mu \psi_\nu | \mathbf{G} | \psi_\xi \psi_o \rangle' \quad (4.26)$$

式(4.12)(4.13)は基底に対する表式であったのに対し、これは分子軌道に対する表式であるのに注意。

これらを式(4.21)に代入し、1次の項についてまとめると、次の行列方程式が得られる。

$$\mathbf{A}\mathbf{U}' = \mathbf{B} \quad (4.27)$$

ここで、 \mathbf{A} は無摂動の項のみを含み、 \mathbf{B} は1次の補正項のみを含む。

式(4.24)と次の定義式を式(4.20)に代入して整理すると式(4.29)を得る。

$$\mathbf{C}\mathbf{F}'\mathbf{C} = f' \quad (4.28)$$

$$f' = -\epsilon\mathbf{U}' + \mathbf{U}'\epsilon + \epsilon' \quad (4.29)$$

更に、行列の非対角項のうち、占有-非占有軌道成分のみが残ることから、次式が得られる。

$$\mathbf{U}' = \frac{f'}{\epsilon_i - \epsilon_a} \quad (4.30)$$

これにより、摂動を受けた1次の電子密度行列 \mathbf{D}' が求まり、それにより分極率 α を計算することができる。

このような手順で \mathbf{U}' を求めるのであるが、LCAO基底では、基底で表現できる関数空間がHilbert空間全体に比べて非常に小さく、原子中心に局在した関数に偏っているため、この \mathbf{U} の探索範囲も非常に限られている。

摂動を受けた1次の分子軌道は、基底状態の波動関数と直交するので、 \mathbf{U}' の探索範囲は、むしろ基底状態を除いた範囲となるのに対し、LCAO基底は基底状態をよく記述するように設計されている。

このようなミスマッチがあるため、分極率、あるいはより高次の超分極率の定量的算定は、難しいことが知られている。

4.3 任意完全基底でのCPHF/KS

式(2.2)は、静電場に対する式だったが、より一般的に角周波数 ω で振動する電場に対しては、次のように展開できる。

$$\mu_\xi(\epsilon_\zeta(\omega)) = \mu_\xi^0 + \alpha_{\xi\zeta}(\omega) \cdot \epsilon_\zeta(\omega) + \dots \quad (\xi, \zeta = x, y, z) \quad (4.31)$$

この時の α は次式で計算できる.

$$\alpha_{\xi\zeta}(\omega) = - \text{Tr} \langle \xi \cdot (\rho_\zeta(+\omega)) \rangle \quad (4.32)$$

$$\rho_\zeta(\omega) = \rho_\zeta^0 + \rho_\zeta(+\omega)\lambda \exp(+\omega) + \rho_\zeta(-\omega)\lambda \exp(-\omega) + \dots \quad (4.33)$$

ここで, ρ_ζ^0 は零次の密度演算子である. ζ, ξ は分極の方向を示す添字であるが, 今後は省略する.

1 次の密度演算子を計算するため, p 番目の電子に対応する response function を次のように定義する.

$$u_p^{(+)} = (1 - \rho^0) \rho(+\omega) \rho_p^0 \quad (4.34)$$

$$u_p^{(-)} = (1 - \rho^0) \rho(-\omega) \rho_p^0 \quad (4.35)$$

ここで, ρ_p^0 は p 番目の電子の, 零次の密度演算子に対する成分である.

$$\rho^0 = \sum_p^{N_{occ}} \rho_p^0 \quad (4.36)$$

これにより, 摂動をかけた密度演算子に対する運動方程式は, response function $u_p^{(+)}, u_p^{(-)}$ の連立方程式で表現できる.

$$\left(F^0 u_p^{(+)} - u_p^{(+)} \varepsilon_p^0 \right) + \left\{ \frac{d_p^\xi}{2} + \frac{\delta g}{\delta \rho} [\rho^0] * \left(\sum_i^{N_{occ}} u_i^{(+)} \phi_i^0 + \sum_i^{N_{occ}} \phi_i^0 u_i^{(-)} \right) \right\} = \omega u_p^{(+)} \quad (4.37)$$

$$\left(F^0 u_p^{(-)} - u_p^{(-)} \varepsilon_p^0 \right) + \left\{ \frac{d_p^\xi}{2} + \frac{\delta g}{\delta \rho} [\rho^0] * \left(\sum_i^{N_{occ}} u_i^{(-)} \phi_i^0 + \sum_i^{N_{occ}} \phi_i^0 u_i^{(+)} \right) \right\} = -\omega u_p^{(-)} \quad (4.38)$$

ここで, $N_{occ} F^0, \phi_p^0, \varepsilon_p^0, d_p^\xi$ はそれぞれ, 電子数, 零次の Hamiltonian, p 番目の電子に対応する軌道および軌道エネルギー, ξ 軸方向の双極子 (electric dipole) 演算子を ψ_p^0 に作用させたものである. $\frac{\delta g}{\delta \rho} [\rho]$ は, 平均場近似に基づく Hartree-Fock 方程式の場合は, 2 電子間反発項 (coulomb term) および交換相互作用項 (exchange term), 密度汎関数理論 (Density Functional Theory; DFT) に基づく Kohn-Sham 方程式の場合は, 2 電子間反発項および交換相関汎関数である. $A * B$ は A と B の畳み込み積分を示す.

p 番目の電子に対する表式であるので, 分子における計算では, $u_p^{(+)}, u_p^{(-)}$ はそれぞれ N_{occ} 個用意する必要がある.

式 (4.37), (4.38) は, 自由空間境界条件における 1 粒子グリーン関数を使って解ける.

$$u_p^{(+)} = G(\omega) * \left[V^0 u_p^{(+)} + \left\{ \frac{d_p^\xi}{2} + \frac{\delta g}{\delta \rho} [\rho^0] * \left(\sum_i^{N_{occ}} u_i^{(+)} \phi_i^0 + \sum_i^{N_{occ}} \phi_i^0 u_i^{(-)} \right) \right\} \right] \quad (4.39)$$

$$u_p^{(-)} = G(-\omega) * \left[V^0 u_p^{(-)} + \left\{ \frac{d_p^\xi}{2} + \frac{\delta g}{\delta \rho} [\rho^0] * \left(\sum_i^{N_{occ}} u_i^{(-)} \phi_i^0 + \sum_i^{N_{occ}} \phi_i^0 u_i^{(+)} \right) \right\} \right] \quad (4.40)$$

ここで, $G(\omega)$ はグリーン関数

$$G(\omega) = \left[\varepsilon_p^0 + \omega - \left(-\frac{1}{2} \nabla^2 \right) \right]^{-1} \quad (4.41)$$

であり, これは系に印加した周波数 ω の関数である. V^0 は零次のポテンシャル項である.

式 (4.39), (4.40) はそれぞれ等式であるが, これを漸化式とみなす. つまり, 与えられた $u_p^{(+)}, u_p^{(-)}$ で右辺を計算し結果を新たな $u_p^{(+)}, u_p^{(-)}$ とする. 両式のいずれも, 異なる電子に対応するものも含めて右辺に $u_p^{(+)}, u_p^{(-)}$ を必要とし, 従って $u_p^{(+)}, u_p^{(-)}$ は連立して計算される.

得られた $u_p^{(+)}, u_p^{(-)}$ により, 1 次の電子密度 $\rho_\zeta(\omega)$ が次のように計算できる.

$$\rho_\zeta(\omega) = \sum_p^{N_{occ}} \left(u_p^{(+)} \psi_p + \psi_p u_p^{(-)} \right) \quad (4.42)$$

これと式 (4.32) により, 分極率 $\alpha_{\xi\zeta}$ が求められる.

式 (4.39), (4.40) を繰り返し解くと, $u_p^{(+)}, u_p^{(-)}$ はゆっくりと収束する. 特に, 系の励起エネルギーと, 系に印加した電場の周波数が近づく共鳴領域 (resonance region) では, 収束が遅くなり非効率である.

Krylov 部分空間を利用した擬ニュートン法 [10] を利用すれば, 効率は改善する. このアルゴリズムは基底状態の自己無撞着計算においても有用である.

4.3.1 分極率と励起状態の関係

外場の周波数 ω に対する $\alpha_{\xi\zeta}(\omega)$ は, 次のようにも書ける.

$$\alpha_{\xi\zeta}(\omega) = \frac{2}{\hbar} \sum_{n \neq 0} \frac{\omega_{n0} \langle \psi_0 | \mu_\xi | \psi_n \rangle \langle \psi_n | \mu_\zeta | \psi_0 \rangle}{\omega_{n0}^2 - \omega^2} \quad (4.43)$$

$$\boldsymbol{\mu} = \sum_i q_i \mathbf{r}_i \quad (4.44)$$

ここで, ψ_0 は基底状態の波動関数, ψ_n は n 番目の励起状態の波動関数, ω_{n0} は対応する励起エネルギー, μ は電気双極子演算子である.

ω が励起エネルギーに近づくにつれて, 分母がゼロに近づくため, α_{xy} は発散していく.これを共鳴周波数という.

共鳴領域近傍では, 値が急激に大きくなるため, 正確な算定は難しい. 貧弱な基底では, 励起エネルギーの見積り自体に正確性を欠く場合もある.

4.3.2 H₂ における分極率 α

z 軸を長軸とした水素分子 H₂ の分極率 α について, Gaussian 基底と MRMW 基底で算定を行った [11]. 核間距離は 1.4[a.u.] とし, 2 電子相互作用は Hartree-Fock を使った.

z 軸上の直線分子であるので, x 軸と y 軸の区別はなくなる.

$$\alpha_{xx} = \alpha_{yy} \quad (4.45)$$

また, 対称分子なので, cross component の要素もゼロとなる.

$$\alpha_{xy} = \alpha_{yx} = \alpha_{yz} = \alpha_{zy} = \alpha_{zx} = \alpha_{xz} = 0 \quad (4.46)$$

従って, α_{xx}, α_{zz} の 2 成分のみが問題となる.

Gaussian 基底としては, cc-pvtz から d-aug-cc-pv5z までの基底を対象とした. この基底関数系のシリーズは, 最後の 2 文字が分極関数の個数を示している.

- cc-pvdz: Double-zeta
- cc-pvtz: Triple-zeta
- cc-pvqz: Quadruple-zeta
- cc-pv5z: Quintuple-zeta

また, 接頭辞 aug-(=augmented) は基底に diffuse 型の Gaussian 型関数を追加していることを示している. d-aug-(=double augmented) は, aug-に更に diffuse 型関数を追加したものである.

これらの Gaussian 基底のうち, 最大のものは d-aug-cc-pv5z であるが, 基底状態の計算では MRMW による全エネルギーが, d-aug-cc-pv5z を含めてすべての Gaussian 基底による全エネルギーを下回る. 変分原理により, MRMW は, これらすべての Gaussian 基底に比べて「より良い」波動関数を得ることができているといえる.

前述のように、分極率計算は励起状態に関係がある。LCAO は基底状態の軌道を元に設計されており、励起状態の記述には必ずしも適切ではない。大きな軌道指数 α を持つ primitive function (diffused function) を追加することで、原子核から離れた位置での表現力を上げることは、分極率計算ではよく行われていることである。

図 4 は、外場の周波数と α_{xx} の関係を示したものである。グラフ横軸が外場周波数、縦軸が分極率 α の xx 要素、すなわち分子の長軸に対して垂直な要素である。

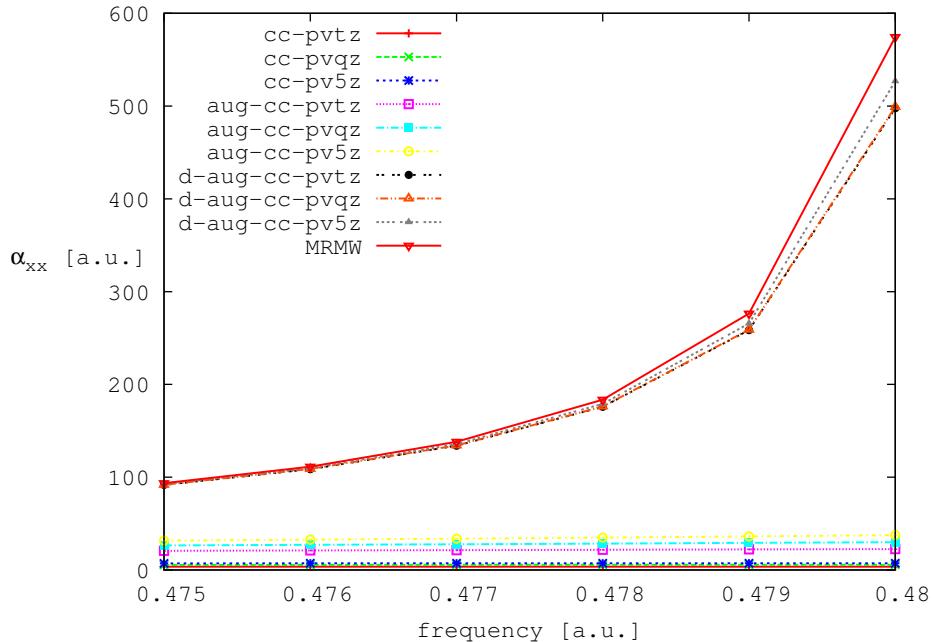


図 4 水素分子の周波数依存分極率（垂直要素）

グラフを見ると、d-aug-ccpv シリーズと MRMW 基底が共鳴周波数に向かって急激に上昇しているのに対し、それ以外の Gaussian 基底ではそのような傾向は見られない。共鳴領域近傍においては、d-aug-ccpv までの diffuse 型関数が定性的記述に必須であることがわかる。

図 5 は、図 4 に関して、MRMW の値に対する各 Gaussian 基底の値の差分を示したものである。グラフ横軸が外場周波数、縦軸が分極率 α_{xx} の差分である。d-aug-cc-pv5z が最も MRMW に近い結果を出しておらず、次に d-aug-cc-pvqz, d-aug-cc-pvtz と続いている。分極関数の追加が、精度の向上に役だっていることがわかる。

次に α_{zz} について見ていく。図 6 がそれである。cc-pvnz についてはここでは省略した。

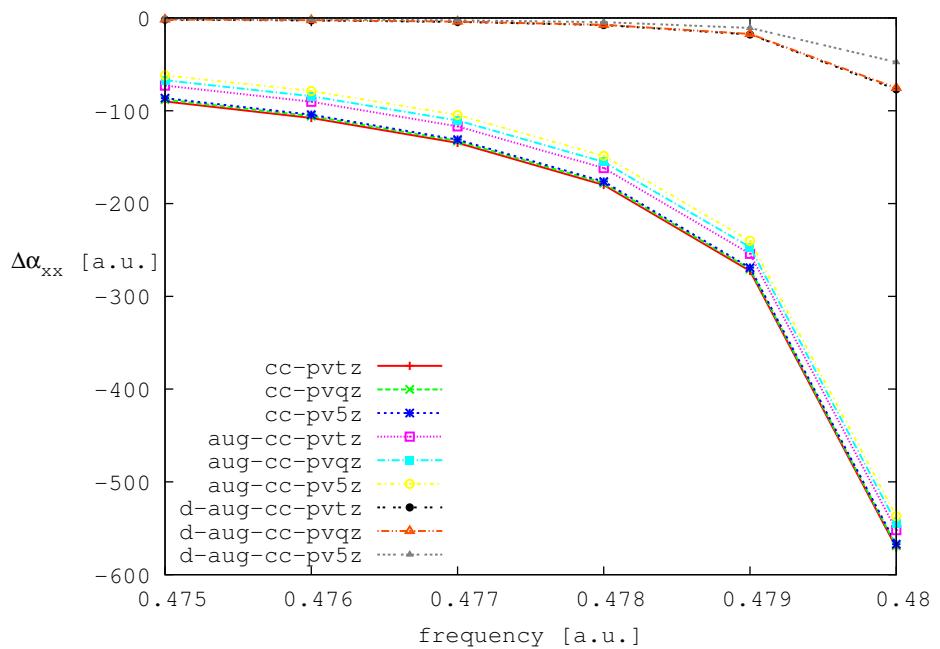


図 5 水素分子の周波数依存分極率（垂直要素）の MRMW に対する差分

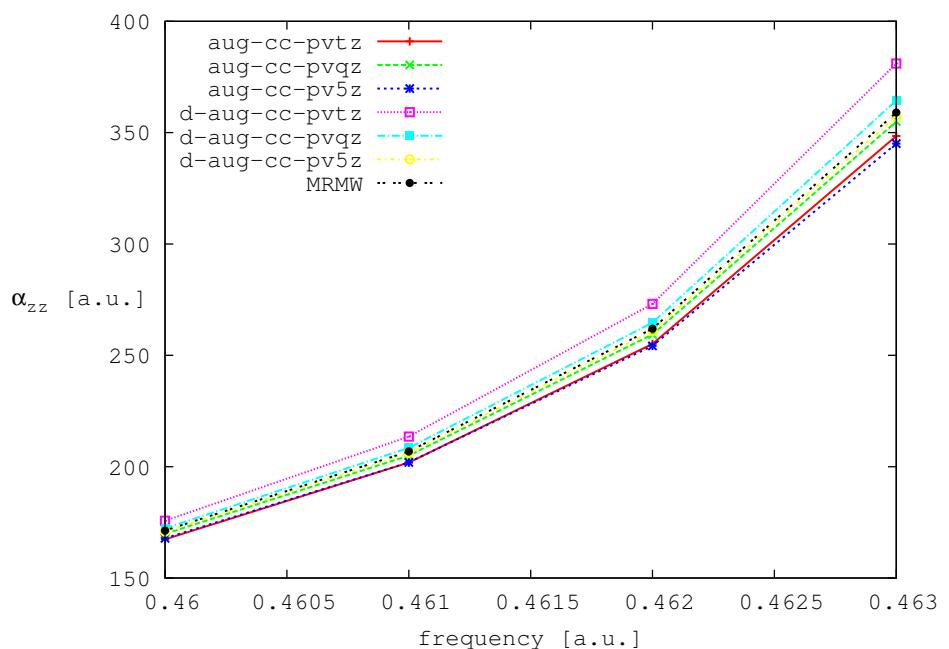


図 6 水素分子の周波数依存分極率（平行要素）

α_{xx} とは異なり、こちらは aug-ccpv についても定性的な傾向が再現できている。また、 α_{xx} とは異なる周波数で共鳴に向かっている。

図 7 は、図 5 と同様、MRMW の値に対する各 Gaussian 基底の値の差分を示したものである。

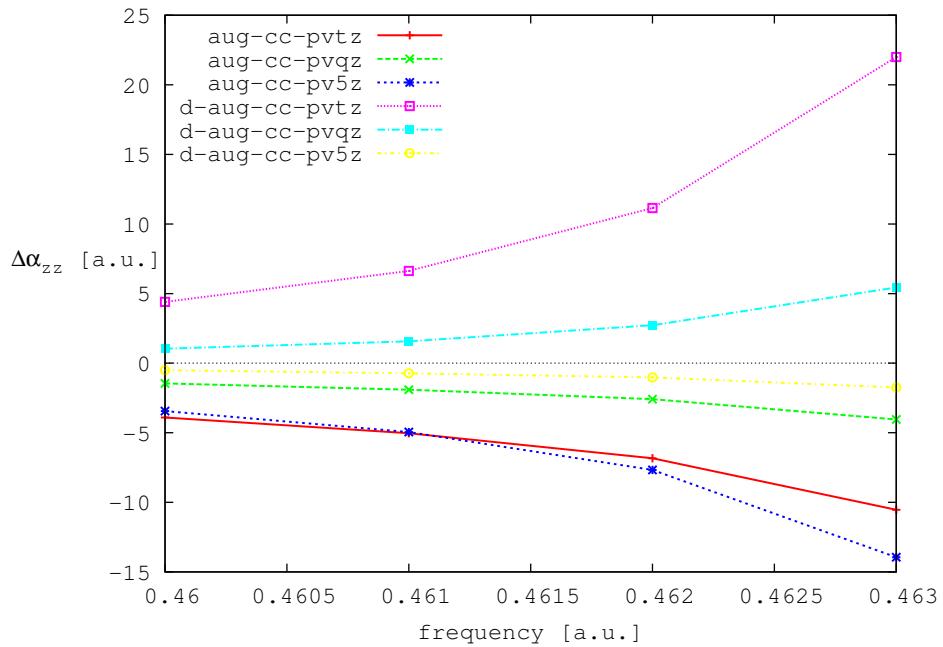


図 7 水素分子の周波数依存分極率（平行要素）の MRMW に対する差分

α_{xx} の場合とは異なる特徴が見て取れる。まず、aug-ccpv シリーズについて見ると、分極関数の追加が単調な精度の向上に寄与しているとはいえない。周波数によって順序が逆転している箇所もある。

d-aug-ccpv シリーズでは、順序こそ分極関数の追加順になっているものの、その極限は MRMW とは異なる位置にある。

このように、平行要素については、規則性を見出すことが難しい。

5 擬ポテンシャル法

巨大系の分子計算や、平面波基底を用いた結晶系の計算などで、計算負荷の低減を目的に用いられてきたのが擬ポテンシャル法である。原子核近傍に局在する内殻電子を、ある種の不变なポテンシャルで置換えることにより、次のような特徴が得られる。

- 僮電子軌道数の削減による計算の高速化
 - 2電子積分
 - Fock 行列の対角化
- 相対論効果の取り込み

金 (Au) のような重金属元素においては、内殻電子は原子核が与える非常に深いクーロンポテンシャルの影響を強く受け、非常に高速に運動する。この速度が光速に近づくことにより、相対論効果が無視できなくなる。一方で、僕電子は内殻に局在した軌道と直交するため、比較的原子核から離れた位置での存在確率が高くなり、それほど高速には運動しない（速度の期待値が低い）ので、軽原子の僕電子と同様に扱うことができる。このような元素を含む系の全電子計算を行う場合には、Dirac-Hartree-Fock 方程式のような、ローレンツ不変性を満たす方程式を使った方法で解く必要がある。しかし内殻電子を擬ポテンシャルに置き換えることにより、このような重元素であっても、Hartree-Fock や Kohn-Sham の枠組みで計算することが可能となる。特に伝導計算では電極として金属元素を多用するため、必須の機能である。

擬ポテンシャル法としては様々なものが提案されており、大きく 3 つの類型に分けることができる。

1. Effective Core Potential (ECP)
2. Pseudo-potential (PP)
3. Model Core Potential (MCP)

これらはおおむね共通して、専用の基底を持つ。パラメータは、軌道エネルギーや軌道形状のような、適当な評価値に基づいて最小二乗法により求められているが、最終的には、研究者の直感により、基底とポテンシャルのバランスを取るように調整される。Gaussian 型基底の設計と同様か、あるいはそれ以上に、ある種の職人芸的な世界であり、理論的な背景だけで説明できるものではない。

また、擬ポテンシャルは一般に局在項 (local part) と非局在項 (non-local part) の両方を与える。局在項は、内殻部に局在する電子が及ぼす古典的クーロンポテンシャルを表現すると考えられる。特に、 $r \gg 1$ の極限では点電荷として近似される。非局在項は、反対称性原理による内殻軌道から僕電子軌道への相互作用、すなわち交換相互作用を表現する。

一般的には、一原子・全電子系における Hamiltonian

$$\hat{H} = -\frac{1}{2}\nabla^2 - \frac{Z}{r} + \sum_{i,j}^{N_{occ}} \frac{1}{r_{ij}} \quad (5.1)$$

を、次のように置き換える。

$$\hat{H}' = -\frac{1}{2}\nabla^2 - \frac{Z}{r} + V^{loc}(r) + \sum_{i,j}^{N_v} \frac{1}{r_{ij}} + \sum_i V^{nloc}(\psi_i) \quad (5.2)$$

但し、 Z は原子番号、 r は原子核中心からの距離、 r_{ij} は電子 i-j 間の距離、 N_{occ}, N_v は占有電子数と価電子数、 $V^{loc}(r)$ 、 $V^{nloc}(\psi_i)$ は擬ポテンシャルの局在項と非局在項である。

以下、3種のポテンシャルについて順に特徴を述べ、最後に MCP について詳述する。

5.1 Effective Core Potential

Effective Core Potential (ECP)[12] は、分子計算において最も普及した形式の擬ポテンシャルである。LANL2DZ や Stuttgart, Hay-Wadt, CRENBL などがあり、多くの分子計算パッケージに取り入れられている。非相対論的に取り扱う場合の一般形は、次のように定式化される。

$$V^{loc}(r) = \frac{1}{2l+1} [lU_{l-1/2}(r) + (l+1)U_{l+1/2}(r)] \quad (5.3)$$

$$V^{nloc}(r) = s \cdot \sum_{l=1}^{L-1} \frac{2}{2l+1} [U_{l-1/2}(r) + U_{l+1/2}(r)] \sum_{mm'} \langle lm | \hat{l} | lm \rangle \langle lm | \quad (5.4)$$

$$U_l(r) = \sum_k A_{lk} r^{n_{lk}-2} e^{-B_{lk}r^2} \quad (5.5)$$

ここで、 L はその原子が持つ最大の軌道角運動量より 1 大きな値、 l は軌道角運動量量子数、 \hat{l} は軌道角運動量演算子、 s はスピン演算子である。 $A_{lk}, B_{lk}, n_{lk} = 0, 1, 2$ はパラメータである。

ECP による解として得られる軌道は “pseudo-orbital” と呼ばれる。ECP では、原子ごとに半径を設定し、その半径以遠を実際の価電子軌道にフィッティングし、内側は多項式でなめらかに繋ぐので、全電子計算による解とは nodal structure が異なるためである。この半径は、通常、結合距離よりも短く取る。この特徴により、分子の構造最適化によく用いられる。また、価電子軌道の内殻領域がなだらかなので、基底が必要とする primitive function の数を削減できる。

非局在項に含まれる、射影演算子 $|lm\rangle\langle lm|$ は、LCAO 基底における各原子軌道の持つ l, m に応じて、当該原子軌道に相当する成分を、分子軌道から選択的に取り出す、という意味合いになる。

例を挙げよう。今、ある分子軌道 $\psi(r)$ が次のように Slater 型 LCAO 基底の上で表現されているとする。

$$\begin{aligned}\psi(\mathbf{r}) = & C_s Y_{0,0} N_{\zeta_s} \exp(-\zeta_s |r|) \\ & + C_{-1} Y_{1,-1} N_{\zeta_p} \exp(-\zeta_p |r|) \\ & + C_0 Y_{1,0} N_{\zeta_p} \exp(-\zeta_p |r|) \\ & + C_1 Y_{1,1} N_{\zeta_p} \exp(-\zeta_p |r|)\end{aligned}\quad (5.6)$$

ここで、 Y_{lm} は軌道角運動量量子数 l 、磁気量子数 m を持つ球面調和関数、 ζ_s, ζ_p は s 型、p 型の軌道指数、 N_{ζ_l} は ζ_l に対応する規格化定数、 C_s, C_{-1}, C_0, C_1 は分子軌道係数である。ここに左から射影演算子を作用させると、

$$|0,0\rangle\langle 0,0| \psi = C_s Y_{0,0} N_{\zeta_s} \exp(-\zeta_s |r|) \quad (5.7)$$

$$|1,-1\rangle\langle 1,-1| \psi = C_{-1} Y_{1,-1} N_{\zeta_p} \exp(-\zeta_p |r|) \quad (5.8)$$

$$|1,0\rangle\langle 1,0| \psi = C_0 Y_{1,0} N_{\zeta_p} \exp(-\zeta_p |r|) \quad (5.9)$$

$$|1,1\rangle\langle 1,1| \psi = C_1 Y_{1,1} N_{\zeta_p} \exp(-\zeta_p |r|) \quad (5.10)$$

となる。

このように LCAO 基底では、射影演算子の作用は項の選択的抽出で済む。

一方、3 次元デカルト座標上に定義される基底で展開された波動関数について、このような射影を行うには、LCAO 基底が規格直交であり、かつ LCAO へのユニタリ変換が既知である必要があるが、本質的に LCAO は非直交基底であり、互いに重なりを持つ。

特定の軌道指数集合 $\{\zeta_l\}$ を仮定して近似的に展開することはできるが、その選択は常に恣意的であり、妥当性は保証されない。LCAO は非直交基底であるのでノルムも保存されない。更に、その際に失われる情報は膨大であり、ポテンシャルの妥当性そのものが疑わしくなる。

従って、非 LCAO の基底では、このような形式の射影演算子を持つ擬ポテンシャルを採用する意味が無いといえる。

5.2 Pseudo Potential

MCP と ECP が分子計算で用いられてきたのに対し、Pseudo Potential (PP) は、結晶などの連続系を平面波 (plane wave) 基底で解く際に用いられてきた擬ポテンシャルで

ある。以下の形式を持つものが知られている [13]。

$$-\frac{Z}{r} + V^{loc}(r) = -\frac{Z_{ion}}{r} \operatorname{erf}\left(\frac{r}{\sqrt{2}r_{loc}}\right) + \exp\left[-\frac{1}{2}\left(\frac{r}{r_{loc}}\right)^2\right] \sum_i C_i \left(\frac{r}{r_{loc}}\right)^{2i} \quad (5.11)$$

$$V^{nloc}(r) = \sum_l^3 \sum_{i=1}^3 \sum_{j=1}^3 \sum_m |Y_{lm} p_i^l\rangle h_{i,j}^l \langle Y_{lm} p_j^l| \quad (5.12)$$

$$p_i^l(r) = N_{l,i} \left(\frac{r}{r_l}\right)^{l+2(i-1)} \exp\left[-\frac{1}{2}\left(\frac{r}{r_l}\right)^2\right] \quad (5.13)$$

$$\int p_i^l(r) p_i^l(r) r^2 dr = 1 \quad (5.14)$$

ここで、 Z_{ion} は価電子を除いたときの核・内殻電子系が持つ総電荷、 r_{loc}, r_l はフィッティング境界となる半径パラメータ、 $N_{l,i}$ は規格化定数である。式 (5.14) の通り、 $p_i^l(r)$ は規格化されている。

この擬ポテンシャルは、ECP と同様、pseudo-orbital を形成する。正しい nodal structure を持つ価電子軌道は、局在してかつ振動するので、平面波基底で表現する場合には非常に高い波数を持った基底を多数追加する必要がある。pseudo-orbital の持つ、内殻領域がなだらかという特徴は、平面波基底で解く場合、基底関数を削減する効果が高いため、都合が良い。

$Y_{lm} p_i^l$ は、 r, θ, ϕ を引数に取る 3 次元関数となる。ECP の射影演算子と異なるのは、動径成分を含んでいる点である。この場合、3 次元デカルト座標上に定義される基底で一度展開すれば、単純な内積と定数倍操作により、射影演算子を作用させた軌道を計算することができます。

従って、非 LCAO の基底でも、このような形式の射影演算子を持つ擬ポテンシャルを実装できる可能性はある。

5.3 Model Core Potential

Model Core Potential (MCP)[16] は、分子計算で用いられる、価電子軌道の nodal structure を保存する擬ポテンシャルである。GAMESS[17] に実装例がある。

一般形は次のとおりである。

$$V^{loc}(r) = \sum_k A_k r^{n_k-2} \exp(-\alpha_k r^2) \quad (5.15)$$

$$V^{nloc}(r) = \sum_c -f \epsilon_c |\psi_c\rangle \langle \psi_c| \quad (5.16)$$

ここで, N_c は内殻電子数, ψ_c, ϵ_c は内殻軌道と対応する軌道エネルギー, $f = 2.0, A_k, n_k, \alpha_k$ はパラメータである.

非局所項に内殻軌道そのものを含んでいるのが特徴的である. この射影演算子により, 内殻軌道の成分に対応する固有値をゼロより上まで飛ばし, 最低固有値を持つ軌道が node を持った価電子軌道となり, 全電子の場合と全く同じ価電子軌道を生成できる. この射影演算子も, 3次元空間上に表現可能な関数で構成されているため, 前述の理由で MRMW との親和性が高い.

図 8 は, Si 原子に関して, MCP の射影演算子によりエネルギー構造がどのように変化するのか示したものである.

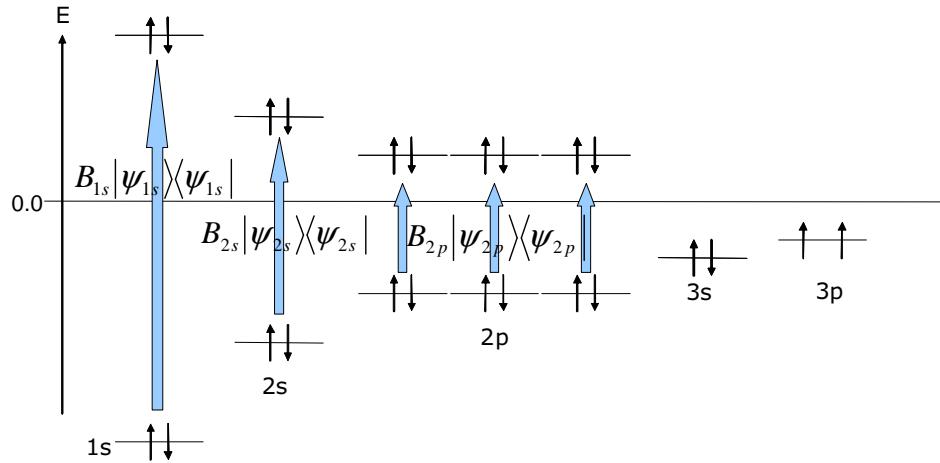


図 8 Si 原子のシフト

Si 原子では $2p$ 軌道までを内殻とみなす. 各軌道に対応する内殻軌道 $|\psi_c\rangle$ と, $B_c = -f \epsilon_c$ により, 固有値を内殻軌道の持つ固有値を上に押し上げる. これにより, 最低固有値が $3s$ 軌道となる.

5.3.1 MCP の基底依存性

MCP は、専用の基底として、MINI-4 をベースに、primitive function の削減や、複数の基底を結合して固定するなどの調整を施した専用のものを用意している。これ以外の基底では、定性的に誤った結果をもたらす場合すらある。

GAMESS を用い、1 原子開殻非制限軌道 Hartree-Fock 計算を行った場合の軌道エネルギーを表 1 に示す。

表 1 GAMESS による Na 原子開殻非制限軌道 Hartree-Fock

method	[a.u.]			[kcal/mol]		
	ϵ_{2p}^α	ϵ_{2p}^β	ϵ_{3s}^α	$\Delta\epsilon_{2p}^\alpha$	$\Delta\epsilon_{2p}^\beta$	$\Delta\epsilon_{3s}^\alpha$
All Electrons aug-cc-pvqz	-1.5191	-1.5171	-0.1822	-	-	-
MCP mcp-tzp	-1.5308	-1.5294	-0.1830	7.3419	7.7184	0.5020
MCP 6-311++g**	-1.5206	-1.5199	-0.1832	0.94126	1.7570	0.62751
MCP 6-311++g**(prim.)	-0.9833	-0.8206	-1.3323	-336.22	-437.06	721.70
MCP aug-cc-pvqz	-1.5146	-1.5140	-0.1831	-2.8238	-1.9453	0.56476
MCP aug-cc-pvqz(prim.)	-0.9862	-0.8255	-1.3357	-334.40	-433.99	723.83

ここで、“All Electrons” は全電子，“MCP” は擬ポテンシャルとして MCP を指定した計算である。mcp-tzp は、MCP ポテンシャルを使用するためにつくられた基底のうち、安定的に解けて、かつ最も大きなもの、6-311++g**(prim.) 及び aug-cc-pvqz(prim.) は、6-311++g**, aug-cc-pvqz の基底の線形結合を分割し、すべての基底が 1 つの primitive function で表現されるように加工した基底である。この操作により、基底の自由度が上るので、同じ primitive function 集合から構成されていても、より完全に近い基底関数系となる。 $\epsilon_{2p}^\alpha, \epsilon_{2p}^\beta, \epsilon_{3s}^\alpha$ は、まず軌道から低エネルギーの 1s/2s 軌道を除い

た上で、残ったもののうち p 型基底の成分でつくられているものを 2p, s 型基底の成分でつくられているものを 3s としている。スピンについては、電子の多い方を α とした。 $\Delta\epsilon_{2p}^\alpha, \Delta\epsilon_{2p}^\beta, \Delta\epsilon_{3s}^\alpha$ は、対応する軌道エネルギーの全電子解との誤差を示した。

表を見て明らかなように、完全に近い基底での MCP の使用は、2p 軌道と 3s 軌道のエネルギーの逆転という、定性的にありえない結果をもたらす。

MCP の射影演算子は MCP の想定する内殻軌道で構成されている。`6-311++g**` と `aug-cc-pvqz` は、いずれも内殻軌道を固定して価電子部分の自由度を重視する基底である。MCP の想定する内殻軌道と、これら基底の持つ内殻軌道には差異があるが、内殻部における基底の自由度が無いため、基底を無加工で使う限りでは、価電子軌道が内殻に落ち込むことなく計算できる。

しかし、MCP の内殻軌道表現は貧弱であるため、基底の内殻部を分解して自由度を高めてしまうと、その基底が表現する空間において、価電子と併せて自己無撞着となる内殻軌道と MCP の内殻軌道との差が大きくなり、固有値を引き上げ切れなくなる。その結果、3s にならなければならぬ軌道が 1s ないし 2s の成分を含み、2p との間でエネルギー準位の逆転を生ずる。

MRMW 基底は近似的に完全基底であるため、これと同様な現象が起こる。従って、MRMW 基底で MCP を利用するには、MRMW 基底と consistent なポテンシャルパラメータおよび内殻軌道の作成が不可欠である。

5.3.2 MRMW へのパラメータフィッティング

■フィッティング方法の検討 オリジナルの MCP パラメータは、次の “pivotal quantity” Δ を最小化するようにフィッティングされている。

$$\Delta = \sum_i^{N_v} w_i \left| \epsilon_i^{ref} - \epsilon_i^{mcp} \right|^2 + \sum_i^{N_v} \sum_k^K W_k \left[r_k R_i^{ref}(r_k) - r_k R_i^{mcp}(r_k) \right]^2 \quad (5.17)$$

但し、 N_v は価電子数、 $R_i^{ref}(r), \epsilon_i^{ref}$ は全電子計算の解となる i 番目の価電子軌道の動径成分と軌道エネルギー、 $R_i^{mcp}(r), \epsilon_i^{mcp}$ は MCP を用いて計算した i 番目の価電子軌道の動径成分と軌道エネルギー、 K はサンプリング点数、 w_i, W_k は適当な重みである。

この方法では Schrödinger 方程式を 1 次元に還元し、1 回ごとに解きながら、パラメータをフィッティングしていく。

MRMW の場合、1 ステップごとに SCF を解く必要があるが、これには時間がかかり、また、SCF が収束しなかった場合には、最適化プロセスが停止してしまう。そこで、新た

なフィッティング手法を考案した.

まず, MCP の定義について考える. 式 (5.1) と式 (5.2) との比較により, 自己無撞着となり, 全電子解が得られた場合, 次が成り立つ.

$$V^{loc}(r) + \sum_i V^{nloc}(\psi_i) = \sum_{i,j}^{N_c} \frac{1}{r_{ij}} \quad (5.18)$$

左辺を MCP で, 右辺を Fock 演算子で書きなおせば,

$$\sum_k A_k r^{n_k-2} \exp(-\alpha_k r^2) + \sum_c -f\epsilon_c |\psi_c\rangle \langle \psi_c| = \sum_c \hat{J}_c - \hat{K}_c \quad (5.19)$$

となる. ここでは開殻表示だが, 閉殻 ($\hat{J}_c - \hat{K}_c \rightarrow 2\hat{J}_c - \hat{K}_c$) でも同様である.

これが成り立つように A_k, n_k, α_k をフィッティングすればよい. つまり, 一度全電子解が得られれば, Schrödinger 方程式を解く必要は無い.

次に “pivotal quantity” Δ の定義を変更する. 式 (5.19) より, 次のように定義できる.

$$\Delta' = \sum_i^{N_v} \int [r U^{mcp} \psi_i(\mathbf{r}) - r U^{ex} \psi_i(\mathbf{r})]^2 d\mathbf{r} \quad (5.20)$$

$$U^{ex} = \sum_c^{N_c} \hat{J}_c - \hat{K}_c \quad (5.21)$$

$$U^{mcp} = \sum_k A_k r^{n_k-2} \exp(-\alpha_k r^2) + \sum_c -f\epsilon_c |\psi_c\rangle \langle \psi_c| \quad (5.22)$$

基本的には通常の最小 2 乗法である. 交換演算子 \hat{K}_c があるため, 全電子解の価電子軌道 ψ_i を含める必要がある. また, 重み関数として r を導入した. これにより, 内殻部に比べ, 結合付近の精度が向上する. 以降の議論は, 他のパラメータに依存せず, かつ十分なめらかな任意の関数を重み関数として選んでも成立する.

前述のとおり, 局所項の長距離成分は $\frac{N_c}{r}$ で近似される. そこで, この項は予め各項から引いておく. つまり, $A_0 = N_c, n_0 = 1, \alpha_0 = 0$ とし,

$$U^{ex} = -\frac{N_c}{r} + \sum_c^{N_c} \hat{J}_c - \hat{K}_c \quad (5.23)$$

とする.

ここで, $\{A_k, n_k, \alpha_k\}$ に対する最小 2 乗フィッティングが可能になる.

■線形フィッティング $\{n_k, \alpha_k\}$ を固定した時, 式 (5.20) を最小化する $\{A_k\}$ は線形に解ける. 以下にそれを示す.

まず,

$$\chi_k(r) = r^{n_k-2} \exp(-\alpha_k r^2) \quad (5.24)$$

$$\hat{\Omega} = \sum_c^{N_c} -f \epsilon_c |\psi_c\rangle \langle \psi_c| \quad (5.25)$$

として, Δ' を書き直し, A_m で微分する.

$$\Delta' = \sum_i^{N_v} \int \left[r \left(\sum_k A_k \chi_k(r) + \hat{\Omega} \right) \psi_i(\mathbf{r}) - r U^{ex} \psi_i(\mathbf{r}) \right]^2 d\mathbf{r} \quad (5.26)$$

$$\frac{d\Delta'}{dA_m} = 2 \sum_i^{N_v} \int \left[r \left(\sum_k A_k \chi_k(r) + \hat{\Omega} \right) \psi_i(\mathbf{r}) - r U^{ex} \psi_i(\mathbf{r}) \right] [r \chi_m(r) \psi_i(\mathbf{r})] d\mathbf{r} \quad (5.27)$$

ところで

$$\sum_i^{N_v} \left[r \left(\sum_k A_k \chi_k(r) + \hat{\Omega} \right) \psi_i(\mathbf{r}) - r U^{ex} \psi_i(\mathbf{r}) \right] = 0 \quad (5.28)$$

であれば, $\frac{d\Delta'}{dA_m} = 0$ を満たすので, この A_m は Δ' を極小化する.

式 (5.28) を (条件を強めて^{*2}) 整理すると,

$$\sum_i^{N_v} \sum_k A_k r \chi_k(r) \psi_i(\mathbf{r}) = \sum_i^{N_v} r \left[U^{ex} - \hat{\Omega} \right] \psi_i(\mathbf{r}) \quad (5.29)$$

となる.

両辺に $[r \chi_l(r) \psi_i(\mathbf{r})]^*$ をかけて積分する.

$$\sum_i^{N_v} \sum_k \int [r \chi_l(r) \psi_i(\mathbf{r})]^* A_k r \chi_k(r) \psi_i(\mathbf{r}) dr = \sum_i^{N_v} \int [r \chi_l(r) \psi_i(\mathbf{r})]^* r \left[U^{ex} - \hat{\Omega} \right] \psi_i(\mathbf{r}) dr \quad (5.30)$$

次のように置くと, これは $\{A_k\}$ に関する線形方程式となる.

^{*2} この移項は, 総和がゼロになるのではなく, 各項がすべてゼロになるときのみ成立つため.

$$C_{lk} = \sum_i^{N_v} \sum_k \int [r\chi_l(r)\psi_i(\mathbf{r})]^* r\chi_k(r)\psi_i(\mathbf{r}) d\mathbf{r} \quad (5.31)$$

$$B_l = \sum_i^{N_v} \int [r\chi_l(r)\psi_i(\mathbf{r})]^* r [U^{ex} - \hat{\Omega}] \psi_i(\mathbf{r}) d\mathbf{r} \quad (5.32)$$

$$\sum_k C_{lk} A_k = B_l \quad (5.33)$$

$$CA = B \quad (5.34)$$

これを解けば、式(5.28)を常に満たし、すべての A_m について Δ' を最小化するパラメータセット $\{A_k\}$ が得られる。

■線形フィッティングしたパラメータによる計算結果 ここでは公開されている MCP パラメータ [18] から $\{n_k, \alpha_k\}$ に該当するものを採用し、 $\{A_k\}$ は先に述べたアルゴリズムで決定した。射影演算子 $f, \{\epsilon_c, \psi_c\}$ については、 $f = 2.0$ のみオリジナルのものを採用、 $\{\epsilon_c, \psi_c\}$ については、ほとんどはオリジナルのものを用いたが、Na については cc-pvqz 基底による 1 原子 Hartree-Fock-SCF の解を用いた。

閉殻系については制限軌道 Hartree-Fock SCF、開殻系については非制限軌道 Hartree-Fock SCF を行った。

$\{A_k\}$ の決定に必要な全電子系計算、 $\{A_k\}$ の決定、MCP による計算のすべてを、MADNESS のライブラリを利用して開発したプログラムで行った。

Li, Be, B, C, N, O, F, Ne, Na, Mg について MCP を使用した Hartree-Fock 計算を行い、全電子による軌道エネルギーと比較したものを表 2, 3 に示す。閉殻元素については制限軌道で、開殻元素については非制限軌道で解いた。開殻の場合、電子の多い方を α スピン、少ない方を β スピンとした。軌道名は、軌道形状から水素原子の解析解に当てはめた類推名である。

一部を除き、概ね、数 [kcal/mol] 程度の精度を得た。表 1 における mcp-tzp の値と比較すると、遜色ない結果であると言える。内殻電子・価電子の別なく、完全性の高い基底での計算であるにも関わらず、すべての原子で、物理的に自然な結果をもたらしている。オリジナルの MCP ではポテンシャルが基底に非常に強く依存しているにも関わらず、全く同じ関数形を仮定したポテンシャルでこの結果を得たということは、この関数形自体には基底依存の問題はないこと、また、本研究で提案したパラメータフィッティング手法が、基底変更に対する高いロバスト性を持つことを示している。

オリジナルの MCP においては内殻軌道 $\{\psi_c\}$ についても primitive function の削減を

行なっていたためか, Na に関しては $\{A_k\}$ のパラメータフィッティングのみでは正しい結果は得られなかった. そのため, 別途計算した, より大きな基底で解いた軌道を利用する必要があった.

より精度を上げる方法としては, $\{n_k, \alpha_k\}$ について非線形最適化を施すことが考えられるが, これについては意味のある結果は得られなかった. 項数を増やせば, それだけ自由度が上がるが, 精度はそれほど上昇しない. これは, MCP が想定する関数形では表現できない成分の寄与が, 実際の全電子計算では重要である, ということであろう.

本研究では Hartree-Fock を利用したが, 一般には U^{ex} として任意の二電子項を与えることができる. つまり DFT にある種々の交換相関汎関数を与えることができる.

本研究では考慮しなかったが, 相対論効果が無視できない重元素のポテンシャルを作成する際には, Hartree-Fock はリファレンスとして不適切であるので, Dirac-Hartree-Fock 計算が必要になると考えられる. MCP の射影演算子は内殻軌道と対応する軌道エネルギーを利用しているが, これは, 非相対論的な Hartree-Fock 計算で現れる内殻成分の固有値をシフトアップするためのものであり, 全電子 Dirac-Hartree-Fock 計算によって得られた内殻軌道とは, 一般に異なる. 従って, 相対論的 MCP を本研究による手法で最適化するには, 以下の様な手順により繰り返しパラメータセットを計算し, 収束させる必要があると考えられる.

1. 射影演算子項を除いて最適化を施し, パラメータセットを得る.
2. 得られたパラメータから構成したポテンシャルで非相対論的 1 原子 Hartree-Fock を解き, 内殻成分の軌道と軌道エネルギーを得る.
3. 得られた軌道と軌道エネルギーから射影演算子を構成し, 再度最適化を施してパラメータセットを得る.
4. 前回のパラメータセットと十分近ければ終了. そうでなければ 2 に戻る.

表 2 原子 Hartree-Fock による軌道エネルギー (Li-N)

原子と軌道	軌道エネルギー [a.u.]		エネルギー差 [kcal/mol]
	全電子	MCP	
Li 2s(α)	-0.19637	-0.19462	-1.0981
Be 2s	-0.30927	-0.31021	0.74234
B 2p ₁ (α)	-0.31887	-0.30342	-9.6950
B 2s(α)	-0.54522	-0.56286	11.069
B 2s(β)	-0.44614	-0.46691	13.033
C 2p ₂ (α)	-0.43909	-0.43443	-2.9242
C 2p ₁ (α)	-0.43909	-0.43443	-2.9242
C 2s(α)	-0.82921	-0.83648	4.5620
C 2s(β)	-0.58341	-0.59114	4.8506
N 2p ₃ (α)	-0.57092	-0.56624	-2.9367
N 2p ₂ (α)	-0.57092	-0.56624	-2.9367
N 2p ₁ (α)	-0.57092	-0.56624	-2.9367
N 2s(α)	-1.1630	-1.1746	7.2791
N 2s(β)	-0.72579	-0.73738	7.2728

表3 原子 Hartree-Fock による軌道エネルギー (O-Mg)

原子と軌道	軌道エネルギー [a.u.]		エネルギー差 [kcal/mol]	
	全電子	MCP		
O	2p ₃ (α)	-0.61170	-0.60167	-6.2939
	2p ₂ (α)	-0.71105	-0.70030	-6.7457
	2p ₁ (α)	-0.71105	-0.70030	-6.7457
	2s(α)	-1.4183	-1.4619	27.359
	2p ₁ (β)	-0.52164	-0.51248	-5.7480
	2s(β)	-1.0755	-1.1193	27.485
F	2p ₃ (α)	-0.73160	-0.72155	-6.3065
	2p ₂ (α)	-0.73160	-0.72155	-6.3065
	2p ₁ (α)	-0.84535	-0.83452	-6.7959
	2s(α)	-1.6724	-1.7296	35.894
	2p ₂ (β)	-0.67985	-0.67012	-6.1057
	2p ₁ (β)	-0.67985	-0.67012	-6.1057
	2s(β)	-1.4769	-1.5338	35.705
Ne	2p ₃	-0.8504	-0.84755	-1.7884
	2p ₂	-0.8504	-0.84755	-1.7884
	2p ₁	-0.8504	-0.84755	-1.7884
	2s	-1.9304	-1.9494	11.923
Na	3s(α)	-0.18219	-0.18266	0.29493
	2p ₃ (α)	-1.5191	-1.5170	-1.3178
	2p ₂ (α)	-1.5191	-1.5170	-1.3178
	2p ₁ (α)	-1.5191	-1.5170	-1.3178
	2p ₃ (β)	-1.5171	-1.5159	-0.75301
	2p ₂ (β)	-1.5171	-1.5160	-0.69026
	2p ₁ (β)	-1.5171	-1.5160	-0.69026
Mg	3s	-0.25305	-0.25193	-0.70281
	2p ₃	-2.2822	-2.2918	6.0241
	2p ₂	-2.2822	-2.2918	6.0241
	2p ₁	-2.2822	-2.2918	6.0241

一般に擬ポテンシャル法を利用した計算では、全エネルギーを全電子計算と直接比較することはできないが、距離に応じた全エネルギーの形状、すなわち PES(Potential Energy Surface)は、適当にスケーリングすることで比較できる。この性質により、擬ポテンシャル法は分子の構造最適化に広く用いられている。

線形フィッティングにより求めたパラメータを用いて、Hartree-Fock 近似により PES を計算し、全電子のものと比較した。対象は LiH（水素化リチウム）および BH（水素化ホウ素）とした。

各分子の原子間距離を無限遠に近づけた時の全エネルギー、及び結合解離エネルギーを表 4 に示す。原子間距離無限遠の極限では、LiH は Li^+ と H^- に、BH は B^+ と H^- に解離する。

図 9 に LiH の、10 に BH の、PES を図示する。横軸は原子間距離 [a.u.]、縦軸は、左が全電子計算における全エネルギー、右が MCP を利用した計算における全エネルギーである。グラフ上端が、原子間距離を無限遠にしたときのエネルギーである。いずれの分子も、距離を離すに従ってなめらかに解離状態のエネルギーへ漸近した。

また、それぞれの最安定距離を表 5 に示す。

いずれも、形状がよく一致しているのがわかる。特に最安定距離がほぼ等しく、軌道エネルギーの絶対値としてはそれなりに誤差を含んでいても、距離に対する全エネルギーの相対的な変化については、問題ないといえる。

表 4 Hartree-Fock による結合解離エネルギー

	$E_{R=\infty}^{total}$ [a.u.]		結合解離エネルギー [kcal/mol]			
	全電子	MCP	全電子	MCP	差分	相対誤差
LiH	-7.724	-0.4879	165.036	166.506	1.470	0.8907%
BH	-24.73	-2.782	254.875	249.869	-5.007	-1.9643%

表 5 Hartree-Fock による最安定距離 [a.u.]

誤差	全電子	MCP
LiH	3.05	3.00
BH	2.30	2.30

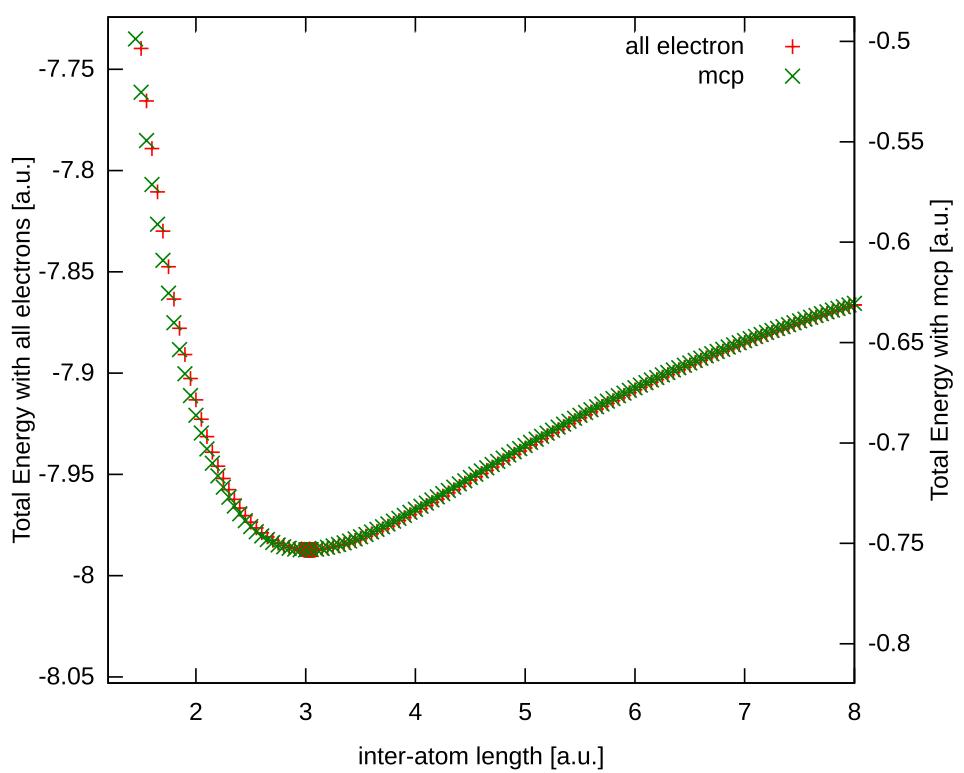


図 9 MCP による PES(LiH)

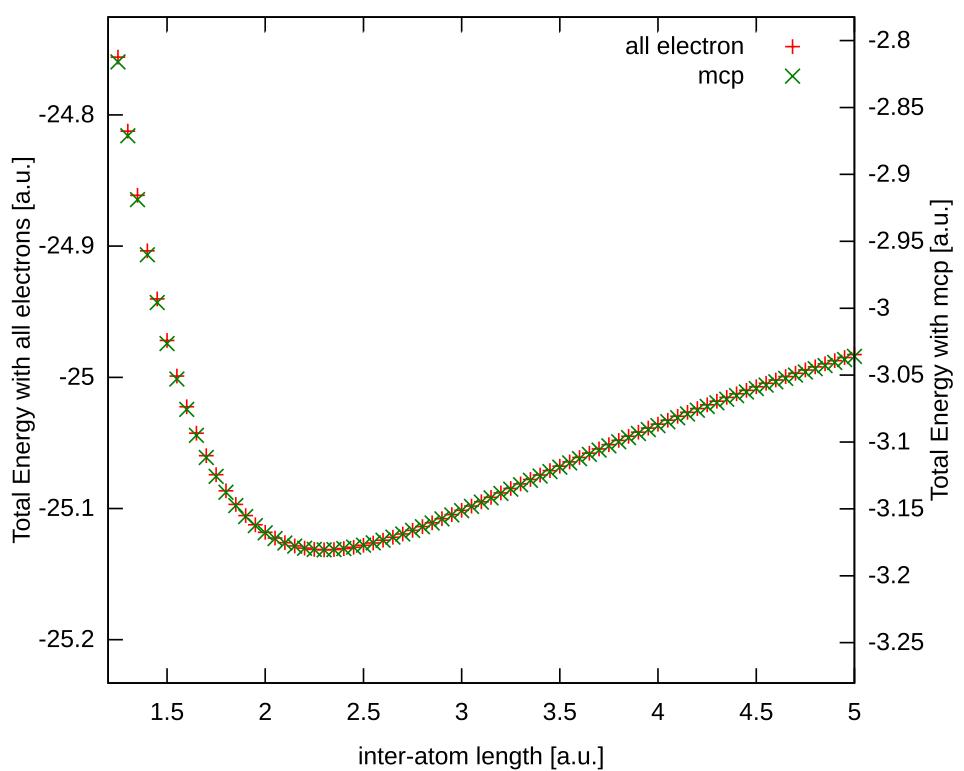


図 10 MCP による PES(BH)

6 総括

本研究では、既存の量子化学計算に利用される技術を、MADNESSを利用して動作するプログラムへ実装することにより、既存技術の LCAO 依存性を明らかにし、3 次元直交基底上での応用可能性について調べた。

LCAO での CPHF/KS では、占有空間に対する補空間が必要であるにもかかわらず、非常に限られた基底で計算を行なっていたために起こっていた問題の一端を明らかにできた。

MCP の実装では、ポテンシャルの基底依存性の高さと、相性の悪い基底での計算結果の問題を明らかにし、それに対して基底依存性の低い手法を新たにつくり、結果を評価した。

LCAO 基底は、現代から見れば非常に低い計算能力でも有用な計算が可能な、優れた手法であり、数十年に渡って、多くの研究者により、様々な応用が考えられてきたパラダイムである。意識的か無意識的かを問わず、それが当然の前提とされてきただけに見落とされてきた、あるいは近似により切り捨てられてきた部分はまだ多くあると考えられ、物理の理解と観測値の再現のために、見なおしていく必要があるだろう。

この研究がその一助となれば幸いである。

謝辞

本研究にあたり、指導教員として終始あたたかく助言と指導頂いた豊橋技術科学大学教授 関野秀男先生、研究内容のみでなく生活面でも指導頂いた同大学助教 墨智成先生、共同研究という形でさまざまな支援、指導を頂いた、米 University of Tennessee および Oak Ridge National Laboratory 兼任教授 Robert J. Harrison 先生、2 度の渡米にあたって資金援助を頂いた神野教育財団様、そして研究室の皆様に心から御礼申し上げます。

参考文献

- [1] S. Jaffard, Y. Meyer, and R. D. Ryan, *Wavelets: Tools for Science & Technology* (Society for Industrial and Applied Mathematics, Philadelphia, 2001).
- [2] I. Daubechies, *Ten Lectures on Wavelets* (SIAM, Philadelphia, 1992).
- [3] B. Alpert, G. Beylkin, D. Gines, and L. Vozovoi, *J. Comput. Phys.* 182, 149

(2002).

- [4] T. Yanai, G. I. Fann, Z. Gan, R. J. Harrison, and G. Beylkin, *J. Chem. Phys.* **121**, 2866 (2004); **121**, 6680 (2004).
- [5] R. J. Harrison, G. I. Fann, T. Yanai, Z. Gan, and G. Beylkin, *J. Chem. Phys.* **121**, 11587 (2004).
- [6] H. Sekino, Y. Maeda, T. Yanai, and R. J. Harrison, *J. Chem. Phys.* **129**, 034111 (2008).
- [7] R.J. Harrison, G.I. Fann, T. Yanai and G. Beylkin, Computational Science - ICCS 2003, Lecture Notes in Computer Science (Springer, 2003), pp. 103-110.
コードベース : <http://code.google.com/p/m-a-d-n-e-s-s/>
- [8] S. F. Boys, *Proc. Roy. Soc.(London)*, **A200**, 542 (1950).
- [9] R. McWeeny, *Proc. Camb.Phil.Soc.* **45**, 315(1949).
- [10] R. J. Harrison, *J. Comput. Chem.* **25**, 328-334 (2004).
- [11] Kato, T., Yokoi, Y. and Sekino, H. (2012), *Int. J. Quantum Chem.* doi: 10.1002/qua.24148
- [12] L. F. Pacios, P. A. Christiansen, *J. Chem. Phys.* **82**(6), 2664 (1984).
- [13] C. Hartwigsen, S. Goedecker, and J. Hutter, *Physical Review B* **58**(7), 3641, (1998).
- [14] V. Bonifacic, S. Huzinaga, *J. Chem. Phys.* **60**(7), 2779, (1974).; **62**(4), 1507, (1975).; **62**(4), 1509, (1975).; **64**(3), 956, (1976).; **65**(6), 2322, (1976).
- [15] Y. Sakai, S. Huzinaga, *J. Chem. Phys.* **76**(5), 2537, (1982).; **76**(5), 2552, (1982).
- [16] Y. Sakai and E. Miyoshi, M. Klobukowski, S. Huzinaga, *J. Chem. Phys.* **106**(19), 8084, (1997).
- [17] <http://www.msg.ameslab.gov/gamess/>
- [18] <http://meg.cube.kyushu-u.ac.jp/~meg/MCP1.html> ※ 2012 年 11 月現在消滅。
一覧は WebArchive に残っているが、各原子のパラメータを閲覧することはできない。GAMESS[17] からパラメータセットを取得することができる。

付録 A

MCP を組み込んだ分子 Hartree-Fock/DFT 計算プログラム `moldft` のソースコードのうち, MCP に関わる一部を抜粋し, かつ紙面に収まるように改変して掲載する. メインプログラムは `moldft.cc` であるが, `main` 関数は省略した. また, `moldft` を構成するコードはこれだけではない.

最新のソースコードは [7] で参照できる.

`corepotential.h`

```
/*
This file is part of MADNESS.
```

Copyright (C) 2007,2010 Oak Ridge National Laboratory

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more information please contact:

Robert J. Harrison

Oak Ridge National Laboratory
One Bethel Valley Road
P.O. Box 2008, MS-6367

email: harrisonrj@ornl.gov
tel: 865-241-3937
fax: 865-572-0680

```
$Id$  
*/  
  
#ifndef MADNESS_COREPOTENTIAL_H  
#define MADNESS_COREPOTENTIAL_H  
  
/// \file corepotential.h  
/// \brief Declaration of core potential related class  
  
#include <madness_config.h>  
#include <constants.h>  
#include <moldft/atomutil.h>  
#include <world/print.h>  
#include <map>  
#include <set>  
#include <vector>  
#include <string>  
#include <cmath>  
#include <iostream>  
#include <sstream>  
using std::cout;  
using std::endl;  
using std::vector;
```

```

/// Represents a core potential

/// General Core Potential is able to write down as following form:
/// 
$$U(r) = \sum_k A_k r^{(n_k-2)} \exp(-\alpha_k r^2)$$

/// 
$$\sum_m |Y_{lm}| \langle Y_{lm} |$$

/// CorePotential holds these parameters (l,n,A,alpha)
///

/// Note: CorePotential::eval() currently ignores 'l'.
/// (It means ' $\sum_m |Y_{lm}| \langle Y_{lm} |$ ' is always '1'.)

struct CorePotential {

    std::vector<int> l; // Angular momentum = 0, 1, 2, ...
    std::vector<int> n;
    std::vector<double> A;
    std::vector<double> alpha;
    double eprec, rcut0, rcut;

    CorePotential() : l(), n(), A(), alpha() {};
    CorePotential(const std::vector<int>& l,
                  const std::vector<int>& n,
                  const std::vector<double>& A,
                  const std::vector<double>& alpha)
        : l(l), n(n), A(A), alpha(alpha), eprec(1e-4),
          rcut0(1.0), rcut(1.0) {};

    double eval(double r) const;

    double eval_derivative(double xi, double r) const;

    std::string to_string () const;

    template <typename Archive>
    void serialize(Archive& ar) {

```

```

        ar & l & n & A & alpha & eprec & rcut0 & rcut;
    }

};

struct CoreOrbital {
    double Bc;
    int type;
    vector<double> coeff, expnt;
    double rsqmax;

    CoreOrbital() : Bc(0), type(0), coeff(), expnt(),
        rsqmax(0.0) {}

    CoreOrbital(int type,
                const std::vector<double>& coeff,
                const std::vector<double>& expnt,
                double Bc)
        : Bc(Bc), type(type), coeff(coeff),
          expnt(expnt)

    {
        double minexpnt = expnt[0];
        for (unsigned int i=1; i<expnt.size(); ++i)
            minexpnt = std::min(minexpnt,expnt[i]);
        rsqmax = 18.4/minexpnt;
    };

    double eval_radial(double rsq) const;

    double eval_radial_derivative(double rsq, double xi) const;

    double eval_spherical_harmonics(int m, double x, double y,
                                    double z, double& dp, int axis) const;

    double eval(int m, double rsq, double x, double y,

```

```

        double z) const;

double eval_derivative(int m, int axis, double xi,
                      double rsq, double x, double y, double z) const;

template <typename Archive>
void serialize(Archive& ar) {
    ar & Bc & type & rsqmax;
    ar & coeff & expnt;
}

};

struct AtomCore {
    unsigned int atomic_number;
    unsigned int ncore;
    std::vector<CoreOrbital> orbital;
    CorePotential potential;

    AtomCore() : atomic_number(0), ncore(0), orbital(),
                 potential() {};

    inline unsigned int n_orbital() const { return ncore; };

    template <typename Archive>
    void serialize(Archive& ar) {
        ar & atomic_number & ncore;
        ar & potential;
        for (std::vector<CoreOrbital>::iterator it=orbital.begin();
             it != orbital.end();
             ++it) {
            ar & (*it);
        }
    }
}

```

```

};

class CorePotentialManager {
    static const double fc;
    std::string core_type;
    ///< core potential type (eg. mcp)
    std::string guess_filename;
    ///< filename of initial guess density data
    std::map<unsigned int,AtomCore> atom_core;
    ///< core potential data mapped atn to potential

public:
    CorePotentialManager() :
        core_type(""),
        guess_filename(""),
        atom_core() {}

    CorePotentialManager(std::string filename, double eprec)
        : core_type(filename) {
        read_file(filename, std::set<unsigned int>(), eprec);
    }

    inline bool is_defined(const unsigned int atn) const {
        return (atom_core.find(atn) != atom_core.end());
    }

    inline unsigned int n_core_orb(const unsigned int atn) const {
        return (*(atom_core.find(atn))).second.n_orbital();
    }

    inline unsigned int n_core_orb_base(const unsigned int atn)
        const {
        return (*(atom_core.find(atn))).second.orbital.size();
    }
}

```

```

}

inline std::string guess_file() const {
    return guess_filename;
}

AtomCore get_atom_core(unsigned int atn) const {
    return atom_core.find(atn)->second;
}

CorePotential get_potential(unsigned int atn) const {
    return atom_core.find(atn)->second.potential;
}

inline unsigned int get_core_l(unsigned int atn,
                               unsigned int core) const {
    return get_atom_core(atn).orbital[core].type;
}

inline double get_core_bc(unsigned int atn,
                          unsigned int core) const {
    return get_atom_core(atn).orbital[core].Bc*fc/2;
}

inline double core_eval(unsigned int atn, unsigned int core,
                       int m, double rsq, double x, double y, double z) const {
    return get_atom_core(atn).orbital[core].eval(m, rsq, x, y, z);
}

inline double core_derivative(unsigned int atn,
                             unsigned int core, int m, int axis, double xi,
                             double rsq, double x, double y, double z) const {
    return get_atom_core(atn).orbital[core]

```

```

.eval_derivative(m, axis, xi, rsq, x, y, z);
}

double potential(unsigned int atn, double r) const {
    AtomCore ac = (*(atom_core.find(atn))).second;
    return ac.potential.eval(r);
}

double potential_derivative(unsigned int atn, double xi,
                           double r) const {
    AtomCore ac = (*(atom_core.find(atn))).second;
    return ac.potential.eval_derivative(xi, r);
}

void read_file(std::string filename,
               std::set<unsigned int> atomset, double eprec);

void set_eprec(double value);

void set_rcut(double value);

template <typename Archive>
void serialize(Archive& ar) {
    ar & core_type;
    ar & guess_filename;
    typedef std::map<unsigned int, AtomCore>::iterator Iter;
    for (Iter it=atom_core.begin();
         it != atom_core.end();
         ++it) {
        ar & it->first & it->second;
    }
}
};


```

```
#endif
```

```
corepotential.cc
```

```
/*
```

```
This file is part of MADNESS.
```

```
Copyright (C) 2007,2010 Oak Ridge National Laboratory
```

```
This program is free software; you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published by  
the Free Software Foundation; either version 2 of the License, or  
(at your option) any later version.
```

```
This program is distributed in the hope that it will be useful,  
but WITHOUT ANY WARRANTY; without even the implied warranty of  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public License  
along with this program; if not, write to the Free Software  
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

```
For more information please contact:
```

Robert J. Harrison
Oak Ridge National Laboratory
One Bethel Valley Road
P.O. Box 2008, MS-6367

```
email: harrisonrj@ornl.gov
tel: 865-241-3937
fax: 865-572-0680
```

```
$Id$
*/
/// \file corepotential.cc
/// \brief Simple management of core potential and orbital information

#include <madness_config.h>
#include <constants.h>
#include <mra/mra.h>
#include <tinyxml/tinyxml.h>
#include <moldft/corepotential.h>
#include <cstdio>
#include <cmath>
#include <string>
#include <sstream>
#include <vector>
#include <utility>
#include <map>
#include <set>
using std::string;
using std::vector;
using namespace madness;

typedef Vector<double,3> coordT;
typedef std::shared_ptr< FunctionFunctorInterface<double,3> > functorT;
typedef Function<double,3> functionT;
typedef FunctionFactory<double,3> factoryT;
typedef vector<functionT> vecfuncT;
```

```

const double CorePotentialManager::fc = 2.0;

double CorePotential::eval(double r) const {
    double u = 0.0;
    double rr = r*r;
    double sp_n = smoothed_potential(r*rcut)*rcut;
    for (unsigned int i=0; i<A.size(); ++i) {
        double rn = 1.0;
        double sp = sp_n;
        if (i==0) {
            sp = smoothed_potential(r*rcut0)*rcut0;
        }
        switch (n[i]) {
            //case 0: rn = sp*sp; break;
            case 1: rn = sp; break;
            case 2: rn = 1.0; break;
            //case 3: rn = r; break;
            //case 4: rn = rr; break;
            default: rn = pow(r, n[i] - 2);
        }
        u += A[i] * rn * exp(-alpha[i] * rr);
    }
    return u;
}

double CorePotential::eval_derivative(double xi, double r) const {
    double u = 0.0;
    double rr = r*r;
    double sp_n = smoothed_potential(r*rcut)*rcut;
    double dsp_n = -dsSmoothed_potential(r*rcut)*rcut*rcut;
    for (unsigned int i=0; i<A.size(); ++i) {
        double rn2 = 1.0;

```

```

        double rn4 = 1.0;
        double sp = sp_n;
        double dsp = dsp_n;
        if (i==0) {
            sp = smoothed_potential(r*rcut0)*rcut0;
            dsp = -dsmoothed_potential(r*rcut0)*rcut0*rcut0;
        }
        switch (n[i]) {
            //case 0: rn2 = sp*sp; rn4 = rn2*rn2; break;
            case 1: rn2 = sp; rn4 = dsp*sp; break;
            case 2: rn2 = 1.0; rn4 = dsp; break;
            //case 3: rn2 = r; rn4 = sp; break;
            //case 4: rn2 = rr; rn4 = 1.0; break;
            default: rn2 = pow(r, n[i] - 2); rn4 = pow(r, n[i] - 4);
        }
        u += A[i] * xi * exp(-alpha[i] * rr)
            * ((n[i]-2) * rn4 - 2.0 * alpha[i] * rn2);
    }
    return u;
}

string CorePotential::to_string () const {
    std::ostringstream oss;
    for (unsigned int i=0; i<A.size(); ++i) {
        oss.precision(8);
        oss << std::scientific;
        std::string sep = "      ";
        oss << l[i] << sep
            << n[i] << sep
            << alpha[i] << sep
            << A[i] << endl;
    }
    return oss.str();
}

```

```

}

double CoreOrbital::eval_radial(double rsq) const {
    double s=0.0;
    for (unsigned int k=0; k<expnt.size(); ++k) {
        s += coeff[k] * pow((2 * expnt[k] / madness::constants::pi), 0.75)
            * exp(-1.0 * expnt[k] * rsq);
    }
    return s;
}

double CoreOrbital::eval_radial_derivative(double rsq, double xi) const {
    double s=0.0;
    for (unsigned int k=0; k<expnt.size(); ++k) {
        s += coeff[k] * pow((2 * expnt[k] / madness::constants::pi), 0.75)
            * exp(-1.0 * expnt[k] * rsq) * (-2.0 * expnt[k] * xi);
    }
    return s;
}

double CoreOrbital::eval_spherical_harmonics(int m, double x, double y,
                                             double z, double& dp, int axis=0) const {
    double p = 1.0;
    dp = 0.0;
    switch (type) {
        case 0:
            break;
        case 1:
            switch (m) {
                case 0: p *= x; if (axis == 0) dp = 1.0; break;
                case 1: p *= y; if (axis == 1) dp = 1.0; break;
                case 2: p *= z; if (axis == 2) dp = 1.0; break;
                default: throw "INVALID MAGNETIC QUANTUM NUMBER";
            }
    }
}

```

```

    }

    break;

case 2:

{ // braces need by some compilers to limit scope of fac
    static const double fac = 1.0; //sqrt(3.0);

    switch (m) {

        case 0: p *= x*x; if (axis == 0) dp = 2*x; break;
        case 1: p *= x*y*fac;
                  if (axis == 0) dp = y*fac;
                  else if (axis == 1) dp = x*fac;
                  break;
        case 2: p *= x*z*fac;
                  if (axis == 0) dp = z*fac;
                  else if (axis == 2) dp = x*fac;
                  break;
        case 3: p *= y*y; if (axis == 1) dp = 2*y; break;
        case 4: p *= y*z*fac;
                  if (axis == 1) dp = z*fac;
                  else if (axis == 2) dp = y*fac;
                  break;
        case 5: p *= z*z; if (axis == 2) dp = 2*z; break;
        default: throw "INVALID MAGNETIC QUANTUM NUMBER";
    }
}

break;

case 3:

switch (m) {

    case 0: p *= x*x*x;
              if (axis == 0) dp = 3*x*x;
              break;
    case 1: p *= x*x*y;
              if (axis == 0) dp = 2*x*y;
              else if (axis == 1) dp = x*x;

```

```

        break;
case 2: p *= x*x*z;
        if (axis == 0) dp = 2*x*z;
        else if (axis == 2) dp = x*x;
        break;
case 3: p *= x*y*y;
        if (axis == 0) dp = y*y;
        else if (axis == 1) dp = 2*x*y;
        break;
case 4: p *= x*y*z;
        if (axis == 0) dp = y*z;
        else if (axis == 1) dp = x*z;
        else dp = x*y;
        break;
case 5: p *= x*z*z;
        if (axis == 0) dp = z*z;
        else if (axis == 2) dp = 2*x*z;
        break;
case 6: p *= y*y*y;
        if (axis == 1) dp = 3*y*y;
        break;
case 7: p *= y*y*z;
        if (axis == 1) dp = 2*y*z;
        else if (axis == 2) dp = y*y;
        break;
case 8: p *= y*z*z;
        if (axis == 1) dp = z*z;
        else if (axis == 2) dp = 2*y*z;
        break;
case 9: p *= z*z*z;
        if (axis == 2) dp = 3*z*z;
        break;
default: throw "INVALID MAGNETIC QUANTUM NUMBER";

```

```

    }

    break;

default:
    throw "UNKNOWN ANGULAR MOMENTUM";
}

return p;
}

double CoreOrbital::eval(int m, double rsq, double x, double y,
    double z) const {
if (m < 0 || m >= (type+1)*(type+2)/2)
    throw "INVALID MAGNETIC QUANTUM NUMBER";
double R = eval_radial(rsq);
if (fabs(R) < 1e-8) {
    return 0.0;
}
double dummy;
double p = eval_spherical_harmonics(m, x, y, z, dummy);
return R*p;
}

double CoreOrbital::eval_derivative(int m, int axis, double xi,
    double rsq, double x, double y, double z) const {
if (m < 0 || m >= (type+1)*(type+2)/2)
    throw "INVALID MAGNETIC QUANTUM NUMBER";
double R = eval_radial(rsq);
double dR = eval_radial_derivative(rsq, xi);
if (fabs(R) < 1e-8) {
    return 0.0;
}
double dp;
double p = eval_spherical_harmonics(m, x, y, z, dp, axis);

```

```

    return dR*p + R*dp;
}

static const string dir = "coredata/";

static bool read_potential(TiXmlElement* elem, AtomCore& ac, double eprec) {
    TiXmlElement* p = elem->FirstChildElement("potential");
    if (!p) return false;

    std::istringstream iss(p->GetText());
    int l, n;
    double e, c;
    vector<int> vl, vn;
    vector<double> ve, vc;
    while (iss >> l) {
        iss >> n >> e >> c;
        if (l<0) continue;
        vl.push_back(l);
        vn.push_back(n);
        ve.push_back(e);
        vc.push_back(c);
    }
    ac.potential.l = vl;
    ac.potential.n = vn;
    ac.potential.A = vc;
    ac.potential.alpha = ve;
    ac.potential.eprec = eprec;
    int atn = ac.atomic_number;
    int ncore = ac.ncore;
    //ac.potential.rcut0 = 1.0/smoothing_parameter(ncore*2, eprec);
    ac.potential.rcut0 = 1.0/smoothing_parameter(ncore*2, 1.0);
    //ac.potential.rcut = 1.0/smoothing_parameter(atn-ncore*2, eprec);
    ac.potential.rcut = 1.0/smoothing_parameter(atn-ncore*2, 1.0);
}

```

```

    return true;
}

static bool read_orbital(TiXmlElement* e, AtomCore& ac) {
    TiXmlElement* p = e->FirstChildElement("core");
    if (!p) return false;

    std::istringstream i_num(p->Attribute("num"));
    i_num >> ac.ncore;

    vector<CoreOrbital> vc;

    for (TiXmlElement* node = p->FirstChildElement("orbital");
         node;
         node = node->NextSiblingElement("orbital")) {
        int type;
        vector<double> coeff, expnt;
        double c, e;
        double bc;
        std::istringstream i_bc(node->Attribute("bc"));
        i_bc >> bc;
        std::istringstream i_type(node->Attribute("type"));
        i_type >> type;
        std::istringstream iss(node->GetText());
        while (iss >> e) {
            iss >> c;
            coeff.push_back(c);
            expnt.push_back(e);
        }
        CoreOrbital co(type, coeff, expnt, bc);
        vc.push_back(co);
    }
}

```

```

    ac.orbital = vc;

    return true;
}

static AtomCore read_atom(TiXmlElement* e, unsigned int atn, double eprec) {
    AtomCore ac;
    ac.atomic_number = atn;

    if (!read_orbital(e, ac)) {
        MADNESS_EXCEPTION("CORE_INFO: read_orbital failed.", -1);
    }
    if (!read_potential(e, ac, eprec)) {
        MADNESS_EXCEPTION("CORE_INFO: read_potential failed.", -1);
    }

    return ac;
}

void CorePotentialManager::read_file(string filename,
                                     std::set<unsigned int> atomset, double eprec) {
    core_type = filename;
    guess_filename = dir + filename + "_guess";

    TiXmlDocument doc(dir + core_type);
    if (!doc.LoadFile()) {
        MADNESS_EXCEPTION(("CORE_INFO: Failed to load core_info data file: "
                           + dir + core_type).c_str(), -1);
        return;
    }
    TiXmlElement* core_info = doc.FirstChildElement();
    if (!core_info) {
        MADNESS_EXCEPTION("CORE_INFO: core_info data file is not valid.", -1);
    }
}

```

```

        return;
    }

    for (TiXmlElement* node = core_info->FirstChildElement("atom");
         node;
         node = node->NextSiblingElement("atom")) {
        unsigned int atn = symbol_to_atomic_number(node->Attribute("symbol"));
        if (atomset.find(atn) != atomset.end()) {
            AtomCore ac = read_atom(node, atn, eprec);
            if (ac.n_orbital() == 0) {
                MADNESS_EXCEPTION("CORE_INFO: read_atom Failed.", -1);
                return;
            }
            atom_core.insert(std::pair<unsigned int,AtomCore>(atn, ac));
        }
    }

    vector<unsigned int> atns;
    typedef std::map<unsigned int, AtomCore>::iterator Iter;
    for (Iter it = atom_core.begin(); it != atom_core.end(); ++it) {
        atns.push_back(it->first);
    }
    madness::print("MCP parameters loaded for atomic numbers:", atns);
}

void CorePotentialManager::set_eperc(double value) {
    typedef std::map<unsigned int,AtomCore>::iterator Iter;
    for (Iter it=atom_core.begin(); it != atom_core.end(); ++it) {
        it->second.potential.eperc = value;
        double q0 = it->second.ncore * 2;
        double q = it->first - it->second.ncore * 2;
        it->second.potential.rcut0 = 1.0 / smoothing_parameter(q0, value);
        it->second.potential.rcut = 1.0 / smoothing_parameter(q, value);
    }
}

```

```
    }

}

void CorePotentialManager::set_rcut(double value) {
    typedef std::map<unsigned int,AtomCore>::iterator Iter;
    for (Iter it=atom_core.begin(); it != atom_core.end(); ++it) {
        it->second.potential.rcut0 = (value<=0.0) ? 1.0 : value;
        it->second.potential.rcut = (value<=0.0) ? 1.0 : value;
    }
}
```

molecule.h

```
/*
This file is part of MADNESS.
```

Copyright (C) 2007,2010 Oak Ridge National Laboratory

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more information please contact:

Robert J. Harrison
Oak Ridge National Laboratory
One Bethel Valley Road
P.O. Box 2008, MS-6367

email: harrisonrj@ornl.gov
tel: 865-241-3937
fax: 865-572-0680

```
$Id$  
*/  
#ifndef MADNESS_MOLECULE_H  
#define MADNESS_MOLECULE_H  
  
/// \file moldft/molecule.h  
/// \brief Declaration of molecule related classes and functions  
  
#include <moldft/corepotential.h>  
#include <moldft/atomutil.h>  
#include <world/array.h>  
#include <vector>  
#include <string>  
#include <iostream>  
#include <fstream>  
#include <sstream>  
#include <algorithm>  
#include <cctype.h>  
#include <cmath>  
#include <tensor/tensor.h>  
#include <misc/misc.h>
```

```

class Atom {

public:
    double x, y, z, q;           ///< Coordinate and charge in atomic units
    unsigned int atomic_number;  ///< Atomic number

    explicit Atom(double x, double y, double z, double q,
                  unsigned int atomic_number)
        : x(x), y(y), z(z), q(q), atomic_number(atomic_number) {}

    Atom(const Atom& a)
        : x(a.x), y(a.y), z(a.z), q(a.q),
          atomic_number(a.atomic_number) {}

    /// Default construct makes a zero charge ghost atom at origin
    Atom()
        : x(0), y(0), z(0), q(0), atomic_number(0) {}

    madness::Vector<double,3> get_coords() const {
        return madness::vec(x, y, z);
    }

    template <typename Archive>
    void serialize(Archive& ar) {
        ar & x & y & z & q & atomic_number;
    }
};

std::ostream& operator<<(std::ostream& s, const Atom& atom);

class Molecule {
private:

```

```

// If you add more fields don't forget to serialize them
std::vector<Atom> atoms;
std::vector<double> rcut; // Reciprocal of the smoothing radius
double eprec; // Error in energy/atom due to smoothing
CorePotentialManager core_pot;
madness::Tensor<double> field;

void swapaxes(int ix, int iy);

template <typename opT>
bool test_for_op(double xaxis, double yaxis, double zaxis, opT op) const;

bool test_for_c2(double xaxis, double yaxis, double zaxis) const;

bool test_for_sigma(double xaxis, double yaxis, double zaxis) const;

bool test_for_inverse() const;

public:
    /// Makes a molecule with zero atoms
    Molecule() : atoms(), rcut(), eprec(1e-4), core_pot(), field(3L) {};

    Molecule(const std::string& filename);

    void read_file(const std::string& filename);

    void read_core_file(const std::string& filename);

    std::string guess_file() const { return core_pot.guess_file(); };

    unsigned int n_core_orb_all() const ;

    unsigned int n_core_orb(unsigned int atn) const {

```

```

    if (core_pot.is_defined(atn))
        return core_pot.n_core_orb_base(atn);
    else
        return 0;
};

unsigned int get_core_l(unsigned int atn, unsigned int c) const {
    return core_pot.get_core_l(atn, c);
}

double get_core_bc(unsigned int atn, unsigned int c) const {
    return core_pot.get_core_bc(atn, c);
}

double core_eval(int atom, unsigned int core, int m, double x,
                 double y, double z) const;

double core_derivative(int atom, int axis, unsigned int core, int m,
                      double x, double y, double z) const;

bool is_potential_defined(unsigned int atn) const {
    return core_pot.is_defined(atn);
};

bool is_potential_defined_atom(int i) const {
    return core_pot.is_defined(atoms[i].atomic_number);
};

void add_atom(double x, double y, double z, double q, int atn);

int natom() const {
    return atoms.size();
};

```

```
void set_atom_coords(unsigned int i, double x, double y, double z);

madness::Tensor<double> get_all_coords() const;

std::vector< madness::Vector<double,3> > get_all_coords_vec() const;

std::vector<double> atomic_radii;

void set_all_coords(const madness::Tensor<double>& newcoords);

void set_eperc(double value);

void set_rcut(double value);

void set_core_eperc(double value) {
    core_pot.set_eperc(value);
}

void set_core_rcut(double value) {
    core_pot.set_rcut(value);
}

double get_eperc() const {
    return eperc;
}

double bounding_cube() const;

const Atom& get_atom(unsigned int i) const;

void print() const;
```

```
double inter_atomic_distance(unsigned int i,unsigned int j) const;

double nuclear_repulsion_energy() const;

double nuclear_repulsion_derivative(int i, int j) const;

double nuclear_dipole(int axis) const;

double nuclear_charge_density(double x, double y, double z) const;

double mol_nuclear_charge_density(double x, double y, double z) const;

double smallest_length_scale() const;

void identify_point_group();

void center();

void orient();

double total_nuclear_charge() const;

double nuclear_attraction_potential(double x, double y, double z) const;

double molecular_core_potential(double x, double y, double z) const;

double core_potential_derivative(int atom, int axis, double x, double y,
                                double z) const;

double nuclear_attraction_potential_derivative(int atom, int axis,
                                                double x, double y, double z) const;

template <typename Archive>
```

```
void serialize(Archive& ar) {
    ar & atoms & rcut & eprec & core_pot;
}
};

#endif
```

molecule.cc

```
/*
This file is part of MADNESS.
```

Copyright (C) 2007,2010 Oak Ridge National Laboratory

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more information please contact:

Robert J. Harrison

Oak Ridge National Laboratory
One Bethel Valley Road
P.O. Box 2008, MS-6367

email: harrisonrj@ornl.gov
tel: 865-241-3937
fax: 865-572-0680

```
$Id: test.cc 257 2007-06-25 19:09:38Z HartmanBaker $  
*/  
  
/// \file moldft/molecule.cc  
/// \brief Simple management of molecular information and potential  
  
double Molecule::nuclear_repulsion_energy() const {  
    double sum = 0.0;  
    for (unsigned int i=0; i<atoms.size(); ++i) {  
        unsigned int z1 = atoms[i].atomic_number;  
        if (core_pot.is_defined(z1)) z1 -= core_pot.n_core_orb(z1) * 2;  
        for (unsigned int j=i+1; j<atoms.size(); ++j) {  
            unsigned int z2 = atoms[j].atomic_number;  
            if (core_pot.is_defined(z2)) z2 -= core_pot.n_core_orb(z2) * 2;  
            sum += z1 * z2 / inter_atomic_distance(i,j);  
        }  
    }  
    return sum;  
}  
  
double Molecule::nuclear_dipole(int axis) const {  
    double sum = 0.0;  
    for (unsigned int atom = 0; atom < atoms.size(); ++atom) {
```

```

        unsigned int z = atoms[atom].atomic_number;
        if (core_pot.is_defined(z)) z -= core_pot.n_core_orb(z) * 2;
        double r;
        switch (axis) {
            case 0: r = atoms[atom].x; break;
            case 1: r = atoms[atom].y; break;
            case 2: r = atoms[atom].z; break;
            default: MADNESS_EXCEPTION("invalid axis", 0);
        }
        sum += r*z;
    }

    return sum;
}

double Molecule::nuclear_repulsion_derivative(int i, int axis) const {
    double sum = 0.0;
    unsigned int z1 = atoms[i].atomic_number;
    if (core_pot.is_defined(z1)) z1 -= core_pot.n_core_orb(z1) * 2;
    for (unsigned int j=0; j<atoms.size(); ++j) {
        if (j != (unsigned int)(i)) {
            unsigned int z2 = atoms[j].atomic_number;
            if (core_pot.is_defined(z2)) z2 -= core_pot.n_core_orb(z2) * 2;
            double r = inter_atomic_distance(i,j);
            double xx;
            if (axis == 0) xx = atoms[i].x - atoms[j].x;
            else if (axis == 1) xx = atoms[i].y - atoms[j].y;
            else xx = atoms[i].z - atoms[j].z;
            sum -= xx * z1 * z2/ (r * r * r);
        }
    }
    return sum;
}

```

```

unsigned int Molecule::n_core_orb_all() const {
    int natom = atoms.size();
    unsigned int sum = 0;

    for (int i=0; i<natom; ++i) {
        unsigned int atn = atoms[i].atomic_number;
        if (core_pot.is_defined(atn)) sum += core_pot.n_core_orb(atn);
    }

    return sum;
}

double Molecule::core_eval(int atom, unsigned int core, int m,
                           double x, double y, double z) const {
    unsigned int atn = atoms[atom].atomic_number;
    double xx = x - atoms[atom].x;
    double yy = y - atoms[atom].y;
    double zz = z - atoms[atom].z;
    double rsq = xx*xx + yy*yy + zz*zz;
    return core_pot.core_eval(atn, core, m, rsq, xx, yy, zz);
}

double Molecule::core_derivative(int atom, int axis, unsigned int core,
                                 int m, double x, double y, double z) const {
    unsigned int atn = atoms[atom].atomic_number;
    double xx = x - atoms[atom].x;
    double yy = y - atoms[atom].y;
    double zz = z - atoms[atom].z;
    double rsq = xx*xx + yy*yy + zz*zz;
    double xi;
    if (axis == 0) xi = xx;

```

```

    else if (axis == 1) xi = yy;
    else xi = zz;
    return core_pot.core_derivative(atn, core, m, axis, xi, rsq, xx, yy, zz);
}

double Molecule::molecular_core_potential(double x, double y,
                                           double z) const {
    int natom = atoms.size();
    double sum = 0.0;

    for (int i=0; i<natom; ++i) {
        unsigned int atn = atoms[i].atomic_number;
        if (core_pot.is_defined(atn)) {
            double r = distance(atoms[i].x, atoms[i].y, atoms[i].z, x, y, z);
            sum += core_pot.potential(atn, r);
        }
    }

    return sum;
}

double Molecule::core_potential_derivative(int atom, int axis, double x,
                                            double y, double z) const {
    int natom = atoms.size();
    if (natom <= atom) return 0.0;

    unsigned int atn = atoms[atom].atomic_number;
    //if (!core_pot.is_defined(atn)) return 0.0;

    double xi;
    if (axis == 0) xi = x-atoms[atom].x;
    else if (axis == 1) xi = y-atoms[atom].y;
    else xi = z-atoms[atom].z;
}

```

```

    double r = distance(atoms[atom].x, atoms[atom].y, atoms[atom].z, x, y, z);
    return core_pot.potential_derivative(atn, xi, r);
}

void Molecule::read_core_file(const std::string& filename) {
    std::set<unsigned int> atomset;
    int natom = atoms.size();
    for (int i=0; i<natom; ++i) {
        if (atomset.count(atoms[i].atomic_number) == 0)
            atomset.insert(atoms[i].atomic_number);
    }

    core_pot.read_file(filename, atomset, eprec);

    //return;

    // rcut update
    for (int i=0; i<natom; ++i) {
        unsigned int atn = atoms[i].atomic_number;
        if (core_pot.is_defined(atn)) {
            double q = atoms[i].q - core_pot.n_core_orb(atn) * 2;
            if (q == 0.0) {
                rcut[i] = 1.0;
                continue;
            }
            double r = rcut[i];
            rcut[i] = 1.0/smoothing_parameter(q, eprec);
            //rcut[i] = 1.0/smoothing_parameter(q, 1.0);
            madness::print("rcut update", i, r, "to", rcut[i]);
        }
    }

    return;
}

```

}

moldft.cc

/*

This file is part of MADNESS.

Copyright (C) 2007,2010 Oak Ridge National Laboratory

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more information please contact:

Robert J. Harrison
Oak Ridge National Laboratory
One Bethel Valley Road
P.O. Box 2008, MS-6367

email: harrisonrj@ornl.gov

```

tel: 865-241-3937
fax: 865-572-0680

$Id$
*/
/// \file moldft.cc
/// \brief Molecular HF and DFT code
/// \defgroup moldft The molecular density funcitonal and Hartree-Fock code

class MolecularCorePotentialFunctor
    : public FunctionFunctorInterface<double,3> {
private:
    const Molecule& molecule;
public:
    MolecularCorePotentialFunctor(const Molecule& molecule)
        : molecule(molecule) {}

    double operator()(const coordT& x) const {
        return molecule.molecular_core_potential(x[0], x[1], x[2]);
    }

    std::vector<coordT> special_points() const {
        return molecule.get_all_coords_vec();
    }
};

class CorePotentialDerivativeFunctor
    : public FunctionFunctorInterface<double,3> {
private:
    const Molecule& molecule;

```

```

    const int atom;
    const int axis;
    std::vector<coordT> specialpt;
public:
    CorePotentialDerivativeFunctor(const Molecule& molecule,
                                    int atom, int axis)
        : molecule(molecule), atom(atom), axis(axis) {}

    double operator()(const coordT& r) const {
        return molecule.core_potential_derivative(atom, axis,
                                                    r[0], r[1], r[2]);
    }
};

class CoreOrbitalFunctor : public FunctionFunctorInterface<double,3> {
    const Molecule molecule;
    const int atom;
    const unsigned int core;
    const int m;
public:
    CoreOrbitalFunctor(Molecule& molecule, int atom,
                        unsigned int core, int m)
        : molecule(molecule), atom(atom), core(core), m(m) {};
    double operator()(const coordT& r) const {
        return molecule.core_eval(atom, core, m, r[0], r[1], r[2]);
    }
};

class CoreOrbitalDerivativeFunctor
    : public FunctionFunctorInterface<double,3> {
    const Molecule molecule;
    const int atom, axis;
    const unsigned int core;

```

```

    const int m;

public:
    CoreOrbitalDerivativeFunctor(Molecule& molecule, int atom, int axis,
        unsigned int core, int m)
        : molecule(molecule), atom(atom), axis(axis), core(core), m(m) {};
    double operator()(const coordT& r) const {
        return molecule.core_derivative(atom, axis, core, m,
            r[0], r[1], r[2]);
    };
};

struct Calculation {
    Molecule molecule;
    CalculationParameters param;
    XCfunctional xc;
    AtomicBasisSet aobasis;
    functionT vnucl;
    functionT vacuo_rho;
    functionT rhoT;
    functionT rho_elec;
    functionT rhon;
    functionT mol_mask;
    functionT Uabinit;
    functionT mask;
    vecfuncT amo, bmo;
    std::vector<int> aset, bset;
    vecfuncT ao;
    std::vector<int> at_to_bf, at_nbf;
    tensorT aocc, bocc;
    tensorT aeps, beps;
    poperatorT coulop;
    std::vector< std::shared_ptr<real_derivative_3d> > gradop;
    double vtol;
};

```

```

double current_energy;
double esol;//etot;
double vacuo_energy;

Calculation(World & world, const char *filename)
{
    if(world.rank() == 0) {
        molecule.read_file(filename);
        param.read_file(filename);
        unsigned int n_core = 0;
        if (param.core_type != "") {
            molecule.read_core_file(param.core_type);
            param.aobasis = molecule.guess_file();
            n_core = molecule.n_core_orb_all();
        }

        molecule.orient();
        aobasis.read_file(param.aobasis);
        param.set_molecular_info(molecule, aobasis, n_core);
    }

    world.gop.broadcast_serializable(molecule, 0);
    world.gop.broadcast_serializable(param, 0);
    world.gop.broadcast_serializable(aobasis, 0);

    xc.initialize(param.xc_data, !param.spin_restricted);
    //xc.plot();

    FunctionDefaults<3>::set_cubic_cell(-param.L, param.L);
    set_protocol(world, 1e-4);
}

void load_mos(World& world) {
    const double trantol = vtol / std::min(30.0, double(param.nalpha));
}

```

```

const double thresh = FunctionDefaults<3>::get_thresh();
const int k = FunctionDefaults<3>::get_k();
unsigned int nmo;
bool spinrest;
amo.clear(); bmo.clear();

archive::ParallelInputArchive ar(world, "restartdata");

/*
File format:

bool spinrestricted --> if true only alpha orbitals are present

unsigned int nmo_alpha;
Tensor<double> aeps;
Tensor<double> aocc;
vector<int> aset;
for i from 0 to nalpha-1:
.   Function<double,3> amo[i]

repeat for beta if !spinrestricted

*/
// LOTS OF LOGIC MISSING HERE TO CHANGE OCCUPATION NO., SET,
// EPS, SWAP, ... sigh

ar & spinrest;

ar & nmo;
MADNESS_ASSERT(nmo >= unsigned(param.nmo_alpha));
ar & aeps & aocc & aset;
amo.resize(nmo);

```

```

for (unsigned int i=0; i<amo.size(); ++i) ar & amo[i];
unsigned int n_core = molecule.n_core_orb_all();
if (nmo > unsigned(param.nmo_alpha)) {
    aset = vector<int>(aset.begin()+n_core,
                         aset.begin()+n_core+param.nmo_alpha);
    amo = vecfuncT(amo.begin()+n_core,
                   amo.begin()+n_core+param.nmo_alpha);
    aeps = copy(aeps(Slice(n_core, n_core+param.nmo_alpha-1)));
    aocc = copy(aocc(Slice(n_core, n_core+param.nmo_alpha-1)));
}

if (amo[0].k() != k) {
    reconstruct(world, amo);
    for(unsigned int i = 0;i < amo.size();++i)
        amo[i] = madness::project(amo[i], k, thresh, false);
    world.gop.fence();
}
normalize(world, amo);
amo = transform(world, amo, Q3(matrix_inner(world, amo, amo)),
               trantol, true);
truncate(world, amo);
normalize(world, amo);

if (!param.spin_restricted) {

    if (spinrest) { // Only alpha spin orbitals were on disk
        MADNESS_ASSERT(param.nmo_alpha >= param.nmo_beta);
        bmo.resize(param.nmo_beta);
        bset.resize(param.nmo_beta);
        beps = copy(aeps(Slice(0,param.nmo_beta-1)));
        bocc = copy(aocc(Slice(0,param.nmo_beta-1)));
        for (int i=0; i<param.nmo_beta; ++i) bmo[i] = copy(amo[i]);
    }
}

```

```

else {
    ar & nmo;
    ar & beps & bocc & bset;

    bmo.resize(nmo);
    for (unsigned int i=0; i<bmo.size(); ++i) ar & bmo[i];

    if (nmo > unsigned(param.nmo_beta)) {
        bset = vector<int>(bset.begin()+n_core,
                             bset.begin()+n_core+param.nmo_beta);
        bmo = vecfuncT(bmo.begin()+n_core,
                       bmo.begin()+n_core+param.nmo_beta);
        beps = copy(beps(Slice(n_core, n_core+param.nmo_beta-1)));
        bocc = copy(bocc(Slice(n_core, n_core+param.nmo_beta-1)));
    }

    if (bmo[0].k() != k) {
        reconstruct(world,bmo);
        for(unsigned int i = 0;i < bmo.size();++i)
            bmo[i] = madness::project(bmo[i], k, thresh, false);
        world.gop.fence();
    }

    normalize(world, bmo);
    bmo = transform(world, bmo,
                    Q3(matrix_inner(world, bmo, bmo)), trantol, true);
    truncate(world, bmo);
    normalize(world, bmo);

}
}

```

```

void make_nuclear_potential(World & world)
{
    START_TIMER(world);
    vnucl = factoryT(world)
        .functor(functorT(new MolecularPotentialFunctor(molecule)))
        .thresh(vtol)
        .truncate_on_project();
    vnucl.set_thresh(FunctionDefaults<3>::get_thresh());
    vnucl.reconstruct();
    END_TIMER(world, "Project vnucl");
    if (param.core_type != "") {
        START_TIMER(world);
        functionT c_pot =
            factoryT(world)
                .functor(functorT(
                    new MolecularCorePotentialFunctor(molecule)))
                .thresh(vtol)
                .initial_level(4);
        c_pot.set_thresh(FunctionDefaults<3>::get_thresh());
        c_pot.reconstruct();
        END_TIMER(world, "Project Core Pot.");
        vnucl += c_pot;
        vnucl.truncate();
    }
}

vecfuncT core_projection(World & world, const vecfuncT & psi,
                        const bool include_Bc = true)
{
    int npsi = psi.size();
    if (npsi == 0) return psi;
    int natom = molecule.natom();
    vecfuncT proj = zero_functions<double,3>(world, npsi);

```

```

tensorT overlap_sum(static_cast<long>(npsi));

for (int i=0; i<natom; ++i) {
    Atom at = molecule.get_atom(i);
    unsigned int atn = at.atomic_number;
    unsigned int nshell = molecule.n_core_orb(atn);
    if (nshell == 0) continue;
    for (unsigned int c=0; c<nshell; ++c) {
        unsigned int l = molecule.get_core_l(atn, c);
        int max_m = (l+1)*(l+2)/2;
        nshell -= max_m - 1;
        for (int m=0; m<max_m; ++m) {
            functionT core =
                factoryT(world)
                    .functor(functorT(
                        new CoreOrbitalFunctor(molecule, i, c, m)));
            tensorT overlap = inner(world, core, psi);
            overlap_sum += overlap;
            for (int j=0; j<npsi; ++j) {
                if (include_Bc)
                    overlap[j] *=molecule.get_core_bc(atn, c);
                proj[j] += core.scale(overlap[j]);
            }
        }
        world.gop.fence();
    }
    if (world.rank() == 0) print("sum_k <core_k|psi_i>:", overlap_sum);
    return proj;
}

double core_projector_derivative(World & world, const vecfuncT & mo,
                                const tensorT & occ, int atom, int axis)

```

```

{

    vecfuncT cores, dcores;
    std::vector<double> bc;
    unsigned int atn = molecule.get_atom(atom).atomic_number;
    unsigned int ncore = molecule.n_core_orb(atn);

    // projecting core & d/dx core
    for (unsigned int c=0; c<ncore; ++c) {
        unsigned int l = molecule.get_core_l(atn, c);
        int max_m = (l+1)*(l+2)/2;
        for (int m=0; m<max_m; ++m) {
            functorT func =
                functorT(new CoreOrbitalFunctor(molecule, atom, c, m));
            cores.push_back(functionT(
                factoryT(world)
                    .functor(func)
                    .truncate_on_project()));
            func = functorT(new CoreOrbitalDerivativeFunctor(molecule,
                atom, axis, c, m));
            dcores.push_back(functionT(
                factoryT(world)
                    .functor(func)
                    .truncate_on_project()));
            bc.push_back(molecule.get_core_bc(atn, c));
        }
    }

    // calc \sum_i occ_i <psi_i|(\sum_c Bc d/dx |core><core|)|psi_i>
    double r = 0.0;
    for (unsigned int c=0; c<cores.size(); ++c) {
        double rcore= 0.0;
        tensorT rcores = inner(world, cores[c], mo);
        tensorT rdcores = inner(world, dcores[c], mo);

```

```

        for (unsigned int i=0; i<mo.size(); ++i) {
            rcore += rdcores[i] * rcores[i] * occ[i];
        }
        r += 2.0 * bc[c] * rcore;
    }

    return r;
}

void initial_guess(World & world)
{
    START_TIMER(world);
    if (param.restart) {
        load_mos(world);
    }
    else {
        // Use the initial density and potential
        // to generate a better process map
        functionT rho =
            factoryT(world)
            .functor(functorT(
                new MolecularGuessDensityFunctor(molecule, aobasis)))
            .truncate_on_project();
    END_TIMER(world, "guess density");
    double nel = rho.trace();
    if(world.rank() == 0)
        print("guess dens trace", nel);

    if(world.size() > 1) {
        START_TIMER(world);
        LoadBalanceDeux<3> lb(world);
        lb.add_tree(vnuc, lbcost<double,3>(1.0, 0.0), false);
        lb.add_tree(rho, lbcost<double,3>(1.0, 1.0), true);
    }
}

```

```

        FunctionDefaults<3>::redistribute(world,
                                             lb.load_balance(6.0));
        END_TIMER(world, "guess loadbal");
    }

// Diag approximate fock matrix to get initial mos
functionT vlocal;
if(param.nalpha + param.nbeta > 1){
    START_TIMER(world);
    vlocal = vnuc + apply(*coulop, rho);
    END_TIMER(world, "guess Coulomb potn");
    bool save = param.spin_restricted;
    param.spin_restricted = true;
    vlocal = vlocal + make_lda_potential(world, rho);
    vlocal.truncate();
    param.spin_restricted = save;
} else {
    vlocal = vnuc;
}
rho.clear();
vlocal.reconstruct();
if(world.size() > 1){
    LoadBalanceDeux<3> lb(world);
    lb.add_tree(vnuc, lbcost<double,3>(1.0, 1.0), false);
    for(unsigned int i = 0;i < ao.size();++i){
        lb.add_tree(ao[i], lbcost<double,3>(1.0, 1.0), false);
    }
}

FunctionDefaults<3>::redistribute(world, lb.load_balance(6.0));
}

tensorT overlap = matrix_inner(world, ao, ao, true);

```

```

tensorT kinetic = kinetic_energy_matrix(world, ao);
reconstruct(world, ao);
vlocal.reconstruct();
vecfunct vpsi = mul_sparse(world, vlocal, ao, vtol);
compress(world, vpsi);
truncate(world, vpsi);
compress(world, ao);
tensorT potential = matrix_inner(world, vpsi, ao, true);
vpsi.clear();
tensorT fock = kinetic + potential;
fock = 0.5 * (fock + transpose(fock));
tensorT c, e;
sygv(fock, overlap, 1, c, e);
world.gop.broadcast(c.ptr(), c.size(), 0);
world.gop.broadcast(e.ptr(), e.size(), 0);
if(world.rank() == 0 && 0){
    print("initial eigenvalues");
    print(e);
    print("\n\nWSTHORNTON: initial eigenvectors");
    print(c);
}
compress(world, ao);

unsigned int ncore = 0;
if (param.core_type != "") {
    ncore = molecule.n_core_orb_all();
}
amo = transform(world, ao,
                c(_, Slice(ncore, ncore + param.nmo_alpha - 1)), 0.0, true);
truncate(world, amo);
normalize(world, amo);
aeps = e(Slice(ncore, ncore + param.nmo_alpha - 1));

```

```

aocc = tensorT(param.nmo_alpha);
for(int i = 0;i < param.nalpha;++i)
    aocc[i] = 1.0;

aset = std::vector<int>(param.nmo_alpha,0);
if(world.rank() == 0)
    std::cout << "alpha set " << 0 << " " << 0 << "-";

for(int i = 1;i < param.nmo_alpha;++i) {
    aset[i] = aset[i - 1];
    if(aeps[i] - aeps[i - 1] > 1.5 || aocc[i] != 1.0){
        ++(aset[i]);
        if(world.rank() == 0){
            std::cout << i - 1 << std::endl;
            std::cout << "alpha set " << aset[i]
                << " " << i << "-";
        }
    }
}
if(world.rank() == 0)
    std::cout << param.nmo_alpha - 1 << std::endl;

if(param.nbeta && !param.spin_restricted){
    bmo = transform(world, ao,
                    c(_ , Slice(ncore, ncore + param.nmo_beta - 1)),
                    0.0, true);
    truncate(world, bmo);
    normalize(world, bmo);
    beps = e(Slice(ncore, ncore + param.nmo_beta - 1));
    bocc = tensorT(param.nmo_beta);
    for(int i = 0;i < param.nbeta;++i)
        bocc[i] = 1.0;
}

```

```

bset = std::vector<int>(param.nmo_beta,0);
if(world.rank() == 0)
    std::cout << " beta set " << 0 << " " << 0 << "-";
}

for(int i = 1;i < param.nmo_beta;++i) {
    bset[i] = bset[i - 1];
    if(beps[i] - beps[i - 1] > 1.5 || bocc[i] != 1.0){
        ++(bset[i]);
        if(world.rank() == 0){
            std::cout << i - 1 << std::endl;
            std::cout << " beta set " << bset[i]
            << " " << i << "-";
        }
    }
}

if(world.rank() == 0)
    std::cout << param.nmo_beta - 1 << std::endl;
}

}

vecfuncT apply_potential(World & world, const tensorT & occ,
    const vecfuncT & amo, const vecfuncT& vf,
    const vecfuncT& delrho, const functionT & vlocal,
    double & exc, int ispin)
{
    functionT vloc = vlocal;
    exc = 0.0;

    if (xc.is_dft() && !(xc.hf_exchange_coefficient()==1.0)) {
        START_TIMER(world);
#endif MADNESS_HAS_LIBXC
        exc = make_dft_energy(world, vf, ispin);
}

```

```

#else
    if (ispin == 0) exc = make_dft_energy(world, vf, ispin);
#endif
    vloc = vloc + make_dft_potential(world, vf, ispin, 0);
    //print("VLOC1", vloc.trace(), vloc.norm2());

#ifndef MADNESS_HAS_LIBXC
    if (xc.is_gga() ) {
        real_function_3d vsig = make_dft_potential(world,
            vf, ispin, 1);
        //print("VSIG", vsig.trace(), vsig.norm2());
        real_function_3d vr(world);
        for (int axis=0; axis<3; axis++) {
            vr += (*gradop[axis])(vsi);
            //print("VR", vr.trace(), vr.norm2());
        }
        vloc = vloc - vr;
    }
#endif
    END_TIMER(world, "DFT potential");
}

START_TIMER(world);
vecfuncT Vpsi = mul_sparse(world, vloc, amo, vtol);
END_TIMER(world, "V*psi");
if(xc.hf_exchange_coefficient()){
    START_TIMER(world);
    vecfuncT Kamo = apply_hf_exchange(world, occ, amo, amo);
    tensorT excv = inner(world, Kamo, amo);
    double exchf = 0.0;
    for(unsigned long i = 0;i < amo.size();++i){
        exchf -= 0.5 * excv[i] * occ[i];
    }
}

```

```

        if (!xc.is_spin_polarized()) exchf *= 2.0;
        gaxpy(world, 1.0, Vpsi, -xc.hf_exchange_coefficient(), Kamo);
        Kamo.clear();
        END_TIMER(world, "HF exchange");
        exc = exchf* xc.hf_exchange_coefficient() + exc;
    }

    if (param.core_type.substr(0,3) == "mcp") {
        START_TIMER(world);
        gaxpy(world, 1.0, Vpsi, 1.0, core_projection(world, amo));
        END_TIMER(world, "MCP Core Projector");
    }

    START_TIMER(world);
    truncate(world, Vpsi);
    END_TIMER(world, "Truncate Vpsi");
    world.gop.fence();
    return Vpsi;
}

tensorT derivatives(World & world)
{
    START_TIMER(world);

    functionT rho = make_density(world, aocc, amo);
    functionT brho = rho;
    if (!param.spin_restricted) brho = make_density(world, bocc, bmo);
    rho.gaxpy(1.0, brho, 1.0);

    vecfuncT dv(molecule.natom() * 3);
    vecfuncT du = zero_functions<double,3>(world, molecule.natom() * 3);
    tensorT rc(molecule.natom() * 3);
    for(int atom = 0; atom < molecule.natom(); ++atom){

```

```

for(int axis = 0;axis < 3;++axis){
    functorT func(new MolecularDerivativeFunctor(molecule,
        atom, axis));
    dv[atom * 3 + axis] = functionT(
        factoryT(world)
            .functor(func)
            .nofence()
            .truncate_on_project());
    if (param.core_type != ""
        && molecule.is_potential_defined_atom(atom)) {
        // core potential contribution
        func = functorT(
            new CorePotentialDerivativeFunctor(molecule,
                atom, axis));
        du[atom * 3 + axis] = functionT(
            factoryT(world)
                .functor(func)
                .truncate_on_project());

        // core projector contribution
        rc[atom * 3 + axis] =
            core_projector_derivative(world, amo, aocc,
                atom, axis);
        if (!param.spin_restricted) {
            if (param.nbeta) rc[atom * 3 + axis] +=
                core_projector_derivative(world, bmo,
                    bocc, atom, axis);
        }
        else {
            rc[atom * 3 + axis] *= 2 * 2;
            // because of 2 electrons in each
            // valence orbital bra+ket
        }
    }
}

```

```

        }

    }

}

world.gop.fence();
tensorT r = inner(world, rho, dv);
world.gop.fence();
tensorT ru = inner(world, rho, du);
dv.clear();
du.clear();
world.gop.fence();
tensorT ra(r.size());
for(int atom = 0;atom < molecule.natom();++atom){
    for(int axis = 0;axis < 3;++axis){
        ra[atom * 3 + axis] =
            molecule.nuclear_repulsion_derivative(atom, axis);
    }
}
//if (world.rank() == 0) print("derivatives:\n", r, ru, rc, ra);
r += ra + ru + rc;
END_TIMER(world,"derivatives");

if (world.rank() == 0) {
    print("\n Derivatives (a.u.)\n -----");
    print(" atom           x           y           z
          "      dE/dx      dE/dy      dE/dz");
    print(" ----- ----- ----- -----");
    for (int i=0; i<molecule.natom(); ++i) {
        const Atom& atom = molecule.get_atom(i);
        printf(" %5d %12.6f %12.6f %12.6f %12.6f %12.6f %12.6f\n",
               i, atom.x, atom.y, atom.z,
               r[i*3+0], r[i*3+1], r[i*3+2]);
    }
}

```

```
    }
}
return r;
}
};
```

付録 B

MCP パラメータを求めるために作成したプログラムのソースコードを掲載する。

```
/*
This file is part of MADNESS.
```

Copyright (C) 2007,2011 Oak Ridge National Laboratory

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

For more information please contact:

Robert J. Harrison
Oak Ridge National Laboratory

One Bethel Valley Road
P.O. Box 2008, MS-6367

email: harrisonrj@ornl.gov
tel: 865-241-3937
fax: 865-572-0680

```
$Id$  
*/  
  
/// \file mcpfit.cc  
/// \brief fitting parameters of Model Core Potential  
  
#include <mra/mra.h>  
#include <linalg/solvers.h>  
#include <moldft/corepotential.h>  
#include <iostream>  
#include <iomanip>  
#include <set>  
#include <cstdio>  
using namespace madness;  
  
class LevelPmap : public WorldDCPmapInterface< Key<3> > {  
private:  
    const int nproc;  
public:  
    LevelPmap() : nproc(0) {}  
  
    LevelPmap(World& world) : nproc(world.nproc()) {}  
  
    /// Find the owner of a given key  
    ProcessID owner(const Key<3>& key) const {
```

```

        Level n = key.level();
        if (n == 0) return 0;
        hashT hash;
        if (n <= 3 || (n&0x1)) hash = key.hash();
        else hash = key.parent().hash();
        //hashT hash = key.hash();
        return hash%nproc;
    }
};

typedef std::shared_ptr< WorldDCPmapInterface< Key<3> > > pmapT;
typedef Vector<double,3> coordT;
typedef std::shared_ptr< FunctionFunctorInterface<double,3> > functorT;
typedef Function<double,3> functionT;
typedef vector<functionT> vecfuncT;
typedef Tensor<double> tensorT;
typedef FunctionFactory<double,3> factoryT;
typedef SeparatedConvolution<double,3> operatorT;
typedef std::shared_ptr<operatorT> poperatorT;

static double ttt, sss;
void START_TIMER(World& world) {
    world.gop.fence(); ttt=wall_time(); sss=cpu_time();
}

void END_TIMER(World& world, const char* msg) {
    ttt=wall_time()-ttt; sss=cpu_time()-sss;
    if (world.rank()==0) printf("timer: %20.20s %8.2fs %8.2fs\n",
        msg, sss, ttt);
}

inline double mask1(double x) {

```

```

/* Iterated first beta function to switch smoothly
   from 0->1 in [0,1].  n iterations produce 2*n-1
   zero derivatives at the end points. Order of polyn
   is 3^n.

Currently use one iteration so that first deriv.
is zero at interior boundary and is exactly representable
by low order multiwavelet without refinement */

x = (x*x*(3.-2.*x));
return x;
}

double mask3(const coordT& ruser) {
    coordT rsim;
    user_to_sim(ruser, rsim);
    double x= rsim[0], y=rsim[1], z=rsim[2];
    double lo = 0.0625, hi = 1.0-lo, result = 1.0;
    double rlo = 1.0/lo;

    if (x<lo)
        result *= mask1(x*rlo);
    else if (x>hi)
        result *= mask1((1.0-x)*rlo);
    if (y<lo)
        result *= mask1(y*rlo);
    else if (y>hi)
        result *= mask1((1.0-y)*rlo);
    if (z<lo)
        result *= mask1(z*rlo);
    else if (z>hi)
        result *= mask1((1.0-z)*rlo);
}

```

```

    return result;
}

/// Given overlap matrix, return rotation with 3rd order error
/// to orthonormalize the vectors
tensorT Q3(const tensorT& s) {
    tensorT Q = inner(s,s);
    Q.gaxpy(0.2,s,-2.0/3.0);
    for (int i=0; i<s.dim(0); i++) Q(i,i) += 1.0;
    return Q.scale(15.0/8.0);
}

/*
static double dot_product (const tensorT & t1, const tensorT & t2) {
    double s = 0.0;
    for (int i=0; i<t1.dim(0); i++) {
        s += t1[i] * t2[i];
    }
    return s;
}
*/
static tensorT vec2tensor (const vector<double> & v) {
    size_t s = v.size();
    tensorT t(static_cast<long>(s));
    for (unsigned int i=0; i<s; i++) {
        t[i] = v[i];
    }
    return t;
}

static vector<double> tensor2vec (const tensorT & t) {
    size_t s = t.size();

```

```

vector<double> v(s);
for (unsigned int i=0; i<s; i++) {
    v[i] = t[i];
}
return v;
}

void print (const tensorT & t) {
    std::cout << std::scientific << std::setprecision(12);
    std::cout << "[" << std::endl;
    for (unsigned int i=0; i<t.dim(0); ++i) {
        printf("%02d: %.12e\n", i, t[i]);
    }
}

class CorePotentialFunctor : public FunctionFunctorInterface<double,3> {
private:
    const CorePotential& pot;
public:
    CorePotentialFunctor(const CorePotential& pot) : pot(pot) {}
    double operator()(const coordT& x) const {
        double r = sqrt(x[0]*x[0] + x[1]*x[1] + x[2]*x[2]);
        return pot.eval(r);
    }
};

inline static double radius_function (double rsq) {
    //return sqrt(rsq); // r
    return rsq; // r^2
}

class RadiusFunctor : public FunctionFunctorInterface<double,3> {
public:

```

```

RadiusFunctor () {}

double operator() (const coordT& x) const {
    return radius_function(x[0]*x[0] + x[1]*x[1] + x[2]*x[2]);
}

};

class RadiusSquareFunctor : public FunctionFunctorInterface<double,3> {
public:
    RadiusSquareFunctor () {}

    double operator() (const coordT& x) const {
        return x[0]*x[0] + x[1]*x[1] + x[2]*x[2];
    }

};

class GaussianFunctor : public FunctionFunctorInterface<double,3> {
    double alpha;
public:
    GaussianFunctor (double alpha) : alpha(alpha) {}

    double operator() (const coordT& x) const {
        double rsq = x[0]*x[0] + x[1]*x[1] + x[2]*x[2];
        return exp(-alpha*rsq);
    }

};

class PotentialBasisFunctor : public FunctionFunctorInterface<double,3> {
    int n;
    double alpha, rcut;
    double rn;
public:
    PotentialBasisFunctor (int n, double alpha, double rcut)
        : n(n), alpha(alpha), rcut(rcut) {}

    double operator() (const coordT& x) const {
        double rsq = x[0]*x[0] + x[1]*x[1] + x[2]*x[2];

```

```

        double r = sqrt(rsq);
        double sp = smoothed_potential(r*rcut)*rcut;
        double rn = 1.0;
        switch (n) {
            case 0: rn = sp*sp; break;
            case 1: rn = sp; break;
            case 2: rn = 1.0; break;
            case 3: rn = r; break;
            case 4: rn = r*r; break;
            default: rn = pow(r, n - 2);
        }
        return rn * exp(-alpha*rsq);
    }
};

class NcOverR : public FunctionFunctorInterface<double,3> {
    int Nc;
    double c;
public:
    NcOverR (int Nc, double c) : Nc(Nc), c(c) {}
    double operator() (const coordT& x) const {
        return Nc * smoothed_potential(
            sqrt(x[0]*x[0] + x[1]*x[1] + x[2]*x[2]) / c) / c;
    }
};

class CoreOrbitalFunctor : public FunctionFunctorInterface<double,3> {
    CorePotentialManager & cpm;
    unsigned int atn;
    unsigned int core;
    int m;
public:
    CoreOrbitalFunctor (CorePotentialManager & cpm,

```

```

        unsigned int atn, unsigned int core, int m)
: cpm(cpm), atn(atn), core(core), m(m) {}

double operator() (const coordT& x) const {
    double xx = x[0];
    double yy = x[1];
    double zz = x[2];
    double rsq = xx*xx + yy*yy + zz*zz;
    return cpm.core_eval(atn, core, m, rsq, xx, yy, zz);
}

};

struct CalculationParameters {
    double dconv;      ///< conversion criteria
    double L;          ///< box size
    int maxiter;       ///< max number of iteration
    std::string symbol;  ///< symbol of target atom
    double lo;         ///< smallest length scale we need to resolve
    double thresh;     ///< truncation threshold
    double eprec;      ///< precision of smoothing parameter
    double delta;
    ///< step size to calc numerical derivative of residual
    int nio;          ///< number of I/O node
    bool nonlinear;   ///< if true do non linear optimization
    bool plot;         ///< if true plot vmo or umo
};

CalculationParameters()
: dconv(1e-6)
, L(50.0)
, maxiter(50)
, symbol("Li")
, lo(1e-10)
, thresh(1e-6)
, eprec(1e-4)

```

```

        , delta(0.1)
        , nio(1)
        , nonlinear(false)
        , plot(false)
    {}

void print_info () {
    print(" ** Parameter settings ** ");
    print("          target atom", symbol);
    print("          box size", L);
    print("          lo of mra", lo);
    print("          thresh of mra", thresh);
    print("          dconv", dconv);
    print("          max iter", maxiter);
    print("eprec for smoothed pot.", eprec);
    print("  delta for derivative", delta);
    print("  number of io node", nio);
    print("non linear optimization", nonlinear);
    print("          plot", plot);
}

void read_file (const std::string filename) {
    std::ifstream f(filename.c_str());
    std::string s;
    while (f >> s) {
        if (s == "atom") {
            f >> symbol;
        }
        else if (s == "conv") {
            f >> dconv;
        }
        else if (s == "maxiter") {
            f >> maxiter;
        }
    }
}

```

```

    }

else if (s == "L") {
    f >> L;
}

else if (s == "thresh") {
    f >> thresh;
}

else if (s == "delta") {
    f >> delta;
}

else if (s == "nio") {
    f >> nio;
}

else if (s == "nonlinear") {
    nonlinear = true;
}

else if (s == "plot") {
    plot = true;
}

else {
    throw "Unknown input parameter:" + s;
}

print_info();
}

template <typename Archive>
void serialize(Archive& ar) {
    ar & dconv & L & maxiter & symbol & lo
        & thresh & eprec & delta & nio;
}
};


```

```

struct Calculation {
    CalculationParameters param;
    CorePotentialManager corepot;
    bool spin_restricted;
    tensorT aeps, beps;
    tensorT aocc, bocc;
    vector<int> aset, bset;
    vecfuncT amo, bmo;
    vecfuncT vamo, vbmo;
    poperatorT coulop;
    unsigned int atn;           ///< atomic number of target atom
    unsigned int ncore;         ///< number of core orbitals
    unsigned int nalpha, nbeta;
    ///< number of valence orbitals alpha/beta
    double vtol;               ///< multiplication tolerance
    functionT mask;
}

Calculation (World & world)
{
    if (world.rank() == 0) {
        param.read_file("fitinput");
        std::set<unsigned int> atomset;
        atn = symbol_to_atomic_number(param.symbol);
        atomset.insert(atn);
        corepot.read_file("mcp", atomset, param.eprec);
        MADNESS_ASSERT(corepot.is_defined(atn));
    }
    world.gop.broadcast_serializable(param, 0);
    world.gop.broadcast_serializable(corepot, 0);
    FunctionDefaults<3>::set_cubic_cell(-param.L, param.L);
    atn = symbol_to_atomic_number(param.symbol);
    ncore = corepot.n_core_orb(atn);
    set_protocol(world, param.thresh);
}

```

```

load_mos(world, param.symbol);
print_info();
vamo = make_reference(world, amo, aocc);
if (!spin_restricted && nbeta)
    vbmo = make_reference(world, bmo, bocc);

// erase core vector
vecfuncT val_amo = vecfuncT(amo.begin()+ncore, amo.end());
amo = val_amo;
val_amo.clear();
if (!spin_restricted && nbeta) {
    vecfuncT val_bmo = vecfuncT(bmo.begin()+ncore, bmo.end());
    bmo = val_bmo;
    val_bmo.clear();
}
print("norm2 of references");
print(norm2(world, vamo));
if (!spin_restricted && nbeta)
    print(norm2(world, vbmo));
}

void print_info () {
    print("    atomic number :", atn);
    print("    number of core :", ncore);
    print("    nalpha, nbeta :", nalpha, nbeta);
    print("    spin_restricted :", spin_restricted);
    print("            aeps :", aeps);
    print("            aocc :", aocc);
    if (!spin_restricted && nbeta) {
        print("            beps :", beps);
        print("            bocc :", bocc);
    }
}

```

```

void set_protocol (World & world, double thresh)
{
    int k;
    if(thresh >= 1e-2)
        k = 4;
    else if(thresh >= 1e-4)
        k = 6;
    else if(thresh >= 1e-6)
        k = 8;
    else if(thresh >= 1e-8)
        k = 10;
    else
        k = 12;

    FunctionDefaults<3>::set_k(k);
    FunctionDefaults<3>::set_thresh(thresh);
    FunctionDefaults<3>::set_refine(true);
    FunctionDefaults<3>::set_initial_level(2);
    FunctionDefaults<3>::set_truncate_mode(1);
    FunctionDefaults<3>::set_autorefine(false);
    FunctionDefaults<3>::set_apply_randomize(false);
    FunctionDefaults<3>::set_project_randomize(false);
    GaussianConvolution1DCache<double>::map.clear();
    double safety = 0.1;
    vtol = FunctionDefaults<3>::get_thresh() * safety;
    coulop = poperatorT(CoulombOperatorPtr(world, param.lo, thresh));
    mask = functionT(
        factoryT(world)
            .f(mask3)
            .initial_level(4)
            .norefine());
    if(world.rank() == 0){

```

```

        print("\nSolving with thresh", thresh, "    k",
              FunctionDefaults<3>::get_k(), "    conv",
              std::max(thresh, param.dconv), "\n");
    }

}

void load_mos (World& world, std::string atom)
{
    const double thresh = FunctionDefaults<3>::get_thresh();
    const int k = FunctionDefaults<3>::get_k();
    unsigned int nmo;
    amo.clear(); bmo.clear();

    std::string refdir = "references/";
    std::string filename = refdir + atom;
    try {
        archive::ParallelInputArchive ar(world, filename.c_str());
        /*
           File format:

           bool spinrestricted --> if true only
               alpha orbitals are present

           unsigned int nmo_alpha;
           Tensor<double> aeps;
           Tensor<double> aocc;
           vector<int> aset;
           for i from 0 to nalpha-1:
               .   Function<double,3> amo[i]

           repeat for beta if !spinrestricted
    */
}

```

```

ar & spin_restricted;

ar & nmo;
nalpha = nmo - ncore;
nbeta = 0;
const double trantol = vtol / std::min(30.0, double(nalpha));
ar & aeps & aocc & aset;
amo.resize(nmo);
for (unsigned int i=0; i<amo.size(); i++) ar & amo[i];

if (amo[0].k() != k) {
    reconstruct(world,amo);
    for(unsigned int i = 0;i < amo.size();i++)
        amo[i] = madness::project(amo[i], k, thresh, false);
    world.gop.fence();
}
normalize(world, amo);
amo = transform(world, amo,
                Q3(matrix_inner(world, amo, amo)), trantol, true);
truncate(world, amo);
normalize(world, amo);

if (!spin_restricted) {
    ar & nmo;
    nbeta = nmo - ncore;
    ar & beps & bocc & bset;

    bmo.resize(nmo);
    for (unsigned int i=0; i<bmo.size(); i++) ar & bmo[i];

    if (bmo[0].k() != k) {
        reconstruct(world,bmo);

```

```

        for(unsigned int i = 0;i < bmo.size();i++)
            bmo[i] = madness::project(bmo[i], k, thresh, false);
        world.gop.fence();
    }

    normalize(world, bmo);
    bmo = transform(world, bmo,
                    Q3(matrix_inner(world, bmo, bmo)), trantol, true);
    truncate(world, bmo);
    normalize(world, bmo);
}
}

catch (const madness::MadnessException & e) {
    print(e);
    std::string err_msg("The 'mcpfit' requires ");
    err_msg += filename;
    err_msg += ".00000 (and additional numbers of nio)"
               "to make reference potential.\n"
               "It can be produced by atom SCF using moldft.";
    error(err_msg.c_str());
    throw e;
}

}

vecfuncT make_reference (World & world, vecfuncT & mo, tensorT & occ)
{
    int nv = mo.size() - ncore;
    vecfuncT vmo = zero_functions<double,3>(world, nv);
    compress(world, vmo);
    reconstruct(world, amo);
    norm_tree(world, amo);
    if (!spin_restricted && bmo.size()) {

```

```

    reconstruct(world, bmo);
    norm_tree(world, bmo);
}

// coulomb potential
START_TIMER(world);
vecfuncT psi_c(amo.begin(), amo.begin()+ncore);
vecfuncT vsq = square(world, psi_c);
compress(world, vsq);
functionT rho_c = factoryT(world);
rho_c.compress();
for (unsigned int i=0; i<vsq.size(); i++) {
    if (occ[i])
        rho_c.gaxpy(1.0, vsq[i], aocc[i], false);
}
world.gop.fence();
vsq.clear();
if (!spin_restricted && bmo.size()) {
    psi_c = vecfuncT(bmo.begin(), bmo.begin()+ncore);
    vsq = square(world, psi_c);
    compress(world, vsq);
    for (unsigned int i=0; i<vsq.size(); i++) {
        if (occ[i])
            rho_c.gaxpy(1.0, vsq[i], bocc[i], false);
    }
    world.gop.fence();
    vsq.clear();
}
else {
    rho_c *= 2.0;
}
rho_c.truncate();
functionT coul = apply(*coulop, rho_c);

```

```

// coulomb - Nc/r (long range term)
functionT nc_over_r = factoryT(world).functor(
    functorT(new NcOverR(2*ncore,
        smoothing_parameter(2*ncore, param.eprec))))
    .thresh(vtol).initial_level(4);
nc_over_r.reconstruct();
coul -= nc_over_r;
nc_over_r.clear();

vecfuncT valencemo(mo.begin()+ncore, mo.end());
gaxpy(world, 1.0, vmo, 1.0, mul_sparse(world, coul,
    valencemo, vtol));
truncate(world, vmo);
END_TIMER(world, "coulomb pot.");

// exchange potential
START_TIMER(world);
psi_c = vecfuncT(mo.begin(), mo.begin()+ncore);
vecfuncT psif;
for (unsigned int c=0; c<ncore; c++) {
    for (unsigned int i=ncore; i<mo.size(); i++) {
        psif.push_back(mul_sparse(mo[i], mo[c], vtol,
            false));
    }
}
world.gop.fence();
truncate(world, psif, vtol);
psif = apply(world, *coulop, psif);
truncate(world, psif, vtol);
reconstruct(world, psif);
norm_tree(world, psif);
for (unsigned int c=0; c<ncore; c++) {

```

```

        for (unsigned int i=ncore; i<mo.size(); i++) {
            functionT psipsif =
                mul_sparse(psif[c*nv+i-ncore], mo[c], vtol, false);
            world.gop.fence();
            psipsif.compress();
            vmo[i-ncore].gaxpy(1.0, psipsif, -1.0, false);
        }
    }

    truncate(world, vmo);
    END_TIMER(world, "exchange pot.");

    // shift operator
    START_TIMER(world);
    unsigned int nshell = ncore;
    for (unsigned int c = 0; c < nshell; ++c) {
        unsigned int l = corepot.get_core_l(atn, c);
        unsigned int max_m = (l+1)*(l+2)/2;
        nshell -= max_m - 1;
        double bc = corepot.get_core_bc(atn, c);
        for (unsigned int m = 0; m < max_m; ++m) {
            functionT core =
                factoryT(world)
                    .functor(functorT(
                        new CoreOrbitalFunctor(corepot, atn, c, m)));
            tensorT overlap = inner(world, core, mo);
            print("for m=", m, " core orbital of ", c, "'th shell");
            print("  <core|mo> = ", overlap);
            overlap *= bc;
            for (unsigned int i = ncore; i < mo.size(); ++i) {
                vmo[i-ncore] -= overlap[i] * core;
            }
        }
    }
}

```

```

        truncate(world, vmo);
        END_TIMER(world, "shift operator");

    return vmo;
}

functionT project_potential_basis (World & world,
                                    CorePotential & cp, int i)
{
    int level = 4;
    if (cp.alpha[i] > 100) level += 2;
    functionT u1;
    double norm = 0.0;
    while (norm < 1e-3) {
        u1 = functionT(factoryT(world)
                        .functor(functorT(
                            new PotentialBasisFunctor(cp.n[i], cp.alpha[i], cp.rcut)))
                        .initial_level(++level).truncate_on_project().noautorefine());
        print("  norm2 of potential basis", i, ":", u1.norm2());
        norm = u1.norm2();
    }
    u1.truncate();

    return u1;
}

vecfunctT make_Upsi (World & world, CorePotential & cp, vecfunct & mo)
{
    START_TIMER(world);
    functionT umcp =
        factoryT(world)
            .functor(functorT(new CorePotentialFunctor(cp)))
            .thresh(FunctionDefaults<3>::get_thresh())

```

```

        .initial_level(5);

umcp.truncate();
umcp.reconstruct();
vecfuncT umo = mul_sparse(world, umcp, mo, vtol);
umcp.clear();
truncate(world, umo);
world.gop.fence();
END_TIMER(world, "make Upsi");
return umo;
}

void plot_z (const char* filename, const functionT & f)
{
    const int npt = 2049;
    Vector<double,3> lo(vector<double>(3,0.0)),
        hi(vector<double>(3,0.0));
    hi[2] = 5.0;
    f.reconstruct();
    plot_line(filename, npt, lo, hi, f);
}

void plot_p (const char* filename, const vecfuncT& f)
{
    const int npt = 2049;
    Vector<double,3> lo(vector<double>(3,0.0)),
        hi(vector<double>(3,0.0));
    hi[2] = 5.0;
    Vector<double,3> h = (hi - lo)*(1.0/(npt-1));

    double sum = 0.0;
    for (int i=0; i<3; i++) sum += h[i]*h[i];
    sum = sqrt(sum);
}

```

```

World& world = f[0].world();
reconstruct(world, f);
if (world.rank() == 0) {
    FILE* file = fopen(filename, "w");
    for (int i=0; i<npt; i++) {
        coordT r = lo + h*double(i);
        fprintf(file, "%.14e ", i*sum);
        plot_line_print_value(file, f[0].eval(r));
        double f0 = f[0].eval(r);
        double f1 = f[1].eval(r);
        double f2 = f[2].eval(r);
        double v = f0*f0+f1*f1+f2*f2;
        fprintf(file, " %.14e", v);
        fprintf(file, "\n");
    }
    fclose(file);
}
world.gop.fence();
}

CorePotential calc_optimal_coeffs (World & world,
                                   CorePotential & cp)
{
    long npParam = cp.alpha.size() - 1;
    vecfuncT pbs(npParam);
    functionT rf =
        factoryT(world)
            .functor(functorT(new RadiusFunctor()));
    rf.reconstruct();

    print("alpha: ", cp.alpha);
    print("n: ", cp.n);
    for (unsigned int i=1; i<cp.alpha.size(); i++) {

```

```

printf("making pb for alpha[%d]=%e\n", i, cp.alpha[i]);
pbs[i-1] = project_potential_basis(world, cp, i);
pbs[i-1].reconstruct();
pbs[i-1] = rf * pbs[i-1];
pbs[i-1].truncate();
}

tensorT S(nparam, nparam);
tensorT b(nparam);
print("making S, b");
print("nparam=", nparam, "nalpha,nbeta=", nalpha, nbeta);
for (unsigned int j=0; j<nalpha; j++) {
    print("for alpha elec", j);
    vecfuncT pb_amo = mul_sparse(world, amo[j], pbs, vtol);
    truncate(world, pb_amo);
    S += matrix_inner(world, pb_amo, pb_amo);
    b += inner(world, rf*vamo[j], pb_amo);
    pb_amo.clear();
    world.gop.fence();
}
if (!spin_restricted && nbeta) {
    for (unsigned int j=0; j<nbeta; j++) {
        print("for beta elec", j);
        vecfuncT pb_bmo = mul_sparse(world, bmo[j], pbs, vtol);
        truncate(world, pb_bmo);
        S += matrix_inner(world, pb_bmo, pb_bmo);
        b += inner(world, rf*vbmo[j], pb_bmo);
        pb_bmo.clear();
        world.gop.fence();
    }
}
print("S,b=\n", S, b);

```

```

tensorT c, s, sumsq;
long rank;
if (world.rank() == 0)
    gelss(S, b, -1, c, s, rank, sumsq);
    // solve linear equation
world.gop.broadcast_serializable(c, 0);
world.gop.broadcast_serializable(s, 0);
world.gop.broadcast_serializable(sumsq, 0);
world.gop.broadcast_serializable(rank, 0);
print("s,sumsq=", s, sumsq);
print("rank=", rank);
print("result potential:");
printf("      %d      %d      %.6e      %.12e\n",
       cp.l[0], cp.n[0], cp.alpha[0], cp.A[0]);
for (unsigned int i=0; i<cp.alpha.size()-1; ++i) {
    printf("      %d      %d      %.6e      %.12e\n",
           cp.l[i+1], cp.n[i+1], cp.alpha[i+1], c[i]);
}
CorePotential result = cp;
vector<double> A = tensor2vec(c);
A.insert(A.begin(), cp.A[0]);
result.A = A;

return result;
}

double compute_residuals (World & world, CorePotential & cp,
                        vecfuncT & mo, vecfuncT & vmo)
{
//CorePotential cp_opt = calc_optimal_coeffs(world, cp);
//cp = cp_opt;
vecfuncT umo = make_Upsi(world, cp, mo);

```

```

for (unsigned int i=0; i<vmo.size(); i++) {
    print("norm(vmo)", vmo[i].norm2());
    print("norm(umo)", umo[i].norm2());
    //double normratio = vmo[i].norm2() / umo[i].norm2();
    //umo[i] *= normratio;
}
vecfuncT rv = sub(world, vmo, umo);
truncate(world, rv);
functionT rf =
    factoryT(world)
        .functor(functorT(new RadiusFunctor()));
vecfuncT err = mul_sparse(world, rf, rv, vtol);
if (param.plot) {
    static unsigned int num=0;
    for (unsigned int i=0; i<rv.size(); i++) {
        char fn[256];
        sprintf(fn, "umo%02d.txt", num);
        plot_z(fn, umo[i]);
        sprintf(fn, "vmo%02d.txt", num);
        plot_z(fn, vmo[i]);
        sprintf(fn, "vmo-umo%02d.txt", num);
        plot_z(fn, rv[i]);
        sprintf(fn, "rvmo-rumo%02d.txt", num);
        plot_z(fn, err[i]);
        num++;
    }
}
umo.clear();
rv.clear();
world.gop.fence();
//tensorT rnorm = vec2tensor(norm2s(world, err));
//    // sqrt of norm2's
tensorT rnorm(static_cast<long>(err.size()));

```

```

for (unsigned int i=0; i<err.size(); i++)
    rnorm[i] = err[i].norm2();
print("residual norm:", rnorm);

double r = 0.0;
for (int i=0; i<rnorm.dim(0); i++) {
    r += rnorm[i];
}
return r / rnorm.dim(0);
}

tensorT calc_deriv (World & world, CorePotential & cp, double r)
{
    size_t npParam = cp.alpha.size();
    tensorT deps(static_cast<long>(npParam));
    for (unsigned int i=1; i<npParam; i++) {
        double a = cp.alpha[i];
        double da = a * param.delta;
        CorePotential cp_mod = cp;
        cp_mod.alpha[i] = a+da;
        cp_mod = calc_optimal_coeffs(world, cp_mod);
        double rmod = compute_residuals(world, cp_mod, amo, vamo);
        if (!spin_restricted && nbeta)
            rmod += compute_residuals(world, cp_mod, bmo, vbmo);
        deps(i) = (rmod - r) / da;
    }

    return deps;
}

CorePotential update (World & world, CorePotential & cp,
                     tensorT & deriv, double & r)
{

```

```

const double tau = 0.5;

CorePotential cp_mod = cp;
double normd = deriv.normf();
double step = tau / normd;
if (step > 1) step = 1;
double rmod;
while (1) {
    //double rule_value = r - step * normd; // armijo
    double rule_value = r; // simple descent rule
    if (false && rule_value < 0) {
        step *= tau;
        continue;
    }
    if (step < param.dconv) throw "searching vector failed";
    printf("step = %15e, Rule value =%15e\n", step, rule_value);
    vector<double> amod =
        tensor2vec(vec2tensor(cp.alpha) - step * deriv);
    cp_mod.alpha = amod;
    rmod = compute_residuals(world, cp_mod, amo, vamo);
    if (!spin_restricted && nbeta)
        rmod += compute_residuals(world, cp_mod, bmo, vbmo);
    printf("r(amod) = %15e\n\n", rmod);

    // wolfe curvature constraint
    //tensorT d2 = calc_deriv(world, cp_mod, rmod);
    //double lhs = dot_product(d2, -deriv);
    //double rhs = -normd;
    //print("deriv");
    //print(d2);
    //printf("lhs:%10e, rhs:%10e\n\n", lhs, rhs);
    printf("rmod:%10e, rule:%10e\n", rmod, rule_value);
}

```

```

        if (rmod < rule_value) break; // found
        //if (rmod < rule_value && lhs > rhs) break; // found

        step *= tau;
    }

    r = rmod;

    return cp_mod;
}
};

struct CoreFittingTarget : public OptimizationTargetInterface {
    World & world;
    Calculation & calc;
    CorePotential & cp;
    mutable double xsq;
    mutable double r;
    mutable tensorT lastx;

    CoreFittingTarget (World & world, Calculation & calc,
                       CorePotential & cp)
        : world(world), calc(calc), cp(cp), xsq(0.0), r(100.0)
    {}

    bool provides_gradient() const {return true;}

    double value (const tensorT & x) {
        double xsq = x.sumsq();
        if (fabs(xsq - this->xsq) < 1e-10) {
            print("target: using current value (delta xsq =",
                  xsq - this->xsq, ")");
        }
        return this->r;
    }
};

```

```

    }

    this->xsq = xsq;
    lastx = copy(x);
    print("target: calc for alpha = ", x);
    CorePotential cp_mod = cp;
    cp_mod.alpha = tensor2vec(x);
    cp_mod = calc.calc_optimal_coeffs(world, cp_mod);
    double r = calc.compute_residuals(world,
        cp_mod, calc.amo, calc.vamo);
    if (!calc.spin_restricted && calc.nbeta)
        r += calc.compute_residuals(world,
            cp_mod, calc.bmo, calc.vbmo);
    world.gop.fence();
    print("target: value for ", x, " = ", r);
    this->r = r;
    return r;
}

tensorT gradient(const tensorT& x) {
    CorePotential cp_mod = cp;
    cp_mod.alpha = tensor2vec(x);
    double r = value(x);
    tensorT g = calc.calc_deriv(world, cp_mod, r);
    world.gop.fence();
    print("target: gradient for ", x, " = ", g);
    return g;
}
};

class MySteepestDescent : public OptimizerInterface {
    std::shared_ptr<OptimizationTargetInterface> target;
    const double tol;
    const double value_precision;

```

```

// Numerical precision of value
const double gradient_precision;

// Numerical precision of each element of residual
double f;
double gnorm;

public:
MySteepestDescent(
    const std::shared_ptr<OptimizationTargetInterface>& target,
    double tol = 1e-6,
    double value_precision = 1e-12,
    double gradient_precision = 1e-12)
: target(target)
, tol(tol)
, value_precision(value_precision)
, gradient_precision(gradient_precision)
, gnorm(tol*1e16)
{
    if (!target->provides_gradient())
        throw "Steepest descent requires the gradient";
}

bool check_positive (tensorT & x)
{
    for (TensorIterator<double> e = x.unary_iterator();
         e != x.end();
         ++e) {
        if (*e < 0.0) return false;
    }

    return true;
}

```

```

bool optimize(Tensor<double>& x)
{
    double step = 10.0;
    double fnew;
    Tensor<double> g;
    target->value_and_gradient(x,f,g);
    gnorm = g.normf();
    for (int i=0; i<100; i++) {
        while (1) {
            Tensor<double> gnew;
            x.gaxpy(1.0, g, -step);
            if (check_positive(x)) {
                target->value_and_gradient(x,fnew,gnew);
                if (fnew < f) {
                    f = fnew;
                    g = gnew;
                    break;
                }
            }
            x.gaxpy(1.0, g, step);
            step *= 0.5;
            print("reducing step size",
                  f, fnew, "(", f - fnew, ")",
                  step);
        }
        Tensor<double> g = target->gradient(x);
        gnorm = g.normf();
        print("iteration",i,"value",f,"gradient",gnorm);
        if (converged()) break;
    }
    return converged();
}

bool converged() const

```

```

    {
        return gnorm < tol;
    }

    double gradient_norm() const
    {
        return gnorm;
    }

    double value() const
    {
        return f;
    }

    virtual ~MySteepestDescent(){}
};

int main (int argc, char **argv) {
    initialize(argc, argv);

    {

        World world(SafeMPI::COMM_WORLD);

        try {
            startup(world, argc, argv);
            FunctionDefaults<3>::set_pmap(pmapT(new LevelPmap(world)));
            Calculation calc(world);
            CorePotential cp = calc.corepot.get_potential(calc.atn);

            //double r1 = calc.compute_residuals(world,
            //        cp, calc.amo, calc.vamo);
            //if (!calc.spin_restricted)
            //    r1 += calc.compute_residuals(world,

```

```

//          cp, calc.bmo, calc.vbmo);
//print("r1=", r1);

if (!calc.param.nonlinear) {
    print("==== start linear optimization");
    cp = calc.calc_optimal_coeffs(world, cp);
    print("==== linear optimal coeffs prepared");
    double r2 = calc.compute_residuals(world,
                                         cp, calc.amo, calc.vamo);
    if (!calc.spin_restricted && calc.nbeta)
        r2 += calc.compute_residuals(world,
                                     cp, calc.bmo, calc.vbmo);
    //print("r1=", r1);
    print("r2=", r2);
}

else {
    print("==== start non linear optimization");
    //double tol = 1e-4; // tolerance
    //double prec = calc.param.eprec; // precision
    //double gprec = calc.param.eprec; // gradient precision
    std::shared_ptr<CoreFittingTarget>
        target_p(new CoreFittingTarget(world, calc, cp));
    //QuasiNewton optimizer(target_p, tol, prec, gprec);
    MySteepestDescent optimizer(target_p);
    tensorT x = vec2tensor(cp.alpha);
    optimizer.optimize(x);
    print("last x = ", target_p->lastx);
    cp.alpha = tensor2vec(target_p->lastx);
    double r = calc.compute_residuals(world,
                                      cp, calc.amo, calc.vamo);
    if (!calc.spin_restricted && calc.nbeta)
        r += calc.compute_residuals(world,
                                    cp, calc.bmo, calc.vbmo);
}

```

```

        print("last r=", r);
    }
}

catch (const SafeMPI::Exception& e) {
    print(e);
    error("caught an MPI exception");
}

catch (const madness::MadnessException& e) {
    print(e);
    error("caught a MADNESS exception");
}

catch (const madness::TensorException& e) {
    print(e);
    error("caught a Tensor exception");
}

catch (char* s) {
    print(s);
    error("caught a string exception");
}

catch (const char* s) {
    print(s);
    error("caught a string exception");
}

catch (const std::string& s) {
    print(s);
    error("caught a string (class) exception");
}

catch (std::bad_alloc & b) {
    error("operator new failed");
}

catch (const std::exception& e) {
    print(e.what());
    error("caught an STL exception");
}

```

```
}

catch (...) {
    error("caught unhandled exception");
}

// Nearly all memory will be freed at this point
world.gop.fence();
world.gop.fence();
ThreadPool::end();
print_stats(world);

}

finalize();

return 0;
}
```

付録 C

本研究で求めた MCP パラメータを掲載する。データは XML 形式である。

ルート要素は<core_info>であり、各原子のパラメータ定義を行う<atom>要素の羅列を子として持つ。

<atom>要素は、局在ポテンシャルを表す<potential>要素、及び内殻軌道群を表す<core>要素を持つ。

<potential>要素では、式 (5.15) における各項を 1 行で表す。1 行は空白区切りで 4 要素からなり、第 1 要素は軌道角運動量量子数、第 2 要素は n_k 、第 3 要素は α_k 、第 4 要素は A_k である。この内 A_k 以外のパラメータについては [18] に準じた。

<core>要素は、内殻軌道を表す<orbital>要素を、内殻軌道とみなすものについて収容する。

<orbital>要素は、1 行で規格化された Gaussian 型 primitive function を示している。1 行は空白区切りで 2 要素からなり、第 1 要素は軌道指数、第 2 要素は係数を示している。ほとんどのパラメータは [18] に準じたが、Na については cc-pvqz 基底による 1 原子 Hartree-Fock-SCF の解を用いた。

```

<?xml version="1.0" encoding="UTF-8" ?>
<core_info>
    <atom id="Li" symbol="Li" atomic_number="3">
        <potential>
            0      1      0.000000e+00      2.000000e+00
            0      1      2.718422e+00      -1.404070e+00
            0      1      1.061906e+00      -2.040843e-01
        </potential>
        <core num="1">
            <orbital type="0" bc="4.9554840">
                7.105793e+02      2.009533e-03
                1.034368e+02      1.482329e-02
                2.443300e+01      6.640480e-02
                7.308967e+00      1.990801e-01
                2.506894e+00      3.819108e-01
                9.438003e-01      3.884663e-01
                3.709710e-01      1.099967e-01
            </orbital>
        </core>
    </atom>
    <atom id="Be" symbol="Be" atomic_number="4">
        <potential>
            0      1      0.000000e+00      2.000000e+00
            0      1      5.188992e+00      -1.397212e+00
            0      1      2.023980e+00      -2.122203e-01
        </potential>
        <core num="1">
            <orbital type="0" bc="9.4653380">
                1.311616e+03      1.969894e-03
                1.913861e+02      1.451971e-02
                4.527883e+01      6.534285e-02
                1.359732e+01      1.982694e-01
                4.719701e+00      3.835307e-01
            </orbital>
        </core>
    </atom>

```

```

    1.817344e+00      3.845043e-01
    7.340121e-01      1.096769e-01
    </orbital>
  </core>
</atom>
<atom id="B" symbol="B" atomic_number="5">
  <potential>
    0      1      0.000000e+00      2.000000e+00
    0      1      1.603560e+01      -1.518700e+00
    0      1      2.322329e+00      -2.808786e-01
  </potential>
  <core num="1">
    <orbital type="0" bc="15.390670">
      2.787867e+02      2.477011e-02
      3.826330e+01      1.678633e-01
      8.576375e+00      4.977570e-01
      2.360978e+00      4.542732e-01
      2.929847e-01      8.554471e-03
      9.375142e-02      6.986010e-03
    </orbital>
  </core>
</atom>
<atom id="C" symbol="C" atomic_number="6">
  <potential>
    0      1      0.000000e+00      2.000000e+00
    0      1      1.864605e+01      -1.582358e+00
    0      1      3.512100e+00      -2.257663e-01
  </potential>
  <core num="1">
    <orbital type="0" bc="22.651036">
      4.090699e+02      2.446620e-02
      5.616343e+01      1.667309e-01
      1.263216e+01      4.985016e-01

```

```

    3.495336e+00      4.533167e-01
    4.607115e-01      9.069870e-03
    1.436420e-01      7.456826e-03
    </orbital>
</core>
</atom>

<atom id="N" symbol="N" atomic_number="7">
    <potential>
        0      1      0.000000e+00      2.000000e+00
        0      1      2.763930e+01      -1.595784e+00
        0      1      5.022100e+00      -2.624201e-01
    </potential>
    <core num="1">
        <orbital type="0" bc="31.258000">
            5.635094e+02      2.422667e-02
            7.738753e+01      1.657101e-01
            1.744855e+01      4.979887e-01
            4.843508e+00      4.511053e-01
            7.820957e+00      1.334650e-03
            6.633018e-01      9.410938e-03
            2.029459e-01      7.779541e-03
        </orbital>
    </core>
</atom>

<atom id="O" symbol="O" atomic_number="8">
    <potential>
        0      1      0.000000e+00      2.000000e+00
        0      1      4.064864e+01      -1.695591e+00
        0      1      5.175230e+00      -2.183506e-01
    </potential>
    <core num="1">
        <orbital type="0" bc="41.337314">
            7.432411e+02      2.413306e-02

```

```

    1.020870e+02      1.655533e-01
    2.305897e+01      4.996164e-01
    6.416186e+00      4.517950e-01
    9.065320e-01      9.761350e-03
    2.740610e-01      7.992775e-03
    </orbital>
</core>
</atom>

<atom id="F" symbol="F" atomic_number="9">
    <potential>
        0      1      0.000000e+00      2.000000e+00
        0      1      5.202781e+01      -1.715359e+00
        0      1      6.206354e+00      -2.299835e-01
    </potential>
    <core num="1">
        <orbital type="0" bc="52.765500">
            9.476157e+02      2.401760e-02
            1.301841e+02      1.651131e-01
            2.944885e+01      4.998040e-01
            8.208941e+00      4.514980e-01
            1.184390e+00      1.005990e-02
            3.545967e-01      8.202500e-03
        </orbital>
    </core>
</atom>

<atom id="Ne" symbol="Ne" atomic_number="10">
    <potential>
        0      1      0.000000e+00      2.000000e+00
        0      1      5.407668e+01      -1.635342e+00
        0      1      7.612973e+00      -2.108811e-01
    </potential>
    <core num="1">
        <orbital type="0" bc="65.544850">

```

```

    1.175870e+03    2.394744e-02
    1.615680e+02    1.648909e-01
    3.659069e+01    5.001582e-01
    1.021284e+01    4.509687e-01
    1.497330e+00    1.025659e-02
    4.446700e-01    8.352764e-03
  </orbital>
</core>
</atom>
<atom id="Na" symbol="Na" atomic_number="11">
  <potential>
    0    1    0.000000e+00    4.000000e+00
    0    1    3.050656e+02   -1.743249e+00
    0    1    1.464675e+01   -1.513124e+00
    0    1    2.136867e+00   -8.039920e-01
    0    2    4.718427e+02   -1.590759e+02
    0    2    2.052377e+02   -4.548875e+00
    0    2    6.894609e-01   -4.851126e-01
  </potential>
  <core num="2">
    <orbital type="0" bc="80.957000">
      1.437387e+03    2.371636e-02
      1.975242e+02    1.636571e-01
      4.477766e+01    4.987114e-01
      1.251831e+01    4.529699e-01
      1.910040e+00    1.062412e-02
      6.071710e-01    8.547746e-03
    </orbital>
    <orbital type="0" bc="5.5940000">
      1.437387e+03    6.294799e-03
      1.975242e+02    4.343787e-02
      4.477766e+01    1.323680e-01
      1.251831e+01    1.202273e-01

```

```

    2.156724e+01      8.952974e-02
  </orbital>
</core>
</atom>

<atom id="Mg" symbol="Mg" atomic_number="12">
  <potential>
    0      1      0.000000e+00      4.000000e+00
    0      1      3.371815e+02     -1.693663e+00
    0      1      1.582292e+01     -1.422796e+00
    0      1      2.309331e+00     -7.518502e-01
    0      2      5.552475e+02     -1.581608e+02
    0      2      2.797939e+02     -4.956377e+00
    0      2      7.681323e-01     -4.550130e-01
  </potential>
  <core num="2">
    <orbital type="0" bc="98.063470">
      1.726576e+03      2.350363e-02
      2.373122e+02      1.624787e-01
      5.384661e+01      4.972212e-01
      1.507884e+01      4.549977e-01
      2.375296e+00      1.102567e-02
      7.937981e-01      8.641283e-03
    </orbital>
    <orbital type="0" bc="7.5354400">
      1.726576e+03      6.466155e-03
      2.373122e+02      4.470000e-02
      5.384661e+01      1.367920e-01
      1.507884e+01      1.251758e-01
      2.586516e+01      9.284621e-02
    </orbital>
  </core>
</atom>
</core_info>
```