開発

1　開発者

- **石川 岳志**

  岐阜大学 人獣感染防御研究センター 計算創薬研究分野

管理

- プログラムの管理
- テキストの管理

プログラムの作成

- FMO 法のベースシステム
- 分子積分
- RHF エネルギー
- MP2 エネルギー
- 軌道局在化
- 局在化 MP2 エネルギー
- RI–MP2 エネルギー
- RHF 動的分極率計算
- RHF エネルギー勾配および部分エネルギー勾配
- 構造最適化のベースシステム

テキストの作成

- マニュアルの作成

2　開発履歴

2011.01

- ソースコードの公開開始

2010.10

- 部分エネルギー勾配
- 構造最適化ルーチン

2010.04

- バイナリの公開開始

2010.02

- RHF（動的分極率）

2009.04

- RI–MP2（エネルギー）

2008.10

- RHF（エネルギー勾配）

2008.07

- FMO 計算のベースシステム
- 分子積分
- RHF（エネルギー）
- MP2（エネルギー）
- 軌道局在化
- 局在化 MP2（エネルギー）

3 論文

PAICS を使用した場合、以下の論文を引用すること。

- **Teoretical study of the prion protein based on the fragment molecular orbital method**, T. Ishikawa, T. Ishikura, and K. Kuwata, *J. Comput. Chem., 30 (2009) 2594–2601*

PAICS の開発に関連する論文。

- **Fragment interaction analysis based on local MP2**, T. Ishiskawa, Y. Mochizuki, S. Amarai, T. Nakano, H. Tokiwa, S. Tanaka, and K. Tanaka, *Theor. Chem. Acc., 118 (2007) 937-945*

- **An application of fragment interaction analysis based on local MP2**, T. Ishikawa, Y. Mochizuki, S. Amari, T. Nakano, S. Tanaka, and K. Tanaka, *Chem. Phys. Lett., 463 (2008) 189-194*

- **Teoretical study of the prion protein based on the fragment molecular orbital method**, T. Ishikawa, T. Ishikura, and K. Kuwata, *J. Comput. Chem., 30 (2009) 2594–2601*

- **Fragment molecular orbital calculation using the RI–MP2 method**, T. Ishikawa and K. Kuwata, *Chem. Phys. Lett., 474 (2009) 195–198*

- **Acceleration of monomer self-consistent charge process in fragment molecular orbital method**, T. Ishikawa and K. Kuwata, *C.B.I.J., 10 (2010) 24–31*

- **Partial energy gradient based on the fragment molecular orbital method: application to geometry optimization**, T. Ishikawa, N. Yamamoto, and K. Kuwata, *Chem. Phys. Lett., 500 (2010) 149-154*

コンパイルと計算の実行

1 コンパイル

1.1 ディレクトリ構成とファイル

配布物（paics.zip）のファイル構成は以下の通り。

- Makefile_paics

  プログラムをコンパイルするための Makefile。基本的にこのファイルは修正せず、make.inc を修正する。

- make.inc

  Makefile にインクルードされるファイル。ユーザーの環境に合わせて修正する。

- make.sh

  プログラムをコンパイルするためのスクリプト（bash）。

- clean.sh

  コンパイル前の状態に戻すスクリプト（bash）。

- main.c

  PAICS のメインプログラム。

- paics.run.sh

  MPICH でジョブを流すための基本スクリプトの例。ユーザーの環境に合うように修正する。

- paics.run.lsf.sh

  LSF（ジョブ管理システム）経由でででジョブを流すためのスクリプトの例。ユーザーの環境に合うように修正する。

- man /

  マニュアル（PDF）が格納されている。以下のファイルが存在する（これらは、***PAICS*** のホームページからも取得可能）。

development.pdf	***PAICS*** の開発
compile–and–execute.pdf	コンパイルと計算の実行（このファイル）
input.pdf	入力ファイルの記述
theory.pdf	理論との対応
variable.pdf	ソースコードのマニュアル（グローバル変数）
function.pdf	ソースコードのマニュアル（関数）

- src /

  ソースコードが格納されているディレクトリ。以下のような構成になっている。

include /	ヘッダーファイル
parallel_control /	並列化制御に関連するソースコード
memoryl_control /	メモリ制御に関連するソースコード
paics /	*PAICS* 全般に関するソースコード
input /	入力に関連するソースコード
output /	出力に関連するソースコード
fmt /	誤差関数に関連するソースコード
oneint /	1 電子積分に関連するソースコード
oneint_grad /	1 電子積分の微分に関連するソースコード
eri /	電子反発積分に関連するソースコード
eri_grad /	電子反発積分の微分に関連するソースコード（未公開）
esp /	環境静電ポテンシャルの計算に関連するソースコード
projection /	射影演算子に関連するソースコード
fragment /	フラグメント全般に関するソースコード
monomer_scc /	モノマー SCC 計算に関連するソースコード
monomer /	モノマー計算に関連するソースコード
dimer_es /	ダイマー ES 計算に関連するソースコード
dimer /	ダイマー計算に関連するソースコード
rhf /	RHF 計算に関連するソースコード
cmp2 /	MP2 計算に関連するソースコード
ri_cmp2 /	RI–MP2 計算に関連するソースコード
localize /	軌道局在化計算に関連するソースコード
lmp2 /	局在化 MP2 計算に関連するソースコード
pol /	分極率計算に関連するソースコード（未公開）
geom_opt /	構造最適化に関連するソースコード（未公開）
abmd /	分子動力学計算に関連するソースコード（未公開）

- basis /

  基底関数のデータファイルが格納されているディレクトリ。以下のファイルが存在する。

user.dat	ユーザー定義の基底関数データ
sto3g.dat	STO-3G の基底関数データ
631g.dat	6-31G の基底関数データ
631gdp.dat	6-31G**の基底関数データ
cc-pVDZ.dat	cc-pVDZ の基底関数データ

cc-pVTZ.dat	cc-pVTZ の基底関数データ
cc-pVQZ.dat	cc-pVQZ の基底関数データ
cc-pVDZso.dat	cc-pVDZ（ segmented–opt ）の基底関数データ
cc-pVTZso.dat	cc-pVTZ（ segmented–opt ）の基底関数データ
cc-pVDZri.dat	RI 近似に使われる cc-pVDZ 用の補助基底のデータ
cc-pVTZri.dat	RI 近似に使われる cc-pVTZ 用の補助基底のデータ
cc-pVQZri.dat	RI 近似に使われる cc-pVQZ 用の補助基底のデータ

- sample /

  入力ファイルのサンプルとそれらの実行結果が格納されているディレクトリ。以下のファイルが存在する。

h2o-4-ccpvdz.inp	H_2O の 4 量体の非 FMO 計算
h2o-4-ccpvdz.np1.out	上の計算の実行結果
fmo-h2o-4-ccpvdz.inp	H_2O の 4 量体の FMO 計算
fmo-h2o-4-ccpvdz.np1.out	上の計算の実行結果
c12h26-ccpvdz.inp	$C_{12}H_{26}$ の非 FMO 計算
c12h26-ccpvdz.np1.out	上の計算の実行結果
fmo-c12h26-ccpvdz.inp	$C_{12}H_{26}$ の FMO 計算
fmo-c12h26-ccpvdz.np1.out	上の計算の実行結果
gly5-ccpvdz.inp	GLY_5 の非 FMO 計算
gly5-ccpvdz.np1.out	上の計算の実行結果
fmo-gly5-ccpvdz.inp	GLY_5 の FMO 計算
fmo-gly5-ccpvdz.np1.out	上の計算の実行結果
fmo-prion-ccpvdz.inp	プリオンの FMO 計算
fmo-prion-ccpvdz.np8.out	上の計算の実行結果
fmo-prion-gn8-ccpvdz.inp	プリオンと GN8 の FMO 計算
fmo-prion-gn8-ccpvdz.np8.out	上の計算の実行結果

1.2 コンパイラとライブラリ

PAICS は、MPI による並列プログラムなので、コンパイルには、MPI の環境が必要となる。また、LAPACK を用いているので、関連ライブラリをリンクできるシステムが要求される。以下、このような環境が、既に用意されているものとする。

1.3 コンパイル

1. 「make.inc」ファイルの修正

「make.inc」を各自のシステムに合わせて修正する。

- ROOT_DIR

  PAICS のルートディレクトリを絶対パスで指定する。

- CC

  MPI コンパイラ（C 言語）を指定する。

- LIB

  必要なライブラリ（LAPACK 関連）を指定する。

- PAICS_INCDIR

  インクルードファイルが格納されているディレクトリを指定する。通常は、

  ```
  PAICS_INCDIR = ${ROOT_DIR}/src/include
  ```

  と指定されていればよい。

- CFLAGS

  コンパイル時のフラグを指定する。通常は、

  ```
  CFLAGS = -c -O3 -I{PAICS_INCDIR}
  ```

  と指定されていればよい。

- LFLAGS

  リンク時のフラグを指定する。システムおよびコンパイラに依存するが、通常は特に指定する必要は無い。

2. 「make.sh」を起動する

ルートディレクトリにある「make.sh」を起動する。成功すれば、「main.exe」が作成される。分子積分ルーチンのコンパイルにはかなり時間が掛かるので注意すること。配布物を解凍した段階では、make.sh に実行許可が与えられていない可能性があるので、その場合、以下のコマンドで実行許可を与えて、make.sh を起動する。

```
% chmod 755 make.sh
```

2　計算の実行

2.1　入力ファイルの作成

PAICSで計算を行うには、計算手法、メモリ、CPU 数、原子座標、フラグメント定義などを記述した入力ファイル（テキスト形式）を作成する必要がある。生体分子の場合、原子の数は数千、フラグメントの数は数百となり、直接入力ファイルを作成するのは困難である。そこで、***PAICS***では、入力ファイルの作成を支援するプログラム「***PaicsView***」を提供している。

2.2　JOB の投入

以下のことに注意する。

- 環境変数 PAICS_ROOT に ***PAICS*** のルートディレクトリの絶対パスをセットしておく。この環境変数の値は実行中に取得され、基底関数のデータファイルを読む際に使われる。
- 入力ファイルを引数として指定し、main.exe を MPI を通じて起動する。第一引数は実行中に取得され、MPI ランク 0 の CPU がこのファイルを開き、入力ファイルに記述された情報を取得する。

PAICSを起動するスクリプトの例を**図 1** に示す。

2.3　計算結果

PAICSは、標準出力に計算結果を書き出すので、リダイレクションで計算結果をファイルに保存する。計算が終わったら、WARNING が出ていないかを必ず確認する。

図 1: ***PAICS*** を起動するためのスクリプトの例

< mpirun を直接起動する場合（ paics.run.sh ）>

```
#!/bin/bash

export PAICS_ROOT=/home/ishi/program/paics
INP=$1
NCPU=$2
DIR=‘pwd‘

mpirun -np $NCPU $PAICS_ROOT/main.exe $DIR/$INP
```

以上のスクリプトを

```
% paics.run.sh [ 入力ファイル名 ] [ CPU 数 ] >& [ 出力ファイル名 ] &
```

のように実行する。

< lsf を通じて起動する場合（ paics.run.lsf.sh ）>

```
#!/bin/bash

export PAICS_ROOT=/home/ishi/paics/paics-20080703-2
DIR=‘pwd‘
BSUB_DIR=$DIR

INP_FILE=$1
OUT_FILE=$2
NCPU=$3

rm -f $BSUB_DIR/bsub.log
rm -f $BSUB_DIR/bsub.out

bsub -o $BSUB_DIR/bsub.out -e $BSUB_DIR/bsub.log -n $NCPU \
   "mpijob mpirun $PAICS_ROOT/main.exe $DIR/$INP_FILE >& $DIR/$OUT_FILE"
```

以上のスクリプトを

```
% paics.run.lsf.sh [ 入力ファイル名 ] [ 出力ファイル名 ] [ CPU 数 ]
```

のように実行する。

3　テスト計算

3.1　テスト計算の実行

インストール後、2 節の「計算の実行」を参考に、サンプルの入力ファイルを使ってテスト計算を行うことを推奨する。この際、実行の手順やスクリプトの書き方などは、各自の計算機システムに依存するので、多少の試行錯誤は必要となる。テスト計算は、プリオンタンパクに関するもの以外は、シングルコアでも実行可能である。参考として、実行時間の例を以下に示す。これらは、CPU やメモリの指定を含め、サンプルの入力ファイルを一切変更せずに計算を行った結果である。

- h2o-4-ccpvdz.inp
  - 12 原子、96 基底、フラグメント数 1 の非 FMO 計算（RHF、RI–MP2）
  - Core 2 Quad Q9650、コア当たり 2.0 GByte メモリの計算機で、1 コアのみ使用
  - 計算時間：3.53 秒
- fmo-h2o-4-ccpvdz.inp
  - 12 原子、96 基底、フラグメント数 4 の FMO 計算（RHF、RI–MP2）
  - Core 2 Quad Q9650、コア当たり 2.0 GByte メモリの計算機で、1 コアのみ使用
  - 計算時間：2.49 秒
- c12h26-ccpvdz.inp
  - 38 原子、298 基底、フラグメント数 1 の非 FMO 計算（RHF、RI–MP2）
  - Core 2 Quad Q9650、コア当たり 2.0 GByte メモリの計算機で、1 コアのみ使用
  - 計算時間：247.11 秒
- fmo-c12h26-ccpvdz.inp
  - 38 原子、298 基底、フラグメント数 3 の FMO 計算（RHF、RI–MP2）
  - Core 2 Quad Q9650、コア当たり 2.0 GByte メモリの計算機で、1 コアのみ使用
  - 計算時間：271.02 秒
- gly5-ccpvdz.inp
  - 38 原子、379 基底、フラグメント数 1 の非 FMO 計算（RHF、RI–MP2）
  - Core 2 Quad Q9650、コア当たり 2.0 GByte メモリの計算機で、1 コアのみ使用
  - 計算時間：1183.19 秒
- fmo-gly5-ccpvdz.inp
  - 38 原子、379 基底、フラグメント数 3 の FMO 計算（RHF、RI–MP2）
  - Core 2 Quad Q9650、コア当たり 2.0 GByte メモリの計算機で、1 コアのみ使用

- 計算時間：988.21 秒

- fmo-prion-ccpvdz.inp
  - 1666 原子、16142 基底、フラグメント数 102 の FMO 計算（RHF、RI–MP2）
  - XeonE5429、コア当たり 2.0 GByte メモリの計算機で、8 コア使用
  - 計算時間：107616.05 秒（29.9 時間）

- fmo-prion-gn8-ccpvdz.inp
  - 1792 原子、16736 基底、フラグメント数 106 の FMO 計算（RHF、RI–MP2）
  - XeonE5429、コア当たり 2.0 GByte メモリの計算機で、8 コア使用
  - 計算時間： 114628.66 秒（31.8 時間）

[注意]

PAICS は、FMO 計算のみを想定しているので、通常の量子化学計算（非 FMO の計算）を実行する場合は、フラグメント数が 1 の FMO 計算を実行することになる。非 FMO 計算の入力ファイルであるにもかかわらず、フラグメントの定義が記述されているのはこのためである。

3.2 実行結果の確認

計算結果の出力ファイルが配布物に含まれているので、テスト計算が終了した後、結果を比較し正しく動作していることを確認する。

[注意 1]

FMO 計算の場合、モノマー SCC 計算、モノマー計算、ダイマー ES 計算、ダイマー計算の順に計算が進められる。モノマー計算が終了した時点で、FMO1 エネルギー（1 体のエネルギー）が出力され、ダイマー計算が終了した時点で、FMO2 エネルギー（2 体のエネルギー）が出力される（これらの定義の詳細は、マニュアルの「理論との対応」を参照)。一方、非 FMO 計算の場合は、フラグメント数が 1 の FMO 計算として扱われ、モノマー SCC 計算は行われず、モノマー計算が（環境静電ポテンシャルが無い状態で）1 回だけ実行され、計算が終了する。出力ファイルを先頭から眺めて、このような手順で計算が進んでいることを確認し、エネルギーなどが添付の計算結果と一致していることを確かめて頂きたい。

[注意 2]

PAICS では、モノマー SCC 計算のイタレーションの回数を減らすため、ダイナミックアップデートを採用している。従って、並列のしかたで収束過程が異なり、モノマー SCC 計算のイタレーショ

ンの回数が変化する。（並列のしかたで収束過程が異なることに疑問をもつかもしれないが、ダイナミックアップデートの特徴であり、論文にも明記されている。基本的に、モノマー SCC 計算の閾値に応じて、同じ電子密度に収束する。）また、***PAICS*** のモノマー SCC 計算の収束の閾値は、エネルギーで 10^{-6} となっている。従って、最終的な計算結果は、通常の量子化学計算（非 FMO 計算）よりも若い桁数で差がでる可能性があり、添付の計算結果と比較する際は、注意して頂きたい。ただし、明らかに変な場合は、コンパイルや実行のしかたに問題があると考えるべき。（もちろん、バグの可能性もあるので、解決できない場合は開発グループに連絡して下さい。）

入力

1　キーワード

1.1　一般ルール

キーワードをを記述した後「スペース」「改行」のいずれかでで区切り、値を記述する。

[注意]
× `mpi_np = 4` ···· このように＝を使って値を指定してはいけない
○ `mpi_np 4` ···· 単にスペースで区切り値を指定する

1.2　並列関連

- mpi_np　[**int**]

  各フラグメントもしくはフラグメントペアの計算に使われる CPU（コア）の数。例えば全体の CPU（コア）数が 8 で、このキーワードで 2 が指定された場合、各フラグメントやフラグメントペアの計算が 2CPU（コア）で実行され、それらの計算が 4 並列で進められる。従って、全体の CPU（コア）数をこの値で割り算した場合、必ず割り切れなければならない。デフォルト値は 1。以下のキーワードを使い、モノマー SCC 計算、モノマー計算、ダイマー計算に関して個別に指定することも出来る。また、全体の CPU（コア）数はプログラム実行時に、MPI のオプションで指定される。FMO 計算の場合、フラグメントおよびフラグメントペアの計算は 1 つの CPU（コア）で実行し、それぞれの計算を並列に実行するのが最も並列効率が高い。従って、特別な理由が無い限り、この値は 1 に設定するべき。メモリの制限などで、1CPU（コア）では実行不可能なサイズのフラグメントやフラグメントペアが存在する場合のみ、2 以上にする。（例えば、比較的大きなリガンド分子をフラグメント分割せずに計算する場合など。）

- mpi_np_scc　[**int**]

  モノマー SCC 計算において、各フラグメントの計算に使われる CPU（コア）の数。指定されなければ、mpi_np の値が使用される。mpi_np 同様、全体の CPU（コア）数をこの値で割り算した場合、必ず割り切れなければならない。***PAICS*** では、モノマー SCC 計算にダイナミックアップデートを採用しているため、この値によってイタレーションの回数が変わる。この値を全 CPU 数と同じ値に設定した場合、イタレーションの回数が最小になり、逆に、この値を 1 に設定した場合、イタレーションの回数が最大になる。一方、mpi_np キーワードで説明したとおり、並列効率は 1 に設定した場合が最も高い。

- mpi_np_mon　[**int**]

  モノマー計算において、各フラグメントの計算に使われる CPU（コア）の数。指定されなければ、mpi_np の値が使用される。mpi_np 同様、全体の CPU（コア）数をこの値で割り算した場合、必ず割り切れなければならない。mpi_np キーワードで説明したとおり、特別な理由が無い限り 1 に設定する。

- mpi_np_dim　[**int**]

  ダイマー計算において、各フラグメントペアの計算に使われる CPU

```
0 : bohr
1 : オングストローム
```

デフォルトは 0 で「bohr」となっている。

- w_result_ le ［ **char** ］

  結果を出力する際のファイル名に使われる文字列。絶対パスで指定する。1020 文字以内で指定する。基本的に計算結果は標準出力に書き出されるので、必ず指定しなければならないわけではないが、ログファイルやモノマー SCC の密度を書き出す場合は出力する必要がある。

- w_log_ le ［ **int** ］

  ログファイルを作るか否かの指定。ファイル名は、「 [w_result_ le] _ [mpi_rank] . log 」となる。以下のように整数で指定する。

  ```
  0 : 作らない
  1 : 作る
  ```

  デフォルトは 0。

- w_scc ［ **int** ］

  モノマー SCC 計算の結果をファイルに書き出すか否かの指定。ファイル名は、「 [w_result_ le] . scc 」となる。以下のように整数で指定する。

  ```
  0 : 作らない
  1 : 作る
  ```

  デフォルトは 0。

- r_result_ le ［ **char** ］

  結果を読み込む際のファイル名に使われる文字列。絶対パスで指定する。1020 文字以内で指定する。計算結果を書き出したファイルを読む場合に指定する必要がある。

- r_scc ［ **int** ］

  モノマー SCC 計算の結果をファイルから読むか否かの指定。読まれた各フラグメントの密度情報は、モノマー SCC 計算の初期密度として利用される。読み込むファイル名は、「 [r_result_ le] . scc 」となる。以下のように整数で指定する。

  ```
  0 : 読まない
  1 : 読む
  ```

デフォルトは 0。

- atom

  原子と基底関数を指定するキーワード。必ず指定されるべきキーワード。指定のしかたは後述。（atom キーワードの代わりに、nucleus キーワードと basis キーワードを使用することも可能）。

- nucleus（別名：charge）

  原子（原子核）を指定するキーワード。このキーワードを用いた場合は、basis キーワードを用いて基底関数を別に指定する必要がある。通常、基底関数と原子は同じ位置に置かれるので、atom キーワードを使うことを推奨する。

- basis

  基底関数を指定するキーワード。このキーワードを用いた場合は、nucleus キーワードを用いて原子（原子核）を別に指定する必要がある。通常、基底関数と原子は同じ位置に置かれるので、atom キーワードを使うことを推奨する。

- fragment [**int**]

  フラグメントの数を指定するキーワード。必ず指定されるべきキーワード。1 を指定すれば、FMO 計算ではなく通常の量子化学計算（フラグメントの数が 1 の FMO 計算）が実行される。

- frag_atom

  フラグメント分割の定義を指定するキーワード。必ず指定されるべきキーワード。fragment キーワードで指定した数と同数の定義が記述されていなければならない。fragment キーワードで 1 が指定された場合も記述する必要がある。記述の方法は後述。

- frag_def（別名：frag）

  フラグメント分割の定義を指定するキーワード。通常は、frag_atom キーワードを用いる。（開発初期はこのキーワードを使ってフラグメントを定義していたが、後に frag_atom キーワードが導入された。）

- basis_def

  基底関数の定義を記述する際に使用されるキーワード。このキーワードを用いて、代表的な基底関数のいくつかの定義を記述したファイルが用意されている。、ユーザーが新たに基底関数を定義することも可能。指定の方法は後述。

- ex_point_charge

  外部静電ポテンシャルとして作用する点電荷を指定するキーワード。以下のように、点電荷の数を指定した後、電荷の値と座標を、通し番号を付けて記述する。

```
ex_point_charge [ 点電荷の数 ]
      1 [ 電荷の値 ] [ x 座標 ] [ y 座標 ] [ z 座標 ]
      2 [ 電荷の値 ] [ x 座標 ] [ y 座標 ] [ z 座標 ]
              .
              .
              .
```

- position

  空間の位置を指定するキーワード。以下のように、位置の数を指定した後、座標を通し番号を付けて記述する。このキーワードで指定された位置は、電子密度、静電ポテンシャル、電場などの物理量を計算する際に参照される。

```
position [ 位置の数 ]
      1 [ x 座標 ] [ y 座標 ] [ z 座標 ]
      2 [ x 座標 ] [ y 座標 ] [ z 座標 ]
              .
              .
              .
```

1.5 積分関連

- eri_tv [**double**]

  積分計算の際の、K_{ab} によるスクリーニングの閾値を指定。デフォルト値は 1.0E−12。

- eri_cauchy_tv [**double**]

  積分計算の際の、コーシー・シュバルツの不等式によるスクリーニングの閾値を指定。デフォルト値は 1.0E−10。

- eri_use_gen [**int**]

  4 中心電子反発積分の計算に、一般ルーチンを使うか否かを指定。以下のように整数で指定する。

```
0 : 特化ルーチンを使用
1 : 一般ルーチンを使用
```

デフォルトは 0。通常、一般ルーチンを使用することは無い（デバッグ用）。

- eri_3cen_use_gen ［ **int** ］

  3 中心電子反発積分の計算に、一般ルーチンを使うか否かを指定。以下のように整数で指定する。

  ```
  0 : 特化ルーチンを使用
  1 : 一般ルーチンを使用
  ```

  デフォルトは 0。通常、一般ルーチンを使用することは無い（デバッグ用）。

1.6 FMO 法全般

- scc_maxit ［ **int** ］

  モノマー SCC 計算における、イタレーションの最大値。デフォルトは 999。

- scc_tv_1 ［ **double** ］

  モノマー SCC 計算の収束判定の閾値の 1 つ。すべてのモノマーエネルギーの変化が、この値よりも小さくなったところで収束と見なす。デフォルトは 1.0E−6。scc_tv_1 と scc_tv_2 の両方の基準が同時に満たされたときに収束となる。

- scc_tv_2 ［ **double** ］

  モノマー SCC 計算の収束判定の閾値の 1 つ。系全体の FMO1 エネルギーの変化が、この値よりも小さくなったところで収束と見なす。デフォルトは 1.0E−6。scc_tv_1 と scc_tv_2 の両方の基準が同時に満たされたときに収束となる。

- ldimer ［ **double** ］

  ダイマー ES 近似を適用する閾値。ファンデルワールス半径の倍数で指定する。フラグメント間の再近接の原子間距離がこの距離よりも大きいフラグメントペアには、ダイマー ES 近似が適用される。デフォルトは 2.0 。

- lptc ［ **double** ］

  環境静電ポテンシャルの計算に点電荷近似を適用する閾値。ファンデルワールス半径の倍数で指定する。フラグメント間の再近接の原子間距離がこの距離よりも大きいフラグメントからの環境静電ポテンシャルの計算には、点電荷近似が適用される。デフォルトは 2.0。

- laoc [**double**]

  環境静電ポテンシャルの計算に 3 中心積分の近似を適用する閾値。ファンデルワールス半径の倍数で指定する。フラグメント間の再近接の原子間距離がこの距離よりも大きいフラグメントからの環境静電ポテンシャルの計算には、3 中心積分の近似が適用される。デフォルトは 0.0。

- projection_tv [**double**]

  結合の切断に伴う projection 演算子に掛かる正の数値」。デフォルトは 1.0E+6。

- cp_corr [**int**]

  結合を共有していないフラグメント間の IFIE に対して、counter-poise 法による BSSE の補正を見積もるか否かの指定。以下のように整数で指定される。

  ```
  1 : CP 補正を行う
  0 : CP 補正を行わない
  ```

  デフォルトは 0。RHF、MP2、RI–MP2 計算で BSSE の補正が計算される。BSSE の補正は、対象のフラグメントペアを真空中に抜き出して（環境静電ポテンシャルの影響が無い状況で）算出される。従って、正確な CP 補正が行われるわけではなく、BSSE の値を見積もる程度と考えるべき。（1 にした場合は、BSSE の補正が適用されていない IFIE も出力される。）

- scc_no_dyn [**int**]

  モノマー SCC 計算でダイナミックアップデートを使うか否かを指定。以下のように整数で指定する。

  ```
  1 : ダイナミックアップデートを使わない
  0 : ダイナミックアップデートを使う
  ```

  デフォルトは 0（従ってダイナミックアップデートを行う）。ダイナミックアップデートとはモノマー SCC のイタレーションの回数を減らすための工夫であり、mpi_np_scc もしくは mpi_np の値によって、収束までのイタレーションの回数が異なるという特徴がある。基本的に、ダイナミックアップデートの使用に関わらず、得られる電子密度は（収束判定の範囲内で）同じとなる。（他のプログラムでは採用されていないので、気になるようであれば 1 に設定すれば良い。）

- pro_loc_tv [**double**]

  projection 演算子を自動生成する際の局在化の閾値。デフォルトは 1.0E−8。変える必要は無い（デバッグ用）。***PAICS*** では、結合の切断に使用する射影演算子を、計算を実行するた

びに自動的に生成する（他のプログラムでは、基底関数ごとにデーターベースとして保持している場合が多い)。従って、モノマー SCC 計算を実行する前に、結合の切断の数だけ、メタン分子の RHF 計算を実行することになる。

- pro_loc_maxit　[**int**]

  projection 演算子を自動生成する際のイタレーション回数の最大値。デフォルト値は 999。変える必要は無い（デバッグ用)。

- pro_h_basis　[**char ***]

  projection 演算子を自動生成する際に CH_4 分子の水素原子上に設置される基底関数の種類を指定。デフォルトは STO-3G_001。変える必要は無い（デバッグ用)。

- frag_calc_pair　[int] [list ...]

  ダイマー計算を実行するフラグメントペアを指定するキーワード。通常、FMO 計算では全てのダイマーペアの計算を実行するが、フラグメント間の相互作用エネルギー（IFIE）のみが必要な場合は、対応するダイマーペアの計算だけで良い。(これを指定した場合は、トータルエネルギーや電子密度などの全体のプロパティは得られない)。以下のように指定する。

```
frag_calc_pair [ リストの数 ]
    [ リスト 1 ]
    [ リスト 2 ]
          .
          .
          .
```

  [リスト] にはフラグメントペアを指定する記述が入り、以下のように記述する。

```
例 1 : ifrag jfrag          ---> ifrag と jfrag のペア
例 2 : ifrag jfrag1-jfrag2  ---> ifrag と jfrag1 から jfrag2 までのペア
例 3 : ifrag ALL            ---> ifrag と全てのフラグメントペア
```

1.7　RHF 関連

- rhf_chk　[**int**]

  RHF 計算を行うか否かを指定するフラグ。以下のように整数で指定。

```
1 : RHF 計算を行う
0 : RHF 計算を行わない。
```

デフォルトは 1。

- rhf_chk_no_int_bu

  RHF の際、積分を（可能な限り）メモリに保存し再利用するか否かを指定するフラグ。以下のように整数で指定する。

  ```
  1 : 積分値は一切保存せず毎回全ての積分を計算する
  0 : 可能な限り積分値を保存し再利用する
  ```

  デフォルトは 0。通常はデフォルトのままで計算を行う（1 はデバッグ用）。

- rhf_lprint_1 [**int**]

  モノマー RHF 計算のプリントフラグ。-1 にすると推奨の出力になる。デフォルト値は-1。

- rhf_lprint_2 [**int**]

  ダイマー RHF 計算のプリントフラグ。-1 にすると推奨の出力になる。デフォルト値は -1。

- rhf_maxit [**int**]

  SCF イタレーションの最大数。デフォルト値は 999 。

- rhf_ndiis [**int**]

  DIIS のエラーベクトルを計算するために保持する FOCK 行列の数。デフォルト値は 4。全てのモノマーおよびダイマーペアにこのキーワードで指定した値が適用される。

- rhf_diis_tv [**double**]

  DIIS に切り替える閾値。エラーベクトルの最大値がこの値よりも小さくなったら DIIS を行う。デフォルト値は 1.0。

- rhf_orth [**int**]

  基底関数の直交化を行う手法の指定。以下のように整数で指定する。

  ```
  0 : 正準直交化
  1 : 対称直交化
  ```

  デフォルト値は 0。

- rhf_init_mo [**int**]

  初期軌道を作成する手法の選択。

```
0 : HCORE
1 : 射影
```

デフォルト値は 1。FMO 計算の場合、モノマー SCC 計算の 1 回目のイタレーションの時だけ、初期軌道を作成する必要がある。2 回目以降のイタレーションでは、前回の解が初期軌道として使用される。また、モノマー RHF 計算では、モノマー SCC 計算で得られた解が初期軌道として使われる（この場合、初期軌道が解そのもの）。ダイマー RHF 計算では、各フラグメントのモノマー RHF 計算の解を合わせたものが初期軌道として使われる。

- rhf_orth_tv [**double**]

  正準直交化の際に参照される重なり行列の固有値の閾値。デフォルト値は 1.0E−6。

- rhf_eng_tv [**double**]

  収束の判定を行う際のエネルギーの閾値。デフォルト値は 1.0E−8。

1.8 MP2 関連（カノニカル MP2）

- cmp2_chk [**int**]

  MP2 計算を行うか否かのフラグ。以下のように整数で指定する。

```
0 : 計算しない
1 : 計算する
```

  デフォルト値は 0。（こちらの MP2 ではなく、RI–MP2 を使用することを推奨する）。

- cmp2_lprint_1 [**int**]

  のモノマー MP2 計算の出力フラグ。-1 だと推奨の出力になる。デフォルト値は-1。

- cmp2_lprint_2 [**int**]

  ダイマー MP2 計算の出力フラグ。-1 だと推奨の出力になる。デフォルト値は-1。

- cmp2_th_iajs [**double**]

  MP2 の積分変換に使用される閾値の 1 つ。デフォルト値は 1.0E−8。

- cmp2_th_iars [**double**]

  MP2 の積分変換に使用される閾値の 1 つ。デフォルト値は 1.0E−8。

- cmp2_th_pqrs　[**double**]

  MP2 の積分変換の際に使用される閾値の 1 つ。デフォルト値は 1.0E−8。

1.9　RI–MP2 関連

- ri_cmp2_chk　[**int**]

  RI–MP2 計算を行うか否かのフラグ。以下のように整数で指定する。

  ```
  0 : 計算しない
  1 : 計算する
  ```

  デフォルト値は 0。RI–MP2 計算では補助基底が使用され、補助基底の種類は自動的に選択される。現在は、cc–pVDZ、cc–pVTZ の補助基底のみ（それぞれ、cc–pVDZri、cc–pVTZri）が定義されており、

  `cc-pVDZri` → `cc-pVDZ`、`cc-pVDZso`、`6-31G`、`6-31G**`の場合使用される。
  `cc-pVTZri` → `cc-pVTZ` の場合使用される。

- ri_cmp2_lprint_1　[**int**]

  モノマー RI–MP2 のプリントフラグ。-1 で推奨の出力になる。デフォルト値は-1。

- ri_cmp2_lprint_2　[**int**]

  ダイマー RI–LMP2 のプリントフラグ。-1 で推奨の出力になる。デフォルト値は-1。

1.10　局在化 MP2 関連

- lmp2_chk　[**int**]

  局在化 MP2 計算を行うか否かのフラグ。以下のように整数で指定する。

  ```
  0 : 計算しない
  1 : 計算する
  ```

  デフォルト値は 0。***PAICS*** の LMP2 は、高速化のためではなく、相互作用解析（FILM）を実行するために実装されている。従って、フラグメントペア間のカットオフの閾値が厳しく設定されており、計算には時間が掛かる。

- lmp2_lprint_1　[**int**]

  モノマー LMP2 のプリントフラグ。-1 で推奨の出力になる。デフォルト値は-1。

- lmp2_lprint_2 [**int**]

  ダイマー LMP2 のプリントフラグ。-1 で推奨の出力になる。デフォルト値は-1。

- lmp2_lprint_loc_1 [**int**]

  モノマー LMP2 の局在化に関するプリントフラグ。-1 で推奨の出力になる。デフォルト値は -1。

- lmp2_lprint_loc_2 [**int**]

  ダイマー LMP2 の局在化に関するプリントフラグ。-1 で推奨の出力になる。デフォルト値は -1。

- lmp2_loc [**int**]

  局在化の手法の指定。以下のように整数で指定する。

  ```
  0 : Pipek-Mezey
  1 : Boys
  2 : 局在化せず
  ```

  デフォルト値は 0。

- lmp2_max_itr [**int**]

  線形方程式のイタレーションの最大数。デフォルト値は 30。

- lmp2_th_1 [**double**]

  ドメインの決定に使用される閾値の 1 つ。デフォルト値は 0.02。

- lmp2_th_1_dim [**double**]

  ドメインの決定に使用される閾値の 1 つ。デフォルト値は 0.001。

- lmp2_th_2 [**double**]

  軌道ペアの選択に使用される閾値の 1 つ。デフォルト値は 4.0。

- lmp2_th_2_dim [**double**]

  軌道ペアの選択に使用される閾値の 1 つ。デフォルト値は 8.0。

- lmp2_th_3 [**double**]

  積分変換に使用される閾値。デフォルト値は 0.004。

- lmp2_th_4 [**double**]

  積分変換に使用される閾値。デフォルト値は 1.0E−12。

2　原子と基底関数

PAICS では、原子と基底関数を atom キーワードを用いて、以下のように指定する。

3　フラグメント分割

FRAGMENT キーワードでフラグメントの数を指定し、FRAG_ATOM キーワードで各フラグメントの定義を指定する。従って FRAG_ATOM キーワードは、フラグメントの数だけ記述されていなければならない。以下のよう記述する。

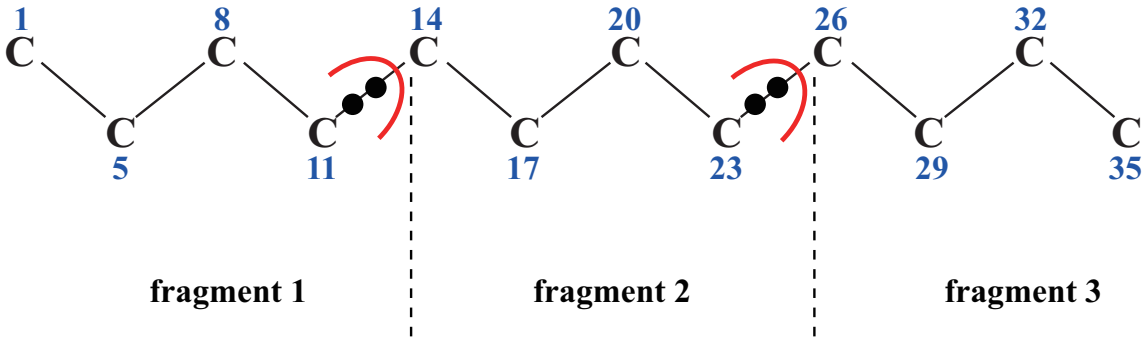
```
FRAGMENT
 [ フラグメントの数 ]

FRAG_ATOM [ 電荷 ] [ 原子の数 ] [ 結合の切断に伴う追加の原子の数 ]
   [ 原子の通し番号を [ 原子の数 ] だけ並べる . . . ]
   [ 追加の原子の通し番号を [ 追加の原子の数 ] だけ並べる . . . ]

FRAG_ATOM [ 電荷 ] [ 原子の数 ] [ 結合の切断に伴う追加の原子の数 ]
   [ 原子の通し番号を [ 原子の数 ] だけ並べる . . . ]
   [ 追加の原子の通し番号を [ 追加の原子の数 ] だけ並べる . . . ]

FRAG_ATOM
   .
   .
   .
```

基本的には、各フラグメントの電荷、原子の数、結合の切断に伴う追加の原子の数を指定し、原子の通し番号、追加の原子の通し番号を記述する。例として、$C_{12}H_{26}$ を**図 1** のように分割した場合を示す。

```
FRAGMENT
 3

FRAG_ATOM  0  13  1
 1 2 3 4 5 6 7 8 9 10 11 12 13
 14

FRAG_ATOM  0  12  1
 14 15 16 17 18 19 20 21 22 23 24 25
 26

FRAG_ATOM  0  13  0
 26 27 28 29 30 31 32 33 34 35 36 37 38
```

14 番目の炭素原子は fragment1 の原子核ではないが、結合の切断に伴い fragment1 に追加される。（厳密には 14 番目の炭素原子の+6 のうち、+1 だけが追加される。）よって、fragment1 における追加の原子は 14 番目の原子となる。同様に、fragment2 には 26 番目の原子が追加される。fragment3 には追加される原子は存在しない。

図 1: $C_{12}H_{26}$ 分子（3 分割）の切断の定義。（図では水素原子を省略。数字は原子の通し番号）。



4 基底関数の定義

PAICS には以下の基底関数が定義されている。

	基底関数	定義名	ファイル	原子番号
1	**STO-3G**	STO-3G _ ***	basis / sto3g.dat	$1 \sim 53$
2	**6-31G**	6-31G _ ***	basis / 631g.dat	$1 \sim 30$
3	**6-31G****	6-31G** _ ***	basis / 631gdp.dat	$1 \sim 30$
4	**cc-pVDZ**	cc-pVDZ _ ***	basis / cc-pVDZ.dat	$1 \sim 36$
5	**cc-pVTZ**	cc-pVTZ _ ***	basis / cc-pVTZ.dat	$1 \sim 36$
6	**cc-pVQZ**	cc-pVQZ _ ***	basis / cc-pVQZ.dat	$1 \sim 36$
7	**cc-pVDZso**	cc-pVDZso _ ***	basis / cc-pVDZso.dat	$1 \sim 36$
8	**cc-pVTZso**	cc-pVTZso _ ***	basis / cc-pVTZso.dat	$1 \sim 36$
9	**cc-pVDZri**	cc-pVDZri _ ***	basis / cc-pVDZri.dat	$1 \sim 36$
10	**cc-pVTZri**	cc-pVTZri _ ***	basis / cc-pVTZri.dat	$1 \sim 36$

また、用意されている基底関数の以外に、独自に基底関数を定義することも可能で、入力ファイル中で BASIS_DEF キーワードを用いて以下のように記述する。

```
BASIS_DEF
   [ 定義名 ]
   [ シェルの数 ]
      [ 角運動量 ]  [ 短縮数 ]
           [ 通し番号 ]  [ 原始ガウス関数の係数 ]  [ 原始ガウス関数の指数 ]
                         .
                         .

      [ 角運動量 ]   [ 短縮数 ]
           [ 通し番号 ]  [ 原始ガウス関数の係数 ]  [ 原始ガウス関数の指数 ]
                         .
                         .

      [ 角運動量 ]   [ 短縮数 ]
                .
                .
```

図 2 に炭素原子の cc-pVDZ の例を示す。ユーザーが基底関数の定義を追加するための定義ファイル user_def.dat も用意されている。***PAICS*** を実行すると、上の基底関数定義ファイルと uset_def.dat が自動的に読まれる。また、基底関数の定義を、インプットファイルに記述しておくことも出来る。

図 2: 炭素原子の cc-pVDZ の定義。

```
BASIS_DEF
  cc-pVDZ_006
  6
  0     8
         1            0.0006920         6665.0000000
         2            0.0053290         1000.0000000
         3            0.0270770          228.0000000
         4            0.1017180           64.7100000
         5            0.2747400           21.0600000
         6            0.4485640            7.4950000
         7            0.2850740            2.7970000
         8            0.0152040            0.5215000
  0     8
         1           -0.0001460         6665.0000000
         2           -0.0011540         1000.0000000
         3           -0.0057250          228.0000000
         4           -0.0233120           64.7100000
         5           -0.0639550           21.0600000
         6           -0.1499810            7.4950000
         7           -0.1272620            2.7970000
         8            0.5445290            0.5215000
  0     1
         1            1.0000000            0.1596000
  1     3
         1            0.0381090            9.4390000
         2            0.2094800            2.0020000
         3            0.5085570            0.5456000
  1     1
         1            1.0000000            0.1517000
  2     1
         1            1.0000000            0.5500000
```

理論との対応

1 FMO–RHF（エネルギー）

1.1 エネルギー

FMO 法におけるモノマーおよびダイマーの RHF 計算では、通常の Fock 演算子に「他のフラグメントからの静電ポテンシャル演算子」と「結合の切断に伴う射影演算子」を加えた以下のような Fock 演算子を用いる（X は、フラグメントおよびフラグメントペアを示すインデックスで、モノマー計算では $X = I$、ダイマー計算では $X = IJ$ と考える）。

$$\tilde{f}^X = f^X + \sum_{K \neq X} \left\{ u^{(K)} + v^{(K)} \right\} + P^X \tag{1.1}$$

$$u^{(K)} = \sum_{A \in K} \frac{-Z_A}{\mid \mathbf{R}_A - \mathbf{r}_1 \mid} \tag{1.2}$$

$$v^{(K)} = \int d\mathbf{r}_2 \frac{\rho^K(\mathbf{r}_2)}{\mid \mathbf{r}_1 - \mathbf{r}_2 \mid} \tag{1.3}$$

$$\hat{P}^X = B \sum_k \mid \theta^X{}_k \rangle \langle \theta^X{}_k \mid \tag{1.4}$$

ここで、$u^{(K)}$ は K 番目のフラグメントの原子核からの引力ポテンシャル演算子、$v^{(K)}$ は K 番目のフラグメントの電子からの反発ポテンシャル演算子である（通常、これらを環境静電ポテンシャルと呼ぶ）。P^X は結合の切断に伴う射影演算子である。また、f^X は各フラグメントおよびフラグメントペアに関する通常の Fock 演算子で、

$$f^X = h^X + 2\, J^X - K^X \tag{1.5}$$

$$h^X = -\frac{1}{2} \nabla^2 + \sum_{A \in X} \frac{-Z_A}{\mid \mathbf{R}_A - \mathbf{r_1} \mid} \tag{1.6}$$

である。また、J^X および K^X は、それぞれクーロン演算子および交換演算子である。X に含まれる基底関数を $\mu, \nu \cdots$ とし、各演算子の表現行列を、

$$\mathbf{H_{core}}^X{}_{\mu\nu} = \langle \mu \mid h^X \mid \nu \rangle \tag{1.7}$$

$$\mathbf{u}^{X(K)}{}_{\mu\nu} = \langle \mu \,|\, u^K \,|\, \nu \rangle \tag{1.8}$$

$$\mathbf{v}^{X(K)}{}_{\mu\nu} = \langle \mu \,|\, v^K \,|\, \nu \rangle \tag{1.9}$$

$$\mathbf{P}^{X}{}_{\mu\nu} = \langle \mu \,|\, P \,|\, \nu \rangle \tag{1.10}$$

$$\tilde{\mathbf{G}}^{X}{}_{\mu\nu} = \langle \mu \,|\, 2J^X - K^X \,|\, \nu \rangle \tag{1.11}$$

と定義する。さらに、

$$\mathbf{V}^{X(K)} = \mathbf{u}^{X(K)} + \mathbf{v}^{X(K)} \tag{1.12}$$

$$\mathbf{V}^{X} = \sum_{K \neq X} \mathbf{V}^{X(K)} \tag{1.13}$$

$$\tilde{\mathbf{H}}_{\mathbf{core}}{}^{X} = \mathbf{H}_{\mathbf{core}}{}^{X} + \mathbf{V}^{X} + \mathbf{P}^{X} \tag{1.14}$$

$$\tilde{\mathbf{F}}^{X} = \tilde{\mathbf{H}}_{\mathbf{core}}{}^{X} + \tilde{\mathbf{G}}^{X} \tag{1.15}$$

の行列を定義する。また、$\tilde{f}^{\,X}$ による Fock 方程式を解いて得られた分子軌道を、

$$\tilde{\psi}_{\,i}{}^{X} = \sum_{\mu} \tilde{C}^{X}{}_{\mu i}\, \phi_\mu \tag{1.16}$$

と定義し、密度行列を、

$$\tilde{\mathbf{D}}^{X}{}_{\mu\nu} = 2 \sum_{i} \tilde{C}^{X}{}_{\mu i}\, \tilde{C}^{X}{}_{\nu i} \tag{1.17}$$

と定義する。

[注意]

PAICSでは、モノマー計算に関しては、分子軌道の数、軌道エネルギー、軌道係数行列を保存するが、ダイマー計算に関しては、そのペアの計算が終わった時点で値を破棄する。

モノマーおよびダイマーのエネルギーは

$$E^{HF}{}_X = \frac{1}{2}\, Tr\left\{ \tilde{\mathbf{D}}^X \left(\tilde{\mathbf{H}}_{\mathbf{core}}{}^X + \tilde{\mathbf{F}}^X \right) \right\} \tag{1.18}$$

と書ける。さらに、ここから環境静電ポテンシャルの寄与を取り除いたものを E'^{HF} とし、

$$E'^{HF}{}_X = E^{HF}{}_X - Tr\left\{ \tilde{\mathbf{D}}^X \mathbf{V}^X \right\} \tag{1.19}$$

と定義する。全系の FMO1−RHF エネルギーおよび FMO2−RHF エネルギーは

$$E^{HF}{}_{fmo1} = \sum_I E'^{HF}{}_I \tag{1.20}$$

$$E^{HF}{}_{fmo2} = \sum_{I<J} E^{HF}{}_{IJ} - (N-2) \sum_I E^{HF}{}_I \tag{1.21}$$

と書ける。$E^{HF}{}_{fmo2}$ を書き換えると、

$$E^{HF}{}_{fmo2} = \sum_I E^{HF}{}_I + \sum_{I>J} \left(E^{HF}{}_{IJ} - E^{HF}{}_I - E^{HF}{}_J \right) \tag{1.22}$$

となる。さらに E'^{HF} を用いて書き直すと、

$$\begin{aligned} E^{HF}{}_{fmo2} = & \sum_I E'^{HF}{}_I + \sum_{I>J} \left(E'^{HF}{}_{IJ} - E'^{HF}{}_I - E'^{HF}{}_J \right) \\ & + \sum_I Tr\left\{ \tilde{\mathbf{D}}^I \mathbf{V}^I \right\} + \sum_{I>J} \left(Tr\left\{ \tilde{\mathbf{D}}^{IJ} \mathbf{V}^{IJ} \right\} - Tr\left\{ \tilde{\mathbf{D}}^I \mathbf{V}^I \right\} - Tr\left\{ \tilde{\mathbf{D}}^J \mathbf{V}^J \right\} \right) \end{aligned} \tag{1.23}$$

となり、

$$E^{HF}{}_{fmo2} = \sum_I E'^{HF}{}_I + \sum_{I>J} \left(E'^{HF}{}_{IJ} - E'^{HF}{}_I - E'^{HF}{}_J \right) + \sum_{I>J} \left(Tr\left\{ \tilde{\mathbf{D}}^{IJ} \mathbf{V}^{IJ} \right\} - Tr\left\{ \tilde{\mathbf{D}}^{I(J)} \mathbf{V}^{IJ} \right\} - Tr\left\{ \tilde{\mathbf{D}}^{J(I)} \mathbf{V}^{IJ} \right\} \right) \quad (1.24)$$

となる。ここで、$\tilde{\mathbf{D}}^{I(J)}$ は、IJ ペアの基底関数と同じ次元数の行列で、

$$\mu \in I \text{ かつ } \nu \in I \text{ の場合} \qquad \tilde{\mathbf{D}}^{I(J)}{}_{\mu\nu} = \tilde{\mathbf{D}}^{I}{}_{\mu\nu}$$

$$\text{それ以外の場合} \qquad \tilde{\mathbf{D}}^{I(J)}{}_{\mu\nu} = 0$$

と定義される。さらに、$\Delta\tilde{\mathbf{D}}^{IJ}$ を

$$\Delta\tilde{\mathbf{D}}^{IJ} = \tilde{\mathbf{D}}^{IJ} - \tilde{\mathbf{D}}^{I(J)} - \tilde{\mathbf{D}}^{J(I)} \quad (1.25)$$

を定義すると、(1.24) は

$$E^{HF}{}_{fmo2} = \sum_I E'^{HF}{}_I + \sum_{I>J} \left(E'^{HF}{}_{IJ} - E'^{HF}{}_I - E'^{HF}{}_J \right) + \sum_{I>J} Tr\left\{ \Delta\tilde{\mathbf{D}}^{IJ}\, \mathbf{V}^{IJ} \right\} \quad (1.26)$$

となる。新たに $\Delta E^{HF}{}_{IJ}$ を、

$$\Delta E^{HF}{}_{IJ} = \left(E'^{HF}{}_{IJ} - E'^{HF}{}_I - E'^{HF}{}_J \right) + Tr\left\{ \Delta\tilde{\mathbf{D}}^{IJ} \mathbf{V}^{IJ} \right\} \quad (1.27)$$

と定義すると、FMO2−RHF エネルギーは

$$E^{HF}{}_{fmo2} = \sum_I E'^{HF}{}_I + \sum_{I>J} \Delta E^{HF}{}_{IJ} \quad (1.28)$$

となる。第 1 項は FMO1–RHF エネルギーなので、

$$E^{HF}{}_{fmo2} = E^{HF}{}_{fmo1} + \sum_{I>J} \Delta E^{HF}{}_{IJ} \quad (1.29)$$

と書ける。従って、$\Delta E^{HF}{}_{IJ}$ は FMO1–RHF エネルギーに対する 2 体の補正項となっており、これをフラグメント間の相互作用エネルギーと考える。通常、この値は、inter fragment interaction energy（IFIE）もしくは pair interaction energy（PIE）と呼ばれる。各数値とプログラム中の変数との対応関係を**表 1** および**表 2** に記載する。

1.2 環境静電ポテンシャルの近似

環境静電ポテンシャル行列（$\mathbf{V}^X$）は、他のフラグメントの原子核からの引力ポテンシャル（$\mathbf{u}^X$）と電子密度からの反発ポテンシャル（$\mathbf{v}^X$）の和として

$$\mathbf{V}^X = \sum_{K \neq X} \mathbf{u}^{X(K)} + \sum_{K \neq X} \mathbf{v}^{X(K)} \tag{1.30}$$

と計算される。$\mathbf{u}^{X(K)}$ の項は

$$\sum_{K \neq X} \mathbf{u}^{X(K)}{}_{\mu\nu} = \langle\, \mu \mid \sum_{K \neq X} \sum_{A \in K} \frac{-Z_A}{\mid \mathbf{R}_A - \mathbf{r}_1 \mid} \mid \nu \,\rangle \tag{1.31}$$

のように計算される。一方、$\mathbf{v}^{X(K)}$ の項は 3 つの近似レベルで計算される。

1. 相手のフラグメントが作る電荷密度からの静電ポテンシャルを、近似を用いずに 4 中心積分から計算する。
2. 相手のフラグメントが作る電子密度からの静電ポテンシャルを、3 中心積分の近似を用いて計算する（esp-aoc 近似）
3. 相手のフラグメントが作る電子密度を原子核上のポピュレーションと近似し、静電ポテンシャルを計算する（esp-ptc 近似）

ここで、$\mathbf{v}^{X(K)}$ を各計算レベルからの寄与に分けて、

$$\sum_{K \neq X} \mathbf{v}^{X(K)} = \sum_{K_4} \mathbf{v_4}^{X(K_4)} + \sum_{K_3} \mathbf{v_3}^{X(K_3)} + \sum_{K_p} \mathbf{v_p}^{X(K_p)} \tag{1.32}$$

と記述する。

＜ 4 中心積分による環境静電ポテンシャルの計算 ＞

（1.32）の $\mathbf{v_4}^{X(K_4)}$ は、4 中心積分から計算される環境静電ポテンシャルである。K_4 番目のフラグメントの電子密度 $\rho^{K_4}(\mathbf{r})$ から、

$$\sum_{K_4} \mathbf{v_4}^{X(K_4)}{}_{\mu\nu} = \sum_{K_4} \langle\, \mu \mid \int d\mathbf{r_2}\, \frac{\rho^{K_4}(\mathbf{r}_2)}{\mid \mathbf{r}_1 - \mathbf{r}_2 \mid} \mid \nu \,\rangle \tag{1.33}$$

のように計算される。ここで、$\rho^{K_4}(\mathbf{r})$ を、モノマー SCC 計算で得られた各フラグメントの密度行列 ${\mathbf{D}_s}^X$

PAICS では、6 成分系の d 軌道以上の基底関数は規格化されていない（5 成分系の基底関数は規格化されている）。通常、このことは特に注意される必要は無いが、（1.38）を計算する場合は規格化因子を考慮して、

$$\sum_{K_3} \mathbf{v_3}^{X(K_3)}{}_{\mu\nu} = \sum_{K_3} \sum_{\lambda \in K_3} \left(\tilde{\mathbf{D}}^{K_3} \mathbf{S} \right)_{\lambda\lambda} \frac{1}{\langle \lambda \mid \lambda \rangle} (\mu\nu \mid \lambda\lambda) \tag{1.39}$$

としなければならない。

< 点電荷近似による環境静電ポテンシャルの計算 >

（1.32）の $\mathbf{v_p}^{X(K_p)}$ は、点電荷近似による環境静電ポテンシャルである。モノマー SCC 計算で得られた各フラグメントの電子密度を、原子核（B）上のポピュレーション（p_{cB}）で近似し、それらから感じるポテンシャルとして、

$$\sum_{K_p} \mathbf{v_p}^{X(K_p)}{}_{\mu\nu} = \sum_{K_p} \langle \mu \mid \sum_{B \in K_p} \frac{p_{cB}}{\mid \mathbf{R}_B - \mathbf{r}_1 \mid} \mid \nu \rangle \tag{1.40}$$

と記述する。ここで出てきた各数値とプログラム中の変数との対応関係を**表 1** および**表 2** に記載する。

1.3 ダイマー ES 近似

ダイマー ES 近似とは、十分に離れたフラグメントペアに関して、ダイマー RHF 計算を行わずに ${E'}^{HF}{}_{IJ}$ を算出する近似法である。フラグメント間の静電的な相互作用のみを考慮して、以下のように計算される。

$$\begin{aligned} {E'}^{HF}{}_{IJ} &= {E'}^{HF}{}_{I} + {E'}^{HF}{}_{J} \\ &+ \sum_{A \in J} \int \frac{-Z_A \ \rho^I(\mathbf{r_1})}{\mid \mathbf{R}_A - \mathbf{r}_1 \mid} d\mathbf{r_1} + \sum_{A \in I} \int \frac{-Z_A \ \rho^J(\mathbf{r_1})}{\mid \mathbf{R}_A - \mathbf{r}_1 \mid} d\mathbf{r_1} + \int \frac{\rho^I(\mathbf{r_1}) \rho^J(\mathbf{r_2})}{\mid \mathbf{r_1} - \mathbf{r_2} \mid} d\mathbf{r_1} d\mathbf{r_2} \end{aligned} \tag{1.41}$$

これを密度行列を用いて書き直すと、

$$\begin{aligned} {E'}^{HF}{}_{IJ} &= {E'}^{HF}{}_{I} + {E'}^{HF}{}_{J} \\ &+ \sum_{\mu\nu \in I} \tilde{\mathbf{D}}^I_{\mu\nu} \mathbf{u}^{I(J)}{}_{\mu\nu} + \sum_{\mu\nu \in J} \tilde{\mathbf{D}}^J_{\mu\nu} \mathbf{u}^{J(I)}{}_{\mu\nu} + \sum_{\mu\nu \in I} \sum_{\lambda\sigma \in J} \tilde{\mathbf{D}}^I_{\mu\nu} \tilde{\mathbf{D}}^J_{\lambda\sigma} (\mu\nu \mid \lambda\sigma) \end{aligned} \tag{1.42}$$

となる。また、ダイマー ES 近似を用いた場合、そのペアに関しては、

$$Tr\left\{\mathbf{\Delta\tilde{D}}^{IJ}\,\mathbf{V}^{IJ}\right\} = Tr\left\{\mathbf{\tilde{D}}^{IJ}\,\mathbf{V}^{IJ}\right\} - Tr\left\{\mathbf{\tilde{D}}^{I(J)}\,\mathbf{V}^{IJ}\right\} - Tr\left\{\mathbf{\tilde{D}}^{J(I)}\,\mathbf{V}^{IJ}\right\} = 0 \tag{1.43}$$

なので、(1.27) より、$\Delta {E^{HF}}_{IJ}$ は、

$$\Delta {E^{HF}}_{IJ} = {E'^{HF}}_{IJ} - {E'^{HF}}_{I} - {E'^{HF}}_{J} \tag{1.44}$$

となる。よって、(1.42) を (1.44) へ代入すると、

$$\Delta {E^{HF}}_{IJ} = \sum_{\mu\nu\in I} \tilde{\mathbf{D}}^{I}_{\mu\nu}\mathbf{u}^{I(J)}{}_{\mu\nu} + \sum_{\mu\nu\in J} \tilde{\mathbf{D}}^{J}_{\mu\nu}\mathbf{u}^{J(I)}{}_{\mu\nu} + \sum_{\mu\nu\in I}\sum_{\lambda\sigma\in J} \tilde{\mathbf{D}}^{I}_{\mu\nu}\tilde{\mathbf{D}}^{J}_{\lambda\sigma}(\,\mu\nu\mid\lambda\sigma\,) \tag{1.45}$$

が得られる。ここで出てきた各数値とプログラム中の変数との対応関係を**表 1** および**表 2** に記載する。

1.4 原子核を含めたエネルギー

(1.18) のモノマーおよびダイマーのエネルギーは電子エネルギーのみを考慮したものだが、原子核のポテンシャルエネルギーを考慮すると、

$$\begin{aligned}\bar{E}^{HF}{}_{X} = \frac{1}{2}\,Tr\left\{\,\tilde{\mathbf{D}}^{X}\left(\,\tilde{\mathbf{H}}_{\mathbf{core}}{}^{X} + \tilde{\mathbf{F}}^{X}\,\right)\right\} + \sum_{(A<B)\,\in X}\frac{z_A z_B}{\mid \mathbf{R}_A - \mathbf{R}_B \mid} \\ + \sum_{K\neq X}\sum_{A\in X}\sum_{C\in K}\frac{z_A z_C}{\mid \mathbf{R}_A - \mathbf{R}_C \mid} + \sum_{K\neq X}\sum_{A\in X}\int\frac{-z_A\,\rho^K(\mathbf{r})}{\mid \mathbf{R}_A - \mathbf{r} \mid}\,d\mathbf{r}\end{aligned} \tag{1.46}$$

となる(上付の印は、原子核のエネルギーを含めた表式と電子のみのエネルギー表式を区別するため)。第 2 項目が「X 内の原子核の核間反発」、第 3 項目が「X 内の原子核と他のフラグメントの原子核との核間反発ポテンシャル」、第 4 項目が「X 内の原子核と他のフラグメントの電子との核引力ポテンシャル」である。ここで、それぞれを、

$${Z_u}^{X} = \sum_{(A<B)\,\in X}\frac{z_A z_B}{\mid \mathbf{R}_A - \mathbf{R}_B \mid} \tag{1.47}$$

$${Z_u}^{X-K} = \sum_{A\in X}\sum_{C\in K}\frac{z_A z_C}{\mid \mathbf{R}_A - \mathbf{R}_C \mid} \tag{1.48}$$

$$Z_e{}^{X-K} = \sum_{A \in X} \int \frac{z_A \, \rho^K(\mathbf{r})}{\mid \mathbf{R}_A - \mathbf{r} \mid} \, d\mathbf{r} \tag{1.49}$$

と表す。(1.46) は、

$$\bar{E}^{HF}{}_X = E^{HF}{}_X + Z_u{}^X + \sum_{K \neq X} Z_u{}^{X-K} + \sum_{K \neq X} Z_e{}^{X-K} \tag{1.50}$$

となる。

他のフラグメントからの静電ポテンシャルの寄与を取り除いたモノマーエネルギーの和が FMO1–RHF エネルギーなので、

$$\bar{E}^{HF}{}_{fmo1} = \sum_I E'{}^{HF}{}_I + \sum_I Z_u{}^I \tag{1.51}$$

と書ける。つまり、電子のみの FMO1–RHF エネルギー（第 1 項）に各フラグメント内の核間反発エネルギーの和（第 2 項）を加えたものが、原子核のポテンシャルエネルギーまで含めた FMO1–RHF エネルギーとなる。また、FMO2–RHF エネルギーは、

$$\bar{E}^{HF}{}_{fmo2} = \sum_{I<J} \bar{E}^{HF}{}_{IJ} - (N-2) \sum_I \bar{E}^{HF}{}_I \tag{1.52}$$

であり、これを計算すると、

$$\bar{E}^{HF}{}_{fmo2} = \sum_{I<J} E^{HF}{}_{IJ} - (N-2) \sum_I E^{HF}{}_I + \Big\{ \sum_I Z_u{}^I + \sum_{I<J} Z_u{}^{I-J} \Big\} \tag{1.53}$$

が得られる。つまり、電子のみの FMO2–RHF エネルギー（第 1 項、第 2 項）に、系全体の核間反発エネルギー（第 3 項、第 4 項）を加えたものが、原子核のポテンシャルエネルギーまで含めた FMO2–RHF エネルギーとなる。また、この式は、

$$\bar{E}^{HF}{}_{fmo2} = \bar{E}^{HF}{}_{fmo1} + \sum_{I<J} \Big(\Delta E^{HF}{}_{IJ} + Z_u{}^{I-J} \Big) \tag{1.54}$$

と書け、第 2 項が 2 体の補正を表している。よって、IFIE は、

$$\Delta \bar{E}^{HF}{}_{IJ} = \Delta E^{HF}{}_{IJ} + Z_u{}^{I-J} \tag{1.55}$$

となる。つまり、電子のみの IFIE（第 1 項）にフラグメント間の核反発エネルギー（第 2 項）を加えたものが、原子核のポテンシャルエネルギーまで含めた IFIE となる。

1.5 外部静電ポテンシャルが存在する場合

外部静電ポテンシャルが存在する場合、(1.1) の fock 演算子は、

$$\tilde{f}^{X} = f^{X} + \sum_{K \neq X} \left\{ u^{K} + v^{K} \right\} + P^{X} + {v_e}^{X} \tag{1.56}$$

となる。また、${v_e}^{X}(\mathbf{r})$ の基底関数による表現行列を、

$${\mathbf{V_e}}^{X}{}_{\mu\nu} = \langle \mu | v_e | \nu \rangle \tag{1.57}$$

とする。${E^{HF}}_{X}$ および ${E'^{HF}}_{X}$ は、(1.56) を用いた fock 方程式から得られた密度行列を用いて (1.19) および (1.18) のように定義される。従って、${E'^{HF}}_{X}$ は X の内部エネルギーと外部静電ポテンシャルとの相互作用エネルギーを含んでいる（つまり、他のフラグメントからの静電ポテンシャルのみ除外されている)。${E'^{HF}}_{X}$ から、さらに外部静電ポテンシャルとの相互作用エネルギーを取り除いたものを、

$${E''^{HF}}_{X} = {E'^{HF}}_{X} - Tr\left\{ \tilde{\mathbf{D}}^{X} {\mathbf{V_e}}^{X} \right\} \tag{1.58}$$

と定義すると、(1.20) から、FMO1–RHF エネルギーは、

$${E^{HF}}_{fmo1} = \sum_{I} {E''^{HF}}_{I} + \sum_{I} Tr\left\{ \tilde{\mathbf{D}}^{I} {\mathbf{V_e}}^{I} \right\} \tag{1.59}$$

と書ける。第 1 項が内部エネルギーを示し、第 2 項が外部静電ポテンシャルとの相互作用エネルギーを示している。FMO2–RHF エネルギーは (1.26) なので、${E''^{HF}}_{X}$ を用いると

$$\begin{aligned} {E^{HF}}_{fmo2} = & \sum_{I} {E''^{HF}}_{I} + \sum_{I>J} \left({E''^{HF}}_{IJ} - {E''^{HF}}_{I} - {E''^{HF}}_{J} \right) + \sum_{I>J} Tr\left\{ \Delta\tilde{\mathbf{D}}^{IJ}\, \mathbf{V}^{IJ} \right\} \\ & + \sum_{I} Tr\left\{ \tilde{\mathbf{D}}^{I} {\mathbf{V_e}}^{I} \right\} + \sum_{I>J} \left(Tr\left\{ \tilde{\mathbf{D}}^{IJ} {\mathbf{V_e}}^{IJ} \right\} - Tr\left\{ \tilde{\mathbf{D}}^{I} {\mathbf{V_e}}^{I} \right\} - Tr\left\{ \tilde{\mathbf{D}}^{J} {\mathbf{V_e}}^{J} \right\} \right) \end{aligned} \tag{1.60}$$

となる。さらに、$\mathbf{V_e}$ の場合は、単純に、

$$Tr\left\{ \tilde{\mathbf{D}}^{I} {\mathbf{V_e}}^{I} \right\} = Tr\left\{ \tilde{\mathbf{D}}^{I(J)} {\mathbf{V_e}}^{IJ} \right\} \tag{1.61}$$

$$Tr\left\{\tilde{\mathbf{D}}^{J}\,{\mathbf{V_e}}^{J}\right\} = Tr\left\{\tilde{\mathbf{D}}^{J(I)}\,{\mathbf{V_e}}^{IJ}\right\} \tag{1.62}$$

なので、

$$\begin{aligned} {E^{HF}}_{fmo2} = \sum_{I} {E''^{HF}}_{I} + \sum_{I>J}\left({E''^{HF}}_{IJ} - {E''^{HF}}_{I} - {E''^{HF}}_{J}\right) + \sum_{I>J} Tr\left\{\Delta\tilde{\mathbf{D}}^{IJ}\,\mathbf{V}^{IJ}\right\} \\ + \sum_{I} Tr\left\{\tilde{\mathbf{D}}^{I}\,{\mathbf{V_e}}^{I}\right\} + \sum_{I>J} Tr\left\{\boldsymbol{\Delta}\tilde{\mathbf{D}}^{IJ}\,{\mathbf{V_e}}^{IJ}\right\} \end{aligned} \tag{1.63}$$

となる。第 1、2、3 項が内部エネルギーを示し、第 4、5 項が外部静電ポテンシャルとの相互作用エネルギーを示している。さらに、(1.63) は、

$$\begin{aligned} {E^{HF}}_{fmo2} = {E^{HF}}_{fmo1} + \sum_{I>J}\Big\{ \left({E''^{HF}}_{IJ} - {E''^{HF}}_{I} - {E''^{HF}}_{J}\right) \\ + Tr\left(\Delta\tilde{\mathbf{D}}^{IJ}\,\mathbf{V}^{IJ}\right) + Tr\left(\boldsymbol{\Delta}\tilde{\mathbf{D}}^{IJ}\,{\mathbf{V_e}}^{IJ}\right)\Big\} \end{aligned} \tag{1.64}$$

と書け、この式の第 2 項が 2 体の補正となっている。よって、外部静電ポテンシャルが存在する場合の IFIE を、以下のように定義することができる。

$$\begin{aligned} \Delta{E^{HF}}_{IJ} = \left({E''^{HF}}_{IJ} - {E''^{HF}}_{I} - {E''^{HF}}_{J}\right) \\ + Tr\left(\Delta\tilde{\mathbf{D}}^{IJ}\,\mathbf{V}^{IJ}\right) + Tr\left(\boldsymbol{\Delta}\tilde{\mathbf{D}}^{IJ}\,{\mathbf{V_e}}^{IJ}\right) \end{aligned} \tag{1.65}$$

外部静電ポテンシャルが存在しない場合は、${E''^{HF}}_{X}$ は ${E'^{HF}}_{X}$ と同じ値で、${\mathbf{V_e}}^{X}$ はゼロ行列なので、(1.27) の定義と一致する。まとめると、

- FMO1–RHF の全エネルギーが (1.59) で、そのうち、内部エネルギーが第 1 項で、外部静電ポテンシャルとの相互作用エネルギーが第 2 項。
- FMO2–RHF の全エネルギーが (1.63) で、そのうち、内部エネルギーが第 1、2、3 項で、外部静電ポテンシャルとの相互作用エネルギーが第 4、5 項。
- IFIE は (1.65)。

である。ダイマー ES ペアに関しては、(1.63) の第 3 項と第 5 項、同様に (1.64) の中括弧の第 2 項と第 3 項がゼロになる。

< 原子核を含めたエネルギー >

外部静電ポテンシャルが存在すると、原子核もポテンシャルエネルギーを感じる。よって、原子核のポテンシャルエネルギーまで考慮する場合、

$$\sum_I Z^{I-ext} \tag{1.66}$$

を追加する必要がある。ここで、Z^{X-ext} は、X の原子核が外部静電ポテンシャルから感じるポテンシャルエネルギーである。各数値とプログラム中の変数との対応関係を**表 1** および**表 2** に記載する。

1.6 ダイマーペアの計算を制限した場合

全系を F_1 と F_2 に分け、F_1 のフラグメントのインデックスを I、J とし、F_2 のフラグメントのインデックスを K、L とする。この場合、FMO1–RHF のエネルギー表式（1.20）は、

$$E^{HF}{}_{fmo1} = \sum_I E'^{HF}{}_I + \sum_K E'^{HF}{}_K \tag{1.67}$$

と書ける。また、FMO2–RHF のエネルギー表式（1.28）は、

$$E^{HF}{}_{fmo2} = \sum_I E'^{HF}{}_I + \sum_{I>J} \Delta E^{HF}{}_{IJ} + \sum_I \sum_K \Delta E^{HF}{}_{IK} + \sum_K E'^{HF}{}_K + \sum_{K>L} \Delta E^{HF}{}_{KL} \tag{1.68}$$

と書ける。今、F_1 のフラグメントを含むペアに関してはダイマー計算を実行するが、他のペアに関しては実行しない場合を考える。このとき、（1.68）の第 5 項が計算されないため、全系の FMO2–RHF エネルギーを得ることはできない。しかし、

$$F_1 \text{のフラグメント数} \ll F_2 \text{のフラグメント数}$$

の場合、計算時間の多くが第 5 項の計算に使われるため、大幅に計算時間を減らすことができる。これは、F_1 と F_2 間の相互作用解析が目的の場合には有効である。例えば、リガンドを F_1 としタンパクを F_2 とすると、全エネルギーを計算する場合に比べて 1/10 程度の時間で、リガンド－タンパク間の IFIE を全て計算することができる。

＜ 部分エネルギーの定義 ＞

（1.68）の第 1 項と第 2 項は、F_1 の内部エネルギーを表している。また、第 3 項は、F_1 が F_2 から感じる相互作用エネルギーを表している。そこで、F_1 の「部分エネルギー」を

$$E^{HF}{}_{fmo2(F_1)} = \sum_I E'^{HF}{}_I + \sum_{I>J} \Delta E^{HF}{}_{IJ} + \sum_I \sum_K \Delta E^{HF}{}_{IK} \tag{1.69}$$

と定義する。この表式は「部分エネルギー勾配」の定式化で使用される。

1.7 BSSE の補正

FMO 法では、環境静電ポテンシャルを含めた計算を行うので、CP 補正は単純ではない。また、結合の切断がある場合は、切断部分の基底関数が二重に使用されるので、さらに困難となる。そこで、「結合を互いにシェアしていないフラグメント間」の IFIE における BSSE の値のみを、「環境静電ポテンシャルの影響を無視した真空中の計算から見積もる」ことにする。そのためには、通常のモノマー計算およびダイマー計算に加えて、

- モノマー J の位置にも基底関数を設置した真空中のモノマー I の計算 ： ${E_{vac}}^{HF}{}_{I\,(IJ)}$
- モノマー I の位置にも基底関数を設置した真空中のモノマー J の計算 ： ${E_{vac}}^{HF}{}_{J\,(IJ)}$
- 真空中のモノマー I の計算 ： ${E_{vac}}^{HF}{}_I$
- 真空中のモノマー J の計算 ： ${E_{vac}}^{HF}{}_J$

の 4 つのエネルギー計算が必要になる。これらを用いると、フラグメントペアの BSSE は、

$$E^{HF-BSSE}{}_{IJ} = (\ {E_{vac}}^{HF}{}_{I\,(IJ)} - {E_{vac}}^{HF}{}_I\ ) + (\ {E_{vac}}^{HF}{}_{J\,(IJ)} - {E_{vac}}^{HF}{}_J\ ) \tag{1.70}$$

と見積もられる。よって、CP 補正された IFIE は

$$\Delta E^{HF-CP}{}_{IJ} = \Delta E^{HF}{}_{IJ} - E^{HF-BSSE}{}_{IJ} \tag{1.71}$$

となる。ここで出てきた各数値とプログラム中の変数との対応関係を**表 1** および**表 2** に記載する。

1.8 計算結果として出力される数値

＜モノマー SCC 計算終了後＞

- フラグメントの正電荷の中心 ： + charge center

  モノマー SCC 計算の結果に無関係に、原子核の位置が指定された段階で決まる値。各フラグメントごとに出力される。

- フラグメントの負電荷の中心 ： - charge center

  モノマー SCC 計算で得られた各フラグメントの電子密度から計算される。各フラグメントごとに出力される。

＜モノマー計算終了後＞

- モノマーエネルギー 1 ： rhf (E')

  原子核のポテンシャルエネルギーおよび外部静電ポテンシャルとの相互作用エネルギーを含めたモノマーのエネルギー。環境静電ポテンシャルからの寄与は除外されている。

$$E'^{HF}{}_I + Z_u{}^I + Z^{I-ext} \tag{1.72}$$

- モノマーエネルギー 2 ： rhf (E'')

  モノマーエネルギー 1 から、外部静電ポテンシャルとの相互作用エネルギーを取り除いた値。外部静電ポテンシャルが存在しない場合は、モノマーエネルギー 1 と同じ値となる。

$$E''^{HF}{}_I + Z_u{}^I \tag{1.73}$$

- FMO1–RHF エネルギー 1 ： total

  原子核のポテンシャルエネルギーおよび外部静電ポテンシャルとの相互作用エネルギーを含めた FMO1–RHF エネルギー。

$$\sum_I \left\{ E'^{HF}{}_I + Z_u{}^I + Z^{I-ext} \right\} \tag{1.74}$$

- FMO1–RHF エネルギー 2 ： internal

  FMO1–RHF エネルギー 1 から、外部静電ポテンシャルとの相互作用エネルギーを取り除いた値。外部静電ポテンシャルが存在しない場合は、FMO1–RHF エネルギー 1 と同じ値となる。

$$\sum_I \left\{ E''^{HF}{}_I + Z_u{}^I \right\} \tag{1.75}$$

- FMO1–RHF エネルギー 3 ： external

  FMO1–RHF エネルギー 1 のうち、外部静電ポテンシャルとの相互作用エネルギーのみの値。FMO1–RHF エネルギー 2 と FMO1–RHF エネルギー 3 を足すと FMO1–RHF エネルギー 1 になる。外部静電ポテンシャルが存在しない場合はゼロとなる。

$$\sum_I \Big\{ Tr\big(\mathbf{D}^I {\mathbf{V_e}}^I\big) + Z^{I-ext} \Big\} \tag{1.76}$$

＜ダイマー計算終了後＞

- RHF–IFIE 1 ： rhf

  原子核のポテンシャルエネルギーと外部静電ポテンシャルとの相互作用エネルギーを含めた IFIE。

$$\Delta {E^{HF}}_{IJ} + {Z_u}^{I-J} \tag{1.77}$$

- RHF–IFIE 2 ： rhf (cp)

  RHF–IFIE 1 から BSSE の見積もり値を引いた値。

$$\Delta {E^{HF}}_{IJ} + {Z_u}^{I-J} - {E^{HF-BSSE}}_{IJ} \tag{1.78}$$

- FMO2–RHF エネルギー 1 ： total

  原子核のポテンシャルエネルギーと外部静電ポテンシャルとの相互作用エネルギーを含めた FMO2–RHF エネルギー。

$$\begin{aligned} \sum_I \Big\{ {E''^{HF}}_I + {Z_u}^I + Tr\big(\mathbf{D}^I {\mathbf{V_e}}^I\big) + Z^{I-ext} \Big\} + \\ \sum_{I>J} \Big\{ \big({E''^{HF}}_{IJ} - {E''^{HF}}_I - {E''^{HF}}_J \big) \\ + Tr\left(\boldsymbol{\Delta\tilde{\mathbf{D}}}^{IJ}\, \mathbf{V}^{IJ}\right) + {Z_u}^{I-J} + Tr\left(\boldsymbol{\Delta\tilde{\mathbf{D}}}^{IJ} {\mathbf{V_e}}^{IJ}\right) \Big\} \end{aligned} \tag{1.79}$$

- FMO2–RHF エネルギー 2 ： internal

  FMO2–RHF エネルギー 1 から、外部静電ポテンシャルとの相互作用エネルギーを取り除いた値。外部静電ポテンシャルが存在しない場合は、FMO2–RHF エネルギー 1 と同じ値となる。

$$\begin{aligned} \sum_I \Big\{ {E''^{HF}}_I + {Z_u}^I \Big\} + \\ \sum_{I>J} \Big\{ \big({E''^{HF}}_{IJ} - {E''^{HF}}_I - {E''^{HF}}_J \big) \\ + Tr\left(\Delta\tilde{\mathbf{D}}^{IJ}\, \mathbf{V}^{IJ}\right) + {Z_u}^{I-J} \Big\} \end{aligned} \tag{1.80}$$

- FMO2–RHF エネルギー 3 ： external

  FMO2–RHF エネルギー 1 のうち、外部静電ポテンシャルとの相互作用エネルギーのみの値。FMO2–RHF エネルギー 2 と FMO2–RHF エネルギー 3 を足すと FMO2–RHF エネルギー 1 になる。外部静電ポテンシャルが存在しない場合はゼロとなる。

$$\sum_{I}\left\{Tr\left(\mathbf{D}^{I}\mathbf{V_e}^{I}\right)+Z^{I-ext}\right\}+\sum_{I>J}Tr\left(\mathbf{\Delta\tilde{D}}^{IJ}\mathbf{V_e}^{IJ}\right) \tag{1.81}$$

表 1: グローバル変数との対応（FMO−RHF エネルギー関連）

変数名	次元数	値
rhf_mon_energy_d	フラグメント数	${E'}^{HF}{}_I$
rhf_mon_trdv	フラグメント数	$Tr\left\{\mathbf{D}^I\,\mathbf{V}^I\right\}$
rhf_mon_trdv_e	フラグメント数	$Tr\left\{\mathbf{D}^I\,{\mathbf{V_e}}^I\right\}$
rhf_mon_energy_vac_cp	フラグメント数	${E_{vac}}^{HF}{}_I$
rhf_dim_ifie	フラグメントペア数	$\Delta {E^{HF}}_{IJ}$
rhf_dim_trdv_e	フラグメントペア数	$Tr\left\{\Delta\mathbf{D}^{IJ}\,{\mathbf{V_e}}^{IJ}\right\}$
rhf_dim_ifie_cp	フラグメントペア数	${E^{HF-BSSE}}_{IJ}$

表 2: グローバル変数との対応（FMO−RHF エネルギー関連）

変数名	次元数	値
frag_nuclear_repulsion	フラグメント数	${Z_u}^I$
frag_nuclear_ext	フラグメント数	Z^{I-ext}
scc_d_mat	フラグメント数 * nbo ^{2}	${\tilde{\mathbf{D}}_s}^{X\,(a)}$
scc_mulliken_ba	基底関数の数	${p_{sB}}^{(a)}$
scc_mulliken_bo	nbo	$\left({\tilde{\mathbf{D}}_s}^X\mathbf{S}\right)_{\lambda\lambda}^{(a)}$
rhf_mon_nmo	フラグメント数	分子軌道の数 $^{(b)}$
rhf_mon_moe	フラグメント数 * nbo	軌道エネルギー
rhf_mon_moe_vac_cp	フラグメント数 * nbo	軌道エネルギー $^{(c)}$
rhf_mon_c_mat	フラグメント数 * nbo ^{2}	係数行列
rhf_mon_c_mat_vac_cp	フラグメント数 * nbo ^{2}	係数行列 $^{(c)}$

$^{(a)}$ モノマー SCC 計算で得られた値。各フラグメントごとに記録される。イタレーションごとに上書きされ、最終的な値が残る。

$^{(b)}$ 正準直交化を使う場合、nbo と異なる可能性がある。

$^{(c)}$ 真空中で計算された値（つまり、環境静電ポテンシャルを考慮せず計算された値）。BSSE の値を見積もる際に使用される。

2 FMO–RHF（電子密度）

2.1 電子密度

フラグメントおよびフラグメントペアの電子密度 $\rho^X(\mathbf{r})$ は、モノマーおよびダイマーの分子軌道 $\tilde{\psi}^X{}_i(\mathbf{r})$ を用いて以下のように書ける。

$$\rho^X(\mathbf{r}) = \sum_{i\ \in X}^{occ.} n_i \mid \tilde{\psi}^X{}_i^*(\mathbf{r})\ \tilde{\psi}^X{}_i(\mathbf{r}) \mid \tag{2.1}$$

ここで n_i は占有数で、RHF 計算では 2 となる。（2.1）は、

$$\rho^X(\mathbf{r}) = \sum_{\mu\nu\in X} \Big(\sum_i n_i\ \tilde{C}^X{}_{\mu i}\ \tilde{C}^X{}_{\nu i} \Big)\ \phi_\mu(\mathbf{r})\ \phi_\nu(\mathbf{r}) \tag{2.2}$$

となり、密度行列を用いると、

$$\rho^X(\mathbf{r}) = \sum_{\mu\nu\in X} \tilde{\mathbf{D}}^X{}_{\mu\nu}\ \phi_\mu(\mathbf{r})\ \phi_\nu(\mathbf{r}) \tag{2.3}$$

となる。FMO 計算の全電子密度は、モノマーおよびダイマーの電子密度から、以下のように定義される。

$$\rho\ {}_{fmo1}\ (\mathbf{r}) = \sum_I \rho^I(\mathbf{r}) \tag{2.4}$$

$$\rho\ {}_{fmo2}\ (\mathbf{r}) = \sum_{I>J} \rho^{IJ}(\mathbf{r}) - (N-2) \sum_I \rho^I(\mathbf{r}) \tag{2.5}$$

さらに計算を進めると、$\rho\ {}_{fmo2}\ (\mathbf{r})$ は、

$$\rho\ {}_{fmo2}\ (\mathbf{r}) = \rho\ {}_{fmo1}\ (\mathbf{r}) + \sum_{I>J} \Big\{\ \rho^{IJ}(\mathbf{r}) - \rho^I(\mathbf{r}) - \rho^J(\mathbf{r})\ \Big\} \tag{2.6}$$

となる。ここで、

$$\Delta\rho^{IJ}(\mathbf{r}) = \rho^{IJ}(\mathbf{r}) - \rho^I(\mathbf{r}) - \rho^J(\mathbf{r}) = \sum_{\mu\nu\in IJ} \Delta\mathbf{D}^{IJ}{}_{\mu\nu}\ \phi_\mu(\mathbf{r})\ \phi_\nu(\mathbf{r}) \tag{2.7}$$

を定義すると、

$$\rho_{\ fmo2}\left(\mathbf{r}\right) = \rho_{\ fmo1}\left(\mathbf{r}\right) + \sum_{I>J} \Delta\rho^{IJ}(\mathbf{r}) \tag{2.8}$$

と書ける。ここまでに出てきた各数値とプログラム中の変数との対応関係を**表 3** に記載する。

2.2 Mulliken 密度解析

全電子数を N_{elec} とすると、

$$N_{elec} = \int \rho\left(\mathbf{r}\right)\, d\mathbf{r} \tag{2.9}$$

なので、基底関数の重なり行列 $\mathbf{S}$ を用いて

$$N_{elec\ (fmo1)} = \sum_{I} \sum_{\mu \in I} \left(\tilde{\mathbf{D}}^{I}\ \mathbf{S}\right)_{\mu\mu} \tag{2.10}$$

$$N_{elec\ (fmo2)} = N_{elec\ (fmo1)} + \sum_{I>J} \sum_{\mu \in IJ} \left(\mathbf{\Delta}\tilde{\mathbf{D}}^{IJ}\ \mathbf{S}\right)_{\mu\mu} \tag{2.11}$$

と書ける。ここで、

$$N_{elec} = N_{elec\ (fmo1)} = N_{elec\ (fmo2)} \tag{2.12}$$

である。原子 A 上の電子密度は、

$$N_{A\ fmo1} = \sum_{I} \sum_{\mu \in A} \left(\tilde{\mathbf{D}}^{I}\ \mathbf{S}\right)_{\mu\mu} \tag{2.13}$$

$$N_{A\ fmo2} = N_{A\ fmo1} + \left\{ \sum_{I>J} \sum_{\mu \in A} \left(\mathbf{\Delta}\tilde{\mathbf{D}}^{IJ}\ \mathbf{S}\right)_{\mu\mu} \right\} \tag{2.14}$$

となる。ここで、${N_A}^{I}$ と $\Delta {N_A}^{IJ}$ を以下のように定義する。

$$N_A{}^I = \sum_{\mu \in A} \left(\tilde{\mathbf{D}}^I \ \mathbf{S} \right)_{\mu\mu} \tag{2.15}$$

$$\Delta N_A{}^{IJ} = \sum_{\mu \in A} \left(\boldsymbol{\Delta}\tilde{\mathbf{D}}^{IJ} \ \mathbf{S} \right)_{\mu\mu} \tag{2.16}$$

これらを用いると、各原子上の電子密度は、

$$N_{A\ fmo1} = \sum_I \ N_A{}^I \tag{2.17}$$

$$N_{A\ fmo2} = N_{A\ fmo1} + \sum_{I>J} \Delta N_A{}^{IJ} \tag{2.18}$$

となる。ここまでに出てきた各数値とプログラム中の変数との対応関係を**表 3** に記載する。

2.3 静電ポテンシャル

電子密度 $\rho(\mathbf{r})$ が作る、座標 $\mathbf{r}_m$ の静電ポテンシャルは、

$$\phi(\mathbf{r}_m) = \int \frac{-\rho(\mathbf{r})}{\mid \mathbf{r}_m - \mathbf{r} \mid} d\mathbf{r} \tag{2.19}$$

である。FMO 計算による電子密度は、(2.4) および (2.8) なので、これらを (2.19) に代入すると、

$$\phi_{fmo1}(\mathbf{r}_m) = \sum_I \sum_{\mu\nu \in I} \tilde{\mathbf{D}}^I{}_{\mu\nu} \langle \ \mu \mid \frac{-1}{\mid \mathbf{r}_m - \mathbf{r} \mid} \mid \nu \ \rangle \tag{2.20}$$

$$\phi_{fmo2}(\mathbf{r}_m) = \phi_{fmo1}(\mathbf{r}_m) + \sum_{I>J} \sum_{\mu\nu \in IJ} \boldsymbol{\Delta}\tilde{\mathbf{D}}^{IJ}{}_{\mu\nu} \langle \ \mu \mid \frac{-1}{\mid \mathbf{r}_m - \mathbf{r} \mid} \mid \nu \ \rangle \tag{2.21}$$

となる。$\mathbf{u}^X(\mathbf{r}_m)$ 行列を

$$\mathbf{u}^X(\mathbf{r}_m)_{\mu\nu} = \langle \ \mu \mid \frac{-1}{\mid \mathbf{r}_m - \mathbf{r} \mid} \mid \nu \ \rangle \tag{2.22}$$

と定義すると、

$$\phi_{fmo1}(\mathbf{r}_m) = \sum_I Tr\Big(\tilde{\mathbf{D}}^I\,\mathbf{u}^I(\mathbf{r_m})\Big) \tag{2.23}$$

$$\phi_{fmo2}(\mathbf{r}_m) = \phi_{fmo1}(\mathbf{r}_m) + \sum_{I>J} Tr\Big(\boldsymbol{\Delta}\tilde{\mathbf{D}}^{IJ}\,\mathbf{u}^{IJ}(\mathbf{r}_m)\Big) \tag{2.24}$$

となる。

2.4 電場

FMO 計算による静電ポテンシャルは（2.23）および（2.24）なので、電場は、

$$\mathbf{E}_{fmo1}(\mathbf{r}_m) = -\frac{\partial}{\partial\,\mathbf{r}_m}\,\phi_{fmo1} = -\sum_I Tr\,\Big\{\,\tilde{\mathbf{D}}^I\Big(\frac{\partial}{\partial\,\mathbf{r}_m}\,\mathbf{u}^I(\mathbf{r}_m)\Big)\,\Big\} \tag{2.25}$$

$$\mathbf{E}_{fmo2}(\mathbf{r}_m) = -\frac{\partial}{\partial\,\mathbf{r}_m}\,\phi_{fmo2} = \mathbf{E}_{fmo1}(\mathbf{r}_m) - \sum_{I>J} Tr\,\Big\{\,\boldsymbol{\Delta}\tilde{\mathbf{D}}^{IJ}\,\Big(\frac{\partial}{\partial\,\mathbf{r}_m}\,\mathbf{u}^{IJ}(\mathbf{r}_m)\Big)\,\Big\} \tag{2.26}$$

となる。ここで、

$$\mathbf{g_m}^I = Tr\,\Big\{\,\tilde{\mathbf{D}}^I\,\Big(\frac{\partial}{\partial\mathbf{r}_m}\,\mathbf{u}^I(\mathbf{r}_m)\Big)\,\Big\} \tag{2.27}$$

$$\boldsymbol{\Delta}\mathbf{g_m}^{IJ} = Tr\,\Big\{\,\boldsymbol{\Delta}\tilde{\mathbf{D}}^{IJ}\,\Big(\frac{\partial}{\partial\mathbf{r}_m}\,\mathbf{u}^{IJ}(\mathbf{r}_m)\Big)\,\Big\} \tag{2.28}$$

と定義すると、

$$\mathbf{E}_{fmo1}(\mathbf{r}_m) = -\sum_I \mathbf{g_m}^I \tag{2.29}$$

$$\mathbf{E}_{fmo2}(\mathbf{r}_m) = \mathbf{E}_{fmo1}(\mathbf{r}_m) - \sum_{I>J} \Delta\mathbf{g_m}^{IJ} \tag{2.30}$$

と書ける。ダイマー ES ペアに関しては $\boldsymbol{\Delta}\mathbf{g_m}^{IJ}$ がゼロとなる。また、$\mathbf{u}^X(\mathbf{r}_m)$ の微分は、

$$\frac{\partial}{\partial\,\mathbf{r}_m}\,\mathbf{u}^X(\mathbf{r}_m)_{\mu\nu} = \frac{\partial}{\partial\,\mathbf{r}_m}\,\langle\,\mu\,|\frac{-1}{|\,\mathbf{r}_m-\mathbf{r}\,|}|\,\nu\,\rangle = \langle\,\mu\,|\,\Big(\frac{\partial}{\partial\mathbf{r}_m}\,\frac{-1}{|\,\mathbf{r}_m-\mathbf{r}\,|}\Big)\,|\,\nu\,\rangle \tag{2.31}$$

のように計算できる（基底関数 μ、ν は座標 $\mathbf{r}_m$ に依存してない）。

2.5 計算結果として出力される数値

＜ダイマー計算終了後＞

- Mulliken population 1 ： rhf–pop.

  基底関数上の電子数。

- Mulliken population 2 ： rhf–charge

  Mulliken population 1 に核電荷を加えた値。

- 電子密度 ： density

  指定された位置における電子密度。

- 静電ポテンシャル 1 ： esp (elec.)

  指定された位置における、電子密度が作る静電ポテンシャル。

- 静電ポテンシャル 2 ： esp (nuc.)

  指定された位置における、原子核が作る静電ポテンシャル。原子核の位置と指定された位置が同じ場合、該当する原子核からの寄与を除いた静電ポテンシャルが出力される。その場合、表示に*が付く。

- 静電ポテンシャル 3 ： esp (sum)

  静電ポテンシャル 1 と静電ポテンシャル 2 を足し合わせた値。外部静電ポテンシャルが存在する場合、これも電場に寄与するが、その分は含まれていない。

- 電場 1 ： (ele)

  指定された位置における、電子密度が作る電場。

- 電場 2 ： (nuc)

  指定された位置における、原子核が作る電場。原子核の位置と指定された位置が同じ場合、該当する原子核からの寄与を除いた電場が出力される。その場合、表示に*が付く。

- 電場 3 ： (sum)

  電場 1 と電場 2 を足し合わせた値。外部静電ポテンシャルが存在する場合、これも電場に寄与するが、その分は含まれていない。

表 3: グローバル変数との対応（FMO−RHF の密度関連）

変数名	次元数	値
rhf_mon_m_pop_ba	基底関数の数	$\sum_I {N_A}^I$
rhf_mon_pos_dns	位置の数	$\sum_I \rho^I(\mathbf{r})$
rhf_mon_pos_esp	位置の数	$\sum_I Tr\Big(\tilde{\mathbf{D}}^I \, \mathbf{u}^I(\mathbf{r}_m)\Big)$
rhf_mon_pos_efi_x	位置の数	$\sum_I {\mathbf{g_m}}^I$（ x 成分 ）
rhf_mon_pos_efi_y	位置の数	$\sum_I {\mathbf{g_m}}^I$（ y 成分 ）
rhf_mon_pos_efi_z	位置の数	$\sum_I {\mathbf{g_m}}^I$（ z 成分 ）
rhf_dim_m_pop_ba	基底関数の数	$\sum_{I>J} \Delta {N_A}^{IJ}$
rhf_dim_pos_dns	位置の数	$\sum_{I>J} \Delta \rho^{IJ}(\mathbf{r})$
rhf_dim_pos_esp	位置の数	$\sum_{I>J} Tr\Big(\boldsymbol{\Delta}\tilde{\mathbf{D}}^{IJ} \, \mathbf{u}^{IJ}(\mathbf{r}_m)\Big)$
rhf_dim_pos_efi_x	位置の数	$\sum_{I>J} \boldsymbol{\Delta}{\mathbf{g_m}}^{IJ}$（ x 成分 ）
rhf_dim_pos_efi_y	位置の数	$\sum_{I>J} \boldsymbol{\Delta}{\mathbf{g_m}}^{IJ}$（ y 成分 ）
rhf_dim_pos_efi_z	位置の数	$\sum_{I>J} \boldsymbol{\Delta}{\mathbf{g_m}}^{IJ}$（ z 成分 ）

3　FMO–MP2 計算（エネルギー）

3.1　エネルギー計算

分子全体の FMO1−MP2 エネルギーおよび FMO2−MP2 エネルギーは、モノマーの MP2 エネルギー（${E^{MP2}}_I$）とダイマーの MP2 エネルギー（${E^{MP2}}_{IJ}$）から

$$ {E^{MP2}}_{fmo1} = \sum_I {E^{MP2}}_I \tag{3.1} $$

$$ {E^{MP2}}_{fmo2} = \sum_{I>J} {E^{MP2}}_{IJ} \ - (N-2) \sum_I {E^{MP2}}_I \tag{3.2} $$

のように計算される。ここで、

$$ {E^{MP2}}_I = {E^{HF}}_I + {E^{corr(MP2)}}_I \tag{3.3} $$

$$ {E^{MP2}}_{IJ} = {E^{HF}}_{IJ} + {E^{corr(MP2)}}_{IJ} \tag{3.4} $$

であり、${E^{corr(MP2)}}_I$ と ${E^{corr(MP2)}}_{IJ}$ はモノマーとダイマーの MP2 相関エネルギーである。よって、(3.1) および (3.2) は

$$ {E^{MP2}}_{fmo1} = {E^{HF}}_{fmo1} + \sum_I {E^{corr(MP2)}}_I \tag{3.5} $$

$$ {E^{MP2}}_{fmo2} = {E^{HF}}_{fmo2} + \left(\sum_{I>J} {E^{corr(MP2)}}_{IJ} \ - (N-2) \sum_I {E^{corr(MP2)}}_I \right) \tag{3.6} $$

と書ける。さらに (3.6) は、

$$ {E^{MP2}}_{fmo2} = {E^{HF}}_{fmo2} + \left(\sum_I {E^{corr(MP2)}}_I + \sum_{I>J} \Delta {E^{corr(MP2)}}_{IJ} \right) \tag{3.7} $$

となる。ここで、

$$\Delta E^{corr(MP2)}{}_{IJ} = E^{corr(MP2)}{}_{IJ} - E^{corr(MP2)}{}_{I} - E^{corr(MP2)}{}_{J} \tag{3.8}$$

である。さらに、

$$E^{corr(MP2)}{}_{fmo1} = \sum_{I} E^{corr(MP2)}{}_{I} \tag{3.9}$$

$$E^{corr(MP2)}{}_{fmo2} = \sum_{I} E^{corr(MP2)}{}_{I} + \sum_{I>J} \Delta E^{corr(MP2)}{}_{IJ} \tag{3.10}$$

と定義すると、FMO1–MP2 エネルギーおよび FMO2–MP2 エネルギーは以下のように記述できる。

$$E^{MP2}{}_{fmo1} = E^{HF}{}_{fmo1} + E^{corr(MP2)}{}_{fmo1} \tag{3.11}$$

$$E^{MP2}{}_{fmo2} = E^{HF}{}_{fmo2} + E^{corr(MP2)}{}_{fmo2} \tag{3.12}$$

また、IFIE は HF 計算で得られた IFIE に MP2 の補正を加える形で、

$$\Delta E^{MP2}{}_{IJ} = \Delta E^{HF}{}_{IJ} + \Delta E^{corr(MP2)}{}_{IJ} \tag{3.13}$$

のように計算できる。spin-compornent-scaled MP2（SCS–MP2）法の場合は、上の式の各エネルギー値の代わりに「 スケールされたエネルギー値 」を用いるだけでよい。ここまでに出てきた数値とプログラム中での変数名の対応関係を**表 4** にまとめる。

3.2 BSSE の見積もり

以下、式が煩雑になるので上付の「(MP2)」を省略する。HF 計算で行ったように、「互いに結合をシェアしていないフラグメント間」に限り、BSSE の値を CP 補正から見積もることを考える。そのためには、通常のモノマー計算とダイマー計算に加えて、

- モノマー J の位置にも基底関数を設置した真空中のモノマー I の計算: ${E_{vac}}^{corr}{}_{I\,(IJ)}$
- モノマー I の位置にも基底関数を設置した真空中のモノマー J の計算: ${E_{vac}}^{corr}{}_{J\,(IJ)}$
- 真空中のモノマー I の計算: ${E_{vac}}^{corr}{}_{I}$
- 真空中のモノマー J の計算: ${E_{vac}}^{corr}{}_{J}$

を計算する必要がある。これらを用いた MP2 相関エネルギーにおける BSSE の値は

$$E^{corr-BSSE}{}_{IJ} = (E_{vac}{}^{corr}{}_{I\,(IJ)} - E_{vac}{}^{corr}{}_{I}) + (E_{vac}{}^{corr}{}_{J\,(IJ)} - E_{vac}{}^{corr}{}_{J}) \tag{3.14}$$

と見積もられる。よって、CP 補正された MP2–IFIE は

$$\Delta E^{MP2-CP}{}_{IJ} = \Delta E^{HF-CP}{}_{IJ} + (\Delta E^{corr}{}_{IJ} - E^{corr-BSSE}{}_{IJ}) \tag{3.15}$$

となる。計算されなければならない 4 つのエネルギーのうち、$E_{vac}{}^{corr}{}_{I}$ と $E_{vac}{}^{corr}{}_{J}$ はモノマー MP2 計算の際に計算され、$E_{vac}{}^{corr}{}_{I\,(IJ)}$ と $E_{vac}{}^{corr}{}_{J\,(IJ)}$ はダイマー MP2 計算の際に計算される。これらの計算において、HF 計算の場合と同様、シェル積分は共通に使用することが出来るが、異なる分子軌道に関して積分変換を行うので、積分変換は個別に行わなければならない。よって、この補正を行うと MP2 計算の時間が倍程度になる。ここまでに出てきた数値とプログラム中での変数名の対応関係を**表 4** にまとめる。

3.3 計算結果として出力される数値

＜モノマー計算終了後＞

- モノマー MP2 エネルギー 1 ： cmp2 (normal)

  通常の MP2 相関エネルギー。フラグメントごとに出力される。

- モノマー MP2 エネルギー 2 ： cmp2 (grimme)

  Grimme によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギー。フラグメントごとに出力される。

- モノマー MP2 エネルギー 3 ： cmp2 (jung)

  Jung によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギー。フラグメントごとに出力される。

- モノマー MP2 エネルギー 4 ： cmp2 (hill)

  Hill によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギー。フラグメントごとに出力される。

- FMO1–MP2 エネルギー 1 ： corr.

  通常の MP2 相関エネルギー。相関エネルギーのみの値と、FMO1–RHF のエネルギーを加えた値が出力される。

- FMO1–MP2 エネルギー 2 ： Grimm's scs

  Grimme によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギー。相関エネルギーのみの値と、FMO1–RHF のエネルギーを加えた値が出力される。

- FMO1–MP2 エネルギー 3 ： Jung's scs

  Jung によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギー。相関エネルギーのみの値と、FMO1–RHF のエネルギーを加えた値が出力される。

- FMO1–MP2 エネルギー 4 ： Hill's scs

  Hill によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギー。相関エネルギーのみの値と、FMO1–RHF のエネルギーを加えた値が出力される。

＜ダイマー計算終了後＞

- MP2–IFIE 1 ： normal (not scs)

  IFIE に対する MP2 相関エネルギーの寄与。通常の RI–MP2 計算の結果を用いる。CP 補正を行った場合は、BSSE の見積もり値を引いたものも出力される。

- MP2–IFIE 2 ： Grimme's scs

  IFIE に対する MP2 相関エネルギーの寄与。Grimme によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギーが使われる。CP 補正を行った場合は、BSSE の見積もり値を引いたものも出力される。

- MP2–IFIE 3 ： Jung's scs

  IFIE に対する MP2 相関エネルギーの寄与。Jung によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギーが使われる。CP 補正を行った場合は、BSSE の見積もり値を引いたものも出力される。

- MP2–IFIE 4 ： Hill's scs

  IFIE に対する RI–MP2 相関エネルギーの寄与。Jung によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギーが使われる。CP 補正を行った場合は、BSSE の見積もり値を引いたものも出力される。

- FMO2–MP2 エネルギー 1 ： corr.

  通常の MP2 相関エネルギー。相関エネルギーのみの値と、FMO2–RHF のエネルギーを加えた値が出力される。

- FMO2–MP2 エネルギー ： Grimm's scs

  Grimme によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギー。相関エネルギーのみの値と、FMO2–RHF のエネルギーを加えた値が出力される。

- FMO2–MP2 エネルギー : Jung's scs

  Jung によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギー。相関エネルギーのみの値と、FMO2–RHF のエネルギーを加えた値が出力される。

- FMO2–MP2 エネルギー ： Hill's scs

  Hill によって提案されたスケーリング定数を用いた SCS–MP2 相関エネルギー。相関エネルギーのみの値と、FMO2–RHF のエネルギーを加えた値が出力される。

表 4: グローバル変数との対応（FMO−MP2 エネルギー関連）。SCS–MP2 の値を算出するために、スピン平行からの寄与とスピン反平行からの寄与に分けて記録している。各数値のうち、「 _s 」で終わっている変数がシングレットカップリングペアからの寄与で、「 _t 」で終わっている変数がトリプレットカップリングペアからの寄与。

変数名	次元数	値
cmp2_mon_energy_s	フラグメント数	$E^{corr(MP2)}{}_I$
cmp2_mon_energy_t	フラグメント数	$E^{corr(MP2)}{}_I$
cmp2_mon_energy_vac_cp_s	フラグメント数	${E_{vac}^{corr}}_I$
cmp2_mon_energy_vac_cp_t	フラグメント数	${E_{vac}^{corr}}_I$
cmp2_dim_ifie_s	フラグメントペア数	$\Delta E^{corr(MP2)}{}_{IJ}$
cmp2_dim_ifie_t	フラグメントペア数	$\Delta E^{corr(MP2)}{}_{IJ}$
cmp2_dim_ifie_cp_s	フラグメントペア数	$E^{corr-BSSE}{}_{IJ}$
cmp2_dim_ifie_cp_t	フラグメントペア数	$E^{corr-BSSE}{}_{IJ}$
cmp2_fmo_1_s	1	$E^{corr(MP2)}{}_{fmo1}$
cmp2_fmo_1_t	1	$E^{corr(MP2)}{}_{fmo1}$
cmp2_fmo_2_s	1	$E^{corr(MP2)}{}_{fmo2}$
cmp2_fmo_2_t	1	$E^{corr(MP2)}{}_{fmo2}$

4　FMO–RI–MP2 計算（エネルギー）

4.1　エネルギー

FMO–MP2 計算におけるフラグメントおよびフラグメントペアのエネルギー計算を、RI–MP2 で実行するだけでよい。プログラム中での変数名の対応関係を**表 5** にまとめる。

4.2　BSSE の見積もり

FMO–MP2 計算と全く同様に見積もることができる。プログラム中での変数名の対応関係を**表 5** にまとめる。

4.3　計算結果として出力される数値

＜モノマー計算終了後＞

- モノマー RI–MP2 エネルギー 1 ： ri–cmp2 (normal)

  通常の RI–MP2 相関エネルギー。フラグメントごとに出力される。

- モノマー RI–MP2 エネルギー 2 ： ri–cmp2 (grimme)

  Grimme によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギー。フラグメントごとに出力される。

- モノマー RI–MP2 エネルギー 3 ： ri–cmp2 (jung)

  Jung によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギー。フラグメントごとに出力される。

- モノマー RI–MP2 エネルギー 4 ： ri–cmp2 (hill)

  Hill によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギー。フラグメントごとに出力される。

- FMO1–RI–MP2 エネルギー 1 ： corr.

  通常の RI–MP2 相関エネルギー。相関エネルギーのみの値と、FMO1–RHF のエネルギーを加えた値が出力される。

- FMO1–RI–MP2 エネルギー 2 ： Grimm's scs

  Grimme によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギー。相関エネルギーのみの値と、FMO1–RHF のエネルギーを加えた値が出力される。

- FMO1–RI–MP2 エネルギー 3 ： Jung's scs

  Jung によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギー。相関エネルギーのみの値と、FMO1–RHF のエネルギーを加えた値が出力される。

- FMO1–RI–MP2 エネルギー 4 ： Hill's scs

  Hill によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギー。相関エネルギーのみの値と、FMO1–RHF のエネルギーを加えた値が出力される。

＜ダイマー計算終了後＞

- MP2–RI–IFIE ： normal (not scs)

  IFIE に対する RI–MP2 相関エネルギーの寄与。通常の RI–MP2 計算の結果を用いる。CP 補正を行った場合は、BSSE の見積もり値を引いたものも出力される。

- MP2–RI–IFIE ： Grimme's scs

  IFIE に対する RI–MP2 相関エネルギーの寄与。Grimme によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギーが使われる。CP 補正を行った場合は、BSSE の見積もり値を引いたものも出力される。

- MP2–RI–IFIE ： Jung's scs

  IFIE に対する RI–MP2 相関エネルギーの寄与。Jung によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギーが使われる。CP 補正を行った場合は、BSSE の見積もり値を引いたものも出力される。

- MP2–RI–IFIE ： Hill's scs

  IFIE に対する RI–MP2 相関エネルギーの寄与。Jung によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギーが使われる。CP 補正を行った場合は、BSSE の見積もり値を引いたものも出力される。

- FMO2–RI–MP2 相関エネルギー ： corr.

  通常の RI–MP2 相関エネルギー。相関エネルギーのみの値と、FMO2–RHF のエネルギーを加えた値が出力される。

- FMO2–RI–MP2 相関エネルギー ： Grimm's scs

  Grimme によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギー。相関エネルギーのみの値と、FMO2–RHF のエネルギーを加えた値が出力される。

- FMO2–RI–MP2 相関エネルギー : Jung's scs

  Jung によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギー。相関エネルギーのみの値と、FMO2–RHF のエネルギーを加えた値が出力される。

- FMO2–RI–MP2 相関エネルギー ： Hill's scs

  Hill によって提案されたスケーリング定数を用いた SCS–RI–MP2 相関エネルギー。相関エネルギーのみの値と、FMO2–RHF のエネルギーを加えた値が出力される。

表 5: グローバル変数との対応（FMO–RI–MP2 のエネルギー関連）。SCS–RI–MP2 の値を算出するために、スピン平行からの寄与とスピン反平行からの寄与に分けて記録している。各数値のうち、「 _s 」で終わっている変数がシングレットカップリングペアからの寄与で、「 _t 」で終わっている変数がトリプレットカップリングペアからの寄与。

変数名	次元数	値
ri_cmp2_mon_energy_s	フラグメント数	$E^{corr(MP2)}{}_I$
ri_cmp2_mon_energy_t	フラグメント数	$E^{corr(MP2)}{}_I$
ri_cmp2_mon_energy_vac_cp_s	フラグメント数	$E_{vac}^{corr}{}_I$
ri_cmp2_mon_energy_vac_cp_t	フラグメント数	$E_{vac}^{corr}{}_I$
ri_cmp2_dim_ifie_s	フラグメントペア数	$\Delta E^{corr(MP2)}{}_{IJ}$
ri_cmp2_dim_ifie_t	フラグメントペア数	$\Delta E^{corr(MP2)}{}_{IJ}$
ri_cmp2_dim_ifie_cp_s	フラグメントペア数	$E^{corr-BSSE}{}_{IJ}$
ri_cmp2_dim_ifie_cp_t	フラグメントペア数	$E^{corr-BSSE}{}_{IJ}$
ri_cmp2_fmo_1_s	1	$E^{corr(MP2)}{}_{fmo1}$
ri_cmp2_fmo_1_t	1	$E^{corr(MP2)}{}_{fmo1}$
ri_cmp2_fmo_2_s	1	$E^{corr(MP2)}{}_{fmo2}$
ri_cmp2_fmo_2_t	1	$E^{corr(MP2)}{}_{fmo2}$

5　FMO–LMP2 計算（エネルギー）

5.1　エネルギー

FMO1–LMP2 エネルギーおよび FMO2–LMP2 エネルギーは、FMO–MP2 計算におけるフラグメントおよびフラグメントペアのエネルギー計算を、LMP2 で実行するだけでよい。よって、分子全体の FMO1–LMP2 エネルギーおよび FMO2–LMP2 エネルギーは、モノマーの LMP2 エネルギー（${E^{LMP2}}_I$）とダイマーの LMP2 エネルギー（${E^{LMP2}}_{IJ}$）から

$$ {E^{LMP2}}_{fmo1} = \sum_I {E^{LMP2}}_I \tag{5.1} $$

$$ {E^{LMP2}}_{fmo2} = \sum_{I>J} {E^{LMP2}}_{IJ} \ - (N-2) \sum_I {E^{LMP2}}_I \tag{5.2} $$

のように計算される。ここで、

$$ {E^{LMP2}}_I = {E^{HF}}_I + {E^{corr(LMP2)}}_I \tag{5.3} $$

$$ {E^{LMP2}}_{IJ} = {E^{HF}}_{IJ} + {E^{corr(LMP2)}}_{IJ} \tag{5.4} $$

であり、${E^{corr(LMP2)}}_I$ および ${E^{corr(LMP2)}}_{IJ}$ は、モノマーおよびダイマーの MP2 相関エネルギーである。よって、(5.1) および (5.2) は

$$ {E^{LMP2}}_{fmo1} = {E^{HF}}_{fmo1} + \sum_I {E^{corr(LMP2)}}_I \tag{5.5} $$

$$ {E^{LMP2}}_{fmo2} = {E^{HF}}_{fmo2} + \left(\sum_{I>J} {E^{corr(LMP2)}}_{IJ} \ - (N-2) \sum_I {E^{corr(LMP2)}}_I \right) \tag{5.6} $$

と書ける。さらに (5.6) は、

$$ {E^{LMP2}}_{fmo2} = {E^{HF}}_{fmo2} + \left(\sum_I {E^{corr(LMP2)}}_I + \sum_{I>J} {\Delta E_{\rm sub}}^{corr(LMP2)}{}_{IJ} \right) \tag{5.7} $$

となる。ここで、

$$\Delta E_{\rm sub}{}^{corr(LMP2)}{}_{IJ} = E^{corr(LMP2)}{}_{IJ} - E^{corr(LMP2)}{}_{I} - E^{corr(LMP2)}{}_{J} \tag{5.8}$$

である。さらに、

$$E^{corr(LMP2)}{}_{fmo1} = \sum_{I} E^{corr(LMP2)}{}_{I} \tag{5.9}$$

$$E_{\rm sub}{}^{corr(LMP2)}{}_{fmo2} = \sum_{I} E^{corr(MP2)}{}_{I} + \sum_{I>J} \Delta E_{\rm sub}{}^{corr(LMP2)}{}_{IJ} \tag{5.10}$$

と定義すると、FMO1–LMP2 エネルギーおよび FMO2–LMP2 エネルギーは以下のように記述できる。

$$E^{LMP2}{}_{fmo1} = E^{HF}{}_{fmo1} + E^{corr(LMP2)}{}_{fmo1} \tag{5.11}$$

$$E^{LMP2}{}_{fmo2} = E^{HF}{}_{fmo2} + E_{\rm sub}{}^{corr(LMP2)}{}_{fmo2} \tag{5.12}$$

また、IFIE は HF 計算で得られた IFIE に電子相関による補正を加える形で、

$$\Delta E^{LMP2}{}_{IJ} = \Delta E^{HF}{}_{IJ} + \Delta E_{\rm sum}{}^{corr(LMP2)}{}_{IJ} \tag{5.13}$$

のように記述される。ここで、$\Delta E_{\rm sum}{}^{corr(LMP2)}{}_{IJ}$ は、ダイマー LMP2 計算で得られるペア相関エネルギーから、以下のように計算される。

$$\Delta E_{\rm sum}{}^{corr(LMP2)}{}_{IJ} = \sum_{i\in I}\sum_{j\in J} \epsilon_{ij} \tag{5.14}$$

ここで、ϵ_{ij} がダイマー LMP2 計算で得られるペア相関エネルギーである（この式が示すように、IFIE に対する電子相関の補正が局在化軌道のペアに分割された形で与えられる）。また、

$$E_{\rm sum}{}^{corr(LMP2)}{}_{fmo2} = \sum_{I} E^{corr(MP2)}{}_{I} + \sum_{I>J} \Delta E_{\rm sum}{}^{corr(LMP2)}{}_{IJ} \tag{5.15}$$

と定義し、(5.12) で $E_{\rm sub}{}^{corr(LMP2)}{}_{fmo2}$ の代わりに使うことも考えられる。この場合、ダイマーの結果からモノマーの結果を引いて 2 体補正を求めるのではなく、ダイマーのペア相関エネルギーから 2 体補正を求めるので、結果的に FMO2–LMP2 エネルギーが異なる。ここまでに出てきた数値とプログラム中での変数名の対応関係を**表 6** にまとめる。

5.2 BSSE の見積もり

局在化 MP2 では BSSE の補正は行わない。

5.3 計算結果として出力される数値

＜モノマー計算終了後＞

- モノマー LMP2 エネルギー ： lmp2

  LMP2 相関エネルギー。フラグメントごとに出力される。

- FMO1-LMP2 相関エネルギー ： corr.

  LMP2 相関エネルギー。相関エネルギーのみの値と、FMO1–RHF エネルギーを加えられたものが出力される。

- LMP2–IFIE ： lmp2

  IFIE に対する LMP2 相関エネルギーの寄与。

- FMO2-LMP2 相関エネルギー 1 ： corr. (sub.)

  LMP2 相関エネルギー。相関エネルギーのみの値と、FMO2–RHF エネルギーを加えた値が出力される。

$$E_{\mathrm{sub}}{}^{corr(LMP2)}{}_{fmo2} \tag{5.16}$$

- FMO2-LMP2 相関エネルギー 2 ： corr. (sum.)

  LMP2 相関エネルギー。相関エネルギーのみの値と、FMO2–RHF エネルギーを加えた値が出力される。

$$E_{\mathrm{sum}}{}^{corr(LMP2)}{}_{fmo2} \tag{5.17}$$

表 6: グローバル変数との対応（FMO–LMP2 エネルギー関連）。

変数名	次元数	値
lmp2_mon_energy	フラグメント数	${E^{corr(LMP2)}}_I$
lmp2_dim_ifie_sub	フラグメントペア数	$\Delta {E_{\mathrm{sub}}}^{corr(LMP2)}{}_{IJ}$
lmp2_dim_ifie_sum	フラグメントペア数	$\Delta {E_{\mathrm{sum}}}^{corr(LMP2)}{}_{IJ}$
lmp2_fmo_1	1	${E^{corr(LMP2)}}_{fmo1}$
lmp2_fmo_2_sub	1	${E_{\mathrm{sub}}}^{corr(LMP2)}{}_{fmo2}$
lmp2_fmo_2_sum	1	${E_{\mathrm{sum}}}^{corr(LMP2)}{}_{fmo2}$

グローバル変数

1 マクロ定義

- PARA_MPI_MAX_PROC **1024**

  並列化に使用できるプロセッサー数（コア数）の最大値。

- MEM_MAX_VAL **1024**

  memory_alloc 関数を用いて同時に allocate でき配列の種類の最大数。

- ANG_L6_MAX **12**

  角運動量のパラメータ（ang_l6max）のデフォルト値。

- ANG_L5_MAX **4**

  角運動量のパラメータ（ang_l5max）のデフォルト値。

- ANG_L6_XYZ_MAX **91**

  角運動量のパラメータ（angl_l6xyz_max）のデフォルト値。

- CONST_MAX_IATOM **112**

  プログラムで定義されている原子番号の数。0（ ダミー原子 ）~111 までの 112。

- BDEF_MAX **128**

  基底関数の定義に関するパラメータ（bdef_max）のデフォルト値。

- BDEF_MAX_SH **40**

  基底関数の定義に関するパラメータ（bdef_max_sh）のデフォルト値。

- BDEF_MAX_NC **30**

  基底関数の定義に関するパラメータ（bdef_max_nc）のデフォルト値。

- BDEF_MAX_L **4**

  基底関数の定義に関するパラメータ（bdef_max_l）のデフォルト値。

- PRJ_MAX_FRAG_ORB **16**

  射影演算子の作成に関わるパラメータのデフォルト値。

- PRJ_MAX_NBA **4**

  射影演算子の作成に関わるパラメータのデフォルト値。

- PRJ_MAX_NBO　**64**

  射影演算子の作成に関わるパラメータのデフォルト値。

- FRAG_MAX_LINK　**8**

  射影演算子の作成に関わるパラメータのデフォルト値。

2 並列化制御

- **int** para_mpi_nproc

  MPI 並列の全プロセッサー数（MPI_COMM_WORLD のプロセッサー数）を保持している変数。initialize_mpi 関数で以下のように値がセットされる。

```
MPI_Comm_size(MPI_COMM_WORLD,&para_mpi_nproc);
```

- **int** para_mpi_my_rank

  MPI_COMM_WORLD におけるランクを保持している変数。initialize_mpi 関数で以下のように値がセットされる。

```
MPI_Comm_rank(MPI_COMM_WORLD,&para_mpi_my_rank);
```

- **int** para_mpi_print_rank

  標準出力にプリントアウトするプロセッサーの MPI_COMM_WORLD におけるランクを保持している変数。initialize_mpi 関数で 0 がセットされる。入力ファイルから指定できる。

- **MPI_Comm** para_mpi_single_comm

  各プロセッサーが単独のグループを形成するコミュニケーターを保持している変数。initialize_mpi 関数で以下のように値がセットされる。

```
my_key=para_mpi_my_rank/1;
MPI_Comm_split(MPI_COMM_WORLD,my_key,para_mpi_my_rank,&para_mpi_single_comm);
```

- **char** para_mpi_proc_name [PARA_MPI_MAX_PROC] [128]

  各プロセッサーの名前を保持している変数。マシンのネットワーク上のアドレスが入る。名前は 127 文字まで。PARA_MPI_MAX_PROC はマクロ置換される。initialize_mpi 関数で以下のように初期化される。

```
MPI_Get_processor_name(&para_mpi_proc_name[para_mpi_my_rank][0],&itmp);
for(i=0;i<para_mpi_nproc;i++){
  MPI_Bcast(&para_mpi_proc_name[i][0],128,MPI_CHAR,i,MPI_COMM_WORLD);
}
```

- **int** para_mpi_np_scc

  モノマー SCC 計算で、各フラグメントの計算をいくつの CPU で実行するかを保持している変数。全 CPU 数（para_mpi_nproc）の約数になる。入力リストで指定可能。

- **int** para_mpi_np_mon

  モノマー計算で、各フラグメントの計算をいくつの CPU で実行するかを保持している変数。全 CPU 数（para_mpi_nproc）の約数になる。入力リストで指定可能。

- **int** para_mpi_np_dim

  ダイマー計算で、フラグメントペアの計算をいくつの CPU で実行するかを保持している変数。全 CPU 数（para_mpi_nproc）の約数になる。入力リストで指定可能。

3 メモリー制御

- **double *** mem_core ([mem_size])

  実行時に確保されたメモリ領域の先頭ポインタを保持する変数。initialize_memory 関数で以下のように初期化される。

```
mem_core = ( double * ) malloc ( mem_size * 8 ) ;
```

- **long int** mem_size

  実行時に確保されたメモリが double 型で何個分かを保持している変数。initialize_memory 関数で値が代入される。32bit 環境の場合、long int は「2 147 483 647」までカウント出来るので、

```
2 147 483 647 * 8 / 1024 / 1024 / 1024 = 15.99999 GByte
```

  となり、1CPU 当たり 16GByte のメモリまで対応。64bit 環境の場合は実質的にメモリの制限は無い。

- **int** mem_mbyte

  実行時に確保されたメモリが何 MByte かを保持している変数。mem_size とは以下の関係にある。

```
mem_size = mem_mbyte * ( 1024 * 1024 / 8 ) ;
```

  入力の「mem_mbyte」キーワードで指定した値が initialize_memory 関数でセットされる。

- **int** mem_nval

  mem_core に割り当てられた配列の数を保持している変数。MEM_MAX_VAL 個まで割り当て可能。

- **char** max_val_name [MEM_MAX_VAL] [40]

  mem_core に割り当てられた配列の「名前」を保持する配列。名前は 39 文字まで。mem_core に割り当てられた配列を識別するために使用される。MEM_MAX_VAL はマクロ置換される。

- **long int** mem_val_size [MEM_MAX_VAL]

  mem_core に割り当てられた各配列のサイズを保持する配列。double 型で何個分かを保持している。EM_MAX_VAL はマクロ置換される。

4 プログラム全体の制御

- **int** cntrl_run_type

  計算のタイプを指定する変数。入力ファイルから指定可能。read_input_list_01 関数でセット。以下のように、整数で指定する。

  ```
  0 : 単一構造の計算
  1 : 構造最適化
  2 : 分子動力学計算
  ```

  デフォルトは 0。

- **int** cntrl_lprint_pcs

  PAICS 全般の出力フラグ。整数で指定される。

  - 0 :
    何も出力しない。
  - 1 :
    計算開始時のタイトル、全体の時間のプロファイル、計算終了時のタイトルが出力される。
  - 2 :
    上の出力に追加して、メモリの情報、フラグメントの情報、フラグメントペアの情報が出力される。

- **int** cntrl_lprint_inp

- **int** cntrl_lprint_prj

- **int** cntrl_lprint_scc

  モノマー SCC 計算の出力フラグ。整数で指定される。

  - 0 :
    何も出力しない。
  - 1 :
    イタレーションのプロセスを出力する。
  - 2 :
    上の出力に追加して、モノマー SCC 計算の結果を出力する。
  - 3 :
    上の出力に追加して、モノマー RHF 計算のプロセスを出力する。

- **int** cntrl_lprint_mon

  モノマー計算の出力フラグ。整数で指定される。

  - 0 :
    何も出力しない。
  - 1 :
    計算されているモノマーの番号が出力される。
  - 2 :
    上の出力に追加して、モノマー計算の結果を出力する。
  - 3 :
    上の出力に追加して、モノマー計算のプロセスを出力する。

- **int** cntrl_lprint_des

- **int** cntrl_lprint_dim

- **int** cntrl_coord_unit

  座標の入力の際の単位の指定

  ```
  0 : bohr
  1 : angstrom
  ```

  read_input_list_01 関数でセットされる。デフォルトは 0（つまり、何も指定しなければ、bohr 単位で座標を入力することになる）。

- **int** cntrl_w_log_file

  「log ファイル」への書き出しを行うか否か。

  ```
  0 : 書き出さない
  1 : 書き出す
  ```

  デフォルトは 0。

- **int** cntrl_r_scc

  あらかじめ書き出された電子密度を、モノマー SCC 計算の初期電子密度として読むか否か。

  ```
  0 : 読まない。
  1 : 読む。
  ```

デフォルトは 0。

- **int** cntrl_w_scc

  収束したモノマー SCC の電子密度をファイルに書き出すか否か。

  ```
  0 : 書き出す。
  1 : 書き出さない。
  ```

  デフォルトは 0。cntrl_w_result_file で指定される文字列に「.scc」を追加した名前のファイルへ書き出す。

- **char** * cntrl_w_result_file [1024]

  入力に関連するファイルの名前に使われる文字列。

- **char** cntrl_r_result_file [1024]

  入力に関連するファイルの名前に使われる文字列。

- **char** cntrl_w_log_file_name [1024]

  「log ファイル」のファイル名。cntrl_w_result_file の文字列から自動的に決められる（cntrl_w_result_file の文字列に「_[mpi-rnak].log」を追加したものがファイル名となる）。

- **char** cntrl_w_scc_file_name [1024]

  モノマー SCC の電子密度が出力されるファイル名。cntrl_r_result_file の文字列から自動的に決められる（cntrl_w_result_file の文字列に「.scc」を追加した文字列がファイル名となる）。

- **char** cntrl_r_scc_file_name [1024]

  モノマー SCC の電子密度を読み込むファイル名。cntrl_r_result_file の文字列から自動的に決められる（cntrl_r_result_file の文字列に「.scc」を追加した文字列がファイル名となる）。

- **double** cntrl_scc_time1

  モノマー SCC 計算に掛かった時間（秒）。

- **double** cntrl_mon_time1

  モノマー計算に掛かった時間（秒）。

- **double** cntrl_dim_time1

  ダイマー計算に掛かった時間（秒）。

- **double** cntrl_des_time1

  ダイマー ES 計算に掛かった時間（秒)。

- **double** cntrl_total_time1

  計算の開始時間（ システム時間)。

- **double** cntrl_total_time2

  計算の終了時間（ システム時間)。

5 軌道角運動量パラメータ

- **int** ang_l6max = ANG_L6_MAX

  角運動量に関するパラメータの 1 つで、最大の軌道角運動量の値。4 中心積分が可能なのは、この値の半分の軌道角運動量まで。ANG_L6_MAX はマクロ置換される。これらの値は、積分ルーチンで使用される。

- **int** ang_l6xyz_max = ANG_L6_XYZ_MAX

  角運動量に関するパラメータの 1 つで、カルテシアン表示の「指数の組み合わせ」の最大値。ANG_L6_XYZ_MAX はマクロ置換される。これらの値は、積分ルーチンで使用される。

- **int** * ang_l6n（ [ang_l6max+1] ）

  角運動量に関するパラメータの 1 つで、各角運動量に関する原子軌道の「カルテシアン表示における指数の組み合わせ」が格納されている配列。これらの値は、積分ルーチンで使用される。この配列に格納されている数値を**表 1** に記載する。

- **int** * ang_l6sum（ [ang_l6max+1] ）

  角運動量に関するパラメータの 1 つで、各角運動量の原子軌道までの「カルテシアン表示における指数の組み合わせ」を合計した数を保持している。これらの値は、積分ルーチンで使用される。この配列に格納されている数値を**表 1** に記載する。例えば、軌道角運動量が l の原子軌道の「全体の通し番号」によるループは以下のようになる。

```
int l1 = ang_l6sum [l] - ang_l6n [l]
int l2 = ang_l6sum [l]
for ( i=l1 ; i<l2         ; i++ ) { ... }
```

- **int** * ang_l6xyz（ [ang_l6max+1] [ang_l6xyz_max] [3] ）

  角運動量に関するパラメータの 1 つで、カルテシアン表示における「x の指数」「y の指数」「z の指数」を保持している配列。これらの値は、積分ルーチンで使用される。軌道角運動量が l の i 番目の軌道の指数は、以下のように参照される。

```
x の指数 : ang_l6xyz [ ( ang_l6xyz_max * 3 ) * l  + 3 * i + 0 ]
y の指数 : ang_l6xyz [ ( ang_l6xyz_max * 3 ) * l  + 3 * i + 1 ]
z の指数 : ang_l6xyz [ ( ang_l6xyz_max * 3 ) * l  + 3 * i + 2 ]
```

- **int** * ang_l6xyz_idx（ [ang_l6max+1] [ang_l6max+1] [ang_l6max+1] ）

  角運動量に関するパラメータの 1 つで、カルテシアン表示における「x の指数」「y の指数」「z の指数」を指定し、対応する軌道が全体の通し番号で何番目かを返す。つまり、ang_l6xyz_xyz の逆引き。これらの値は、積分ルーチンで使用される。例えば、

```
int p01=(ang_l6max+1)*(ang_l6max+1);
int p02=(ang_l6max+1);

ang_l6xyz_idx [ p01 * 0 + p02 * 0 + 0 ] → 0
ang_l6xyz_idx [ p01 * 1 + p02 * 0 + 0 ] → 1
ang_l6xyz_idx [ p01 * 2 + p02 * 0 + 0 ] → 4
```

- **int** ang_l5max = ANG_L5_MAX

  角運動量に関するパラメータの球面調和型における最大の角運動量の値。カルテシアン型から球面調和型に変換できる角運動量の最大値でもある。これらの値は、積分ルーチンで使用される。ANG_L5_MAX はマクロ置換される。

- **int *** ang_l5n（ [ang_l5max+1] ）

  プログラムが保持しているパラメータの 1 つで、各角運動量の球面調和型の軌道数が格納されている配列。これらの値は、積分ルーチンで使用される。この配列に格納されている数値を **表 2** に記載する。

- **int * ang_l5sum（ [ang_l5max+1] ）**

  角運動量に関するパラメータの 1 つで、各角運動量の原子軌道までの「球面調和型表示における指数の組み合わせ」を合計した数を保持している。これらの値は、積分ルーチンで使用される。この配列に格納されている数値を **表 2** に記載する。

表 1: ang_l6n と ang_l6sum に格納されている値

角運動量	ang_l6n	ang_l6sum
0	1	1
1	3	4
2	6	10
3	10	20
4	15	35
5	21	56
6	28	84
7	36	120
8	45	165
9	55	220
10	66	286
11	78	364
12	91	455

表 2: sys_l5n と sys_l5sum に格納されている値

角運動量	ang_l5n	ang_l5sum
0	1	1
1	3	4
2	5	9
3	7	16
4	9	25

6 定数

- **double** const_bohr = 0.5291772108

  ボーア半径。

- **double** const_vdw_radius [CONST_MAX_IATOM]

  各原子のファンデルワールス半径。原子番号 0〜111 番まで。0 はダミー原子用。CONST_MAX_IATOM はマクロ置換される。

- **int** const_atomic_nfzc [CONST_MAX_IATOM]

  各原子のフローズンコア数。原子番号は 0〜111 番まで。0 はダミー原子用。「CONST_MAX_IATOM」はマクロ置換される。

- **char** const_atomic_label [CONST_MAX_IATOM] [4]

  原子のラベルが格納されている。「CONST_MAX_IATOM」はマクロ置換される。

7　誤差関数

- **int** fmt_mmax = 17

  誤差関数の値を 4 項展開で求めるか級数展開で求めるかの閾値を記録している「m」の最大値。実際に計算できる「m」の値がこの数値。従って、4 中心積分では g 関数の 1 次微分までとなる。

- **int** fmt_mt = 20

  誤差関数の値を 4 項展開で求めるために保持している数値テーブルの「m」の値。必要な「m」の値 +3 までのテーブルが必要となる。

- **int** fmt_nt = 158638

  各「m」の数値テーブルに保持している値の数。

- **double** fmt_dt = 4.8828125E-4

  この値ごとに数値テーブルを保持している。

- **double** * fmt_table

  数値テーブルを保持している配列。

- **double** fmt_tf [18]

  誤差関数の値を計算する際に、4 項展開から級数展開に切り替える閾値を保持している配列。この値より小さな場合は 4 項展開で計算される。基本的にはこの値さえ決めれば、計算できる「m」の値を大きくすることが出来る。

8 電子反発積分

- **double** eri_tv

  電子反発積分のループでスクリーニングに使用される閾値。積分値に対してこの値よりも小さな寄与しかしないと思われる計算は行わない。デフォルトは 1.0E−12 で read_input_list_01 関数でセットされる。入力リストの「eri_tv」キーワードで変更可能。

- **double** eri_cauchy_tv

  コーシー・シュバルツの不等式で積分を実行するかどうか決める際の閾値。不等式の判定でこれよりも小さい場合は積分計算を行わない。デフォルトは 1.0E−10 で read_input_list_01 関数でセットされる。入力リストの「eri_cauchy_tv」キーワードで変更可能。

- **double** eri_grad_tv

  電子反発積分の微分のループでスクリーニングに使用される閾値。積分の微分値に対してこの値よりも小さな寄与しかしないと思われる計算は行わない。デフォルトは 1.0E−12 で read_input_list_01 関数でセットされる。入力リストの「eri_grad_tv」キーワードで変更可能。

- **double** eri_use_gen

  電子反発積分の計算で、一般ルーチンを使用するか否か。以下のように整数で指定される。

  ```
  0 : 使用しない
  1 : 使用する
  ```

  デフォルト値は 0。read_input_list_01 関数でセットされる。入力リストの「eri_use_gen」で変更可能。通常は特化ルーチンを使用するので、デバッグの目的以外で使用することは無い。

- **double** eri_3cen_use_gen

  3 中心積分の計算で、一般ルーチンを使用するか否か。以下のように整数で指定する。

  ```
  0 : 使用しない
  1 : 使用する
  ```

  デフォルト値は 0。read_input_list_01 関数でセットされる。入力リストの「eri_use_gen」で変更可能。通常は特化ルーチンを使用するので、デバッグの目的以外で使用することは無い。

- **double** eri_grad_use_gen

  電子反発積分の微分の計算で、一般ルーチンを使用するか否か。以下のように整数で指定する。

0 : 使用しない
1 : 使用する

デフォルト値は 0。read_input_list_01 関数でセットされる。入力リストの「eri_use_gen」で変更可能。通常は特化ルーチンを使用するので、デバッグの目的以外で使用することは無い。

9 基底関数の定義

- **int** bdef_max = BDEF_MAX

  「基底関数の定義」の最大数。「BDEF_MAX」はマクロ置換される。

- **int** bdef_max_sh = BDEF_MAX_SH

  「基底関数の定義」におけるシェルの数の最大値。例えば、炭素原子の cc-pVDZ では 6（sssppd）となる値。「BDEF_MAX_SH」はマクロ置換される。

- **int** bdef_max_nc = BDEF_MAX_NC

  「基底関数の定義」における短縮数の最大値。例えば、炭素原子の cc-pVDZ では 8 となるり、6-31G では 6 となる値。「BDEF_MAX_NC」はマクロ置換される。

- **int** bdef_max_l = BDEF_MAX_L

  「基底関数の定義」における角運動量の最大値。積分計算のプログラムと誤差関数のプログラムに依存して決まる。「BDEF_MAX_L」はマクロ置換される。

- **int** bdef_n

  読み込んだ「基底関数の定義」の数を格納している変数。

- **char** * bdef_name ([bdef_max * 21])

  読み込んだ「基底関数の定義」の名前（20 文字以内）を格納している配列。read_basis_define 関数で確保され、read_basis_define_file 関数でセットされる。

- **int** * bdef_nsh ([bdef_max])

  読み込んだ「基底関数の定義」のシェルの数を格納している配列。read_basis_define 関数で確保され、read_basis_define_file 関数でセットされる。

- **int** * bdef_nbo ([bdef_max])

  読み込んだ「基底関数の定義」の nbo を格納している配列。read_basis_define 関数で確保され、read_basis_define_file 関数でセットされる。球面調和型の場合とカルテシアン型の場合で異なる値がセットされる。例えば、炭素原子の cc-pVDZ の場合は 14（1+1+1+3+3+5）となる。

- **int** * bdef_sh_nc ([bdef_max * bdef_max_sh])

  読み込んだ「基底関数の定義」のシェルの短縮数を格納している配列。read_basis_define 関数で確保され、read_basis_define_file 関数でセットされる。i 番目の基底関数の定義の j 番目のシェルの短縮数は、以下のように参照される。

```
bdef_sh_nc [ bdef_max_sh * i + j ]
```

- **int** * bdef_sh_l ([bdef_max * bdef_max_sh])

  読み込んだ「基底関数の定義」のシェルの角運動量を格納している配列。read_basis_define 関数で確保され、read_basis_define_file 関数でセットされる。i 番目の基底関数の定義の j 番目のシェルの角運動量の値は、以下のように参照される。

```
bdef_sh_l [ bdef_max_sh * i + j ]
```

- **double** * bdef_sh_expon ([bdef_max * bdef_max_sh * bdef_max_nc])

  読み込んだ「基底関数の定義」の原始ガウス関数の指数の値を格納している配列。read_basis_define 関数で確保され、read_basis_define_file 関数でセットされる。i 番目の基底関数の定義の j 番目のシェルの k 番目の原始ガウス関数の指数の値は、以下のように参照される。

```
bdef_sh_expon [ bdef_max_sh * bdef_max_nc * i + bdef_max_nc * j + k ]
```

- **double** * bdef_sh_coeff ([bdef_max * bdef_max_sh * bdef_max_nc])

  読み込んだ「基底関数の定義」の原始ガウス関数の短縮係数を格納している配列。read_basis_define 関数で確保され、read_basis_define_file 関数でセットされる。i 番目の基底関数の定義の j 番目のシェルの k 番目の原始ガウス関数の係数の値は、以下のように参照される。

```
bdef_sh_coeff [ bdef_max_sh * bdef_max_nc * i + bdef_max_nc * j + k ]
```

10 基底関数

- **int** basis_spher

  計算に用いられる基底関数のタイプ。以下のように整数で指定される。

  ```
  0 : カルテシアン型
  1 : 球面調和型
  ```

  read_basis 関数でセットされる。全基底関数で共通に参照されるので、ある基底関数だけカルテシアン型といったような計算は出来ない。

- **int** basis_n

  基底関数の数。read_basis 関数でセットされる。必ずしも原子核の数と一緒とは限らない。

- **int** basis_nbo_total

  基底関数の全 nbo の数。basis_define_assign 関数でセットされる。

- **int** basis_max_nc

  基底関数のなかで最大の短縮の数。basis_define_assign 関数でセットされる。

- **int** basis_max_l

  基底関数のなかで最大の角運動量。basis_define_assign 関数でセットされる。

- **int** basis_aux

  RI 近似で使用する補助基底が正しく指定されていれば 1 が格納され、そうでなければ 0 が格納される。RI 近似を用いた計算が実行されない場合は 0 のままでも問題ない。

- **int** * basis_type ([basis_n])

  基底関数が「基底関数の定義」のうちの何番目かを格納する配列。read_basis 関数で確保され、basis_define_assign 関数でセットされる。***PAICS*** では、短縮係数や指数といった基底関数の具体的な情報は、「基底関数の定義」として保存されており、基底関数ごとに個別に保持しているわけではない。従って、基底関数が対応する「基底関数の定義」を参照し、短縮係数などの数値を取得する。具体的には、iba 番目の基底関数の各数値は以下のように参照される。

  ```
  bdef_nsh [ basis_type [ iba ] ]
  bdef_nbo [ basis_type [ iba ] ]
  ```

- **int** * basis_type_ini ([basis_n])

  初期軌道の計算に使用される基底関数が「基底関数の定義」のうちの何番目かを格納する配列。read_basis 関数で確保され、basis_define_assign 関数でセットされる。***PAICS*** では、最初に小さな基底関数で RHF 計算を行い、得られた軌道を指定された基底関数空間に射影し初期軌道としている。

- **int** * basis_type_aux ([basis_n])

  RI 近似の補助基底として使用される基底関数が「基底関数の定義」のうちの何番目かを格納する配列。

- **char** * basis_name ([basis_n * 21])

  基底関数の名前を格納している配列（最大 20 文字)。read_basis 関数で確保され、セットされる。「基底関数の定義」の名前がセットされる。

- **char** * basis_name_ini ([basis_n * 21])

  初期軌道の計算に使用される基底関数の名前を格納している配列（最大 20 文字)。

- **char** * basis_name_aux ([basis_n * 21])

  RI 近似の補助基底として使用される基底関数の名前を格納している配列（最大 20 文字)。

- **int** * basis_nucleus ([basis_n])

  何番目の原子核の上に置かれたものかを格納している配列（つまり、基底関数の番号と原子核の番号の対応関係を示す配列)。原子核の場所以外に置かれている場合は -1 が入る。

- **double** * basis_x ([basis_n])

  基底関数の x 座標の値。read_basis 関数で確保されセットされる。構造最適化ではイタレーションごとにアップデートされる。単一構造計算の場合は、basis_x_orig のアドレスが格納され、メモリ領域は確保されない。

- **double** * basis_y ([basis_n])

  基底関数の y 座標の値。read_basis 関数で確保されセットされる。構造最適化ではイタレーションごとにアップデートされる。単一構造計算の場合は、basis_y_orig のアドレスが格納され、メモリ領域は確保されない。

- **double** * basis_z ([basis_n])

  基底関数の z 座標の値。read_basis 関数で確保されセットされる。構造最適化ではイタレーションごとにアップデートされる。単一構造計算の場合は、basis_z_orig のアドレスが格納され、メモリ領域は確保されない。

- **double** * basis_x_orig ([basis_n])

  基底関数の x 座標の値。read_basis 関数で確保されセットされる。入力ファイルから読まれた値がそのまま格納される。構造最適化でもアップデートされない。

- **double** * basis_y_orig ([basis_n])

  基底関数の y 座標の値。read_basis 関数で確保されセットされる。入力ファイルから読まれた値がそのまま格納される。構造最適化でもアップデートされない。

- **double** * basis_z_orig ([basis_n])

  基底関数の z 座標の値。read_basis 関数で確保されセットされる。入力ファイルから読まれた値がそのまま格納される。構造最適化でもアップデートされない。

- **int** * basis_aux_to_basis ([basis_n])

  RI 近似で使用される補助基底と通常の基底関数の対応を格納している配列。補助基底に関しては、個別に座標情報などは記録されておらず、これらを取得する場合は、対応する基底関数の情報を参照し取得する。

- **char** * basis_label [2] [5] [15] [6]

  「基底関数の種類を示すラベル」が格納されている配列。分子軌道の出力などに使用される。このプログラムで採用した「基底関数の種類を表すラベル」を **表 3** にまとめる。

表 3: basis_label [ ] [i] [j] [k] に格納されている文字列。

i	j	k	basis_label [0] [ ] [ ] [ ]	i	j	k	basis_label [1] [ ] [ ] [ ]
0	0	0	s	1	0	0	s
0	1	0	px	1	1	0	px
0	1	1	py	1	1	1	py
0	1	2	pz	1	1	2	pz
0	2	0	d 200	1	2	0	d+0
0	2	1	d 110	1	2	1	d+1
0	2	2	d 101	1	2	2	d−1
0	2	3	d 020	1	2	3	d+2
0	2	4	d 011	1	2	4	d−2
0	2	5	d 002				
0	3	0	f 300	1	3	0	f+0
0	3	1	f 210	1	3	1	f+1
0	3	2	f 201	1	3	2	f−1
0	3	3	f 120	1	3	3	f+2
0	3	4	f 111	1	3	4	f−2
0	3	5	f 102	1	3	5	f+3
0	3	6	f 030	1	3	6	f−3
0	3	7	f 021				
0	3	8	f 012				
0	3	9	f 003				
0	4	0	g 400	1	4	0	g+0
0	4	1	g 310	1	4	1	g+1
0	4	2	g 301	1	4	2	g−1
0	4	3	g 220	1	4	3	g+2
0	4	4	g 211	1	4	4	g−2
0	4	5	g 202	1	4	5	g+3
0	4	6	g 130	1	4	6	g−3
0	4	7	g 121	1	4	7	g+4
0	4	8	g 112	1	4	8	g−4
0	4	9	g 103				
0	4	10	g 040				
0	4	11	g 031				
0	4	12	g 022				
0	4	13	g 013				
0	4	14	g 004				

11 原子核

- **int** nucleus_n

  原子核の数。

- **double *** nucleus_u ([nucleus_n])

  原子核の電荷。

- **double *** nucleus_x ([nucleus_n])

  原子核の x 座標。構造最適化ではイタレーションことにアップデートされる。単一構造計算の場合は、nucleus_x_orig のアドレスが格納され、メモリ領域は確保されない。

- **double *** nucleus_y ([nucleus_n])

  原子核の y 座標。構造最適化ではイタレーションことにアップデートされる。単一構造計算の場合は、nucleus_y_orig のアドレスが格納され、メモリ領域は確保されない。

- **double *** nucleus_z ([nucleus_n])

  原子核 z 座標。構造最適化ではイタレーションことにアップデートされる。単一構造計算の場合は、nucleus_z_orig のアドレスが格納され、メモリ領域は確保されない。

- **double *** nucleus_x_orig ([nucleus_n])

  原子核の x 座標。入力ファイルから読まれた値がそのまま格納される。構造最適化でもアップデートされない。

- **double *** nucleus_y_orig ([nucleus_n])

  原子核の y 座標。入力ファイルから読まれた値がそのまま格納される。構造最適化でもアップデートされない。

- **double *** nucleus_z_orig ([nucleus_n])

  原子核の z 座標。入力ファイルから読まれた値がそのまま格納される。構造最適化でもアップデートされない。

- **double *** nucleus_vx ([nucleus_n])

  原子核の速度の x 成分。分子動力学計算の際に使用され、時間ステップごとにアップデートされる。

- **double *** nucleus_vy ([nucleus_n])

原子核の速度の y 成分。分子動力学計算の際に使用され、時間ステップごとにアップデートされる。

- **double *** nucleus_vz ([nucleus_n])

  原子核の速度の z 成分。分子動力学計算の際に使用され、時間ステップごとにアップデートされる。

- **double *** nucleus_vx_orig ([nucleus_n])

  原子核の速度の x 成分。分子動力学計算の際に使用され、時間ステップでアップデートされない（初期速度がずっと格納されている）。

- **double *** nucleus_vy_orig ([nucleus_n])

  原子核の速度の y 成分。分子動力学計算の際に使用され、時間ステップでアップデートされない（初期速度がずっと格納されている）。

- **double *** nucleus_vz_orig ([nucleus_n])

  原子核の速度の z 成分。分子動力学計算の際に使用され、時間ステップでアップデートされない（初期速度がずっと格納されている）。

- **int *** nucleus_atomic ([nucleus_n])

  原子核の原子番号。MCP を使用する場合など、電荷の値（nucleus_u）と一致しないこともあり得る。

- **char *** nucleus_opt_fix ([nucleus_n])

  部分構造最適化の時、原子核の位置を動かすか否かを記録している配列。以下のように整数で指定される。

  ```
  0 : 動かす
  1 : 動かさない
  ```

  部分エネルギー勾配を使う場合、エネルギー勾配が計算されない原子核は 0 になっていなければならない。

- **char *** nucleus_abmd_fix ([nucleus_n])

  分子動力学計算で、位置を動かすか否かを各原子核ごとに記録している配列。指定の仕方は nucleus_opt_fix と同じ。

- **double *** nucleus_u_grad_xyz ([nucleus_n * 3])

  核間反発エネルギーの微分。calc_nuclear_repulsion_grad で計算され、セットされる。

12　外部静電ポテンシャル

- **int** ext_point_charge_n

  外部点電荷の数。

- **double *** ext_point_charge_x ([ext_point_charge_n])

  外部点電荷の x 座標。

- **double *** ext_point_charge_y ([ext_point_charge_n])

  外部点電荷の y 座標。

- **double *** ext_point_charge_z ([ext_point_charge_n])

  外部点電荷の z 座標。

- **double *** ext_point_charge_q ([ext_point_charge_n])

  外部点電荷の電荷。

13　位置の指定

- **int** pos_n

  位置の数。

- **double *** pos_x ([pos_n])

  位置の x 座標。

- **double *** pos_y ([pos_n])

  位置の y 座標。

- **double *** pos_z ([pos_n])

  位置の z 座標。

14 フラグメント

- **int** * frag_list_esp_4_n ([frag_n])

  各モノマーに関して、環境静電ポテンシャルを計算する際に、4 中心積分から計算する距離にあるフラグメント数を保持する配列。

- **int** * frag_list_esp_3_n ([frag_n])

  各モノマーに関して、環境静電ポテンシャルを計算する際に、3 中心積分の近似を用いる距離にあるフラグメント数を保持する配列。

- **int** * frag_list_dscf_n ([frag_n])

  各モノマーに関して、ダイマー計算を行う際、SCF 計算を行ってエネルギーを計算する距離にあるフラグメントの数を保持する配列。

- **int** * frag_list_esp_4_idx ([frag_n])

  frag_list_exp_4 を参照するためのインデックス。

- **int** * frag_list_esp_3_idx ([frag_n])

  frag_list_exp_3 を参照するためのインデックス。

- **int** * frag_list_dscf_idx ([frag_n])

  frag_list_dscf を参照するためのインデックス。

- **int** * frag_list_esp_4 ([ ])

  各モノマーに関して、環境静電ポテンシャルを計算する際に、4 中心積分から計算する距離にあるフラグメントの番号を保持する配列。

- **int** * frag_list_esp_3 ([ ])

  各モノマーに関して、環境静電ポテンシャルを計算する際に、3 中心積分の近似を用いる距離にあるフラグメントの番号を保持する配列。

- **int** * frag_list_dscf ([ ])

  各モノマーに関して、ダイマーのエネルギーを計算する際に、SCF 計算を行ってエネルギーを計算する距離にあるフラグメントの番号を保持する配列。

- **int** frag_n

  全フラグメント数。分子動力学計算の場合、自動フラグメント分割が実行されるため、時間

ステップごとにアップデートされる。単一構造計算の場合は、最後までインプットファイルで指定されたフラグメント数のまま。

- **int** frag_n_orig

  全フラグメント数。入力ファイルで指定されたフラグメント数が代入され、分子動力学計算の場合もアップデートされない。

- **int** * frag_nelec ([frag_n])

  各フラグメントに含まれる電子数。read_fragment 関数でセットされる。分子動力学計算の場合、時間ステップごとにフラグメントの定義が変化するため、この値もステップごとにアップデートされる。

- **int** * frag_nelec_orig ([frag_n])

  各フラグメントに含まれる電子数。read_fragment 関数でセットされる。入力ファイルで指定された電子数がセットされ、分子動力学計算の場合もアップデートされない。

- **int** frag_nelec_all

  系全体の電子数。initialize_fragment 関数でセットされる。

- **int** * frag_nfzc ([frag_n])

  各フラグメントのフローズンコアの数。initialize_fragment 関数でセットされる。

- **int** * frag_npch ([frag_n])

  各フラグメントの正電荷（核電荷）数。initialize_fragment 関数でセットされる。

- **int** frag_nnu1_all

  各フラグメントに含まれる原子核の数（frag_nch1）を合計した値。結合の切断に伴い原子核が複数のフラグメントに共有されることがあるので、系全体の原子核の数（charge_n）と同じではない。initialize_fragment 関数でセットされる。

- **int** frag_nnu1_max

  各フラグメントに含まれる原子核の数（frag_nch1）の最大値。initialize_fragment 関数でセットされる。

- **int** * frag_nnu1 ([frag_n])

  各フラグメントに含まれる原子核の数（nch1）。分子動力学計算の場合、時間ステップごとにアップデートされる。

- **int** * frag_nnu1_orig ([frag_n])

  各フラグメントに含まれる原子核の数（nch1）。read_fragment 関数でセットされる。分子動力学計算の場合もアップデートされない。

- **int** * frag_nnu1_in ([frag_n])

  各フラグメントに含まれる原子核のうち「フラグメント内部の原子核」の数。read_fragment 関数でセットされる。分子動力学計算の場合、時間ステップごとにアップデートされる。結合の切断が有る場合は自分以外のフラグメントに属する原子核の電荷の一部を計算に追加することになるが、追加される原子核を除いたものを「フラグメント内部の原子核」と定義する。つまり、

  ```
  frag_nch1 = frag_nnu1_in + 「追加される核電荷」
  ```

  である。***PAICS*** では、「追加される核電荷」と「追加される基底関数」を指定することで結合の切断が定義される。

- **int** * frag_nnu1_in_orig ([frag_n])

  各フラグメントに含まれる原子核のうち「フラグメント内部の原子核」の数。read_fragment 関数でセットされる。分子動力学計算の場合もアップデートされない。

- **int** * frag_nnu1_idx ([frag_n])

  各フラグメントに含まれる原子核を 1 次元の配列に並べたときのインデックス。分子動力学計算の場合、時間ステップごとにアップデートされる。read_fragment 関数でセットされる。

- **int** * frag_nnu1_idx_orig ([frag_n])

  各フラグメントに含まれる原子核を 1 次元の配列に並べたときのインデックス。分子動力学計算の場合もアップデートされない。read_fragment 関数でセットされる。

- **int** * frag_nucleus ([frag_nch1_all])

  各フラグメントに属する原子核が全体のうち何番目の原子核かを保持する配列。分子動力学計算の場合、時間ステップごとにアップデートされる。「nucleus_」ではじまるグローバル変数が原子核全体の情報を管理しており、各フラグメントは、自身に含まれる原子核の全体の通し番号のみを保持している。ifrag 番目のフラグメントの ich 番目の原子核が何番目の原子核かは、以下のように参照される。

  ```
  frag_charge [ frag_nch1_idx [ ifrag ] + ich ]
  ```

- **int** * frag_nucleus_orig ([frag_nch1_all])

  各フラグメントに属する原子核が全体のうち何番目の原子核かを保持する配列。分子動力学計算の場合もアップデートされない。

- **double** * frag_nucleus_u ([frag_nch1_all])

  各フラグメントに含まれる原子核の「修正された核電荷の値」を格納している配列。read_fragment 関数でセットされる。結合の切断が有る場合、自身のフラグメントに属する原子核の核電荷の一部を他のフラグメントに割り当てたり、他のフラグメントに属する原子核の核電荷の一部を自身のフラグメントに含めたりするので、原子核の核電荷がそのまま各フラグメントの計算を行う時の核電荷になる訳ではない。このような核電荷の値を「修正された核電荷の値」と定義する。どのように修正されるかは、フラグメント分割の定義に依存する。ifrag 番目の ich 番目の「修正された核電荷の値」は以下のように参照される。

  ```
  frag_charge_u [ frag_nch1_idx [ ifrag ] + ich ]
  ```

- **int** frag_nba1_all

  各フラグメントに含まれる基底関数（frag_nba1）の合計。結合の切断がある場合、基底関数が複数のフラグメントに重複して使用されるので、系全体の基底関数の数（basis_n）と同じにはならない。read_fragment 関数でセットされる。

- **int** frag_nba1_max

  フラグメントに属する基底関数の最大値。

- **int** * frag_nba1 ([frag_n])

  各フラグメントに含まれる基底関数。分子動力学計算の場合、時間ステップごとにアップデートされる。read_fragment 関数でセットされる。

- **int** * frag_nba1_orig ([frag_n])

  各フラグメントに含まれる基底関数。分子動力学計算の場合もアップデートされない。read_fragment 関数でセットされる。

- **int** * frag_nba1_in ([frag_n])

  各フラグメントに含まれる基底関数のうち「フラグメント内部の基底関数」の数。read_fragment 関数でセットされる。結合の切断が有る場合、自分以外のフラグメントに属する基底関数も計算に追加されることになるが、この追加された基底関数を除いたものを「フラグメント内部の基底関数」と定義する。つまり、

  ```
  frag_nba1 = frag_nba1_in + 「追加された基底関数」
  ```

である。

- **int** * frag_nba1_in_orig ([frag_n])

  各フラグメントに含まれる基底関数のうち「 フラグメント内部の基底関数 」の数。分子動力学計算の場合もアップデートされない。read_fragment 関数でセットされる。

- **int** * frag_nba1_idx ([frag_n])

  各フラグメントに含まれる基底関数を 1 次元の配列に並べたときのインデックス。分子動力学計算の場合、時間ステップごとにアップデートされる。read_fragment 関数でセットされる。

- **int** * frag_nba1_idx_orig ([frag_n])

  各フラグメントに含まれる原子核を 1 次元の配列に並べたときのインデックス。分子動力学計算の場合もアップデートされない。read_fragment 関数でセットされる。

- **int** * frag_basis ([frag_nba1_all])

  各フラグメントに含まれる基底関数の通し番号が格納されている配列。read_fragment 関数でセットされる。分子動力学計算の場合、時間ステップごとにアップデートされる。ifrag 番目の iba 番目の基底関数の番号は以下のように参照される。

  ```
  frag_basis [ frag_nba1_idx [ ifrag ] + iba ]
  ```

- **int** * frag_basis_orig ([frag_nba1_all])

  各フラグメントに含まれる基底関数の通し番号が格納されている配列。分子動力学計算の場合もアップデートされない。read_fragment 関数でセットされる。

- **int** * frag_basis_add_frag

  各フラグメントに属する基底関数のうち、「追加された基底関数」（つまり「フラグメント内部の基底関数」以外）が、どのフラグメントに属する基底関数かを保持する配列。ifrag 番目のモノマーに「追加された基底関数」のうちの iba 番目の基底関数が、どのフラグメントに属しているかは、以下のように参照される。

  ```
  mon_basis_add_frag [ mon_basis_add_frag_idx [ ifrag ] + iba ]
  ```

- **int** * frag_basis_add_frag_idx ([frag_n])

  mon_basis_add_frag を参照するためのインデックス。

- **int** frag_nsh1_all

  各フラグメントに属するシェルの数の合計。

- **int** frag_nsh1_max

  各フラグメントに属するシェルの最大値。

- **int** * frag_nsh1 ([frag_n])

  各フラグメントに属するシェルの数を保持する配列。

- **int** * frag_nsh1_idx ([frag_n])

  各フラグメントに属するシェルの情報を 1 次元の配列に並べた際のインデックス。

- **int** * frag_nsh1_xyz_idx ([frag_n])

  各フラグメントに属するシェルの座標（x,y,z）を 1 次元の配列に並べた際のインデックス。

- **int** * frag_shell ([frag_nsh1_all * 3])

  各フラグメントーに属するシェルの情報を保持する配列。***PAICS*** では、各シェルに関して 3 つの情報を保持しており、それらは、

  ```
  0 → シェルが属する「基底関数の定義」の番号
  1 → シェルが属する「基底関数の定義」のうち何番目のシェルか
  2 → シェルの先頭の ao がモノマー内の ao の通し番号で何番目か
  ```

  である。ifrag 番目のモノマーの ish 番目のシェルの各情報は、

  ```
  0 → frag_shell [ frag_nsh1_idx [ ifrag ] + 3 * ish + 0 ]
  1 → frag_shell [ frag_nsh1_idx [ ifrag ] + 3 * ish + 1 ]
  2 → frag_shell [ frag_nsh1_idx [ ifrag ] + 3 * ish + 2 ]
  ```

  のように参照される。

- **int** * frag_shell_xyz ([frag_nsh1_all * 3])

  各フラグメントに属するシェルの座標を保持する配列。ifrag 番目のフラグメントの ish 番目のシェルの座標は、

  ```
  frag_shell [ frag_nsh1_xyz_idx [ ifrag ] + 3 * ish + 0 ]   ( x 座標 )
  frag_shell [ frag_nsh1_xyz_idx [ ifrag ] + 3 * ish + 1 ]   ( y 座標 )
  frag_shell [ frag_nsh1_xyz_idx [ ifrag ] + 3 * ish + 2 ]   ( z 座標 )
  ```

  のように参照される。

- **int** frag_nbo1_all

  各フラグメントの nbo1 の合計。

- **int** frag_nbo1_max

  フラグメントの nbo1 の最大値。

- **int** * frag_nbo1 ([frag_n])

  各フラグメントの nbo を保持する配列。

- **int** * frag_nbo1_idx ([frag_n])

  各フラグメントの bo に関する値を 1 次元の配列に並べた際のインデックス。

- **int** frag_nbot_all

  各フラグメントの nbot の合計。

- **int** frag_nbot_max

  フラグメントーの nbot の最大値。

- **int** * frag_nbot ([frag_n])

  各フラグメントの nbo に関する下三角行列の要素数を保持する配列。

- **int** * frag_nbot_idx ([frag_n])

  各フラグメントの bo に関する上三角行列を 1 次元の配列に並べた際のインデックス。

- **int** * frag_nbot_idx ([frag_n])

  各フラグメントの bo に関する上三角行列を 1 次元の配列に並べた際のインデックス。全体の通し番号で ifrag 番目のフラグメントの先頭番地は

  ```
  frag_nbot_idx [ ifrag ]
  ```

  と参照される。

- **int** frag_nbo2_all

  フラグメントの nbo2 の合計。

- **int** frag_nbo2_max

  フラグメントの nbo2 の最大値。

- **int** * frag_nbo2 ([frag_n])

  各フラグメントの bo に関する行列の要素数を保持する配列。つまり、以下の値を保持している。

```
frag_nbo2[ifrag] = frag_nbo1[ifrag] * frag_nbo1[ifrag]
```

- **int** * frag_nbo2_idx ([frag_n])

  各フラグメントの bo に関する行列を 1 次元の配列に並べた際のインデックス。

- **int** frag_pro_n_all

  各フラグメントに含まれる projection orbital の数（frag_pro_n）の合計。make_projection_operator 関数でセットされる。

- **int** * frag_pro_n ([frag_n])

  各フラグメントの projection orbital の数。make_projection_operator 関数でセットされる。

- **int** * frag_pro_idx ([frag_n])

  各フラグメントの projection orbital の番号を並べた際のインデックス。make_projection_operator 関数でセットされる。

- **int** * frag_pro ([frag_pro_n_all])

  各フラグメントの projection orbital の番号を記録している配列。make_projection_operator 関数でセットされる。ifrag 番目のフラグメントの ipro 番目の射影演算子の番号は以下のように参照される。

  ```
  frag_pro [ frag_pro_idx [ ifrag ] + ipro ]
  ```

- **int** frag_link_n_all

  各フラグメントに属する結合の切断の数（frag_link_n）の合計。make_projection_operator 関数でセットされる。

- **int** * frag_link_n ([frag_n])

  各フラグメントに属する結合の切断（自身が結合を奪うタイプの切断）の数。make_projection_operator 関数でセットされる。

- **int** * frag_link_idx ([frag_n])

  各フラグメントに属する結合の切断（自身が結合を奪うタイプの切断）に関するインデックス。make_projection_operator 関数でセットされる。

- **int** * frag_link_1 ([frag_pro_link_n_all])

  結合の切断（自身が結合を奪うタイプの切断）における相手のフラグメント側の原子の通し番号。make_projection_operator 関数でセットされる。

- **int** * frag_link_2 ([frag_pro_link_n_all])

  結合の切断（自身が結合を奪うタイプの切断）における自身のフラグメント側の原子の通し番号。make_projection_operator 関数でセットされる。

- **int** * frag_calc_order ([frag_n])

  計算を行う順番を格納している配列。make_frag_calc_order 関数でセットされる。

- **int** * frag_calc_fprop ([frag_n])

  電子密度、静電ポテンシャル、電場といった任意の位置に対して定義される値を計算する時に、各フラグメントからの寄与を考慮に入れるか否か。以下のように整数で指定する。

  ```
  0 : 考慮に入れない。
  1 : 考慮に入れる。
  ```

  通常は全てのフラグメントからの寄与を考慮する（デフォルト）。

- **int** * frag_ovl_diag ([frag_nbo1_all])

  各フラグメントの重なり行列の対角要素を記録した配列。

- **double** * frag_muclear_repulsion ([frag_n])

  各フラグメントの核反発エネルギー（「フラグメント内部の核電荷」による）を格納している配列。

- **double** * frag_muclear_ext ([frag_n])

  各フラグメントの原子核が感じる、外部静電ポテンシャルとの相互作用エネルギーを格納している配列。

- **int** frag_pair_calc_n

  ダイマー計算を実行するフラグメントペアの数。

- **int** frag_pair_calc_n_orig

  ダイマー計算を実行するフラグメントペアの数。

- **int** frag_pair_calc_n_es

  ダイマー計算を実行するフラグメントペアのうちダイマー ES 近似が適用されるペアの数。make_frag_pair_calc_order_n 関数でセットされる。ダイマー ES 近似が適用されるペアに関しては、その後の MP2 計算なども実行されない。

- **int** frag_pair_calc_n_scf

  ダイマー計算を実行するフラグメントペアのうちダイマー SCF で計算するペアの数。make_frag_pair_calc_order_n 関数でセットされる。ダイマー SCF 計算を行うペアは、その後の MP2 計算なども実行される。

- **int** frag_pair_calc_1_idx

  frag_pair_calc_1 および frag_pair_calc_2 の値を参照するときに使われるインデックス。

- **int** frag_pair_calc_1_idx_orig

  frag_pair_calc_1_orig および frag_pair_calc_2_orig の値を参照するときに使われるインデックス。

- **int** * frag_pair_calc_1 ([frag_pair_calc_n])

  ダイマー計算が実行されるフラグメントペアのうち一方のフラグメントの番号が記録されている配列。read_fragment 関数で値が代入される。

- **int** * frag_pair_calc_2 ([frag_pair_calc_n])

  ダイマー計算が実行されるフラグメントペアのうち一方のフラグメントの番号が記録されている配列。read_fragment 関数で値が代入される。

- **int** * frag_pair_calc_1_orig ([frag_pair_calc_n])

  ダイマー計算が実行されるフラグメントペアのうち一方のフラグメントの番号が記録されている配列。

- **int** * frag_pair_calc_2_orig ([frag_pair_calc_n])

  ダイマー計算が実行されるフラグメントペアのうち一方のフラグメントの番号が記録されている配列。

- **int** * frag_pair_calc_order ([frag_pair_calc_n_scf * 2])

  ダイマー SCF 計算を実行するペアの順番を保持している配列。make_frag_pair_calc_order 関数でセットされる。ダイマー計算を実行するペアかどうかをチェックし、かつ、ダイマー SCF 計算を行うペアかどうかをチェックし、どちらも満たすものがこの順番の対象になる。

- **int** frag_calc_total_chk

  分子全体のプロパティを計算する場合は 1 が入りそうでない場合は 0 が入る。つまり、全てのフラグメントペアのダイマー計算が実行される場合（frag_calc_pair の値が全て 1 の場合）のみ、この変数の値が 1 となり、トータルなプロパティが計算される。計算するダイマーペアを制限した場合はトータルプロパティは計算されない。fragment_initalize 関数でセットされる。

15 射影演算子

- **int** prj_lpring

  射影演算子を自動生成する部分の出力フラグ。

- **int** prj_loc_tv

  射影演算子を自動生成する際の局在化の閾値。

- **int** prj_loc_tv

  射影演算子を自動生成する際の RHF の DIIS の閾値。

- **int** prj_loc_maxit

  射影演算子を自動生成する際の局在化のイタレーション回数の最大値。

- **int** prj_total_n

  射影演算子の総数。ここでは、1 つの局在化軌道が 1 つの射影演算子を構成すると考える。従って、「射影演算子を構成する局在化軌道の総数」と同じ意味。例えば、通常の C–C 単結合の切断の場合、5 つの射影演算子（つまり、局在化軌道）が関与していることになる。

- **int** prj_total_nba

  射影演算子を構成する局在化軌道の基底関数の数を、全射影演算子で合計した値。通常、1 つの局在化軌道は 1 つの基底関数から定義されるので、frag_pro_total_n と同じ値になる（今後の拡張を考えての導入）。

- **int** prj_total_nbo

  射影演算子における局在化軌道の nbo を、全射影演算子で合計した値。

- **int** * prj_nba ([prj_total_n])

  各射影演算子を構成する局在化軌道の基底関数の数。通常は、1 つの局在化軌道は 1 つの基底関数から定義されるので、全て 1 が入る（今後の拡張を考えての導入）。

- **int** * prj_basis_idx ([prj_total_n])

  各局在化軌道の基底関数の番号を並べた際のインデックス。

- **int** * prj_basis ([prj_total_nba])

  各局在化軌道の基底関数の番号を記録した配列。ipro 番目の局在化軌道の iba 番目の基底関数の番号は以下のように参照される。

```
prj_basis [ prj_basis_idx [ ipro ] + iba ]
```

従って、ifrag 番目のフラグメントの ipro 番目の局在化軌道の iba 番目の基底関数の番号は以下のように参照される。

```
prj_basis[prj_basis_idx[frag_pro[frag_pro_idx[ifrag]+ipro]]+iba]
```

- **int** * prj_nbo ([prj_total_n])

  各局在化軌道の bo の数。

- **int** * prj_c_idx ([prj_total_n])

  各局在化軌道の軌道係数を参照するためのインデックス。

- **int** * prj_c ([prj_total_nbo])

  各局在化軌道の軌道係数を格納している配列。ipro 番目の局在化軌道の ibo 番目の bo に対する係数は以下のように参照される。

  ```
  prj_c [ prj_c_idx [ ipro ] + ibo ]
  ```

- **int** * prj_ifrag ([prj_total_n])

  各局在化軌道に関するフラグメントのうち、一方のフラグメント番号を記録している配列。その局在化軌道が構成する射影演算子が属するフラグメントの番号が入る。

- **int** * prj_jfrag ([prj_total_n])

  各局在化軌道に関するフラグメントのうち、一方のフラグメント番号を記録している配列。その局在化軌道が構成する射影演算子が属さないフラグメントの番号が入る。

- **int** * prj_h_basis_name [21]

  局在化軌道の自動生成の際、対象の炭素原子に水素原子を 4 つ付加した仮想的メタン分子の計算を行い局在化軌道を作成するが、その際、水素原子上におかれる基底関数の名前を格納している文字列。通常は STO–3G が使用される（実際の FMO 計算に STO-3G が使用されるわけではない。仮想的なメタン分子の計算で得られる分子軌道は、ほぼ分子の対称性から決まるので、ここで使用する基底関数の種類は、得られる局在化軌道にほとんど影響を与えない)。

- **int** * pro_h_basis_type

  局在化軌道の自動生成の際に水素原子上に置かれる「基底関数の定義」の番号を格納している配列。

16 FMO 計算パラメータ

- **int** fmo_scc_maxit

  モノマー SCC 計算のイタレーションの最大回数。デフォルト値は 99（read_input_list_01 関数の中で設定)。入力リストの「scc_maxit」キーワードで変更可能。

- **int** fmo_scc_no_dyn

  モノマー SCC 計算で、ダイナミックアップデートを行うか否かの指定。

  ```
  0 : 行う、1 : 行わない。
  ```

  デフォルトは 0。

- **int** fmo_cp_corr

  IFIE に対する CP 補正を行うか否か。デフォルト値は 0（read_input_list_01 関数の中で設定)。入力リストの「textsfcp_corr」キーワードで指定可能。

  ```
  0 : 行わない、1 : 行う。
  ```

- **double** fmo_scc_tv

  モノマー SCC の収束の判定に使用される閾値の 1 つ。各モノマーエネルギーの変化がすべてこの閾値よりも小さくなったら収束と判定する。デフォルト値は 1.0E−6（read_input_list_01 関数の中で設定)。入力リストの「scc_tv」キーワードで変更可能。

- **double** fmo_scc_tv_total

  モノマー SCC の収束の判定に使用される閾値の 1 つ。FMO1 エネルギーの変化がこの閾値よりも小さくなったら収束と判定する。デフォルト値は 1.0E−6（read_input_list_01 関数の中で設定)。入力リストの「scc_tv_total」キーワードで変更可能。

- **double** fmo_ldimer

  ダイマー計算を実行する際、フラグメント間距離が

  ```
  [ ファンデルワールス半径 ] * [ fmo_ldimer ]
  ```

  よりも大きければダイマー ES 近似を適用する。デフォルト値は 2.0（read_input_list_01 関数の中で設定)。入力リストの「ldimer」キーワードで変更可能。

- **double** fmo_laoc

  環境静電ポテンシャルを計算する際、相手のフラグメントとの距離が

  ```
  [ ファンデルワールス半径 ] * [ fmo_laoc ]
  ```

  よりも小さければ4中心積分を用いて（近似無しに）計算する。デフォルト値は0.0（read_input_list_01関数の中で設定）。従って、デフォルトのままでは4中心積分を用いた環境静電ポテンシャルの計算は行われない。これよりも大きい距離の場合、3中心積分の近似か点電荷近似で計算される。入力リストの「laoc」キーワードで変更可能。

- **double** fmo_lptc

  環境静電ポテンシャルを計算する際、相手のフラグメントとの距離が

  ```
  [ ファンデルワールス半径 ] * [ fmo_lptc ]
  ```

  よりも大きければ点電荷で近似する。デフォルト値は2.0（read_input_list_01関数の中で設定）。入力リストの「lptc」キーワードで変更可能。

- **double** fmo_projecion_tv

  結合の切断で使用される射影演算子のファクター。正の実数で指定する。デフォルト値は1.0E+6（read_input_list_01関数の中で設定）。入力リストの「projection_tv」キーワードで変更可能。この処理のため、FMO計算におけるフォック行列は、大きな値と小さな値が混合した行列となり、数値的な問題を発生する可能性を含んでいる。従って、大きければ大きいほど良いという訳でもない。

17 モノマー SCC

- **int** * scc_chk_read_frag ([frag_n])

  フラグメントの電子密度をファイルから読み込んだか否かを記録している配列。以下のように、フラグメントごとにに整数で指定される。

  ```
  0 : ファイルから読み込んでない
  1 : ファイルから読み込んだ
  ```

  ファイルから電子密度を読んだ場合 1 が代入され、その密度が、モノマー SCC 計算の初期電子密度として使用される。通常は、フラグメント単体の計算を真空中で実行して得られる電子密度を使用する。

- **int** * scc_d_mat ([frag_nbot_all])

  モノマー SCC 計算の際に、密度行列を格納する配列。そのフラグメントの計算が終わった後、ただちに上書きされる（ダイナミックアップデートの場合に使用される）。上三角行列として保持している。ifrag 番目のフラグメント密度行列の i、j 要素（i≥j）の値は以下のように参照される。

  ```
  scc_d_mat [ frag_nbot_idx[irag] + (i+1)*i/2 + j ]
  ```

- **int** * scc_mulliken_ba ([frag_nba1_all])

  モノマー SCC 計算の際、基底関数ごとの Mulliken ポピュレーションを記録している配列。そのフラグメントの計算が終わった後、ただちに上書きされる（ダイナミックアップデートの場合に使用される）。

- **int** * scc_mulliken_bo ([frag_nbo1_all])

  モノマー SCC 計算の際、bo ごとの Mulliken ポピュレーションを記録している配列。そのフラグメントの計算が終わった後、ただちに上書きされる（ダイナミックアップデートの場合に使用される）。

- **int** * scc_e_charge_center ([frag_n] [3])

  各フラグメントの負電荷の中心が格納されている配列。

- **int** * scc_u_charge_center ([frag_n] [3])

  各フラグメントーの正電荷の中心が格納されている配列。

- **double** scc_time_scf

  モノマー SCC 計算のうち、SCF 計算にかかった時間（秒）が記録される変数。

- **double** scc_time_eps

  モノマー SCC 計算のうち、環境静電ポテンシャルの計算にかかった時間（秒）が記録される変数。

- **double** scc_time_eps_4

  モノマー SCC 計算のうち、4 中心積分による環境静電ポテンシャルの計算にかかった時間（秒）が記録される変数。

- **double** scc_time_eps_3

  モノマー SCC 計算のうち、3 中心積分の近似による環境静電ポテンシャルの計算にかかった時間（秒）が記録される変数。

- **double** scc_time_eps_m

  モノマー SCC 計算のうち、点電荷近似による環境静電ポテンシャルの計算にかかった時間（秒）が記録される変数。

- **double** scc_time_eps_u

  モノマー SCC 計算のうち、原子核による環境静電ポテンシャルの計算にかかった時間（秒）が記録される変数。

- **double** scc_time_eps_ext

  モノマー SCC 計算のうち、外部静電場によるポテンシャルの計算にかかった時間（秒）が記録される変数。

18 ダイマー ES

- **double** esa_time_e

  ダイマー ES 近似のうち電子間反発エネルギーの計算にかかった時間が記録される変数。

- **double** esa_time_u

  ダイマー ES 近似のうち電子 – 核間引力エネルギーの計算にかかった時間が記録される変数。

- **double** esa_time_grad_e

  ダイマー ES 近似のうち電子間反発エネルギーの微分計算にかかった時間が記録される変数。

- **double** esa_time_frag_u

  ダイマー ES 近似のうち電子 – 核間引力エネルギーの微分計算にかかった時間が記録される変数。

19 RHF 計算

- **int** rhf_chk

  RHF 計算を行うか否か。以下のように整数で指定される。

  ```
  0 : 計算しない
  1 : 計算する。
  ```

  通常の FMO 計算では常に RHF 計算を行うので 1 がセットされる。

- **int** rhf_chk_grad

  RHF 計算の一次微分を計算するか否か。デフォルトは 0。

  ```
  0 : 計算しない
  1 : 計算する。
  ```

- **int** rhf_chk_no_int_buff

  分子積分の保存用にバッファーを用意するか否か。

  ```
  0 : 保存する
  1 : 保存しない。
  ```

  デフォルトは 0 で 1 はデバッグ用。0 の場合、RHF 計算で、与えられたメモリが許す限り 4 中心積分を保存し、イタレーションで再利用する。保存できない 4 中心積分はイタレーション毎に計算される。

- **int** rhf_chk_mon_worb

  モノマー RHF の計算で得られた分子軌道の情報をファイルに書き出すか否か。

  ```
  0 : 書き出さない
  1 : 書き出す
  ```

  分子軌道を可視化ソフトなどで描画したい場合などに利用する。cntrl_w_result_file からファイル名が自動的に決められる。

- **int** rhf_lprint_1

  モノマーの RHF 計算におけるプリントフラグ。-1 にセットした場合は推奨の出力レベルとなる。デフォルトは -1。

3 : 通常の出力（ -1 の場合と同じ ）
4 : 初期軌道作成の詳細、分子軌道、Mulliken 電荷の情報が出力される。

- **int** rhf_lprint_2

  ダイマーの RHF 計算におけるプリントフラグ。-1 にセットした場合は推奨の出力レベルとなる。デフォルトは -1。

  1 : 通常の出力。
  2 : BSSE 補正の詳細などが出力される。

- **double** rhf_eng_tv

  RHF イタレーションの収束を判定する閾値。前回とのエネルギー差がこの値よりも小さくなったら収束と判定する。

- **double** rhf_orth_type

  RHF 計算で原子基底関数を直交化する際に使用する手法のフラグ。

  0 : 正準直交化
  1 : 対称直交化

- **double** rhf_orth_tv

  RHF 計算で原子基底関数を直交化する際の閾値。正準直交化の際に有効で、重なり積分行列の対角要素がこれ以下の基底関数は、変分空間から除外される。

- **int** rhf_init_mo_type

  RHF 計算の初期軌道を作成する手法のフラグ。

  0 : H-CORE
  1 : 小さな基底からの射影

- **int** rhf_maxit

  RHF 計算のイタレーション回数の最大値。

- **int** rhf_maxit_mkinit

  RHF 計算の初期軌道を作成する際のイタレーション回数の最大値（rhf_init_mo_type）が 1 の場合に使用される。

- **int** rhf_ndiis

  RHF 計算で DIIS を行う際、エラーベクトルの計算のために確保しておく結果の数。

- **int** rhf_diis_tv

  RHF 計算で、エラーベクトルの最大値がこの値より小さくなったら DIIS に切り替える。

- **int** rhf_diis_tv_ini

  初期軌道を作成する時の RHF 計算で、エラーベクトル最大値がこの値より小さくなったら、DIIS に切り替える。

- **double** rhf_diff_fock_tv

  RHF 計算で differential fock 法を用いる際の閾値。現在は未実装。

- **char** rhf_mon_conv_chk_all

  全フラグメントのモノマー RHF が収束したか否かを記録する配列。monomer_calculation 関数で値がセットされる。収束していないモノマー RHF が 1 つでも存在すれば、トータルプロパティは計算することが出来ない。

  ```
  0 : 収束しないモノマー RHF が存在する
  1 : 全てのモノマー RHF が収束。
  ```

- **char** * rhf_mon_conv_chk ([frag_n])

  各モノマー RHF が収束したか否かを記録する配列。

  ```
  0 : 収束せず
  1 : 収束
  ```

- **int** * rhf_mon_nmo ([frag_n])

  各モノマーの nmo を格納する配列。正準直行化を使った場合、nbo > nmo となることがある。

- **doube** * rhf_mon_moe ([frag_nbo1_all])

  各モノマーの軌道エネルギーを格納する配列。ifrag 番目のフラグメントの imo 番目の軌道エネルギーは以下のように参照される。

  ```
  rhf_mon_moe[frag_nbo1_idx[ifrag]+imo]
  ```

- **doube** * rhf_mon_c_mat ([frag_nbo2_all])

  各モノマーの分子軌道を格納する配列。ifrag 番目のフラグメントの imo 番目の軌道の ibo に対する係数は以下のように参照される。

```
rhf_mon_c_mat[frag_nbo2_idx[ifrag]+frag_nbo1[ifrag]*imo+ibo]
```

- **double *** rhf_mon_energy_d ([frag_n])

  各フラグメントのモノマー RHF エネルギーから環境静電ポテンシャルの寄与を取り除いたもの（E'）を格納している配列。rhf_initialize 関数で allocte され、rhf_monomer 関数で値がセットされる。

- **double *** rhf_mon_trdv ([frag_n])

  各フラグメントが他のフラグメントから感じるポテンシャル。rhf_initialize 関数で allocte され、rhf_monomer 関数で値がセットされる。

- **double *** rhf_mon_trdv_e ([frag_n])

  各フラグメントが外部静電ポテンシャルから感じるポテンシャル。rhf_initialize 関数で allocte され、rhf_monomer 関数で値がセットされる。

- **double *** rhf_mon_m_pop_ba_frag ([frag_nba1_all])

  各フラグメントのモノマーの RHF 計算による基底関数ごとのマリケンポピュレーション。

- **double *** rhf_mon_moe_vac_cp ([frag_nbo1_all])

  モノマーの真空中での RHF 計算で得られた軌道エネルギー。BSSE の CP 補正の算出に使用される。

- **double *** rhf_mon_c_mat_vac_cp ([frag_nbo2_all])

  モノマーの真空中での RHF 計算で得られた軌道係数。BSSE の CP 補正の算出に使用される。

- **double *** rhf_mon_energy_vac_cp ([frag_n])

  モノマーの真空中での RHF 計算で得られたエネルギー。BSSE の CP 補正の算出に使用される。

- **double *** rhf_mon_m_pop_ba ([basis_n])

  FMO1–RHF の基底関数ごとの Mulliken population。

- **double *** rhf_mon_pos_dns ([pos_n])

  FMO1–RHF の電子密度（指定された位置に関して計算される）。

- **double *** rhf_mon_pos_esp ([pos_n])

  FMO1–RHF の電子密度が作る静電ポテンシャル（指定された位置に関して計算される）。

- **double *** rhf_mon_pos_efi_x ([pos_n])

  FMO1–RHF の電子密度が作る電場（指定された位置に関して計算される）。

- **double *** rhf_mon_pos_efi_y ([pos_n])

  FMO1–RHF の電子密度が作る電場（指定された位置に関して計算される）。

- **double *** rhf_mon_pos_efi_z ([pos_n])

  FMO1–RHF の電子密度が作る電場（指定された位置に関して計算される）。

- **char** rhf_dim_conv_chk_all

  全ダイマー RHF が収束したか否かを記録する配列。dimer_calculation 関数で値がセットされる。収束していないダイマー RHF が 1 つでも存在すれば、トータルプロパティは計算されない。計算されるべきダイマーが収束しなかった場合のみ 0 になる。従って、ペアの計算を制限したときに計算されたなかったダイマーは、判定の対象外となる。

  ```
  0 : 収束しないダイマー RHF が存在
  1 : 全ダイマー RHF が収束
  ```

- **char *** rhf_dim_conv_chk ([(frag_n -1) * frag_n / 2])

  ダイマー RHF が収束したか否かを記録する配列。calculation_dimer_es 関数および calculation_dimer 関数でセットされる。

  ```
  0 : 収束せず
  1 : 収束
  ```

  ifrag 番目のフラグメントと jfrag 番目のフラグメントが収束したか否かは、以下のように参照される（ifrag ＞ jfrag）。

  ```
  rhf_dim_conv_chk [ ( frag_0[ifrag] - 1 ) * frag_0[ifrag] / 2 + frag_0[jfrag] ]
  ```

- **double *** rhf_dim_ifie ([(frag_n - 1) * frag_n / 2])

  IFIE を保持する変数。dimer_initialize 関数で確保され、dimer_rhf 関数で値がセットされる。ifrag 番目のフラグメントと jfrag 番目のフラグメントの IFIE は、以下のように参照される（ifrag ＞ jfrag）。

  ```
  rhf_dim_ifie [ ( frag[ifrag] - 1 ) * frag[ifrag] / 2 + frag[jfrag] ]
  ```

- **double** * rhf_dim_trdv ([(frag_n - 1) * frag_n / 2])

  ダイマー RHF 計算での、他のフラグメントからの環境静電ポテンシャル。dimer_initialize 関数で確保され、dimer_rhf 関数で値がセットされる。ifrag 番目のフラグメントと jfrag 番目のフラグメントの値は、以下のように参照される（ifrag ＞ jfrag)。

  ```
  rhf_dim_trdv [ ( frag[ifrag] - 1 ) * frag[ifrag] / 2 + frag[jfrag] ]
  ```

- **double** * rhf_dim_trdv_i ([(frag_n - 1) * frag_n / 2])

  ダイマー RHF 計算での、他のフラグメントからの環境静電ポテンシャル。dimer_initialize 関数で確保され、dimer_rhf 関数で値がセットされる。ifrag 番目のフラグメントと jfrag 番目のフラグメントの値は、以下のように参照される（ifrag ＞ jfrag)。

  ```
  rhf_dim_trdv_i [ ( frag[ifrag] - 1 ) * frag[ifrag] / 2 + frag[jfrag] ]
  ```

- **double** * rhf_dim_trdv_j ([(frag_0_n - 1) * frag_0_n / 2])

  ダイマー RHF 計算での、他のフラグメントからの環境静電ポテンシャル。dimer_initialize 関数で確保され、dimer_rhf 関数で値がセットされる。ifrag 番目のフラグメントと jfrag 番目のフラグメントの値は、以下のように参照される（ifrag ＞ jfrag)。

  ```
  rhf_dim_trdv_j [ ( frag[ifrag] - 1 ) * frag[ifrag] / 2 + frag[jfrag] ]
  ```

- **double** * rhf_dim_trdv_e ([(frag_n - 1) * frag_n / 2])

  ダイマー RHF 計算での、外部静電ポテンシャルから感じるポテンシャル。dimer_initialize 関数で確保され、dimer_rhf 関数で値がセットされる。

- **double** * rhf_dim_trdv_e_i ([(frag_n - 1) * frag_n / 2])

  ダイマー RHF 計算での、外部静電ポテンシャルから感じるポテンシャル。dimer_initialize 関数で確保され、dimer_rhf 関数で値がセットされる。

- **double** * rhf_dim_trdv_e_j ([(frag_n - 1) * frag_n / 2])

  ダイマー RHF 計算での、外部静電ポテンシャルから感じるポテンシャル。dimer_initialize 関数で確保され、dimer_rhf 関数で値がセットされる。

- **double** * rhf_dim_ifie_cp ([(frag_0_n - 1) * frag_0_n / 2])

  IFIE における BSSE の値。

- **double** * rhf_dim_m_pop_ba ([basis_n])

  FMO2-RHF のマリケンポピュレーション。

- **double *** rhf_dim_pos_dns ([pos_n])

  FMO2–RHF による電子密度の補正分（指定された位置に関して計算される)。

- **double *** rhf_dim_pos_esp ([pos_n])

  FMO2–RHF による補正分の電子密度が作る静電ポテンシャル（指定された位置に関して計算される)。

- **double *** rhf_dim_pos_efi_x ([pos_n])

  FMO2–RHF による補正分の電子密度が作る電場（指定された位置に関して計算される)。

- **double *** rhf_dim_pos_efi_y ([pos_n])

  FMO2–RHF による補正分の電子密度が作る電場（指定された位置に関して計算される)。

- **double *** rhf_dim_pos_efi_z ([pos_n])

  FMO2–RHF による補正分の電子密度が作る電場（指定された位置に関して計算される)。

- **double** rhf_fmo_1

  FMO1–RHF エネルギーを保持する変数。

  ```
  rhf_fmo_1 = rhf_fmo_1_int + rhf_fmo_1_ext
  ```

- **double** rhf_fmo_1_int

  FMO1–RHF エネルギーのうち、内部エネルギーを保持する変数。

- **double** rhf_fmo_1_ext

  FMO1–RHF エネルギーのうち、外部静電ポテンシャルとの相互作用エネルギーを保持する変数。

- **double** rhf_fmo_2

  FMO2–RHF エネルギーを保持する変数。

  ```
  rhf_fmo_2 = rhf_fmo_2_int + rhf_fmo_2_ext
  ```

- **double** rhf_fmo_2_int

  FMO2–RHF エネルギーのうち、内部エネルギーを保持する変数。

- **double** rhf_fmo_2_ext

  FMO2–RHF エネルギーのうち、外部静電ポテンシャルとの相互作用エネルギーを保持する変数。

- **double** rhf_mon_time

  モノマー RHF 計算にかかった時間（秒)。

- **double** rhf_mon_time_esp

  モノマー RHF 計算における環境静電ポテンシャルの計算にかかった時間（秒)。

- **double** rhf_mon_time_esp_4

  モノマー RHF 計算における、4 中心積分による環境静電ポテンシャルの計算にかかった時間（秒)。

- **double** rhf_mon_time_esp_3

  モノマー RHF 計算における、3 中心積分の近似による環境静電ポテンシャルの計算にかかった時間（秒)。

- **double** rhf_mon_time_esp_m

  モノマー RHF 計算における、点電荷近似による環境静電ポテンシャルの計算にかかった時間（秒)。

- **double** rhf_mon_time_esp_u

  モノマー RHF 計算における、原子核による環境静電ポテンシャルの計算にかかった時間（秒)。

- **double** rhf_mon_time_esp_ext

  モノマー RHF 計算における、外部静電ポテンシャルの計算にかかった時間（秒)。

- **double** rhf_mon_time_scf

  モノマー RHF 計算における、SCF にかかった時間（秒)。

- **double** rhf_mon_time_grad

  モノマー RHF 計算における一次微分の計算にかかった時間（ 秒)。

- **double** rhf_dim_time

  ダイマー RHF 計算にかかった時間（ 秒)。

- **double** rhf_dim_time_esp

  ダイマー RHF 計算における環境静電ポテンシャルの計算にかかった合計時間（ 秒)。

- **double** rhf_dim_time_esp_4

  ダイマー RHF 計算における、4 中心積分による環境静電ポテンシャルの計算にかかった時間（ 秒 ）。

- **double** rhf_dim_time_esp_3

  ダイマー RHF 計算における、3 中心積分の近似による環境静電ポテンシャルの計算にかかった時間（ 秒 ）。

- **double** rhf_dim_time_esp_m

  ダイマー RHF 計算における、点電荷近似による環境静電ポテンシャルの計算にかかった時間（ 秒 ）。

- **double** rhf_dim_time_esp_u

  ダイマー RHF 計算における、原子核による環境静電ポテンシャルの計算にかかった時間（ 秒 ）。

- **double** rhf_dim_time_scf

  ダイマー RHF 計算における SCF イタレーションにかかった時間（ 秒 ）。

- **double** rhf_dim_time_grad

  ダイマー RHF 計算における一次微分の計算にかかった時間（ 秒 ）。

20 MP2 計算

- **int** cmp2_chk

  Canonical MP2 計算を行うか否かのフラグ。

- **int** cmp2_lprint_1

  モノマー MP2 計算のプリントフラグ。

- **int** cmp2_lprint_2

  ダイマー MP2 計算のプリントフラグ。

- **double** cmp2_th_iajs

  MP2 計算の積分変換の閾値の 1 つ。

- **double** cmp2_th_iars

  MP2 計算の積分変換の閾値の 1 つ。

- **double** cmp2_th_pqrs

  MP2 計算の積分変換の閾値の 1 つ。

- **double** cmp2_grimme_ps = 1.2000000000000000

  Grimme 等によって提案された、SCS-MP2 計算におけるスピン反平行の電子ペアの電子相関に対するスケーリング係数。

- **double** cmp2_grimme_pt = 0.3333333333333333

  Grimme 等によって提案された、SCS-MP2 計算におけるスピン平行の電子ペアの電子相関に対するスケーリング係数。

- **double** cmp2_jung_ps = 1.3000000000000000

  Jung 等によって提案された、SCS-MP2 計算におけるスピン反平行の電子ペアの電子相関に対するスケーリング係数。

- **double** cmp2_jung_pt = 0.0000000000000000

  Jung 等によって提案された、SCS-MP2 計算におけるスピン平行の電子ペアの電子相関に対するスケーリング係数。

- **double** cmp2_hill_ps = 0.0000000000000000

Hill 等によって提案された、SCS-MP2 計算におけるスピン反平行の電子ペアの電子相関に対するスケーリング係数。

- **double** cmp2_hill_pt = 1.7600000000000000

  Hill 等によって提案された、SCS-MP2 計算におけるスピン平行の電子ペアの電子相関に対するスケーリング係数。

- **double *** cmp2_mon_energy_s ([frag_n])

  モノマーの MP2 計算の電子相関エネルギーのうち、スピン反平行の電子ペアの電子相関エネルギーが記録される配列。

- **double *** cmp2_mon_energy_t ([frag_n])

  モノマーの MP2 計算の電子相関エネルギーのうち、スピン平行の電子ペアの電子相関エネルギーが記録される配列。

- **double *** cmp2_mon_energy_vac_cp_s ([frag_n])

  真空中で計算されたモノマーの MP2 計算の電子相関エネルギーのうち、スピン反平行の電子ペアの電子相関エネルギーが記録される配列。BSSE を見積もる際に使用される。

- **double *** cmp2_mon_energy_vac_cp_t ([frag_n])

  真空中で計算されたモノマーの MP2 計算の電子相関エネルギーのうち、スピン平行の電子ペアの電子相関エネルギーが記録される配列。BSSE を見積もる際に使用される。

- **double *** cmp2_dim_ifie_s ([(frag_n - 1) * frag_n / 2])

  MP2 電子相関エネルギーの IFIE うち、スピン反平行の電子ペアからの寄与が記録される配列。dimer_es_calculation 関数および cmp2_dimer 関数で結果がセットされる。

- **double *** cmp2_dim_ifie_t ([(frag_n - 1) * frag_n / 2])

  MP2 電子相関エネルギーの IFIE のうち、スピン平行の電子ペアからの寄与が記録される配列。dimer_es_calculation 関数および cmp2_dimer 関数で結果がセットされる。

- **double *** cmp2_dim_ifie_cp_s ([(frag_n - 1) * frag_n / 2])

  MP2 相関エネルギーの IFIE に対する BSSE の値。

- **double *** cmp2_dim_ifie_cp_t ([(frag_n - 1) * frag_n / 2])

  MP2 相関エネルギーの IFIE に対する BSSE の値。

- **double** cmp2_fmo_1_s

FMO1–MP2 の電子相関エネルギーのうち、スピン反平行の電子ペアの電子相関エネルギーが記録される配列。

- **double** cmp2_fmo_1_t

  FMO1–MP2 の電子相関エネルギーのうち、スピン平行の電子ペアの電子相関エネルギーが記録される配列。

- **double** cmp2_fmo_2_s

  FMO2–MP2 の電子相関エネルギーのうち、スピン反平行の電子ペアの電子相関エネルギーが記録される配列。

- **double** cmp2_fmo_2_t

  FMO2–MP2 の電子相関エネルギーのうち、スピン平行の電子ペアの電子相関エネルギーが記録される配列。

- **double** cmp2_mon_time

  モノマー MP2 計算にかかった時間（ 秒 ）。

- **double** cmp2_dim_time

  ダイマー MP2 計算にかかった時間（ 秒 ）。

21 RI–MP2

- **int** ri_cmp2_chk

  RI–MP2 計算を行うか否かのフラグ。

- **int** ri_cmp2_lprint_1

  モノマー RI–MP2 計算のプリントフラグ。

- **int** ri_cmp2_lprint_2

  ダイマー RI–MP2 計算のプリントフラグ。

- **double *** ri_cmp2_mon_energy_s ([frag_n])

  モノマーの RI–MP2 計算の電子相関エネルギーのうち、スピン反平行の電子ペアの電子相関エネルギーが記録される配列。

- **double *** ri_cmp2_mon_energy_t ([frag_n])

  モノマーの RI–MP2 計算の電子相関エネルギーのうち、スピン平行の電子ペアの電子相関エネルギーが記録される配列。

- **double *** ri_cmp2_mon_energy_vac_cp_s ([frag_n])

  真空中で計算されたモノマーの RI–MP2 計算の電子相関エネルギーのうち、スピン反平行の電子ペアの電子相関エネルギーが記録される配列。BSSE の補正の見積もりで使用される。

- **double *** ri_cmp2_mon_energy_vac_cp_t ([frag_n])

  真空中で計算されたモノマーの RI–MP2 計算の電子相関エネルギーのうち、スピン平行の電子ペアの電子相関エネルギーが記録される配列。BSSE の補正の見積もりで使用される。

- **double *** ri_cmp2_dim_ifie_s ([(frag_n - 1) * frag_n / 2])

  RI–MP2 の IFIE のうち、スピン反平行の電子ペアの電子相関エネルギーが記録される配列。ri_cmp2_dimer 関数で結果がセットされる。

- **double *** ri_cmp2_dim_ifie_t ([(frag_n - 1) * frag_n / 2])

  RI–MP2 の IFIE のうち、スピン平行の電子ペアの電子相関エネルギーが記録される配列。ri_cmp2_dimer 関数で結果がセットされる。

- **double *** ri_cmp2_dim_ifie_cp_s ([(frag_n - 1) * frag_n / 2])

  RI–MP2 の IFIE における BSSE の値のうちスピン反平行のペアからの寄与。

- **double** * ri_cmp2_dim_ifie_cp_t ([(frag_n - 1) * frag_n / 2])

  RI–MP2 の IFIE における BSSE の値のうちスピン平行のペアからの寄与。

- **double** ri_cmp2_fmo_1_s

  FMO1–RI–MP2 の電子相関エネルギーのうち、スピン反平行の電子ペアの電子相関エネルギーが記録される配列。

- **double** ri_cmp2_fmo_1_t

  FMO1–RI–MP2 の電子相関エネルギーのうち、スピン平行の電子ペアの電子相関エネルギーが記録される配列。

- **double** ri_cmp2_fmo_2_s

  FMO2–RI–MP2 の電子相関エネルギーのうち、スピン反平行の電子ペアの電子相関エネルギーが記録される配列。

- **double** ri_cmp2_fmo_2_t

  FMO1–RI–MP2 の電子相関エネルギーのうち、スピン平行の電子ペアの電子相関エネルギーが記録される配列。

- **double** ri_cmp2_mon_time

  モノマーの RI–MP2 計算にかかった時間（秒）。

- **double** ri_cmp2_dim_time

  ダイマーの RI–MP2 計算にかかった時間（秒）。

22 局在化 MP2 計算

- **int** lmp2_chk

  LMP2 計算を行うか否かのフラグ。

- **int** lmp2_ch_worb

  LMP2 計算で使用された局在化軌道をファイルに書き出すか否かのフラグ。

- **int** lmp2_lprint_1

  モノマー LMP2 計算のプリントフラグ。-1 だと推奨の出力となる。

- **int** lmp2_lprint_2

  ダイマー LMP2 計算のプリントフラグ。-1 だと推奨の出力となる。

- **int** lmp2_lprint_loc_1

  モノマー LMP2 計算における局在化のプリントフラグ。-1 だと推奨の出力となる。

- **int** lmp2_lprint_loc_2

  ダイマー LMP2 計算における局在化のプリントフラグ。-1 だと推奨の出力となる。

- **int** lmp2_loc

  LMP2 計算における局在化の手法のフラグ。

  ```
  0 : population localization
  1 : moment localization
  2 : not locallizatin
  ```

  局在化しないのはデバッグ用。

- **int** lmp2_max_itr

  LMP2 の線形方程式を解く際のイタレーションの回数の最大値。

- **double** lmp2_th_1

  LMP2 計算に使用される閾値。

- **double** lmp2_th_1_dim

  LMP2 計算に使用される閾値。ダイマー計算におけるフラグメント間の軌道ペアに適用される。

- **double** lmp2_th_2

  LMP2 計算に使用される閾値。

- **double** lmp2_th_2_dim

  LMP2 計算に使用される閾値。ダイマー計算におけるフラグメント間の軌道ペアに適用される。

- **double** lmp2_th_3

  LMP2 計算に使用される閾値。

- **double** lmp2_th_4

  LMP2 計算に使用される閾値。

- **double** lmp2_th_l1

  LMP2 計算に使用される閾値。

- **double** lmp2_th_l2

  LMP2 計算に使用される閾値。

- **double** * lmp2_mon_energy ([frag_n])

  モノマーの LMP2 計算の相関エネルギーが記録される配列。

- **double** * lmp2_dim_ifie_sum ([(frag_n - 1) * frag_n / 2])

  LMP2 計算で得られた IFIE が記録される配列。

- **double** * lmp2_dim_ifie_sub ([(frag_n - 1) * frag_n / 2])

  LMP2 計算で得られた IFIE が記録される配列。

- **double** lmp2_fmo_1

  LMP2 計算の FMO-1 相関エネルギーが記録される変数。

- **double** lmp2_fmo_2_sum

  LMP2 計算の FMO-2 相関エネルギーが記録される変数。

- **double** lmp2_fmo_2_sub

  LMP2 計算の FMO-2 相関エネルギーが記録される変数。

- **double** lmp2_mon_time

  モノマーの LMP2 計算にかかった時間（秒）。

- **double** lmp2_dim_time

  ダイマーの LMP2 計算にかかった時間（秒）。

関数

1 List

2.1	void	main
3.1	void	initialize_mpi
3.2	void	finalize_mpi
4.1	void	initialize_memory
4.2	void	finalize_memory
4.3	double *	memory_alloc
4.4	int	memory_dealloc
4.5	long int	get_remain_memory
4.6	double *	memory_get_pointer
4.7	void	memory_shift_left
4.8	int	memory_check_val_name
4.9	void	memory_print
4.10	int	memory_get_nval
4.11	void	memory_get_val_name
4.12	int	memory_get_val_size
5.1	void	paics_main
5.2	void	print_paics_title
5.3	void	print_paics_tiltle_log_file
5.4	void	initialize_paics
5.5	void	paicd_finalize
5.6	void	initialize_angular_parameter
5.7	void	initialize_basis_label
5.8	void	initialize_atomic_label
5.9	double	calc_nuclear_repulsion_1
5.10	double	calc_nuclear_repulsion_2
5.11	void	nuclear_grad
5.12	void	calc_nuclear_repulsion_grad_1
5.13	double	calc_nuclear_repulsion_grad_2
5.14	double	calc_nuclear_repulsion_grad_3
5.15	double	calc_nuclear_repulsion_grad_4
5.16	void	calc_nuclear_esp
5.17	void	calc_nuclear_efi

5.18	double	inter_fragment_distance_nucleus
5.19	double	inter_fragment_distance_basis
5.20	void	make_diagonal_matrix
5.21	void	make_inverse_matrix
5.22	void	make_sqrt_matrix
5.23	void	make_inverse_sqrt_matrix
5.24	void	make_d_matrix
5.25	void	make_d_matrix_energy_weight
5.26	void	make_mulliken_population
5.27	void	make_electronic_charge_center
5.28	void	unite_density_matrix
5.29	void	unite_c_matrix
5.30	int	get_nkab
5.31	void	make_kab
5.32	void	transformation_spherical_2
5.33	void	transformation_spherical_2_mom
5.34	void	transformation_spherical_3
5.35	void	transformation_spherical_4
5.36	void	make_cauchy
5.37	void	get_basis_xyz_value
5.38	void	get_orbital_xyz_value
5.39	void	paics_int_to_char
5.40	int	paics_check_char_exist
5.41	int	paics_get_range_char
5.42	int	paics_dgemm
5.43	int	paics_daxpy
5.44	int	paics_daxpy
5.45	int	get_frag_pair_nuclear_repulsion
5.46	void *	basis_nucleus_assign
5.47	void	basis_ini_assign
5.48	void	basis_define_assign
5.49	void	basis_normalize

6.1	void	read_input
6.2	void	read_input_list_01
6.3	void	read_input_list_02
6.4	void	check_input_validity_01
6.5	void	check_input_validity_02

6.6	int	check_open_file_r
6.7	int	check_open_file_w
6.8	void	read_atom
6.9	void	read_basis
6.10	void	read_nucleus
6.11	void	read_fragment
6.12	void	read_ext_point_charge
6.13	void	read_position
6.14	void	read_basis_define
6.15	void	read_basis_define_file
6.16	void	read_monomer_scc_result
7.1	void	print_input_information
7.2	void	print_input_information_rhf
7.3	void	print_input_information_cmp2
7.4	void	print_input_information_ri_cmp2
7.5	void	print_input_information_lmp2
7.6	void	print_nucleus_coordinate
7.7	void	print_basis_set_coordinate
7.8	void	print_basis_set_definition
7.9	void	print_ext_point_charge_coordinate
7.10	void	print_fragment_definition
7.11	void	print_memory_information
7.12	void	print_frag_information
7.13	void	print_frag_pair_information
7.14	void	print_monomer_scc_result
7.15	void	print_monomer_calculation_result
7.16	void	print_dimer_es_result
7.17	void	print_dimer_calculation_result
7.18	void	print_dimer_ifie_result
7.19	void	print_molecular_orbital
7.20	void	print_monomer_mulliken_population
7.21	void	write_monomer_scc_result
7.22	void	write_grid_rhf_monomer_orbital
7.23	void	write_grid_orbital
8.1	void	initialize_fmt
8.2	void	fmt

9.1	void	ovl_recursive
9.2	void	kei_recursive
9.3	void	nai_recursive
9.4	void	efi_recursive
9.5	void	mom_recursive
9.6	void	get_ovl
9.7	void	get_kei
9.8	void	get_nai
9.9	void	get_efi
9.10	void	get_mom
9.11	void	make_ovl_matrix
9.12	void	make_ovl_matrix_inter
9.13	void	make_kei_matrix
9.14	void	make_nai_matrix
9.15	void	make_mom_matrix
10.1	void	ovl_grad_recursive
10.2	void	kei_grad_recursive
10.3	void	nai_grad_recursive
10.4	void	get_ovl_grad
10.5	void	get_kei_grad
10.6	void	get_nai_grad
11.1	void	get_eri
11.2	void	get_eri_2cen
11.3	void	get_eri_3cen
11.4	void	eri_recursive_gen
11.5	void	eri_recursive_2cen_gen
11.6	void	eri_recursive_3cen_gen
11.7	void	eri_recursive
11.8	void	eri_recursive
11.9	void	eri_recursive_3cen
11.10	void	eri_recursive_3cen
12.1	void	make_esp_4_matrix
12.2	void	make_esp_3_matrix

12.3	void	make_esp_m_matrix
13.1	void	projection_operator_initialize
13.2	void	make_projection_operator
13.3	void	mkpro_get_iatom
13.4	void	mkpro_make_ch4_coordinate
13.5	void	mkpro_make_ch2o_coordinate
13.6	void	mkpro_rhf_ch4
13.7	void	mkpro_rhf_ch2o
13.8	void	mkpro_print
13.9	int	get_frag_n_projection
13.10	void	make_frag_projection
13.11	void	get_dimer_n_projection
13.12	void	make_dimer_projection
13.13	void	make_projection_matrix
14.1	void	initialize_fragment
14.2	int	get_frag_nch1_all
14.3	int	get_frag_nba1_all
14.4	int	get_frag_nsh1_all
14.5	int	get_frag_nbo1_all
14.6	int	get_frag_nbot_all
14.7	int	get_frag_nbo2_all
14.8	int	get_frag_nele_all
14.9	int	get_frag_nsh1
14.10	void	get_frag_nsh1_aux
14.11	int	get_frag_nbo1
14.12	void	get_frag_nbo1_aux
14.13	void	make_frag_shell
14.14	void	make_frag_aux_shell
14.15	void	make_frag_bo
14.16	void	make_frag_scf_esa_list
14.17	void	make_frag_calc_order
14.18	int	get_frag_n_nucleus
14.19	void	make_frag_nucleus
14.20	int	make_frag_esp_list_0
14.21	int	make_frag_pair_basis
14.22	void	make_frag_pair_aux_basis

14.23	void	make_frag_pair_shell
14.24	void	get_frag_pair_aux_nsh1
14.25	void	make_frag_pair_aux_shell
14.26	void	make_frag_pair_calc_order_n
14.27	void	make_frag_pair_calc_order
14.28	int	get_frag_pair_n_nucleus
14.29	int	make_frag_pair_nucleus
14.30	void	make_frag_pair_esp_list_0
14.31	void	chk_frag_pair_calc
14.32	void	chk_frag_pair_calc_all
14.33	void	chk_frag_esp_4
14.34	void	chk_frag_esp_3
14.35	void	chk_frag_dscf
14.36	void	get_frag_pair_index
14.37	void	init_frag_esp_list_n
14.38	void	init_frag_esp_list
14.39	void	init_frag_dscf_list_n
14.40	void	init_frag_dscf_list
14.41	void	get_frag_from_charge
14.42	void	get_frag_from_basis

15.1	void	initialize_monomer_scc
15.2	void	monomer_scc_print_title
15.3	void	calculation_monomer_scc
15.4	void	monomer_scc_result_bcast

16.1	void	calculation_monomer
16.2	void	monomer_print_title
16.3	void	monomer_result_bcast_rhf
16.4	void	monomer_result_bcast_cmp2
16.5	void	monomer_result_bcast_ri_cmp2
16.6	void	monomer_result_bcast_lmp2

17.1	void	calculation_dimer_es
17.2	void	dimer_es_print_title
17.3	void	dimer_es_4
17.4	void	dimer_es_u

18.1	void	calculation_dimer
18.2	void	dimer_print_title
18.3	void	dimer_result_bcast_rhf
18.4	void	dimer_result_bcast_cmp2
18.5	void	dimer_result_bcast_ri_cmp2
18.6	void	dimer_result_bcast_lmp2

19.1	void	initialize_rhf
19.2	void	rhf_monomer_scc
19.3	void	rhf_monomer
19.4	void	rhf_monomer_field_property
19.5	void	rhf_dimer
19.6	void	rhf_dimer_field_property
19.7	void	make_init_mo_hcore
19.8	void	make_init_mo_rhf
19.9	void	make_ini_shell
19.10	void	pro_out_init_mo
19.11	void	rhf_iteration
19.12	void	make_g_matrix
19.13	void	add_g_mat
19.14	void	make_effective_fock_diis
19.15	void	rhf_grad_add_ovl
19.16	void	rhf_grad_add_kei
19.17	void	rhf_grad_add_nai
19.18	void	rhf_grad_add_efi
19.19	void	ortonormalize_can
19.20	void	ortonormalize_sym
19.21	void	ortonormalize_low
19.22	void	rhf_make_init_density_matrix
19.23	void	rhf_make_init_density_matrix_basis

20.1	void	localize_mom
20.2	void	localize_pop
20.3	void	make_site_localize_orbital
20.4	void	print_localized_orbital

21.1	void	initialize_cmp2
21.2	void	cmp2_monomer
21.3	void	cmp2_dimer
21.4	void	canonical_mp2
21.5	long int	cmp2_get_remain_core
21.6	void	cmp2_core_print
21.7	void	cmp2_core_alloc
21.8	void	cmp2_core_shift_left
21.9	void	cmp2_core_dealloc
21.10	double *	cmp2_core_get_pointer
21.11	long int	cmp2_core_get_size
21.12	void	cmp2_integral_transformation
21.13	void	cmp2_get_pqrs_fixed_rs
21.14	void	cmp2_allreduce_double

22.1	void	initialize_ri_cmp2
22.2	void	ri_cmp2_monomer
22.3	void	ri_cmp2_dimer
22.4	void	ri_canonical_mp2
22.5	long int	ri_cmp2_get_remain_core
22.6	void	ri_cmp2_core_print
22.7	void	ri_cmp2_core_alloc
22.8	void	ri_cmp2_core_shift_left
22.9	void	ri_cmp2_core_dealloc
22.10	double *	ri_cmp2_core_get_pointer
22.11	long int	ri_cmp2_core_get_size
22.12	void	ri_cmp2_get_rsq_fixed_q
22.13	void	ri_cmp2_make_biap
22.14	void	ri_cmp2_allreduce_double

23.1	void	initialize_lmp2
23.2	void	lmp2_monomer
23.3	void	lmp2_dimer
23.4	void	lmp2_make_ifie
23.5	void	lmp2_dimer_analysis
23.6	void	local_mp2
23.7	long int	lmp2_get_remain_core
23.8	void	lmp2_core_print

23.9	void	lmp2_core_alloc
23.10	void	lmp2_core_shift_left
23.11	void	lmp2_core_dealloc
23.12	double *	lmp2_core_get_pointer
23.13	long int	lmp2_core_get_size
23.14	void	lmp2_make_p
23.15	void	lmp2_make_ovl
23.16	void	lmp2_make_fck
23.17	void	lmp2_make_fck_loc
23.18	void	lmp2_make_basis_pop
23.19	void	lmp2_make_orbital_center
23.20	void	lmp2_make_domain
23.21	void	lmp2_make_domain_ao
23.22	void	lmp2_make_ij_pair
23.23	void	lmp2_make_ij_domain
23.24	void	lmp2_make_ij_domain_ao
23.25	void	lmp2_make_ij_idx
23.26	void	lmp2_make_ij_lst
23.27	void	lmp2_print_input
23.28	void	lmp2_make_c_max
23.29	void	lmp2_make_kmat_ao_idx
23.30	void	lmp2_make_kmat
23.31	void	lmp2_make_ij_batch
23.32	void	lmp2_make_c_mat_pair
23.33	void	lmp2_add_kmat_ao
23.34	void	lmp2_make_tmat
23.35	void	lmp2_make_umat_ij
23.36	void	lmp2_make_rmat_ij
23.37	void	lmp2_make_tmat_ij
23.38	void	lmp2_make_pair_energy
23.39	void	lmp2_set_output
23.40	void	lmp2_allreduce_double
23.41	void	print_domain_information
23.42	void	print_ij_domain_information
23.43	void	lmp2_dimer_write_orbital_file

2 main 関数

2.1 void main

引数 2

1	**const int**	argc	引数の数
2	**const char ***	argv [1]	入力ファイルの名前

プログラムの最上位の関数。MPI の初期化と終了、入力ファイル名の取得のみを行い、paics_min 関数に制御を移す。

3 並列制御

3.1 void initialize_mpi

引数 無し

MPI 並列に関連する初期設定を行う関数。以下のグローバル変数に値が代入される。

```
int      para_mpi_my_rank
int      para_mpi_nproc
char     para_mpi_proc_name [ PARA_MPI_MAX_PROC ] [ 128 ]
MPI_Comm para_mpi_single_comm
int      para_mpi_print_rank
```

3.2 void finalize_mpi

引数 無し

MPI 並列に関連する終了処理を行う。MPI_Finalize 関数が呼ばれるだけ。MPI_Finalize 関数は、main 関数の最後で呼ばれるが、途中で強制終了する場合は、main 関数の最後までたどり着かずに終了する。このように、main 関数の最後までたどり着かずにプログラムを終了する場合、この関数を呼ぶ。

4 メモリ制御

4.1 void initialize_memory

引数 1

1	**const char ***	filename	入力ファイルの名前

メモリ制御に関する初期設定を行う関数。入力ファイルを直接オープンし「mem_mbyte」キーワードの数値を読みグローバル変数の mem_mbyte に代入する。mem_core に、指定されたサイズのメモリを allocate し、グローバル変数の mem_size に double 型単位でのサイズを代入する。グローバル変数の mem_nval もここで 0 に初期化される。これ以降、プログラムでメモリを使用する場合は、mem_core に割り振られた領域が使用され、個々に大きなメモリを allocate することは無い。

4.2 void finalize_memory

引数 無し

メモリ制御に関連する終了操作をする。initialize_memory で確保したメモリ領域を開放する。

4.3 double * memory_alloc

引数 2

1	**const char ***	[40]	name	39 文字以内の文字列
2	**const long int**		size	確保したい配列のサイズ (double 型での個数)

プログラム開始時に確保したメモリ領域（mem_core）に、指定したサイズの領域を割りあてる関数。確保されたメモリ領域の先頭アドレスを double * 型のポインタで返す。***PAICS*** で、最も多く使用される関数。現在、メモリが確保出来なかった場合はプログラムを強制的に止める仕様になっている（今後、改善する必要有り）。***PAICS*** では、メモリ管理を徹底するため、個別に allocate 関数は呼ばず、必ずこの関数を通じて各配列にメモリ領域を割り振る。基本的に、適切な型のポインタ変数へキャストし、以下のようにメモリ領域の先頭アドレスを取得する。

```
[ポインタ] = ([ 型 * ]) memory_alloc("[ 配列名 ]", (long int) [サイズ]) ;
```

double 型でいくつ分かを指定することになるが、各型のビット数を知っておく必要がある。32bit

環境と 64bit 環境では型のサイズが違うが、64bit 環境でのサイズに合わていれば、32bit 環境でも動作するはず。64bit 環境における C 言語の型のサイズを**表 38** にまとめる。

4.4 int memory_dealloc

引数 1

1	**const char ***	name	39 文字以内の文字列

memory_alloc 関数で確保されたメモリ領域を開放する関数。その領域よりも後で確保された領域が、まだ残っている場合、空いた領域を「埋める」ためにコピーが発生する（従って、大きな領域を確保する場合はその順番に注意する）。指定された名前の領域が未確保の場合は、プログラムを強制終了する（この場合、プログラムコードにバグがあるということ）。

4.5 long int get_remain_memory

引数 無し

mem_core のうち未使用のサイズを、double 型の個数で返す。

4.6 double * memory_get_pointer

引数 1

1	**const char ***	name	39 文字以内の文字列

mem_core に確保されている領域のうち引数 name の領域の先頭アドレスを、double 型のポインタで返す。通常は、memory_alloc で確保された時にアドレスがセットされるが、「内側の領域」が解放された場合、それ以降の領域の先頭アドレスがズレる。この場合、memory_get_pointer を使用して先頭アドレスを再取得する必要がある。引数 name

引数 2

1	**const long int**	pos	mem_core 配列上の位置
2	**const long int**	n	配列を左側にシフトする数

mem_core のうち、pos 以降の要素を n 個だけ左側（つまり、若い位置）にずらす。つまり、以下のようなコードが実行される。

```
for (i=pos;i<top;i++) *(mem_core+i-n)=(*(mem_core+i)) ;
```

基本的に、この関数を直接使うことはなく、memory_dealloc 関数の中で呼ばれる。

4.8 int memory_check_val_name

引数 1

1	**const char ***	name	[40]	39 文字以内の文字列

引数（name）で指定した名前の領域が確保されているか否かを調べるための関数。確保されていれば、何番目に確保されているかを返す（0 以上の整数）。確保されていなければ-1 を返す。

4.9 void memory_print

引数 無し

現在確保されている変数を標準出力にプリントする。主にデバッグ用。

4.10 int memory_get_nval

引数 無し

現在確保されている領域の数（mem_nval）を返す。

4.11 void memory_get_val_name

引数 2

1	**const int**	ival		領域の番号
2	**char ***	name	[40]	出力バッファー

現在確保されている領域のうち、ival 番目の領域の名前を、出力バッファー（name）に代入して返す。

4.12 int memory_get_val_size

引数 1

1	**const int**	ival	領域の番号

現在確保されている領域のうち、ival 番目の領域のサイズを、double 型何個分かで返す。

表 38: 64 bit 環境での C 言語の変数型のサイズ。

char	1 バイト
short int	2 バイト
int	4 バイト
long int	8 バイト
double	8 バイト

5 *PAICS* 全般

5.1 void paics_main

引数 1

1	**const char ***	filename	[128]	入力リストが記述されたファイル名

FMO 計算を行う部分の最上位関数。

5.2 void print_paics_title

引数 無し

タイトルを出力する関数。

5.3 void print_paics_tiltle_log_file

引数 無し

PAICS のタイトルを log ファイルに書き出す関数。

5.4 void initialize_paics

引数 1

1	**const char ***	filename	入力ファイルの名前

PAICS の初期化を行う関数。

5.5 void paicd_finalize

引数 5

1	**const int**	n		核電荷の数
2	**const double ***	charge	[n]	核電荷の電荷
3	**const double ***	x	[n]	核電荷の x 座標
4	**const double ***	y	[n]	核電荷の y 座標
5	**const double ***	z	[n]	核電荷の z 座標

与えられた核電荷による核反発エネルギーを計算して返す関数。x 座標、y 座標、z 座標が別々の配列として渡される。

5.10 double calc_nuclear_repulsion_2

引数 3

1	**const int**	n		核電荷の数
2	**const double ***	charge	[n]	核電荷の電荷
3	**const double ***	xyz	[n * 3]	核電荷の座標

与えられた核電荷による核反発エネルギーを計算して返す関数。x 座標、y 座標、z 座標が同じ配列として渡される。

5.11 void nuclear_grad

引数 2

1	**const MPI_Comm**	icomm	MPI コミュニケーター
2	**double ***	grad	出力バッファー

系全体の核反発エネルギーの原子核による 1 次微分を計算して返す関数。この中で、必要に応じて、calc_nuclear_repulsion_grad_* 関数が呼ばれる。

5.12 void calc_nuclear_repulsion_grad_1

引数 5

1	**const int**	n		核電荷の数
2	**const double ***	charge	[n]	核電荷の電荷
3	**const double ***	x	[n]	核電荷の x 座標

4	**const double ***	y	[n]	核電荷の y 座標
5	**const double ***	z	[n]	核電荷の z 座標
6	**double ***	grad	[3* n]	出力バッファー

核反発エネルギーの原子核による 1 次微分を計算して順に出力バッファーに足し込む関数。x 座標、y 座標、z 座標が別々の配列として渡される。

5.13 double calc_nuclear_repulsion_grad_2

引数 3

1	**const int**	n		核電荷の数
2	**const double ***	charge	[n]	核電荷の電荷
3	**const double ***	xyz	[n * 3]	核電荷の座標
4	**double ***	grad	[n * 3]	出力

核反発エネルギーの 1 次微分を計算して順に出力バッファーに足し込む関数。x 座標、y 座標、z 座標が同じ配列として渡される。

5.14 double calc_nuclear_repulsion_grad_3

引数 5

1	**const int**	n		核電荷の数
2	**const double ***	charge	[n]	核電荷の電荷
3	**const double ***	charge_i	[n]	原子核の番号との対応
4	**const double ***	xyz	[n * 3]	核電荷の座標
5	**double ***	grad	[n * 3]	出力

核反発エネルギーの 1 次微分を計算し、出力バッファーに足し込む関数。charge_i は、与えられた原子核のチャージが全体の通し番号で何番目かを格納しており、この値に従った順番で出力バッファーに足し込む。

5.15 double calc_nuclear_repulsion_grad_4

引数 9

1	**const int**	n1		核電荷の数
3	**const int ***	charge_i1	[n2]	原子核の番号との対応

2	**const double ***	charge1	[n2]	核電荷の電荷
4	**const double ***	xyz1	[n2 * 3]	核電荷の座標
5	**const int**	n2		核電荷の数
7	**const int ***	charge_i2	[n2]	原子核の番号との対応
6	**const double ***	charge2	[n2]	核電荷の電荷
8	**const double ***	xyz2	[n2 * 3]	核電荷の座標

原子核から構成される 2 つのグループ間（通常は 2 つのフラグメント間）の核反発エネルギーの 1 次微分を計算して返す関数。charge_i1 および charge_i2 は、与えられた原子核が全体の通し番号で何番目かが格納されており、この値に従った順番で出力バッファーに足し込んでいく。

5.16 void calc_nuclear_esp

引数 11

1	**const int**	cn
2	**const double ***	charge
3	**const double ***	cx
4	**const double ***	cy
5	**const double ***	cz
6	**const int ***	n
7	**const double ***	x
8	**const double ***	y
9	**const double ***	z
10	**double ***	esp
11	**int ***	chk

複数の原子核が任意の位置に作る静電ポテンシャルを計算する関数。

5.17 void calc_nuclear_efi

引数 13

1	**const int**	cn
2	**const double ***	charge
3	**const double ***	cx
4	**const double ***	cy
5	**const double ***	cz
6	**const int ***	n
7	**const double ***	x
8	**const double ***	y
9	**const double ***	z
10	**double ***	efi_x
11	**double ***	efi_y

12	**double ***	efi_z	
13	**int ***	chk	

複数の原子核が任意の位置に作る電場を計算する関数。

5.18 double inter_fragment_distance_nucleus

引数 4

1	**const int**	ifrag		フラグメント番号
2	**const int**	jfrag		フラグメント番号
3	**int ***	ich	[1]	最近接の核電荷の番号
4	**int ***	jch	[1]	最近接の核電荷の番号

2 つのフラグメントの間の距離を返す関数。各フラグメントに属する原子核間の距離のうち最短の距離を返す。結合の切断箇所では、1 つの原子核が 2 つのフラグメントにシェアされるので、ifrag ~~=~~ jfrag の場合でも 0.0 になる場合がある。ich と jch には最近接だった核電荷の番号（全体の通し番号）が入る。

5.19 double inter_fragment_distance_basis

引数 4

1	**const int**	ifrag		フラグメント番号
2	**const int**	jfrag		フラグメント番号
3	**int ***	iba	[1]	最近接の基底関数の番号
4	**int ***	jba	[1]	最近接の基底関数の番号

2 つのフラグメントの間距離を返す関数。各フラグメントに属する基底関数間の距離のうち最短の距離を返す。結合の切断箇所では、1 つの基底関数が 2 つのフラグメントにシェアされるので、ifrag ~~=~~ jfrag の場合でも 0.0 になる場合がある。iba と jba には最近接だった基底関数の番号（全体の通し番号）が入る。原則的には inter_fragment_distance_charge と同じ値を返す（原子核が無い位置に基底関数だけ置くような場合は異なる値になる可能性がある）。

5.20 void make_diagonal_matrix

引数 4

1	**const int**	n		行列の次元数
3	**double ***	m	[n * n]	行列
4	**double ***	val	[n]	対角要素の値
5	**double ***	w	[n * n]	作業領域

実対称行列の対角化を行う関数。この下で、LAPACK の dsyev が呼ばれる。(スパコンなどで特殊な LAPACK を使う時は、場合によってはこの関数を書き換える必要があるかもしれない)。

5.21 void make_inverse_matrix

引数 5

1	**const int**	n		行列の次元数
2	**const double ***	m_in	[n * n]	行列
3	**double ***	m_out	[n * n]	計算された逆行列
4	**double ***	work1	[n * n]	作業領域
5	**double ***	work2	[n * n]	作業領域

実対称行列の逆行列を計算する関数。

5.22 void make_sqrt_matrix

引数 5

1	**const int**	n		行列の次元数
2	**const double ***	m_in	[n * n]	行列
3	**double ***	m_out	[n * n]	計算された逆行列
4	**double ***	work1	[n * n]	作業領域
5	**double ***	work2	[n * n]	作業領域

実対称行列のルートを計算する関数。

5.23 void make_inverse_sqrt_matrix

引数 5

1	**const int**	n		行列の次元数
2	**const double ***	m_in	[n * n]	行列
3	**double ***	m_out	[n * n]	計算された逆行列
4	**double ***	work1	[n * n]	作業領域
5	**double ***	work2	[n * n]	作業領域

実対称行列のルートの逆数を計算する関数。

5.24 void make_d_matrix

引数 4

1	**const int**	nbo		ao の数
2	**const int**	nocc		占有軌道の数
3	**const double ***	c	[nocc * nbo]	係数行列
4	**double ***	d	[nbo * nbo]	計算された密度行列

軌道の係数と占有軌道の数を受け取り、密度行列を計算する関数。「2.0 倍されていない密度行列」を返す（注意）。

5.25 void make_d_matrix_energy_weight

引数 5

1	**const int**	nbo		ao の数
2	**const int**	nocc		占有軌道の数
3	**const double ***	moe	[nocc]	軌道エネルギー
4	**const double ***	c	[nocc * nbo]	係数行列
5	**double ***	d	[nbo * nbo]	計算された密度行列

軌道の係数と占有軌道の数を受け取り、エネルギーの重みを掛けた密度行列を計算する関数。「2.0 を掛け算していない密度行列」を返す（注意）。エネルギー勾配の計算で使用される。

5.26 void make_mulliken_population

引数 7

1	**const int**	nba1		基底関数の数
2	**const int**	nbo1		ao の数
3	**const int ***	basis		基底関数リスト
4	**const double ***	ovl	[nbo1 * nbo1]	重なり行列
5	**const double ***	d	[nbo1 * nbo1]	密度行列
6	**double ***	mulliken_ba	[nba1]	基底関数の mulliken
7	**double ***	mulliken_bo	[nbo1]	ao の mulliken

密度行列を受け取り、Mulliken population を計算する関数。基底関数ごとの Mulliken population と bo ごとの Mulliken population の両方を計算する。「2.0 が掛け算されていない密度行列」を受け取る（注意）。

5.27 void make_electronic_charge_center

引数 9

1	**const int**	nsh1	
2	**const int**	nbo1	
3	**const int**	nocc	
4	**const int ***	shell	[nbo1 * nbo1]
5	**const double ***	shell_xyz	[nbo1 * nbo1]
6	**const double ***	dmat	[nba1]
7	**double ***	x	[nbo1]
8	**double ***	y	[nbo1]
9	**double ***	z	[nbo1]

与えられた電子密度に関する負電荷の中心を計算するプログラム。

5.28 void unite_density_matrix

引数 14

1	**const int**	nbo1
2	**const int**	nbo2
3	**const int**	nob3
4	**const int**	nbasis1
5	**const int**	nbasis2
6	**const int**	nbasis3
7	**const int ***	basis1
8	**const int ***	basis2
9	**const int ***	basis3
10	**const double ***	dmat1
11	**const double ***	dmat2
12	**const int ***	list1
13	**const int ***	list2
14	**const double ***	dmat3

2 つの密度行列を「合わせて」1 つの密度行列にする関数。2 つの密度行列の基底関数空間の和の基底関数空間における密度行列が作られる。FMO 計算では、モノマーの密度行列からダイマーの密度行列を作ることがあるが、この時に使用される。

5.29 void unite_c_matrix

引数 18

1	**const int**	nbo1
2	**const int**	nbo2
3	**const int**	nob3
4	**const int**	nmo1
5	**const int**	nmo2
6	**const int**	nmo3
7	**const int**	nbasis1
8	**const int**	nbasis2
9	**const int**	nbasis3
10	**const int ***	basis1
11	**const int ***	basis2
12	**const int ***	basis3
13	**const double ***	dmat1
14	**const double ***	dmat2
15	**const int ***	list1
16	**const int ***	list2
17	**const double ***	dmat3
18	**const int**	lprint

2 つの分子軌道セットを「合わせて」1 つの分子軌道セットにする関数。2 つの分子軌道セットの基底関数空間の和の基底関数空間における分子軌道を作成する。***PAICS*** では、モノマーの分子軌道を合わせてダイマー計算の初期軌道を作っているが、この際に使用される。この関数では、単純に係数が足し算されるだけで、規格化も直行化もされない。

5.30 int get_nkab

引数 2

1	**const int**	nshell	
2	**const int ***	shell	[nshell * 3]

積分計算に必要なの数値の数を返す関数。

5.31 void make_kab

引数 9

1	**const int**	nshell
2	**const int ***	shell

3	**const double ***	shell_xyz	
4	**int ***	kab_idx	
5	**double ***	kab_sh	出力
6	**double ***	kab	出力
7	**double ***	p_x	出力
8	**doubel ***	p_y	出力
9	**doubel ***	p_z	出力

積分計算に必要な数値を計算する関数。「kab_idx」「kab_sh」「p_x」「p_y」「p_z」を計算する。

5.32 void transformation_spherical_2

引数 8

1	**const int**	la	
2	**const int**	lb	
3	**const int**	na5	
4	**const int**	nb5	
5	**const int**	na6	
6	**const int**	nb6	
7	**const double ***	in	
8	**double ***	out	出力

カルテシアン型（6 成分型）の 2 中心積分を球面調和型（5 成分型）の積分に変換する関数（モーメント積分以外に適用）。

5.33 void transformation_spherical_2_mom

引数 8

1	**const int**	la	
2	**const int**	lb	
3	**const int**	na5	
4	**const int**	nb5	
5	**const int**	na6	
6	**const int**	nb6	
7	**const double ***	in	
8	**double ***	out	出力

カルテシアン型（6 成分型）の 2 中心積分を球面調和型（5 成分型）の積分に変換する関数（モーメント積分に適用）。

5.34 void transformation_spherical_3

引数 11

1	**const int**	la	
2	**const int**	lb	
3	**const int**	lc	
4	**const int**	na5	
5	**const int**	nb5	
6	**const int**	nc5	
7	**const int**	na6	
8	**const int**	nb6	
9	**const int**	nc6	
10	**const double ***	in	
11	**double ***	out	出力

カルテシアン型（6 成分系）の 3 中心積分を球面調和型（5 成分系）の積分に変換する関数。

5.35 void transformation_spherical_4

引数 14

1	**const int**	la	
2	**const int**	lb	
3	**const int**	lc	
4	**const int**	ld	
5	**const int**	na5	
6	**const int**	nb5	
7	**const int**	nc5	
8	**const int**	nd5	
9	**const int**	na6	
10	**const int**	nb6	
11	**const int**	nc6	
12	**const int**	nd6	
13	**const double ***	in	
14	**double ***	out	出力

カルテシアン型（6 成分系）の 4 中心積分を球面調和型（5 成分系）の積分に変換する関数。

5.36 void make_cauchy

引数 11

1	**const int**	nshell

2	**const MPI_Comm**	icomm	
3	**const int ***	shell	
4	**const double ***	shell_xyz	
5	**const int ***	kab_idx	
6	**const double ***	kab	
7	**const double ***	p_x	
8	**const double ***	p_y	
9	**const double ***	p_z	
10	**double ***	cauchy	出力
11	**double ***	cauchy_mpi	作業領域

コーシー・シュバルツの不等式を用いて積分のスクリーニングを行うため、2 インデックスの電子反発積分を計算する関数。

5.37 void get_basis_xyz_value

引数 9

1	**const int**	nsh1
2	**const int**	nbo1
3	**const int ***	shell
4	**const double ***	shell_xyz
5	**const int ***	chk
6	**const double**	x
7	**const double**	y
8	**const double**	z
9	**const double ***	value

ある位置における基底関数の値を計算し返す関数。

5.38 void get_orbital_xyz_value

引数 12

1	**const int**	nsh
2	**const int**	nbo
3	**const double**	coeff_th
4	**const int ***	shell
5	**const double ***	shell_xyz
6	**const double ***	coeff
7	**const double**	x
8	**const double**	y
9	**const double**	z
10	**const int ***	work1
11	**const double ***	work2

12	**const double ***	value

ある位置における分子軌道の値を計算し返す関数。

5.39 void paics_int_to_char

引数 3

1	**const int**	n
2	**const int**	w
3	**char ***	out

int 型の整数を文字列で返す関数。ファイル名を作成する際に使用される。

5.40 int paics_check_char_exist

引数 2

1	**const char**	c
2	**const char ***	in

与えられた文字列の中に、指定された文字があるかどうかを調べる関数。

5.41 int paics_get_range_char

引数 3

1	**const char ***	in
2	**int ***	i1
3	**int ***	i2

整数の範囲を示す文字列が与えられたとき、その範囲の最初の整数と最後の整数を返す関数。入力ファイルを読むときに使用される。

5.42 int paics_dgemm

引数 13

1	**char ***	transa
2	**char ***	transb
3	**long ***	m
4	**long ***	n
5	**long ***	k
6	**double ***	alpha
7	**double ***	a
8	**long ***	lda
9	**double ***	b
10	**long ***	ldb
11	**double ***	beta
12	**double ***	c
13	**long ***	ldc

この下で BLAS の DGEMM が呼ばれる。（スパコンなどで特殊な BLAS を使う時、この関数を書き換えればよい）

5.43 int paics_daxpy

引数 6

1	**long ***	n
2	**double ***	da
3	**double ***	dx
4	**long ***	incx
5	**double ***	dy
6	**long ***	incy

この下で BLAS の DAXPY が呼ばれる。（スパコンなどで特殊な BLAS を使う時、この関数を書き換えればよい）

5.44 int paics_daxpy

引数 5

1	**long ***	n
2	**double ***	dx
3	**long ***	incx
4	**double ***	dy
5	**long ***	incy

この下で BLAS の DDOT が呼ばれる。（スパコンなどで特殊な BLAS を使う時、この関数を書

き換えればよい）

5.45 int get_frag_pair_nuclear_repulsion

引数 2

```
1  const int   ifrag
2  const int   jfrag
```

フラグメントペアの核間反発エネルギーを計算する。

5.46 void * basis_nucleus_assign

引数 無し

どの基底関数がどの原子核の上に置かれたものかを判定する。counterpois 補正用などで、原子核の上に置かれていない基底関数の場合は　1 が格納される。以下のグローバル変数に値がセットされる。

```
int * basis_charge , [ basis_n ] , allocate は read_basis
```

5.47 void basis_ini_assign

引数 無し

設置された各基底関数に、初期軌道を作成するための基底関数を対応付ける。デフォルトでは、初期軌道作成用の基底関数に STO-3G が使用される。以下のグローバル変数に値がセットされる。

```
char * basis_name_ini , [ basis_n * 21 ] , allocate は read_basis
```

5.48 void basis_define_assign

引数 無し

基底関数の定義の番号と、設置された基底関数を対応させる。以下のグローバル変数に値がセットされる。

```
int * basis_type     , [ basis_n ]
int * basis_type_ini , [ basis_n ]
```

5.49 void basis_normalize

引数 無し

基底関数の定義は規格化された原始ガウス関数に対する係数で与えられる。しかし、計算を行う場合は、「生の」原始ガウス関数に対する係数として短縮係数を持ったほうが都合がよいので、基底関数の定義がそうなるように上書きする。また、短縮基底は規格化されている必要は無いが、多くの量子化学計算プログラムは規格化された短縮基底を用いて計算を行う。よって、このプログラムでも規格化されていない短縮基底は規格化し、基底関数の定義を上書きする。以下のグローバル変数の値が書き換えられる。

```
double * bdef_sh_coeff , [ bdef_max * bdef_max_sh * bdef_max_nc ]
```

6 入力

6.1 void read_input

引数 1

1	**const char ***	filename	入力リストが記述されたファイル名

入力ファイルから情報を読む関数。

6.2 void read_input_list_01

引数 1

1	**const char ***	filename	入力リストが記述されたファイル名

入力ファイルから入力リストを読む関数。入力リストに対応するグローバル変数の初期化も行っている。

6.3 void read_input_list_02

引数 1

1	**const char ***	filename	入力リストが記述されたファイル名

入力ファイルから入力リストを読む関数。現在は何もしていない（今後の拡張で利用）。

6.4 void check_input_validity_01

引数 無し

入力ファイルから読まれた内容が適切かどうか判断する。

6.5 void check_input_validity_02

引数 無し

入力ファイルから読まれた内容が適切かどうか判断する。現在はなにもしていない（今後の拡張で利用)。

6.6 int check_open_file_r

引数 無し

1	**const char ***	filename	ファイル名

指定されたファイルが読み込み可能な状況で存在するか否かを調べる関数。

6.7 int check_open_file_w

引数 無し

1	**const char ***	filename	ファイル名

指定されたファイルが書き込み可能な状況で存在するか否かを調べる関数。

6.8 void read_atom

引数 1

1	**const char ***	filename	入力リストが記述されたファイル名

入力ファイルから、基底関数と原子核の情報を読む関数。

6.9 void read_basis

引数 1

1	**const char ***	filename	入力リストが記述されたファイル名

入力ファイルから、基底関数の座標を読む関数（展開係数や指数といった基底関数の定義は別の関数で読まれる）。以下のグローバル変数に領域が割り当てられ、値がセットされる。

```
int      basis_n
int      basis_spher
char *   basis_name     , [ basis_n * 21 ]
double * basis_x        , [ basis_n ]
double * basis_y        , [ basis_n ]
double * basis_z        , [ basis_n ]
int *    basis_type     , [ basis_n ]       , 値は basis_define_assign でセット
char *   basis_name_ini , [ basis_n * 21 ]  , 値は basis_ini_assign    でセット
int *    basis_type_ini , [ basis_n ]       , 値は basis_define_assign でセット
int *    basis_charge   , [ basis_n ]       , 値は basis_charge_assign でセット
```

6.10 void read_nucleus

引数 1

1	**const char ***	filename	入力リストが記述されたファイル名

入力ファイルから原子核の座標を読む関数。

6.11 void read_fragment

引数 1

1	**const char ***	filename	入力リストが記述されたファイル名

入力ファイルからフラグメント分割の情報を読む関数。以下のグローバル変数に領域が割り当てられ、値がセットされる。

```
int       frag_n
int *     frag_nelec    , [ frag_n ]
int *     frag_nch1     , [ frag_n ]
int *     frag_nch1_in  , [ frag_n ]
int *     frag_nch1_idx , [ frag_n ]
int *     frag_charge
double *  frag_charge_u
int *     frag_nba1     , [ frag_n ]
int *     frag_nba1_in  , [ frag_n ]
int *     frag_nba1_idx , [ frag_n ]
int *     frag_basis
```

6.12 void read_ext_point_charge

引数 1

1	**const char ***	filename	入力リストが記述されたファイル名

外部静電ポテンシャルを作る点電荷を読む関数。

6.13 void read_position

引数 1

1	**const char ***	filename	入力リストが記述されたファイル名

位置の指定を読む関数。読まれた位置は、静電ポテンシャルや電場の計算の際に使用される。

6.14 void read_basis_define

引数 1

1	**const char ***	filename	入力リストが記述されたファイル名

計算で使用される基底関数の係数や指数などを、定義ファイルおよび入力ファイルから読みプログラムにロードする。以下のグローバル変数に領域が割り当てられ値がセットされる。

```
int       bdef_n
char *    bdef_name     , [ bdef_max ]
int *     bdef_nsh      , [ bdef_max ]
int *     bdef_nbo      , [ bdef_max ]
int *     bdef_sh_nc    , [ bdef_max * bdef_max_sh ]
int *     bdef_sh_l     , [ bdef_max * bdef_max_sh ]
double *  bdef_sh_expon , [ bdef_max * bdef_max_sh * bdef_max_nc ]
double *  bdef_sh_coeff , [ bdef_max * bdef_max_sh * bdef_max_nc ]
```

6.15 void read_basis_define_file

引数 1

1	**const char ***	filename	基底関数の定義が記述されたファイル名

計算で使用される基底関数の係数や指数などを定義ファイルから読む。

6.16 void read_monomer_scc_result

1	**const double ***	energy	出力バッファー
2	**const double ***	energy_d	出力バッファー

モノマー SCC の結果をファイルから読む。結果的にモノマー SCC の過程が省略される。

7 出力

7.1 void print_input_information

引数 無し

入力された情報を出力する関数。

7.2 void print_input_information_rhf

引数 無し

RHF 計算に関連する入力情報を出力する関数。

7.3 void print_input_information_cmp2

引数 無し

---

canonical MP2 計算に関連する入力情報を出力する関数。

7.4 void print_input_information_ri_cmp2

引数 無し

RI{MP2 計算に関連する入力情報を出力する関数。

7.5 void print_input_information_lmp2

引数 無し

局在化 MP2 計算に関連する入力情報を出力する関数。

7.6 void print_nucleus_coordinate

引数 無し

読み込んだ原子核の座標を出力する関数。

7.7 void print_basis_set_coordinate

引数 無し

読み込んだ基底関数の座標を出力する関数。

7.8 void print_basis_set_definition

引数 無し

読み込んだ基底関数の定義を出力する関数。

7.9 void print_ext_point_charge_coordinate

引数 無し

読み込んだ外部点電荷の座標を表示する関数。

7.10 void print_fragment_definition

引数 無し

読み込んだフラグメントの定義を出力する関数。

7.11 void print_memory_information

引数 無し

計算を実行する前の、メモリの使用状況などを出力する関数。

7.12 void print_frag_information

引数 無し

入力されたフラグメントの定義に従って作られた、各フラグメントの情報を出力する関数。

7.13 void print_frag_pair_information

引数 無し

入力されたフラグメントの定義に従って作られた、フラグメントペアの情報を出力する関数。

7.14 void print_monomer_scc_result

引数 無し

モノマー SCC 計算が終了した段階での結果を出力する関数。

7.15 void print_monomer_calculation_result

引数 無し

モノマー計算が終了した段階での結果を出力する関数。

7.16 void print_dimer_es_result

引数 無し

ダイマー ES 近似計算が終了した段階での結果を出力する関数。

7.17 void print_dimer_calculation_result

引数 無し

ダイマー計算が終了した段階での結果を出力する関数。

7.18 void print_dimer_ifie_result

引数 無し

IFIE を出力する関数。

7.19 void print_molecular_orbital

引数 8

1	**const int**	nmo		プリントする分子軌道の数
2	**const int**	nbo		ao の数
3	**const int**	nsh		シェルの数
4	**const int**	nw		一度に表示する軌道の数
5	**const int**	lprint		プリントフラグ
6	**const int ***	shell	[nsh * 3]	シェルの情報
7	**const double ***	moe	[nmo]	軌道エネルギー
8	**const double ***	c	[nmo * nbo]	軌道の係数

分子軌道の係数を出力する関数。

7.20 void print_monomer_mulliken_population

引数 2

1	**const int**	ifrag	フラグメントの番号
2	**const int**	lprint	出力フラグ

フラグメントごとの Mulliken population を出力する関数（基本的にデバッグ用）。

7.21 void write_monomer_scc_result

引数 無し

モノマー SCC 計算の結果、得られた電子密度などをファイルに書き出す関数。

7.22 void write_grid_rhf_monomer_orbital

引数 無し

モノマーの分子軌道のグリッドデータを書き出す関数。

7.23 void write_grid_orbital

引数 2

1	**const int**	out_type
2	**const int**	ifrag
3	**const int**	iorb
4	**const int**	nba1
5	**const int**	nsh1
6	**const int**	nbo1
7	**const int**	xdiv
8	**const int**	ydiv
9	**const int**	zdiv
10	**const double**	xmag
11	**const double**	ymag
12	**const double**	zmag
13	**const double**	pop_th

14	**const double**	coeff_th
15	**const int ***	basis
16	**const int ***	shell
17	**const double ***	shell_xyz
18	**const double ***	ovl
19	**const double ***	coeff
20	**const char ***	filename
21	**const char ***	pop_order
22	**const char ***	pop_val
23	**const char ***	work1
24	**const char ***	work2

分子軌道のグリッドデータを書き出す関数。

8 誤差関数

8.1 void initialize_fmt

引数 無し

誤差関数を計算するための初期設定を行う。以下の配列に値をセットする。

```
double * fmt_table
```

8.2 void fmt

引数 3

1	**double ***	val	[1]	結果の出力
2	**const int**	m		計算したい誤差関数の m の値
3	**const double**	t		計算したい誤差関数の t の値

「m」と「t」で指定される誤差関数の値を計算する関数。

9 一電子積分

9.1 void ovl_recursive

引数 12

1	**const int**	out_order	出力される数値の並び方の指定
2	**const int**	la	a の角運動量
3	**const int**	lb	b の角運動量
4	**const int**	nca	a の短縮数
5	**const int**	ncb	b の短縮数
6	**const double ***	xyza	a の座標
7	**const double ***	xyzb	b の座標
8	**const double ***	expa	a の指数
9	**const double ***	expb	b の指数
10	**const double ***	coeffa	a の展開係数
11	**const double ***	coeffb	b の展開係数
12	**double ***	output	出力バッファー

重なり積分を小原漸化式から計算するための関数。すべての角運動量の組み合わせをこの関数で計算し、2 電子積分のように角運動量の組み合わせごとの「特化ルーチン」は無い。引数は、必ず「la　lb」となるように渡されなければならない。結果は出力バッファーにセットされるが、並び方は order の値によって異なる。a の ia 番目の関数と b の ib 番目の関数との重なり積分は、以下のように参照される。

```
order = 1 の時 : output [ ang_l6n[lb] * ia + ib ]
order = 2 の時 : output [ ang_l6n[la] * ib + ia ]
```

9.2 void kei_recursive

引数 12

1	**const int**	out_order	
2	**const int**	la	
3	**const int**	lb	
4	**const int**	nca	
5	**const int**	ncb	
6	**const double ***	xyza	
7	**const double ***	xyzb	
8	**const double ***	expa	
9	**const double ***	expb	
10	**const double ***	coeffa	
11	**const double ***	coeffb	
12	**double ***	output	出力バッファー

運動エネルギー積分を小原漸化式から計算するための関数。すべての角運動量の組み合わせをこの関数で計算し、2 電子積分のように角運動量の組み合わせごとの「特化ルーチン」は無い。引数は、必ず「la　lb」となるように渡されなければならない。結果は出力バッファーにセットされるが、並び方は order の値によって異なる。a の ia 番目の関数と b の ib 番目の関数との運動エネルギー積分は、以下のように参照される。

```
order = 1 の時 : output [ ang_l6n[lb] * ia + ib ]
order = 2 の時 : output [ ang_l6n[la] * ib + ia ]
```

9.3 void nai_recursive

引数 15

1	**const int**	out_order	
2	**const int**	la	
3	**const int**	lb	
4	**const int**	nca	
5	**const int**	ncb	
6	**const int**	ncharge	
7	**const double ***	xyza	
8	**const double ***	xyzb	
9	**const double ***	expa	
10	**const double ***	expb	
11	**const double ***	coeffa	
12	**const double ***	coeffb	
13	**const double ***	charge_xyz	
14	**const double ***	charge_val	
15	**double ***	output	出力バッファー

核引力積分を小原漸化式から計算するための関数。すべての角運動量の組み合わせをこの関数で計算し、2 電子積分のように角運動量の組み合わせごとの「特化ルーチン」は無い。

9.4 void efi_recursive

引数 15

1	**const int**	out_order	
2	**const int**	la	
3	**const int**	lb	
4	**const int**	nca	
5	**const int**	ncb	
6	**const int**	ncharge	
7	**const double ***	xyza	
8	**const double ***	xyzb	

9	**const double ***	expa	
10	**const double ***	expb	
11	**const double ***	coeffa	
12	**const double ***	coeffb	
13	**const double ***	charge_xyz	
14	**const double ***	charge_val	
15	**double ***	output	出力バッファー

電場積分を小原漸化式から計算するための関数。すべての角運動量の組み合わせをこの関数で計算し、2 電子積分のように角運動量の組み合わせごとの「特化ルーチン」は無い。

9.5 void mom_recursive

引数 12

1	**const int**	out_order	
2	**const int**	la	
3	**const int**	lb	
4	**const int**	nca	
5	**const int**	ncb	
6	**const double ***	xyza	
7	**const double ***	xyzb	
8	**const double ***	expa	
9	**const double ***	expb	
10	**const double ***	coeffa	
11	**const double ***	coeffb	
12	**double ***	output	出力バッファー

モーメント積分を小原漸化式から計算するための関数。すべての角運動量の組み合わせをこの関数で計算し、2 電子積分のように角運動量の組み合わせごとの「特化ルーチン」は無い。

9.6 void get_ovl

引数 12

1	**const int**	spher	球面調和型かカルテシアン型かの指定
2	**const int**	la	a の角運動量
3	**const int**	lb	b の角運動量
4	**const int**	nca	a の短縮数
5	**const int**	ncb	b の短縮数
6	**const double ***	xyza	a の座標
7	**const double ***	xyzb	b の座標
8	**const double ***	expa	a の原始ガウス関数の指数
9	**const double ***	expb	b の原始ガウス関数の指数

10	**const double ***	coeffa	a の短縮係数
11	**const double ***	coeffb	b の短縮係数
12	**double ***	output	出力バッファー

重なり積分を計算する関数。この関数の中で ovl_recursive 関数および transformation_spherical_2 関数が呼ばれる。また、2 回の対称性に関する「並べ替え」が行われる。出力された積分は以下のように参照される。

```
spher = 0 の時 : output [ ang_l6n[lb] * ia + ib ]
spher = 1 の時 : output [ ang_l5n[la] * ia + ib ]
```

9.7 void get_kei

引数 12

1	**const int**	spher	
2	**const int**	la	
3	**const int**	lb	
4	**const int**	nca	
5	**const int**	ncb	
6	**const double ***	xyza	
7	**const double ***	xyzb	
8	**const double ***	expa	
9	**const double ***	expb	
10	**const double ***	coeffa	
11	**const double ***	coeffb	
12	**double ***	output	出力バッファー

運動エネルギー積分を計算する関数。この関数の中で kei_recursive 関数および transformation_spherical_2 関数が呼ばれる。また、2 回の対称性に関する「並べ替え」が行われる。出力された積分は以下のように参照される。

```
spher = 0 の時 : output [ ang_l6n[lb] * ia + ib ]
spher = 1 の時 : output [ ang_l5n[la] * ia + ib ]
```

9.8 void get_nai

引数 15

1	**const int**	spher
2	**const int**	la
3	**const int**	lb
4	**const int**	nca

5	**const int**	ncb	
6	**const int**	ncharge	
7	**const double ***	xyza	
8	**const double ***	xyzb	
9	**const double ***	expa	
10	**const double ***	expb	
11	**const double ***	coeffa	
12	**const double ***	coeffb	
13	**const double ***	charge_xyz	
14	**const double ***	charge_val	
15	**double ***	output	出力バッファー

核引力積分を計算する関数。この関数の中で nai_recursive 関数および transformation_spherical_2 関数が呼ばれる。また、2 回の対称性に関する「並べ替え」が行われる。出力された積分は以下のように参照される。

```
spher = 0 の時 : output [ ang_l6n[lb] * ia + ib ]
spher = 1 の時 : output [ ang_l5n[la] * ia + ib ]
```

9.9 void get_efi

引数 15

1	**const int**	spher	
2	**const int**	la	
3	**const int**	lb	
4	**const int**	nca	
5	**const int**	ncb	
6	**const int**	ncharge	
7	**const double ***	xyza	
8	**const double ***	xyzb	
9	**const double ***	expa	
10	**const double ***	expb	
11	**const double ***	coeffa	
12	**const double ***	coeffb	
13	**const int ***	charge_i	
14	**const double ***	charge_xyz	
15	**const double ***	charge_val	
16	**double ***	output	出力バッファー

電場積分を計算する関数。この関数の中で efi_recursive 関数および transformation_spherical_2 関数が呼ばれる。また、2 回の対称性に関する「 並べ替え 」が行われる。出力された積分は以下のように参照される。

```
spher = 0 の時 : output [ ang_l6n[lb] * ia + ib ]
spher = 1 の時 : output [ ang_l5n[la] * ia + ib ]
```

9.10 void get_mom

引数 12

1	**const int**	spher	
2	**const int**	la	
3	**const int**	lb	
4	**const int**	nca	
5	**const int**	ncb	
6	**const double ***	xyza	
7	**const double ***	xyzb	
8	**const double ***	expa	
9	**const double ***	expb	
10	**const double ***	coeffa	
11	**const double ***	coeffb	
12	**double ***	output	出力バッファー

モーメント積分を計算する関数。この関数の中で mom_recursive 関数および transformation_spherical_2_mom 関数が呼ばれる。また、2 回の対称性に関する「並べ替え」が行われる。

9.11 void make_ovl_matrix

引数 5

1	**const int**	ns	
2	**const int**	nbo	
3	**const int ***	shell	
4	**const double ***	shell_xyz	
5	**double ***	out	出力バッファー

重なり積分行列を作成する関数。

9.12 void make_ovl_matrix_inter

引数 9

1	**const int**	ns1
2	**const int**	ns2
3	**const int**	nbo1
4	**const int**	nbo2
5	**const int ***	shell_1
6	**const int ***	shell_2
7	**const double ***	shell_xyz_1
8	**const double ***	shell_xyz_2

9	**double ***	out	出力バッファー

異なる基底関数セット間の重なり積分行列を作成する関数。

9.13 void make_kei_matrix

引数 5

1	**const int**	ns	
2	**const int**	nbo	
3	**const int ***	shell	
4	**const double ***	shell_xyz	
5	**double ***	out	出力バッファー

運動エネルギー積分行列を作成する関数。

9.14 void make_nai_matrix

引数 9

1	**const int**	ns	
2	**const int**	ncharge	
3	**const int**	nbo	
4	**MPI_Comm**	icomm	
5	**const int ***	shell	
6	**const double ***	shell_xyz	
7	**const double ***	charge_xyz	
8	**const double ***	charge_val	
9	**double ***	out	出力バッファー

核引力積分行列を作成する関数。

9.15 void make_mom_matrix

引数 5

1	**const int**	ns	
2	**const int**	nbo	
3	**const int ***	shell	
4	**const double ***	shell_xyz	
5	**double ***	out	出力バッファー

モーメント積分行列を作成する関数。各座標成分の ibo、jbo 要素は以下のように参照される。

```
x 成分 : out [ 3 * nbo * ibo + 3 * jbo + 0 ]
y 成分 : out [ 3 * nbo * ibo + 3 * jbo + 1 ]
z 成分 : out [ 3 * nbo * ibo + 3 * jbo + 2 ]
```

10　一電子積分の一次微分

10.1　void ovl_grad_recursive

引数 12

1	**const int**	out_order	
2	**const int**	la	
3	**const int**	lb	
4	**const int**	nca	
5	**const int**	ncb	
6	**const double ***	xyza	
7	**const double ***	xyzb	
8	**const double ***	expa	
9	**const double ***	expb	
10	**const double ***	coeffa	
11	**const double ***	coeffb	
12	**double ***	output	出力バッファー

重なり積分の 1 次微分を小原漸化式から計算する関数。全ての角運動量の組み合わせを計算することが可能。2 電子積分のように角運動量の組み合わせにごとの「特化ルーチン」は無い。

10.2　void kei_grad_recursive

引数 12

1	**const int**	out_order	
2	**const int**	la	
3	**const int**	lb	
4	**const int**	nca	
5	**const int**	ncb	
6	**const double ***	xyza	
7	**const double ***	xyzb	
8	**const double ***	expa	
9	**const double ***	expb	
10	**const double ***	coeffa	
11	**const double ***	coeffb	
12	**double ***	output	出力バッファー

運動エネルギー積分の 1 次微分を小原漸化式から計算する関数。全ての角運動量の組み合わせを計算することが可能。2 電子積分のように角運動量の組み合わせごとの「特化ルーチン」は無い。

10.3 void nai_grad_recursive

引数 12

1	**const int**	out_order	
2	**const int**	la	
3	**const int**	lb	
4	**const int**	nca	
5	**const int**	ncb	
6	**const int**	ncharge	
7	**const double ***	xyza	
8	**const double ***	xyzb	
9	**const double ***	expa	
10	**const double ***	expb	
11	**const double ***	coeffa	
12	**const double ***	coeffb	
13	**const double ***	charge_xyz	
14	**const double ***	charge_val	
15	**double ***	output	出力バッファー

核引力積分の 1 次微分を小原漸化式から計算するための関数。核引力積分の 1 次微分からは、電場積分が出てくるが、これは efi_recursive 関数で別に計算される。全ての角運動量の組み合わせを計算することが可能。2 電子積分のように角運動量の組み合わせに対応した「特化ルーチン」は無い。

10.4 void get_ovl_grad

引数 12

1	**const int**	spher	球面調和型かカルテシアン型かの指定
2	**const int**	la	a の角運動量
3	**const int**	lb	b の角運動量
4	**const int**	nca	a の短縮数
5	**const int**	ncb	b の短縮数
6	**const double ***	xyza	a の座標
7	**const double ***	xyzb	b の座標
8	**const double ***	expa	a の原始ガウス関数の指数
9	**const double ***	expb	b の原始ガウス関数の指数
10	**const double ***	coeffa	a の短縮係数
11	**const double ***	coeffb	b の短縮係数
12	**double ***	output	出力バッファー

重なり積分の 1 次微分を計算する関数。また、2 回の対称性に関する「並べ替え」も行われる。この関数の中で ovl_grad_recursive 関数および transformation_spherical_2 関数が呼ばれる。出力された積分は以下のように参照される。

```
p1 = ang_l6n[la] * ang_l6n[lb] ;
p2 = ang_l5n[la] * ang_l5n[lb] ;
spher = 0 の時 : output [ p1 * ixyz + ang_l6n[la] * ia + ib ]
spher = 1 の時 : output [ p2 * ixyz + ang_l5n[la] * ia + ib ]
```

10.5 void get_kei_grad

引数 12

1	**const int**	spher	球面調和型かカルテシアン型かの指定
2	**const int**	la	a の角運動量
3	**const int**	lb	b の角運動量
4	**const int**	nca	a の短縮数
5	**const int**	ncb	b の短縮数
6	**const double ***	xyza	a の座標
7	**const double ***	xyzb	b の座標
8	**const double ***	expa	a の原始ガウス関数の指数
9	**const double ***	expb	b の原始ガウス関数の指数
10	**const double ***	coeffa	a の短縮係数
11	**const double ***	coeffb	b の短縮係数
12	**double ***	output	出力バッファー

運動エネルギー積分の 1 次微分を計算する関数。また、2 回の対称性に関する「並べ替え」も行われる。この関数の中で kei_grad_recursive 関数および transformation_spherical_2 関数が呼ばれる。出力された積分は以下のように参照される。

```
p1 = ang_l6n[la] * ang_l6n[lb] ;
p2 = ang_l5n[la] * ang_l5n[lb] ;
spher = 0 の時 : output [ p1 * ixyz + ang_l6n[la] * ia + ib ]
spher = 1 の時 : output [ p2 * ixyz + ang_l5n[la] * ia + ib ]
```

10.6 void get_nai_grad

引数 15

1	**const int**	spher
2	**const int**	la
3	**const int**	lb
4	**const int**	nca
5	**const int**	ncb
6	**const int**	ncharge
7	**const double ***	xyza
8	**const double ***	xyzb
9	**const double ***	expa
10	**const double ***	expb

11	**const double ***	coeffa	
12	**const double ***	coeffb	
13	**const double ***	charge_xyz	
14	**const double ***	charge_val	
15	**double ***	output	出力バッファー

核引力積分の 1 次微分を計算する関数。また、2 回の対称性に関する「並べ替え」が行われる。この関数の中で nai_grad_recursive 関数および transformation_spherical_2 関数が呼ばれる。核引力積分の 1 次微分からは電場積分が出てくるが、これは get_efi 関数で別に計算される。出力された積分は以下のように参照される。

```
p1 = ang_l6n[la] * ang_l6n[lb] ;
p2 = ang_l5n[la] * ang_l5n[lb] ;
spher = 0 の時 : output [ p1 * ixyz + ang_l6n[la] * ia + ib ]
spher = 1 の時 : output [ p2 * ixyz + ang_l5n[la] * ia + ib ]
```

11 電子反発積分

11.1 void get_eri

引数 31

1	**const int**	spher	5 成分か 6 成分か
2	**const double**	tv	スクリーニングの閾値
3	**const int**	la	角運動量の値
4	**const int**	lb	角運動量の値
5	**const int**	lc	角運動量の値
6	**const int**	ld	角運動量の値
7	**const int**	noa	
8	**const int**	nob	
9	**const int**	noc	
10	**const int**	nod	
11	**const int**	nca	
12	**const int**	ncb	
13	**const int**	ncc	
14	**const int**	ncd	
15	**const double ***	xyza	基底関数の座標
16	**const double ***	xyzb	基底関数の座標
17	**const double ***	xyzc	基底関数の座標
18	**const double ***	xyzd	基底関数の座標
19	**const double ***	expa	基底関数の指数
20	**const double ***	expb	基底関数の指数
21	**const double ***	expc	基底関数の指数
22	**const double ***	expd	基底関数の指数
23	**const double ***	kab	
24	**const double ***	p_x	
25	**const double ***	p_y	
26	**const double ***	p_z	
27	**const double ***	kcd	
28	**const double ***	q_x	
29	**const double ***	q_y	
30	**const double ***	q_z	
31	**double ***	output	計算結果の出力

指定された 4 つのシェルに対応する電子反発積分を計算する関数。この関数の中で eri_recursive 関数と transformation_spherical_4 関数が呼ばれる。また、8 回対称性による「並べ替え」も行われる。

11.2 void get_eri_2cen

引数 14

1	**const int**	spher	5 成分か 6 成分か

2	**const int**	la	角運動量の値
3	**const int**	lc	角運動量の値
4	**const int**	noa	
5	**const int**	noc	
6	**const int**	nca	
7	**const int**	ncc	
8	**const double ***	xyza	基底関数の座標
9	**const double ***	xyzc	基底関数の座標
10	**const double ***	expa	基底関数の指数
11	**const double ***	expc	基底関数の指数
12	**const double ***	coeffa	
13	**const double ***	coeffc	
14	**double ***	output	計算結果の出力

指定された 2 つのシェルに対応する電子反発積分を計算するための関数。

11.3 void get_eri_3cen

引数 23

1	**const int**	spher	5 成分か 6 成分か
2	**const double**	tv	スクリーニングの閾値
3	**const int**	la	角運動量の値
4	**const int**	lb	角運動量の値
5	**const int**	lc	角運動量の値
6	**const int**	noa	
7	**const int**	nob	
8	**const int**	noc	
9	**const int**	nca	
10	**const int**	ncb	
11	**const int**	ncc	
12	**const double ***	xyza	基底関数の座標
13	**const double ***	xyzb	基底関数の座標
14	**const double ***	xyzc	基底関数の座標
15	**const double ***	expa	基底関数の指数
16	**const double ***	expb	基底関数の指数
17	**const double ***	expc	基底関数の指数
18	**const double ***	coeffc	
19	**const double ***	kab	
20	**const double ***	p_x	
21	**const double ***	p_y	
22	**const double ***	p_z	
23	**double ***	output	計算結果の出力

指定された 3 つのシェルに対応する電子反発積分を計算するための関数。

11.4 void eri_recursive_gen

引数 31

1	**const int**	order	
2	**const double**	tv	
3	**const int**	la	
4	**const int**	lb	
5	**const int**	lc	
6	**const int**	ld	
7	**const int**	noa	
8	**const int**	nob	
9	**const int**	noc	
10	**const int**	nod	
11	**const int**	nca	
12	**const int**	ncb	
13	**const int**	ncc	
14	**const int**	ncd	
15	**const double ***	xyza	
16	**const double ***	xyzb	
17	**const double ***	xyzc	
18	**const double ***	xyzd	
19	**const double ***	expa	
20	**const double ***	expb	
21	**const double ***	expc	
22	**const double ***	expd	
23	**const double ***	kab	
24	**const double ***	p_x	
25	**const double ***	p_y	
26	**const double ***	p_z	
27	**const double ***	kcd	
28	**const double ***	q_x	
29	**const double ***	q_y	
30	**const double ***	q_z	
31	**double ***	output	出力バッファー

任意の角運動量の組み合わせの電子反発積分が計算できる関数（一般ルーチン）。どんな角運動量の組み合わせでも計算出来るが、不要な中間積分や条件分岐などがあるため「特化ルーチン」よりも時間がかかる。

11.5 void eri_recursive_2cen_gen

引数 14

1	**const int**	order	
2	**const int**	la	
3	**const int**	lc	

4	**const int**	noa	
5	**const int**	noc	
6	**const int**	nca	
7	**const int**	ncc	
8	**const double ***	xyza	
9	**const double ***	xyzc	
10	**const double ***	expa	
11	**const double ***	expc	
12	**const double ***	coeffa	
13	**const double ***	coeffc	
14	**double ***	out	出力バッファー

任意の角運動量の組み合わせの 2 つのシェルに関する電子反発積分が計算できる関数。誤差関数計算の制限から (g g j g g) までしか計算出来ない

11.6 void eri_recursive_3cen_gen

引数 23

1	**const int**	order	
2	**const double**	tv	
3	**const int**	la	
4	**const int**	lb	
5	**const int**	lc	
6	**const int**	noa	
7	**const int**	nob	
8	**const int**	noc	
9	**const int**	nca	
10	**const int**	ncb	
11	**const int**	ncc	
12	**const double ***	xyza	
13	**const double ***	xyzb	
14	**const double ***	xyzc	
15	**const double ***	expa	
16	**const double ***	expb	
17	**const double ***	expc	
18	**const double ***	kab	
19	**const double ***	p_x	
20	**const double ***	p_y	
21	**const double ***	p_z	
22	**const double ***	coeff	
23	**double ***	output	出力バッファー

任意の角運動量の組み合わせの 3 つのシェルに関する電子反発積分が計算できる関数（一般ルーチン)。どんな角運動量の組み合わせでも計算出来るが、不要な中間積分も計算するので「特化ルーチン」よりも時間がかかる。

11.7 void eri_recursive

引数 31

1	**const int**	out_order	出力バッファーに格納する「 並び 」の指定
2	**const double**	tv	スクリーニングの閾値
3	**const int**	la	角運動量の値
4	**const int**	lb	角運動量の値
5	**const int**	lc	角運動量の値
6	**const int**	ld	角運動量の値
7	**const int**	noa	
8	**const int**	nob	
9	**const int**	noc	
10	**const int**	nod	
11	**const int**	nca	
12	**const int**	ncb	
13	**const int**	ncc	
14	**const int**	ncd	
15	**const double ***	xyza	基底関数の座標
16	**const double ***	xyzb	基底関数の座標
17	**const double ***	xyzc	基底関数の座標
18	**const double ***	xyzd	基底関数の座標
19	**const double ***	expa	基底関数の指数
20	**const double ***	expb	基底関数の指数
21	**const double ***	expc	基底関数の指数
22	**const double ***	expd	基底関数の指数
23	**const double ***	kab	
24	**const double ***	p_x	
25	**const double ***	p_y	
26	**const double ***	p_z	
27	**const double ***	kcd	
28	**const double ***	q_x	
29	**const double ***	q_y	
30	**const double ***	q_z	
31	**double ***	output	出力バッファー

電子反発積分を計算する関数。この下で、基底関数の角運動量の組み合わせに応じた「特化ルーチン」もしくは「一般ルーチン」が呼ばれる。

11.8 void eri_recursive _ [la+lb] [lc+ld] _ [la] [lc]

引数 27

1	**const int**	order	出力バッファーに格納する並びの指定
2	**const double**	tv	スクリーニングの閾値
3	**const int**	noa	
4	**const int**	nob	

5	**const int**	noc	
6	**const int**	nod	
7	**const int**	nca	
8	**const int**	ncb	
9	**const int**	ncc	
10	**const int**	ncd	
11	**const double ***	xyza	基底関数の座標
12	**const double ***	xyzb	基底関数の座標
13	**const double ***	xyzc	基底関数の座標
14	**const double ***	xyzd	基底関数の座標
15	**const double ***	expa	基底関数の指数
16	**const double ***	expb	基底関数の指数
17	**const double ***	expc	基底関数の指数
18	**const double ***	expd	基底関数の指数
19	**const double ***	kab	
20	**const double ***	p_x	
21	**const double ***	p_y	
22	**const double ***	p_z	
23	**const double ***	kcd	
24	**const double ***	q_x	
25	**const double ***	q_y	
26	**const double ***	q_z	
27	**double ***	output	出力バッファー

決まった角運動量の組み合わせの 4 中心電子反発積分を計算する関数（特化ルーチン）。不要な中間積分も計算する「eri_recursive_gen 関数（一般ルーチン）」よりも、2〜3 倍程度速い。各特化ルーチンは、積分の 8 回対称性で同じ値になるものをすべて担当する。以下の 92 個の特化ルーチンが実装されている。これは f 軌道までのすべての組み合わせを含んでいる。

1	**eri_recursive_00_00**	(	s s	s s	)
2	**eri_recursive_10_10**	(	p s	s s	)
3	**eri_recursive_11_11**	(	p s	p s	)
4	**eri_recursive_20_20**	(	d s	s s	)
5	**eri_recursive_20_10**	(	p p	s s	)
6	**eri_recursive_21_21**	(	d s	p s	)
7	**eri_recursive_21_11**	(	p p	p s	)
8	**eri_recursive_22_22**	(	d s	d s	)
9	**eri_recursive_22_12**	(	p p	d s	)
10	**eri_recursive_22_11**	(	p p	p p	)
11	**eri_recursive_30_30**	(	f s	s s	)
12	**eri_recursive_30_20**	(	d p	s s	)
13	**eri_recursive_31_31**	(	f s	p s	)
14	**eri_recursive_31_21**	(	d p	p s	)
15	**eri_recursive_32_32**	(	f s	d s	)
16	**eri_recursive_32_22**	(	d p	d s	)
17	**eri_recursive_32_31**	(	f s	p p	)
18	**eri_recursive_32_21**	(	d p	p p	)

19	**eri_recursive_33_33**	(	f s	|	f s	)
20	**eri_recursive_33_23**	(	d p	|	f s	)
21	**eri_recursive_33_32**	(	f s	|	d p	)
22	**eri_recursive_33_22**	(	d p	|	d p	)
23	**eri_recursive_40_40**	(	g s	|	s s	)
24	**eri_recursive_40_30**	(	f p	|	s s	)
25	**eri_recursive_40_20**	(	d d	|	s s	)
26	**eri_recursive_41_41**	(	g s	|	p s	)
27	**eri_recursive_41_31**	(	f p	|	p s	)
28	**eri_recursive_41_21**	(	d d	|	p s	)
29	**eri_recursive_42_42**	(	g s	|	d s	)
30	**eri_recursive_42_32**	(	f p	|	d s	)
31	**eri_recursive_42_22**	(	d d	|	d s	)
32	**eri_recursive_42_41**	(	g s	|	p p	)
33	**eri_recursive_42_31**	(	f p	|	p p	)
34	**eri_recursive_42_21**	(	d d	|	p p	)
34	**eri_recursive_43_43**	(	g s	|	f s	)
35	**eri_recursive_43_33**	(	f p	|	f s	)
36	**eri_recursive_43_23**	(	d d	|	f s	)
37	**eri_recursive_43_42**	(	g s	|	d p	)
38	**eri_recursive_43_32**	(	f p	|	d p	)
39	**eri_recursive_43_22**	(	d d	|	d p	)
40	**eri_recursive_44_44**	(	g s	|	g s	)
41	**eri_recursive_44_34**	(	f p	|	g s	)
42	**eri_recursive_44_24**	(	d d	|	g s	)
43	**eri_recursive_44_33**	(	g s	|	f p	)
44	**eri_recursive_44_23**	(	f p	|	f p	)
45	**eri_recursive_44_22**	(	d d	|	d d	)
46	**eri_recursive_50_40**	(	g p	|	s s	)
47	**eri_recursive_50_30**	(	f d	|	s s	)
48	**eri_recursive_51_41**	(	g p	|	p s	)
49	**eri_recursive_51_31**	(	f d	|	p s	)
50	**eri_recursive_52_42**	(	g p	|	d s	)
51	**eri_recursive_52_32**	(	f d	|	d s	)
52	**eri_recursive_52_41**	(	g p	|	p p	)
53	**eri_recursive_52_31**	(	f d	|	p p	)
54	**eri_recursive_53_43**	(	g p	|	f s	)
55	**eri_recursive_53_33**	(	f d	|	f s	)
56	**eri_recursive_53_42**	(	g p	|	d p	)
57	**eri_recursive_53_32**	(	f d	|	d p	)
58	**eri_recursive_54_44**	(	g p	|	g s	)
59	**eri_recursive_54_34**	(	f d	|	g s	)
60	**eri_recursive_54_43**	(	g p	|	f p	)

61	**eri_recursive_54_33**	(	f d	|	f p	)
62	**eri_recursive_54_42**	(	g p	|	d d	)
63	**eri_recursive_54_32**	(	f d	|	d d	)
64	**eri_recursive_55_44**	(	g p	|	g p	)
65	**eri_recursive_55_34**	(	f d	|	g p	)
66	**eri_recursive_55_33**	(	f d	|	f d	)
67	**eri_recursive_60_40**	(	g d	|	s s	)
68	**eri_recursive_60_30**	(	f f	|	s s	)
69	**eri_recursive_61_41**	(	g d	|	p s	)
70	**eri_recursive_61_31**	(	f f	|	p s	)
71	**eri_recursive_62_42**	(	g d	|	d s	)
72	**eri_recursive_62_32**	(	f f	|	d s	)
73	**eri_recursive_62_41**	(	g d	|	p p	)
74	**eri_recursive_62_31**	(	f f	|	p p	)
75	**eri_recursive_63_43**	(	g d	|	f s	)
76	**eri_recursive_63_33**	(	f f	|	f s	)
77	**eri_recursive_63_42**	(	g d	|	d p	)
78	**eri_recursive_63_32**	(	f f	|	d p	)
80	**eri_recursive_64_44**	(	g d	|	g s	)
81	**eri_recursive_64_34**	(	f f	|	g s	)
82	**eri_recursive_64_43**	(	g d	|	f p	)
83	**eri_recursive_64_33**	(	f f	|	f p	)
84	**eri_recursive_64_42**	(	g d	|	d d	)
85	**eri_recursive_64_32**	(	f f	|	d d	)
86	**eri_recursive_65_44**	(	g d	|	g p	)
87	**eri_recursive_65_34**	(	f f	|	g p	)
88	**eri_recursive_65_43**	(	g d	|	f d	)
89	**eri_recursive_65_33**	(	f f	|	f d	)
90	**eri_recursive_66_44**	(	g d	|	g d	)
91	**eri_recursive_66_34**	(	f f	|	g d	)
92	**eri_recursive_66_33**	(	f f	|	f f	)

11.9 void eri_recursive_3cen

引数 31

1	**const int**	out_order	出力バッファーに格納する「 並び 」の指定
2	**const double**	tv	スクリーニングの閾値
3	**const int**	la	角運動量の値
4	**const int**	lb	角運動量の値
5	**const int**	lc	角運動量の値
6	**const int**	noa	
7	**const int**	nob	

8	**const int**	noc	
9	**const int**	nca	
10	**const int**	ncb	
11	**const int**	ncc	
12	**const double ***	xyza	基底関数の座標
13	**const double ***	xyzb	基底関数の座標
14	**const double ***	xyzc	基底関数の座標
15	**const double ***	expa	基底関数の指数
16	**const double ***	expb	基底関数の指数
17	**const double ***	expc	基底関数の指数
18	**const double ***	kab	
19	**const double ***	p_x	
20	**const double ***	p_y	
21	**const double ***	p_z	
22	**const double ***	coeff	
23	**double ***	output	出力バッファー

電子反発積分を計算する関数。この下で、基底関数の角運動量の組み合わせに応じた「特化ルーチン」もしくは「一般ルーチン」が呼ばれる。

11.10 void eri_recursive_3cen _ [la+lb] [lc] _ [la]

引数 20

1	**const int**	order	出力バッファーに格納する並びの指定
2	**const double**	tv	スクリーニングの閾値
3	**const int**	noa	
4	**const int**	nob	
5	**const int**	noc	
6	**const int**	nca	
7	**const int**	ncb	
8	**const int**	ncc	
9	**const double ***	xyza	基底関数の座標
10	**const double ***	xyzb	基底関数の座標
11	**const double ***	xyzc	基底関数の座標
12	**const double ***	expa	基底関数の指数
13	**const double ***	expb	基底関数の指数
14	**const double ***	expc	基底関数の指数
15	**const double ***	kab	
16	**const double ***	p_x	
17	**const double ***	p_y	
18	**const double ***	p_z	
19	**const double ***	coeff	
20	**double ***	output	出力バッファー

3 つのシェルから作られる決まった角運動量の組み合わせの電子反発積分を計算する関数（特化ルーチン）。不要な中間積分も計算する「eri_recursive_gen_3cen 関数（一般ルーチン）」よりも、2〜

3 倍程度速い。各特化ルーチンは、積分の 8 回対称性で同じ値になるものをすべて担当する。以下の 46 個の特化ルーチンが実装されている。

1	**eri_recursive_3cen_00_0**	(	s s	|	s s	)
2	**eri_recursive_3cen_01_0**	(	s s	|	p p	)
3	**eri_recursive_3cen_02_0**	(	s s	|	d d	)
4	**eri_recursive_3cen_03_0**	(	s s	|	f f	)
5	**eri_recursive_3cen_04_0**	(	s s	|	g g	)
6	**eri_recursive_3cen_10_1**	(	p s	|	s s	)
7	**eri_recursive_3cen_11_1**	(	p s	|	p p	)
8	**eri_recursive_3cen_12_1**	(	p s	|	d d	)
9	**eri_recursive_3cen_13_1**	(	p s	|	f f	)
10	**eri_recursive_3cen_14_1**	(	p s	|	g g	)
11	**eri_recursive_3cen_20_2**	(	d s	|	s s	)
12	**eri_recursive_3cen_20_1**	(	p p	|	s s	)
13	**eri_recursive_3cen_21_2**	(	d s	|	p p	)
14	**eri_recursive_3cen_21_1**	(	p p	|	p p	)
15	**eri_recursive_3cen_22_2**	(	d s	|	d d	)
16	**eri_recursive_3cen_22_1**	(	p p	|	d d	)
17	**eri_recursive_3cen_23_2**	(	d s	|	f f	)
18	**eri_recursive_3cen_23_1**	(	p p	|	f f	)
19	**eri_recursive_3cen_24_2**	(	d s	|	g g	)
20	**eri_recursive_3cen_24_1**	(	p p	|	g g	)
21	**eri_recursive_3cen_30_3**	(	f s	|	s s	)
22	**eri_recursive_3cen_30_2**	(	d p	|	s s	)
23	**eri_recursive_3cen_31_3**	(	f s	|	p p	)
24	**eri_recursive_3cen_31_2**	(	d p	|	p p	)
25	**eri_recursive_3cen_32_3**	(	f s	|	d d	)
26	**eri_recursive_3cen_32_2**	(	d p	|	d d	)
27	**eri_recursive_3cen_33_3**	(	f s	|	f f	)
28	**eri_recursive_3cen_33_2**	(	d p	|	f f	)
29	**eri_recursive_3cen_40_3**	(	f p	|	s s	)
30	**eri_recursive_3cen_40_2**	(	d d	|	s s	)
31	**eri_recursive_3cen_41_3**	(	f p	|	p p	)
32	**eri_recursive_3cen_41_2**	(	d d	|	p p	)
33	**eri_recursive_3cen_42_3**	(	f p	|	d d	)
34	**eri_recursive_3cen_42_2**	(	d d	|	d d	)
34	**eri_recursive_3cen_43_3**	(	f p	|	f f	)
35	**eri_recursive_3cen_43_2**	(	d d	|	f f	)
36	**eri_recursive_3cen_44_3**	(	f p	|	g g	)
37	**eri_recursive_3cen_44_2**	(	d d	|	g g	)
38	**eri_recursive_3cen_50_3**	(	f d	|	s s	)
39	**eri_recursive_3cen_51_3**	(	f d	|	p p	)

40	**eri_recursive_3cen_52_3**	(	f	d	|	d	d)
41	**eri_recursive_3cen_54_3**	(	f	d	|	f	f)
42	**eri_recursive_3cen_60_3**	(	f	f	|	s	s)
43	**eri_recursive_3cen_61_3**	(	f	f	|	p	p)
44	**eri_recursive_3cen_62_3**	(	f	f	|	d	d)
45	**eri_recursive_3cen_63_3**	(	f	f	|	f	f)
46	**eri_recursive_3cen_64_3**	(	f	f	|	g	g)

12　環境静電ポテンシャル

12.1　void make_esp_4_matrix

引数 14

1	**const int**	ns	
2	**const int**	nbo	
3	**const int**	n_frag_list	
4	**const MPI_Comm**	icomm	
5	**const int ***	shell	
6	**const double ***	shell_xyz	
7	**const int ***	kab_idx	
8	**const double ***	kab_sh	
9	**const double ***	kab	
10	**const double ***	p_x	
11	**const double ***	p_y	
12	**const double ***	p_z	
13	**const int ***	frag_list	
14	**double**	mat_esp_4	出力

4 中心積分で（近似を用いずに）環境静電ポテンシャルの行列表を計算する関数。

12.2　void make_esp_3_matrix

引数 14

1	**const int**	ns	
2	**const int**	nbo	
3	**const int**	n_frag_list	
4	**const MPI_Comm**	icomm	
5	**const int ***	shell	
6	**const double ***	shell_xyz	
7	**const int ***	kab_idx	
8	**const double ***	kab_sh	
9	**const double ***	kab	
10	**const double ***	p_x	
11	**const double ***	p_y	
12	**const double ***	p_z	
13	**const int ***	frag_list	
14	**double**	mat_esp_3	出力

3 中心積分の近似で環境静電ポテンシャルの行列を計算する関数。

12.3 void make_esp_m_matrix

引数 8

1	**const int**	ns	
2	**const int**	nbo	
3	**const int**	n_frag_list	
4	**const MPI_Comm**	icomm	
5	**const int ***	shell	
6	**const double ***	shell_xyz	
7	**const int ***	frag_list	
8	**double**	mat_esp_n	出力

点電荷近似で環境静電ポテンシャルの行列を計算する関数。

13 射影演算子

13.1 void projection_operator_initialize

引数 無し

射影演算子の自動生成を行う関数。以下のことを順に行っている。

frag_charge、frag_charge_u、frag_basis に記録されている内容を、十分にバッファー領域を取った frag_charge_buf、frag_charge_u_buf、frag_basis_buf に移し、もとの配列領域を解放する。この時、frag_charge、frag_charge_u、frag_basis よりも後で確保された全ての配列のポインタを取得し直す。

make_projection_operator_01 関数で、射影演算子を作成する。

frag_charge、frag_charge_u、frag_basis を再確保し、frag_charge_buf、frag_charge_u_buf、frag_basis_buf の値をこちらに移す（ バッファーの方には射影演算子作成の過程で、適切に修正された値が入っている ）。

frag_charge_buf、frag_charge_u_buf、frag_basis_buf を解放する。この時、これらの配列が確保されてから解放されるまでに、新たに確保された全ての配列のポインタを取得し直す。

すべての CPU で同じプロセスを行うので（ 並列化されていない)、値を「 合わせる 」必要は無いように思えるが、局在化軌道の順番などは、異なる CPU で同じ結果になっている保証がないので rank=0 の値をブロードキャストする（ 射影演算子の自動生成を並列化するのが良いのだが、現在は非並列の状態)。

13.2 void make_projection_operator

引数 無し

1	**int ***	frag_charge_buf	[frag_n * charge_n]	作業領域
2	**double ***	frag_charge_u_buf	[frag_n * charge_n]	作業領域

射影演算子を決める関数。以下のグローバル変数に領域が割り当てられ値がセットされる。

```
int      frag_pro_n_all
int *    frag_pro_n
int *    frag_pro_idx
int *    frag_pro
int      frag_link_n_all
int *    frag_link_n
int *    frag_link_idx
int *    frag_link_1
```

```
int *    frag_link_2
int      prj_total_n
int      prj_total_nbo
int      prj_total_nba
int *    prj_nba
int *    prj_basis_idx
int *    prj_basis
int *    prj_nbo
int *    prj_c_idx
double * prj_c
int *    prj_ifrag
int *    prj_jfrag
```

また、frag_charge_u、frag_chaarge、frag_basis の値が、結合の切断に伴い修正される。これらのグローバル変数の値は、（並列化されずに）各 CPU で別個に計算されるが、projection_operator_initialize 関数で rank=0 の値に同期される。

13.3 void mkpro_get_iatom

引数 5

1	**const int**	ifrag		フラグメントの番号
2	**const int**	jatom		「追加された原子核」の番号
3	**const int ***	frag_charge_buf	[frag_n * charge_n]	各フラグメントに含まれる原子核
4	**int ***	iatom_out		jatom と結合を成す原子核
5	**double ***	dist		iatom_out と jatom の距離

「ifrag に追加された原子核」と結合をなす「ifrag 内部の原子核」をピックアップする関数。「追加された原子核」を jatom として入力し、「内部の原子核」が iatom_out として出力される。具体的には、追加された原子核から再近接にある原子番号が 2 より大きな原子核の番号を、iatom_out として返す。

13.4 void mkpro_make_ch4_coordinate

引数 7

1	**const int**	iatom	「フラグメント内部の原子核」の番号
2	**const int**	jatom	「追加された原子核」の番号
3	**double ***	c1	CH_4 の炭素原子の座標
4	**double ***	h2	CH_4 の水素原子の 1 番目の座標
5	**double ***	h3	CH_4 の水素原子の 2 番目の座標
6	**double ***	h4	CH_4 の水素原子の 3 番目の座標
7	**double ***	h5	CH_4 の水素原子の 4 番目の座標

iatom と jatom 間の結合の切断を行うための、仮想的な CH_4 分子の座標を計算する関数。jatom 上に炭素原子が置かれ iatom の方向に 1 番目の水素原子が置かれる。つまり、1 番目の水素原子がフラグメントの「内側」の方向を向いている。

13.5 void mkpro_make_ch2o_coordinate

引数 9

1	**const int**	iatom	「フラグメント内部の原子核」の番号
2	**const int**	jatom	「追加された原子核」の番号
3	**const int**	jfrag	
4	**const int**	frag_charge_buf	
5	**double ***	iatom_o	CH_4 の炭素原子の座標
6	**double ***	c1	CH_4 の炭素原子の座標
7	**double ***	h2	CH_4 の水素原子の 1 番目の座標
8	**double ***	o3	CH_4 の水素原子の 2 番目の座標
9	**double ***	h4	CH_4 の水素原子の 3 番目の座標

iatom と jatom 間の結合の切断を行うための、仮想的な CH_2O 分子の座標を計算する関数（今後の拡張のための実装）。

13.6 void mkpro_rhf_ch4

引数 18

1	**const int**	jatom		「追加された原子核」の番号
2	**const double ***	c1		CH_4 の炭素原子の座標
3	**const double ***	h2		CH_4 の水素原子の 1 番目の座標
4	**const double ***	h3		CH_4 の水素原子の 2 番目の座標
5	**const double ***	h4		CH_4 の水素原子の 3 番目の座標
6	**cnost double ***	h5		CH_4 の水素原子の 4 番目の座標
7	**int ***	nba		CH_4 の基底関数の数
8	**int ***	nsh		CH_4 のシェルの数
9	**int ***	nbo		CH_4 の ao の数
10	**int ***	itr		RHF 計算のイタレーションの回数
11	**double ***	energy		RHF 計算のエネルギー
12	**int ***	itr_loc		局在化の際のイタレーションの回数
13	**double ***	c_pop		結合軌道の炭素原子上の mulliken
14	**double ***	h_pop		結合軌道の水素原子上の mulliken
15	**int ***	nba_p		projection orbital の基底関数の数
16	**int ***	nbo_p		projection orbital の ao の数
17	**int ***	basis_pro		
18	**double ***	nbo_p	[5 * 256]	projection orbital の 係数

仮想的な CH_4 分子の RHF 計算と軌道局在化を自動的に行い、projection orbital を生成する関数。

13.7 void mkpro_rhf_ch2o

引数 18

1	**const int**	jatom
2	**const int**	iatom_o
3	**const double ***	c1
4	**const double ***	h2
5	**const double ***	o3
6	**const double ***	h4
7	**int ***	nba
8	**int ***	nsh
9	**int ***	nbo
10	**int ***	itr
11	**double ***	energy
12	**int ***	itr_loc
13	**double ***	c_pop
14	**double ***	h_pop
15	**int ***	nba_p
16	**int ***	nbo_p
17	**int ***	basis_pro
18	**double ***	nbo_p

仮想的な CH_2O 分子の RHF 計算と軌道局在化を自動的に行い、projection orbital を生成する関数。（今後の拡張のために実装）。

13.8 void mkpro_print

引数 無し

projection orbital の情報を出力する関数（デバッグ用）。

13.9 int get_frag_n_projection

引数 1

1	**const int**	ifrag	フラグメントの番号

フラグメントに属する射影演算子の局在化軌道の数を返す関数。

13.10 void make_frag_projection

引数 6

1	**const int**	ifrag	フラグメントの番号
2	**const int**	n_prj	射影演算子を構成する軌道の数
3	**const int**	nba1	フラグメントの基底の数
4	**const int**	nbo1	フラグメントの ao の数
5	**const int ***	ba	フラグメントに含まれる基底の番号
6	**double ***	c_prj	射影演算子を構成する軌道の係数

フラグメントに属する射影演算子の局在化軌道の係数を返す関数。

13.11 void get_dimer_n_projection

引数 2

1	**const int**	ifrag	フラグメントの番号
2	**const int**	jfrag	フラグメントの番号

フラグメントペアに属する射影演算子の局在化軌道の数を返す関数。

13.12 void make_dimer_projection

引数 7

1	**const int**	ifrag	フラグメントの番号
2	**const int**	jfrag	フラグメントの番号
3	**const int**	n_prj	
4	**const int**	nba1	
5	**const int**	nbo1	
6	**const int ***	ba	
7	**double ***	c_prj	

フラグメントペアに属する射影演算子の局在化軌道の係数を返す関数。

13.13 void make_projection_matrix

引数 5

1	**const int**	n_prj	
2	**const int**	nbo1	
3	**const int**	c_prjj	
4	**const int**	s	
5	**const int**	mat_prj	出力

射影演算子行列を返す関数。

14 フラグメント

14.1 void initialize_fragment

引数 無し

フラグメントに関連するグローバル変数の領域の確保と値の代入を行う。

14.2 int get_frag_nch1_all

引数 無し

フラグメントに含まれる原子核の数（nch1）の合計返す関数。

14.3 int get_frag_nba1_all

引数 無し

フラグメントに含まれる基底関数の数（nba1）の合計返す関数。

14.4 int get_frag_nsh1_all

引数 無し

フラグメントに含まれるシェルの数（nsh1）の合計返す関数。

14.5 int get_frag_nbo1_all

引数 無し

フラグメントに含まれる bo の数（nbo1）の合計返す関数。

14.6 int get_frag_nbot_all

引数 無し

フラグメントの基底関数に関する上三角行列の要素数（nbot）の合計返す関数。

14.7 int get_frag_nbo2_all

引数 無し

フラグメントの基底関数に関する正方行列の要素数（nbo2）の合計返す関数。

14.8 int get_frag_nele_all

引数 無し

フラグメントに含まれる電子数の合計返す関数。

14.9 int get_frag_nsh1

引数 1

1	**const int**	ifrag	フラグメントの番号

ifrag 番目のフラグメントの nsh1 を返す関数。

14.10 void get_frag_nsh1_aux

引数 1

1	**const int**	ifrag	フラグメントの番号

ifrag 番目のフラグメントの補助基底（RI 近似で使用される）に関する nsh1 を返す関数。

14.11 int get_frag_nbo1

引数 1

1	**const int**	ifrag	フラグメントの番号

ifrag 番目のフラグメントの nbo1 を返す関数。

14.12 void get_frag_nbo1_aux

引数 1

1	**const int**	ifrag	フラグメントの番号

ifrag 番目のフラグメントの補助基底（RI 近似で使用される）に関する nbo1 を返す関数。

14.13 void make_frag_shell

引数 無し

フラグメントのシェルに関連する変数をセットする関数。以下のグローバル変数に値が代入される。

```
int    * frag_nsh1
int    * frag_nsh1_idx
int    * frag_nsh1_xyz_idx
int    * frag_shell
double * frag_shell_xyz
```

14.14 void make_frag_aux_shell

引数 3

1	**const int**	ifrag	フラグメントの番号
2	**const int ***	shell	
3	**const double ***	shell_aux	

フラグメントの補助基底（RI 近似で使用される）に関するシェルに関連する変数をセットする関数。

14.15 void make_frag_bo

引数 無し

フラグメントの bo に関連する変数をセットする関数。以下のグローバル変数に値が代入される。

```
int    * frag_nbo1
int    * frag_nbot
int    * frag_nbo2
int    * frag_nbo1_idx
int    * frag_nbot_idx
int    * frag_nbo2_idx
```

14.16 void make_frag_scf_esa_list

引数 4

1	**int ***	list_scf_n	ダイマー SCF 計算を行う相手の数
2	**int ***	list_es_n	ダイマー ES 計算を行う相手の数
3	**int ***	list_scf	ダイマー SCF 計算を行う相手のリスト
4	**int ***	list_es	ダイマー ES 計算を行う相手のリスト

各フラグメントに関して、ダイマー SCF 計算を行う相手のフラグメントおよびダイマー ES 計算を行う相手のフラグメント決める関数。

14.17 void make_frag_calc_order

引数 3

1	**int ***	mon_calc_order	モノマーの計算順
2	**int ***	mon_para_stage_scc	フラグメントのステージ分割（ scc ）
3	**int ***	mon_para_stage_mon	フラグメントのステージ分割（ mon ）

モノマー計算におけるフラグメントの計算順をセットする関数。以下のグローバル変数に値がセットされる。

```
int *  mon_calc_order
int    mon_para_stage_scc [ PARA_MPI_MAX_STAGE ]
int    mon_para_stage_mon [ PARA_MPI_MAX_STAGE ]
```

14.18 int get_frag_n_nucleus

引数 2

1	**const int**	ifrag	フラグメントの番号
2	**const int**	type	数を返す際のタイプ

指定されたフラグメントに関して、原子核の数を数えて返す関数。タイプで指定された値に対応して、異なる数え方をする。

```
type=0  系全体の原子核の数を返す（ charge_n  ）
type=1  そのフラグメントのみの原子核の数を返す（ frag_nch1[ifrag] ）
type=2  そのフラグメント以外の原子核の数を返す
```

14.19 void make_frag_nucleus

引数 4

1	**const int**	ifrag	フラグメントの番号
2	**const int**	type	タイプ
3	**double ***	charge_i	原子核の番号がセットされる
4	**double ***	charge	原子核の電荷がセットされる
5	**double ***	charge_xyz	原子核の座標がセットされる

指定されたフラグメントの関して、原子核の情報を返す関数。タイプで指定された値に応じて、異なる情報がセットされる。

```
type=0  系全体の原子核を返す（ nucleus_u の電荷、座標 ）。
type=1  そのフラグメントの原子核を返す（ frag_nucleus_u の電荷、座標 ）。
type=2  そのフラグメント以外の原子核を返す（ frag_nucleus_u の電荷、座標 ）。
```

14.20 int make_frag_esp_list_0

引数 7

1	**const int**	ifrag	フラグメントの番号
2	**int ***	frag_0_list_4_n	4 中心で計算する数
3	**int ***	frag_0_list_3_n	3 中心近似で計算する数
4	**int ***	frag_0_list_m_n	点電荷近似で計算する数
5	**int ***	frag_0_list_4	4 中心で計算するフラグメントの番号
6	**int ***	frag_0_list_3	3 中心近似で計算するフラグメントの番号
7	**int ***	frag_0_list_m	点電荷近似で計算するフラグメントの番号

指定されたフラグメントに関して、環境静電ポテンシャルを計算する際に使われる近似に対応するフラグメントリストを作る関数。

14.21 int make_frag_pair_basis

引数 6

1	**const int**	ifrag	フラグメントの番号
2	**const int**	jfrag	フラグメントの番号
3	**int ***	nbasis	ダイマーの基底関数の数がセットされる
4	**int ***	nbasis_in	ダイマーの「内部の基底関数」の数がセットされる
5	**int ***	basis	ダイマーの基底関数の番号がセットされる
6	**int ***	basis_add_frag	「追加された基底関数」のフラグメント番号がセットされる

フラグメントペアに関する基底関数の情報を返す関数。

14.22 void make_frag_pair_aux_basis

引数 4

1	**const int**	ifrag
2	**const int**	jfrag
3	**int ***	nbasis
4	**int ***	basis

フラグメントペアに関する補助基底（RI 近似で使用される）の基底関数の情報を返す関数。

14.23 void make_frag_pair_shell

引数 6

1	**const int**	nbasis
2	**const int**	basis
3	**int ***	nshell
4	**int ***	nbo
5	**int ***	shell
6	**double ***	shell_xyz

フラグメントペアに関するシェルの情報を返す関数。

14.24 void get_frag_pair_aux_nsh1

引数 2

1	**const int**	nbasis
2	**const int ***	basis

フラグメントペアに関する補助基底（RI 近似で使用される）のシェルの数を返す関数。

14.25 void make_frag_pair_aux_shell

引数 5

1	**const int**	nbasis
2	**const int ***	basis
3	**int ***	nbo
4	**int ***	shell
5	**double ***	shell_xyz

フラグメントペアに関する補助基底（RI 近似で使用される）のシェルの情報を返す関数。

14.26 void make_frag_pair_calc_order_n

引数 4

1	**int ***	scf_order_n
2	**int ***	es_order_n

ダイマー計算を実行するフラグメントペアの数を返す関数。

14.27 void make_frag_pair_calc_order

引数 1

1	**int ***	scf_order	

ダイマー計算におけるフラグメントペアの計算順を返す関数。

14.28 int get_frag_pair_n_nucleus

引数 3

1	**const int**	ifrag	フラグメントの番号
2	**const int**	jfrag	フラグメントの番号
3	**const int**	type	数を返す際のタイプ

フラグメントペアに関する原子核の数を数えて返す関数。タイプで指定された値に対応して、異なる数え方をする。

```
type=0  系全体の原子核の数を返す（ nucleus_n の値 ）
type=1  フラグメントペアの原子核の数を返す（ frag_nnu1[ifrag]+frag_nnu1[jfarg] ）
type=2  フラグメントペア以外の原子核の数を返す（ frag_nnu1 [ ] の和 ）
```

14.29 int make_frag_pair_nucleus

引数 6

1	**const int**	ifrag	フラグメントの番号
2	**const int**	jfrag	フラグメントの番号
3	**const int**	type	数を返す際のタイプ
4	**double ***	charge_i	原子核の電荷がセットされる
5	**double ***	charge	原子核の電荷がセットされる
6	**double ***	charge_xyz	原子核の座標がセットされる

フラグメントペアに関する原子核の情報を返す関数。タイプで指定された値に対応して、異なる情報を返す。

```
type = 0  系全体の原子核を返す（ nucleus_u の値 ）
```

type = 1　フラグメントペアの原子核を返す（ frag_nucleus_u[ifrag] と frag_nucleus_u[jfarg] ）

type = 2　フラグメントペア以外の原子核を返す（ frag_nnu1[ ] の和 ）

14.30 void make_frag_pair_esp_list_0

引数 8

1	**const int**	ifrag	フラグメントの番号
2	**const int**	ifrag	フラグメントの番号
3	**int ***	frag_pair_0_list_4_n	4 中心で計算する数
4	**int ***	frag_pair_0_list_3_n	3 中心で計算する数
5	**int ***	frag_pair_0_list_m_n	点電荷で計算する数
6	**int ***	frag_pair_0_list_4	4 中心で計算するフラグメント
7	**int ***	frag_pair_0_list_3	3 中心で計算するフラグメント
8	**int ***	frag_pair_0_list_m	点電荷で計算するフラグメント

指定されたフラグメントペアにおいて、環境静電ポテンシャルを計算する際に使われる近似に対応するフラグメントリストを作る関数。

14.31 void chk_frag_pair_calc

引数 2

1	**const int**	ifrag
2	**const int**	jfrag

指定されたフラグメントペアの計算を行うか否かを返す関数（計算するダイマーペアを制限した計算の場合のみ、否が返る可能性がある）。

14.32 void chk_frag_pair_calc_all

引数 1

1	**const int**	ifrag

指定されたフラグメントに関するペアの計算が全て実行されるか否かを返す関数（計算するダイマーペアを制限した計算の場合のみ、否が返る可能性がある）。

14.33 void chk_frag_esp_4

引数 2

1	**const int**	ifrag
2	**const int**	jfrag

指定されたフラグメントペアが、互いに 4 中心積分で環境静電ポテンシャルを計算する関係にあるか否かを返す関数。

14.34 void chk_frag_esp_3

引数 2

1	**const int**	ifrag
2	**const int**	jfrag

指定されたフラグメントペアが、互いに 3 中心積分の近似を用いて環境静電ポテンシャルを計算する関係にあるか否かを返す関数。

14.35 void chk_frag_dscf

引数 2

1	**const int**	ifrag
2	**const int**	jfrag

指定されたフラグメントペアが、互いにダイマー SCF 計算を実行する関係にあるか否かを返す関数（否の場合、ダイマー ES 近似によって計算される）。

14.36 void get_frag_pair_index

引数 2

1	**const int**	ifrag
2	**const int**	jfrag

指定されたフラグメントペアの配列におけるインデックスを返す関数。指定されたペアが計算が実

行されないペアであれば-1が返る。従って、そのペアが計算されるペアであることをchk_frag_pair_calc関数で確認した上でこの関数を呼ばなければならない（バグの原因となるので注意）。

14.37 void init_frag_esp_list_n

引数 2

1	**int ***	n4
2	**int ***	n3

環境静電ポテンシャルの計算に関するリストを作成する際に呼ばれる関数。リストの要素数を返す。

14.38 void init_frag_esp_list

引数 6

1	**int ***	list_4_n
2	**int ***	list_3_n
3	**int ***	list_4_idx
4	**int ***	list_3_idx
5	**int ***	list_4
6	**int ***	list_3

環境静電ポテンシャルの計算に関するリストを作成する関数。

14.39 void init_frag_dscf_list_n

引数 1

1	**int ***	n_scf

ダイマーSCF計算に関するリストを作成する際に呼ばれる関数。リストの要素数を返す。

```
1   int *    list_scf_n
2   int *    list_scf_idx
3   int *    list_scf
```

ダイマー SCF 計算に関するリストを作成する関数。

14.41 void get_frag_from_charge

引数 2

```
1   const int    icharge
2   int *        frag_list
```

原子核の通し番号が与えられた際、その原子核を含むフラグメントの番号を返す関数。結合の切断がある場合、複数のフラグメントが返される。

14.42 void get_frag_from_basis

引数 2

```
1   const int    ibasis
2   int *        frag_list
```

基底関数の通し番号が与えられた際、その基底関数を含むフラグメントの番号を返す関数。結合の切断がある場合、複数のフラグメントが返される。

15 モノマー SCC 計算

15.1 void initialize_monomer_scc

引数 無し

モノマー SCC 計算に関連するグローバル変数の allocate と初期化を行う関数。

15.2 void monomer_scc_print_title

引数 無し

モノマー SCC 計算のタイトルを出力する関数。

15.3 void calculation_monomer_scc

引数 無し

モノマー SCC 計算を行う関数。この関数の中で monomer_rhf_scc 関数が繰り返し呼ばれる。

15.4 void monomer_scc_result_bcast

引数 4

1	**const int**	ncomm
2	**double ***	e_delta
3	**double ***	energy
4	**double ***	energy_d

モノマー SCC 計算のイタレーションごとに、必要な数値をブロードキャストする関数。

16 モノマー計算

16.1 void calculation_monomer

引数 無し

モノマー計算を行う関数。

16.2 void monomer_print_title

引数 2

1	**const int**	iorder
2	**const int**	ifrag

モノマー計算の際、タイトルをプリントする関数。

16.3 void monomer_result_bcast_rhf

引数 2

1	**const int**	ncomm
2	**const MPI_Comm**	icomm

モノマー RHF 計算の結果をブロードキャストする関数。

16.4 void monomer_result_bcast_cmp2

引数 1

1	**const int**	ncomm

モノマー MP2 計算の結果をブロードキャストする関数。

16.5 void monomer_result_bcast_ri_cmp2

引数 1

1	**const int**	ncomm

モノマー RI{MP2 計算の結果をブロードキャストする関数。

16.6 void monomer_result_bcast_lmp2

引数 1

1	**const int**	ncomm

モノマー LMP2 計算の結果をブロードキャストする関数。

17 ダイマー ES 計算

17.1 void calculation_dimer_es

引数 無し

ダイマー ES 計算を行う関数。

17.2 void dimer_es_print_title

引数 無し

ダイマー ES 計算の際、タイトルをプリントする関数。

17.3 void dimer_es_4

引数 2

1	**double ***	energy_e_ij
2	**const MPI_Com**	icomm

ダイマー ES 近似における電子　電子間の斥力計算を行う。

17.4 void dimer_es_u

引数 3

1	**double ***	energy_u_i
2	**double ***	energy_u_j
3	**const MPI_Com**	icomm

ダイマー ES 近似における電子　核間の引力計算を行う。

18 ダイマー計算

18.1 void calculation_dimer

引数 無し

ダイマー計算を行う関数。

18.2 void dimer_print_title

引数 4

1	**const int**	icount
2	**const int**	ifrag
3	**const int**	jfrag
4	**const double**	dist

ダイマー計算の際、タイトルをプリントする関数。

18.3 void dimer_result_bcast_rhf

引数 2

1	**const int**	ncomm
2	**const MPI_Comm**	icomm

ダイマー RHF 計算の結果をブロードキャストする関数。

18.4 void dimer_result_bcast_cmp2

引数 1

1	**const int**	ncomm

ダイマー MP2 計算の結果をブロードキャストする関数。

18.5 void dimer_result_bcast_ri_cmp2

引数 1

1	**const int**	ncomm

ダイマー RI{MP2 計算の結果をブロードキャストする関数。

18.6 void dimer_result_bcast_lmp2

引数 1

1	**const int**	ncomm

ダイマー LMP2 計算の結果をブロードキャストする関数。

19 RHF 計算

19.1 void initialize_rhf

引数 無し

RHF 計算関連のグローバル変数を確保し、初期化する関数。

19.2 void rhf_monomer_scc

引数 10

1	**const int**	ifrag
2	**const int**	scc_itr
3	**const int**	iorder
4	**const int**	chk_esp
5	**const int**	chk_ini
6	**double ***	energy
7	**double ***	energy_d
8	**double ***	esp_time
9	**double ***	rhf_time
10	**const MPI_Comm**	icomm

モノマー SCC 計算で、モノマー RHF 計算を行う関数。

19.3 void rhf_monomer

引数 9

1	**const int**	ifrag
2	**const int**	chk_esp
3	**const int**	chk_ini
4	**double ***	out_dmat
5	**double ***	out_fmat
6	**double ***	out_vmat
7	**double ***	esp_time
8	**double ***	rhf_time
9	**const MPI_Comm**	icomm

モノマー RHF 計算を行う関数。

19.4 void rhf_monomer_field_property

引数 3

1	**const int**	ifrag
2	**const MPI_Comm**	icomm
3	**const double ***	in_dmat

モノマー計算で、電子密度、静電ポテンシャル、電場を計算する関数。

19.5 void rhf_dimer

引数 17

1	**const int**	ifrag
2	**const int**	jfrag
3	**const int**	dim_nshell
4	**const int**	dim_nbasis
5	**const int**	dim_nbo
6	**const int**	dim_nocc
7	**const int**	dim_cp_corr
8	**const MPI_Comm**	icomm
9	**const int ***	dim_nmo
10	**const double ***	dim_shell
11	**const double ***	dim_shell_xyz
12	**const int ***	dim_basis
13	**const double ***	dim_fck
14	**const double ***	dim_d
15	**const int**	dim_c
16	**const int**	dim_moe
17	**const int**	dim_v

ダイマー RHF 計算を行う関数。

19.6 void rhf_dimer_field_property

引数 10

1	**const int**	ifrag
2	**const int**	jfrag
3	**const int**	dim_nshell
4	**const int**	dim_nbasis
5	**const int**	dim_nbo
6	**const MPI_Comm**	icomm

7	**const double ***	dim_shell
8	**const double ***	dim_shell_xyz
9	**const int ***	dim_basis
10	**const double ***	in_dmat

ダイマー計算で、電子密度、静電ポテンシャル、電場を計算する関数。

19.7 void make_init_mo_hcore

引数 11

1	**const int**	nbo
2	**int ***	nmo
3	**const int**	orth
4	**const double**	tv
5	**const int**	lprint
6	**const double ***	s
7	**const double ***	h
8	**double ***	c
9	**double ***	x
10	**double ***	eng
11	**double ***	work

初期軌道を作成する関数。H{core 法を使う。

19.8 void make_init_mo_rhf

引数 18

1	**const int**	ifrag
2	**const int**	nbo
3	**const int**	nshell
4	**const int**	nbasis
5	**const int**	ncharge
6	**const int**	nelec
7	**const int**	nprj
8	**const double**	nuc_eng
9	**const int**	lprint
10	**const MPI_Comm**	icomm
11	**const int ***	basis
12	**const double ***	charge_xyz
13	**const double ***	charge
14	**const int ***	shell
15	**const double ***	shell_xyz
16	**const dobule ***	ini_ovl

17	**double ***	c_prj
18	**double ***	c_out

初期軌道を作成するための関数。収束しやすい別の基底関数（STO-3G などの小さな基底関数）で RHF 解を求め、本来の基底関数空間に射影する。

19.9 void make_ini_shell

引数 6

1	**const int**	nbasis
2	**const int ***	basis
3	**int ***	ini_nshell
4	**int ***	ini_nbo
5	**int ***	ini_shell
6	**double ***	ini_shell_xyz

初期軌道を作成する際に呼ばれる関数。

19.10 void pro_out_init_mo

引数 13

1	**const int**	nbo
2	**const int**	nmo
3	**const int**	nprj
4	**const double ***	s
5	**const double ***	prj_c
6	**double ***	c
7	**double ***	s_prj
8	**double ***	s_prj_inv
9	**double ***	work1
10	**double ***	work2
11	**double ***	work3
12	**double ***	work4
13	**double ***	work5

初期軌道の作成の際に呼ばれる関数。

19.11 void rhf_iteration

引数 29

1	**const int**	maxit
2	**const int**	nshell
3	**const int**	nbo
4	**const int**	nmo
5	**const int ***	nocc
6	**const int**	ndiis
7	**const int**	nfock
8	**const double**	diis_tv
9	**const double ***	nuc_eng
10	**const int**	lprint
11	**const MPI_Comm**	icomm
12	**const int ***	shell
13	**const double ***	shell_xyz
14	**const double ***	h
15	**const double ***	x
16	**const double ***	s
17	**const int ***	kab_idx
18	**const double ***	kab_sh
19	**const double ***	kab
20	**const double ***	p_x
21	**const double ***	p_y
22	**const double ***	p_z
23	**int**	chk_conv_out
24	**int**	nitr
25	**double ***	total
26	**double ***	moe
27	**double ***	c
28	**double ***	d
29	**double ***	f

RHF イタレーションを行う関数。

19.12 void make_g_matrix

引数 38

1	**const int**	itr l
2	**const int**	nshell
3	**const int**	nbo
4	**const int**	nfock
5	**const long int**	eri_val_num
6	**const long int**	buf_size
7	**long int ***	buf_num
8	**long int ***	get_eri_num
9	**long int ***	shell_rec_num
10	**const double**	tv

19.14 void make_effective_fock_diis

引数 18

1	**const int**	nbo
2	**const int**	nerr
3	**const int**	ndiis
4	**const double**	diis_tv
5	**const double ***	s
6	**int ***	nrec
7	**int ***	chk_diis
8	**double ***	e_max
9	**double ***	e_max_old
10	**double ***	diis_fock
11	**double ***	diis_d
12	**double ***	diis_e
13	**double ***	f
14	**dobule ***	d
15	**double ***	b
16	**double ***	b_inv
17	**dobule ***	buf1
18	**dobule ***	buf2

DIIS を行う際、有効 Fock 行列を計算する関数。

19.15 void rhf_grad_add_ovl

引数 8

1	**const int**	nba		基底関数の数
2	**const int**	nbo		nbo の値
3	**const MPI_Comm**	nbo		nbo の値
4	**const int ***	basis		基底関数の情報
5	**const int ***	shell		シェルの情報
6	**const double ***	shell_xyz		シェルの座標
7	**const double ***	w		energy waited densiti matrix
8	**double ***	grad	[3 * 全体のチャージの数]	出力（ 足し込まれる ）

RHF エネルギーの 1 次微分に対する、重なり積分の寄与を計算し足し込む関数。この関数の中で、get_ovl_grad 関数が呼ばれる。ich 番目の核座標の微分への寄与は grad の以下の位置に足し込まれる。

```
grad [ ich * 3 + 0 ] ---> ich 番目の核の x 座標の微分の寄与
grad [ ich * 3 + 1 ] ---> ich 番目の核の y 座標の微分の寄与
grad [ ich * 3 + 2 ] ---> ich 番目の核の z 座標の微分の寄与
```

この場合の ich は全体の通し番号における核の番号で、フラグメント内の核の番号では無い。

19.16 void rhf_grad_add_kei

引数 8

1	**const int**	nba	基底関数の数
2	**const int**	nbo	nbo の値
3	**const MPI_Comm**	icomm	
4	**const int ***	basis	基底関数の情報
5	**const int ***	shell	シェルの情報
6	**const double ***	shell_xyz	シェルの座標
7	**const double ***	d	densiti matrix
8	**double ***	grad	出力（ 足し込まれる ）

RHF エネルギーの 1 次微分に対する、運動エネルギー積分の寄与を計算し足し込む関数。この関数の中で、get_kei_grad 関数が呼ばれる。

19.17 void rhf_grad_add_nai

引数 10

1	**const int**	nba	基底関数の数
2	**const int**	nbo	nbo の値
3	**const int**	ncharge	原子核の数
4	**const MPI_Comm**	icomm	
5	**const int ***	basis	基底関数の情報
6	**const int ***	shell	シェルの情報
7	**const double ***	shell_xyz	シェルの座標
8	**const double ***	charge_xyz	原子核の座標
9	**const double ***	charge_val	原子核の電荷の値
10	**const double ***	d	densiti matrix
11	**double ***	grad	出力（ 足し込まれる ）

RHF エネルギーの 1 次微分に対する、核引力積分の寄与を計算し足し込む関数。この関数の中で、get_nai_grad 関数が呼ばれる。核引力積分の 1 次微分からは、電場積分の項が出てくるが、これは rhf_grad_add_efi 関数で別に計算される。

19.18 void rhf_grad_add_efi

引数 11

1	**const int**	ns	シェルの数
2	**const int**	nbo	nbo の値
3	**const int**	ncharge	原子核の数
4	**const MPI_Comm**	icomm	MPI コミュニケータ
5	**const int ***	shell	シェルの情報
6	**const double ***	shell_xyz	シェルの座標
7	**const int ***	charge_i	原子核の番号
8	**const double ***	charge_xyz	原子核の座標
9	**const double ***	charge_val	原子核の電荷の値
10	**const double ***	d	密度行列
11	**double ***	grad	出力（ 足し込まれる ）

RHF エネルギーの 1 次微分に対する、電場積分の寄与を計算し足し込む関数。電場積分は核引力積分の 1 次微分から出現する。この関数の中で、get_efi 関数が呼ばれる。

19.19 void ortonormalize_can

引数 7

1	**const int**	n
2	**int ***	m
3	**const double**	tv
4	**const double ***	ovl
5	**double ***	x
6	**double ***	w
7	**double ***	work

正準直交化を行う関数。

19.20 void ortonormalize_sym

引数 5

1	**const int**	n
2	**const double ***	ovl
3	**double ***	x
4	**double ***	w
5	**double ***	work

対称直交化を行う関数。

19.21 void ortonormalize_low

引数 9

1	**const int**	nbo
2	**const int**	nmo
3	**const double ***	w
4	**const double ***	c
5	**const double ***	ovl
6	**double ***	c_out
7	**double ***	work1
8	**double ***	work2
9	**double ***	work3

Lowdin 直交化を行う関数。

19.22 void rhf_make_init_density_matrix

引数 3

1	**const int**	ifrag
2	**const int**	nbo1
3	**double ***	d

初期密度行列を作成する関数。make_init_mo_rhf 関数を用いて初期軌道を作成する時に呼ばれる。

19.23 void rhf_make_init_density_matrix_basis

引数 3

1	**const char ***	basis_name
2	**const double ***	charge
3	**double ***	out

初期密度行列を作成する際に呼ばれる関数。

20 軌道局在化計算

20.1 void localize_mom

引数 12

1	**const int**	max_itr
2	**const double**	b_st_tv
3	**const int**	nbo
4	**const int**	nmo
5	**const double ***	c_mat
6	**const double ***	m_mat
7	**double ***	c_mat_out
8	**double ***	c_s
9	**double ***	c_t
10	**int ***	chk_conv
11	**int ***	itr
12	**int**	lprint

Boys の局在化を行う関数。

20.2 void localize_pop

引数 16

1	**const int**	max_itr
2	**const double**	b_st_tv
3	**const int**	nbo
4	**const int**	nmo
5	**const int**	natom
6	**const double ***	c_mat
7	**const int ***	natom_idx
8	**const double ***	ovl_mat
9	**double ***	c_mat_out
10	**double ***	c_s
11	**double ***	c_t
12	**double ***	k_s
13	**double ***	k_t
14	**int ***	chk_conv
15	**int ***	itr
16	**int**	lprint

Pipek-Mechy の局在化を行う関数。

20.3 void make_site_localize_orbital

引数 8

1	**const int**	nmo
2	**const int**	nbo
3	**const int**	natom
4	**const int ***	atom_idx
5	**const int ***	atom_site
6	**const double ***	c_mat_out
7	**const double ***	ovl_mat
8	**int ***	site

局在化軌道がどの原子上に存在しているかを判定する関数。

20.4 void print_localized_orbital

引数 11

1	**const int**	nmo
2	**const int**	imo1
3	**const int**	imo2
4	**const int**	nbo
5	**const int**	natom
6	**const int**	lprint
7	**const int ***	atom_idx
8	**const double ***	c_mat_out
9	**const double ***	ovl_mat
10	**int ***	atom_order
11	**double ***	atom_pop

局在化軌道を標準出力にプリントする関数。

21 MP2 計算

21.1 void initialize_cmp2

引数 無し

MP2 計算に関連するグローバル変数を確保し、初期化する関数。

21.2 void cmp2_monomer

引数 2

1	**const int**	ifrag
2	**const MPI_Comm**	icomm

モノマー計算の際、MP2 相関エネルギーを計算する関数。

21.3 void cmp2_dimer

引数 13

1	**const int**	ifrag
2	**const int**	jfrag
3	**const int**	dim_nshell
4	**const int**	dim_nbo
5	**const int**	dim_nocc
6	**const int**	dim_nmo
7	**const int**	dim_nfzc
8	**const int**	dim_cp_corr
9	**const MPI_Comm**	icomm
10	**const int ***	dim_shell
11	**const double ***	dim_shell_xyz
12	**const double ***	dim_moe
13	**const double ***	dim_c

ダイマー計算の際、 MP2 相関エネルギーを計算する関数。

21.4 void canonical_mp2

引数 28

1	**const long int**	ncore
2	**double ***	core
3	**const int**	n_orb_set
4	**const int**	nbo
5	**const int**	nmo
6	**const int**	nsh
7	**const int**	nocc
8	**const int**	nfzc
9	**const int**	max_basis_nsh
10	**const double**	th_iajs
11	**const double**	th_iars
12	**const double**	th_pqrs
13	**const int**	lprint
14	**const MPI_Comm**	icmm
15	**const double ***	moe
16	**const double ***	c
17	**const int ***	shell
18	**const double ***	shell_xyz
19	**const int ***	kab_idx
20	**const double ***	kab_sh
21	**const double ***	kab
22	**const double ***	p_x
23	**const double ***	p_y
24	**const double ***	p_z
25	**double ***	cmp2_energy
26	**double ***	cmp2_energy_s
27	**double ***	cmp2_energy_t
28	**double ***	integral_time

MP2 相関エネルギーの計算を行うため関数。core [ncore] として与えられたメモリ領域のみを用いて計算を進める。core 領域を管理するための、MP2 計算専用のメモリ管理システムを持ち、それに関わる関数名は cmp2_core_*となっている。***PAICS*** のメモリ管理システムを使用しないのは、モジュールとして独立させるため。

21.5 long int cmp2_get_remain_core

引数 無し

core 領域を管理するための、MP2 計算専用のメモリ管理システムの関数。core に残っているメモリサイズ（double 型単位）を返す。

21.6 void cmp2_core_print

引数 無し

core 領域を管理するための、MP2 計算専用のメモリ管理システムの関数。core に割り当てられている配列を標準出力にプリントする。

21.7 void cmp2_core_alloc

引数 2

1	**const char ***	name
2	**const long int**	size

core 領域を管理するための、MP2 計算専用のメモリ管理システムの関数。core に、配列を割り当てる関数。

21.8 void cmp2_core_shift_left

引数 2

1	**const long int**	pos
2	**const long int**	n

core 領域を管理するための、MP2 計算専用のメモリ管理システムの関数。core に記録されている値を若い方にシフトする。

21.9 void cmp2_core_dealloc

引数 1

1	**const char ***	name

core 領域を管理するための、MP2 計算専用のメモリ管理システムの関数。core に確保されている配列を解放する。

21.10 double * cmp2_core_get_pointer

引数 1

1	**const char ***	name

core 領域を管理するための、MP2 計算専用のメモリ管理システムの関数。core に確保されている配列の先頭番地のポインター（double * 型）を返す。

21.11 long int cmp2_core_get_size

引数 1

1	**const char ***	name

core 領域を管理するための、MP2 計算専用のメモリ管理システムの関数。core に確保されている配列サイズ（double 型単位）を返す。

21.12 void

22	**const double ***	kab
23	**const double ***	p_x
24	**const double ***	p_y
25	**const double ***	p_z
26	**long int ***	n_int
27	**double ***	t_int
28	**double ***	energy_inout
29	**double ***	energy_inout_s
30	**double ***	energy_inout_t
31	**double ***	cmp2_gpqrs
32	**double ***	cmp2_wiajb
33	**double ***	cmp2_wiajs
34	**double ***	cmp2_wiqrs
35	**double ***	cmp2_mpi_buffer

ao 積分から mo 積分へ変換し、mp2 相関エネルギーを計算する関数。

21.13 void cmp2_get_pqrs_fixed_rs

引数 17

1	**const int**	ish_r
2	**const int**	ish_s
3	**const int**	nshell
4	**const int**	nbo
5	**const double**	tv
6	**const double**	tv_cauchy
7	**const double ***	cauchy
8	**const int ***	shell
9	**const double ***	shell_xyz
10	**const int ***	kab_idx
11	**const double ***	kab_sh
12	**const double ***	kab
13	**const double ***	p_x
14	**const double ***	p_y
15	**const double ***	p_z
16	**long int ***	n_int
17	**double ***	gpqrsr

ao 積分をまとめて取得するための関数。

21.14 void cmp2_allreduce_double

引数 5

1	**const long int**	mpi_val_size
2	**const long int**	mpi_buf_size
3	**double ***	mpi_val
4	**double ***	mpi_buf
5	**MPI_Comm**	icomm

データを All reduce するための関数。与えられたバッファーサイズに応じて複数回 MPI_All_reduce する。

22 RI–MP2 計算

22.1 void initialize_ri_cmp2

引数 無し

RI{MP2 計算に関連するグローバル変数を確保し、初期化する関数。

22.2 void ri_cmp2_monomer

引数 2

1	**const int**	ifrag
2	**const MPI_Comm**	icomm

モノマー計算の際、RI{MP2 相関エネルギーを計算する関数。

22.3 void ri_cmp2_dimer

引数 13

1	**const int**	ifrag
2	**const int**	jfrag
3	**const int**	dim_nshell
4	**const int**	dim_nbo
5	**const int**	dim_nocc
6	**const int**	dim_nmo
7	**const int**	dim_nfzc
8	**const int**	dim_cp_corr
9	**const MPI_Comm**	icomm
10	**const int ***	dim_shell
11	**const double ***	dim_shell_xyz
12	**const double ***	dim_moe
13	**const double ***	dim_c

ダイマー計算の際、RI{MP2 相関エネルギーを計算する関数。

22.4 void ri_canonical_mp2

引数 29

1	**const long int**	ncore
2	**double ***	core
3	**const int**	n_orb_set
4	**const int**	nbo
5	**const int**	nmo
6	**const int**	nsh
7	**const int**	nocc
8	**const int**	nfzc
9	**const int**	aux_nbo
10	**const int**	aux_nsh
11	**const int**	max_basis_nsh
12	**const int**	lprint
13	**const MPI_Comm**	icmm
14	**const double ***	moe
15	**const double ***	c
16	**const int ***	shell
17	**const double ***	shell_xyz
18	**const int ***	aux_shell
19	**const double ***	aux_shell_xyz
10	**const int ***	kab_idx
21	**const double ***	kab_sh
22	**const double ***	kab
23	**const double ***	p_x
24	**const double ***	p_y
25	**const double ***	p_z
26	**double ***	ri_cmp2_energy
27	**double ***	ri_cmp2_energy_s
28	**double ***	ri_cmp2_energy_t
29	**double ***	integral_time

RI{MP2 相関エネルギーの計算を行うため関数。core [ncore] として与えられたメモリ領域のみを用いて計算を進める。core 領域を管理するための、RI{MP2 計算専用のメモリ管理システムを持ち、それに関わる関数名は ri_cmp2_core_*となっている。***PAICS*** のメモリ管理システムを使用しないのは、モジュールとして独立させるため。

22.5 long int ri_cmp2_get_remain_core

引数 無し

core 領域を管理するための、RI{MP2 計算 2 専用のメモリ管理システムの関数。core に残っているメモリサイズ（double 型単位）を返す。

22.6 void ri_cmp2_core_print

引数 無し

core 領域を管理するための、RI{MP2 計算専用のメモリ管理システムの関数。core に割り当てられている配列を標準出力にプリントする。

22.7 void ri_cmp2_core_alloc

引数 2

1	**const char ***	name
2	**const long int**	size

core 領域を管理するための、RI{MP2 計算専用のメモリ管理システムの関数。core に、配列を割り当てる関数。

22.8 void ri_cmp2_core_shift_left

引数 2

1	**const long int**	pos
2	**const long int**	n

core 領域を管理するための、RI{MP2 計算専用のメモリ管理システムの関数。core に記録されている値を若い方にシフトする。

22.9 void ri_cmp2_core_dealloc

引数 1

1	**const char ***	name

core 領域を管理するための、RI{MP2 計算専用のメモリ管理システムの関数。core に確保されている配列を解放する。

22.10 double * ri_cmp2_core_get_pointer

引数 1

1	**const char ***	name

core 領域を管理するための、RI{MP2 計算専用のメモリ管理システムの関数。core に確保されている配列の先頭番地のポインター（double * 型）を返す。

22.11 long int ri_cmp2_core_get_size

引数 1

1	**const char ***	name

core 領域を管理するための、RI{MP2 計算専用のメモリ管理システムの関数。core に確保されている配列サイズ（double 型単位）を返す。

22.12 void ri_cmp2_get_rsq_fixed_q

引数 12

1	**const int**	nsh
2	**const int**	nbo
3	**const int ***	shell
4	**onnst double ***	shell_xyz
5	**const int ***	aux_shell
6	**const double ***	aux_shell_xyz
7	**const int ***	kab_idx
8	**const double ***	kab
9	**const double ***	p_x
10	**const double ***	p_y
11	**const double ***	p_z
12	**double ***	wrsq

RI{MP2 相関エネルギーの計算で使われる関数。

22.13 void ri_cmp2_make_biap

引数 27

1	**const int**	nsh
2	**const int**	nbo
3	**const int**	aux_nbo
4	**const int**	aux_nsh
5	**const int**	nfzc
6	**const int**	nval
7	**const int**	nvir
8	**const int**	aux_nbo_1
9	**const int**	aux_nbo_2
10	**const int ***	shell
11	**onnst double ***	shell_xyz
12	**const int ***	aux_shell
13	**const double ***	aux_shell_xyz
14	**const int ***	kab_idx
15	**const double ***	kab
16	**const double ***	p_x
17	**const double ***	p_y
18	**const double ***	p_z
19	**double ***	c
20	**double ***	vinv
21	**double ***	int_time
22	**double ***	trs_time
23	**double ***	tot_time
24	**double ***	wrsq
25	**double ***	wisq
26	**double ***	wiaq
27	**double ***	biap

RI{MP2 相関エネルギーの計算で使われる関数。

22.14 void ri_cmp2_allreduce_double

引数 5

1	**const long int**	mpi_val_size
2	**const long int**	mpi_buf_size
3	**double ***	mpi_val
4	**double ***	mpi_buf
5	**MPI_Comm**	icomm

データを All reduce するための関数。与えられたバッファーサイズに応じて複数回 MPI_All_reduce する。

23 局在化MP2計算

23.1 void initialize_lmp2

引数 無し

LMP2 計算に関連するグローバル変数を確保し初期化する関数。

23.2 void lmp2_monomer

引数 3

1	**const int**	ifrag
2	**const MPI_Comm**	icomm
3	**const double ***	fmat_in

モノマー計算の際、LMP2 相関エネルギーを計算する関数。

23.3 void lmp2_dimer

引数 16

1	**const int**	ifrag
2	**const int**	jfrag
3	**const int**	dim_nshell
4	**const int**	dim_nbasis
5	**const int**	dim_nbasis_in
6	**const int**	dim_nbo
7	**const int**	dim_nocc
8	**const int**	dim_nmo
9	**const int**	dim_nfzc
10	**const MPI_Comm**	icomm
11	**const int ***	dim_shell
12	**const double ***	dim_shell_xyz
13	**const int ***	dim_basis
14	**const int ***	dim_basis_add_frag
15	**const double ***	dim_fck
16	**const double ***	dim_c

ダイマー計算の際、LMP2 相関エネルギーを計算する関数。

23.4 void lmp2_make_ifie

引数 6

1	**const int**	nij	
2	**const int ***	ij_pair_1	
3	**const int ***	ij_pair_2	
4	**const int ***	site	
5	**const double ***	e_ij	
6	**double ***	lmp2_ifie	

局在化 MP2 の結果から IFIE を計算する関数。

23.5 void lmp2_dimer_analysis

引数 10

1	**const int**	nij	
2	**const int**	norb	
3	**const int**	nbo1	
4	**const int ***	ij_pair_1	
5	**const int ***	ij_pair_2	
6	**const int ***	site	
7	**const double ***	mom	
8	**const double ***	c_lmo	
9	**const double ***	e_ij	

局在化 MP2 の結果を解析する関数。

23.6 void local_mp2

引数 43

1	**const long int**	ncore	core の数
2	**double ***	core	計算で使用されるメモリ
3	**const double**	th_1	閾値
4	**const double**	th_1_dim	閾値
5	**const double**	th_2	閾値
6	**const double**	th_2_dim	閾値
7	**const double**	th_3	閾値
8	**const double**	th_4	閾値
9	**const double**	th_l1	閾値
10	**const double**	th_l2	閾値
11	**const int**	nbo1	ao の数

12	**const int**	nsh1	シェルの数
13	**const int**	ndoc	占有軌道の数
14	**const int**	nfzc	凍結軌道の数
15	**const int**	nba1	基底関数の数
16	**const int**	nprj	FMO の射影演算子の数
17	**const int**	max_itr	線形方程式の最大繰り返し
18	**const int**	lprint	プリントフラグ
19	**const MPI_Comm**	icomm	MPI のコミュニケーター
20	**const double ***	c_lmo	局在化軌道の係数
21	**const double ***	c_prj	局在化軌道の係数
22	**const double ***	fck_ao	ao に関する Fock 行列
23	**const double ***	ovl_ao	ao に関する重なり行列
24	**const double ***	mon_ao	ao に関するモーメント積分行列
25	**const double ***	x	基底関数の座標
26	**const double ***	y	基底関数の座標
27	**const double ***	z	基底関数の座標
28	**const int ***	nsh1_idx	シェルのインデックス
29	**const int ***	nba1_idx	基底関数のインデックス
30	**const int ***	site_lmo	局在化軌道のサイト
31	**const int ***	shell	シェルの情報
32	**const double ***	shell_xyz	シェルの情報
33	**const int ***	kab_idx	積分計算用に数値
34	**const double ***	kab_sh	積分計算用に数値
35	**const double ***	kab	積分計算用に数値
36	**const double ***	p_x	積分計算用の数値
37	**const double ***	p_y	積分計算用の数値
38	**const double ***	p_z	積分計算用の数値
39	**double ***	e_ij	ペア相関エネルギーの出力
40	**double ***	e_total	全相関エネルギー
41	**int ***	n_ij_pair	軌道ペアの数
42	**int ***	ij1	軌道ペアの 1 番目の軌道番号の出力
43	**int ***	ij2	軌道ペアの 2 番目の軌道番号の出力

局在化 MP2 計算を行う関数。core [ncore] として与えられたメモリ領域のみを用いて計算を進める。core 領域を管理するための、lmp2 専用のメモリ管理システムを持ち、それに関わる関数名は lmp2_core_* となっている。***PAICS*** のメモリ管理システムを使用しないのは、モジュールとして独立させるため。

23.7 long int lmp2_get_remain_core

引数 無し

core 領域を管理するための、LMP2 計算専用のメモリ管理システムの関数。core に残っているメモリサイズ（double 型単位）を返す。

23.8 void lmp2_core_print

引数 無し

core 領域を管理するための、LMP2 計算専用のメモリ管理システムの関数。core に割り当てられている配列を標準出力にプリントする。

23.9 void lmp2_core_alloc

引数 2

1	**const char ***	name
2	**const long int**	size

core 領域を管理するための、LMP2 計算専用のメモリ管理システムの関数。core に、配列を割り当てる。

23.10 void lmp2_core_shift_left

引数 2

1	**const long int**	pos
2	**const long int**	n

core 領域を管理するための、LMP2 計算専用のメモリ管理システムの関数。core に記録されている値を若い方にシフトする。

23.11 void lmp2_core_dealloc

引数 1

1	**const char ***	name

core 領域を管理するための、LMP2 専用のメモリ管理システムの関数。core に確保されている配列を解放する。

23.12 double * lmp2_core_get_pointer

引数 1

1	**const char ***	name

core 領域を管理するための、LMP2 計算専用のメモリ管理システムの関数。core に確保されている配列の先頭番地のポインター（double * 型）を返す。

23.13 long int lmp2_core_get_size

引数 1

1	**const char ***	name

core 領域を管理するための、LMP2 計算専用のメモリ管理システムの関数。core に確保されている配列サイズ（double 型単位）を返す。

23.14 void lmp2_make_p

引数 7

1	**const int**	ndoc
2	**const int**	nprj
3	**const int**	nbo1
4	**const double ***	c_lmo
5	**const double ***	c_prj
6	**const double ***	ovl_ao
7	**double ***	lmp2_p

P 行列（projected atomic orbital の係数行列）を作成する関数。

23.15 void lmp2_make_ovl

引数 4

1	**const int**	nbo1
2	**const double ***	ovl_ao
3	**const double ***	lmp2_p

4	**double ***	c_lmo

projected atomic orbital の重なり行列を作成する関数。

23.16 void lmp2_make_fck

引数 4

1	**const int**	nbo1
2	**const double ***	ovl_fck
3	**const double ***	lmp2_p
4	**double ***	lmp2_fck

LMP2 相関エネルギーを計算する際に使用される関数。

23.17 void lmp2_make_fck_loc

引数 5

1	**const int**	norb
2	**const int**	nbo1
3	**const double ***	c_lmp2
4	**const double ***	fck_ao
5	**double ***	lmp2_fck_loc

LMP2 相関エネルギーを計算する際に使用される関数。

23.18 void lmp2_make_basis_pop

引数 8

1	**const int**	norb
2	**const int**	nbo1
3	**const int**	nba1
4	**const int ***	nba1_idx
5	**const double ***	c_lmo
6	**const double ***	ovl_ao
7	**int ***	lmp2_basis_pop_order
8	**double ***	lmp2_basis_pop

LMP2 相関エネルギーを計算する際に使用される関数。

23.19 void lmp2_make_orbital_center

引数 7

1	**const int**	norb
2	**const int**	nbo1
3	**const double ***	c_lmo
4	**const double ***	mon_ao
5	**double ***	xc
6	**double ***	yc
7	**double ***	zc

LMP2 相関エネルギーを計算する際に使用される関数。

23.20 void lmp2_make_domain

引数 12

1	**const double**	tv
2	**const int**	norb
3	**const int**	nbo1
4	**const int**	nba1
5	**const int ***	nba1_idx
6	**const int ***	lmp2_basis_pop_order
7	**const double ***	c_lmo
8	**const double ***	ovl_ao
9	**int ***	lmp2_domain_n
10	**int ***	lmp2_domain
11	**int ***	lmp2_domain_nbo
12	**double ***	lmp2_domain_fval

LMP2 相関エネルギーを計算する際に使用される関数。

23.21 void lmp2_make_domain_ao

引数 14

1	**const double**	tv
2	**const int**	norb
3	**const int**	nbo1

4	**const int**	nba1
5	**const int***	nba1_idx
6	**const double ***	c_lmo
7	**const double ***	ovl_ao
8	**const double ***	mom_ao
9	**const double ***	xc
10	**const double ***	yc
11	**const double ***	zc
12	**int ***	lmp2_domain_ao_n
13	**int ***	lmp2_domain_ao
14	**int ***	lmp2_domain_ao_nbo

LMP2 相関エネルギーを計算する際に使用される関数。

23.22 void lmp2_make_ij_pair

引数 19

1	**const double**	tv
2	**const dobule**	tv_dim
3	**const int**	norb
4	**const int**	nba1
5	**const int ***	domain
6	**const int ***	domain_n
7	**const int ***	site
8	**const double ***	x
9	**const double ***	y
10	**const double ***	z
11	**const double ***	x_c
12	**const double ***	y_c
13	**const double ***	z_c
14	**int ***	nij
15	**double ***	ij_pair_1
16	**double ***	ij_pair_2
17	**double ***	ij_pair_lst
18	**double ***	ij_pair_dist
19	**double ***	ij_pair_dist_cen

LMP2 相関エネルギーを計算する際に使用される関数。

23.23 void lmp2_make_ij_domain

引数 13

1	**const int**	nij

2	**const int**	nba1
3	**const int ***	site
4	**const int ***	ijpair_1
5	**const int ***	ijpair_2
6	**const int ***	domain_n
7	**const int ***	domain
8	**const int ***	domain_dim_n
9	**const int ***	domain_dim
10	**const int ***	nba1_idx
11	**int ***	ij_domain_n
12	**int ***	ij_domain_nbo
13	**int ***	ij_domain

LMP2 相関エネルギーを計算する際に使用される関数。

23.24 void lmp2_make_ij_domain_ao

引数 10

1	**const int**	nij
2	**const int**	nba1
3	**const int ***	ijpair_1
4	**const int ***	ijpair_2
5	**const int ***	domain_ao_n
6	**const int ***	domain_ao
7	**const int ***	nsh1_idx
8	**int ***	ij_domain_ao_n
9	**int ***	ij_domain_ao_nbo
10	**int ***	ij_domain_ao

LMP2 相関エネルギーを計算する際に使用される関数。

23.25 void lmp2_make_ij_idx

引数 7

1	**const int**	nij
2	**const int**	ij_domain_nbo
3	**int ***	nbo_max
4	**long int ***	kmat_sum1
5	**long int ***	kmat_sum2
6	**long int ***	idx1
7	**long int ***	idx2

LMP2 相関エネルギーを計算する際に使用される関数。

23.26 void lmp2_make_ij_lst

引数 10

1	**const int**	nij
2	**const int**	nbo
3	**const int**	nba1
4	**const int ***	nba1_idx
5	**const int ***	ij_domain_n
6	**const int ***	ij_domain
7	**const int ***	ij_domain_nbo
8	**const long int ***	ij_domain_idx1
9	**int ***	ij_domain_lst1
10	**int ***	ij_domain_lst2

LMP2 相関エネルギーを計算する際に使用される関数。

23.27 void lmp2_print_input

引数 22

1	**const long int**	ncore
2	**const double**	th_1
3	**const double**	th_1_dim
4	**const double**	th_2
5	**const double**	th_2_dim
6	**const double**	th_3
7	**const double**	th_4
8	**const double**	th_l1
9	**const double**	th_l2
10	**const int**	nbo1
11	**const int**	nsh1
12	**const int**	ndoc
13	**const int**	nfzc
14	**const int**	nba1
15	**const int**	nprj
16	**const int**	max_itr
17	**const int**	lprint
18	**const int ***	nsh1_idx
19	**const int ***	nba1_idx
20	**double int ***	x
21	**double int ***	y
22	**double int ***	z

LMP2 相関エネルギーを計算する際に使用される関数。

23.28 void lmp2_make_c_max

引数 10

1	**const int**	nbo
2	**const int**	nsh
3	**const int**	norb
4	**const int ***	nsh1_idx
5	**const double ***	c
6	**double ***	max_c_ao
7	**double ***	max_c_sh

LMP2 相関エネルギーを計算する際に使用される関数。

23.29 void lmp2_make_kmat_ao_idx

引数 5

1	**const int**	ipair1
2	**const int**	ipair2
3	**const int**	nbo1
4	**const int ***	ij_domain_ao_lst1
5	**long int ***	kmat_ao_idx

LMP2 相関エネルギーを計算する際に使用される関数。

23.30 void lmp2_make_kmat

引数 30

1	**const double**	th_4
2	**const int**	nbo1
3	**const int**	nshell
4	**const int**	nij
5	**const long int**	kmat_ao_sum2
6	**const int**	lprint
7	**const MPI_Comm**	icomm
8	**const int ***	shell
9	**const double ***	shell_xyz
10	**const int ***	kab_idx

11	**const double ***	kab_sh
12	**const double ***	kab
13	**const double ***	p_x
14	**const double ***	p_y
15	**const double ***	p_z
16	**const double ***	cauchy
17	**const double ***	c_lmo
18	**const double ***	pmat
19	**const int ***	ij_pair_1
20	**const int ***	ij_pair_2
21	**const int ***	nsh1_idx
22	**const int ***	ij_domain_ao_nbo
23	**const int ***	ij_domain_nbo
24	**const long int ***	ij_domain_ao_idx1
25	**const long int ***	ij_domain_idx1
26	**const long int ***	ij_domain_idx2
27	**const int ***	ij_domain_ao_lst1
28	**const int ***	ij_domain_ao_lst2
29	**const int ***	ij_domain_lst2
30	**double ***	kmat

LMP2 相関エネルギーを計算する際に使用される関数。

23.31 void lmp2_make_ij_batch

引数 6

1	**const int**	nij
2	**const long int**	buffer
3	**const int ***	ij_domain_ao_nbo
4	**int ***	n_ij_batch
5	**int ***	ij_batch_idx
6	**long int ***	kmat_ao_buffer

LMP2 相関エネルギーを計算する際に使用される関数。

23.32 void lmp2_make_c_mat_pair

引数 10

1	**const int**	pair1
2	**const int**	pair2
3	**const int**	nsh1
4	**const int**	nbo1
5	**const int ***	ij_pair_1

6	**const int ***	ij_pari_2
7	**const int ***	nsh1_idx
8	**const double ***	c_lmo
9	**double ***	c_max_pair_sh
10	**double ***	c_max_pair_ao

LMP2 相関エネルギーを計算する際に使用される関数。

23.33 void lmp2_add_kmat_ao

引数 20

1	**const double**	th_4
2	**const int**	n_mcs
3	**const int**	n_ncs
4	**const int**	n_rcs
5	**const int**	n_scs
6	**const int**	i_mcs
7	**const int**	i_rcs
8	**const int**	i_scs
9	**const int**	i_ncs
10	**const int**	nbo1
11	**const int**	n_ij_lst
12	**const int ***	ij_lst
13	**const int ***	ij_pair_1
14	**const int ***	ij_pair_2
15	**const long int ***	kmat_ao_idx
16	**const double ***	c_max_pair_ao
17	**const double ***	c_lmo
18	**const double ***	shint
19	**double ***	c_rs
20	**double ***	kmat_ao

LMP2 相関エネルギーを計算する際に使用される関数。

23.34 void lmp2_make_tmat

引数 23

1	**const int**	norb
2	**const int**	nbo1
3	**const int**	nij
4	**const int**	nbo_max
5	**const int**	max_itr
6	**const double**	th_lin1

7	**const double**	th_lin2
8	**const long int**	kmat_sum2
9	**const int**	lprint
11	**const int ***	ij_pair_1
11	**const int ***	ij_pair_2
12	**const int ***	ij_pair_lst
13	**const int ***	ij_domain_nbo
14	**const long int ***	ij_domain_idx1
15	**const long int ***	ij_domain_idx2
16	**const int ***	ij_domain_lst1
17	**const int ***	ij_domain_lst2
18	**const double ***	s
19	**const double ***	f
20	**const double ***	f_loc
21	**const double ***	kmat
22	**double ***	e_total
23	**double ***	e_ij

LMP2 相関エネルギーを計算する際に使用される関数。

23.35 void lmp2_make_umat_ij

引数 16

1	**const int**	ipair
2	**const int**	nbo1
3	**int ***	nu
4	**const int ***	ij_domain_nbo
5	**const long int ***	ij_domain_idx1
6	**const int ***	ij_domain_lst1
7	**const double ***	s
8	**const double ***	f
9	**double ***	fmat_ij
10	**double ***	smat_ij
11	**double ***	xmat_ij
12	**double ***	wmat_ij
13	**double ***	umat_ij
14	**double ***	diat_ij
15	**double ***	buf1
16	**double ***	buf2

LMP2 相関エネルギーを計算する際に使用される関数。

23.36 void lmp2_make_rmat_ij

引数 21

1	**const int**	ipair
2	**const int**	norb
3	**const int**	nbo1
4	**double ***	max_rmat
5	**const int ***	ij_pair_1
6	**const int ***	ij_pair_2
7	**const int ***	ij_pari_lst
8	**const int ***	ij_domain_nbo
9	**const long int ***	ij_domain_idx1
11	**const long int ***	ij_domain_idx2
11	**const int ***	ij_domain_lst2
12	**const double ***	s
13	**const double ***	f
14	**cnost double ***	f_loc
15	**const double ***	kmat_ij
16	**const double ***	tmat_ij
17	**const double ***	tmat
18	**double ***	fmat_ij
19	**double ***	smat_ij
20	**double ***	ftft
21	**double ***	rmat

LMP2 相関エネルギーを計算する際に使用される関数。

23.37 void lmp2_make_tmat_ij

引数 15

1	**const int**	ipair
2	**const int**	norb
3	**double ***	max_delta_t
4	**const int ***	ij_pair_1
5	**const int ***	ij_pair_2
6	**const int ***	umat_nbo
7	**const int ***	ij_domain_nbo
8	**const double ***	rmat_ij
9	**const double ***	umat_ij
10	**const double ***	diag_ij
11	**const double ***	f_loc
12	**double ***	trans_r_ij
13	**double ***	trans_delta_t_ij
14	**double ***	delta_t_ij
15	**double ***	tmat_ij

LMP2 相関エネルギーを計算する際に使用される関数。

23.38 void lmp2_make_pair_energy

引数 9

1	**const int**	nij
2	**const int ***	ij_pair_1
3	**const int ***	ij_pair_2
4	**const int ***	ij_domain_nbo
5	**const long int ***	ij_domain_idx2
6	**const double ***	kmat
7	**const double ***	tmat
8	**double ***	e_total
9	**double ***	e_ij

LMP2 相関エネルギーを計算する際に使用される関数。

23.39 void lmp2_set_output

引数 6

1	**const int**	nij
2	**const int ***	ij_pair_1
3	**const int ***	ij_pair_2
4	**int ***	n_ij_out
5	**int ***	ij_pair_1_out
6	**int ***	ij_pair_2_out

LMP2 相関エネルギーを計算する際に使用される関数。

23.40 void lmp2_allreduce_double

引数 5

1	**const long int**	mpi_val_size
2	**const long int**	mpi_buf_size
3	**double ***	mpi_val
4	**double ***	mpi_buf
5	**MPI_Comm**	icomm

LMP2 相関エネルギーを計算する際に使用される関数。

23.41 void print_domain_information

引数 13

1	**const int**	norb
2	**const int**	nbo1
3	**const double**	th_1
4	**const double**	th_1_dim
5	**const double**	th_3
6	**const int ***	domain_n
7	**const int ***	domain_nbo
8	**const int ***	domain_dim_n
9	**const int ***	domain_dim_nbo
10	**const int ***	domain_ao_n
11	**const int ***	domain_ao_nbo
12	**const int**	lprint

LMP2 相関エネルギーを計算する際に使用される関数。

23.42 void print_ij_domain_information

引数 12

1	**const int**	nij
2	**const int**	norb1
3	**const int ***	ij_pair_1
4	**const int ***	ij_psir_2
5	**const double ***	ij_pair_dist
6	**const double ***	ij_pair_dist_cen
7	**const int ***	site_lmo
8	**const int ***	ij_domain_n
9	**const int ***	ij_domain_nbo
10	**const int ***	ij_domain_ao_n
11	**const int ***	ij_domain_ao_nbo
12	**const int**	lprint

LMP2 相関エネルギーを計算する際に使用される関数。

23.43 void lmp2_dimer_write_orbital_file

引数 16

1	**const int**	ifrag
2	**const int**	jfrag

3	**const int**	nij
4	**const int**	norb
5	**const int**	nba1
6	**const int**	nsh1
7	**const int**	nbo1
8	**const int**	nfzc
9	**const int ***	ij_pair_1
10	**const int ***	ij_pair_2
11	**const int ***	basis
12	**const int ***	shell
13	**const double ***	shell_xyz
14	**const double ***	ovl
15	**const double ***	c_lmo
16	**const double ***	e_ij

LMP2 計算で使われた局在化軌道を出力する関数。