Genarris

Release 2.0

Rithwik Tom, Timothy Rose, Imanuel Bier

Sep 22, 2019

Contents

CHAPTER 1

Installation

1) Setup MPI and MKL If already installed and modules exist, load them after unloading all conflicting modules. Note, in this installation tutorial we will use intel including intel's parallel studio package, but other program environments such as gnu will also work. e.g.:

```
module unload gnu
module unload openmpi
module load intel
module load impi
```

If MKL and MPI are already installed but modules do not exist, include the MPI and MKL directories in your environment variables. e.g.:

```
#Change to your parallel studio path
export $intel=/opt/ohpc/pub/intel/intel18/compilers_and_libraries_2018.3.222/linux
export $intel_parent=/opt/ohpc/pub/intel/intel18

export PATH="$intel/mpi/intel64/bin_ohpc:\
$intel/mpi/intel64/bin:$intel/bin/intel64:$PATH"

export LD_LIBRARY_PATH="$intel/mpi/intel64/lib:$intel/mpi/mic/lib:\
$intel/compiler/lib/intel64:$intel/compiler/lib/intel64_lin:\
$intel/ipp/lib/intel64:$intel/mkl/lib/intel64_lin:\
$intel/tbb/lib/intel64/gcc4.1:\
$intel_parent/debugger_2018/iga/lib:\
$intel_parent/debugger_2018/libipt/intel64/lib:\
$intel/daal/lib/intel64_lin:$intel/tbb/lib/intel64_lin/gcc4.4"
```

Also export LD_PRELOAD to load the parallel studio MKL and Scalapack so importing FHI-aims and numpy does not cause conflict. e.g.:

```
export LD_PRELOAD="$intel/mkl/lib/intel64_lin/libmkl_intel_lp64.so:\
$intel/mkl/lib/intel64_lin/libmkl_sequential.so:\
$intel/mkl/lib/intel64_lin/libmkl_core.so:\
$intel/mkl/lib/intel64_lin/libmkl_blacs_intelmpi_lp64.so:\
```

(continues on next page)

(continued from previous page)

```
$intel/mkl/lib/intel64_lin/libmkl_scalapack_lp64.so:\
$intel/mpi/intel64/lib/libmpi.so.12"
```

2) create a python 3.5+ virtual environment e.g.:

```
#Change this to your desired anaconda install path
export $anaconda=${HOME}/anaconda
mkdir $anaconda
cd $anaconda
```

download and install anaconda e.g.:

```
wget https://repo.anaconda.com/archive/Anaconda3-2019.07-Linux-x86_64.sh
chmod +x Anaconda3-2019.07-Linux-x86_64.sh
./Anaconda3-2019.07-Linux-x86_64.sh
```

Include anaconda's binary in PATH e.g.:

```
export PATH=$anaconda/anaconda3/bin:$PATH
```

Make a python environment called e.g. genarris_env by installing intelpython3_core. e.g.:

```
conda config --add channels intel
conda create -n genarris_env intelpython3_core python=3
```

3) direct your path variables to include the new env e.g.:

```
export PYTHONPATH="$anaconda/anaconda3/envs/genarris_env/lib/python3.6:\
$anaconda/anaconda3/envs/genarris_env/lib/python3.6/site-packages:\
$PYTHONPATH"

export PATH="$intel/mpi/intel64/bin_ohpc:$intel/mpi/intel64/bin:\
$intel/bin/intel64:$anaconda/anaconda3/envs/intelpython3_full/bin:\
$anaconda/anaconda3/bin:$PATH"
```

4) Extract Genarris_v2.tar.gz into a desired directory and enter it e.g.:

```
export $genarris=${HOME}/genarris
mkdir $genarris
cp Genarris_v2.tar.gz $genarris
cd $genarris
tar -xzf Genarris_v2.tar.gz
```

5) Install Genarris. Note, one reason we recommend to create a python virutal env earlier is that running this installation script will remove the ase installation (if any) in the currently active python environment. e.g.:

```
cd $genarris/Genarris
python setup.py install
```

Genarris is now installed. We will first test that Genarris imports and MPI is working correctly with the following test and then the next step will be to compile FHI-aims as a python-importable library if you desire to use FHI-aims.

6) Test that Genarris imports and MPI is working correctly. Modify the submission script for your backend (here, we used slurm).:

```
cd $genarris/documentation/mpi_and_genarris_test
sbatch mpi_and_genarris_test.sh
```

The desired output is that each rank reports a unique number.

7) Compile libaims into a python-importable library

Set ulimit to avoid any possible memory problems:

```
ulimit -s unlimited
ulimit -v unlimited

# Set OMP_NUM_THREADS to 1
export OMP_NUM_THREADS=1
```

Obtain FHI-aims from https://aims-git.rz-berlin.mpg.de/aims/FHIaims If you don't have permissions, ask Volker Blum at volker.blum@duke.edu:

```
export $aims=${HOME}/aims  #Change to your desired location for FHI-aims
```

In its src directory ($aims/src), make sure the Makefile has all compilation flags (user defined settings) commented out. Copy the make.sys file in the documentation directory of Genarris into FHI-aims' src directory. The make.sys is pasted here for reference.:

```
cp $genarris/documentation/make.sys $aims/src
```

Note, this make.sys assumes you are using intel's parallel studio and that your cluster's backend is intel. If this isn't the case, you'll need to set the flags accordingly.:

```
# make.sys
###############
# Basic Flags #
###############
FC = mpiifort
FFLAGS = -O3 -ip -fp-model precise -fPIC
F90FLAGS = $(FFLAGS)
ARCHITECTURE = Generic
LAPACKBLAS = -L${MKLROOT}/lib/intel64 \
             -lmkl_intel_lp64 \
             -lmkl_sequential \
             -lmkl_core \
             -lmkl_blacs_intelmpi_lp64 \
             -lmkl_scalapack_lp64
F90MINFLAGS = -O0 -fp-model precise -fPIC

#########################
# Parallelization Flags #
#########################
USE_MPI = yes
MPIFC = ${FC}
SCALAPACK = ${LAPACKBLAS}

###############
# C,C++ Flags #
###############
CC = icc
CFLAGS = -O3 -ip -fp-model precise -fPIC
```

Compile FHI-aims as a shared library object:

```
cd $aims/src
make -j 20 libaims.scalapack.mpi
```

where the `20` is however many cores you'd like to use for compilation.

Make a directory for compiling FHI-aims as a python library e.g.:

```
mkdir $aims/aims_as_python_lib
cd $aims/aims_as_python_lib
```

# Copy the Makefile and aims_w.f90 in the Genarris documentation directory to this directory. A copy of it has been pasted here for reference. Note that you will need to change the libaims version (currently shown as 190522). Again, you'll need to change the f90exec and/or fcompiler flags if your backend is not intel. aims_w.f90 is a wrapper script to interface with FHI-aims. e.g.:

```
cp $genarris/Genarris/documentation/Makefile $aims/aims_as_python_lib
cp $genarris/Genarris/documentation/aims_w.f90 $aims/aims_as_python_lib
```

Create the Makefile with the following contents:

```
LIBAIMS=${aims}/lib/libaims.190522.scalapack.mpi.so
include_dir=${anaconda}/anaconda3/envs/genarris_env/include

aims_w.so: aims_w.f90
    f2py --f90exec=mpiifort --fcompiler=intelem -m aims_w \
         -c aims_w.f90 ${LIBAIMS} -I${include_dir}

clean:
    rm aims_w.*.so
```

Compile FHI-aims as an importable python library!:

```
make
```

8) Test that FHI-aims can run a job Modify the submission script in the `$genarris/documentation/aims_test` directory to run on your backend (here we used slurm).:

```
export PYTHONPATH=$PYTHONPATH:$aims/aims_as_python_lib
cd $genarris/documentation/aims_test
sbatch aims_test.sh
```

CHAPTER 2

Introduction to Running Genarris

2.1 Configuration File

Genarris is a random crystal structure generation code that can be adapted to perform *ab initio* crystal structure prediction. The modularity of Genarris is achieved through the sequential execution of procedures. The execution of Genarris is controlled by a configuration file. Below is a small example of a configuration file for Genarris.:

```
[Genarris_master]
procedures = ["Pygenarris_Structure_Generation"]

[pygenarris_structure_generation]
# Path to the single molecule file to used for crystal structure generation
molecule_path = relaxed_molecule.in
# Number of cores (MPI ranks) to run this section with
num_cores = 56
# Number of OpenMP Threads
omp_num_threads = 2
num_structures = 5000
Z = 4
sr = 0.85
tol = 0.00001
max_attempts_per_spg_per_rank = 1000000000
geometry_out_filename = glycine_4mpc.out
output_format = json
output_dir = glycine_4mpc_raw_jsons
```

Sections of the configuration file are denoted by square brakets, `[...]`. All parameters that are specified below a section are called **options**. The workflow of Genarris can be precisely controlled by the user by specifying the order of the desired procedures in `[Genarris_master]`. The user must also include the corresponding section for each procedure listed in `[Genarris_master]`. Each section may have many options which are required, optional, or inferred.

This document details the options for procedures that are executed in the Genarris 2.0 *Robust* workflow. In order these are:

```
["Relax_Single_Molecule",
 "Estimate_Unit_Cell_Volume",
 "Pygenarris_Structure_Generation",
 "Run_Rdf_Calc",
 "Affinity_Propagation_Fixed_Clusters",
 "FHI_Aims_Energy_Evaluation",
 "Affinity_Propagation_Fixed_Clusters",
 "Run_FHI_Aims_Batch"]
```

There are many options that can be specified and modified for each section. All of these options are specified in this document under the **Configuration File Options** section of each procedure. For a detailed description of the workflow, see the *detailed instructions* section.

2.2 Option Category

There are three *categories* of **Configuration File Options**. These are *required*, *optional*, and *inferred*. In the **Configuration File Options**, these categories are specified after the *type* of the option, such as *int*, *float*, or *bool*.

1. *Required* options have no category placed after the type in the documentation. These options are required to be in the configuration file for execution of Genarris.
2. *Optional* arguments are specified after the option *type*. These areguments have default settings built into the code perform well in general. The user may specify these *optional* arguments in the configuration file to have more control over the program executing.
3. *Inferred* options are specified after the option *type*. These options may be present in multiple different procedures. For example, the option `aims_lib_dir` is needed in the `Relax_Single_Molecule`, `FHI_Aims_Energy_Evaluation`, and `Run_FHI_Aims_Batch`. But, because it is an inferred parameter, it only needs to be specified once in the earliest procedure in which occurs and then it will be inferred by all further procedures. Options which are inferred are thus optional in all proceeding sections.

2.3 Output Formats

There are three output formats supported within the Genarris source code. These are *json*, *geo*, or *both*.

- The *json* file format is the native structure file format for Genarris. This file format supports storing the structure ID, the geometry, and property information.
- The *geo* file format is the file format support by FHI-aims. Additionally, this file format is support by Jmol , a 3D chemical structure visualizer, and by ASE, the atomic simulation environment tools written for Python.
- The user may also specify *both*, in which case both the *json* file and *geo* file for every structure will be produced.

CHAPTER 3

Running Genarris Tutorial

3.1 Quick start

`cd` to the tutorial/RDF directory and modify `aims_lib_dir` in `ui.conf` to point to the directory containing your aims library wrapper file (the one compiled with f2py). Adapt `sub_genarris.sh` to your cluster schdueling submission script type (the example is slurm) and options (slurm options, mpi executable, number of cores etc.). Then submit e.g.:

```
sbatch sub_genarris.sh
```

3.2 Input options in ui.conf

See *documentation*.

3.3 Description of Log Files

There are multiple log files created when running Genarris. The files are separated by the contents they contain. This makes debugging easier, for example, because all error information is saved in a single location.

- `Genarris.log`: A log of what is currently being run and other info is printed here. The amount of info can be made less verbose by commenting out the verbose option in the ui.conf for the various procedures.
- `Genarris.err`: Error messages may appear here.
- `stdout`: Named something different depending on your submission script, this is the standard output which may contain environment info, cgenarris output log info, and sometimes error messages.

3.4 Detailed Instructions

Genarris will run the procedures specified by the procedures option in the Genarris_master section in the order they appear in the list. It begins with the Relax_Single_Molecule procedure which creates a folder called structure_dir_for_relaxing_single_molecule to store the molecule geometry file. Calls to FHI-aims create a folder structure starting with the folder name inputted with the aims_output_dir option. That folder contains a folder for every structure in the inputted structure directory (in this case, there is just one structure). The inputted control file is copied to each of those subfolders. A copy of the geometry file in FHI-aims and json format is also copied to the corresponding subdirectory. Genarris replicas move from folder to folder, performing an FHI-aims calculation in each one. This creates the aims output file aims.out and possibly a relaxed geometry file geometry.in.next_step. Genarris will look to see if the single molecule was relaxed and if so, use that geometry in subsequent procedures.

When pygenarris is run, each core will output structures to its own geometry.out file. Each of these are geometry.in format concatenated. When pygenarris completes, these individual files will be appended to a single geometry.out file if desired and each structure will be output to the output_dir specified as a json file. A json file is like a python dictionary which contains key, value pairs for metadata about the structure and is required for subsequent steps. pygenarris may also output the cutoff_matrix which contains distance cutoff values between atoms i and j which are derived from the sr inputted (see the paper for more details). Because the number of structures generated currently must be a multiple of the number of allowed space groups for the given molecule and Z, we have:

```
num_structures_per_allowed_SG_per_rank =
                int(np.ceil(float(num_structures) /
                (float(comm.size) * float(num_compatible_spgs))))
```

and so the total number of structures generated could be more than the number specified in ui.conf. See the documentation, but there is an option for choosing to keep them all or only select the num_structures structures desired. Structures are niggli reduced before being output to jsons.

Then the Run_Rdf_Calc procedure is run. It yields a directory of jsons specified by its output_dir option. These jsons are the same as the ones output by Pygenarris except now they have the RDF vector as a recorded piece of metadata. A distance matrix is also output in the form of a memory map which drastically saves on memory usage.

Alternatively, the RCD procedures may be run. RCD_Calculation creates an output_dir with the jsons including their RCD vectors. It also outputs some other log files: RCD_report.out and rcd_vectors.info. RCD_Difference_Folder_Inner will compute the pairwise distances between all structures and output a distance matrix in the form of a memory map.

Next, Affinity Propagation begins by printing the affinity matrix that corresponds to the distance matrix outputted in the previous step. It then outputs a directory with all structures in the raw pool, but now they include more metadata such as the cluster id that AP assigned it to as well as the exemplar of its cluster. AP also outputs a directory of the exemplars, and the distance matrix of those exemplars which has the same name as the first distance matrix file name but with a 1 appended to the name.

The next call to FHI-aims computes the energies of the exemplars outputted in the previous step. It creates an aims_output_dir with name specified in the ui.conf. The resultant jsons are then dumped to the corresponding output_dir which are the same as the exemplars but now have the energy property included.

Then, AP creates the affintity matrix corresponding to the second distance matrix and clusters the structures with energies and outputs a directory for all those structures but now they contain the cluster assigned by this AP. The tutorial asks the second round of clustering to output the structure with the minimum energy from each cluster. These are the structures output to sample_structures_exemplars_2.

These structures are relaxed in the subdirectories of aims_output_dir for Run_FHI_Aims_Batch. The relaxed structures are then niggli reduced and are output to this section's output_dir. The structures output to output_dir also contain other metadata such as spglib's new determination of the space group.

CHAPTER 4

Genarris 2.0 Procedures for Robust Workflow

4.1 Description

This section details all arguments and configuration file options for the procedures executed by the Robust Genarris 2.0 workflow. Each procedure is a class function of the of the `Genarris` master class. The documentation follows a standard format for each procedure. The name of the procedure is given first followed by a short description of the function the function it performs. Below the description is the the configuration file options subsection. This section gives the name, the data type, the *Option Category*, and a description of each option which is accepted by the procedure. By referencing this documentation, the user can obtain precise control over the execution of Genarris procedures.

4.2 Genarris Procedures

class `Genarris.genarris_master.`**`Genarris`**(*inst_path*)

Master class of Genarris. It controls all aspects of the Genarris workflow which can be executed individually or sequantially. Begins by reading and intepreting the configuration file. Calls the defined procedures with the options specified in the configuration file. Some options may be inferred from previous sections if they are not present in every section.

`Affinity_Propagation_Fixed_Clusters`(*comm*)

AP that explores the setting of preference in order to generate desired number of clusters.

Configuration File Options

output_dir [str] Path to the directory where the chosen structures will be stored.

preference_range [list] List of two values as the [min, max] of the range of allowable preference values.

structure_dir [str, inferred] Path to the directory of files to be used for the calculation. Default is to infer this value from the previous section.

dist_mat_input_file [str, inferred] Path to the distance matrix output from the descriptor calculation. Default is to infer this value from the previous sections.

output_format [str, optional] Format the structure files should be saved as. Default is both.

cluster_on_energy [bool, optional] Uses energy values to determine examplars. Structures with the lowest energy values from each cluster are selected. Default is False.

plot_histograms [bool, optional] If histogram plots should be created of the volume and space groups. Default is False.

num_of_clusters [int or float, optional] Float, must be less than 0. Selects a fraction of the structures. Int, selects specific number of structures equal to int. Default is 0.1.

num_of_clusters_tolerance [int, optional] Algorithm will stop if it has generated the number of clusters within the number of desired clusters and this tolerance. Default is 0.

max_sampled_preferences [int, optional] Maximum number of preference values to try.

output_without_success [bool, optional] Whether to perform output procedures if the algorithm has reached the maximum number of sampled preferences without finding the correct number of clusters. Default is False.

affinity_type [list, optional] List of [type of afinity, value] argument Scikit-Learn AP alogrithm.

affinity_matrix_path [str, optional] Path to the affinity matrix to use for the AP algorithm. Default is `affinity_matrix.dat`.

damping [float, optional] damping argument for Scikit-Learn AP algorithm. Default is 0.5.

convergence_iter [int, optional] convergence_iter argument for Scikit-Learn AP algorithm. Default is 15.

max_iter [int, optional] max_iter argument for Scikit-Learn AP algorithm. Default is 1000.

preference [int, optional] preference argument for Scikit-Learn AP algorithm. Default is None.

verbose_output [bool, optional] verbose argument for Scikit-Learn AP algorithm. Default is False.

property_key [str, optional] Key which the AP cluster will be stored in the properties of each structure object. Default is `AP_cluster`.

output_file [str, optional] Path where info about the AP alogrithm execution will be stored. Default is `./AP_cluster.info`.

exemplars_output_dir [str, optional] If provided, will output the examplars of each cluster to this folder. Default is None.

exemplars_output_format [str, optional] File format of structures to be output. Default is both.

structure_suffix [str, optional] Suffix to apply to structure files which are written. Default is `.json`.

output_dir_2: str, inferred Code automatically looks for the option output_dir_2 if the output directory already exists. This is how the code currently identifies that AP is running for a second time. Default behavior is to not use this option if output_dir does not already exist.

num_of_clusters_2: int or float, optional num_of_clusters for second clustering step. Default value is 0.1.

output_file_2 [str, inferred] Use if running AP algorithm twice, such as in the Robust workflow. Default is to use output_file.

exemplars_output_dir_2 [str, inferred] Exemplars output directory if second clustering step is used. Default is to use exemplars_output_dir.

cluster_on_energy_2 [str, inferred] How to choose examplars for the second clustering step. Default is to use cluster_on_energy value.

energy_name_2 [str, inferred] Energy name to use for second clustering step. Default is to use energy_name.

`Estimate_Unit_Cell_Volume`(*comm*)

Performs volume estimation using a machine learned model train on the CSD and based on Monte Carlo volume integration and topological molecular fragments. See Genarris 2.0 paper for full description.

Configuration File Options

volume_mean [float, optional] If provided, uses this value as the volume generation mean without using the ML model to etimate the volume.

volume_std [float, optional] If provided, uses this value for structure generation, otherwise a default value of 0.075 multiplied by the prediction volume per unit cell is provided.

`FHI_Aims_Energy_Evaluation`(*comm*, *world_comm*, *MPI_ANY_SOURCE*, *num_replicas*)

Runs Self-Consistent Field calculation on a pool of structures.

Configuration File Options

See *`Run_FHI_Aims_Batch()`*

`Pygenarris_Structure_Generation`(*comm*)

Uses the Genarris module written in C to perform structure generation. This module enables generation on special positions.

Configuration File Options

molecule_path [str] Path to the relaxed molecule geometry.

output_format [str, optional, default="json"] Determines the type of file which will be output for each structure. Can be one of: json, geo, both.

output_dir [str] Path to the directory which will contain all generated structures which pass the intermolecular distance checks.

num_structures [int] Target number of structures to generate.

Z [int] Number of molecules per cell to generate.

volume_mean [float, optional] See *`Estimate_Unit_Cell_Volume()`*

volume_std [float, optional] See *`Estimate_Unit_Cell_Volume()`*

sr [float, optional] Defines the minimum intermolecular distance that is considered physical by multiplying the sum of the van der Waals radii of the interacting atoms by sr. Default value is 0.85.

tol [float, optional] Tolerance to be used to identify space groups compatible with the input molecule.

max_attempts_per_spg_per_rank [int] Defines the maximum number of attempts the structure generator makes before moving on to the next space group.

num_structures_per_allowed_SG_per_rank [int] Number of structures per space group per rank which will be generated by Pygenarris.

geometry_out_filename [str] Filename where all structures generated by Pygenarris will be found.

omp_num_threads [int] Number of OpenMP threads to pass into Pygenarris

truncate_to_num_structures [bool] If true, will reduce pool to exactly the number defined by num_structures.

`Run_Rdf_Calc`(*comm*)

Runs RDF calculation for the pool of generated structures. RDF descriptor is similar to that described in Behler and Parrinello 2007. Then calculates the structure difference matrix.

Configuration File Options

structure_dir [str, inferred] Path to the directory of structures to evaluate.

dist_mat_fpath [str] Path to file to write distance matrix to.

output_dir [str] Path of directory to write structures to (will create if it DNE). If 'no_new_output_dir' then input structures will be overwritten.

normalize_rdf_vectors: bool,optional Whether to normalize the rdf vectors over the columns of the feature matrix before using them to compute the distance matrix. Default is Falase.

standardize_distance_matrix: bool If True, standardizes the distance matrix. The method is to divide all elements by the max value in the distance matrix. Because it is a distance matrix and thus all elements are positive, the standardized elements will be in the range [0, 1]. Default is False.

save_envs: bool, optional Whether to save the environment vectors calculated by the RDF method in the output structure files. Default is False.

cutoff [float, optional] Cutoff radius to apply to the atom centered symmetry function. Default is 12.

n_D_inter [int, optional] Number of dimensions to use for each type of pair-wise interatomic interaction found in the structure. Default is 12.

init_scheme [str, optional] Can be centered or shifted, as described in Gastegger et al. 2018. Default is shifted.

eta_range [list, optional] List of two floats which define the range for eta parameter in Gastegger et al. 2018. Default is [0.05,0.5].

Rs_range [list, optional] List of two floats which define the range for Rs parameter in Gastegger et al. 2018. Default is [[0.1,12].

pdist_distance_type [str,optional] Input parameter for the pdist function. Default is Euclidean.

`Relax_Single_Molecule`(*comm*, *world_comm*, *MPI_ANY_SOURCE*, *num_replicas*)

Calls run_fhi_aims_batch using the provided single molecule path.

Configuration File Options

See *`Run_FHI_Aims_Batch()`*

`Run_FHI_Aims_Batch`(*comm*, *world_comm*, *MPI_ANY_SOURCE*, *num_replicas*)

Runs FHI-aims calculations on a pool of structures using num_replicas.

Configuration File Options

verbose [bool] Controls verbosity of output.

energy_name [str] Property name which the calculated energy will be stored with in the Structure file.

output_dir [str] Path to the directory where the output structure file will be saved.

aims_output_dir [str] Path where the aims calculation will take place.

aims_lib_dir [str, inferred] Path to the location of the directory containing the FHI-aims library file.

molecule_path [str] Path to the geometry.in file of the molecule to be calculated if called using harris_single_molecule_prep or relax_single_molecule.

structure_dir [str, inferred] Path to the directory of structures to be calculated if calculation was called not using harris_single_molecule_prep or relax_single_molecule.

Z [int, inferred] Number of molecules per cell.