VisBAR Wave Batch

v2.0.1 マニュアル

鳥取大学大学院 工学研究科機械宇宙工学専攻

応用数理工学コース 物理計算工学研究グループ

平成28 年8 月3 日

目 次

図 目 次

第1章　「VisBAR Wave Batch」について」

1.1　概要

VisBAR Wave Batch は、プログラミング言語 Python[1] を用いて開発された、原子構造・波動関数可視化ツールである。Input ファイルに GaussianCube 形式のファイル (以下 Cube ファイル) を用いており、この形式のファイルであればどの計算コードで出力された Cube ファイルでも可視化が可能である。また、複数の Cube ファイルを連続で可視化することが可能である。

以下、付属の README ファイルの内容を付す。

```
VisBAR_wave_batch[*] は、Python による、電子波動関数の可視化ツールです。

VisBAR_wave _batch は、「VisBAR」パッケージ (VisBAR=Visualization tool
with Ball, Arrow , Rod)[*][1] の一部です。
もし VisBAR_wave _batch を出版で使う場合は、論文 [1] を引用してください。

詳細は、日本語版マニュアルに記されています。
英語版マニュアルは準備中です。

[*] http://www.damp.tottori-u.ac.jp/~hoshi/visbar/

[1] Takeo Hoshi, Yohei Akiyama, Tatsunori Tanaka and Takahisa Ohno,
"Ten-million-atom electronic structure calculations on the K computer
with a massively parallel order-N theory",
J. Phys. Soc. Jpn. 82, 023710, 4pp (2013).
http://dx.doi.org/10.7566/JPSJ.82.023710
```

1.2　本マニュアルの構成

本マニュアルは、
第 1 章に概要、パッケージの中身
第 2 章にクイックスタート
第 3 章にソースコードの説明

付録にソースコード原文と、Visualization Tool Kit (VTK)[2] のビルド、開発履歴

という構成になっている。実際に使用していただくに当たって、第 2 章を順番に読み、実行してもらうと、機能を使えるようにしている。

1.3 パッケージファイルの構成

以下は本パッケージファイルの内容を説明である。以下に書いてある 10 つのファイルがパッケージに同梱されている。パッケージをダウンロードしたならば、まず、ファイルが 10 つそろっていることを確認してもらいたい。ちなみに「.py」という拡張子が付いたファイルは Python のスクリプトファイルである。

(1) VisBAR_wave_batch.py
VisBAR Wave Batch のメインプログラム (小規模処理用)

(2) VisBAR_wave_batch_prosess.py
VisBAR Wave Batch のメインプログラム (大規模処理用)(Windows 上では使用不可)

(3) formatconvert.py
Cube ファイルを VTK の形式に変換するプログラム

(4) visualize_isosurface.py 画面に描画を行うプログラム

(5) library.py 原子情報のライブラリ

(6) bond_length.txt 結合の種類・距離を設定しているテキストファイル

(7) visbar_wb_setting_default.txt 可視化のプログラムの変数を指定するテキストファイル

(8) Cube ファイルサンプルとして複数の Cube ファイル (拡張子が.cube となっているもの) を Input フォルダに同梱

(9) 00README_FIRST_JP.txt README テキスト (日本語)。本プログラムの概要と、引用する際の注意事項について

(10) 00README_FIRST_EN.txt README テキスト (英語)。本プログラムの概要と、引用する際の注意事項について

第2章　クイックスタート

本章では、必要なソフトウェアのインストールと、サンプルデータの Cube ファイル (ベンゼン分子) を例として描画を行う。VTK のインストールには、vtkpython(単体インストーラー) を利用する方法 (2.1) と、そうでない方法 (付録 A) がある。 インストールの手軽さ・公式 からの配布という点から考えて、前者を推奨する。

本マニュアルでは特に断らない限り、Python v.2.7、VTK v.6 を仮定している

3 次元的に可視化するにあたって、以下の URL を参考に、動作確認用のサンプルコードを作成した。パッケージの中の sample ディレクトリに置いてある。
`http://www.vtk.org/Wiki/VTK/Examples/Python/GeometricObjects/Display/Sphere`

2.1　vtkpythonのインストール方法

　VTK を Python で実行するためにインストールを行う。
Kitware 社が開発している VTK のバージョン 6.1.0 から Standalone Python Interface (Installer) が提供された。
vtkpython インストーラを使えば、Python がインストールされていなくても、VisBAR Wave Batch が使用できる。

　ここでは、「vtkpython-6.1.0-Windows-32bit.exe」を使用した VisBAR Wave Batch の起動方法を載せておく。
　VTK のインストーラを以下の URL からダウンロードする。
`http://www.vtk.org/VTK/resources/software.html`

　今回使用したのは、「vtkpython-6.1.0-Windows-32bit.exe」である。環境に合わせて選んでいただきたい。
　インストーラを起動
「VTK-6.1.0 セットアップウィザードへようこそ」⇒次へ
図 2.1 を参照

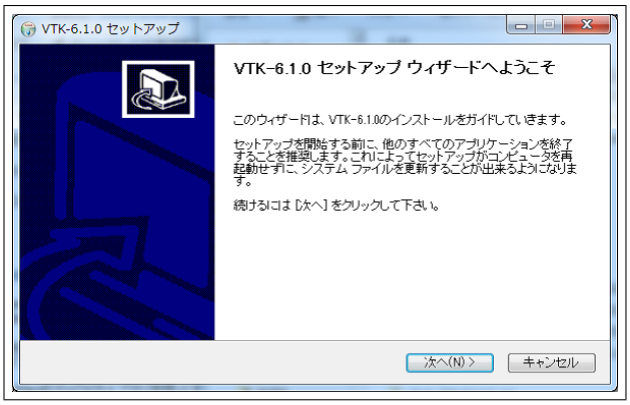

図 2.1: セットアップウィザード

　「ライセンス契約書」⇒同意する
図 2.2 を参照

　「インストール先を選んでください」⇒次へ
デフォルトならば「C:\Program Files\VTK 6.1.0」が選ばれている
図 2.3 を参照

　「スタートメニュー　フォルダを選んでください」
⇒ショートカットを作成しないにチェック

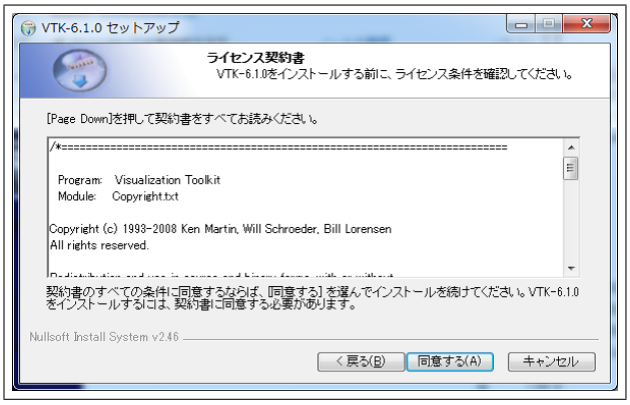

図 2.2: ライセンス契約書

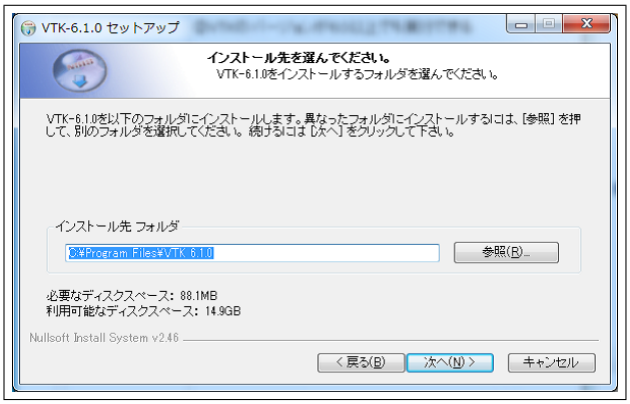

図 2.3: インストール先選択

⇒次へ
図 2.4

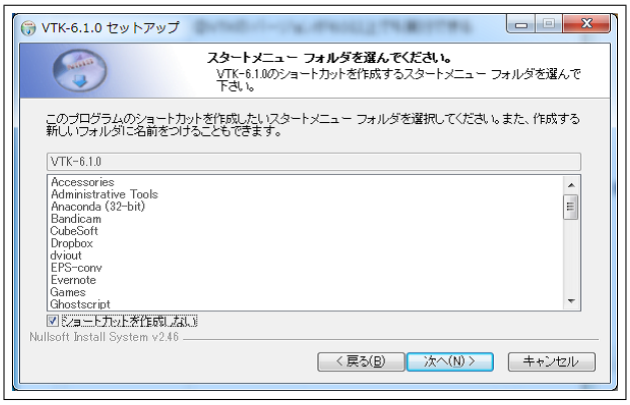

図 2.4: スタートメニュー選択

コンポーネントを選んでください
⇒ VTK にチェックが付いていることを確認
⇒インストールを選択し、インストールが始まる
図 2.5 特に再起動は要求されないが、再起動することを個人的にお勧めする。

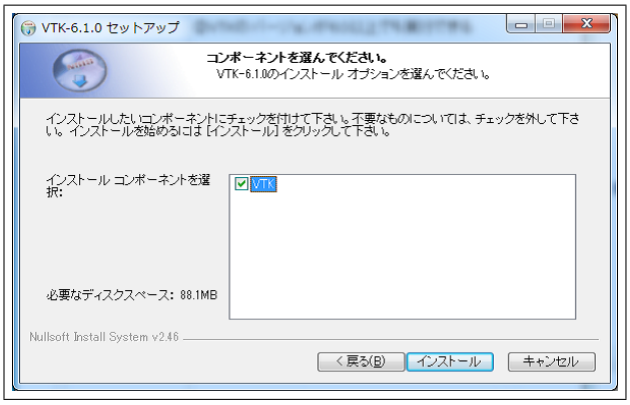

図 2.5: インストールを実行

インストール作業が終わると、デフォルト通りならば「C:\Program Files\VTK 6.1.0\bin」の場所に vtkpython.exe がある。ダブルクリックをして起動確認をしておくとよい。
コマンドラインが起動し、以下の文字が現れる。

```
--------------------------------------------------
vtk version 6.1.0
Python 2.7.3 (default, Jan 14 2014, 23:11:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>>
--------------------------------------------------
```

終了するときは
`>>> exit()`
と、打てば vtkpython から抜け出せる。

次に環境変数を設定する。

スタート⇒コンピュータを右クリック⇒プロパティ⇒システムの詳細設定⇒環境変数を選択

ユーザー環境変数の PATH を編集し、最後に「C:\Program Files\VTK 6.1.0\bin」を付け足す。

前に他の PATH がある場合は区切り文字に「;」を使用すること。
「～～～～;C:\Program Files\VTK 6.1.0\bin」となる。

＜確認方法＞コマンドプロンプトを起動し、「vtkpython」と打ち込む。
先ほど、vtkpython.exe をダブルクリックしたときと同じように実行画面が表示されればよい。

＜＜実行方法　（2 通り説明しておく）＞＞
1. コマンドプロンプトからの実行
2. ダブルクリックで実行する方法

※ vtkpython に IDLE.py は存在するが、正規の Python を優先して実行してしまうため、
正規の Python に VTK がないとエラーが出る。

インストールが終了したら、sample のディレクトリに http://www.vtk.org/Wiki/VTK/Examples/Python/GeometricObjects/Display/Sphere に書いてあるソースコード「Sphere.py」を用意したので、実行する。
sample フォルダの中にある「Sphere.py」を実行し、図 3.1 の画面にある白球が表示されればインストールは成功である。

＜1. コマンドプロンプトからサンプルを実行する方法＞
コマンドプロンプトを実行する。

サンプルのあるディレクトリまで移動する。
`> cd ~/VisBAR_wave_batch_package_v2.*.*_****/sample`

Sphere を実行する。
> vtkpython Sphere.py

＜2. ダブルクリックでサンプルを実行する方法＞
エクスプローラで、VisBAR Wave Batch の中にある sample のディレクトリを開く。

Sphere.py のプログラムを右クリックして、
プロパティ⇒変更⇒参照⇒
⇒ C:\Program Files\VTK 6.1.0\bin\vtkpython.exe を選択し開く⇒ OK

再度 sample のディレクトリから sphere.py をダブルクリックして起動すればよい。

2.2 VisBAR Wave Batch の実行

まず、VisBAR Wave Batch を使うにあたって、まず Cube ファイルを可視化して画像として出力する方法を解説する。

[実行手順] ＜コマンドプロンプトから **VisBAR Wave Batch** を実行する方法＞

コマンドプロンプトを実行する。

VisBAR_Wave Batch のあるディレクトリまで移動する。

```
> cd */VisBAR_wave_batch_package_v2.*.*_****
```

VisBAR Wave Batch を実行する。実行の際、Cube ファイルのある Path を指定する。

```
> vtkpython VisBAR_wave_batch.py ./Input
```

VisBAR Wave Batch の実行時にオプションをつけることが出来る。
[-b] … バッチモードとなり連続で画像化を行う
[-s] … Sign Correction モードとして実行する
[-p] … 周期境界条件を適用する
[-o [ディレクトリ名]] … 画像を出力するディレクトリを指定する。
[-parallel] … 並列モードとして実行する

全てのオプションをつけて実行した場合、以下のコマンドになる

```
> vtkpython VisBAR_wave_batch.py ./Input  -b -s -p -o ./Output -parallel
```

ヘルプを表示する場合は [-h] をつけて実行する。

```
> vtkpython VisBAR_wave_batch.py -h
```

以下に実行したサンプルの結果を載せる。

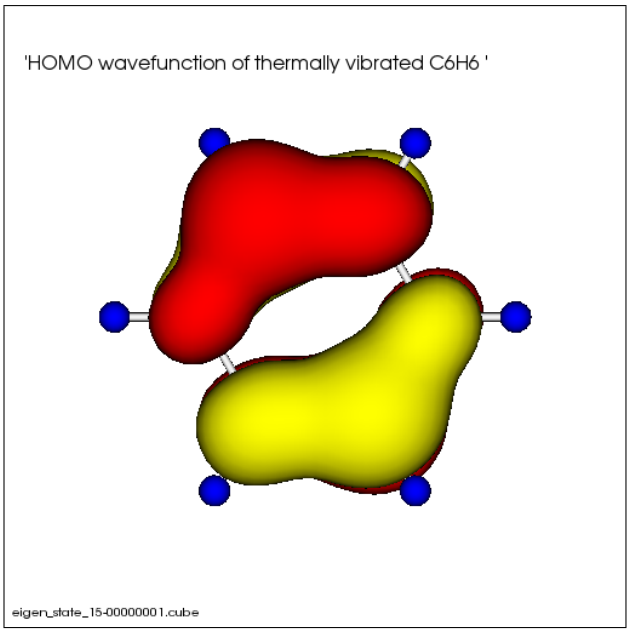

図 2.6: サンプル Cube ファイル実行結果

2.3 連続で Cube ファイルを PNG ファイルにする場合

このプログラムは、複数の Cube ファイルを連続で画像にすることが可能である。
VisBAR Wave Batch 実行の際に、[-b] のオプションを付け、Cube ファイルのあるディレクトリに複数 Cube ファイルを置けば実行が可能となる。

2.4 「sign correction」機能

符号の連続性を担保するために、ファイル入力された波動関数データの符号を反転 (ϕ (r) → − ϕ (r)) させる機能である。これを「sign correction(符号自由度処理)」機能と呼ぶ。具体例を図 2.7 に載せる。符号自由度処理なしの間の一枚の画像が正負反転しているのが分かる。
実行時にオプション [-s] をつけて適用する

解説 : 固有値問題の解としての波動関数には、符号の任意性がある。波動関数ファイルを連続的に可視化場合、前のファイルと符号が逆転している可能性があり、動画としたさいに物理的意味のない符号逆転 (sign flipping) を繰り返すことがある。そのような場合に、この correction を ON にすることで、前の波動関数との符号の整合性が保たれる。内部的には、前の波動関数との内積をとり、内積の値が負であった場合、現在の波動関数の符号を反転させている。

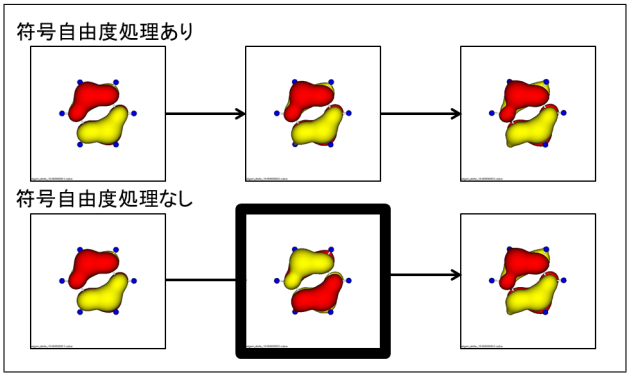

図 2.7: 符号自由度処理具体例

2.5 周期境界条件適用方法

周期系などで、原子の位置座標が可視化の枠の外に出てしまっている場合がある。
その場合は、実行時のオプションに [-p] をつけて、周期境界条件を適用すれば良い。詳しくは 3.2.2 に載せている。

行っていることは、Cube ファイルの波動関数可視化範囲の枠から出た原子座標に対して枠一辺の長さを引き、原子座標を枠の中に戻している。

つまり、一辺長さ L の可視化枠に対して、原子座標 A(x,y,z) の原子 A があるとする。x,y 方向に、原子座標が可視化範囲枠から出ている場合、周期境界条件を適用して、A(x-L, y-L, z) と、このように原子座標をずらしている。
ただし、直鎖系の分子などに対して、周期境界条件を適用してしまうと、はみ出した部分が折り返されて可視化範囲枠内に描画されてしまうので、注意が必要である。

2.6 並列処理方法

複数の Cube ファイルを連続でバッチ処理を行うときに並列で画像にすることが可能である。ただし、「sign correction」を使用する場合は使用できない。
VisBAR Wave Batch 実行の際に、[-b] と [-parallel] のオプションを付け、Cube ファイルのあるディレクトリに複数 Cube ファイルを置けば実行が可能となる。
注)Windows の場合は VisBAR_wave_batch.py の方を使用すること。
並列数を変更する場合は環境変数の OMP_NUM_THREADS の値を変更すると並列数が変わる。

環境変数の OMP_NUM_THREADS を変更する方法

```
linux の場合
$ export OMP_NUM_THREADS=2
Windows の場合
$ set OMP_NUM_THREADS=2
```

2.7 画面の操作方法

ここでは、一つの Cube ファイルを可視化したときに自分で操作して、見たい位置や、見たい波動関数の値を操作する方法を説明する。この方法でもウィンドウを閉じる際に画像は出力される。実行の際は [-b] オプションをつけない。

Visualization Toolkit に表示される画面の操作方法について

まず、画面をクリックしてみるとポインタのある方向に画面が回転する。画面中央にある文字は拡大縮小・移動をすることが出来る。他にも、VTK に備わったオプション (キーボード操作) を使用することで、画面の描画を操作することが出来る。ただし、キーボードでの操作を行う場合、VTK の画面が選択 (ウィンドウをクリック) されている状態でキーボード操作を行わないと反応を示さない。

【画面の操作方法】

入力	画面に反映される内容
「n」	次ステップの画面を表示
「shift + q」	「VisBAR Wave Batch」の終了
「j」＋左クリック	joystick モード (ポインタの位置によって画面が回転する)
「t」＋左クリック	trackball モード (画面をドラッグすることで画面が回転する)
「w」	ワイヤーフレーム表示 (元に戻したいときは「s」)
「s」	サーフェイス表示 (デフォルト設定)
「3」	ステレオ表示 (現在はエラーがでる)
「r」	表示位置リセット
「u」	表示画像を png 形式にて保存される (ファイルは「image*.png」と出力される)
「a」	マウスで選択したモデルを回転・拡大縮小・スライド移動できる (a+左 or 右クリック、a+マウスホイール)
「マウスホイール」	拡大・縮小ができる。ホイールを押し込むとスライド移動を行う
「右クリック」	画面上部で拡大、画面下部で縮小を行う
「c」	設定ファイルの出力。詳しくは 2.8 に記載。
[x]	X 軸を正面から見た位置にカメラを設定
[y]	Y 軸を正面から見た位置にカメラを設定
[z]	Z 軸を正面から見た位置にカメラを設定

【波動関数の等値面の値を増減する操作方法】

波動関数の値は、波動関数の初期値*(1.01)**level である。指定したキーボードを押すと level が変化し、波動関数が増減する。

キーボード	波動関数の増減
「1」	PositiveLevel と NegativeLevel の値が 1 増加し、正と負の波動関数が変化する
「2」	PositiveLevel と NegativeLevel の値が 1 減少し、正と負の波動関数が変化する
「4」	PositiveLevel の値が 1 増加し、正の波動関数が変化する
「5」	PositiveLevel の値が 1 減少し、正の波動関数が変化する
「6」	NegativeLevel の値が 1 増加し、負の波動関数が変化する
「7」	NegativeLevel の値が 1 減少し、負の波動関数が変化する
「0」	デフォルトで設定した PositiveLevel と NegativeLevel の値にリセットする

波動関数を増減するキーボードのボタンを長押しすると、連続で波動関数が変化し動画的に波動関数を見ることが出来る。また、変化させた波動関数の値は端末画面 (コマンドプロンプト) に表示される。

2.8　描画の設定を保存する

設定ファイルのデフォルトとして、「visbar_wb_setting_default.txt」というファイルがあるが、基本的にこのファイルは編集しない。設定を編集したい場合は、実行時「c」を押したときに作成される「visbar_wb_setting_output.txt」の名前を「visbar_wb_setting.txt」に編集することで設定ファイルの設定反映が可能となる。

【設定の編集方法】

1. まず、 VisBAR_wave_batch.py を起動し、インタラクティブに操作できるようにする。そして描画後、「c」を押して設定ファイル「visbar_wb_setting_output.txt」を出力する。この時のカメラの視点と、波動関数の値が設定ファイルに反映される。
2. 「visbar_wb_setting_output.txt」に可視化した状態の設定が書き込まれる。この時、カメラの視点と波動関数の値はキーボードの「c」を押した瞬間の状態が自動的に書き込まれる。
3. そして、一度プログラムを閉じる。
4. 「visbar_wb_setting_output.txt」のファイル名を「visbar_wb_setting.txt」に変える。
5. 「visbar_wb_setting.txt」の中身を編集する。具体的な編集方法は下記。
6. 再度、VisBAR_wave_batch.py を起動すると、設定を反映した実行結果になる。

【設定ファイル編集のルール】

・編集するファイルは「visbar_wb_setting.txt」
・文字と数字、数字と数字の間はスペースで区切る
・キーワードは編集しない。数字のみ編集する
・＃を行頭につけるとその行をコメントアウトできる。
・間に空白があってもいい。
・設定ファイルは 1 行だけでも可、ただし、最後に--end_setting--の一文が必要

【設定ファイルの中身】

visbar_wb_setting_default.txt

```
#---IsoValue---#
IsoValuePositive 0.02
IsoValueNegative -0.02
IsoOpacityPositive 1.0
IsoOpacityNegative 1.0
DrawIsoInWirePositive Off
```

```
DrawIsoInWireNegative Off

#---Color---#
IsoColorPositive 1.0 0.0 0.0
IsoColorNegative 1.0 1.0 0.0
OutLineColor 0.0 0.0 1.0
BackGroundColor 1.0 1.0 1.0

#---WindowSize---#
WindowSize 500 500

#---Camera---#
FocalPoint 0.0 -0.0382356981252 0.0
CameraPosition 0.0 -0.0382356981252 45.0
ParallelScale 10.4748065371
ViewUp 0.0 1.0 0.0
ClippingRange 0.01 1000.01

#---Label---#
#NumberOnly:1
#AtomOnly:2,
#EachAtomNumber:3
#SerialAtomNumber:4
LabelType 4


#---DrawObject---#
DrawBond On
DrawAtom On
DrawWaveFunction On
DrawOutLine On
DrawAxis On
DrawText On
DrawAtomLabel On
WatchParallelView On
--end_setting--
```

【設定ファイルの詳しい内容】

IsoValuePositive	…	正の波動関数を表示するときの等値面の値 (数値で設定)
IsoValueNegative	…	負の波動関数を表示するときの等値面の値 (数値で設定)
IsoOpacityPositive	…	正の波動関数の透過度 (0.0〜1.0)
IsoOpacityNegative	…	負の波動関数の透過度 (0.0〜1.0)
DrawIsoInWirePositive	…	正の波動関数を mesh 表示 (On/Off)
DrawIsoInWireNegative	…	負の波動関数を mesh 表示 (On/Off)
IsoColorPositive	…	正の波動関数の色 (R,G,B)=(0.0〜1.0 , 0.0〜1.0 , 0.0〜1.0)
IsoColorNegative	…	負の波動関数の色 (R,G,B)=(0.0〜1.0 , 0.0〜1.0 , 0.0〜1.0)
OutLineColor	…	外枠の色 (RGB 形式)　(R,G,B)=(0.0〜1.0 , 0.0〜1.0 , 0.0〜1.0)
BackGroundColor	…	背景の色 (RGB 形式) (R,G,B)=(0.0〜1.0 , 0.0〜1.0 , 0.0〜1.0)
Window_size	…	表示するウィンドウのサイズ (縦, 横)
FocalPoint	…	視点の焦点の値 (数値で設定)
CameraPosition	…	視点の位置 (数値で設定)
ParallelScale	…	視点の縮尺 (数値で設定)
ViewUp	…	現在調査中 (数値で設定)
LabelType	…	原子に貼り付けるラベル。1:数字のみ (通し番号)、2:元素記号のみ 3:原子ごとの数字+元素記号、4:通し番号+元素記号
ClippingRange	…	現在調査中 (数値で設定)
DrawBond	…	結合を描くかどうか (On/Off)
DrawAtom	…	原子球を描くかどうか (On/Off)
DrawWaveFunction	…	波動関数を描くかどうか (On/Off)
DrawOutLine	…	外枠を描くかどうか (On/Off)
DrawAxis	…	軸を描くかどうか (On/Off)
DrawText	…	画面中央上部にテキストを表示するか (On/Off)
DrawAtomLabe	…	画面中央上部にテキストを表示するか (On/Off)
WatchParallelView	…	parerellview を設定するか (OnOff)

2.9　結合の範囲を指定する

結合を引く箇所を定めている「bond_length.txt」ファイルについて説明する。

【結合リストファイル編集のルール】
・ファイル名は「bond_length.txt」とする。
・文字と数字、数字と数字の間はスペースで区切る。
・原子記号二つと結合距離はペアで一行に書かなくてはならない。
・原子の種類と結合距離の位置は入れ替えてはいけない。
・設定ファイルは 1 行だけでも可、ただし、最後に--end_setting--の一文が必要。
・文字、空白は全て半角

ファイルの中身は以下のとおり
bond_length.txt

```
C C 3.0
C H 2.2
H C 2.2
H H 1.0
--end_setting--
```

これはC_6H_6のベンゼンを対象としているので、各原子の組み合わせは C-C、C-H、H-C、H-H の 4 種類の組み合わせがあることになる。なので、ファイルには、一つ目の原子、二つ目の原子、結合距離 (単位は a.u.) として書かれている。結合距離より、原子同士の距離が近いとき、結合が引かれる。注意して欲しいのは、違う種類の元素と結合を引く場合、C-H、H-C の二種類を書き込み、結合距離を同じくしておくことが必要である。C と H 以外の原子がある Cube ファイルを対象とする場合、このファイルに結合を引きたい原子のペアと結合距離を書き込むと結合を描くことが出来る。結合リストファイルに書かれている順番はプログラムの実行に関係ない。

2.10　実行時に出力されるファイル

その他、実行時に出力されるファイルについて
v2.0.0 では、可視化終了後に自動で削除されるようになっている。

削除を行わないようにする方法
visualize_isosurface.py の 758 行目あたりの

```
    shutil.rmtree(input_filenametext[:-5])
```

を以下のようにコメントアウトする。

```
    #shutil.rmtree(input_filenametext[:-5])
```

以下のファイルは、全て「入力ファイルと同じディレクトリ」の「入力ファイルと同じ名前」のフォルダに保存されている。

convert_wave_function.vtk.txt

⇒ Cube ファイルから波動関数を抜き出したファイル。改行の位置が変化している。詳しくは 3.3.4 を参照。

make_atom.xyz.txt

⇒ Cube ファイルから原子座標・原子記号を抜き出し、XYZ 形式に直したファイル。詳しくは 3.3.2 を参照。

VTK_out.vtk

⇒ Cube ファイルから VTK ファイルに変換したもの。

2.11 mpi4pyを使用したVisBAR Wave Batchの実行

実行コマンドは、以下のように普通に使用するときのコマンドの前に「mpirun -n 4」などのmpiの実行コマンドをつけるだけである。
(プログラムの中としては可視化するファイルをノードごとに担当するものを決めてあげて、それぞれで普通のVisBAR Wave Batchを実行している。)

```
$ mpirun -n 4 python VisBAR_wave_batch.py -b -parallel ./Input
```

注)Sign Correctionモードは使用しないこと。

第3章　基盤技術

本章では、VisBAR Wave Batch の実行に必要な基盤技術を解説する。

3.1　基本

本節ではプログラム言語「Python」と 3D 可視化ライブラリ「VTK」の説明を行う。

3.1.1　Python

Python は汎用の高水準言語である。特徴としては文字処理に優れ、クロスプラットフォームであり、可読性が高いということがあげられる。これらの性質は特に統合シミュレーション環境構築において重要な要素であるといえる (ここでの可読性とは、特にコードの読みやすさと整合性の事を指す)。また「Python」はスクリプト言語であるため、コンパイルの必要が無く、個々のモジュールを容易にテストでき、利便性が高い。さらに、ガーベージコレクション (プログラムが動的に確保したメモリ領域のうち、不要になった領域を自動的に解放する機能) を持っているため、連続使用において PC 自体が重くなることを防ぐ。利便性の高い大規模な標準ライブラリを備えている。
例として、以下にサンプルコードを記す。

```
f = open("sample","r")
line = f.readline()
while line:
    line = f.readline()
    if line.find("check") >= 0:
        print line
        break
f.cloce()
```

上記のサンプルコードは、ファイル「sample」を一行ずつ読み込み、文字列「check」を検索し、表示するというプログラムである。Python にはファイルを読み込むためのメソッドがいくつかあり、使用目的に応じて使い分けることができる。このコードを見るとわかるように、「python」は、条件文や関数を記述する際は必ずインデントを用いて区別する。さらに特に変

数について型を指定する必要が無いため (値を入れた際に、入れた値の型として勝手に変数を定義してくれる)、大量の変数を使用する場合も見づらくなることは少ない。

3.1.2 VTK

3 次元的に可視化するにあたって、VTK の Python 用 Wrapper を用いた。

パッケージの中の sample ディレクトリに置いてある「Sphere.py」について説明する。このプログラムは以下の URL に載っているものを使用した。

http://www.vtk.org/Wiki/VTK/Examples/Python/GeometricObjects/Display/Sphere

sphere.py

```
import vtk

# create a rendering window and renderer
ren = vtk.vtkRenderer()
renWin = vtk.vtkRenderWindow()
renWin.AddRenderer(ren)

# create a renderwindowinteractor
iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)

# create source
source = vtk.vtkSphereSource()
source.SetCenter(0,0,0)
source.SetRadius(5.0)

# mapper
mapper = vtk.vtkPolyDataMapper()
if vtk.VTK_MAJOR_VERSION <= 5:
    mapper.SetInput(source.GetOutput())
else:
    mapper.SetInputConnection(source.GetOutputPort())

# actor
actor = vtk.vtkActor()
actor.SetMapper(mapper)

# assign actor to the renderer
ren.AddActor(actor)

# enable user interface interactor
iren.Initialize()
renWin.Render()
iren.Start()
```

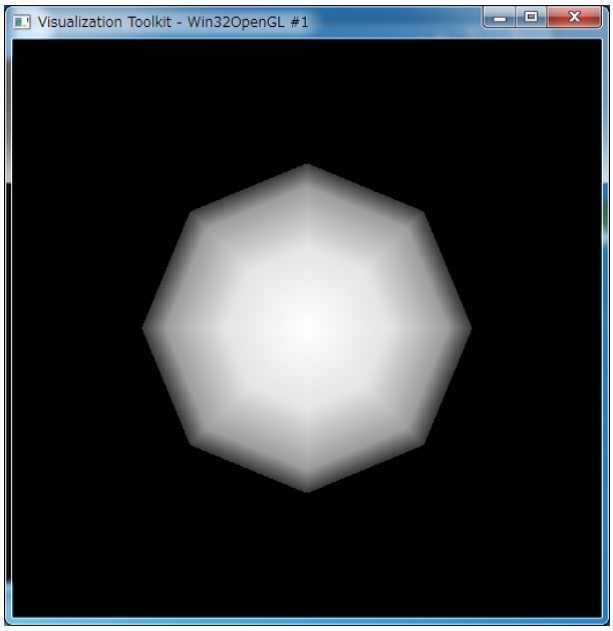

図 3.1: VTK サンプルコード実行結果

解説

[import vtk]
1 行目、vtk モジュールを使うことを宣言する

[ren = vtk.vtkRenderer()]
2 行目、vtkRenderer を ren として定義する

[renWin = vtk.vtkRenderWindow()]
3 行目、vtkRenderWindow を renWin として定義する

[renWin.AddRenderer(ren)]
4 行目、renWin に ren を追加する

[iren = vtk.vtkRenderWindowInteractor()]
5 行目、vtkRenderWindowInteractor を iren として定義する

[iren.SetRenderWindow(renWin)]
6 行目、iren に renWin を設定する

[source = vtk.vtkSphereSource()]
7 行目、vtk に備わっている Sphere のデータを sorce として用いる

[source.SetCenter(0,0,0)]
8 行目、sorce に入れた Sphere の中心点を (x,y,z)=(0,0,0) として設定する

[source.SetRadius(5.0)]
9 行目、sorce に入れた Sphere の半径を 5.0 として設定する

[mapper = vtk.vtkPolyDataMapper()]
10 行目、vtkPolyDataMapper を mapper として定義する

[if vtk.VTK_MAJOR_VERSION <= 5:]
11 行目、VTK のバージョンが 5 以下ならば if 分岐に入る

[mapper.SetInput(source.GetOutput())]
12 行目、7〜9 行目の Sphere のデータを mapper にセットする

[else:]
13 行目、11 行目の if 分岐に入らなかった場合

[mapper.SetInputConnection(source.GetOutputPort())]
14 行目、7〜9 行目の Sphere のデータを mapper にセットする。12 行目と 14 行目はバージョン 5.*と 6.*の違いによる書き方の違いである。

[actor = vtk.vtkActor()]
15 行目、vtkActor を actor として定義する

[actor.SetMapper(mapper)]
16 行目、10〜14 行目の mapper のデータを actor に設定する

[ren.AddActor(actor)]
17 行目、actor を ren に追加する

[iren.Initialize()]
18 行目、iren を初期化する

[renWin.Render()]
19 行目、renWin を設定する

[iren.Start()]
20 行目、iren を実行する

[各用語解説]

sorce ‥ 基本的な形状の Dataset を読み込む or データを作り出す。
(この場合は、もともと VTK にある sphere[球] のデータを呼び出している。)

filter ‥ sorce からのデータを加工する。

mapper ‥ sorce や filter で設定した内容を描画オブジェクト (actor) へ渡す。

actor ‥ 描画オブジェクト。色などオブジェクトのコントロールも行う。

renderer ‥ 描画エンジン。描画する actor をセットする必要がある。

renderWindow ‥ actor のセットされた renderer をシステム上で表示する

iren(renderwindowinteractor) ‥ VTK をシステム上で操作し、実行する。

3.2　プログラムの原理

使用するプログラムは、メインプログラム「VisBAR_wave_batch.py」、フォーマット変換プログラム「formatconvert.py」、可視化プログラム「visualize_isosurface.py」、原子情報ライブラリ「library.py」の 4 つである。フォーマット変換プログラムにて、Cube ファイルから波動関数を表した VTK ファイルと、原子座標を表した xyz ファイルを作り、可視化プログラムで VTK ファイル (波動関数を可視化)、xyz ファイル (原子・結合) を可視化条件設定ファイルで設定し描画している。

下記にプログラムの流れ図を示す。

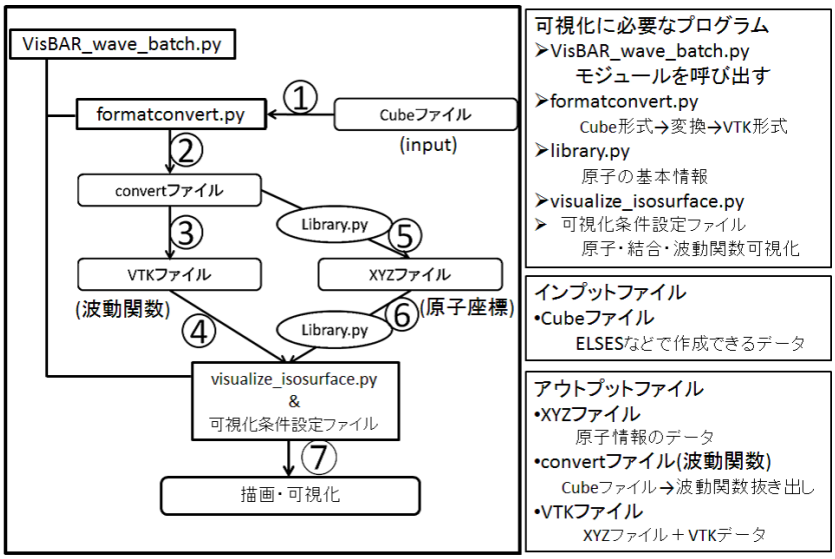

図 3.2: プログラム流れ図 (インタラクティブ可視化)

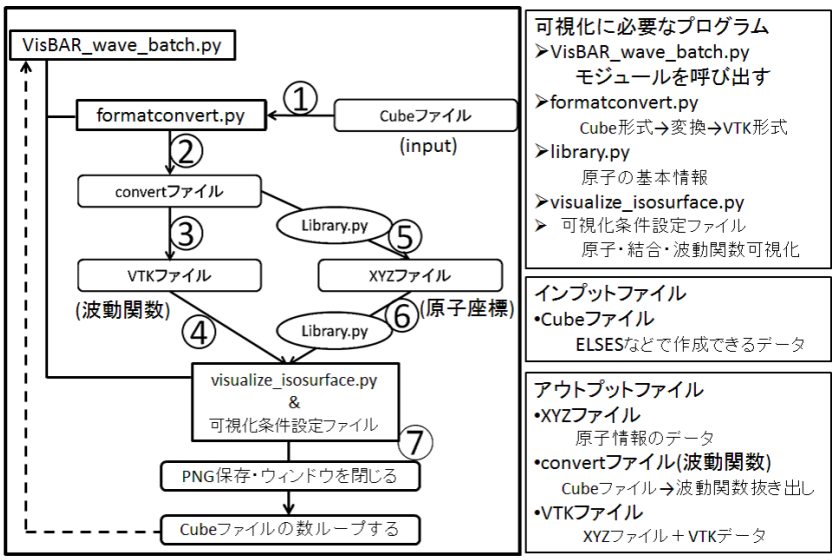

図 3.3: プログラム流れ図 (連続 PNG 保存)

3.2.1　フォーマット変換・可視化プログラムの説明

コンバートプログラム (formatconvert.py) が、図 3.2、図 3.3 の矢印 1 ～ 6 までを実行し、可視化プログラム (visualize_isosurface.py) が、図 3.2、図 3.2 の矢印 7 を実行している。
図 3.2、図 3.2 の違いは、描画方法の違いである。図 3.2 は一つのファイルをインタラクティブに描画・操作でき、図 3.3 は複数のファイルを連続で PNG ファイルにしている。

矢印 1
コンバートプログラムを実行して、Cube ファイルを読み取っている。ここで読み取っている情報は、表示する原子の数、中心座標、グリッド数、グリッド幅、原子番号、原子の位置情報 (x,y,z)、波動関数を読み取っている。グリッド数とは、各軸を分割した数であり、分割幅とは、分割した要素一つの幅である。

矢印 2
コンバートプログラムを用いて Cube ファイルから波動関数のみを取り出す。Cube ファイルでは、6 つのデータ毎に改行が入っているので、不必要な改行を取り出したものを波動関数のデータとして書き出す。

矢印 3
書き込まれた波動関数と VTK 形式で読み取る波動関数は x 軸と z 軸が逆になっている。なので、ここで座標変換を行い、VTK 形式に必要なファイルのフォーマット、データセット構造、グリッド数、グリッド幅、スカラー値データ型、要素数などを VTK ファイルとして書き出す。

矢印 5
コンバートファイルを通して、Cube ファイルから、原子の数、原子番号、原子座標を抜き出し、xyz ファイルに書き出す。

矢印 4,6,7
可視化プログラム (visualize_isosurface.py) を可視化条件設定ファイル (visbar_wb_setting_default.txt、bond_length.txt) で設定して実行し、作成した VTK ファイル、xyz ファイルを読み込んで、VTK モジュールで描画し、ウィンドウ上で可視化または、PNG ファイル保存をしている。

3.2.2　周期境界条件の適用について

周期境界条件は、実行時に「-p」のオプションをつけることで、周期境界条件が適用される。
では、どのように周期境界条件を適用しているのかを説明する。
まず、図 3.4 の可視化の範囲を以下のように定める。ここでは可視化範囲の枠が $(-5 \leq x \leq 5, -5 \leq y \leq 5, -5 \leq z \leq 5)$ の範囲にあるとする (配布している Cube ファイルでは正確には 5 ではなく、± 9.44863439 である)

原点座標が立方体の重心の位置に (0,0,0) あるとし、対象とする座標 A が A(12.5,0,5) にあるとする。

このとき、周期境界条件を適用すると、x 軸の正の方向に可視化範囲の枠を飛び出した原子の x 座標に対して、可視化範囲の枠の一辺 (10) の値を引いた値を原子の座標に計算しなおすことが行われる。

つまり A(12.5,0,2.5) − 枠の一辺 (10,0,0)=A'(2.5,0,2.5) と平行移動させられる。

プログラムでは、Cube ファイルから抜き出してきた原子座標に対し、可視化範囲を超えていると判断した座標に対し実行され、一度メモリに格納された座標を周期境界条件を適用し計算しなおした座標を再度メモリに格納、そして xyz ファイルに出力という流れになっている。

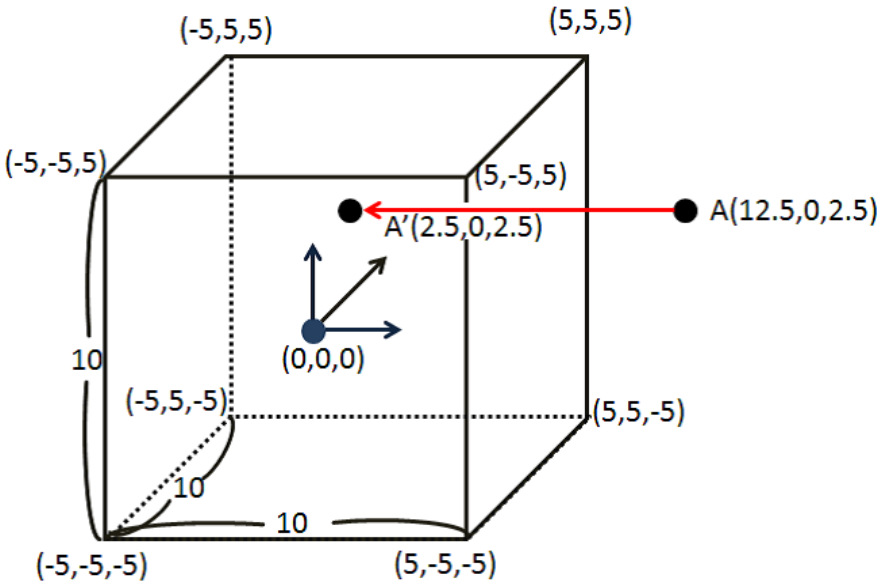

図 3.4: 周期境界条件適用模式図

次に原子座標が離れすぎた場合を説明する、図 3.5 のような場合である。これは、2 次元の図であり、x1 の存在する範囲が可視化の範囲内である。もし仮に x2 の座標に対して、周期境界条件を適用すると x2 の座標は (x2-10,y) となる。x3 の場合は (x3-20,y)、x4 の場合は (x4-30,y) というように座標を平行移動する。

具体的に、プログラムが行っている計算について説明する。
原子が x の正の方向に可視化範囲から出ている場合

原子座標 = 原子座標 - 可視化範囲の一辺

という計算を可視化範囲に入るまで繰り返し計算している。
x 軸の正の方向を例に挙げて説明を行ったが、x 軸の正負・y 軸の正負・z 軸の正負、全てに周期境界条件の修正は適用される。

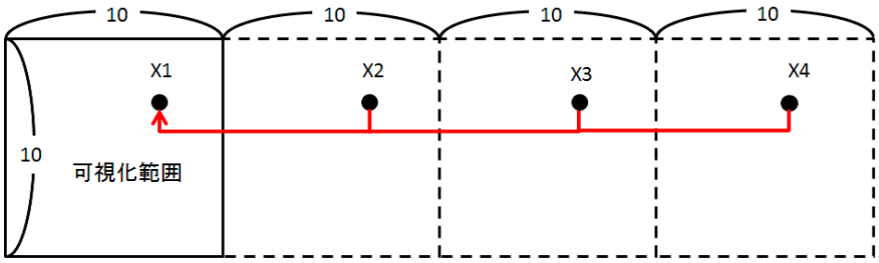

図 3.5: 原子が可視化範囲から離れた場合の模式図

3.3 ファイルフォーマット

この節では、ファイルの読み込み・書き出しに用いたファイルの形式について記す。

3.3.1 Cube ファイルフォーマット

以下に例として示すのは、サンプルデータの Cube ファイル「eigen_state_15-00000001.cube」である。これは、定温ダイナミクス計算を行ったベンゼン (C_6H_6) の HOMO 軌道の波動関数テキストデータである。

```
HOMO wavefunction of thermally vibrated
C6H6
   -12     -9.44863439     -9.44863439     -9.44863439
    80      0.23621586      0.00000000      0.00000000
    80      0.00000000      0.23621586      0.00000000
    80      0.00000000      0.00000000      0.23621586
 6  6       2.64561763      0.00000000      0.00000000
 6  6       1.32280881      2.29118046      0.00000000
 6  6      -1.32280881      2.29116156      0.00000000
 6  6      -2.64561763      0.00000000      0.00000000
 6  6      -1.32280881     -2.29118046      0.00000000
 6  6       1.32280881     -2.29116156      0.00000000
 1  1       4.64872812      0.00000000      0.00000000
 1  1       2.32436406      4.02591193      0.00000000
 1  1      -2.32436406      4.02591193      0.00000000
 1  1      -4.64872812      0.00000000      0.00000000
 1  1      -2.32436406     -4.02591193      0.00000000
 1  1       2.32436406     -4.02591193      0.00000000
         1        15
0.49187329E-09  0.61457295E-09  0.76580700E-09  0.95157949E-09  0.11789702E-08  ····
```

---

1 行目　コメント文
2 行目　コメント文
3 行目　原子数と x,y,z の原点であって、各軸-9.44863439 ずれている
これは、ステップ幅 (0.23621586) ×ステップ数の半分 (40) の値である
4 行目　グリッド数 (80 ステップ)　x 軸のグリッド幅
5 行目　グリッド数 (80 ステップ)　y 軸のグリッド幅
6 行目　グリッド数 (80 ステップ)　z 軸のグリッド幅
7 行目〜12 行目　原子番号と原子の座標 (x,y,z)　この場合は炭素原子である
12 行目〜17 行目　原子番号と原子の座標 (x,y,z)　この場合は水素原子である
18 行目　コメント文
19 行目以降　波動関数のデータが並ぶ

3.3.2　xyz ファイルフォーマット

ベンゼン (C_6H_6)[3.3.1] の Cube ファイルから、原子座標を xyz ファイルに書き直したものである。

```
12
'HOMO wavefunction of thermally vibrated C6H6 '
C 2.64561763 0.00000000 0.00000000 C1
C 1.32280881 2.29118046 0.00000000 C2
C -1.32280881 2.29116156 0.00000000 C3
C -2.64561763 0.00000000 0.00000000 C4
C -1.32280881 -2.29118046 0.00000000 C5
C 1.32280881 -2.29116156 0.00000000 C6
H 4.64872812 0.00000000 0.00000000 H1
H 2.32436406 4.02591193 0.00000000 H2
H -2.32436406 4.02591193 0.00000000 H3
H -4.64872812 0.00000000 0.00000000 H4
H -2.32436406 -4.02591193 0.00000000 H5
H 2.32436406 -4.02591193 0.00000000 H6
```

---

1 行目 原子の個数
2 行目 コメント文 (Cube ファイルの 1 行目+2 行目)
3 行目以降は、1 行が 1 つの原子の情報を表している。それぞれの行には原子の元素記号、 x 座標、 y 座標、 z 座標、原子のラベルが書かれていて、座標は a.u. 単位である。

3.3.3　VTK ファイルフォーマット

ベンゼン (C_6H_6)[3.3.1] の Cube ファイルを VTK 形式に変換したもの。

```
# vtk DataFile Version 2.0
Probability density for the 3d electron position in a hydrogen atom
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 80 80 80
ORIGIN -9.44863439 -9.44863439 -9.44863439
SPACING 0.23621586 0.23621586 0.23621586
POINT_DATA 512000
SCALARS probability_density float
LOOKUP_TABLE default
0.000000 0.000000 0.000000 0.000000 ····
```

---

1 行目　ファイルのバージョンと識別子
2 行目　ヘッダ情報 (データの名前や説明文)
3 行目 ファイルのフォーマット (ASCII か BINARY、通常は ASCII)
4 行目 データセット構造 (等間隔格子状データである)
5 行目 各軸のグリッド数（x,y,z）
6 行目 原点の位置 (x,y,z)
7 行目 データ間のグリッド幅 (X, Y, Z)
8 行目 全てのデータ数 (ここでは 80 × 80 × 80)
9 行目 スカラ値のデータ名、データ型、要素数
10 行目 LookUpTable がデフォルトであることを示す
11 行目以降　波動関数が続く

3.3.4　Cube ファイルと VTK ファイルのフォーマットの違い

Cube ファイルには、原子座標と波動関数が表記されている。しかし、VTK ファイルには原子座標を入れる部分が存在しないので、Cube ファイルから原子座標を抜き出し、xyz 形式のファイルに書き込んで、VTK に読み込ませている。
Cube ファイルと VTK ファイルには、波動関数がそれぞれ書き込まれているが、波動関数の並び方に大きな違いがある。
Cube ファイルでは (x,y,z) 各点の値が 6 個ずつ横に連続して出力される。各値は z → y → x と出力される。つまり、出力用配列　F(Mx,My,Mz) とすると

$$
\begin{array}{cccccc}
F(0,0,0) & F(0,0,1) & F(0,0,2) & F(0,0,3) & F(0,0,4) & F(0,0,5) \\
F(0,0,6) & F(0,0,7) & F(0,0,8) & F(0,0,9) & F(0,0,10) & F(0,0,11) \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
F(0,0,73) & F(0,0,74) & F(0,0,75) & F(0,0,76) & F(0,0,77) & F(0,0,78) \\
F(0,0,79) & F(0,0,80) & & & & \\
F(0,1,0) & F(0,1,1) & F(0,1,2) & F(0,1,3) & F(0,1,4) & F(0,1,5) \\
F(0,1,6) & F(0,1,7) & F(0,1,8) & F(0,1,9) & F(0,1,10) & F(0,1,11) \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
F(0,1,73) & F(0,1,74) & F(0,1,75) & F(0,1,76) & F(0,1,77) & F(0,1,78) \\
F(0,1,79) & F(0,1,80) & & & &
\end{array}
$$

このように出力されている。

対して、VTK ファイルでは、グリッド数分が一行になって出力される。今回の場合、80 個のデータが一行となり、x → y → z の順に出力される。つまり、出力用配列を F(Nx,Ny,Nz) とすると

$$
\begin{array}{cccccc}
F(0,0,0) & F(1,0,0) & F(2,0,0) & \dots & F(79,0,0) & F(80,0,0) \\
F(0,1,0) & F(1,1,0) & F(2,1,0) & \dots & F(79,1,0) & F(80,1,0) \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
F(0,80,0) & F(1,80,0) & F(2,80,0) & \dots & F(79,80,0) & F(80,80,0) \\
F(0,0,1) & F(1,0,1) & F(2,0,1) & \dots & F(79,0,1) & F(80,0,1) \\
F(0,1,1) & F(1,1,1) & F(2,1,1) & \dots & F(79,1,1) & F(80,1,1) \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
F(0,80,80) & F(1,80,80) & F(2,80,80) & \dots & F(79,80,80) & F(80,80,80)
\end{array}
$$

と出力される。

このフォーマットの違いのため、フォーマット変換プログラムで Cube ファイルから不必要な改行を除去することと、座標の変換を行い、Cube 形式を VTK 形式に変換している。

参考文献

[1] Python (公式ホームページ)
http://www.python.org/

[2] Visualization Tool Kit (VTK 公式ホームページ)
http://www.vtk.org/

付 録A　vktpythonを用いない場合のVTKビルド

！注意！: 本付録の内容は旧情報 (2014 年執筆) である。最新とは異なる場合があるので、注意すること。

2 章のインストールでは、VTKPython をインストーラからインストールしたが、ここでは、Python 用に wrapper された VTK のインストール方法を載せる。

ここでは、2 通りのインストール方法を紹介する。

1. Python 用に wrapper された VTK のインストール

2. VTK の公式サイトで配布されているソースファイルからビルドしインストール

A.1　Python用にwrapperされたVTKのインストール

以下に載せるインストールの内容は、

1. Python(公式ホームページ) のインストール

2. インターネット上にアップロードしてある VTK の Python の Wrapper 版インストール

以上 2 つの項目である。本来の VTK のインストールは付録 A.2 の方法にあるソースコードをビルドするインストールが本筋だと考えるが、作業量が多いので、覚悟して行って頂きたい。まず Python 用に wrapper された VTK を使ってみたい方は、作業量の少ない、以下のインストーラの使用方法で試していただきたい。ただし、この配布先はオフィシャルなホームページではないので、使用に際しては自己責任で使用していただく。

【インストール作業手順】

1. Python 公式サイト `http://www.python.org/download/` から「Python2.7.x」の最新インストーラー、2013 年 8 月現在 (Python 2.7.5 Windows Installer (Windows binary ⋯))

をダウンロードし、インストールする。

2. スタートメニュー⇒すべてのプログラム⇒ Python2.7 から「Python(command line)」を開き、「print ”hello”」と入力し、hello と出力されれば Python のインストールは成功である。

3. 最後に、VTK のインストールを行う。自分でビルドする方法もあるが (付録 A.2 を参照) 手間がかかるので、ここでは既存のインストーラーを利用する方法を説明する。Python(ここでは、ver2.7) をインストールしている状態で、以下 1 の URL から、インストーラをダウンロードし、インストールする。インストーラは、URL ページの下部、VTK のカテゴリ内にある。

   2、3 は参考 URL であり、2、3 の URL からダウンロードできるインストーラは共に動作確認は済んでいる。1 の URL が消えたときの Backup 用である。ただし、これら 3 つの URL からダウンロードできるインストーラは公式に配布されているものではないので、使用する際には自己責任で使用していただく。

   1,VTK-5.10.1.win32-py2.7.exe
   `http://www.lfd.uci.edu/~gohlke/pythonlibs/`

   (2,VTK-5.8.0.win32.exe [Python2.7])
   `http://www.mamba-image.org/download.html`

   (3,VTK-5.8.0.win32-py2.7.exe)
   `http://note.sdo.com/u/1519020158/NoteContent/qyCx_~jDGekMLX03k001m_`
   上記 URL からダウンロードし、ダブルクリックするとインストールが開始される。

4. インストールが終了したら、sample のディレクトリに 3.1.2 に書いてあるソースコード「sphere.py」を用意したので、ダブルクリックして実行する。

5. sample フォルダの中にある「sphere.py」を実行し、図 3.1 の画面にある白球が表示されればインストールは成功である。

A.2 VTK の公式サイトで配布されているソースファイルからビルドしインストール

<今回使用した環境>

DELL　VOSTRO3350
Windows 7 Professional　OS32bit　servicepack1
WindowsUpdate を行い最新の状態にしておくことを推奨する。

Python2.7.3
http://www.python.org/

Visual Studio Express 2012 for Windows Desktop
http://www.microsoft.com/visualstudio/jpn/downloads

VTK 5.10.1
vtkdata-5.8.0.zip
CMake(cmake-2.8.10.2-win32-x86.exe)
http://www.vtk.org/VTK/resources/software.html

C++コンパイラー
http://www.embarcadero.com/jp/products/cbuilder/free-compiler

gfortran
http://gcc.gnu.org/wiki/GFortranBinaries
以上の環境を使用した。

<参考にしたホームページ>

主に Linux や計算化学：忘備録
http://home.hiroshima-u.ac.jp/~tyoshida/dokuwiki/vtk

4. VTK を Python で動かす — 3D イメージ可視化に向けて
http://www.nips.ac.jp/huinfo/documents/python/python04.html

Windows で VTK をビルドとインストール（Windows の Visual Studio を使用）
http://www.kkaneko.com/rinkou/cygwin/vtkvisualc.html

VTK 5.8.0 + Python 2.7.2 のビルド
http://blog.livedoor.jp/satoru0503/archives/51814929.html

<実行内容>
(すでに上記環境は整っている状態で)

1. 最終的に VTK がインストールされる場所として、
   C:/Users/damp-tottori/Desktop/VTK_yamazaki_Download　を作成する。

2. VTK のダウンロードサイト (http://www.vtk.org/VTK/resources/software.html) より、VTK のソースファイル vtk.5.10.1zip をデスクトップに解凍する。

3. 例に使用されるデータも、VTKのダウンロードサイトからダウンロード。vtkdata-5.10.1zip を解凍する。

4. CMake を起動。一番上のソースコードのテキストボックスに、VTK のソースファイル (VTK5.10.1) のフォルダ名を入れる。
例えば、C:/Users/damp-tottori/Downloads/vtk-5.10.1 とする。

5. 次のテキストボックスには、作成するものが置かれるフォルダ名を入れる。例えば、C:/Users/damp-tottori/Desktop/vtkBuild とする。フォルダを、作成するか尋ねられるので Yes。

6. 中程にある Configure ボタンをクリック。Visual Studio11 を選択して、compiler の指定は一番上の native のままで、Finish をクリック。
※もし、CMAKE_MAKE_PROGRAM の欄に MAKE_PROGRAM_NOT_FOUND となった場合、「C:/Windows/Microsoft.NET/Framework/v4.0.30319/MSBuild.exe」とする。その後、Configure を選択する。

7. Advanced(上の方の真ん中やや右寄り) をチェックし、赤色になった部分を以下のように変える。
BUILD_SHARED_LIBS → check
CMAKE_CONFIGURATION_TYPES → Release
CMAKE_INSTALL_PREFIX → C:/VTK
VTK_DATA_ROOT → C:/VTK/vtkdata-5.10.1
VTK_WRAP_PYTHON → check

   ※CMAKE_LINKER という欄に「～_NOT_FOUND」という文字が出ていた場合「C:/Program Files/Microsoft Visual Studio 11.0/VC/bin/link.exe」を選択する
   ※ COVERAGE_COMMAND 問いう欄に「～_NOT_FOUND」という文字が出ていた場合、「C:/Program Files/gfortran/bin/gcov.exe」を選択する

8. ここで一度 Configure　をクリックすると、エラーが出てくる。

9. 指示に従って VTK_USE_TK の check をはずし、もう一度 Configure。エラーメッセージが出なければ Configure はこれで終了。

10. Configure ボタンの右にある、Generate ボタンをクリック。これで VTK.sln が vtkBuild 内に作られる。cmake を終了する。

11. vtkBuild 内にある VTK.sln を右ボタンクリックし、プログラムから開き、Microsoft Visula Studio 2012 を起動。

12. Visual Studio の Solution Explorer に ALL_BUILD というプロジェクトが表示されるので、右クリックでビルドを選択。

13. ALL_BUILD が無事終わったら、Solution Explorer の RUN_TESTS を右クリックでビルドを選択で、テストプログラムが走る。すべてのテストプログラムをパス出来るわけではなかった。エラーが出る場合もあるが下記を続行する。

14. Solution Explorer の INSTALL を、右クリックでビルドを選択。Python の関係も含めてプログラム類がインストールフォルダにコピーされる。

15. Python と VTK を結びつける必要がある。通常、追加された Python パッケージは、Python フォルダの C:/Python27/Lib/site-packages 以下に置かれる。
上記のインストールでは VTK の Python 関係ファイルは、VTK インストールフォルダ (C:/Users/damp-tottori/Desktop/VTK_yamazaki_Download) の/lib/site-packages に置かれているので、これらをつなぐために、C:/Python27/Lib/site-packages に vtk.pth という名前のテキストファイルを作成し、
内容はC:/Users/damp-tottori/Desktop/VTK_yamazaki_Download/lib/site-packages とする。

16. C:/Users/damp-tottori/Desktop/VTK_yamazaki_Download/bin　の中にあるすべてのファイルを、C:/Users/damp-tottori/Desktop/VTK_yamazaki_Download/lib/site-packages の中「の vtk」にコピーする。

17. 問題なくインストール出来ていれば、Python を起動し、import vtk でエラーがでない。

<インストール先のフォルダの注意事項>
インストール先を C:\Program Files にしないこと。管理者権限が必要なので、INSTALL を VisualStadio で行ったときにエラーが出ることがあった。

<注意事項>
C++のコンパイラーを入れたので、C++がない場合のビルドは行っていない。
なので、ビルドの成功が C++コンパイラーがインストールされているかどうかは未確認である。

付 録B　リモート操作(オフスクリーンモード)での利用

ここでは、Linux ワークステーション上に本ツールをインストールし、オフスクリーンモードでリモート操作する際の注意点を述べる。
オフスクリーンモードとは、GUI(X window system) を用いずコマンドラインだけでバッチ処理を実行するモードをさす。生成された画像ファイルは、ローカルマシンに転送してから表示することを想定している。
デフォルトの設定のままでは、オフスクリーンモードでの実行時にエラーがでてしまう。エラーを避けるためには、VTK ライブラリをオフスクリーンモードでビルドする必要がある。
以下は、VTK ライブラリをオフスクリーンモードでビルドする手順である。

http://www.vtk.org/download/ から「VTK-6.3.0.zip」をダウンロードする。
「VTK-6.3.0.zip」のあるフォルダに移動後以下のコマンドを実行する。

```
$ su
# yum install cmake
# yum install tk-devel
# yum install mesa-libOSMesa-devel
# exit
$ unzip VTK-6.3.0.zip
$ cd VTK-6.3.0
$ mkdir VTK_build
$ cd VTK_build
$ cmake -DVTK_OPENGL_HAS_OSMESA:BOOL=ON -DVTK_USE_OFFSCREEN:BOOL=ON \
-DVTK_USE_X:BOOL=OFF -DVTK_WRAP_PYTHON:BOOL=ON \
-DBUILD_SHARED_LIBS:BOOL=ON ../ -DCMAKE_INSTALL_PREFIX=$HOME/local
$ make -j 16
$ make install
$ emacs ~/.bashrc
====
export LD_LIBRARY_PATH=$HOME/local/lib:$LD_LIBRARY_PATH #vtk6.3.0
export PYTHONPATH=$HOME/local/lib/python2.6/site-packages #vtk6.3.0
====
```

バックスラッシュは改行なしを表している。

＜参考にしたホームページ＞

http://d.hatena.ne.jp/bettamodoki/20140321/1395400281

付 録C　argparseのインストール (Python v.2.6以前を利用する場合)

VisBAR Wave Batch は python2.7 から標準で搭載された argparse を使用している。
python2.6 以前のバージョンで VisBAR Wave Batch 使用する場合は argparse のインストールする必要がある。
以下は argparse のインストール方法である。

`https://pypi.python.org/pypi/argparse` から「argparse-1.4.0.tar.gz」をダウンロードする。
「argparse-1.4.0.tar.gz」のあるフォルダに移動後以下のコマンドを実行する。

```
$ tar zxvf argparse-1.4.0.tar.gz
$ cd argparse-1.4.0
$ python setup.py build
$ python setup.py install --user
```

付 録D　mpi4pyのインストール

VisBAR Wave Batch は mpi4py を使用することで分散型の並列処理を行うことができる。(1 つ 1 つ可視化時間が短くなるわけではない)
以下は mpi4py のインストール方法である。

`https://bitbucket.org/mpi4py/mpi4py/downloads` から mpi4py-1.3.1.tar.gz をダウンロードする。

```
$ tar zxvf mpi4py-1.3.1.tar.gz
$ cd mpi4py-1.3.1
$ python setup.py build
$ python setup.py install --user
```

付 録E　ソースコード詳細解説(v.1.0.7)

本付録では、v.1.0.7 におけるソースコードの詳細を解説する。最新版とは異なる部分もあるので、注意すること。

ページの関係で表示できなかった部分は折り返し部分に「/」を入れて表示している。

E.1　メインプログラムのソースコード解説

使用するモジュールを適用する。

```
import os
```
⇒オペレーションシステムを参照。

```
import re
```
⇒正規表現モジュールを参照。

```
import formatconvert as convert
```
⇒フォーマット変換プログラムを convert という名前に指定。このプログラムの下記で使用している。

```
import visualize_isosurface as view
```
⇒可視化プログラムを view という名前に指定。このプログラムの下記で使用している。

```
import vtk
```
⇒ vtk のライブラリを参照している。

```
import argparse
```
⇒ argparse をインポートする。

パスの指定、実行した Cube ファイルを記録するログの出力、他初期値の指定。

```
pwd = os.getcwd()
```
⇒パスをカレントディレクトリに指定。

```
files = os.listdir(pwd)
```
⇒ path(カレントディレクトリ) にあるすべてのファイルとサブディレクトリの名前からなるリストを返す。順番は不動。

```
out_log = open('log_VTK.txt','w')
```
⇒ log_VTK.txt と名付けたファイルを書き込み専用で作成し、out_log という変数で置いておく。

```
cnt = 0
```
⇒カウンタを指定し、以下で使用している。

```
parser = argparse.ArgumentParser(prog="VisBAR_wave_batch")
```
⇒ argparse を設定し、プログラム名を VisBAR Wave Batch とする。名前を paser とする。

```
parser.add_argument("input_dir", metavar="[Directry]", type=str , \
help='Set input directry')
```
⇒オプションに入力ディレクトリを input_dir として paser に指定する

```
parser.add_argument("-o", type=str, metavar="[Directry]",\
 help='Select output directry')
```
⇒オプションに出力ディレクトリを -o [ディレクトリ名] として paser に指定する

```
parser.add_argument('-s', default=False, action="store_true", \
help='Sign correction mode')
```
⇒オプションに入力ディレクトリを input_dir として paser に指定する

```
parser.add_argument('-b', default=False, action="store_true", \
help='Batch mode')
```
⇒バッチ処理のオプションを -b として paser に指定する

```
parser.add_argument('-p', default=False, action="store_true", \
help='Periodic mode')
```
⇒周期境界条件のオプションを -b として paser に指定する

```
args = parser.parse_args()
```
⇒ parse の中身を args に代入する

```
vars_args = vars(args)
```
⇒ args の中身をディクショナリ形式にする

```
files = os.listdir(vars_args["input_dir"])
```

⇒インプットディレクトリに置いてあるファイル名を files に入れる。

input_dir = vars_args["input_dir"]
⇒インプットディレクトリまでの path を input_dir に入力する

if vars_args["o"]:
⇒ -o オプションを設定している場合、if 分岐に入る。

    output_dir = vars_args["o"]
⇒ output_dir にオプションで指定したディレクトリ path を入れる

else:
⇒ -o オプションが無い場合 else 分岐に入る

    output_dir = vars_args["input_dir"]
⇒ output_dir に input_dir の path を入れる。

Batch_mode = vars_args["b"]
⇒ Batch_mode にオプション [-b] があるなら True をなければ False を代入する。

sign_correction_mode = vars_args["s"]
⇒ sign_correction_mode にオプション [-s] があるなら True をなければ False を代入する。

boundary_mode = vars_args["p"]
⇒ boundary_mode にオプション [-p] があるなら True をなければ False を代入する。

インプットの Cube ファイルを引数として、フォーマット変換プログラムと、可視化プログラムに渡して実行する。また、1 回目と 2 回目以降に実行するフォーマット変換のプログラムは異なる。

files = filter(lambda f:f.find(".cube")>=0, files)
⇒ファイル名が .cube と付くものを files のなかから探し、files に代入する

for filenametext in files:
⇒ files は、カレントディレクトリにあるファイル名一覧である。カレントディレクトリにあるファイル名を filenametext という変数に入れて、ファイルの数だけループを回す

        print "-------------------------------------"
⇒区切り線。実行内容を区切って分かるようにするため。

        print filenametext

⇒実行しているファイル名を出力する。今どの Cube ファイルを実行しているかわかるようにするため。

```
            out_log.write(filenametext + '\n')
```
⇒ログファイルに実行したファイル名を書き込む。後から、どのファイルが実行されたかわかるようにするため。

```
            if cnt == 0:
```
⇒カウンタが 0 ならば以下の if 文に従う。1 回目と 2 回目以降ではフォーマット変換の処理が異なるため。

```
                check_line=convert.cube_vtk(input_dir,filenametext,boundary_mode)
```
⇒ filenametext の引数を渡して、コンバート変換プログラムを実行する。引数には、入力 Cube ファイルのあるディレクトリ、現在実行しているファイルの名前と周期境界条件を適用するかどうかを選択した、ユーザーインプットの値が入っている。

```
                view.vtk(output_dir,filenametext,Batch_mode)
```
⇒ output_dir,filenametext,Batch_mode の引数を渡して、可視化プログラムを実行する。

```
                if sign_correction_mode == "1":
```
⇒ sign_correction_mode の値を 1 と洗濯しているならば、if 分岐に従う。

```
                    cnt +=1
```
⇒ cnt の値に 1 を加えて、sign_correction を適用する以下の elif 文に沿う。

```
                    print "sign_correction ON!"
```
⇒適用されたことが分かるように print 文を用意した。

```
            elif cnt >= 1:
```
⇒カウンタが 1 以上ならば以下の elif 文に従う。1 回目と 2 回目以降ではフォーマット変換の処理が異なるため。

```
                check_line=convert.cube_vtk2(input_dir,\
filenametext,check_line,boundary_mode)
```
⇒ filenametext の引数を渡して、コンバート変換プログラムを実行する。実行した内容を check_line に入れなおす。引数には、入力 Cube ファイルのあるディレクトリ、現在実行しているファイルの名前と周期境界条件を適用するかどうかを選択した、ユーザーインプットの値が入っている。

```
                view.vtk(output_dir,filenametext,Batch_mode)
```
⇒ output_dir,filenametext,Batch_mode の引数を渡して、可視化プログラムを実行する。

```
out_log.close()
```

⇒ログファイルを閉じる。

E.2　フォーマット変換プログラムのソースコード解説

使用するモジュールを適用し、読み取る **Cube** ファイル、書き出す **xyz** ファイルを開く。

```
import os
```
⇒オペレーションシステムを参照。

```
import re
```
⇒正規表現モジュールを参照。

```
import library as lib
```
⇒ library モジュールを lib と名付けて参照。

```
import visualize_isosurface as view
```
⇒ visualize_isosurface のファイルを View と名付けて参照。

```
def cube_vtk(input_dir,filenametext,boundary_mode):
```
⇒ input_dir,filenametext と boundary_mode を引数としてコンバートプログラムを関数にする。

```
    filenametext = os.path.join(input_dir,filenametext)
```
⇒ filenametext に input_dir の path と filenametext の Cube ファイル名を連結したものを入れる。

```
    f = open(filenametext,'r')
```
⇒ filenametext の変数に入っている Cube ファイルを読み取り専用で開く。

```
    out = open('make_atom.xyz.txt','w')
```
⇒ ('make_atom.xyz.txt','w') のファイルを書き出し専用で開く。

空のリストを作成し、その中に **Cube** ファイルに書かれている要素の上から **6** 行目までを書かれている要素を区切ったものをリストに入れる。

```
    cube_list=[]
```
⇒ cube_list と名付けた空のリストを作成する。

```
    for j in range(0,6):
```
⇒ j を 0〜5 までループする。

```
        cube_read = f.readline()
```

⇒ファイルを 1 行ずつ読み込む。

```
        cube_read = cube_read.rstrip()
```
⇒ 1 行ずつ読んだ行末の空白文字を消す。

```
        cube_read = re.sub("\n","",cube_read)
```
⇒ 1 行ずつ読んだ文の改行を取り除く。

```
        cube_read = re.split(" *",cube_read)
```
⇒ 1 行ずつ読んだ行を空白で区切る。

```
        cube_list +=[cube_read]
```
⇒加工しながら読んだ 1 行ずつのデータを cube_list に入れる。

Cube ファイルのコメント文と原子数の値を抜き出して xyz ファイルに書き込む。

```
    new_cube = int(cube_list[2][1])*-1
```
⇒ cube_list[2][1] は xyz ファイルの原子数が入っており、負の数になっているので、整数表記にして、「-」を掛けている。

```
    out.write(str(new_cube) + '\n')
```
⇒原子数をファイルに書き込む。

```
    comment0=len(cube_list[0])
```
⇒ Cube ファイルの一つ目のコメント文がいくつあるかを判別する。

```
    comment1=len(cube_list[1])
```
⇒ Cube ファイルの二つ目のコメント文がいくつあるかを判別する。

```
    out.write("'")
```
⇒アポストロフィーを書き込む。

```
    for com0 in range(0,comment0):
```
⇒一行目のコメント文の個数だけループを回す。

```
        out.write(str(cube_list[0][com0])+' ')
```
⇒一行目のコメントと空白を書き込む。

```
    for com1 in range(0,comment1):
```
⇒二行目のコメント文の個数だけループを回す。

```
        out.write(str(cube_list[1][com1])+' ')
```
⇒二行目のコメントと空白を書き込む。

```
    out.write("'"+'\n')
```
⇒アポストロフィーと改行を書き込む。

Cube ファイルの原子番号と原子座標 (7 行目から 19 行目まで) を読み込んでリストに入れる。

```
    all_atom=[]
```
⇒ all_atom と名付けた空のリストを作成する。

```
    for j in range(0,new_cube):
```
⇒原子数の数 (new_cube) だけループを回す。

```
        atomxyz = f.readline()
```
⇒ファイルを 1 行ずつ読み込む。

```
        atomxyz = atomxyz.rstrip()
```
⇒ 1 行ずつ読んだ行末の空白文字を消す。

```
        atomxyz = re.sub("\n","",atomxyz)
```
⇒ 1 行ずつ読んだ文の改行を取り除く。

```
        atomxyz = re.split(" *",atomxyz)
```
⇒ 1 行ずつ読んだ行を空白で区切る。

```
        if atomxyz[0] != "":
```
⇒一つ目の要素が ""でなかった場合 if 分岐に入る

```
        atomxyz = [" "] + atomxyz
```
⇒要素の最初に空白を入れる

```
        all_atom +=[atomxyz]
```
⇒加工しながら読んだ 1 行ずつのデータを atomxyz に入れる。

周期境界条件の設定を行った場合、以下の if 文の内容を適用する。Cube ファイルから読み込んだ原子座標を周期境界条件を適用して各座標を修正。その後再度メモリへ格納し、xyz ファ

イルへ書き込みを行う

```
    if boundary_mode == "True":
```
⇒メインプログラムでの引数 boundary_mode が True の場合、以下の if 分岐に入る

```
        for i in range(0,new_cube):
```
⇒原子数の数だけループを回す

```
            if float(all_atom[i][3]) > float(cube_list[2][2])*(-1):
```
⇒原子の x 座標 (all_atom[i][3]) が可視化範囲を示す枠の範囲 ((cube_list[2][2])*(-1))
を超えた場合、if 分岐に入る

```
                kyori=(float(all_atom[i][3])-float(cube_list[2][2]))\
/(float(cube_list[2][2])*(-2))
```
⇒原子の x 座標に (all_atom[i][3])、可視化の枠の半分の距離を足し ((cube_list[2][2])cube_list[2]
の符号は負)、可視化枠の一辺の距離 ((float(cube_list[2][2])*(-2)) で割ったものを kyori
という変数に入れる

```
                all_atom[i][3]=str(float(all_atom[i][3])\
+float(cube_list[2][2])*int(kyori)*2)
```
⇒原子の x 座標に可視化枠の一辺の距離が kyori の整数分引かれる。これによって、x 方向に
可視化枠を飛び出した x 座標が可視化枠内に戻ってくる。

【以下同様】

```
            elif float(all_atom[i][3]) < float(cube_list[2][2]):
```
⇒原子の x 座標が可視化範囲より負の方向に超えているとき

```
                kyori=(float(all_atom[i][3])+float(cube_list[2][2]))\
/(float(cube_list[2][2])*(-2))

                all_atom[i][3]=str(float(all_atom[i][3])\
+float(cube_list[2][2])*int(kyori)*2)

            if float(all_atom[i][4]) > float(cube_list[2][3])*(-1):
```
⇒原子の y 座標が可視化範囲より正の方向に超えているとき

```
                kyori=(float(all_atom[i][4])-float(cube_list[2][3]))\
/(float(cube_list[2][3])*(-2))

                all_atom[i][4]=str(float(all_atom[i][4])\
```

```
+float(cube_list[2][3])*int(kyori)*2)

            elif float(all_atom[i][4]) < float(cube_list[2][3]):
```
⇒原子の y 座標が可視化範囲より負の方向に超えているとき

```
                kyori=(float(all_atom[i][4])+float(cube_list[2][3]))\
/(float(cube_list[2][3])*(-2))

                all_atom[i][4]=str(float(all_atom[i][4])\
+float(cube_list[2][3])*int(kyori)*2)

            if float(all_atom[i][5]) > float(cube_list[2][4])*(-1):
```
⇒原子の z 座標が可視化範囲より正の方向に超えているとき
```
                kyori=(float(all_atom[i][5])-float(cube_list[2][4]))\
/(float(cube_list[2][4])*(-2))

                all_atom[i][5]=str(float(all_atom[i][5])\
+float(cube_list[2][4])*int(kyori)*2)

            elif float(all_atom[i][5]) < float(cube_list[2][4]):
```
⇒原子の z 座標が可視化範囲より負の方向に超えているとき

```
                kyori=(float(all_atom[i][5])+float(cube_list[2][4]))\
/(float(cube_list[2][4])*(-2))

                all_atom[i][5]=str(float(all_atom[i][5])\
+float(cube_list[2][4])*int(kyori)*2)
```

原子番号と原子情報ライブラリ (**library.py**) を用いて元素記号に変換し、原子の位置情報と原子ラベルを **Cube** ファイルから **xyz** ファイルに書き込んでいる。

```
    elem_num_list = []
```
⇒ elem_num_list という名前で空リストを作成する

```
    elem_label = []
```
⇒ elem_label という名前で空リストを作成する

```
    read=view.setting_read()
```
⇒ visualize_isosurface.py にある setting_read の関数を read に代入する。

```
    for k in range(0,int(cube_list[2][1])*-1):
```

⇒ファイルに存在する原子の数だけループを回す。

```
        for m in range(0,117):
```

⇒ library.py に入っている要素の数 (117 個) だけループをまわす。

```
            if lib.library.items()[m][1][0] == int(float(all_atom[k][1])):
```

⇒ library.py に入っている原子番号と Cube ファイルに書かれている原子番号が一緒の場合、if 文分岐に入る。

```
                elem_num_list += lib.library.items()[m][0]
```

⇒ elem_num_list に if 分岐に合致した元素記号を代入する。

```
    for i,elem in enumerate(elem_num_list) :
```

⇒ elem_num_list に入っているものを一つずつ elem に代入する。またループの回数を i に入力する。

```
        if read["LabelType"][0] == "1":
```

⇒設定ファイルの LabelType の値が 1 のとき if 分岐に入る

```
            elem_label += [str(i+1)]
```

⇒ elem_label にループの回転数を入れる

```
        if read["LabelType"][0] == "2":
```

⇒設定ファイルの LabelType の値が 2 のとき if 分岐に入る

```
            elem_label += [elem ]
```

⇒ elem_label に元素記号を elem_label に入れる

```
        if read["LabelType"][0] == "3":
```

⇒設定ファイルの LabelType の値が 3 のとき if 分岐に入る

```
            elem_label += [str(elem_num_list[0:i].count(elem)+1) + elem]
```

⇒元素ごとの数と、元素記号を elem_label に入れる

```
        if read["LabelType"][0] == "4":
```

⇒設定ファイルの LabelType の値が 4 のとき if 分岐に入る

```
            elem_label += [str(i+1)+ elem ]
```

⇒原子の通し番号と元素記号を elem_label に入れる

```
        else :
```

⇒それ以外の入力がある場合

```
            elem_label += [str(i+1)+ elem ]
```
⇒原子の通し番号と元素記号を elem_label に入れる

```
    for k in range(0,int(cube_list[2][1])*-1):
```
⇒ファイルに存在する原子の数だけループを回す。

```
        for m in range(0,117):
```
⇒ library.py に入っている要素の数 (117 個) だけループをまわす。

```
            if lib.library.items()[m][1][0] == int(float(all_atom[k][1])):
```
⇒ library.py に入っている原子番号と Cube ファイルに書かれている原子番号が一緒の場合、if 文分岐に入る。

```
                out.write(lib.library.items()[m][0]+' ')
```
⇒ library.py に入っている原子番号を書く。

```
            else: continue
```
⇒ if 文の分岐に沿わなければ、ループを続行する。

```
        for l in range (3,7):
```
⇒ループ「l」を 3〜5 の値で回す。

```
            if l==6:
```
⇒「l」が 6 の場合。

```
                out.write(elem_label[k]+'\n')
```
⇒座標にあるラベルを書き。改行する。

```
            else:
```
⇒「l」が 6 以外の場合。

```
                out.write(all_atom[k][l]+' ')
```
⇒ xyz 形式の z 座標と空白を書く。

Cube ファイルと書き込んだ xyz ファイルを閉じている。

```
    out.close()
```
⇒書き込んだ xyz ファイルを閉じる。

```
    f.close()
```
⇒読み込んだ Cube ファイルを閉じる。

読み込む Cube ファイルを再び開き、Cube ファイルから波動関数を抜き出して書き込むファイルを開く。

```
    f = open(filenametext,'r')
```
⇒ filenametext の変数に入っている Cube ファイルを読み取り専用で開く。

```
    out = open('convert_wave_function.vtk.txt','w')
```
⇒ convert_wave_function.vtk.txt を波動関数抜き出し用に開く。

Cube ファイルの先頭から波動関数までをスキップする。

```
    check = 0
```
⇒ check に 0 という値を入れる。

```
    check2 = 1
```
⇒ check2 に 1 という値を入れる。

```
    for n in range(0,7+int(cube_list[2][1])*-1):
```
⇒ Cube ファイルの設定部分＋原子数だけループを回す。

```
        f.readline()
```
⇒ファイルを 1 行読み飛ばす。

Cube ファイルの波動関数を読み取っている。上記「L」と「mod」を使って、Cube ファイルの 6 データ毎改行を変更する。そうすることで、Cube ファイルでは波動関数が 6 行ごとに改行されていたが、波動関数を VTK ファイルの形式にあわせるため 1 行に出力する。

```
    if int(cube_list[5][1])%6 == 0 :
```
⇒ z 軸のメッシュ数が 6 で割り切れる場合 if 分岐に入る

```
        L = int(cube_list[5][1]) / 6
```
⇒ z 軸のメッシュ数を 6 で割った数を L に代入する

```
    elif int(cube_list[5][1])%6 != 0:
```
⇒ z 軸のメッシュ数が 6 で割り切れない場合 elif 分岐に入る

```
        L = int(cube_list[5][1]) / 6 + 1
```
⇒ z 軸のメッシュ数を 6 で割り、 1 を足した数を L に代入する

```
    for xy in range(0, int(cube_list[3][1]) * int(cube_list[4][1])):
```
⇒ x 方向と y 方向のメッシュ数を掛けて、その回数分ループを回す。

```
        for line_num_in_xy in range(0, L):
```
⇒ z 方向のデータを読むのに必要なループの回数 L を回す

```
            line = f.readline()
```
⇒ファイルを一行読み込む

```
            line = line.strip()
```
⇒ line で読み込んだ文字列の先頭と末尾から文字を取り除いた文字列にする

```
            if line == "":  # Skip empty line.
```
⇒空行が存在する場合、 if 分岐に入る

```
                line = f.readline()
```
⇒ファイルを一行読み込む

```
                line = line.strip()
```
⇒ line で読み込んだ文字列の先頭と末尾から文字を取り除いた文字列にする

```
            line = " ".join(map(lambda s: str(float(s)), re.split(" *", line)))
```
⇒読み込んだデータを空行で区切り、浮動小数表示にし、スペースを区切り文字として一つの文字列を作成する。

```
            out.write(line + " ")
```
⇒ line のデータとスペースを書き込む

```
        out.write("\n")
```
⇒改行文字を書き込む

```
    out.close()
```
⇒ out のファイルを閉じる

```
    f.close()
```
⇒ cube ファイルを閉じる。

波動関数の改行を変更した波動関数の可視化プログラムで読み込める **VTK** の形式をしたファイルを作成している。

```
    f = open('convert_wave_function.vtk.txt','r')
```
⇒波動関数を抜き出したファイルを開く。

```
    out = open('VTK_out.vtk','w')
```
⇒ VTK のファイルを書きだすファイルを作成する。

```
    total_line=[]
```
⇒ total_line の空リストを作成する。

```
    mirror = []
```
⇒ mirror の空リストを作成する。

```
    for j in range (0,int(cube_list[3][1])):
```
⇒ x 軸のグリッド数回分 j についてループを回す。

```
        all_line=[]
```
⇒ all_line の空リストを作成する。

```
        for i in range(0,int(cube_list[4][1])):
```
⇒ y 軸のグリッド数回分 i についてループを回す。

```
            line = f.readline()
```
⇒ファイルを 1 行読み込む。

```
            line = line.rstrip()
```
⇒ 1 行ずつ読んだ行末の空白文字を消す。

```
            line = re.sub("\n","",line)
```
⇒ 1 行ずつ読んだ文の改行を取り除く。

```
            line = re.split(" ",line)
```
⇒ 1 行ずつ読んだ行を空白で区切る。

```
            all_line += [line]
```
⇒加工しながら読んだ 1 行ずつのデータを all_line に入れる。

```
        total_line += [all_line]
```
⇒さらに all_line を total_line に入れる。

VTK のフォーマットを書き、x 軸と z 軸の座標変換を行っている。

```
    out.write('# vtk DataFile Version 2.0'+'\n')
```
⇒ VTK ファイルフォーマットバージョン情報を、VTK ファイルに書き込む。

```
    out.write('Probability density for the 3d electron position in /
a hydrogen atom'+'\n')
```
⇒ファイルの内容を書き込んでいる。

```
    out.write('ASCII'+'\n')
```
⇒ VTK ファイルに ASCII 形式で書きこむ。(ASCII と BAINARY がある)

```
    out.write('DATASET STRUCTURED_POINTS'+'\n')
```
⇒データが STRUCTURED_POINTS の形式であることを書き込む。

```
    out.write('DIMENSIONS'+' '+cube_list[3][1]+' '+cube_list[4][1]+/
' '+cube_list[5][1]+'\n')
```
⇒各軸のグリッド数を書き込んでいる。「eigen_state_000002.cube.txt」の場合 (x 軸,y 軸,z 軸)=(80,80,80) である。

```
    out.write('ORIGIN'+' '+cube_list[2][2]+' '+cube_list[2][3]+/
' '+cube_list[2][4]+'\n')
```
⇒原点の位置を書き込んでいる。「eigen_state_000001.cube.txt」の場合、(x,y,z)=(-9.44863439,-9.44863439,-9.44863439)

```
    out.write('SPACING'+' '+cube_list[3][2]+' '+cube_list[4][3]+/
' '+cube_list[5][4]+'\n')
```
⇒グリッドの間隔を書き込んでいる。「eigen_state_000001.cube.txt」の場合 (x 軸,y 軸,z 軸)=(0.23621586,0.23621586,0.23621586) である。

```
    out.write('POINT_DATA'+' '+str(int(cube_list[3][1])*int(cube_list[4][1])*/
int(cube_list[5][1]))+' '+'\n')
```
⇒全てのデータ数はいくつか書き込んでいる。「eigen_state_000001.cube.txt」の場合、(x 軸のグリッド数 80) × (y 軸のグリッド数 80) × (z 軸のグリッド数 80)=『512000』となる。

```
    out.write('SCALARS probability_density float'+'\n')
```
⇒スカラー値で書き込むことを宣言する。

```
    out.write('LOOKUP_TABLE default'+'\n')
```
⇒ LOOKUP_TABLE default であることを指定する。

```
    for k in range (0,int(cube_list[5][1])):
```
⇒ z 軸のグリッド数でループを回す。

```
        for j in range (0,int(cube_list[4][1])):
```
⇒ y 軸のグリッド数でループを回す。

```
            for i in range (0,int(cube_list[3][1])):
```
⇒ x 軸のグリッド数でループを回す。

```
                if i==(int(cube_list[3][1])-1):
```
⇒ x 軸のグリッド数で回すループが 80 回、ループしたとき if 分岐に入る。

```
                    out.write(total_line[i][j][k]+'\n')
```
⇒波動関数 +改行を書き込む。

```
                else:
```
⇒ x 軸のグリッド数で回すループが 80 回でないとき else 文に従う。

```
                    out.write(total_line[i][j][k]+' ')
```
⇒波動関数 +空白を書き込む。

```
    out.close()
```
⇒ VTK ファイルを閉じる。

```
    f.close()
```
⇒波動関数を変換したファイルを閉じる。

```
    print "end"
```
「end」と Python command line に表示する。

```
    return total_line
```
⇒ total_line を戻り値としてメインモジュールに戻す。

---

2 個目以降の Cube ファイルを連続して実行する場合使用する関数

```
def cube_vtk2(input_dir,filenametext,check_line,boundary_mode):
```
⇒ input_dir,filenametext、checl_line、boundary_mode を引数としてコンバートプログラムを関数にする。

```
    filenametext = os.path.join(input_dir,filenametext)
```

⇒ filenametext に input_dir の path と filenametext の Cube ファイル名を連結したものを入れる。

```
    f = open(filenametext,'r')
```
⇒ filenametext の変数に入っている Cube ファイルを読み取り専用で開く。

```
    out = open('make_atom.xyz.txt','w')
```
⇒ ('make_atom.xyz.txt','w') のファイルを書き出し専用で開く。

空のリストを作成し、その中に Cube ファイルに書かれている要素の上から 6 行目までを書かれている要素を区切ったものをリストに入れる。

```
    cube_list=[]
```
⇒ cube_list と名付けた空のリストを作成する。

```
    for j in range(0,6):
```
⇒ j を 0〜5 までループする。

```
        cube_read = f.readline()
```
⇒ファイルを 1 行ずつ読み込む。

```
        cube_read = cube_read.rstrip()
```
⇒ 1 行ずつ読んだ行末の空白文字を消す。

```
        cube_read = re.sub("\n","",cube_read)
```
⇒ 1 行ずつ読んだ文の改行を取り除く。

```
        cube_read = re.split(" *",cube_read)
```
⇒ 1 行ずつ読んだ行を空白で区切る。

```
        cube_list +=[cube_read]
```
⇒加工しながら読んだ 1 行ずつのデータを cube_list に入れる。

Cube ファイルのコメント文と原子数の値を抜き出して xyz ファイルに書き込む。

```
    new_cube = int(cube_list[2][1])*-1
```
⇒ cube_list[2][1] は xyz ファイルの原子数が入っており、負の数になっているので、整数表記にして、「-」を掛けている。

```
    out.write(str(new_cube) + '\n')
```
⇒原子数をファイルに書き込む。

```
    comment0=len(cube_list[0])
```
⇒ Cube ファイルの一つ目のコメント文がいくつあるかを判別する。

```
    comment1=len(cube_list[1])
```
⇒ Cube ファイルの二つ目のコメント文がいくつあるかを判別する。

```
    out.write("'")
```
⇒アポストロフィーを書き込む。

```
    for com0 in range(0,comment0):
```
⇒一行目のコメント文の個数だけループを回す。

```
        out.write(str(cube_list[0][com0])+' ')
```
⇒一行目のコメントと空白を書き込む。

```
    for com1 in range(0,comment1):
```
⇒二行目のコメント文の個数だけループを回す。

```
        out.write(str(cube_list[1][com1])+' ')
```
⇒二行目のコメントと空白を書き込む。

```
    out.write("'"+'\n')
```
⇒アポストロフィーと改行を書き込む。

Cube ファイルの原子番号と原子座標 (7 行目から 19 行目まで) を読み込んでリストに入れる。

```
    all_atom=[]
```
⇒ all_atom と名付けた空のリストを作成する。

```
    for j in range(0,new_cube):
```
⇒原子数の数 (new_cube) だけループを回す。

```
        atomxyz = f.readline()
```
⇒ファイルを 1 行ずつ読み込む。

```
        atomxyz = atomxyz.rstrip()
```
⇒ 1 行ずつ読んだ行末の空白文字を消す。

```
        atomxyz = re.sub("\n","",atomxyz)
```
⇒ 1 行ずつ読んだ文の改行を取り除く。

```
        atomxyz = re.split(" *",atomxyz)
```
⇒ 1 行ずつ読んだ行を空白で区切る。

```
        if atomxyz[0] != "":
```
⇒一つ目の要素が ""でなかった場合 if 分岐に入る

```
        atomxyz = [" "] + atomxyz
```
⇒要素の最初に空白を入れる

```
        all_atom +=[atomxyz]
```
⇒加工しながら読んだ 1 行ずつのデータを atomxyz に入れる。

周期境界条件の設定を行った場合、以下の if 文の内容を適用する。Cube ファイルから読み込んだ原子座標を周期境界条件を適用して各座標を修正。その後再度メモリへ格納し、xyz ファイルへ書き込みを行う

```
    if boundary_mode == "True":
```
⇒メインプログラムでの引数 boundary_mode が True の場合、以下の if 分岐に入る

```
        for i in range(0,new_cube):
```
⇒原子数の数だけループを回す

```
            if float(all_atom[i][3]) > float(cube_list[2][2])*(-1):
```
⇒原子の x 座標 (all_atom[i][3]) が可視化範囲を示す枠の範囲 ((cube_list[2][2])*(-1)) を超えた場合、if 分岐に入る

```
                kyori=(float(all_atom[i][3])-float(cube_list[2][2]))\
/(float(cube_list[2][2])*(-2))
```
⇒原子の x 座標に (all_atom[i][3])、可視化の枠の半分の距離を足し ((cube_list[2][2])cube_list[2]
の符号は負)、可視化枠の一辺の距離 ((float(cube_list[2][2])*(-2)) で割ったものを kyori という変数に入れる

```
                all_atom[i][3]=str(float(all_atom[i][3])\
+float(cube_list[2][2])*int(kyori)*2)
```

⇒原子の x 座標に可視化枠の一辺の距離が kyori の整数分引かれる。これによって、x 方向に可視化枠を飛び出した x 座標が可視化枠内に戻ってくる。

【以下同様】

```
            elif float(all_atom[i][3]) < float(cube_list[2][2]):
```
⇒原子の x 座標が可視化範囲より負の方向に超えているとき

```
                kyori=(float(all_atom[i][3])+float(cube_list[2][2]))\
/(float(cube_list[2][2])*(-2))

                all_atom[i][3]=str(float(all_atom[i][3])\
+float(cube_list[2][2])*int(kyori)*2)

            if float(all_atom[i][4]) > float(cube_list[2][3])*(-1):
```
⇒原子の y 座標が可視化範囲より正の方向に超えているとき

```
                kyori=(float(all_atom[i][4])-float(cube_list[2][3]))\
/(float(cube_list[2][3])*(-2))

                all_atom[i][4]=str(float(all_atom[i][4])\
+float(cube_list[2][3])*int(kyori)*2)

            elif float(all_atom[i][4]) < float(cube_list[2][3]):
```
⇒原子の y 座標が可視化範囲より負の方向に超えているとき

```
                kyori=(float(all_atom[i][4])+float(cube_list[2][3]))\
/(float(cube_list[2][3])*(-2))

                all_atom[i][4]=str(float(all_atom[i][4])\
+float(cube_list[2][3])*int(kyori)*2)

            if float(all_atom[i][5]) > float(cube_list[2][4])*(-1):
```
⇒原子の z 座標が可視化範囲より正の方向に超えているとき
```
                kyori=(float(all_atom[i][5])-float(cube_list[2][4]))\
/(float(cube_list[2][4])*(-2))

                all_atom[i][5]=str(float(all_atom[i][5])\
+float(cube_list[2][4])*int(kyori)*2)

            elif float(all_atom[i][5]) < float(cube_list[2][4]):
```
⇒原子の z 座標が可視化範囲より負の方向に超えているとき

```
                kyori=(float(all_atom[i][5])+float(cube_list[2][4]))\
/(float(cube_list[2][4])*(-2))

                all_atom[i][5]=str(float(all_atom[i][5])\
+float(cube_list[2][4])*int(kyori)*2)
```

原子番号と原子情報ライブラリ (library.py) を用いて元素記号に変換し、原子の位置情報と原子ラベルを Cube ファイルから xyz ファイルに書き込んでいる。

```
    elem_num_list = []
```
⇒ elem_num_list という名前で空リストを作成する

```
    elem_label = []
```
⇒ elem_label という名前で空リストを作成する

```
    read=view.setting_read()
```
⇒ visualize_isosurface.py にある setting_read の関数を read に代入する。

```
    for k in range(0,int(cube_list[2][1])*-1):
```
⇒ファイルに存在する原子の数だけループを回す。

```
        for m in range(0,117):
```
⇒ library.py に入っている要素の数 (117 個) だけループをまわす。

```
            if lib.library.items()[m][1][0] == int(float(all_atom[k][1])):
```
⇒ library.py に入っている原子番号と Cube ファイルに書かれている原子番号が一緒の場合、if 文分岐に入る。

```
                elem_num_list += lib.library.items()[m][0]
```
⇒ elem_num_list に if 分岐に合致した元素記号を代入する。

```
    for i,elem in enumerate(elem_num_list) :
```
⇒ elem_num_list に入っているものを一つずつ elem に代入する。またループの回数を i に入力する。

```
        if read["LabelType"][0] == "1":
```
⇒設定ファイルの LabelType の値が 1 のとき if 分岐に入る

```
            elem_label += [str(i+1)]
```
⇒ elem_label にループの回転数を入れる

```
        if read["LabelType"][0] == "2":
```
⇒設定ファイルの LabelType の値が 2 のとき if 分岐に入る

```
            elem_label += [elem ]
```
⇒ elem_label に元素記号を elem_label に入れる

```
        if read["LabelType"][0] == "3":
```
⇒設定ファイルの LabelType の値が 3 のとき if 分岐に入る

```
            elem_label += [str(elem_num_list[0:i].count(elem)+1) + elem]
```
⇒元素ごとの数と、元素記号を elem_label に入れる

```
        if read["LabelType"][0] == "4":
```
⇒設定ファイルの LabelType の値が 4 のとき if 分岐に入る

```
            elem_label += [str(i+1)+ elem ]
```
⇒原子の通し番号と元素記号を elem_label に入れる

```
        else :
```
⇒それ以外の入力がある場合

```
            elem_label += [str(i+1)+ elem ]
```
⇒原子の通し番号と元素記号を elem_label に入れる

```
    for k in range(0,int(cube_list[2][1])*-1):
```
⇒ファイルに存在する原子の数だけループを回す。

```
        for m in range(0,117):
```
⇒ library.py に入っている要素の数 (117 個) だけループをまわす。

```
                out.write(lib.library.items()[m][0]+' ')
```
⇒ library.py に入っている原子番号を書く。

```
            else: continue
```
⇒ if 文の分岐に沿わなければ、ループを続行する。

```
            if l==6:
```
⇒「 l」が 6 の場合。

```
                out.write(elem_label[k]+'\n')
```
⇒座標にあるラベルを書き。改行する。

```
            else:
```
⇒「l」が 6 以外の場合。

```
                out.write(all_atom[k][l]+' ')
```
⇒ xyz 形式の z 座標と空白を書く。

Cube ファイルと書き込んだ xyz ファイルを閉じている。

```
    out.close()
```
⇒書き込んだ xyz ファイルを閉じる。

```
    f.close()
```
⇒読み込んだ Cube ファイルを閉じる。

読み込む Cube ファイルを再び開き、Cube ファイルから波動関数を抜き出して書き込むファイルを開く。

```
    f = open(filenametext,'r')
```
⇒ filenametext の変数に入っている Cube ファイルを読み取り専用で開く。

```
    out = open('convert_wave_function.vtk.txt','w')
```
⇒ convert_wave_function.vtk.txt を波動関数抜き出し用に開く。

Cube ファイルの先頭から波動関数までをスキップする。

```
    check = 0
```
⇒ check に 0 という値を入れる。

```
    check2 = 1
```
⇒ check2 に 1 という値を入れる。

```
    for n in range(0,7+int(cube_list[2][1])*-1):
```
⇒ Cube ファイルの設定部分＋原子数だけループを回す。

```
        f.readline()
```
⇒ファイルを 1 行読み飛ばす。

Cube ファイルの波動関数を読み取っている。上記「**L**」と「**mod**」を使って、**Cube** ファイルの **6** データ毎改行を変更する。そうすることで、**Cube** ファイルでは波動関数が **6** 行ごとに改行されていたが、波動関数を **VTK** ファイルの形式にあわせるため 1 行に出力することができる

```
    if int(cube_list[5][1])%6 == 0 :
```
⇒ z 軸のメッシュ数が 6 で割り切れる場合 if 分岐に入る

```
        L = int(cube_list[5][1]) / 6
```
⇒ z 軸のメッシュ数を 6 で割った数を L に代入する

```
    elif int(cube_list[5][1])%6 != 0:
```
⇒ z 軸のメッシュ数が 6 で割り切れない場合 elif 分岐に入る

```
        L = int(cube_list[5][1]) / 6 + 1
```
⇒ z 軸のメッシュ数を 6 で割り、 1 を足した数を L に代入する

```
    for xy in range(0, int(cube_list[3][1]) * int(cube_list[4][1])):
```
⇒ x 方向と y 方向のメッシュ数を掛けて、その回数分ループを回す。

```
        for line_num_in_xy in range(0, L):
```
⇒ z 方向のデータを読むのに必要なループの回数 L を回す

```
            line = f.readline()
```
⇒ファイルを一行読み込む

```
            line = line.strip()
```
⇒ line で読み込んだ文字列の先頭と末尾から文字を取り除いた文字列にする

```
            if line == "":  # Skip empty line.
```
⇒空行が存在する場合、 if 分岐に入る

```
                line = f.readline()
```
⇒ファイルを一行読み込む

```
                line = line.strip()
```
⇒ line で読み込んだ文字列の先頭と末尾から文字を取り除いた文字列にする

```
            line = " ".join(map(lambda s: str(float(s)), re.split(" *", line)))
```
⇒ 読み込んだデータを空行で区切り、浮動小数表示にし、スペースを区切り文字として一つの文

字列を作成する。

```
            out.write(line + " ")
```
⇒ line のデータとスペースを書き込む

```
        out.write("\n")
```
⇒改行文字を書き込む

```
    out.close()
```
⇒ out のファイルを閉じる

```
    f.close()
```
⇒ cube ファイルを閉じる。

波動関数の改行を変更した波動関数の可視化プログラムで読み込める **VTK** の形式をしたファイルを作成している。

```
    f = open('convert_wave_function.vtk.txt','r')
```
⇒波動関数を抜き出したファイルを開く。

```
    out = open('VTK_out.vtk','w')
```
⇒ VTK のファイルを書きだすファイルを作成する。

```
    total_line=[]
```
⇒ total_line の空リストを作成する。

```
    mirror = []
```
⇒ mirror の空リストを作成する。

```
    for j in range (0,int(cube_list[3][1])):
```
⇒ x 軸のグリッド数回分 j についてループを回す。

```
        all_line=[]
```
⇒ all_line の空リストを作成する。

```
        for i in range(0,int(cube_list[4][1])):
```
⇒ y 軸のグリッド数回分 i についてループを回す。

```
            line = f.readline()
```
⇒ファイルを 1 行読み込む。

```
            line = line.rstrip()
```
⇒ 1 行ずつ読んだ行末の空白文字を消す。

```
            line = re.sub("\n","",line)
```
⇒ 1 行ずつ読んだ文の改行を取り除く。

```
            line = re.split(" ",line)
```
⇒ 1 行ずつ読んだ行を空白で区切る。

```
            all_line += [line]
```
⇒加工しながら読んだ 1 行ずつのデータを all_line に入れる。

```
        total_line += [all_line]
```
⇒さらに all_line を total_line に入れる。

Cube ファイルをそのまま実行すると波動関数の符号が逆転する場合がある。この問題を解決するため、直前に実行したファイルの波動関数と現在読み込んでいる波動関数の内積を取って、プラスであれば、現在の波動関数をそのまま出力。マイナスであれば、現在の波動関数全てに－１を掛けて出力している。

```
    sum_wave = 0
```
⇒初期値 sum_wave に 0 を代入する。

```
    for i in range(0,int(cube_list[3][1])):
```
⇒ i を変数として、ループを回す。cube_list[3][1] は Cube ファイルの x 軸メッシュ数。

```
        for j in range(0,int(cube_list[4][1])):
```
⇒ j を変数として、ループを回す。cube_list[4][1] は Cube ファイルの y 軸メッシュ数。

```
            for k in range(0,int(cube_list[5][1])):
```
⇒ k を変数として、ループを回す。cube_list[5][1] は Cube ファイルの z 軸メッシュ数。

```
                total=total_line[i][j][k]
```
⇒現在の Cube ファイルの波動関数の値を total という変数に代入。

```
                check_wave=check_line[i][j][k]
```
⇒直前に実行した Cube ファイルの波動関数の値を check_wave という変数に代入。

```
                total=float(total)
```
⇒ total の値を全て浮動小数点型にする。

```
                check_wave=float(check_wave)
```
⇒ check_wave の値を全て浮動小数点型にする。

```
                sum_wave += total*check_wave
```
⇒ sum_wave に total と check_wave を掛け合わせたものを足し合わせる。

```
    if sum_wave >= 0:
```
⇒ sum_wave が足し合わさった後、その値が 0 以上ならば if 分岐に入る。

```
        print "plus",sum_wave
```
⇒ "plus"という文字を出力し、sum_wave の値を出力する。

```
    elif sum_wave < 0:
```
⇒ sum_wave の値が 0 未満ならば以下の if 分岐入る。

```
        print "minus",sum_wave
```
⇒ "minus"という文字を出力し、sum_wave の値を出力する。

```
        for i in range(0,int(cube_list[3][1])):
```
⇒ i を変数として、ループを回す。cube_list[3][1] は Cube ファイルの x 軸メッシュ数。

```
            for j in range(0,int(cube_list[4][1])):
```
⇒ j を変数として、ループを回す。cube_list[4][1] は Cube ファイルの y 軸メッシュ数。

```
                for k in range(0,int(cube_list[5][1])):
```
⇒ k を変数として、ループを回す。cube_list[5][1] は Cube ファイルの z 軸メッシュ数。

```
                    dammy=total_line[i][j][k]
```
⇒ dammy という変数に今回の波動関数の値を代入する。

```
                    dammy=float(dammy)*-1
```
⇒ dammy という変数に-1 を掛ける。

```
                    total_line[i][j][k]=str(dammy)
```
⇒ total_line に文字列として dammy を代入する。

```
    else :
```
⇒それ以外に if 分岐があった場合。

```
        print "NG"
```
⇒ NG と Python command line に表示させる。

VTK のフォーマットを書き、x 軸と z 軸の座標変換を行っている。

```
    out.write('# vtk DataFile Version 2.0'+'\n')
```
⇒ VTK ファイルフォーマットバージョン情報を、VTK ファイルに書き込む。

```
    out.write('Probability density for the 3d electron position in /
a hydrogen atom'+'\n')
```
⇒ファイルの内容を書き込んでいる。

```
    out.write('ASCII'+'\n')
```
⇒ VTK ファイルに ASCII 形式で書きこむ。(ASCII と BAINARY がある)

```
    out.write('DATASET STRUCTURED_POINTS'+'\n')
```
⇒データが STRUCTURED_POINTS の形式であることを書き込む。

```
    out.write('DIMENSIONS'+' '+cube_list[3][1]+' '+cube_list[4][1]+/
' '+cube_list[5][1]+'\n')
```
⇒各軸のグリッド数を書き込んでいる。「eigen_state_15-00000001.cube」の場合 (x 軸,y 軸,z 軸)=(80,80,80) である。

```
    out.write('ORIGIN'+' '+cube_list[2][2]+' '+cube_list[2][3]+/
' '+cube_list[2][4]+'\n')
```
⇒原点の位置を書き込んでいる。「eigen_state_15-00000001.cube」の場合、(x,y,z)=(-9.44863439,-9.44863439,-9.44863439)

```
    out.write('SPACING'+' '+cube_list[3][2]+' '+cube_list[4][3]+/
' '+cube_list[5][4]+'\n')
```
⇒グリッドの間隔を書き込んでいる。「eigen_state_15-00000001.cube」の場合 (x 軸,y 軸,z 軸)=(0.23621586,0.23621586,0.23621586) である。

```
    out.write('POINT_DATA'+' '+str(int(cube_list[3][1])*int(cube_list[4][1])*/
int(cube_list[5][1]))+' '+'\n')
```
⇒全てのデータ数はいくつか書き込んでいる。「eigen_state_15-00000001.cube」の場合、(x 軸のグリッド数 80) × (y 軸のグリッド数 80) × (z 軸のグリッド数 80)=『512000』となる。

```
    out.write('SCALARS probability_density float'+'\n')
```
⇒スカラー値で書き込むことを宣言する。

```
    out.write('LOOKUP_TABLE default'+'\n')
```
⇒ LOOKUP_TABLE default であることを指定する。

```
    for k in range (0,int(cube_list[5][1])):
```
⇒ z 軸のグリッド数でループを回す。

```
        for j in range (0,int(cube_list[4][1])):
```
⇒ y 軸のグリッド数でループを回す。

```
            for i in range (0,int(cube_list[3][1])):
```
⇒ x 軸のグリッド数でループを回す。

```
                if i==(int(cube_list[3][1])-1):
```
⇒ x 軸のグリッド数で回すループがデータの分割数、サンプルデータでは 80 回ループしたとき if 分岐に入る。

```
                    out.write(total_line[i][j][k]+'\n')
```
⇒波動関数 +改行を書き込む。

```
                else:
```
⇒ x 軸のグリッド数で回すループが分割数、サンプルデータでは 80 回目でないとき else 文に従う。

```
                    out.write(total_line[i][j][k]+' ')
```
⇒波動関数 +空白を書き込む。

```
    out.close()
```
⇒ VTK ファイルを閉じる。

```
    f.close()
```
⇒波動関数を変換したファイルを閉じる。

```
    print "end"
```
「end」と Python command line に表示する。

```
    return total_line
```
⇒ total_line を戻り値としてメインモジュールに戻す。

E.3　可視化プログラムのソースコード解説

必要なモジュールを適用する。

```
import os
```
⇒オペレーションシステムを参照。

```
import re
```
⇒正規表現モジュールを参照。

```
from vtk import*
```
⇒ VTK モジュールを参照。

```
import library as lib
```
⇒ library モジュールを lib と名付けて参照。

```
import math
```
⇒ math(数学関係のモジュール) を参照。

```
def setting_read():
```
⇒ setting_read と名付けた関数を定義する。

```
    pwd = os.getcwd()
```
⇒カレントディレクトリへのパスを pwd にする。

```
    files = os.listdir(pwd)
```
⇒ path(カレントディレクトリにある) 全てのファイルとサブディレクトリの名前からなるリストを返す。順番は不動。

```
    if "visbar_wb_setting_default.txt" in files >= 0:
```
⇒ visbar_wb_setting_default.txt のファイルがあった場合、if 分岐に従う

```
        setting_file_name="visbar_wb_setting_default.txt"
```
⇒ setting_file_name を visbar_wb_setting_default.txt とする

```
        f = open(setting_file_name,'r')
```
⇒ visbar_wb_setting_default.txt のテキストファイルを読み取り専用で開く。

```
        a=1
```
⇒ a に初期値 1 を入れる。

```
        cnt=0
```
⇒カウンタの初期値を 0 に設定。

```
        dic={}
```
⇒空のディクショナリを作成する。

```
        while True:
```
⇒ while ループを回す。

```
            findsharp=-1
```
⇒ findsharp の初期値を-1 に設定する。

```
            line = f.readline()
```
⇒ファイルを 1 行読み込む。

```
            if line.find("--end_setting--") >=0 :
```
⇒読み込んだ一行の中に「 --end_setting--」の一文が含まれていたら、直下にある break する。

```
                break
```
⇒ while 文から抜ける break を行う。

```
            all_line=[]
```
⇒ all_line という空のリストを作成。

```
            line = line.replace("\t"," ")
```
⇒読み込んだ一行の中にタブがあれば、それをスペースに変換する。間違ってタブがあっても区切れるようにするため。

```
            line = line.rstrip()
```
⇒末尾に空白文字がある場合、空白文字を削除する。

```
            line = re.sub("\n","",line)
```
⇒改行文字を取り消している。

```
            line = re.split(" *",line)
```
⇒空白で line の中の要素を分けている。

```
            if line[0] == "":
```
⇒ line の要素の一番目が何も無ければ、if 分岐の次の行 continue を実行する。

```
                continue
```
⇒ while ループの一番最初の文にループをもどす。

```
            if line[0].find("#") >=0:
```
⇒ line の要素の 1 番目にシャープが含まれていた場合、if 分岐に従い continue を実行する。

```
                continue
```
⇒ while ループの一番最初の文にループをもどす。

```
            all_line += line
```
⇒ all_line に line の要素を入れる。

```
            del all_line[0]
```
⇒ 0 番目の要素を削除する。

```
            for i in range(0,len(all_line)):
```
⇒ i という変数で all_line に入っている要素の数だけループを回す。

```
                if all_line[i].find("#") >=0:
```
⇒ all_line のどこかにシャープが入っていた場合、以下の if 分岐に従う。

```
                    findsharp=i
```
⇒ i の変数を findsharp として代入する。

```
            if findsharp >=0:
```
⇒ findsharp の値が 0 以上だった場合、if 分岐に従う。

```
                del all_line[findsharp:]
```
⇒ all_line のシャープが入っていた要素から後ろの要素全てを取り除く。

```
                if len(all_line) ==0:
```
⇒ all_line の要素が全く無かった場合、if 分岐に従う。

```
                    print "Dictionary isn't possible" +" "+ "["+line[0]+"]"
```
⇒ Dictionary isn't possible と、ディクショナリのキーを出力させる。

```
                    continue
```
⇒ while ループの一番最初の文にループをもどす。

```
            para=[]
```
⇒ para という空のリストを作成する。

```
            para += all_line
```
⇒ para に all_line を追加する。

```
            dic[line[0]]=para
```
⇒ディクショナリに line の一番目をディクショナリのキー、para を値として、ディクショナリに追加する。{line[0]:para} となり、ディクショナリに追加される。

```
        f.close()
```
⇒ visbar_wb_setting.txt を閉じる。

```
        if "visbar_wb_setting.txt" in files >= 0:
```
⇒ visbar_wb_setting.txt がディレクトリにあった場合、if 分岐に入る。

```
            setting_file_name="visbar_wb_setting.txt"
```
⇒ visbar_wb_setting.txt を setting_file_name という変数に入れる。

```
            f = open(setting_file_name,'r')
```
⇒ setting_file_name の変数に代入されている (visbar_wb_setting.txt) を読み取り専用で開く。

```
            a=1
```
⇒ a に 1 を代入する。

```
            cnt=0
```
⇒ cnt の変数に 0 を代入する。

```
            dic2={}
```
⇒ dic2 という空のディクショナリを作成する。

```
            while True:
```
⇒ while ループを回す。

```
                findsharp=-1
```
⇒ findsharp の初期値を-1 に設定する。

```
                line = f.readline()
```
⇒ファイルを 1 行読み込む。

```
                if line.find("--end_setting--") >=0 :
```
⇒読み込んだ 1 行の中に「--end_setting--」の一文が含まれていたら、直下にある break を実行する。

```
                    break
```

⇒ while 文から抜ける break を実行する。

```
                all_line=[]
```
⇒ all_line という空のリストを作成。

```
                line = line.replace("\t"," ")
```
⇒読み込んだ一行の中にタブがあれば、それをスペースに変換する。間違ってタブがあっても区切れるようにするため。

```
                line = line.rstrip()
```
⇒末尾に空白文字がある場合、空白文字を削除。

```
                line = re.sub("\n","",line)
```
⇒改行文字を取り消している。

```
                line = re.split(" *",line)
```
⇒空白で line の中の要素を分けている。

```
                if line[0] == "":
```
⇒ line の要素の一番目が何も無ければ、if 分岐の次の行 continue を実行する。

```
                    continue
```
⇒ while ループの一番最初の文にループをもどす。

```
                if line[0].find("#") >=0:
```
⇒ line の要素の 1 番目にシャープが含まれていた場合、if 分岐に従い continue を実行する。

```
                    continue
```
⇒ while ループの一番最初の文にループをもどす。

```
                all_line += line
```
⇒ all_line に line の要素を入れる。

```
                del all_line[0]
```
⇒ 0 番目の要素を削除する。

```
                for i in range(0,len(all_line)):
```
⇒ i という変数で all_line に入っている要素の数だけループを回す。

```
                    if all_line[i].find("#") >=0:
```
⇒ all_line のどこかにシャープが入っていた場合、以下の if 分岐に従う。

```
                        findsharp=i
```

⇒ i の変数を findsharp として代入する。

```
                if findsharp >=0:
```
⇒ findsharp の値が 0 以上だった場合、if 分岐に従う。

```
                    del all_line[findsharp:]
```
⇒ all_line のシャープが入っていた要素から後ろの要素全てを取り除く。

```
                    if len(all_line) ==0:
```
⇒ findsharp の値が 0 以上だった場合、if 分岐に従う。

```
                        print "Dictionary isn't possible" +" "+ "["+line[0]+"]"
```
⇒ Dictionary isn't possible と、ディクショナリのキーを出力させる。

```
                        continue
```
⇒ while ループの一番最初からループをやり直す。

```
                para=[]
```
⇒ para という空のリストを作成する。

```
                para += all_line
```
⇒ para に all_line を追加する。

```
                dic2[line[0]]=para
```
⇒ディクショナリに line の一番目をディクショナリのキー、para を値として、ディクショナリに追加する。{line[0]:para} となり、ディクショナリに追加される。

```
                dic.update(dic2)
```
⇒ dic をベースに dic2 のディクショナリを上書きしたディクショナリを dic として定義する。

```
    return dic
```
⇒ dic を戻り値とする。

```
read=setting_read()
```
⇒ setting_read() の関数を実行し、実行した値を read に入れる。

```
def setting_bond():
```
⇒ setting_bond の関数を定義する

```
    f = open("bond_length.txt",'r')
```
⇒ bond_length.txt のテキストファイルを読み取り専用で開く。

```
    a=1
```

⇒ a に初期値 1 を入れる。

```
    cnt=0
```
⇒カウンタの初期値を 0 に設定。

```
    dic={}
```
⇒空のディクショナリを作成する。

```
    while True:
```
⇒ while ループを回す。

```
        findsharp=-1
```
⇒ findsharp の初期値を-1 に設定する。

```
        line = f.readline()
```
⇒ファイルを 1 行読み込む。

```
        if line.find("--end_setting--") >=0 :
```
⇒読み込んだ一行の中に「 --end_setting--」の一文が含まれていたら、直下にある break する。

```
            break
```
⇒ while 文から抜ける break を行う。

```
        all_line=[]
```
⇒ all_line という空のリストを作成。

```
        line = line.replace("\t"," ")
```
⇒読み込んだ一行の中にタブがあれば、それをスペースに変換する。間違ってタブがあっても区切れるようにするため。

```
        line = line.rstrip()
```
⇒末尾に空白文字がある場合、空白文字を削除。

```
        line = re.sub("\n","",line)
```
⇒改行文字を取り消している。

```
        line = re.split(" *",line)
```
⇒空白で line の中の要素を分けている。

```
        if line[0] == "":
```
⇒ line の要素の一番目が何も無ければ、if 分岐の次の行 continue を実行する。

```
            continue
```
⇒ while ループの一番最初の文にループをもどす。

```
        if line[0].find("#") >=0:
```
⇒ line の要素の 1 番目にシャープが含まれていた場合、if 分岐に従い continue を実行する。

```
            continue
```
⇒ while ループの一番最初の文にループをもどす。

```
        all_line += line
```
⇒ all_line に line の要素を入れる。

```
        del all_line[0:2]
```
⇒ 0 番目,1 番目の要素を削除する。

```
        for i in range(0,len(all_line)):
```
⇒ i という変数で all_line に入っている要素の数だけループを回す。

```
            if all_line[i].find("#") >=0:
```
⇒ all_line のどこかにシャープが入っていた場合、以下の if 分岐に従う。

```
                findsharp=i
```
⇒ i の変数を findsharp として代入する。

```
        if findsharp >=0:
```
⇒ findsharp の値が 0 以上だった場合、if 分岐に従う。

```
            del all_line[findsharp:]
```
⇒ all_line のシャープが入っていた要素から後ろの要素全てを取り除く。

```
            if len(all_line) ==0:
```
⇒ all_line の要素が全く無かった場合、if 分岐に従う

```
                print "Dictionary isn't possible" +" "+ "["+line[0]+"]"
```
⇒ Dictionary isn't possible と、ディクショナリのキーを出力させる。

```
                continue
```
⇒ while ループの一番最初の文にループをもどす。

```
        dic[line[0],line[1]]=line[2]
```
⇒ディクショナリに line の一番目と二番目をディクショナリのキー、line の三番目を値として、ディクショナリに追加する。{line[0],line[1]]:line[2]} となり、ディクショナリに追加される。

```
    f.close()
```
⇒ bond_length.txt を閉じる。

```
retur   dic
```
⇒戻り値に dic を指定する。

```
read_bond=setting_bond()
```
⇒ setting_bond の戻り値を read_bond に代入する。

可視化時に必要な初期値を上に書いた setting_read() の関数から持ってくる。

```
save_step = 0
```
⇒ save_step の値に 0 を与える。

```
isovalue=float(read["IsoValuePositive"][0])
```
⇒ isovalue の初期値を setting ファイルのディクショナリから持ってくる。

```
stableiso=float(read["IsoValuePositive"][0])
```
⇒ stableiso の初期値を setting ファイルのディクショナリから持ってくる。

```
isovalueminus=float(read["IsoValueNegative"][0])
```
⇒ isovalueminus の初期値を setting ファイルのディクショナリから持ってくる。

```
stableisominus=float(read["IsoValueNegative"][0])
```
⇒ stableisominus の初期値を setting ファイルのディクショナリから持ってくる。

```
PositiveLevel=0
```
⇒ PositiveLevel の初期値を 0 とする。

```
NegativeLevel=0
```
⇒ NegativeLevel の初期値を 0 とする。

可視化時に「u」を押した時に png 形式で画像を保存するように関数を作成する save_step を画像を保存時に増える変数とした。

```
def vtk(output_dir,filenametext,Batch_mode):
```
⇒ vtk という関数を output_dir,filenametext,Batch_mode の引数を持って定義している。ここからが可視化プログラムの根幹である。

```
    save_step = 0
```
⇒ save_step という初期値を 0 に設定する。

```
    def userMethod(obj, arg):
```
⇒定義文で、以下の画像を保存するモジュールを定義する。

```
      global save_step
```
⇒ save_step を global に定義して、定義文の外と中で変数を受け渡しできるようにした。

```
      print "Save

Figure"
```
⇒この定義文を使用したときに「Save Figure」と表示されるようにする。

```
      isosurface.SetValue(0,stableiso)
```
⇒ isosurface.SetValue の値を設定する。

```
      isosurfaceminus.SetValue(0,stableisominus)
```
⇒ isosurfaceminus.SetValue の値を設定する。

```
      renWin.Render()
```
⇒ renWin の Render の関数を使用し、図を再描画する。

```
      windowToImageFilter=vtkWindowToImageFilter()
```
⇒ windowToImageFilter を vtkWindowToImageFilter() と定義する。

```
      windowToImageFilter.SetInput(renWin)
```
⇒ windowToImageFilter に renwin をインプットする。

```
      windowToImageFilter.Update()
```
⇒ windowToImageFilter をアップデートする。

```
      pngWriter=vtkPNGWriter()
```
⇒ pngWriter を vtkPNGWriter() として定義する。

```
      if VTK_MAJOR_VERSION <= 5:
```
⇒ VTK のバージョンが 5 以下ならば、if 分岐に入る

```
          pngWriter.SetInput(windowToImageFilter.GetOutput())
```
⇒ pngWriter に windowToImageFilter の出力をインプットする。ver5.*系の場合

```
      else:
```

⇒ VTK のバージョンが 6 以上ならば else 文に入る

```
        pngWriter.SetInputData(windowToImageFilter.GetOutput())
```
⇒ pngWriter に windowToImageFilter の出力をインプットする。ver6.*系の場合

```
    pngWriter.SetFileName("image"+ str(save_step) +".png")
```
⇒保存するファイルの名前を指定する。「image+番号+.png」の形式となっている。

```
    pngWriter.Write()
```
⇒ pngWriter の Write() 関数を用いて画像を保存する。

```
    save_step += 1
```
⇒ save_step の値に 1 を加える。

---

setting_read の関数で読み込むことの出来る setting ファイルを作成するモジュールを定義する。

```
  def setting_write():
```
⇒ setting_write の関数を定義する。

```
      print "Output Setting File"
```
⇒ "Output Setting File"の文字列を端末に表示する

```
      isosurface.SetValue(0,stableiso)
```
⇒ isosurface.SetValue の値を設定する。

```
      isosurfaceminus.SetValue(0,stableisominus)
```
⇒ isosurfaceminus.SetValue の値を設定する。

```
      renWin.Render()
```
⇒ renWin の Render の関数を使用し、図を再描画する。

```
      out = open("visbar_wb_setting_output.txt",'w')
```
⇒ visbar_wb_setting_output.txt を書き込み専用で開く。

```
      out.write('#---IsoValue---#' + '\n')
```
⇒ #---IsoValue---#の文字列をファイルに書き込む

```
      out.write('IsoValuePositive' +' '+ str(isovalue*(1.01**PositiveLevel)) + '\n')
```
⇒ IsoValuePositive の文字と、現在の正の波動関数の値を書き込む。

```
      out.write('IsoValueNegative'+' '+ str(isovalueminus*\
```

```
(1.01**NegativeLevel)) + '\n')
```
⇒ IsoValueNegative の文字と、現在の負の波動関数の値を書き込む。

```
out.write('IsoOpacityPositive'+' '+ str(isosurfaceActor.GetProperty()\
.GetOpacity()) + '\n')
```
⇒ IsoOpacityPositive の文字と正の波動関数の透明度の値を書き込む。

```
out.write('IsoOpacityNegative'+' '+ str(isosurfaceminusActor\
.GetProperty().GetOpacity()) + '\n')
```
⇒ IsoOpacityNegative の文字と負の波動関数の透明度の値を書き込む。

```
out.write('DrawIsoInWirePositive'+' '+ "Off" + '\n')
```
⇒ DrawIsoInWirePositive と正の波動関数のワイヤーフレーム、Off 設定で書き込む

```
out.write('DrawIsoInWireNegative'+' '+ "Off" + '\n')
```
⇒ DrawIsoInWireNegative と負の波動関数のワイヤーフレーム、Off 設定で書き込む

```
out.write('\n' + '#---Color---#' + '\n')
```
⇒ #---Color---#の文字列を書き込む

```
out.write('IsoColorPositive'+' '+ str(read["IsoColorPositive"][0])
                                  +' '+ str(read["IsoColorPositive"][1])
                                  +' '+ str(read["IsoColorPositive"][2])+ '\n')
```
⇒ IsoColorPositive の文字列と正の波動関数の色を RGB 形式で書き込む

```
        out.write('IsoColorNegative'+' '+ str(read["IsoColorNegative"][0])
                                  +' '+ str(read["IsoColorNegative"][1])
                                  +' '+ str(read["IsoColorNegative"][2])+ '\n')
```
⇒ IsoColorNegative の文字列と負の波動関数の色を RGB 形式で書き込む

```
        out.write('OutLineColor'+' '+ str( outlineActor.GetProperty()\
.GetAmbientColor()[0])\
                               +' '+ str( outlineActor.GetProperty()\
.GetAmbientColor()[1])\
                               +' '+ str( outlineActor.GetProperty()\
.GetAmbientColor()[2])+ '\n')
```
⇒ OutLineColor の文字列と外枠の色を RGB 形式で書き込む

```
        out.write('BackGroundColor'+' '+str(ren.GetBackground()[0])
                                  +' '+str(ren.GetBackground()[1])
                                  +' '+str(ren.GetBackground()[2])+ '\n')
```
⇒ BackGroundColor の文字列と背景の色を RGB 形式で書き込む

```
        out.write('\n' + '#---WindowSize---#' + '\n')
```
⇒ #---WindowSize---#の文字列を書き込む

```
        out.write('Window_size' +' '+ str(renWin.GetSize()[0])\
                                 +' '+ str(renWin.GetSize()[1])+ '\n')
```
⇒ Window_size の文字とウィンドウのサイズ (x,y) の座標を書き込む。

```
        out.write('\n' + '#---Camera---#' + '\n')
```
⇒ #---Camera---#の文字列を書き込む

```
        out.write('FocalPoint'+' '+str(camera.GetFocalPoint()[0])
                                 +' '+str(camera.GetFocalPoint()[1])
                                 +' '+str(camera.GetFocalPoint()[2])+ '\n')
```
⇒ FocalPoint の文字と FocalPoint の値を書き込む。

```
        out.write('CameraPosition'  +' '+str(camera.GetPosition()[0])
                                    +' '+str(camera.GetPosition()[1])
                                    +' '+str(camera.GetPosition()[2])+ '\n')
```
⇒ CameraPosition の文字列と Camera の位置の値を書き込む

```
        out.write('ParallelScale'+' '+str(camera.GetParallelScale())+ '\n')
```
⇒ ParallelScale の文字と ParallelScale の値を書き込んでいる。

```
        out.write('ViewUp'    +' '+str(camera.GetViewUp()[0])
                              +' '+str(camera.GetViewUp()[1])
                              +' '+str(camera.GetViewUp()[2])+ '\n')
```
⇒ ViewUp の文字と ViewUp の値を書き込んでいる。

```
        out.write('ClippingRange'+' '+str(camera.GetClippingRange()[0])
                                    +' '+str(camera.GetClippingRange()[1])+ '\n')
```
⇒ ClippingRange の文字と ClippingRange の値を書き込んでいる。

```
        out.write('\n' + '#---Label---#' + '\n')
```
⇒ #---Label---#の文字列を書き込む

```
        out.write('#NumberOnly:1' + '\n' +'#ElementOnly:2,' + '\n' +\
'#EachNumber+Element:3,' + '\n' +'#SerialNumber+Element:4,' + '\n' +\
 'LabelType 4' + '\n')
```
⇒括弧の中の文字列を書き込む

```
        out.write('\n' + '#---DrawObject---#' + '\n')
```
⇒ #---DrawObject---#の文字列を書き込む

```
        all_parts=["DrawBond","DrawAtom","DrawWaveFunction","DrawOutLine",\
"DrawAxis","DrawText","DrawAtomLabel","WatchParallelView"]
```
⇒⇒ all_parts の中に、リストの要素が入っている。

```
        isosurface.SetValue(0,stableiso)
```
⇒ isosurface.SetValue の値を設定する。

```
        isosurfaceminus.SetValue(0,stableisominus)
```
⇒ isosurfaceminus.SetValue の値を設定する。

```
        renWin.Render()
```
⇒ renWin の Render の関数を使用し、図を再描画する。

```
        for parts in all_parts:
```
⇒ all_parts の要素を parts に代入しながらループを回す。

```
            out.write(parts + " " + "On" +"\n")
```
⇒要素名と On の設定をファイルに書き込む。

```
        out.write('--end_setting--'+ '\n')
```
⇒ --end_setting--の文字を書き込んでいる。

```
        out.close()
```
⇒ visbar_wb_setting_output.txt のファイルを閉じる。

StructuredPoints(グリッド幅が一定) の形式で VTK_out.vtk のファイルを読み込む。

```
    reader = vtkStructuredPointsReader()
```
⇒ reader に vtkStructuredPointsReader() を定義する。StructuredPoints とは、均一・等間隔に並んだデータの集合体のことである。

```
    reader.SetFileName("VTK_out.vtk")
```
⇒ VTK_out.vtk のファイルを読み込む。

画面内の外枠を作成。

```
    outline = vtkOutlineFilter()
```

⇒ outline に vtkOutlineFilter() を定義する。

```
    outline.SetInputConnection(reader.GetOutputPort())
```
⇒ outline に reader で読み込んだデータをインプットする。

```
    outlineMapper = vtkPolyDataMapper()
```
⇒ outlineMapper に vtkPolyDataMapper() を定義する。

```
    outlineMapper.SetInputConnection(outline.GetOutputPort())
```
⇒ outlineMapper に outline のデータをインプットする。

```
    outlineActor = vtkActor()
```
⇒ outlineActor に vtkActor() を定義する。

```
    outlineActor.SetMapper(outlineMapper)
```
⇒ outlineActor に outlineMapper のデータを代入する。

```
    outlineActor.GetProperty().SetColor(float(read["OutLineColor"][0]),
                                        float(read["OutLineColor"][1]),
                                        float(read["OutLineColor"][2]))
```
⇒ outlineActor の色を指定している。RGB 形式なので、(R,G,B) の値を setting ファイルから持ってきている。

波動関数 (価電子) の値と色の対応を設定。

```
    isominusR=float(read["IsoColorPositive"][0])
```
⇒負の波動関数の RGB の R の値を setting ファイルから持ってきている。

```
    isominusG=float(read["IsoColorPositive"][1])
```
⇒負の波動関数の RGB の G の値を setting ファイルから持ってきている。

```
    isominusB=float(read["IsoColorPositive"][2])
```
⇒負の波動関数の RGB の B の値を setting ファイルから持ってきている。

```
    isoplusR=float(read["IsoColorNegative"][0])
```
⇒正の波動関数の RGB の R の値を setting ファイルから持ってきている。

```
    isoplusG=float(read["IsoColorNegative"][1])
```
⇒正の波動関数の RGB の G の値を setting ファイルから持ってきている。

```
    isoplusB=float(read["IsoColorNegative"][2])
```

⇒正の波動関数の RGB の B の値を setting ファイルから持ってきている。

```
    lut=vtkColorTransferFunction()
```
⇒スカラー値を色に変換する関数を lut として定義する。

```
    lut.AddRGBPoint(-0.1,isominusR,isominusG,isominusB)
```
⇒波動関数が -0.1 の値のとき RGB(isominusR,isominusG,isominusB) の色で表示する。
-------以下同様--------

```
    lut.AddRGBPoint(-0.075,isominusR,isominusG,isominusB)
    lut.AddRGBPoint(-0.05,isominusR,isominusG,isominusB)
    lut.AddRGBPoint(-0.025,isominusR,isominusG,isominusB)
    lut.AddRGBPoint(-0.001,isominusR,isominusG,isominusB)
    lut.AddRGBPoint(0,1,0,0)
    lut.AddRGBPoint(0.001,isoplusR,isoplusG,isoplusB)
    lut.AddRGBPoint(0.025,isoplusR,isoplusG,isoplusB)
    lut.AddRGBPoint(0.05,isoplusR,isoplusG,isoplusB)
    lut.AddRGBPoint(0.075,isoplusR,isoplusG,isoplusB)
    lut.AddRGBPoint(0.1,isoplusR,isoplusG,isoplusB)
```

---

global 変数を指定する。

```
    global isovalue
```
⇒ isovalue の値 global 変数として扱うことを宣言。

```
    global isovalueminus
```
⇒ isovalueminus の値 global 変数として扱うことを宣言。

表示する正の波動関数を設定し、描画オブジェクトに落とし込んでいる。

```
    isosurface = vtkContourFilter()
```
⇒ isosurface に vtkContourFilter() を定義している。

```
    isosurface.SetInputConnection(reader.GetOutputPort())
```
⇒ isosurface に reader のデータをインプットする。

```
    isosurface.SetValue(0,isovalue)
```
⇒ iosurface に一番最初に表示する値を設定する。

```
    isosurfaceMapper = vtkPolyDataMapper()
```

⇒ isosurfaceMapper に vtkPolyDataMapper() を定義する。

```
    isosurfaceMapper.SetLookupTable(lut)
```
⇒ isosurfaceMapper に lut の値にあった色づけする。

```
    isosurfaceMapper.SetInputConnection(isosurface.GetOutputPort())
```
⇒ isosurfaceMapper に isosurface のデータをインプットする。

```
    isosurfaceActor = vtkActor()
```
⇒ isosurfaceActor に vtkActor() を定義する。

```
    isosurfaceActor.SetMapper(isosurfaceMapper)
```
⇒ isosurfaceActor に isosurface をセットする。

```
    isosurfaceActor.GetProperty().SetOpacity(float(read["IsoOpacityPositive"][0]))
```
⇒波動関数の透明度の値を setting ファイルから持ってきてセットする。

```
    if read["DrawIsoInWirePositive"][0] == "On":
```
⇒ setting_file に書かれている DrawIsoInWirePositive の値が On ならば、if 分岐に入る。

```
        isosurfaceActor.GetProperty().SetRepresentationToWireframe()
```
⇒波動関数の等値面を wireframe 表示にする。

表示する負の波動関数を設定し、描画オブジェクトに落とし込んでいる。

⇒正の場合と同様

```
    isosurfaceminus = vtkContourFilter()
    isosurfaceminus.SetInputConnection(reader.GetOutputPort())
    isosurfaceminus.SetValue(0,isovalueminus)
    isosurfaceminusMapper = vtkPolyDataMapper()
    isosurfaceminusMapper.SetLookupTable(lut)
    isosurfaceminusMapper.SetInputConnection(isosurfaceminus.GetOutputPort())
    isosurfaceminusActor = vtkActor()
    isosurfaceminusActor.SetMapper(isosurfaceminusMapper)
    isosurfaceminusActor.GetProperty().SetOpacity(float(read["IsoOpacityNegative"][0]))
    if read["DrawIsoInWireNegative"][0] == "On":
        isosurfaceminusActor.GetProperty().SetRepresentationToWireframe()
```

描画エンジンを設定し、背景の色設定を行っている。

```
    ren = vtkRenderer()
```
⇒ ren に vtkRenderer() を定義している。

```
    ren.SetBackground(float(read["BackGroundColor"][0]),
                      float(read["BackGroundColor"][1]),
                      float(read["BackGroundColor"][2]))
```
⇒背景の色を設定している。背景の値を setting ファイルから持ってきてセットする。

---

xyz ファイルから、原子情報を読み出している。

```
    f=open('make_atom.xyz.txt','r')
```
⇒ xyz ファイルを読み取り専用で開く。

```
    atom_list=[]
```
⇒ atom_list の空リストを作成する。

```
    number=f.readline()
```
⇒ number という変数にファイルの一行目を代入する。ここでは、原子数が当てはまる。

```
    comment=f.readline()
```
⇒ comment という変数にファイルの二行目を代入する。ここでは、コメント文が当てはまる。

```
    for j in range(0,int(number)):
```
⇒ number(原子) の値だけループを回す。

```
        atom = f.readline()
```
⇒原子のデータを 1 行ずつ読み込む。

```
        atom = atom.rstrip()
```
⇒ 1 行ずつ読んだ行末の空白文字を消す。

```
        atom = re.sub("\n","",atom)
```
⇒ 1 行ずつ読んだ文の改行を取り除く。

```
        atom = re.split(" *",atom)
```
⇒ 1 行ずつ読んだ行を空白で区切る。

```
        atom_list +=[atom]
```

⇒加工しながら読んだ 1 行ずつのデータを atom_list に入れる。

```
    f.close()
```
⇒ xyz ファイルを閉じる。

---

xyz ファイルから読み出した原子座標を元に Sphere を原子球に見立てて描画している。

```
    for k in range(0,int(number)):
```
⇒原子数の数だけループを回している。

```
    # create source
        source0 = vtkSphereSource()
```
⇒ source0 に vtkSphereSource() を定義している。vtkSphereSource() は、sphere(球) を描く関数である。

```
        source0.SetCenter(float(atom_list[k][1]),float(atom_list[k][2]),/
float(atom_list[k][3]))
```
⇒球の座標を決めている。

```
        source0.SetRadius(float(lib.library[atom_list[k][0]][1]))
```
⇒球の半径を決めている。

```
        source0.SetThetaResolution(15)
```
⇒球体の縦方向の分割数を定めている。(スイカの縦しま模様の方向。縦しまの分割数を決めていると考えればよい。)

```
        source0.SetPhiResolution(15)
```
⇒球体の横方向の分割数を定めている。(スイカを指定した数輪切りにするように考える。)

```
    # mapper
        mapper0 = vtkPolyDataMapper()
```
⇒ mapper0 に vtkPolyDataMapper() を定義する。

```
      if VTK_MAJOR_VERSION <= 5:
```
⇒ VTK のバージョンが 5 以下ならば、if 分岐に入る

```
        mapper0.SetInput(source0.GetOutput())
```
⇒ mapper0 に sorce() のデータをインプットする。ver5.*系の場合。

```
      else:
```
⇒ VTK のバージョンが 6 以上ならば else 文に入る

```
          mapper0.SetInputConnection(source0.GetOutputPort())
```
⇒ mapper0 に sorce() のデータをインプットする。ver6.*系の場合。

```
    # actor
        actor0 = vtkActor()
```
⇒ actor0 に vtkActor() を定義する。

```
        actor0.SetMapper(mapper0)
```
⇒ actor0 に mapper0 をセットする。

```
        actor0.GetProperty().SetColor(lib.library[atom_list[k][0]][2][0]/255,/
lib.library[atom_list[k][0]][2][1]/255,lib.library[atom_list[k][0]][2][2]/255)
```
⇒ actor0 の色を library.py から色情報を RGB 形式で呼び出して、設定している。

```
        if read["DrawAtom"][0] == "On":
```
⇒設定で、 atom を On に設定している場合、直下の actor0 を描画するオブジェクトに加える。

```
            ren.AddActor(actor0)
```
⇒ ren に actor0 を加えている。

原子に貼り付けるラベルを設定している

```
    pd = vtkPolyData()
```
⇒ vtkPolyData() を pd という名前にする

```
    pts = vtkPoints()
```
⇒ vtkPoints() を pts という名前にする

```
    orient = vtkDoubleArray()
```
⇒ vtkDoubleArray() を orient という名前にする

```
    orient.SetName('orientation')
```
⇒ orient の名前を orientation とする

```
    label = vtkStringArray()
```
⇒ vtkStringArray() を label という名前にする

```
    label.SetName('label')
```
⇒ label の名前を'label' とする

```
    for k in range(0,int(number)):
```

⇒原子の数だけループを回す

```
        pts.InsertNextPoint(float(atom_list[k][1]),\
float(atom_list[k][2]),float(atom_list[k][3]))
```
⇒原子座標を pts に入力する

```
        label.InsertNextValue(str(atom_list[k][4]))
```
⇒ label に原子番号・元素記号を入力する

```
    pd.SetPoints(pts)
```
⇒ pd に pts のデータ入れる

```
    pd.GetPointData().AddArray(label)
```
⇒ pd が label のデータを参照する

```
    pd.GetPointData().AddArray(orient)
```
⇒ pd が orient のデータを参照する

```
    hier = vtkPointSetToLabelHierarchy()
```
⇒ vtkPointSetToLabelHierarchy() を hier という名前にする

```
    if VTK_MAJOR_VERSION <= 5:
```
⇒ VTK のバージョンが 5 以下ならば if 分岐に入る

```
        hier.SetInput(pd)
```
⇒ hire に pd を代入する

```
    else:
```
⇒ VTK のバージョンが 5 より上ならば else 分岐に入る

```
        hier.SetInputData(pd)
```
⇒ hire に pd を代入する

```
    hier.SetOrientationArrayName('orientation')
```
⇒ hier の OrientationArrayName を orientation とする

```
    hier.SetLabelArrayName('label')
```
⇒ hier の LabelArrayName を label とする

```
    hier.GetTextProperty().SetColor(0.0, 0.0, 0.0)
```
⇒ hier の色を指定している

```
    lmapper = vtkLabelPlacementMapper()
```

⇒ vtkLabelPlacementMapper() を lmapper という名前にする

```
    lmapper.SetInputConnection(hier.GetOutputPort())
```
⇒ lmapper に hier のデータを入力する

```
    lmapper.SetShapeToRoundedRect()
```
⇒不明・調査中

```
    lmapper.SetBackgroundColor(1.0, 1.0, 0.7)
```
⇒ lmapper の背景色を指定している

```
    lmapper.SetBackgroundOpacity(0.8)
```
⇒ lmapper の背景色透過度を指定している

```
    lmapper.SetMargin(3)
```
⇒不明・調査中

```
    lactor = vtkActor2D()
```
⇒ vtkActor2D() を lactor という名前にしている

```
    lactor.SetMapper(lmapper)
```
⇒ lactor に lmapper を入力している

```
    if read["DrawAtomLabel"][0] == "On":
```
⇒設定ファイルに DrawAtomLabel が On の場合 if 分岐に入る

```
        ren.AddActor(lactor)
```
⇒ ren に lactor を追加する

```
    all_atom=atom_list
```
⇒ atom_list に all_atom の値を代入する。

```
    for i in range(0,int(number)-1):
```
⇒ i の値を 0〜(number)-1 の値を代入してループする。

```
        for j in range(i+1,int(number)):
```
⇒ j の値を i+1 から number の値を代入してループする。

```
            all_atom[i][1]=float(all_atom[i][1])
```
⇒ all_atom の x 座標の値を浮動小数表示にする。

```
            all_atom[i][2]=float(all_atom[i][2])
```
⇒ all_atom の y 座標の値を浮動小数表示にする。

```
            all_atom[i][3]=float(all_atom[i][3])
```
⇒ all_atom の z 座標の値を浮動小数表示にする。

```
            all_atom[j][1]=float(all_atom[j][1])
```
⇒ all_atom の x 座標の値を浮動小数表示にする。

```
            all_atom[j][2]=float(all_atom[j][2])
```
⇒ all_atom の y 座標の値を浮動小数表示にする。

```
            all_atom[j][3]=float(all_atom[j][3])
```
⇒ all_atom の z 座標の値を浮動小数表示にする。

```
            length =  ((all_atom[j][1] - all_atom[i][1])**2.0\
                      +(all_atom[j][2] - all_atom[i][2])**2.0\
                      +(all_atom[j][3] - all_atom[i][3])**2.0)**0.5
```
⇒各原子の座標から距離を測定。

```
            if length <= float(read_bond[all_atom[i][0],all_atom[j][0]]):
```
⇒ bond_length に設定された結合の種類と、結合間距離よりも length の値が小さければ、if 分岐に入る。

```
                radius=0.1
```
⇒ radius の値を 0.1 と設定。

```
                res=10
```
⇒ res の値を 10 と設定。

```
                axis=[all_atom[i][1]-all_atom[j][1],\
all_atom[i][2]-all_atom[j][2],all_atom[i][3]-all_atom[j][3]]
```
⇒二つの原子の各座標ごとの距離を計算。

```
                pos=[(all_atom[i][1]+all_atom[j][1])/2.0,\
(all_atom[i][2]+all_atom[j][2])/2.0,(all_atom[i][3]+all_atom[j][3])/2.0]
```
⇒二つの原子の中点を計算。

```
                height=math.sqrt(axis[0]**2+axis[1]**2+axis[2]**2)
```
⇒二つの原子の距離を出力。

```
                theta = math.acos(axis[1] / height)
```
⇒結合をどの程度回転させるかを計算。

```
                v = v=[axis[2], 0, -axis[0]]
```
⇒ v に配列 [axis[2], 0, -axis[0] を代入。

```
                cylinder = vtkCylinderSource()
```
⇒ cylinder に vtkCylinderSource() という結合の基本情報を設定。

```
                cylinder.SetResolution(res)
```
⇒結合をいくつの面で表現するかを設定。

```
                cylinder.SetHeight(height)
```
⇒結合の長さを設定。

```
                cylinder.SetRadius(radius)
```
⇒結合の半径を設定。

```
                mapper = vtkPolyDataMapper()
```
⇒ mapper に vtkPolyDataMapper() を設定。

```
                if VTK_MAJOR_VERSION <= 5:
```
⇒ VTK のバージョンが 5 以下ならば、if 分岐に入る

```
                    mapper.SetInput(cylinder.GetOutput())
```
⇒ mapper のインプットに cylinder.GetOutput() を入れる。ver5.*系の場合

```
                else:
```
⇒ VTK のバージョンが 6 以上ならば else 文に入る

```
                    mapper.SetInputData(cylinder.GetOutput())
```
⇒ mapper のインプットに cylinder.GetOutput() を入れる。ver6.*系の場合

```
                mapper.SetInput(cylinder.GetOutput())
```
⇒ mapper のインプットに cylinder.GetOutput() を入れる。

```
                actor = vtkActor()
```
⇒ actor に vtkActor() を設定。

```
                actor.SetMapper(mapper)
```
⇒ actor に mapper を設定。

```
                actor.RotateWXYZ(theta / math.pi * 180, v[0], v[1], v[2])
```
⇒ actor の回転度を設定。

```
                actor.SetPosition(pos)
```
⇒ actor の存在する position を設定。

```
                if read["bond"][0] == "On":
```
⇒ bond の設定 On ならば if 分岐に入る。

```
                    ren.AddActor(actor)
```
⇒ actor を描画するオブジェクトに追加する。

視点を設定している。

```
    camera=vtkCamera()
```
⇒ camera に vtkCamera() を定義する。

```
    camera.SetFocalPoint(float(read["FocalPoint"][0]),
                         float(read["FocalPoint"][1]),
                         float(read["FocalPoint"][2]))
```
⇒ camera の中心軸からどれだけずらすかを setting ファイルから参照して設定している。

```
    camera.SetPosition(float(read["CameraPosition"][0]),
                       float(read["CameraPosition"][1]),
                       float(read["CameraPosition"][2]))
```
⇒ camera をどの位置から見るかを setting ファイルから値 (x,y,z) を参照して設定している。

```
    camera.ComputeViewPlaneNormal
```
⇒ camera を ComputeViewPlaneNormal で表示する。

```
    camera.SetParallelScale(float(read["ParallelScale"][0]))
```
⇒ parallelscale の値を setting ファイルから値を参照して設定している。

```
    camera.SetViewUp(float(read["ViewUp"][0]),
                     float(read["ViewUp"][1]),
                     float(read["ViewUp"][2]))
```
⇒ camera を ……? 現在調べ中。

```
    camera.UseHorizontalViewAngleOff
```
⇒ camera を ····? 現在調べ中。

```
    camera.SetClippingRange(float(read["ClippingRange"][0]),
                            float(read["ClippingRange"][1]))
```
⇒ camera の ClippingRange を設定する。現在調べ中。

```
    ren.SetActiveCamera(camera)
```
⇒ ren に camera をセットする。

```
    if read["WatchParallelView"][0] == "On":
```
⇒ WatchParallelView の設定が On ならば、if 分岐に入る。

```
        ren.GetActiveCamera().ParallelProjectionOn()
```
⇒パラレルビューを On にした描画に設定する。

表示する **Window** の上部に文字を挿入している。

```
    txt = vtkTextActor()
```
⇒ txt に vtkTextActor() を定義する。

```
    txt.SetInput(comment)
```
⇒ txt に comment のデータを代入する。

```
    txtprop=txt.GetTextProperty()
```
⇒ txtprop に txt.GetTextProperty() を定義する。

```
    txtprop.SetFontFamilyToArial()
```
⇒画面に表示するテキストのフォントを Arial と指定する。

```
    txtprop.SetFontSize(16)
```
⇒文字のフォントサイズを設定する。

```
    txtprop.SetColor(0,0,0)
```
⇒文字の色を指定している。

```
    txt.SetDisplayPosition(30,450)
```
⇒文字を画面に出す位置を設定している。

```
    comment2=filenametext
```
⇒ comment2 に実行している Cube ファイルの名前を入れる。

```
    txt2 = vtkTextActor()
```
⇒ txt2 を vtkTextActor() にする。

```
    txt2.SetInput(comment2)
```
⇒ txt2 に comment2 を入力する。

```
    txtprop2=txt2.GetTextProperty()
```
⇒ txtprop2 を GetTextProperty() にする。

```
    txtprop2.SetFontFamilyToArial()
```
⇒テキストの字体を Arial にしている。

```
    txtprop2.SetFontSize(10)
```
⇒テキストのフォントサイズを指定している。

```
    txtprop2.SetColor(0,0,0)
```
⇒テキストの色を指定している。

```
    ren.AddActor(txt2)
```
⇒ txt2 を Actor に追加。

```
    text_representation = vtkTextRepresentation()
```
⇒ vtkTextRepresentation() を text_representation という名前で定義。

```
    text_representation.GetPositionCoordinate().SetValue(0.05, 0.85)
```
⇒調査中。

```
    text_representation.GetPosition2Coordinate().SetValue(0.8, 0.1)
```
⇒調査中。

```
    text_widget = vtkTextWidget()
```
⇒ vtkTextWidget() を text_widget という名で定義。

```
    text_widget.SetRepresentation(text_representation)
```
⇒ text_widget.SetRepresentation に text_representation を入力。

設定した外枠、正負共に波動関数、を描画エンジンに加え、表示方法を設定する。

```
    if read["DrawOutLine"][0] == "On":
```
⇒ DrawOutLine の設定を On にしている場合、if 分岐に入る。

```
            ren.AddActor(outlineActor)
```

⇒ ren に outline を加える。

```
    if read["DrawWaveFunction"][0] == "On":
```
⇒ DrawWaveFunction の設定を On にしている場合、if 分岐に入る。

```
        ren.AddActor(isosurfaceActor)
```
⇒ ren に isosurfaceActor を加える。

```
        ren.AddActor(isosurfaceminusActor)
```
⇒ ren に isosurfaceminusActor を加える。

```
    renWin = vtkRenderWindow()
```
⇒ renWin に vtkRenderWindow() を定義する。

```
    renWin.SetWindowName("VisBAR wave batch")
```
⇒ renWin の名前を「VisBAR wave batch」とする。

```
    renWin.SetSize(int(read["Window_size"][0]),
                   int(read["Window_size"][1]))
```
⇒表示する Window のサイズを整数型の数値で指定している。ウィンドウのサイズの値を setting ファイルから参照している。

```
    renWin.AddRenderer(ren)
```
⇒ renWin に ren を加える。

指定したキーを押すと、波動関数の等値面における基準値を増減させる関数を作成する。

```
    def Keypress(obj, event):
```
⇒定義文で波動関数の等値面の基準値を増減させる関数を定義する。

```
        global isovalue,isovalueminus,PositiveLevel,\
NegativeLevel,stableiso,stableisominus
```
⇒定義文の外と内で共通の変数として「 isovalue,isovalueminus, PositiveLevel,NegativeLevel,stableiso,stableisominus」を指定する。

```
        key = obj.GetKeySym()
```
⇒ key に obj.GetKeySym() を指定する。

```
        if key == "4":
```

⇒キーボードの 4 を押したとき。

```
            PositiveLevel +=1
```

⇒ PositiveLevel の値に 1 を追加する。

```
            stableiso = isovalue*(1.01**PositiveLevel)
```

⇒ isovalue に 1.01 に lebel を乗じたものを掛けて、stableiso に代入している。

```
            print  "-----------------------------------"
            print  "[PositiveLevel,PositiveValue =" ,PositiveLevel,",",stableiso,"]"
            print  "[NegativeLevel,NegativeValue =" ,NegativeLevel,",",\stableisominus,"
```

⇒区切り線、 PositiveLevel,PositiveValue,NegativeLevel,NegativeValue を端末に表示
している。

----------以下同様----------
ただし、「0」を押した場合デフォルトで設定した isovalue にリセットされる。

```
        elif key == "5":
            PositiveLevel -= 1
            stableiso = isovalue*(1.01**PositiveLevel)
            print  "-----------------------------------"
            print  "[PositiveLevel,PositiveValue =" ,PositiveLevel,",",stableiso,"]"
            print  "[NegativeLevel,NegativeValue =" ,NegativeLevel,",",stableisominus,"]
        elif key == "6":
            NegativeLevel += 1
            stableisominus = isovalueminus*(1.01**NegativeLevel)
            print  "-----------------------------------"
            print  "[PositiveLevel,PositiveValue =" ,PositiveLevel,",",stableiso,"]"
            print  "[NegativeLevel,NegativeValue =" ,NegativeLevel,",",stableisominus,"]
        elif key == "7":
            NegativeLevel -= 1
            stableisominus = isovalueminus*(1.01**NegativeLevel)
            print  "-----------------------------------"
            print  "[PositiveLevel,PositiveValue =" ,PositiveLevel,",",stableiso,"]"
            print  "[NegativeLevel,NegativeValue =" ,NegativeLevel,",",stableisominus,"]
        elif key == "1":
            PositiveLevel += 1
            NegativeLevel += 1
            stableiso = isovalue*(1.01**PositiveLevel)
            stableisominus = isovalueminus*(1.01**NegativeLevel)
            print  "-----------------------------------"
            print  "[PositiveLevel,PositiveValue =" ,PositiveLevel,",",stableiso,"]"
            print  "[NegativeLevel,NegativeValue =" ,NegativeLevel,",",stableisominus,"]
```

```
        elif key == "2":
            PositiveLevel -= 1
            NegativeLevel -= 1
            stableiso = isovalue*(1.01**PositiveLevel)
            stableisominus = isovalueminus*(1.01**NegativeLevel)
            print  "-----------------------------------"
            print  "[PositiveLevel,PositiveValue =" ,PositiveLevel,",",stableiso,"]"
            print  "[NegativeLevel,NegativeValue =" ,NegativeLevel,",",stableisominus,"]
        elif key == "0":
            PositiveLevel = 0
            NegativeLevel = 0
            stableiso = isovalue
            stableisominus = isovalueminus
            print  "-----------------------------------"
            print  "[PositiveLevel,PositiveValue =" ,PositiveLevel,",",stableiso,"]"
            print  "[NegativeLevel,NegativeValue =" ,NegativeLevel,",",stableisominus,"]

        elif key == "C":
```

⇒「shift+c」が押されたとき、カメラの情報が出力される。開発用。

```
            print camera
```

⇒ camera の値を出力する

```
        elif key == "x":
```

⇒「x」が押されたとき、

```
            camera.SetPosition(45.0,0.0,0.0)
```

⇒ camera の position を設定する

```
            camera.SetViewUp(0.0,1.0,0.0)
```

⇒ camera の ViewUp を設定する

```
            camera.SetFocalPoint(0.0,-0.0382357,0.0)
```

⇒ camera の FocalPoint を設定する

```
            renWin.Render()
```

⇒再描画する。

----------以下同様----------

```
        elif key == "y":
            camera.SetPosition(0.0,45.0,0.0)
            camera.SetViewUp(0.0,0.0,-1.0)
            camera.SetFocalPoint(0.0,-0.0382357,0.0)
```

```
            renWin.Render()
        elif key == "z":
            camera.SetPosition(0.0,0.0,45.0)
            camera.SetViewUp(0.0,1.0,0.0)
            camera.SetFocalPoint(0.0,-0.0382357,0.0)
            renWin.Render()

        elif key == "c":
```
⇒キーボードの c を押したとき。

```
            setting_write()
```
⇒ setting_write() の関数を起動し、カレントディレクトリに setting ファイルを出力する。

```
        elif key == "Q":
```
⇒「shift+q」が押されたとき、elif 分岐に入る

```
            exit()
```
⇒プログラムを終了する。

```
        elif key == "h":
            print  "-----------------------------------"
            print  "<<< How to Use VisBAR Wave Batch>>>"
            print  "-----------------------------------"
            print
            print  "< Exit >"
            print  "[Shift + q] Exit VisBAR Wave Batch"
            print  "[e or q] Close current Window"
            print
            print  "< Mouse >"
            print  "[Left Click] Rotate Object"
            print  "[Right Click or Mouse wheel]  Zooming"
            print  "[Press Mouse wheel] Parallel translation"
            print
            print  "< KeyBoard >"
            print  "[j] Joystick mode, [t] Trackball mode"
            print  "[w] Wireframe mode, [s] Surface mode"
            print  "[r] Reset Scaling"
            print  "[u] Save current figure to .png"
            print  "[a] Select individual object"
            print  "[x] Set camera direction to X axis"
            print  "[y] Set camera direction to Y axis"
            print  "[z] Set camera direction to Z axis"
            print
```

```
            print  "< Customize >"
            print  "Press [c] Output Customize file. Filename is\
 [visbar_wb_setting_output.txt]"
            print  "Edit this file. And then, File rename [visbar_wb_setting.txt].\
  Restart VisBAR Wave Batch."
            print
            print  "< Isovalue >"
            print  "[1] PositiveLevel + 1 ,NegativeLevel + 1"
            print  "[2] PositiveLevel - 1 ,NegativeLevel - 1"
            print  "[4] PositiveLevel + 1"
            print  "[5] PositiveLevel - 1"
            print  "[6] NegativeLevel + 1"
            print  "[7] NegativeLevel - 1"
            print  "[0] Reset to the default Isovalue"
```

⇒「h」が押されたときにヘルプを表示する。

```
        isosurface.SetValue(0,stableiso)
```

⇒ isosurface.SetValue の値を設定する。

```
        isosurfaceminus.SetValue(0,stableisominus)
```

⇒ isosurfaceminus.SetValue の値を設定する。

```
        renWin.Render()
```

⇒ renWin の Render の関数を使用し、図を再描画する。

ここまでに作成した描画オブジェクトを描画エンジンで実行する。

```
    if Batch_mode == "False" :
```

⇒ Batch_mode が False ならば、if 分岐に従う。

```
        w2if = vtkWindowToImageFilter()
```

⇒ vtkWindowToImageFilter() を w2if という名前にする。

```
        w2if.SetInput(renWin)
```

⇒ w2if に renWin を代入する。

```
        writer = vtkPNGWriter()
```

⇒ vtkPNGWriter() を writer という名前にする。

```
        if VTK_MAJOR_VERSION <= 5:
```
⇒ VTK のバージョンが 5 以下ならば、if 分岐に入る

```
            writer.SetInput(w2if.GetOutput())
```
⇒ writer に w2if のアウトプットを入れる。ver5.*系の場合

```
        else:
```
⇒ VTK のバージョンが 6 以上ならば else 文に入る

```
          writer.SetInputConnection(w2if.GetOutputPort())
```
⇒ writer に w2if のアウトプットを入れる。ver6.*系の場合

```
        dst = filenametext.replace(".cube", ".png")
```
⇒ filenametext の「.Cube」という文字列を「.png」に替えて dst に代入する

```
        writer.SetFileName(os.path.join(output_dir, dst))
```
⇒出力される画像ファイルの名前を output_dir の path と dst に入っているファイル名をくっつける。

```
        writer.Write()
```
⇒ writer を実行する。

```
       iren = vtkRenderWindowInteractor()
```
⇒ iren に vtkRenderWindowInteractor() の関数を加える。

```
       iren.AddObserver("UserEvent", userMethod)
```
⇒ png 保存する定義文を iren に加えている。

```
       iren.SetRenderWindow(renWin)
```
⇒ iren に renWin をセットする。

```
       iren.AddObserver("KeyPressEvent", Keypress)
```
⇒ iren に波動関数を増減させる定義文を加える。

```
        if read["DrawAxis"][0] == "On":
```
⇒設定ファイルの DrawAxis が On の場合 if 分岐に入る

```
            transform = vtkTransform()
```
⇒ vtkTransform() を transform という名前にする

```
            transform.Scale(5,5,5)
```
⇒軸の矢印の大きさを設定している

```
            transform.Translate(0.0,0.0,0.0)
```
⇒軸の矢印の原点を設定している

```
            axes = vtkAxesActor()
```
⇒ vtkAxesActor() を axes という名前にしている

```
            axes.GetXAxisCaptionActor2D().GetCaptionTextProperty().SetColor(0.0,0.0,0.0)
```
⇒ x の文字の色を黒色に設定

```
            axes.GetYAxisCaptionActor2D().GetCaptionTextProperty().SetColor(0.0,0.0,0.0)
```
⇒ y の文字の色を黒色に設定

```
            axes.GetZAxisCaptionActor2D().GetCaptionTextProperty().SetColor(0.0,0.0,0.0)
```
⇒ z の文字の色を黒色に設定

```
            axes.SetUserTransform(transform)
```
⇒ axes に transform を代入

```
            axesWidget = vtkOrientationMarkerWidget()
```
⇒ vtkOrientationMarkerWidget() の名前を axesWidget とする

```
            axesWidget.SetOrientationMarker(axes)
```
⇒ axesWidget に axes を入力する

```
            axesWidget.SetInteractor(iren)
```
⇒ axesWidget に iren を入力する

```
            axesWidget.EnabledOn()
```
⇒不明・調査中

```
            axesWidget.InteractiveOn()
```
⇒不明調査中

```
        text_widget = vtkTextWidget()
```
⇒ vtkTextWidget() を text_widget という名前にする。

```
        text_widget.SetRepresentation(text_representation)
```
⇒ text_widget.SetRepresentation に text_representation をセットする。

```
        text_widget.SetInteractor(iren)
```
⇒ text_widget.SetInteractor に iren をセットする。

```
        text_widget.SetTextActor(txt)
```

⇒ text_widget に txt をセットする。

```
        text_widget.SelectableOff()
```
⇒調査中。

```
        if read["DrawText"][0] == "On":
```
⇒設定ファイルの text の項目が On ならば、if 分岐に入る。

```
            text_widget.On()
```
⇒ text_widget を適用する。

```
       iren.Initialize()
```
⇒ iren の Initialize() 関数を使用する。

```
       iren.Start()
```
⇒ VTK を実行する。

```
    elif Batch_mode == "True" :
```
⇒ Batch_mode が True の場合、if 分岐に従う。

```
        if read["DrawText"][0] == "On":
```
⇒設定ファイルの text の欄が On ならば、if 分岐に従う。

```
            ren.AddActor(txt)
```
⇒コメントを表示する actor を画面に表示するオブジェクトに加える。

```
        w2if = vtkWindowToImageFilter()
```
⇒ vtkWindowToImageFilter() を w2if という名前にする。

```
        w2if.SetInput(renWin)
```
⇒ w2if に renWin を代入する。

```
        writer = vtkPNGWriter()
```
⇒ vtkPNGWriter() を writer という名前にする。

```
        if VTK_MAJOR_VERSION <= 5:
```
⇒ VTK のバージョンが 5 以下ならば、if 分岐に入る

```
            writer.SetInput(w2if.GetOutput())
```
⇒ writer に w2if のアウトプットを入れる。ver5.*系の場合

```
        else:
```
⇒ VTK のバージョンが 6 以上ならば else 文に入る

```
            writer.SetInputConnection(w2if.GetOutputPort())
```
⇒ writer に w2if のアウトプットを入れる。ver6.*系の場合

```
        dst = filenametext.replace(".cube", ".png")
```
⇒ filenametext の「.Cube」という文字列を「.png」に替えて dst に代入する

```
        writer.SetFileName(os.path.join(output_dir, dst))
```
⇒出力される画像ファイルの名前を output_dir の path と dst に入っているファイル名をくっつける。

```
        writer.Write()
```
⇒ writer を実行する。

```
        print "mode Batch_mode"
```
Python command line に Batch_mode と出力する。

```
    else :
```
⇒ if 分岐が a でも、 c でもなければ、以下の else 分岐に従う。

```
        print "An unforeseen error"
```
⇒ Python command line に An unforeseen error と出力する。

E.4　原子情報ファイル　library.py の解説

ファイルの内容は以下になっている。ここでは水素 H からカルシウム Ca までを書いているが、ファイルには 118 個分の原子情報が載っている。
原子情報ファイルは Python のディクショナリー機能を使用している。ディクショナリー機能とは、値に対応するキーが設定されており、キーを指定すると対応する値が返って出される機能である。

'H':[1,0.37,(0.,0.,255.)] が水素の一部分である。
'H':[1,0.37,(0.,0.,255.)]=' 元素記号':[原子番号, 原子半径,(色指定 RGB 形式)]
このように書かれている。

```
# -*- coding: cp932 -*-
library={'H':[1,0.37,(0.,0.,255.)],
         'He':[2,1.5,(0.,255.,0.)],
         'Li':[3,1.52,(82.,82.,255.)],
         'Be':[4,1.13,(128.,0.,255.)],
         'B':[5,0.9,(230.,230.,10.)],
         'C':[6,0.77,(80.,80.,80.)],
         'N':[7,0.53,(255.,128.,0.)],
         'O':[8,0.61,(255.,0.,128.)],
         'F':[9,0.71,(0.,255.,128)],
         'Ne':[10,1.59,(70.,255.,70.)],
         'Na':[11,1.86,(122.,122.,255.)],
         'Mg':[12,1.6,(128.,64.,192.)],
         'Al':[13,1.43,(255.,210.,0.)],
         'Si':[14,1.17,(120.,120.,120.)],
         'P':[15,1.09,(192.,128.,64.)],
         'S':[16,1.02,(192.,64.,128.)],
         'Cl':[17,1.01,(64.,192.,128.)],
         'Ar':[18,1.91,(140.,255.,140.)],
         'K':[19,2.26,(160.,160.,255.)],
         'Ca':[20,1.97,(128.,80.,150.)],
```

以下同様にデータが続く。

付 録F　開発履歴

F.1　v0.9.4 (2013年9月11日,9月22日)

・初公開。
・ファイル名修正のみ。機能変化なし。(v0.9.4r2, 2013年9月22日)

＜機能まとめ＞

- 波動関数・原子構造・原子結合を描画可能。
- 波動関数の透明度設定あり。
- 波動関数のインタラクティブな変動が可能。
- 波動関数のデフォルトの値を± 0.02に設定。
- 複数のCubeファイルを連続でCubeファイルに出来る。
- 背景は白色、外枠青色。
- ワイヤーフレーム表示が出来る。
- 「透視図法」PerspectiveViewと「投影図法」ProjectionViewの設定が可能。
- 画像の保存が出来る。連番で保存も可能。保存するタイミングはウィンドウを閉じたとき&キーボードでの操作時。
- テキストを画面に書き込めて、画面内に移動が可能。
- どのCubeファイルを実行したかLogを出力できる。
- 原子球、結合、波動関数、外枠を描画する・しないを選択できる。
- 視点の変更が可能。
- 結合の種類、結合距離を設定できる。
- 可視化の設定をユーザーが設定できる。

F.2　v0.9.5 (2013年11月14日)

・周期境界条件をプログラムに適用。ユーザー選択可

F.3　v1.0.0 (2014年02月12日)

【bugfix】
・Window size の引数を実数型から整数型へ変更をソースコードに反映。
謝辞：高木博和氏（筑波大学)。詳細説明：(F.3.1 の内容)

F.3.1　Window sizeの引数を実数型から整数型へ変更

Linux で Python のバージョンは 2.7.2、VTK（python-VTK）のバージョンは 5.6.1、numpy のバー ジョンは 1.5.1 の環境下において、可視化モジュール「visualize_isosurface.py」で以下のエラーが発生することが報告された。

```
eigen_state_15-00000001.cube
end
Traceback (most recent call last):
File "VisBAR_wave_batch.py", line 24, in <module>
view.vtk(filenametext,Batch_mode)
File
"/home/user/tmp/VisBAR_wave_batch_package_v0.9.4_20130911/\
visualize_isosurface.py",
line 512, in vtk
float(read["Window_size"][1]))
TypeError: function takes exactly 1 argument (2 given)
```

調査の結果、このエラーは可視化モジュール「visualize_isosurface.py」の 511,512 行目の

```
   renWin.SetSize(float(read["Window_size"][0]),
                  float(read["Window_size"][1]))
```

この部分で、エラーが発生しており、TypeError より「本来一つの引数を受け取る関数に二つの引数を与えていることによるもの」と判断できた。

更に調査を行い、renWin.SetSize で与える変数は実数型ではなく、整数型である必要がわかったので、エラーの出た部分を

```
   renWin.SetSize(int(read["Window_size"][0]),
                  int(read["Window_size"][1]))
```

と変更することで、与える引数を整数型にし、実行が出来るようにソースコードの修正を行った。

F.4　v1.0.1 (2014年11月05日)

【bugfix】
・原子番号二桁の Cube ファイルに原子番号の前の空白が無いことが原因でエラー発生した。そのエラーを、原子番号の前空白が無い場合に、空白を入れることで解決した。
下記の変更内容を (formatconvert.py 41 行目、269 行目付近) 該当箇所二つに適用した。

```
↓↓↓↓変更前↓↓↓↓↓
    all_atom=[]
    for j in range(0,new_cube):
        atomxyz = f.readline()
        atomxyz = atomxyz.rstrip()
        atomxyz = re.sub("\n","",atomxyz)
        atomxyz = re.split(" *",atomxyz)
        all_atom +=[atomxyz]

↓↓↓↓変更後↓↓↓↓↓
    all_atom=[]
    for j in range(0,new_cube):
        atomxyz = f.readline()
        atomxyz = atomxyz.rstrip()
        atomxyz = re.sub("\n","",atomxyz)
        atomxyz = re.split(" *",atomxyz)
        if atomxyz[0] != "":
            atomxyz = [" "] + atomxyz
        all_atom +=[atomxyz]
```

F.5　v1.0.2 (2014年12月10日)

・本マニュアルにソースコードを掲載。

・2.1 に vtkpython のインストール・実行方法を掲載。

・ファイルを閉じるコマンド close() に「()」が抜けていたのを修正。13 箇所。

・Numpy のパッケージを使用せずに VisBAR wave batch を実行できるように修正。419 行目、422 行目周辺。

```
↓↓↓↓変更前↓↓↓↓↓
axis=[all_atom[i][1]-all_atom[j][1],all_atom[i][2]-all_atom[j][2],\
all_atom[i][3]-all_atom[j][3]]
```

```
pos=[(all_atom[i][1]+all_atom[j][1])/2.0,\
(all_atom[i][2]+all_atom[j][2])/2.0,(all_atom[i][3]+all_atom[j][3])/2.0]

height=math.sqrt(axis[0]**2+axis[1]**2+axis[2]**2)

theta = math.acos(axis[1] / np.linalg.norm(axis))

v = np.array([axis[2], 0, -axis[0]])



↓↓↓↓変更後↓↓↓↓↓
axis=[all_atom[i][1]-all_atom[j][1],all_atom[i][2]-all_atom[j][2],\
all_atom[i][3]-all_atom[j][3]]

pos=[(all_atom[i][1]+all_atom[j][1])/2.0,\
(all_atom[i][2]+all_atom[j][2])/2.0,(all_atom[i][3]+all_atom[j][3])/2.0]

height=math.sqrt(axis[0]**2+axis[1]**2+axis[2]**2)

theta = math.acos(axis[1] / height)

v=[axis[2], 0, -axis[0]]
```

・VTK のバージョン 5.*系と 6.*系両方のバージョンで実行できるように修正を行った。

```
171 行目　visualize_isosurface.py
変更前
pngWriter.SetInput(windowToImageFilter.GetOutput())

↓↓
変更後

if VTK_MAJOR_VERSION <= 5:
pngWriter.SetInput(windowToImageFilter.GetOutput())
else:
pngWriter.SetInputData(windowToImageFilter.GetOutput())
-------------------------------------------------

377 行目　visualize_isosurface.py
変更前
mapper0.SetInput(source0.GetOutput())
```

↓↓
変更後
```
if VTK_MAJOR_VERSION <= 5:
mapper0.SetInput(source0.GetOutput())
else:
mapper0.SetInputConnection(source0.GetOutputPort())
```

428行目　visualize_isosurface.py
変更前
```
mapper.SetInput(cylinder.GetOutput())
```

↓↓

変更後
```
if VTK_MAJOR_VERSION <= 5:
mapper.SetInput(cylinder.GetOutput())
else:
mapper.SetInputData(cylinder.GetOutputPort())
```

565行目　visualize_isosurface.py
変更前
```
writer.SetInput(w2if.GetOutput())
```

↓↓

変更後
```
if VTK_MAJOR_VERSION <= 5:
writer.SetInput(w2if.GetOutput())
else:
writer.SetInputConnection(w2if.GetOutputPort())
```

597行目　visualize_isosurface.py

```
writer.SetInput(w2if.GetOutput())
```

```
↓↓

変更後
if VTK_MAJOR_VERSION <= 5:
writer.SetInput(w2if.GetOutput())
else:
writer.SetInputConnection(w2if.GetOutputPort())
--------------------------------------------------
```

F.6　v1.0.4 (2014年12月29日)

・符号自由度処理を実行時に選択できるように修正

```
13行目　VisBAR_wave_batch(追加)

sign_correction_mode = raw_input("sign_correction ON:1, OFF:0=")

--------------------------------------------------

26〜28行目　VisBAR_wave_batch(追加・修正)

if sign_correction_mode == "1":
cnt +=1
print "sign_correction ON!"
```

・キーボードから操作する実行時強制終了のコマンドを追加

```
549〜553行目　visualise_isosurface.py(追加)

end_select = raw_input("Can program end really? [End:1, Continue:0]=")
if end_select == "1":
print "Program Ended."
exit()
```

・ユーザーが設定する内容がOnを1、Offを0というように統一。今までは、項目ごとで違っていたので、ユーザーの混乱を防ぐため。

```
12行目　VisBAR_wave_batch.py(修正)

修正前
boundary_mode = raw_input("Periodic:0, No_Periodic:1=")
```

```
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

修正後
boundary_mode = raw_input("Periodic:1, No_Periodic:0=")


--------------------------------------------------


47 行目   formatconvert.py(修正)

修正前
if boundary_mode == "0":

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

修正後
if boundary_mode == "1":


--------------------------------------------------


269 行目   formatconvert.py(修正)

修正前
if boundary_mode == "0":

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

修正後
if boundary_mode == "1":
```

F.7　v1.0.5 (2015 年 01 月 23 日)

・第 1 章引用論文修正
・convert_wave_function.vtk.txt 出力部分を修正。
・camera の変数に ClippingRange を追加
・波動関数等値面を wireframe 表示できるように修正。

・convert_wave_function.vtk.txt 出力部分を修正。

```
120〜131 行目、307 行目〜318 行目　formatconvert.py(修正)
謝辞：井町宏人氏（鳥取大学）

    L = int(cube_list[5][1]) / 6 + 1
    for xy in range(0, int(cube_list[3][1]) * int(cube_list[4][1])):
        for line_num_in_xy in range(0, L):
            line = f.readline()
            line = line.strip()
            if line == "":  # Skip empty line.
                continue
            line = " ".join(map(lambda s: str(float(s)), re.split(" *", line)))  # Conv
            out.write(line + " ")
        out.write("\n")
    out.close()
    f.close()
```

・camera の変数に ClippingRange を追加

```
215,216 行目　　visualize_isosurface.py（追加）

out.write('ClippingRange'+' '+str(camera.GetClippingRange()[0])
                          +' '+str(camera.GetClippingRange()[1])+ '\n')

--------------------------------------------------

468,469 行目　　visualize_isosurface.py（追加）

camera.SetClippingRange(float(read["ClippingRange"][0]),
                            float(read["ClippingRange"][1]))
```

・波動関数等値面を wireframe 表示できるように修正

```
468,469 行目　　visualize_isosurface.py（追加）
if read["isoWireplus"][0] == "On":
    isosurfaceActor.GetProperty().SetRepresentationToWireframe()

--------------------------------------------------
```

```
468,469 行目     visualize_isosurface.py (追加)
if read["isoWireminus"][0] == "On":
    isosurfaceminusActor.GetProperty().SetRepresentationToWireframe()
```

F.8 v1.0.5 (2015 年 01 月 25 日)

・2.1 実行コマンドに「vtkpython」の文字がマニュアルに抜けていたので修正。
・2.5「周期境界条件適用方法」の節を追加。
・2.4「「sign correction」機能」の節を追加。
・誤字脱字を修正。

F.9 v1.0.7 (2015 年 03 月 20 日)

＜マニュアル＞
・表記を「VisBAR Wave Batch」に統一。
・3 章でソースコードが一部切れている部分を修正

＜プログラム＞
・コマンドラインからの実行のみに修正

・コマンドラインオプションを設定
⇒インプットディレクトリ、アウトプットディレクトリ、 Batch モード、signcorrection モード、周期境界条件適用、ヘルプ表示

・z 軸メッシュ数が 6 の倍数で Cube ファイルに空白行がありなしに関わらずプログラムが実行できるよう修正

・原子にラベルを貼り付け可視化できる機能を追加

・波動関数の値を正・負それぞれ個別に変化できるよう修正
⇒また、初期値に戻るキーコマンドを設定

・波動関数を正負それぞれ透過できる度合いを設定

・キーコマンドで x,y,z 軸にカメラを合わせるようにした

・強制終了コマンドを追加「shift+q」

・実行時 VisBAR Wave Batch のヘルプを表示出来るようにした

＜開発者＞
鳥取大学大学院工学研究科機械宇宙工学専攻
応用数理工学コース物理計算工学研究グループ
山崎渓太

山崎の主な開発を終了します。

F.10　v1.0.7 (2016年01月08日)

＜マニュアル＞
・p.16の【画面の操作方法】においてキーボード入力 q or e に関する説明を変更。
「q or e」: 現在表示している画面の終了⇒「q」: 次ステップの画面を表示

・p.22及びp.35の座標単位を「オングストローム」から「a.u.」へ修正。

＜更新者＞
鳥取大学大学院工学研究科機械宇宙工学専攻
応用数理工学コース物理計算工学研究グループ
梶　貴美

F.11　v2.0.0 (2016年01月08日)

＜マニュアル＞
・クイックスタートをv2.0.0に対応させた。
　・v1.*.*と描かれているところをv2.0.0に変更
　・「並列処理方法」を記載
　・「実行時に出力されるファイル」の章をv2.0.0に対応させた
　・「画面の操作方法」の章をv2.0.0に対応させた
・ソースコードの部分をコメントアウトした。
・付録に、OSMesaを使用したVTKのビルドのログとargparseのインストール方法を追加した

＜プログラム＞
・どこのディレクトリでも実行できるようにした。(VisBAR_wave_batch.pyのあるフォルダでしか実行できなかったため)
・visbar_wb_setting.txtを現在のディレクトリにあるものを読み込むようにした。
・中間ファイルをそれぞれ別のフォルダに作成するようにした。(実行ディレクトリに同じ名前でファイルを保存していて、並列化の際に問題になったため)
・可視化し終わったあとに中間ファイルをディレクトリごと削除するようにした。(中間ファイルのファイル要領が大きく邪魔になったため)
・log_VTK.txtを出力しないようにした。

・mpi4py と mlutiprocessing を使用して並列化を行った
・並列数を OMP_NUM_THREADS から取得するようにした。
・mlutiprocessing の並列方法でメインのファイルを 2 つに分けた (prosess の方を使用した場合はメモリエラーが起こらない)
・原子数が一万個を超えてもエラーが出ないように修正した。
・デフォルトの色 vesta カラーにした
・mpi4py が入っている環境では mpi4py を使用し、入ってなければ使用しないようにしました。

・-parallel(並列処理) と -b(バッチモード) を併用しなければ止まるようにした。
・-s と -parallel を併用すれば止まるようにした。
・OMP_NUM_THREADS を取得できない場合は最大コア数になるようにした。
・VTK のオフスクリーンレンダリングをバッチモードのときオンになるようにした。
・インタラクティブに描画していくモードで次の画像に移る機能が正常に動作しない問題を修正した。現在は「n」ボタンに割り振っています。

＜更新者＞
鳥取大学大学院工学研究科機械宇宙工学専攻
応用数理工学コース物理計算工学研究グループ
安部　友樹也

F.12　v2.0.0 (2016 年 01 月 11 日)(マニュアル更新のみ)

＜マニュアル＞
・「argparse のインストール」を修正
・「リモート操作 (オフスクリーンモード) での利用」を修正

＜更新者＞
鳥取大学大学院工学研究科機械宇宙工学専攻
応用数理工学コース物理計算工学研究グループ
安部　友樹也

F.13　v2.0.0 (2016 年 01 月 11 日)(マニュアル更新のみ)

＜マニュアル＞
・sec.2.2 の内容を付録 A に統合し、付録 A のタイトルを変更
・v1.0.7 のソースコードの解説を付録として復元
・「argparse のインストール」のタイトル修正
・chap.2 の冒頭部分の修正

＜更新者＞
鳥取大学大学院工学研究科機械宇宙工学専攻
応用数理工学コース物理計算工学研究グループ

安部　友樹也

F.14　v2.0.0 (2016年01月22日)

＜マニュアル＞
・2.1　サンプルコードを持ってきたURLを記載
・3.1.2　サンプルコードの変更に合わせてサンプルコードの説明を変更した。

＜プログラム＞
・サンプルコードの変更

＜更新者＞
鳥取大学大学院工学研究科機械宇宙工学専攻
応用数理工学コース物理計算工学研究グループ
安部　友樹也

F.15　v2.0.0 (2016年04月15日)

＜マニュアル＞
・2.11　mpi4pyを使用したVisBAR Wave Batchの実行を追加
・付録D mpi4pyのインストールを追加

＜プログラム＞
・mpi4pyを使用時にエラーが発生していたのでそれを修正
・sampleにオフスクリーン用のコードを追加

＜更新者＞
鳥取大学大学院工学研究科機械宇宙工学専攻
応用数理工学コース物理計算工学研究グループ
安部　友樹也

F.16　v2.0.1 (2016年08月03日)

＜マニュアル＞
・3.2.2　周期境界条件の計算方法をプログラムにあわせて変更

＜プログラム＞
・周期境界条件の計算が間違っていたようなので修正

＜更新者＞
鳥取大学大学院工学研究科機械宇宙工学専攻

応用数理工学コース物理計算工学研究グループ
安部　友樹也