README.md

PSCF Particle Phase Field Generator

A tool to generate PSCF initial guess files for bulk morphologies involving assemblies of 3D spherical or 2D cylindrical particles.

NOTE: This is a beta release. See Notes section at the bottom of this file for special assumptions made in its operation.

Contents

Requirements

Back to Top

Use of this tool requires Python 3.5 or later as it makes use of some of the newer additions to the standard library. This tool has been developed using the Anaconda distribution of Python 3.7 on MacOS, Ubuntu Linux, and Unix. Because of this, **Python 3.7 is the minimum recommended version**; earlier versions may encounter unanticipated issues with back-compatibility.

The following required modules should be included in most Python distributions (Version 3.5 or later) as part of the standard library, and should not require additional installation. They should, however, be available in the active python environment.

- abc
- argparse
- copy
- enum
- itertools
- pathlib
- re
- string
- sys

In addition, the following libraries are also required:

- numpy
- scipy
- sympy

All three of these libraries are included standard with Anaconda Python. For installation instructions for other Python distributions, see the project sites for these packages.

To check that the modules are available in the current python environment, open an interactive python terminal and attempt to import each library using `import [library_name]` such as `import abc` .

Installation

Back to Top

Obtaining Source Code

Back to Top

The source code for the tool is hosted on Github. The easiest way to obtain the code is with a git version control client. If such a client is installed on your computer, first `cd` into the directory in which you want to place the pscfFieldGen root directory. From there, the command

```
$ git clone https://github.umn.edu/case0234/pscfFieldGen.git
```

will create a complete working copy of the source code in a subdirectory called `pscfFieldGen/` . Users without a git client can download a `.zip` folder from the Github website and extract its contents into an analogous folder.

Modifying Search Paths

Back to Top

To allow the operating system and python interpreter to find the pscfFieldGen program, you will have to make some modifications to environment variables.

Adding to PYTHONPATH

Back to Top

Many python installations make use of the environment variable PYTHONPATH when searching for modules. To add pscfFieldGen to this search path, use the following command

```
$  PYTHONPATH=$PYTHONPATH:/path/to/root/pscfFieldGen
```

Executing this on the command line only modifies the path until the end of the terminal session. To make the change permanent, add the above command to the file ~/.bashrc (on linux) or to ~/.profile (on Mac OS).

Anaconda Python

Back to Top

With Anaconda Python and other conda-managed environments, changes to the PYTHONPATH environment variable often are not reflected in the python interpreter's effective path. Instead, one must add pscfFieldGen to the environment's site-packages. If you use multiple environments, activate the one you wish to install to using `conda activate` before proceeding.

The easiest way to add the tool to site-packages, is using the conda-develop command included in the conda-build package. Install this package using

```
$ conda install conda-build
```

When that installation completes, enter the following command

```
$ conda-develop /path/to/root/pscfFieldGen
```

where "/path/to/root/" represents the absolute path to the directory from which you cloned the git repository, as that folder should then contain the pscfFieldGen/ subdirectory. This will create a file called 'conda.pth' in the environment's site-packages which will contain the path you gave in the last command.

You can also complete this step manually. To do so, first navigate to your environment's site-packages directory. For Anaconda's base environment, this is located at

```
/path/to/anaconda/lib/pythonX.X/site-packages/
```

where "/path/to/anaconda" is the path to anaconda's installation directory (commonly ~/anaconda3 or similar), and "X.X" represents your Python version. Other environments would be found at

```
/path/to/anaconda/envs/{NAME_OF_ENVIRONMENT}/lib/pythonX.X/site-packages/
```

Finally, if you have saved an environment outside of the main Anaconda file tree (for example, to a user home directory tree on a shared supercomputing system), this would be located instead at

```
/path/to/environment/lib/pythonX.X/site-packages/
```

Once in the site-packages directory, create a `.pth` file containing the path to pscfFieldGen. This file can be named anything, as long as it ends with `.pth` . A name such as "pscfFieldGen.pth" is one possibility.

Running pscfFieldGen

Back to Top

Running the software requires 2 files:

- A Model file specifying filenames and particle positions.
- A PSCF parameter file.

In order to simplify input for the user, crystallographic and composition information are taken from a PSCF parameter file. Detailed information about the model file is provided in the next subsection.

After the tool has been installed, and is discoverable by your Python interpreter, and after you have produced the two necessary input files, the program can be run using the command

```
$ python -m pscfFieldGen -f model_file
```

In the above command, the -m flag tells the python interpreter to look for the module's `__main__.py` script. The -f flag tells the program that the model file is about to be specified, and 'model_file' represents the name of your model file.

pscfFieldGen can also be called with a -t or --trace flag to print a detailed trace of the software execution to the terminal. This would echo the data read from the model file, as well as the Lattice, and crystal structure details. In order to redirect this trace to a file, the command can be executed as:

```
$ python -m pscfFieldGen -f model_file -t > trace_file
```

where "trace_file" is the name of the file storing the trace data.

Example files for a range of calculations are included in the `examples` directory in the root of the project repository.

Model File

Back to Top

The Model file acts as the primary input for the program. Data in this file is specified by *case-sensitive* keywords. Formatting is flexible, requiring only that individual entries be separated by some amount of whitespace (spaces, tabs, newlines).

Below is an example of what the contents of a model file might look like for a BCC phase.

```
software            pscf
parameter_file      param_kgrid
output_file         rho_kgrid

core_monomer        0

coord_input_style   basis
N_particles         2
particle_positions
        0.0     0.0     0.0
        0.5     0.5     0.5

finish
```

Four fields are required:

- `software` : This keyword would be followed by a flag indicating the PSCF version this execution is targeting. Currently flag *pscf* (for the Fortran version) and *pscfpp* (for the C++/Cuda versions) are the only acceptable entries. This should be the first entry in the model file, and is required before specifying `parameter_file` .
- `parameter_file` : This keyword would be followed by a single file name referencing the parameter file. The 'file name' in this case can be any path that would allow the file to be found from the current directory.
- `N_particles` : This keyword is followed by an integer giving the number of particles whose positions will be specified in this input file.
- `particle_positions` : This keyword would be followed by a list of fractional coordinates for each particle. For a 2-Dimensional system, this means 2(`N_particles`) coordinates are expected. For a 3-Dimensional system, 3(`N_particles`) coordinates are expected. Both `parameter_file` and `N_particles` must be specified before `particle_positions` .

Three additional fields are recommended, but not required. If omitted, default values will be assumed, and a warning message will be printed informing the user that a default will be used. Each of these fields can be specified anywhere in the file, but a convention for each is given in its description.

- `output_file` : This keyword is followed by a single file name to which the generated field should be written. As with `parameter_file` , this can be any path recognizable from the current directory. When specified, it is recommended that you place this field immediately following the `parameter_file` specification. If not specified, the program defaults to a file 'rho_kgrid'.
- `coord_input_style` : This keyword is followed by one of two flags, *motif* or *basis*. If *motif* is specified, the given particle positions will be used along with space group symmetry to generate a full list of particles in the unit cell. If *basis* is used, the given particle positions are assumed to be the full set of particles in the unit cell. When specified, it is advised to specify it immediately before `N_particles` . If omitted, the default is *motif*.
- `core_monomer` : This keyword specifies, by monomer id, which monomer should be taken to form the core of the particles in the assembly. Monomers are indexed starting at 0 and counting up. (This numbering differs from the Fortran numbering, which starts at 1). The default value is 0. If specified, it is typically included after the file names, and before structure information.

Finally, the keyword `finish` is followed by no data and identifies the end of the model file. Use of the `finish` keyword is entirely optional, and is included as an aesthetic option for users who prefer to have explicit file termination markers.

Presence of any unrecognized keywords will raise an error and terminate the program.

Parameter File

Back to Top

For detailed information regarding the parameter file format, please see the User manual for the specific version of PSCF.

System specifications are taken from a PSCF Parameter File. This is done to simplify user input, with the assumption that the user will first generate the parameter file for the desired calculation, and use it to generate the initial guess.

Special Notes

Back to Top

Input particle positions should be precise to at least 4 decimal places: When generating the unit cell structure, particle positions are considered identical when all components of fractional coordinate position differ by less than 0.001. If multiple symmetry operation sequences yield a position that has been seen before (or after consecutive applications of the same operation yield the original position) the new position is rejected to avoid duplicates. For precise coordinates (such as 0.0, 0.25, or 0.5), this will not cause a problem; the imprecision from truncated values (such as 0.3333, or 0.6667) will cascade through symmetry operations resulting in some error in the resultant positions. When values are truncated above the positional tolerance, duplicate particles can be missed. *User control of this tolerance can later be added*.

For blends, handling of Grand Canonical Ensemble (or input chemical potentials) is experimental: For both the Fortran and C++/Cuda versions of PSCF, this generator will handle inputs in the Grand Canonical Ensemble (Fortran version) as well as any combination of *phi* and *mu* specified polymers (C++/Cuda version). Cases using explicit system composition (Canonical ensemble; *phi* specified for all species) are considered the normal case for this software. When molecules are specified by chemical potential (Grand Canonical; *mu* specified for more than one molecule), their contribution to the volume fraction of monomer species is calculated assuming that all Grand Canonical species are present in equal number (same number of moles) and share the volume fraction not granted to a canonical (*phi*-specified) species. For calculation of monomer volume fractions, this is analogous to treating a single, canonical "molecular complex" species (which contains one molecule of each Grand Canonical species) with sufficient *phi* to result in total volume fraction of 1. Reliability of this approach (particularly for mixtures with solvents or significantly uneven polymer sizes) has not yet been robustly tested, and field guesses involving these inputs should be carefully inspected before use.

Back to Top