チュートリアルと解説

探索アルゴリズムや構造生成手法の解説とともに、Si 8原子の系を対象にしたランダムサーチによる構造探索のチュートリアルを行う。ここでは現時点（2018年11月）で最新バージョンであるCrySPY 0.6.4を用いる。構造最適化ソフトはQuantum ESPRESSO 6.1を用いる（バージョン5.xではエラーが出る）。

- 1. インストール (01_install.html)
  - 1.1. find_wy (01_install.html#find-wy)
  - 1.2. Python (01_install.html#python)
  - 1.3. 構造最適化プログラム (01_install.html#id3)
  - 1.4. CrySPY (01_install.html#cryspy)
- 2. 初期構造生成 (02_init_struc.html)
  - 2.1. 入力ファイル (02_init_struc.html#id2)
  - 2.2. CrySPY実行スクリプト作成 (02_init_struc.html#cryspy)
  - 2.3. 初期構造生成 (02_init_struc.html#id5)
- 3. ランダム構造生成の解説 (03_random_generation_method.html)
  - 3.1. 関連するインプット変数 (03_random_generation_method.html#id2)
  - 3.2. フローチャート (03_random_generation_method.html#id3)
- 4. 構造最適化 (04_struc_opt.html)
  - 4.1. 初期構造生成のチェック (04_struc_opt.html#id2)
  - 4.2. 入力ファイル (04_struc_opt.html#id3)
  - 4.3. バックアアップ (04_struc_opt.html#id4)
  - 4.4. 計算の再開 (04_struc_opt.html#id5)
  - 4.5. 構造最適化の実行 (04_struc_opt.html#id6)
- 5. インプットファイルの解説 (05_input_detail.html)
  - 5.1. [basic]セクション (05_input_detail.html#basic)
  - 5.2. [QE]セクション (05_input_detail.html#qe)
- 6. 小技 (06_tips.html)
  - 6.1. 初期構造の追加 (06_tips.html#id2)
  - 6.2. 初期構造の読み込み (06_tips.html#id3)
  - 6.3. stop_next_struc (06_tips.html#stop-next-struc)
  - 6.4. kppvolの目安 (06_tips.html#kppvol)
- 7. データ解析と可視化 (07_data_visualization.html)
  - 7.1. cryspy_analyzer_RS.ipynb (07_data_visualization.html#cryspy-analyzer-rs-ipynb)

© Copyright 2017, Tomoki Yamashita.
最終更新: 2020年01月22日

Back to top

チュートリアルと解説 (outline.html)

1. インストール
   - 1.1. FIND_WY
     - 1.1.1. M_TSPACE
     - 1.1.2. FIND_WY
   - 1.2. PYTHON
   - 1.3. 構造最適化プログラム
   - 1.4. CRYSPY
     - 1.4.1. ディレクトリ構成
     - 1.4.2. FIND_WY
     - 1.4.3. F-FINGERPRINT

1. インストール

必要な外部ソフト、ライブラリおよびCrySPY本体のインストールを行う。

1.1. find_wy

構造生成時に任意の空間群を維持した構造を生成する場合に必要（デフォルト: `spgnum = all`）。空間群の情報を使わない場合は必要はない（ `spgnum = 0`）。 `find_wy` をインストールするためには `m_tspace` が必要となるので `m_tspace` からインストールする。

1.1.1. m_tspace

https://github.com/nim-hrkn/m_tspace (https://github.com/nim-hrkn/m_tspace) から任意の保存先に `git clone` するかダウンロードする。ここでは例として、 `~/local/src` ディレクトリに保存する。つまり、 `~/local/src/m_tspace` 。

参考

https://github.com/nim-hrkn/m_tspace/wiki (https://github.com/nim-hrkn/m_tspace/wiki)

注釈

git hubから `git clone` するとディレクトリ名は `m_tspace` 、zipファイルをダウンロードしてきた場合、展開するとディレクトリ名は `m_tspace-master` となる。

リンク先にも書いてある通り、柳瀬先生のTSPACEのプログラムファイルが2つ必要になるのでダウンロードして `~/local/src/m_tspace` ディレクトリに移す。

tsp98.f: http://phoenix.mp.es.osaka-u.ac.jp/~tspace/tspace_main/tsp07/tsp98.f (http://phoenix.mp.es.osaka-u.ac.jp/~tspace/tspace_main/tsp07/tsp98.f)
prmtsp.f: http://phoenix.mp.es.osaka-u.ac.jp/~tspace/tspace_main/tsp07/prmtsp.f (http://phoenix.mp.es.osaka-u.ac.jp/~tspace/tspace_main/tsp07/prmtsp.f)

上記ファイルが準備できたら、 `makefile` を編集して各環境に合わせて `make` する。

`m_tspace` :

```
$ cd ~/local/src/m_tspace
$ emacs makefile
$ make
```

警告

`ifort` を用いる場合、オプションの `-check all` をつけたままだとwarningが出て動作が遅くなる場合があるので外した方が良い。最適化レベルも `-O2` くらいにしておくと良い。

m_tsp.a というライブラリが生成される。これを find_wy のコンパイルで使用する。

1.1.2. find_wy

https://github.com/nim-hrkn/find_wy (https://github.com/nim-hrkn/find_wy) から任意の保存先に git clone するかダウンロードする。ここでは m_tspace の時と同様に例として、 ~/local/src ディレクトリに保存する。つまり、 ~/local/src/find_wy 。

参考

https://github.com/nim-hrkn/find_wy/wiki (https://github.com/nim-hrkn/find_wy/wiki)

make.inc を編集して先ほど生成した m_tsp.a へのパス（TSPPATH）を設定し、各環境に合わせて make する。

find_wy :

```
$ cd ~/local/src/find_wy
$ emacs make.inc
$ make
```

make.inc 編集後:

```
TSPPATH=~/local/src/m_tspace
#INCPATH = -I $(TSPPATH)
TSP=$(TSPPATH)/m_tsp.a
...
```

実行ファイル find_wy ができればコンパイル成功。 input_sample ディレクトリに入力ファイルのサンプルが入ってるので動作確認を行う。

find_wy の動作確認:

```
$ ./find_wy input_sample/input_si4o8.txt
```

エラーもなく POS_WY_SKEL_ALL.json などが生成されていれば問題なし。CrySPYではこの POS_WY_SKEL_ALL.json を利用して構造生成している。

1.2. Python

以下のものを使える状態にしておく。Python本体とCOMBO以外のものは pip で環境を構築するか、anacondaを使っても良い。

- Python 2.7.x
- COMBO (https://github.com/tsudalab/combo)
- numpy
- pandas
- pymatgen (http://pymatgen.org/)

1.3. 構造最適化プログラム

以下の中から、自分が使いたい構造最適化プログラムを使えるようにしておく。ここではQuantum ESPRESSOを例に説明する。

- VASP (https://www.vasp.at (https://www.vasp.at))
- Quantum Espresso (http://www.quantum-espresso.org (http://www.quantum-espresso.org))
- soiap (https://github.com/nbsato/soiap (https://github.com/nbsato/soiap))
- LAMMPS (http://lammps.sandia.gov (http://lammps.sandia.gov))

1.4. CrySPY

CrySPY本体をダウンロードする。保存先は任意のディレクトリで構わないが、ここでは例として ~/CrySPY_root/CrySPY-0.6.4 に保存する。 gitを使う場合は、

```
$ mkdir ~/CrySPY_root
$ cd ~/CrySPY_root
$ git clone https://github.com/Tomoki-YAMASHITA/CrySPY.git CrySPY-0.6.4
```

gitを使わずにzipやtar.gzファイルを取ってくる場合は以下のリンクからダウンロードする。

https://github.com/Tomoki-YAMASHITA/CrySPY/releases (https://github.com/Tomoki-YAMASHITA/CrySPY/releases)

1.4.1. ディレクトリ構成

`cryspy.py` がメインスクリプトで、これを何度も実行することで構造探索を行う。

`docs/` にはドミュメントのhtmlファイルが保存されている。

`example/` には入力ファイル例。

`utility/` には結果解析用のjupyter notebookやちょっとしたスクリプトなどが入っている。

`CrySPY/gen_struc/` には構造生成プログラムが入っていて、CrySPYとは独立にimportして用いることも可能。ランダム構造の生成だけを行ったり、次のバージョン（0.7.0）からは進化的アルゴリズムを用いた構造生成だけを行うといったこともできる。

`~/CrySPY_root/CrySPY-0.6.4`：

```
CrySPY-0.6.4/
├── CHANGELOG.md
├── CrySPY/
│   ├── BO/
│   ├── IO/
│   ├── LAQA/
│   ├── __init__.py
│   ├── calc_dscrpt/
│   ├── f-fingerprint/
│   ├── find_wy/
│   ├── gen_struc/
│   ├── interface/
│   ├── job/
│   └── start/
│   └── utility.py
├── LICENSE
├── README.md
├── cryspy.py
├── docs/
├── example/
└── utility/
```

1.4.2. find_wy

空間群の情報を用いて初期構造を生成する場合は、先にインストールしておいた `find_wy` を `~/CrySPY_root/CrySPY-0.6.4/CrySPY/find_wy/` ディレクトリの中にコピーする。CrySPYはこの場所にある `find_wy` を実行する。つまり、`~/CrySPY_root/CrySPY-0.6.4/CrySPY/find_wy/find_wy` 。

```
$ cd ~/CrySPY_root/CrySPY-0.6.4/CrySPY/find_wy
$ cp ~/local/src/find_wy/find_wy .
```

1.4.3. f-fingerprint

ベイズ最適化を使う場合は、結晶構造の記述子を計算するプログラム、 `cal_fingerpirnt` のコンパイルが必要。各環境に合わせて `makefile` を編集して `make` する。

```
$ cd ~/CrySPY_root/CrySPY-0.6.4/CrySPY/f-fingerprint
$ emacs makefile
$ make
```

`~/CrySPY_root/CrySPY-0.6.4/CrySPY/f-fingerprint/` ディレクトリの中に `cal_fingerpirnt` という実行ファイルがあれば良い。

© Copyright 2017, Tomoki Yamashita.
最終更新: 2020年01月22日

Back to top

チュートリアルと解説 (outline.html)

2. 初期構造生成

2.1. 入力ファイル

2.1.1. CRYSPY.IN

2.1.2. CALC_IN/

2.2. CRYSPY実行スクリプト作成

2.2.1. 通常の実行スクリプト

2.2.2. ジョブ投入用の実行スクリプト

2.3. 初期構造生成

2.3.1. CRYSPY実行

2.3.2. チェック

2. 初期構造生成

詳細な解説の前に、動作チェックも兼ねてとりあえず初期構造生成までCrySPYを動かしてみる。 初期構造生成は場合によっては少し時間がかかることもあるので、その待ち時間を利用してインプットファイルの解説に移る。

このチュートリアルでは、 `~/tutorial_CrySPY/` というディレクトリで計算を行う。ディレクトリを作成して移動する。

```
$ mkdir ~/tutorial_CrySPY
$ cd ~/tutorial_CrySPY
```

2.1. 入力ファイル

入力ファイル例がCrySPYに同梱されているのでまずはそれをコピーするところから始める。 Quantum ESPRESSO用のサンプルを使う。

```
$ cp -r ~/CrySPY_root/CrySPY-0.6.4/example/QE_RS_Si8/ .
$ cd QE_RS_Si8/
```

`QE_RS_Si8/` の中に必要な入力ファイルが入っている。

- `calc_in/`
- `cryspy.in`

`calc_in/` は構造最適化ソフト用の入力ファイルを用意するディレクトリで、Quantum ESPRESSOの入力ファイルはここに準備する。 `cryspy.in` がCrySPYの入力ファイル。

2.1.1. cryspy.in

`cryspy.in` の内容を確認してみる。

```
$ cat cryspy.in
[basic]
algo = RS
calc_code = QE
tot_struc = 20
natot = 8
atype = Si
nat = 8
nstage = 3
njob = 20
jobcmd = qsub2
jobfile = job_cryspy

[lattice]
minlen = 4
maxlen = 10
dangle = 20
mindist_1 = 1.8

[QE]
qe_infile = pwscf.in
qe_outfile = pwscf.out
kppvol = 40 100 100

[option]
```

使用するスパコンやサーバに合わせて編集しなければいけない箇所が色々あるが後回しにして、 とりあえず初期構造生成に深く関連するハイライトされた部分を見てみる。

なんとなく分かるかもしれないが、Si 8個から構成される初期構造を20構造生成する。 [lattice]セクションのところは格子定数の最小、最大値や原子間距離への制限を決めるパラメーターとなっている。

2.1.2. calc_in/

`calc_in/` ディレクトリの中身は以下のようになっている。

- job_cryspy
- pwscf.in_1
- pwscf.in_2
- pwscf.in_3

これらは構造最適化の計算に必要な入力ファイルやジョブスクリプトなので今は説明はスキップする。

2.2. CrySPY実行スクリプト作成

2.2.1. 通常の実行スクリプト

入力ファイルさえ準備してしまえばメインスクリプトである `~/CrySPY_root/CrySPY-0.6.4/cryspy.py` を実行すればCrySPYは動作する。これは何度も行う必要があって、毎回コマンドを手打ちするのは面倒なので実行スクリプトファイルを作っておくと便利である。必須ではないので作らなくても良い。

ファイル名は何でも良いが、例として `cryspy.sh` という実行スクリプトを作成する。

```
$ emacs cryspy.sh
```

`cryspy.sh` :

```
#!/bin/bash

EXE='/path_to_python/python'
$EXE -u ~/CrySPY_root/CrySPY-0.6.4/cryspy.py  1>> log 2>> err
```

必要なライブラリをインストールしたpythonへのパスが通っている場合は必要ないが、パスが通ってない場合は `/path_to_python/` のところでpythonへのパスを指定する。

標準出力を `log` にエラー出力を `err` に追記する形で書き出すようにしている。pythonでは標準出力をリダイレクトで吐き出すとバッファリングしてリアルタイムで出力してくれなくなるので、 `-u` オプションをつけてバッファリングを無効にしている。

2.2.2. ジョブ投入用の実行スクリプト

スパコンなどでは通常フロントエンドで負荷のかかるプログラムや長時間実行するプログラムは流せないので、初期構造生成やベイズ最適化、進化的アルゴリズムによる構造生成など、時間がかかりそうなプログラムを実行する時はPBSなどのジョブ管理システムにジョブを投入する。時間がかかっても問題がない環境であればこれは必須ではない。

警告

構造最適化を行う場合にはこのジョブ投入用スクリプトは使ってはいけない。 例えばPBSの場合 `qsub` で投入したジョブスクリプトから2重に `qsub` でジョブを投げるようなことはできない。 初期構造生成やベイズ最適化、進化的アルゴリズムによる構造生成など、ジョブ管理システムにジョブを投入しない場合にのみ用いるようにする。

ここでは例として、物性研スパコンシステムBで使用しているジョブスクリプトを載せる。ファイル名は任意。

```
$ emacs job_struc
```

`job_struc` :

```
#!/bin/sh -l
#QSUB -queue F4cpu
#QSUB -node 1
#PBS -l walltime=24:00:00
#PBS -N struc_gen
cd $PBS_O_WORKDIR


EXE='/path_to_python/python'
$EXE -u ~/CrySPY_root/CrySPY-0.6.4/cryspy.py  1>> log 2>> err
```

最初の6行はジョブ管理システムを使うためのものなので、各環境に合わせて書き換える。あとは通常の実行スクリプトと同じ。

2.3. 初期構造生成

2.3.1. CrySPY実行

準備ができたら初期構造生成を行ってみよう。

スパコンなどではなく、自分のPCや自前のワークステーションなど、ジョブ管理システムを使う必要がない場合は `cryspy.sh` を実行する。

```
$ bash cryspy.sh
```

スパコンなどでジョブ管理システムを使う場合は、 `job_struc` を使ってジョブを投入する。例えば `qsub` の場合は

```
$ qsub job_struc
```

2.3.2. チェック

うまく動作していれば以下のファイルおよびディレクトリが作成される。

- `cryspy.out` : 出力ファイル。 `log` の簡易版。
- `cryspy.stat` : ステータスファイル。現在の状況が確認できる。このファイルが存在すればCrySPYは途中から計算を再開する。
- `err` : エラー出力。
- `log` : 標準出力。 `cryspy.out` の詳細版。
- `data/init_POSCARS` : POSCAR形式（VASPの入出力形式）で構造データが追記されていく。このファイルをVESTAで直接開くことで構造を可視化できる。
- `data/pkl_data/` : pkl形式データの保存ディレクトリ。計算再開時はこれらのデータを読み込む。
- `gen_struc` : 初期構造生成時の一時保存用ディレクトリ。あとで削除しても良い。

`log` ファイルを見ると現在いくつまで構造を生成したか確認できる。動作を確認したら、構造生成している間に次のランダム構造生成の解説 (03_random_generation_method.html)まで読み進もう。

© Copyright 2017, Tomoki Yamashita.
最終更新: 2020年01月22日

Back to top

チュートリアルと解説 (outline.html)

3. ランダム構造生成の解説
3.1. 関連するインプット変数
3.1.1. [BASIC]セクション
3.1.1.1. ATYPE
3.1.1.2. NAT
3.1.2. [LATTICE]セクション
3.1.2.1. MINLENとMAXLEN
3.1.2.2. DANGLE
3.1.2.3. MINDIST_?
3.1.3. [OPTION]セクション
3.1.3.1. MAXCNT
3.1.3.2. SPGNUM
3.2. フローチャート

3. ランダム構造生成の解説

`cryspy.in` のランダム構造生成に関連する部分だけ抜き出す。[basic]セクションと[lattice]セクションのものは必須の入力変数となっており、 [option]セクションの二つはデフォルトで値が設定されているので必須ではない。

```
[basic]
tot_struc = 20
natot = 8
atype = Si
nat = 8

[lattice]
minlen = 4
maxlen = 10
dangle = 20
mindist_1 = 1.8

[option]
maxcnt = 200
spgnum = all
```

3.1. 関連するインプット変数

参考

https://tomoki-yamashita.github.io/CrySPY/input_file.html (https://tomoki-yamashita.github.io/CrySPY/input_file.html)

3.1.1. [basic]セクション

名前	値	デフォルト	説明
`tot_struc`	int		生成する構造の数
`natot`	int		ユニットセル内の総原子数
`atype`	atomic symbol [atomic symbol ...]		原子種

名前	値	デフォルト	説明
nat	int [int ...]		原子種ごとの原子数

3.1.1.1. atype

原子種の指定。原子種が2種類以上の場合は、スペースを挟んでそれぞれ指定する。

```
atype = Y Co
```

3.1.1.2. nat

原子数の指定。原子種が2種類以上の場合は、スペースを挟んでそれぞれ指定する。

```
nat = 1 5
```

この例では atype で指定した1種類目の原子が1個、2種類目が5個の計6個の原子となる。

3.1.2. [lattice]セクション

名前	値	デフォルト	説明
minlen	float		格子ベクトルの最小値 [Å]
maxlen	float		格子ベクトルの最大値 [Å]
dangle	float		角度のズレの最大値 [°]
mindist_?	float [float ...]		原子間距離の制限 [Å]

3.1.2.1. minlenとmaxlen

格子ベクトルの長さは minlen から maxlen までの間の数値がランダムで選ばれる。 格子ベクトルの長さをあまり小さくしすぎると、ユニットセルの中に原子がぎゅうぎゅうに詰まり、原子間距離が mindist_? 未満になりやすく、構造生成に時間がかかる。 逆に格子ベクトルの長さを長くすると、構造生成は早くできるが、スカスカの結晶ができてしまう。

3.1.2.2. dangle

dangle = θ は格子定数 α, β および γ に以下のように制限を与える。

三斜晶系	Type 1	$90° - \theta \le \alpha, \beta, \gamma < 90°$
	Type 2	$90° \le \alpha, \beta, \gamma \le 90° + \theta$
単斜晶系		$90° \le \beta \le 90° + \theta$
菱面体晶系		$90° - \theta \le \alpha \le 90° + \theta$

3.1.2.3. mindist_?

構造生成時にあまりに原子同士が近すぎると計算でエラーが出ることがあるので最低限引き離したい距離を設定する。 原子種が1種だけの時は mindist_1 がそのまま1行1列の行列となる。 原子種が2種以上の時は mindist_1 , mindist_2 ... を並べた行列が対称行列となるようにする。 例えば以下のように設定すると

```
atype = Y Co
```

```
mindist_1 = 2.0  1.8
mindist_2 = 1.8  1.5
```

mindist 対称行列は

$$\begin{pmatrix} 2.0 & 1.8 \\ 1.8 & 1.5 \end{pmatrix}$$

となる。これはY-Y, Y-Co, Co-Co間の最小原子間距離が2.0, 1.8, 1.5 Å にそれぞれ設定されていることを意味する。

警告

mindist行列は対称行列でないとエラーが出る

3.1.3. [option]セクション

名前	値	デフォルト	説明
`maxcnt`	int	200	構造生成時、原子を配置する最大試行回数
`spgnum`	`all`, 0, space group number	`all`	0ならば空間群の情報を用いない。 `all` ならば1から230全ての空間群を候補とする。空間群の番号を指定して制限することも可能

3.1.3.1. maxcnt

下記フローチャートを参考。値を小さくすると構造生成は早くなるが、簡単な構造だけ生成していることになるので注意。値を大きくすると何度も頑張って原子を配置しようとするので時間がかかる。 どの程度がベストかはわからないが、とりあえずデフォルトは200にしている。

3.1.3.2. spgnum

空間群を制限する場合、利用する空間群の番号をスペース区切りで入力する。二つの番号をハイフンでつないだ場合は連番だと判断する。 例えば、空間群番号10と100だけに制限する場合は、

```
spgnum = 10 100
```

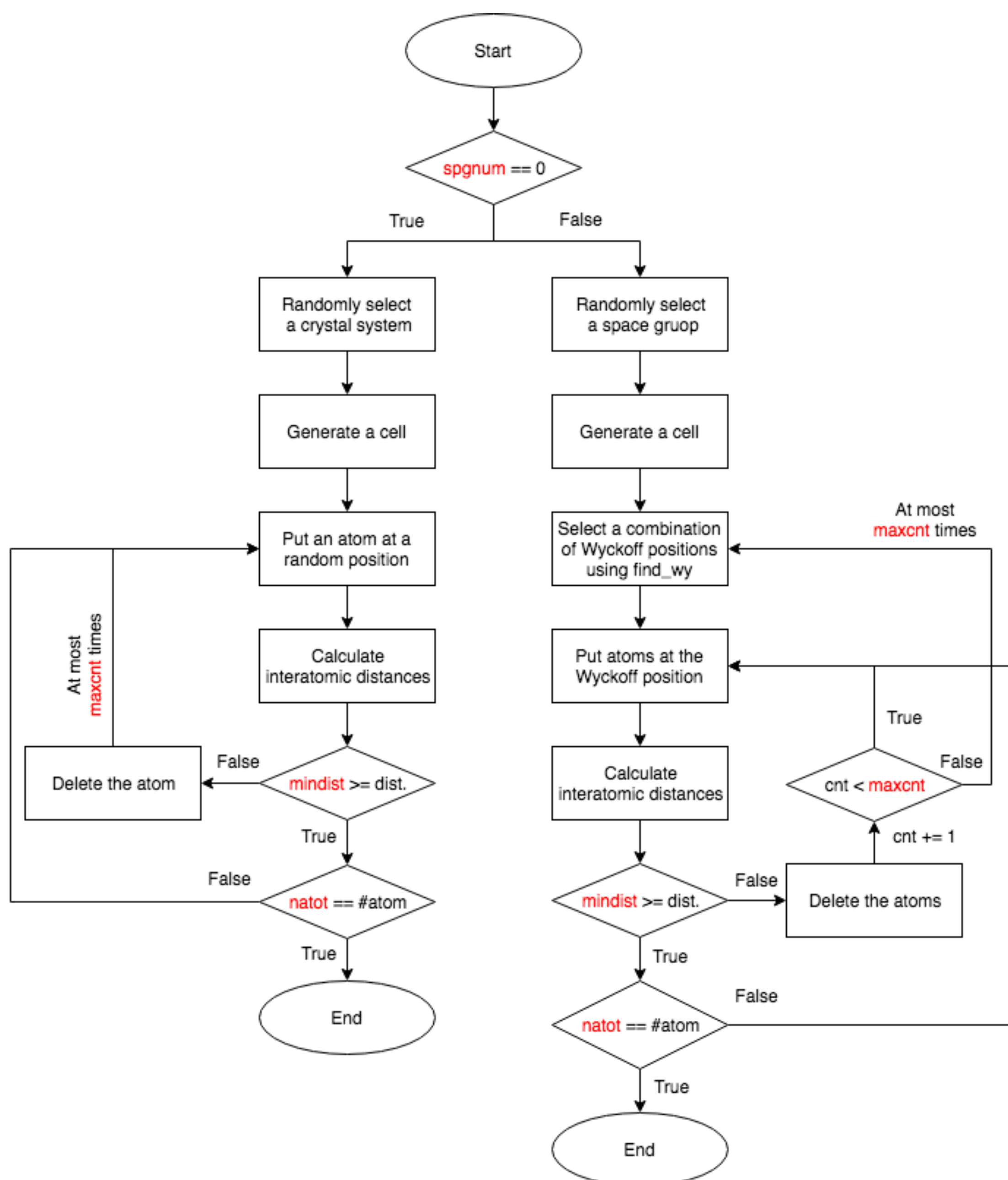
立方晶だけに制限する場合は、

```
spgnum = 195-230
```

空間群番号10と100と立方晶に制限するならば、

```
spgnum = 10 100 195-230
```

3.2. フローチャート



© Copyright 2017, Tomoki Yamashita.
最終更新: 2020年01月22日

Back to top

チュートリアルと解説 (outline.html)

4. 構造最適化

4.1. 初期構造生成のチェック

4.2. 入力ファイル

4.2.1. CRYSPY.IN

4.2.2. CALC_IN/

4.2.2.1. JOB_CRYSPY

4.2.2.2. PWSCF.IN_?

4.3. バックアアップ

4.4. 計算の再開

4.5. 構造最適化の実行

4.5.1. 実行

4.5.2. 確認

4.5.3. 次のステージへ

4.5.4. 計算がうまくいかなかった場合

4.5.5. 次の構造へ

4.5.6. データの確認

4.5.6.1. CRYSPY_RSLT*

4.5.6.2. KPTS_RSLT

4.5.6.3. OPT_*

4.5.7. 計算を進める

4. 構造最適化

4.1. 初期構造生成のチェック

初期構造生成のジョブが終了していれば、 `tot_struc` の数だけ初期構造が生成されている。 `log` ファイルを見てチェックする。下記の例では、ID 0からID 19までの20構造が生成されている。

```
$ cat log
2018/11/02 14:21:49
CrySPY 0.6.4
Start cryspy.py

Read input file, cryspy.in
Write input data in cryspy.out
Save input data in cryspy.stat
Make data directory

# --------- Generate initial structures
Structure ID       0 was generated. Space group:  77 -->  77 P4_2
Structure ID       1 was generated. Space group: 120 --> 120 I-4c2
Structure ID       2 was generated. Space group: 126 --> 123 P4/mmm
Structure ID       3 was generated. Space group: 147 --> 147 P-3
Structure ID       4 was generated. Space group: 128 --> 139 I4/mmm
Structure ID       5 was generated. Space group:  48 --> 139 I4/mmm
Structure ID       6 was generated. Space group: 132 --> 123 P4/mmm
Structure ID       7 was generated. Space group: 149 --> 187 P-6m2
Structure ID       8 was generated. Space group:   2 -->   2 P-1
Structure ID       9 was generated. Space group: 156 --> 156 P3m1
Structure ID      10 was generated. Space group: 217 --> 217 I-43m
Structure ID      11 was generated. Space group:  90 --> 129 P4/nmm
Structure ID      12 was generated. Space group:  60 -->  63 Cmcm
Structure ID      13 was generated. Space group: 120 --> 140 I4/mcm
Structure ID      14 was generated. Space group: 220 --> 220 I-43d
Structure ID      15 was generated. Space group:  51 -->  51 Pmma
Structure ID      16 was generated. Space group: 163 --> 194 P6_3/mmc
Structure ID      17 was generated. Space group: 195 --> 215 P-43m
Structure ID      18 was generated. Space group:  31 -->  31 Pmn2_1
Structure ID      19 was generated. Space group: 223 --> 221 Pm-3m
```

4.2. 入力ファイル

4.2.1. cryspy.in

`cryspy.in` でまだ説明していない部分をハイライトすると

```
[basic]
algo = RS
calc_code = QE
tot_struc = 20
natot = 8
atype = Si
nat = 8
nstage = 3
njob = 20
jobcmd = qsub2
jobfile = job_cryspy

[lattice]
minlen = 4
maxlen = 10
dangle = 20
mindist_1 = 1.8

[QE]
qe_infile = pwscf.in
qe_outfile = pwscf.out
kppvol = 40 100 100

[option]
```

詳細は後で説明するとして、とりあえず動かすためにファイルを修正する。

```
$ emacs cryspy.in
```

`nstage` の値を2に変更する。

`njob` の値を2に変更する。

`jobcmd` の値も編集する。ジョブ投入コマンドをここに書く。ここでは例として、スパコンで一般的によく利用される `qsub` コマンドを用いておく。 自前のワークステーションなど、ジョブ管理システムを用いずにそのまま実行する場合は `qsub` の代わりに `bash` などで実行すれば良い。

`kppvol` の値を `60 80` にする。

編集後は以下の通り。

```
[basic]
algo = RS
calc_code = QE
tot_struc = 20
natot = 8
atype = Si
nat = 8
nstage = 2
njob = 2
jobcmd = qsub
jobfile = job_cryspy

[lattice]
minlen = 4
maxlen = 10
dangle = 20
mindist_1 = 1.8

[QE]
qe_infile = pwscf.in
qe_outfile = pwscf.out
kppvol = 60 80

[option]
```

4.2.2. calc_in/

`calc_in` ディレクトリには構造最適化ソフトのための入力ファイルを準備する。中に入って確認すると

```
$ cd calc_in/
$ ls
job_cryspy  pwscf.in_1  pwscf.in_2  pwscf.in_3
```

`job_cryspy` は構造最適化を実行する時の（今の場合はQEを実行するための）ジョブファイル。ファイル名は任意だが、 `cryspy.in` の `jobfile` の値と合わせる必要がある。

`pwscf.in_?` はQEの入力ファイルで、3つあるのは一つの構造につき3ステージに分けた最適化を行うため。他のコードの場合もファイル名は、入力ファイル名にアンダースコアと数字を付け足したものにする。数字はゼロではなく1から始める。例えば、ステージ1ではセルを固定しておいて内部座標のみの緩和を行って、ステージ2ではセルと内部座標をフルに緩和させる、ステージ3ではさらにk点を増やして計算するなど自由に設定する。

ステージ数は `cryspy.in` の `nstage` で変更可能。 `nstage` で指定した数の入力ファイルを準備しておく必要がある。

今回は軽い計算にしておきたいので `nstage` は2にしている。ステージ3の設定は不要なので `pwscf.in_3` は削除しても良い。

```
$ rm pwscf.in_3
```

注釈

VASPの場合は `job_cryspy` 、 `INCAR_?` 、 `POTCAR` ファイルだけ用意する。 `POSCAR` と `KPOINTS` ファイルは自動的に生成される。

4.2.2.1. job_cryspy

構造最適化を行うための実行スクリプトを普段通りに書く。ただし、CrySPYではスクリプトの最後に `sed -i -e '3 s/^.*$/done/' stat_job` という行を加える必要がある。このコマンドでジョブが終了する際に `stat_job` というファイルの3行目を `done` に書き換えて、ジョブが終了しているかどうかがプログラムから判別できるようにしている。

また、このジョブファイル中の `CrySPY_ID` という文字列は構造IDに変換されるようになっているので、例えばジョブ名などを設定するところに書いておくとどのジョブがどの構造のものなのか判別できる。

exampleの job_cryspy はとあるスパコン用の設定が書いてあるが、ここでは物性研スパコンで用いるジョブスクリプトの一例を載せておく。QEのパスは各自で設定する。

```
$ emacs job_cryspy
```

```
#!/bin/sh -l
#QSUB -queue F4cpu
#QSUB -node 1
#QSUB -mpi 24
#QSUB -omp 1
#PBS -l walltime=01:00:00
#PBS -N CrySPY_ID
cd $PBS_O_WORKDIR

#---------- QE
EXEPATH='/path_to_qe/'
EXE='pw.x'

echo "--------------------------------"
echo ""
echo "Running $EXEPATH/$EXE ..."
date
echo ""
echo "--------------------------------"

mpijob ${EXEPATH}/$EXE -npool 8 < pwscf.in > pwscf.out

echo "--------------------------------"
echo ""
echo "done!"
date
echo ""
echo "--------------------------------"


#---------- for cryspy
sed -i -e '3 s/^.*$/done/' stat_job
```

注釈

`sed -i -e '3 s/^.*$/done/' stat_job` を最後に加える。
`CrySPY_ID` という文字列は構造IDに置換される。
`pwscf.in` 、 `pwscf.out` のファイル名は `cryspy.in` の `qe_infile` と `qe_outfile` に合わせる。

4.2.2.2. pwscf.in_?

CrySPYではQEのインプットを作る時に構造データの部分とk点の部分は書かなくて良い。 格子の情報は後でCrySPYが自動的に追記するので、 `ibrav` の値は0にすること。 下記は一例。ポテンシャルのパスは各自で編集する。

```
$ emacs pwscf.in_1
$ emacs pwscf.in_2
```

```
 &control
    title = 'Si8'
    calculation = 'relax'
    nstep = 100
    restart_mode = 'from_scratch',
    wf_collect = .true.
    pseudo_dir = '/path/to/PSEUDOPOTENTIALS/'
    outdir='./out.d/'
 /

 &system
    ibrav = 0
    nat = 8
    ntyp = 1
    ecutwfc = 44.0
    occupations = 'smearing'
    degauss = 0.014
    smearing = 'mp'
 /

 &electrons
 /

 &ions
 /

 &cell
 /

ATOMIC_SPECIES
  Si  28.086  Si.pbe-n-kjpaw_psl.1.0.0.UPF
```

実際に計算を行う際には自動的に以下のように追記される。

```
 &control
    title = 'Si8'
    calculation = 'relax'
    nstep = 100
    restart_mode = 'from_scratch',
    wf_collect = .true.
    pseudo_dir = '/path/to/PSEUDOPOTENTIALS/'
    outdir='./out.d/'
 /

 &system
    ibrav = 0
    nat = 8
    ntyp = 1
    ecutwfc = 44.0
    occupations = 'smearing'
    degauss = 0.014
    smearing = 'mp'
 /

 &electrons
 /

 &ions
 /

 &cell
 /

ATOMIC_SPECIES
  Si  28.086  Si.pbe-n-kjpaw_psl.1.0.0.UPF

CELL_PARAMETERS angstrom
5.825343 0.000000 0.000000
0.000000 5.825343 0.000000
0.000000 0.000000 5.202697
ATOMIC_POSITIONS crystal
Si  0.000000 0.000000 0.490249
Si  0.000000 0.000000 0.990249
Si  0.000000 0.500000 0.830468
Si  0.500000 0.000000 1.330468
Si  0.093267 0.282349 0.302006
Si  -0.093267 -0.282349 0.302006
Si  -0.282349 0.093267 0.802006
Si  0.282349 -0.093267 0.802006

K_POINTS automatic
3 3 4  0 0 0
```

4.3. バックアアップ

必須ではないが、バックアップをとっておくことをお勧めする。途中まで時間をかけて計算を進めたのにエラーが出てどうしようもなくなったという場合に備える。ディレクトリごとコピーしておく。今回だけでなく定期的にバックアップしておくと良い。

```
$ cd ../..
$ ls
QE_RS_Si8
$ cp -r QE_RS_Si8/ backup_00
$ ls
QE_RS_Si8  backup_00
$ cd QE_RS_Si8
```

4.4. 計算の再開

CrySPYではカレントディレクトリに `cryspy.stat` ファイルがあれば自動でデータを読み込んで計算を再開するようになっている。なければはじめから開始する。

4.5. 構造最適化の実行

4.5.1. 実行

カレントディレクトリが `~/tutorial_CrySPY/QE_RS_Si8` であることと、以前のジョブがしっかり終了していることを確認する。ジョブが終わってないままCrySPYを2重に実行するとデータがおかしくなって取り返しがつかなくなるので注意。

注釈

次のバージョンの0.7.0からは、2重実行防止機能を入れた。CrySPY実行中は `lock_cryspy` というファイルが生成され、このファイルが存在している間は2重にCrySPYを実行してもすぐ止まるようにしてある。CrySPYが終了するとこのファイルも削除される。

問題なければCrySPYを実行する。今回はQEを実行するために内部で `jobcmd` で指定されたコマンドを実行する（例えば `qsub` ）。ジョブ投入コマンドを2重に実行しないように、 `job_struc` ではなく、通常の `cryspy.sh` を実行する。

```
$ bash cryspy.sh
```

4.5.2. 確認

このコマンドはすぐ終了するはず。まずは `ls` でカレントディレクトリを確認してみよう。

```
$ ls
calc_in    cryspy.in~  cryspy.sh    data  gen_struc  job_struc~  stderr.1120599.log  work0000
cryspy.in  cryspy.out  cryspy.stat  err   job_struc  log         stdout.1120599.log  work0001
```

`log` ファイルも確認しておく。

```
$ cat
...
...
2018/11/12 17:16:54
CrySPY 0.6.4
Restart cryspy.py


Changed nstage from 3 to 2
Changed njob from 20 to 2
Changed jobcmd from qsub2 to qsub
Changed kppvol from [40, 100, 100] to [40, 60]
Save input data in cryspy.stat

# ---------- job status
work0000: submit job, structure ID 0 Stage 1
work0001: submit job, structure ID 1 Stage 1
```

このような部分が追記されているはず。インプットを変更した箇所がしっかり読み込まれているのが確認できる。 また、ID 0とID 1の構造のStage 1のジョブがサブミットされたのも確認できる。

ジョブが流れているか確認しておくと良い。例えばPBSの場合は `qstat` コマンドを用いることが多い。 構造最適化の計算がまだ終了していなければ、2つジョブが投入されているはず。 また、ジョブネームに `CrySPY_ID` を設定したので、IDが表示されているはず。

```
$ qstat
(ジョブのステータスなど)
```

`cryspy.stat` ファイルも見てみよう。

```
$ cat cryspy.stat
[input]
algo = RS
calc_code = QE
tot_struc = 20
natot = 8
atype = Si
nat = 8
nstage = 2
njob = 2
jobcmd = sbatch
jobfile = job_cryspy
minlen = 4.0
maxlen = 10.0
dangle = 20.0
mindist_1 = 1.8
qe_infile = pwscf.in
qe_outfile = pwscf.out
kppvol = 40 60
force_gamma = False
maxcnt = 200
stop_chkpt = 0
symtolei = 0.001
symtoler = 0.1
spgnum = all
load_struc_flag = False
stop_next_struc = False
energy_step_flag = False
struc_step_flag = False
fs_step_flag = False

[status]
next_id = 2
work0000 = ID        0, Stage 1
work0001 = ID        1, Stage 1
```

[status]セクションをみると現在の状況がわかる。ID 0が `work0000` で、ID 1が `work0001` で計算中で、次（ `next_id` ）に計算されるのがID 2。

`work0000` の中を確認してみる。

```
$ cd work0000/
$ ls
job_cryspy  out.d  pwscf.in  pwscf.out  stat_job  stderr.1123273.log  stdout.1123273.log  sublog
```

QEの計算がちゃんと行われていればこのようになっているはず。 `stat_job` ファイルを見てみる。

計算がまだ終わっていない場合は

```
$ cat stat_job
0           # Structure ID
1           # Stage
submitted
```

計算が終わっていれば

```
$ cat stat_job
0           # Structure ID
1           # Stage
done
```

3行目がこのように `done` になる。

4.5.3. 次のステージへ

`work0000` から戻って計算が2つとも終了していたら、stage 2へ計算を進める。

```
$ cd ..
$ bash cryspy.sh
```

log ファイルを見ると、次のステージに進んでいることが確認できる。

```
$ tail log
CrySPY 0.6.4
Restart cryspy.py



# ---------- job status
work0000: Structure ID 0 Stage 1 Done!
    submit job, structure ID 0 Stage 2
work0001: Structure ID 1 Stage 1 Done!
    submit job, structure ID 1 Stage 2
```

4.5.4. 計算がうまくいかなかった場合

第一原理計算ソフトのエラーや、スパコンでの時間切れなどでうまくいかなった場合、可能ならば手動で計算を再開させて（ work000? で直接ファイルを修正して手動でジョブをサブミットし直す）様子を見る。

どうしてもうまくいかない場合はその構造をスキップして切り捨てることもできる。必ずその構造のジョブが終了していることを確認して、 stat_job ファイルの3行目を skip に書き換える。 work0000 を例にすると

```
$ cd work0000
$ emacs stat_job
$ cd ..
```

```
0           # Structure ID
2           # Stage
skip
```

警告

必ずそのIDの構造最適化のジョブが終了していることを確認すること。ジョブが残っているのにスキップさせるとファイルが更新されておかしなことになる。

4.5.5. 次の構造へ

stage 2の計算が終了後にもう一度CrySPYを実行すると、ID 0とID 1の計算結果のデータを収集して次の構造の計算が始まる。CrySPYを実行して終わったら log を見てみよう。

```
$ bash cryspy.sh
$ cat log
...
...
# ---------- job status
work0000: Skip Structure ID0
work0000: submit job, structure ID 2 Stage 1
work0001: Structure ID 1 Stage 2 Done!
    collect results: E = -853.245687239
work0001: submit job, structure ID 3 Stage 1
```

ID 0はスキップさせて、ID 1のデータを収集。ID 2と3の構造最適化を開始した。

4.5.6. データの確認

data ディレクトリに入って中身を見てみよう。いろいろファイルが増えているはず。

```
$ cd data/
$ ls
cryspy_rslt             init_POSCARS  opt_CIFS.cif  opt_qe-structure
cryspy_rslt_energy_asc  kpts_rslt     opt_POSCARS   pkl_data
```

4.5.6.1. cryspy_rslt*

cryspy_rslt と cryspy_rslt_energy_asc はほぼ同じ内容でエネルギーなどの結果が出力されている。前者はIDの順番、後者はエネルギーが低い順。

```
Struc_ID  Spg_num Spg_sym  Spg_num_opt Spg_sym_opt      Energy  Magmom      Opt
       0       77    P4_2            0        None         NaN     NaN     skip
       1      120   I-4c2          142    I4_1/acd -853.245687     NaN  not_yet
```

Spg_num , Spg_sym は初期構造の空間群情報で、 Spg_num_opt , Spg_sym_opt が最適化後の空間群。 エネルギーはユニットセルあたりのeV単位。 Magmom は対応しているコードを用いた場合、スピン磁気モーメントの値が表示される。 Opt は構造最適化がちゃんと終了しているかどうかの判断。 not_yet の場合、最適化ソフトで設定した条件を満たさずに終了している可能性がある。 done になっていればきちんとしている。

注釈

VASPの場合、通常は構造最適化後に、セルや原子を動かさずにSCF計算をもう1回やり直す必要がある。例えば nstage が4だとすると、 最後のstage 4はセルや原子を固定した計算に当てるので、最適化が終了しているかの判断はstage 3の結果を見て判断している。

4.5.6.2. kpts_rslt

各ステージごとのk点分割の情報。

```
$ cat kpts_rslt
  Struc_ID    k-points
         0  [[3, 3, 4], [4, 4, 4]]
         1  [[5, 5, 5], [5, 5, 5]]
         2  [[4, 4, 2]]
         3  [[4, 4, 4]]
```

4.5.6.3. opt_*

最適化後の構造データ。通常はcif形式とPOSCAR形式の opt_CIFS.cif , opt_POSCARS が出力される。これらはVESTAで直接開くことで可視化できる。 QEの場合は opt_qe-structure も出力される。

4.5.7. 計算を進める

data ディレクトリから一つ上のディレクトリに戻り

```
$ cd ..
```

あとは同様にして、しばらく計算を進める。最適化計算のジョブが終わったら

```
$ bash cryspy.sh
```

を実行していく。

計算を進めている間にインプットファイルの解説 (05_input_detail.html)を行う。

© Copyright 2017, Tomoki Yamashita.
最終更新: 2020年01月22日

Back to top

チュートリアルと解説 (outline.html)

- 5. インプットファイルの解説
  - 5.1. [BASIC]セクション
    - 5.1.1. ALGO
    - 5.1.2. CALC_CODE
    - 5.1.3. NSTAGE
    - 5.1.4. NJOB
    - 5.1.5. JOBCMDとJOBFILE
  - 5.2. [QE]セクション
    - 5.2.1. KPPVOL
    - 5.2.2. QE_INFILEとQE_OUTFILE

5. インプットファイルの解説

計算の流れが理解できたと思うので、すでに説明した構造生成に関連するところ以外の部分のインプットファイルの解説を行う。

```
$ cat crypy.in
[basic]
algo = RS
calc_code = QE
tot_struc = 20
natot = 8
atype = Si
nat = 8
nstage = 2
njob = 2
jobcmd = sbatch
jobfile = job_cryspy

[lattice]
minlen = 4
maxlen = 10
dangle = 20
mindist_1 = 1.8

[QE]
qe_infile = pwscf.in
qe_outfile = pwscf.out
kppvol = 40 60

[option]
```

インプットファイルはPythonのConfigParserクラスを用いて読み込んでいる。ConfigParserクラスの書き方に従う。

注釈

Python 3では ConfigParserはconfigpaerserに改名されている。

`cryspy.in` は各セクションごとに書く。

5.1. [basic]セクション

名前	値	デフォルト	説明
`algo`	`RS` , `BO` , `LAQA` , (`EA`)		探索アルゴリズム

名前	値	デフォルト	説明
calc_code	VASP , QE , soiap , LAMMPS		構造最適化に用いるプログラムコード
nstage	int		各構造における最適化のステージ数
njob	int		同時にサブミットするジョブの数
jobcmd	str		ジョブをサブミットするときのコマンド名
jobfile	str		ジョブをサブミットするときのスクリプト名

5.1.1. algo

探索アルゴリズムを指定する。バージョン0.6.4では

- RS : Random Search
- BO : Bayesian Optimization
- LAQA : Look Ahead based on Quadratic Approximation

が使用可能で、次のバージョン0.7.0からは

- EA : Evolutionary Algorithm

も使えるようになる。途中で変更不可能。

5.1.2. calc_code

構造最適化に使用するプログラムコードを指定する。

- VASP : VASP (https://www.vasp.at (https://www.vasp.at))
- QE : Quantum Espresso (http://www.quantum-espresso.org (http://www.quantum-espresso.org))
- soiap : soiap (https://github.com/nbsato/soiap (https://github.com/nbsato/soiap))
- LAMMPS : LAMMPS (http://lammps.sandia.gov (http://lammps.sandia.gov))

今のところ、途中で変更不可能。 VASPとQuantum Espressoは有名な第一原理計算プログラム。 soiapは原子間ポテンシャルによる構造最適化が行えるソフトウェアで、CrySPYの共同開発者である産総研の佐藤さん（三宅グループ）を中心に開発されている。 LAMMPSは有名な古典力場を用いたソフト。最適化を行うと対称性が崩れるのであまり3次元バルク結晶を扱うのに向いていないかもしれない。

calc_code に対応して cryspy.in 中に VASP なら[VASP]セクション、 QE なら[QE]セクションが必要。 どのようなインプット変数が必要になるかは、CrySPYのドキュメントを参照すること。

5.1.3. nstage

最適化のステージ数を指定する。まずセルを固定して内部座標だけを緩和しておいて、次のステージでセルも含めてフルに緩和するなどの段階的な構造最適化を行うことができる。波動関数のカットオフやk点なども調整すると良い。

5.1.4. njob

同時に最適化する構造の数、つまり同時にいくつジョブを流すかを指定する。 ここで指定した値の数だけカレントディレクトリに work???? というワーキングディレクトリが作られる。

途中で変更することも可能。ただし、数を少なくする場合は問題がある。例えば njob を5で始めて、 work0000 -- work0004 で計算を進めている途中で njob を3に変更すると、プログラムでは njob の数だけしかチェックしにいかない、つまり work0000 -- work0002 までしかチェックしないようになってしまい、 work0003 , work0004 のデータが収集されずに放置される（放置しておいてあとで再開しても良い）。あとで説明する stop_next_struc オプションなどを用いて work0003 , work0004 のデータ収集まで終わらせてから njob の値を減らす方が良い。 njob の値を増やす分には新しいディレクトリが作成されるだけなので問題無い。

5.1.5. jobcmdとjobfile

構造最適化のジョブをサブミットするときのコマンド名とスクリプト名を指定する。

例えば jobcmd = qsub , jobfile = job_cryspy とすれば、CrySPYが qsub job_cryspy というコマンドを実行する。 ジョブ管理システムを使わないのであれば jobcmd を bash などにしておけば良い。 jobfile で指定したファイル名（任意）のスクリプトを calc_in/ ディレクトリに準備する必要がある。

5.2. [QE]セクション

名前	値	デフォルト	説明
`kppvol`	int [int ...]		各ステージのk点のグリッド密度。逆空間でÅ^{-3} 当たりの密度。
`qe_infile`	str		QEのインプットファイル名。
`qe_outfile`	str		QEのアウトプットファイル名。

5.2.1. kppvol

ここで指定された密度に近くなるようにセルの大きさに応じて自動的にk点分割数を決める。VASPの時も同様。Pymatgenの機能を利用している。 目安として100だと十分多い感覚。構造探索には60などのもう少し小さな値を指定しても良いかもしれない。 後で説明する `kpt_check.py` というスクリプトを使えば、分割数を試しに計算することができるので利用しても良い。

ステージ数と同じ分だけ指定する。例えば `nstage = 4`ならば

```
kppvol = 40 60 80 80
```

5.2.2. qe_infileとqe_outfile

QEではインプットファイルとアウトプットファイル名が任意なので、ファイル名をCrySPYに与えておく。 `calc_in/` の `jobfile` で指定したスクリプトに書いたものと同じにしておく。

© Copyright 2017, Tomoki Yamashita.
最終更新: 2020年01月22日

Back to top

チュートリアルと解説 (outline.html)

6. 小技

6.1. 初期構造の追加

6.2. 初期構造の読み込み

6.3. STOP_NEXT_STRUC

6.4. KPPVOLの目安

6. 小技

よく使うその他の機能を幾つか紹介する。

6.1. 初期構造の追加

初期構造を途中から追加して構造数を増やす。 `cryspy.in` を編集して `tot_struc` の値を増やす。

```
$ head -n 5 cryspy.in
[basic]
algo = RS
calc_code = QE
tot_struc = 20
natot = 8
```

```
$ emacs cryspy.in
```

例えば20から30に増加させる。

```
$ head -n 5 cryspy.in
[basic]
algo = RS
calc_code = QE
tot_struc = 30
natot = 8
```

初めに初期構造を生成した時と同じように念のため `job_struc` スクリプトを実行する。構造生成モードに入ると、構造最適化ジョブのサブミットやデータの収集などは一切行わないので、構造最適化のジョブは実行中でも問題はない。ただし、CrySPYの2重実行はいかなる時も禁止なので構造生成中はCrySPYを実行してはいけない。

例えば `qsub` の場合は、

```
$ qsub job_struc
```

終了して `log` ファイルを見ると以下のように10構造追加されているはず。

```
$ tail -n 19 log
CrySPY 0.6.4
Restart cryspy.py


Changed tot_struc from 20 to 30
Save input data in cryspy.stat

# ---------- Append structures
Structure ID      20 was generated. Space group: 135 --> 139 I4/mmm
Structure ID      21 was generated. Space group: 128 --> 123 P4/mmm
Structure ID      22 was generated. Space group:  47 -->  47 Pmmm
Structure ID      23 was generated. Space group: 110 --> 110 I4_1cd
Structure ID      24 was generated. Space group:  93 -->  93 P4_222
Structure ID      25 was generated. Space group:  67 -->  69 Fmmm
Structure ID      26 was generated. Space group:  18 -->  18 P2_12_12
Structure ID      27 was generated. Space group: 124 --> 124 P4/mcc
Structure ID      28 was generated. Space group: 118 --> 119 I-4m2
Structure ID      29 was generated. Space group: 166 --> 166 R-3m
```

6.2. 初期構造の読み込み

最初のCrySPYの実行で初期構造生成から始めるのではなく、 あらかじめ作成しておいた初期構造データをロードすることができる。

共通の初期構造を用いて、異なるアルゴリズムで探索を比較したり、 自分で作成した初期構造をCrySPYに読み込ませたい時に使う。

新たにテスト用のディレクトリを作成して試してみよう。

```
$ cd ..
$ ls
QE_RS_Si8  backup_00
$ mkdir test_init_load
$ cd mkdir test_init_load/
```

必要なファイルをコピーして、 `cryspy.in` を編集して、[option]セクションに `load_struc_flag = True` を追加する。

```
$ cp -r ../QE_RS_Si8/calc_in/ .
$ cp -r ../QE_RS_Si8/cryspy.in .
$ cp -r ../QE_RS_Si8/cryspy.sh .
$ ls
calc_in  cryspy.in  cryspy.sh
$ emacs cryspy.in
$ tail cryspy.in
dangle = 20
mindist_1 = 1.8

[QE]
qe_infile = pwscf.in
qe_outfile = pwscf.out
kppvol = 40 60

[option]
load_struc_flag = True
```

`data/pkl_data/` ディレクトリに初期構造データを `init_struc_data.pkl` というファイル名で用意する。 今回はそのままコピーして試してみる。

```
$ mkdir -p data/pkl_data
$ cp ../QE_RS_Si8/data/pkl_data/init_struc_data.pkl ./data/pkl_data/
```

`init_struc_data.pkl` のフォーマットなどはCrySPYのドキュメントを参照すること。 注意点として、 `tot_struc` で指定した構造数とデータの構造数を合わせないとエラーになる。

CrySPYを実行する。構造生成ではなくロードなので時間はかからない。 `cryspy.sh` を使って行う。

```
$ bash cryspy.sh &
$ cat log
2018/11/15 17:18:19
CrySPY 0.6.4
Start cryspy.py

Read input file, cryspy.in
Write input data in cryspy.out
Save input data in cryspy.stat

# --------- Load initial structure data
Load ./data/pkl_data/init_struc_data.pkl
```

`log` ファイルにこのように出力されていれば初期構造をロードできている。

6.3. stop_next_struc

今計算中の構造に関しては最適化を終了させてデータ収集までやりたいが、次の構造へは進まないようにしたい時は次のように `cryspy.in` でオプションを設定する。 次の構造へは進まないが、次のステージへは進むので注意。

`cryspy.in` :

```
...
...
[option]
stop_next_struc = True
```

6.4. kppvolの目安

`~/CrySPY_root/CrySPY-0.6.4/utility/kpt_check.py` を使えば、あらかじめk点分割数を計算することができるので、試してから `kppvol` の値を設定すると良い。

以下のように使用する。

```
$ /path_to_python/python ~/CrySPY_root/CrySPY-0.6.4/utility/kpt_check.py ./data/pkl_data/init_str

# ---------- 0th structure
a = 5.8253428658
b = 5.8253428658
c = 5.2026970229
    Lattice vector
5.825343 0.000000 0.000000
0.000000 5.825343 0.000000
0.000000 0.000000 5.202697

kppvol:  100
k-points:  [5, 5, 5]


# ---------- 1th structure
a = 4.693758669650065
b = 4.693758669650065
c = 4.693758669650065
    Lattice vector
-2.347990 2.347990 3.317417
2.347990 -2.347990 3.317417
2.347990 2.347990 -3.317417

kppvol:  100
k-points:  [6, 6, 6]


# ---------- 2th structure
a = 4.7451945734
b = 4.7451945734
c = 9.4938288189
    Lattice vector
4.745195 0.000000 0.000000
0.000000 4.745195 0.000000
0.000000 0.000000 9.493829

kppvol:  100
k-points:  [6, 6, 3]


# ---------- 3th structure
a = 4.8690653213
b = 4.869065321247598
c = 4.9111279454
    Lattice vector
4.869065 0.000000 0.000000
-2.434533 4.216734 0.000000
0.000000 0.000000 4.911128

kppvol:  100
k-points:  [6, 6, 6]


# ---------- 4th structure
a = 8.6335570774
b = 8.6335570774
c = 7.8518827404
    Lattice vector
8.633557 0.000000 0.000000
0.000000 8.633557 0.000000
0.000000 0.000000 7.851883

kppvol:  100
k-points:  [3, 3, 3]
```

第一引数で構造データを、第二引数で `kppvol` の値を指定する。 デフォルトでは、はじめの5構造だけチェックを行うようにしている。 チェックする構造の数を変更したい場合は `-n` オプションを用いる。 `init_struc_data.pkl` だけではなく、VASPのPOSCARやCONTCARにも使用可能。 詳細はCrySPYのドキュメントを見ること。

Back to top

チュートリアルと解説 (outline.html)

7. データ解析と可視化

7.1. CRYSPY_ANALYZER_RS.IPYNB

7. データ解析と可視化

CrySPYでは `data` ディレクトリ直下にテキスト形式で計算結果を出力している。 それを見ることで結果を確認できるし、解析にも使えるのであるが、 それらの元データは `data/pkl_data/` にpickleを使って保存してある。

保存データをPythonで読み込んでグラフを描くためのJupyter notebookがCrySPYの `utility` ディレクトリに入っているのでそれを使う。

ここではスパコンなどで計算したデータをローカルのPCにダウンロードして、ローカルPCでデータ解析を行うことを想定している。 ローカルPCにCrySPYで必要になるPythonライブラリに加えて、jupyterとmatplotlibが必要になる。

7.1. cryspy_analyzer_RS.ipynb

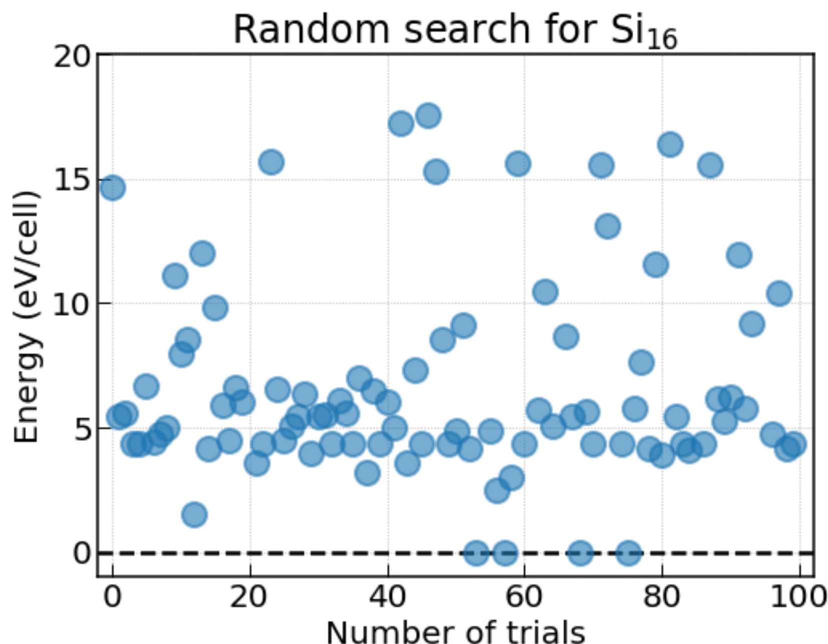
CrySPYに同梱されているノートブックでランダムサーチの解析、可視化に使える。

https://github.com/Tomoki-YAMASHITA/CrySPY/blob/master/utility/cryspy_analyzer_RS.ipynb (https://github.com/Tomoki-YAMASHITA/CrySPY/blob/master/utility/cryspy_analyzer_RS.ipynb)

基本的な使い方は、ノートブックを `data` ディレクトリにコピーして、 Jupyterで開く。

あとはパスを確認しながら順番にセルに書かれたスクリプトを実行すれば良い。

今回のチュートリアルで扱ったものとは別の結果であるが、Si 16原子で100構造探索した結果を例にすると、以下のようなグラフが得られる。



（注）次のバージョン0.7.0で同梱予定のノートブックを使用した。少しグラフの設定が異なるかもしれない。

© Copyright 2017, Tomoki Yamashita.
最終更新: 2020年01月22日

Back to top