Garleek Documentation

Release 0.3.1+4.gd26ca5d

Jaime Rodriguez-Guerra
Ignacio Funes-Ardoiz

Feb 12, 2019

User guide

Garleek (recursive acronym for *Garleek is Automated Resources for Layered Energies using External Keyword*) allows you to perform QM/MM calculations by interfacing Gaussian with MM programs via its `external` keyword. See *Supported software* for compatible programs.

> *Wel loved he garleek, oynons, and eek lekes,*
> *And for to drynken strong wyn, reed as blood.*
> – **Geoffrey Chaucer**

CHAPTER 1

Introduction

Garleek allows you to perform QM/MM calculations by interfacing Gaussian with MM programs via its `external` keyword.

1.1 Supported software

Garleek currently supports the following programs. It expects them to be available in `$PATH`.

1.1.1 QM engines

- Gaussian 16
- Gaussian 09A, 09B, 09C, 09D

1.1.2 MM engines

- Tinker 8.1 (only `analyze`, `testgrad` and `testhess` are needed)

1.2 How does it work

Let's explain it with an example: doing an ONIOM calculation with Gaussian and Tinker.

When Garleek does an ONIOM calculation with Gaussian & Tinker, Gaussian handles the majority of the calculations: QM stuff and ONIOM scheme. The MM part is configured with the `external` keyword, which specifies that the MM calculations will be performed with an external program. In this mode, Gaussian will write a series of files to disk and call the requested program.

In this case, it's `garleek-backend`, which will take one of the files Gaussian generates in each iteration (the `*.EIn` file), transform it and pass it to Tinker binaries to obtain the requested data: potential energy, dipole moment, polarizability and/or hessian matrix, depending on the calculation. Gaussian expects these data written back to an `*.EOu` file in a very specific format.

In a nutshell, all Garleek does is interfacing Gaussian and Tinker following the `external` protocol:

1. Parse Gaussian's `EIn` files
2. Convert to whatever the MM engine is expecting
3. Calculate the requested data (defined in the `EIn` header)
4. Convert the obtained MM data to Gaussian units and write it to the `EOu` file

The main difficulty here is making sure that the atom types are understandable by both parts of the scheme: the Gaussian input file should contain atom types compatible with the MM engine. Since this is rarely the case, a previous step is needed: patching the original Gaussian file so it contains adequate atom types. We take advantage of this step to inject the correct `garleek-backend` calls in the `external` keyword so the user does not have to worry about those details. For further information on the practical usage, please read our Tutorials section. First one: *A simple CH4 molecule*.

CHAPTER 2

Installation

Garleek is designed to be as simple as possible, using almost no 3rd party dependencies. You only need Python (2.7 or 3.4+) and NumPy. This can be easily achieved with `pip`:

```
# Stable version
python -m pip install garleek
# Latest, development version (might be unstable)
python -m pip install -U https://github.com/insilichem/garleek/archive/master.zip
```

2.1 Testing

Unit-tests are provided in the `tests/` directory. Structures included are very small (5-20 atoms), so it should run reasonably fast locally. Input files are adjusted for 8 CPUs and 8GB RAM. To run them, you must install additional dependencies:

```
pip install pytest cclib
```

Additionally, the interfaced software must be properly initialized (ie, executables should be available in `$PATH`, any needed environment variable exported, etc.). To do this, you'd normally use `module load` or similar utilities. An example on how to prepare the testing environment is provided in `tests/run.sh`. Feel free to modify it to your own needs.

CHAPTER 3

Usage

Garleek is composed of two small programs:

- `garleek-prepare`, used to patch the QM-provided input file with correct atom types
- `garleek-backend`, which interfaces the QM program with the MM engine to make them understand each other

Usually, `garleek-prepare` will inject the necessary `garleek-backend` calls in the QM input file, so you should only run the following command. Everything else is handled automatically.

```
garleek-prepare --qm gaussian --mm tinker --ff mm3 --types uff_to_mm3 INPUTFILE
```

- `INPUTFILE`: QM input file to be patched and renamed. For example, `file.in` would be renamed as `file.garleek.in`. Currently supported: Gaussian input files (`.gjf`, `.com`, `.in`).
- `--qm`: QM program handling the QM/MM calculation. Currently supported keywords include: *gaussian*, *gaussian_09a*, *gaussian_09b*, *gaussian_09c*, *gaussian_09d*, *gaussian_16*.
- `--mm`: MM engine to be used. Currently supported keywords include: *tinker*, *tinker_qmcharges*. Read below for more details.
- `--ff`: force field to be used in the MM calculation. Supported files and keywords depend on the value of `mm`.
- `--types`: dictionary handling the conversion between the MM atom types provided in the original input file and those the MM engine actually needs for the chosen force field. Supported files and keywords depend on the value of `mm` and `qm`. Usually is a simple two-column plain-text file that contains `Original-MM-type Converted-MM-type` mappings. This file is used to replace the original types in `MyInputFile.in`. Matching is case-insensitive, but the file must list them ALWAYS uppercased.

TIP: Updated CLI arguments will be always available if you run `garleek-prepare -h`.

Note: For more details and specific use-cases, please refer to the tutorials section. Begin with the first one by clicking here: *A simple CH4 molecule*.

3.1 About –qm and –mm tags

These flags (`--qm` and `--mm`) are used to guarantee that the two programs involved are compatible. They expect a value formatted like this:

```
<program>[_<version>]
```

Valid values include `gaussian`, `tinker`, `gaussian_16` and `tinker_8`. The version can be omitted and will default to the latest supported (Gaussian 16 and Tinker 8.2, for example). We use the version to customize the behavior of the `garleek-backend` executable. For example, Gaussian 09A generates slightly different exchange files than Gaussian 09B and above, so we use the version tag to enable a different parser.

The `version` does not need to be a number; in fact, we can use it to provide different behaviors as well. Available versions are listed `garleek/{qm,mm}/<program>.py`. Their behavior is documented below:

3.1.1 Gaussian

- `gaussian`, `gaussian_09b`, `gaussian_09c`, `gaussian_09d`, `gaussian_16`: Default behavior.
- `gaussian_09a`: Different `EIn` parser.

3.1.2 Tinker

- `tinker`, `tinker_8`, `tinker_8.1`: Default behavior.
- `tinker_qmcharges`: Use charges provided in the QM input file, instead of the one available in the PRM or KEY files. For this to work, a new KEY `garleek.charges.key` file is generated on the fly for each iteration, containing the appropriate `CHARGE SERIAL_NUMBER VALUE` lines.

CHAPTER 4

Developers

Note: The following content is autogenerated from the Python docstrings. Don't expect a well written story!

4.1 The package

4.1.1 Code structure

Garleek code is organized in two layers: the library and the application layers. The former controls the business logic while the latter exposes this functionality as a command-line interface and, if useful, a high-level Python function.

Library layer

- `qm` and `mm` subpackages host low-level QM and MM software interfacing. Both feature a documented, standardized dictionary representation on the data each are expecting. The communication logic for each `qm` and `mm` modules is performed in the `connectors` module. Refer to each module documentation for more information.
- `atom_types` controls `atom_type` file parsing, and `units` stores unit conversion factors.

Application layer

- `cli` module lists the CLI entry-points for users (frontend) and QM softwares handling the ONIOM calculation (backend)

4.2 Library layer

4.2.1 mm/

The `mm` subpackage hosts all the code that handles calculations involving MM software.

Each module in this subpackage is expected to perform the following tasks:

1. Take the standardized dictionary (as explained) in the `qm` module and convert the contained data into the representation requested by the interfaced MM software (units included).
2. Calculate the requested data (depending on `derivatives` value) with the MM software.
3. Organize the obtained data into a standardized representation, as described below.

Standardized object for interfaced data

The QM engine will be expecting a dictionary containing the following keys and values. Unit conversion is not handled here (that's responsibility of the *connector*), so just use those employed by the MM software (but document those in the docstring!)

energy [float] Potential energy

gradients [np.array with shape (3*n_atom,)] Gradient on each tom

hessian [np.array with shape (9*n_atom,)] Flattened hessian matrix (force constants)

dipole_moment [np.array with shape (3,), optional] Dipole X,Y,Z-Components

polarizability [np.array with shape (6,), optional] Atom polarizability

dipole_derivatives [np.array with shape (9*n_atom)] Dipole derivatives with respect to X,Y,Z components for each atom

4.2.2 mm.tinker.py

Garleek - Tinker bridge

`garleek.mm.tinker.`**`patch_tinker_output_for_inactive_atoms`**(*results*, *indices*, *n_atoms*)

TODO: Patch 'hessian' to support FREQ calculations with inactive

`garleek.mm.tinker.`**`prepare_tinker_key`**(*forcefield*, *atoms=None*, *version=None*)

Prepare a file ready for TINKER's -k option.

Parameters

- **`forcefield`** (*`str`*) – `forcefield` should be either a:
  - `*.prm`: proper forcefield file
  - `*.key`, `*.par`: key file that can call `*.prm files` and

  add more parameters

  If a .prm file is provided, a .key file will be written to accommodate the forcefield in a `parameters *` call.
- **`atoms`** (*`OrderedDict, optional=None`*) – Set of atoms to write, following convention defined in *`garleek.qm`*.

- **version** (*str, optional=None*) – Specific behavior flag. Supports: - qmcharges, which would write charges provided by QM engine.

  Needs atoms to be passed.

Returns **path** – Absolute path to the generated TINKER .key file

Return type str

garleek.mm.tinker.**prepare_tinker_xyz** (*atoms*, *bonds=None*, *version=None*)

Write a TINKER-style XYZ file. This is similar to a normal XYZ, but with more fields:

```
atom_index element x y z type bonded_atom_1 bonded_order_1 ...
```

TINKER expects coordinates in Angstrom.

Parameters

- **atoms** (*OrderedDict*) – Set of atoms to write, following convention defined in *garleek.qm*.
- **bonds** (*OrderedDict*) – Connectivity information, following convention defined in *garleek.qm*.
- **version** (*str, optional=None*) – Specific behavior flag, if needed. Like 'qm-charges'

Returns **xyzblock** – String with XYZ contents

Return type str

garleek.mm.tinker.**run_tinker** (*xyz_data*, *n_atoms*, *key*, *energy=True*, *dipole_moment=True*, *gradients=True*, *hessian=True*)

4.2.3 qm/

The qm subpackage hosts all the code that handles calculations involving QM software.

All modules listed here are expected to perform the following tasks:

- Patch the INPUT file with proper garleek-backend calls and atom type conversion.
- Parse the intermediate files as provided by the QM software into a standardized object (details below).
- Write the output file expected by the QM software.
- List supported versions and the default ones in two tuples named supported_versions and default_version, respectively.

Standardized object for interfaced data

The intermediate representation of the parsed data, which will be passed to the MM engine, should be a dict with these keys and values:

n_atoms [int] Number of atoms in the structure or substructure

derivatives [int] Calculations requested for the MM part:

- 0: energy only
- 1: calculate gradient
- 2: calculate hessian

These values are cumulative, so if `2` is requested, `0` and `1` should be computed as well.

charge [float] Global charge of the structure

spin [int] Multiplicity of the system

atoms [OrderedDict of dicts] Ordered dictionary mapping atom index with another dictionary containing these values:

- `element` : `str`. Chemical element
- `type` : `str`. Atom type as expected by the MM part
- `xyz` : `np.array` with shape (3,). Cartesian coordinates
- `charge` : `float`. Atom point charge for the MM part

bonds [OrderedDict of lists of 2-tuples] Ordered dictionary mapping atom-index to a list of 2-tuples containing bonded atom index (int) and bond order (float)

4.2.4 qm.gaussian.py

Garleek - Gaussian bridge

class `garleek.qm.gaussian.`**`GaussianPatcher`**(*filename*, *atom_types*, *mm='tinker'*, *qm='gaussian'*, *forcefield=None*, *version='16'*)

Bases: `object`

`patch`()

`garleek.qm.gaussian.`**`parse_gaussian_EIn`**(*ein_filename*, *version='16'*)

Parse the `*.EIn` file produced by Gaussian `external` keyword.

This file contains the following data (taken from http://gaussian.com/external)

```
n_atoms  derivatives-requested  charge  spin
atom_name  x  y  z  MM-charge [atom_type]
atom_name  x  y  z  MM-charge [atom_type]
atom_name  x  y  z  MM-charge [atom_type]
...
```

- `derivatives-requested` can be `0` (energy only), `1` (first derivatives) or `2` (second derivatives).
- `version` must be one of `garleek.qm.gaussian.supported_versions`

`garleek.qm.gaussian.`**`patch_gaussian_input`**(**a*, ***kw*)

`garleek.qm.gaussian.`**`prepare_gaussian_EOu`**(*n_atoms*, *energy*, *dipole_moment*, *gradients=None*, *hessian=None*, *polarizability=None*, *dipole_polarizability=None*)

Generate the `*.EOu` file Gaussian expects after `external` launch.

After performing the MM calculations, Gaussian expects a file with the following information (all in atomic units; taken from http://gaussian.com/external)

Items	Pseudo Code	Line Format
energy, dipole-moment (xyz)	E, Dip(I), I=1,3	4D20.12
gradient on atom (xyz)	FX(J,I), J=1,3; I=1,NAtoms	3D20.12
polarizability	Polar(I), I=1,6	3D20.12
dipole derivatives	DDip(I), I=1,9*NAtoms	3D20.12
force constants	FFX(I), I=1,(3*NAtoms*(3*NAtoms+1))/2	3D20.12

The second section is present only if first derivatives or frequencies were requested, and the final section is present only if frequencies were requested. In the latter case, the Hessian is given in lower triangular form: αij, i=1 to N, j=1 to i. The dipole moment, polarizability, and dipole derivatives can be zero if none are available.

4.2.5 connectors.py

This module hosts high-level functions that *connect* different engines together, handling input/output files delivery and unit conversion.

A `CONNECTORS` dict is maintained at the end of the file listing the connectors available. It's a dict of dicts, where the primary keys are QM engines and secondary keys, MM engines.

`garleek.connectors.`**`gaussian_tinker`**(*qmargs*, *forcefield='mm3.prm'*, *write_file=True*, *qm_version='16'*, *mm_version=None*, ***kwargs*)

Connects QM engine `gaussian` with MM engine `tinker`.

When Gaussian does an ONIOM calculation with Garleek, the MM part is configured with the `external` keyword, meaning that Gaussian will write a series of files to disk and call the requested program. Gaussian expects some data written back to an `*.EOu` file, which should contain potential energy, dipole moment, polarizability and/or hessian matrix, depending on the calculation. The called program is expected to take those input files, convert them to the format expected by the MM program, obtaint the needed data and write them to the EOu file with the adequate syntax. So, that's what we are doing here:

1. Parse Gaussian EIn file
2. Convert it to TINKER's XYZ and KEY files
3. Run TINKER to obtain energy, dipole, etc
4. Convert units and write the EOu file

Parameters

- **`qmargs`** (*`tuple`*) – CLI arguments passed by Gaussian. Depending on the version, its length can vary, but we only care about qmargs[1], so it's not usually a problem
- **`forcefield`** (*`str, optional=mm3.prm`*) – Path to file listing the TINKER forcefield to use. It can be a `*.prm` file or a `*.key` file. PRM files are full forcefields with no modifications. KEY files can import PRM files with `parameters` and then list custom parameters below.
- **`write_file`** (*`bool, optional=True`*) – Wether to write the resulting EOu file to disk.
- **`qm_version`** (*`string, optional=16`*) – Gaussian version in use. Needed to cover the slight differences between Gaussian versions (EIn/EOu syntax, number of args, and so on).
- **`mm_version`** (*`string, optional=None`*) – TINKER behavior. If QM-charges must be considered for the MM part, set it to 'qmcharges'

Returns **eou_data** – Contents of the EOu file Gaussian expects back.

Return type str

4.2.6 atom_types.py

Utilities to deal with `atom_types` mappings.

These files are needed to convert atom types found in the QM engine to those expected by the MM engine. This is a key part of the whole QM/MM calculation, so those types must be chosen wisely. While we provide a few default mappings (check `BUILTIN_TYPES` list), the user is encouraged to define his or her own conversions if needed.

An `atom_types` file format is very simple: just two columns of plain text, where the first field is the QM type and the MM type is in the second field. `garleek-prepare` will just replace the QM types with the corresponding MM type. If a QM type is not found in the file, it will throw an error.

Comments can be inserted with a preceding `#` character, in its own line or after any valid content (just like Python). Blank lines are ignored as well.

This is valid syntax:

```
# atomic number, mm3 type, description

1           5             # H_norm
2           51            # He
3           163           # Li
4           165           # Be
5           26            # B_sp2
6           1             # C_sp3
7           8             # N_sp3
8           6             # O_sp3
```

`garleek.atom_types.`**`get_file`**(*filename*)
Get file from one of the default locations

`garleek.atom_types.`**`parse`**(*atom_types_filename*)
Parse `atom_types` file

4.2.7 units.py

Conversion factors are listed here

The convention is to import this module when needed, using `u` as an alias:

```
>>> from garleek import units as u
```

4.3 Application layer

4.3.1 cli.py

This module contains the command-line interfaces for both `garleek-prepare` (the user-friendly patcher) and `garleek-backend` (the program the QM engine calls behind the scenes to handle the MM calculations).

`garleek-preare` takes a naive QM input file for ONIOM and patches it so the MM part is performed through `garleek-backend`, which will be interfacing with the configured MM engine. For this to work, the atom types featured in the QM input file should be understandable by the MM engine, so `garleek-prepare` will replace those too, using the `atom_types` file mapping to do so.

In general, the worfklow is the following:

1. Build a standard ONIOM calculation, with layers, link atoms and so on. The `garleek-prepare` keyword should be present in the MM layer configuration so the patcher can find it and properly configure it.
2. Patch the QM input file with `garleek-prepare`:

```
garleek-prepare --qm <QM_engine> -mm <MM_engine> --ff <MM_forcefield> \
                --types <QM/MM_atom_type_dictionary> QM_input_file.in
```

3. Submit the calculation with the resulting patched file, named `QM_input_file.garleek.in` with the desired QM software:

```
QM_engine QM_input_file.garleek.in
```

4. Profit!

`garleek.cli.`**`backend_app`**(*qmargs*, *qm='gaussian'*, *mm='tinker'*, *ff='mm3.prm'*, ***kw*)
`garleek-backend` Python entry-point

Parameters

- **`qmargs`** (*`tuple`*) – CLI arguments passed by the QM engine. This can be anything!
- **`qm`** (*`str`*) – QM engine to use. Must be one of `QM_ENGINES`, optionally followed by `_version` to indicate slight differences in the QM logic. For example, `gaussian` defaults to `gaussian_16`, but `gaussian_09a` exports the connectivity differently.
- **`mm`** (*`str`*) – MM engine to use. Must be one of `MM_ENGINES`,optionally followed by `_version` to indicate slight differences in the MM logic.
- **`ff`** (*`str`*) – Forcefield to use in the MM part. This can be anything that the MM engine is able to use as a forcefield (normally a path to a file).

Returns Whatever the QM-MM connector returns

Return type result

`garleek.cli.`**`backend_app_main`**(*argv=None*)
`garleek-backend` CLI entry-point

`garleek.cli.`**`frontend_app`**(*input_file*, *types='uff_to_mm3'*, *qm='gaussian'*, *mm='tinker'*, *ff='mm3.prm'*, ***kw*)
`garleek-prepare` Python entry-point

Parameters

- **`input_file`** (*`str`*) – Path to the QM input file that should be patched so Garleek can handle the MM part through the desired MM engine.
- **`types`** (*`str, default=uff_to_mm3`*) – Path to a file listing the mapping between the QM atom types present in `input_file` and the MM atom types expected by the MM engine given the current forcefield. Atom types are case INSENSITIVE. They will be uppercased upon processing.
- **`qm`** (*`str`*) – QM engine to use. Must be one of `QM_ENGINES`, optionally followed by `_version` to indicate slight differences in the QM logic. For example, `gaussian` defaults to `gaussian_16`, but `gaussian_09a` exports the connectivity differently. This is only needed so the patched `garleek-backend` calls include this argument.

- **mm** (*str*) – MM engine to use. Must be one of `MM_ENGINES`,optionally followed by `_version` to indicate slight differences in the MM logic. This is only needed so the patched `garleek-backend` calls include this argument.
- **ff** (*str*) – Path to the forcefield the MM engine will be using to compute values requested by the QM engine. It should conform to the specified `types` mapping. This is only needed so the patched `garleek-backend` calls include this argument.

Returns **outname** – Path to patched input file. It will always be a derivative of `input_file`. If `input_file` is `input.in`, `outname` will be `input.garleek.in`.

Return type str

`garleek.cli.`**`frontend_app_main`**`(`*argv=None*`)`
`garleek-prepare` CLI entry-point

CHAPTER 5

Citation & Funding

Garleek is scientific software, funded by public research grants: Spanish MINECO (project `CTQ2017-87889-P`, `CTQ2017-87792-R`, Severo Ochoa Excellence Accreditation 2014–2018 `SEV-2013-0319`), Generalitat de Catalunya (project `2014SGR989`, CERCA Programme); J.R.-G.P.: Generalitat de Catalunya (grant `2017FI_B2_00168`); I.F.-A.: Spanish MINECO (Severo Ochoa predoctoral training fellowship `SVP-2014-068662`). If you make use of Garleek in scientific publications, please link back to this repository (a publication is under preparation). It will help measure the impact of our research and future funding!

CHAPTER 6

Help & Support

If you have any question, doubt or suggestion, please submit an issue in our GitHub repository.

6.1 Frequently Asked Questions

6.1.1 Where can I find Gaussian's documentation for `external` keyword?

Follow these links: Gaussian v16, v09D, v09B, v03

6.1.2 Why don't you support more MM software?

Please, follow the ongoing discussion in this issue. PRs are welcome!

6.1.3 Does Garleek support electronic embedding?

Yes, as of v0.3, Garleek supports electronic embedding if you use the `EmbedCharge` option in the `ONIOM` keyword of the Gaussian input file. Differences in the `EIn` files are handled automatically.

6.1.4 Common Garleek/Gaussian/Tinker errors

Garleek only provides a light wrapper around Gaussian's `external` keyword, so it still uses Gaussian's ONIOM implementation, which can be a bit picky depending on the requested operations. MM is tricky itself if the parameters are not available. The list below provides some common errors we have found in our ONIOM journey. It's not meant to be exhaustive, but a collection of the results of long debugging sessions.

Gaussian complains about basis sets not being available

Even though the MM calculation needs no basis sets, Gaussian still runs `l301` on the MM part, which is responsible for preparation tasks like basis sets reading. If your system contains *exotic* elements (say, Rhenium or Osmium) and you are already using `genecp`, it won't find compatible basis sets. The solution is to provide the basis sets manually by appending the `gen` option to the external call and appending two additional basis sets sections (but no pseudopotentials!) around the original basis set section corresponding to the QM `genecp` basis sets. Since this task is repetitive and prone to error, Garleek will do it for you. If `gen` or `genecp` options are present in the `ONIOM()` call, the `external` option will be patched with the `gen` keyword and the basis sets section will be amended as needed. Similarly, if the basis set is globally specified directly in the `ONIOM()` call (no `genecp` nor `gen` keywords, but a specific basis set like `6-31G*`), it will be added to the `external` option just in case. When this patch is applied, you will get an informative message in the output. See `tests/data/Os/Os.in` for an example input file what will benefit from this correction.

Gaussian says "MOMM, but no IMMCRS"

`opt=nomicro` is almost compulsoryin optimization jobs. Garleek will try to add it automatically if it's not present, but it might fail in some cases. Please check the option is there in the route section.

Garleek-prepare says "KeyError: 75 not found in line XXXXX"

The atom type `XX` is not listed in the provided `--types` file. Due to Garleek fallback strategies, `XX` can be the atom type label (ie, `C-XX 0.000 1.000 2.000` in the cartesian geometry specification) or the atomic number corresponding to that element.

This `--types` file must list atom type correspondences between the originally present atom type labels in the Gaussian input and the atom types Tinker is expecting. This way, `garleek-prepare` can replace them in the file and provide the Tinker ones in the patched `garleek.in` file. If you are using custom residues, you must make sure the Tinker atom type and its parameters is correctly defined in the PRM or KEY file (specified as the `--ff` flag).

Tinker says "Unusual number of connected atoms"

The atom type specified for that atom center does not expect covalently bonded species and is missing the bond parameters. You can either (a) provide those parameters or (b) delete the involved bonds in the connectivity section (by hand or in GaussView).

The following tutorials assume you have already installed Garleek, Gaussian and Tinker. If that's not the case, please refer to *Installation*.

CHAPTER 7

A simple CH_4 molecule

Organic species are easy to model, but with ONIOM there's always the added difficulty of setting up layers, link atoms and so on. This simple example will help describe the general workflow for setting up a two-layer QM:MM ONIOM job in GaussView.

Tip: The `tests/data/` directory in the Garleek source contains several toy examples that are easy and fast to compute. Use them for your own tests and training!

7.1 01 - Build the input file

In GaussView or any similar software build a CH_4 molecule and create and B3LYP/UFF ONIOM input file named `tutorial1.in`. You should obtain something like this:

```
%nprocshared=8
%mem=8000MB
%chk=tutorial1
#p opt oniom(B3LYP/6-31G*:UFF) geom=connectivity

ONIOM(QM:MM) OPT FREQ with Garleek

0 1 0 1 0 1
 C-C_3           0    0.44534414   -2.95546554    0.00000000 H
 H-H_            0    0.80199857   -3.96427554    0.00000000 H
 H-H_            0    0.80201698   -2.45106735    0.87365150 L H-H_ 1
 H-H_            0    0.80201698   -2.45106735   -0.87365150 H
 H-H_            0   -0.62465586   -2.95545236    0.00000000 H

1 2 1.0 3 1.0 4 1.0 5 1.0
2
3
```

(continues on next page)

(continued from previous page)

```
4
5
```

Note: In ONIOM calculations, the geometry is specified slightly differently. First, there are more than one `charge multiplicity`. These correspond to the QM, MM real and MM model layers. Additionally, each atom line has more information:

- The first field is usually the element symbol or its atomic number. In ONIOM, the element is extended with a second subfield for the atom type. In the first atom of the example above (`C-C_3`), `C` is the symbol for carbon, and `C_3` its UFF atom type (sp3 carbon). If the force field specified is not UFF, the atom type subfield can (and most probably will) be different.
- A new column before the cartesian coordinates is present. Its value (`0` or `-1`) determines if the atom is frozen or not.
- After the cartesian coordinates, at least one additional column is present, listing the ONIOM layer: `H` (high), `M` (middle), or `L` (low).
- The layer column can be followed by the link atom specifiers, when there's a covalent bond between atoms sitting in different layers. It must contain the link atom specifier (element + atom type) and the serial index of the bonded atom. In the only example above, the link atom is a simple hydrogen atom that will replace the originally bonded carbon atom.

The connectivity matrix is always needed because the MM program usually requires it. It lists every atom by serial number, and, if it is bonded to any atom(s), the serial number of those atoms, each followed by its corresponding bond order. In the example above, the atom 1 (carbon) is bonded to they hydrogens (atoms 2, 3, 4 and 5) with single bonds (bond order = 1.0).

This file will run just fine in Gaussian, but it will use the simple UFF force field. If you want to use a more accurate one (other than Amber and Dreiding), Garleek is needed. Let's try to use the MMFF force field for this molecule.

7.2 02 - Patch the input file with Garleek

The file as created by GaussView needs some modifications to work with Garleek. Don't worry, `garleek-prepare` will do everything for you! For this system to work with MM3, you would use the following command:

```
garleek-prepare --qm gaussian_09d --mm tinker --ff MMFF --types uff_to_mm3 tutorial1.
↪in
```

- `--qm` lists the QM engine in use. By default it's `gaussian_16`, so you don't have to specify it if that's the version you are using. In this case, we are using Gaussian 09D, so that's why we included it.
- `--mm` lists the MM engine in use. Only Tinker is supported now, so you can omit it in this example.
- `--ff` specifies the force field to be used by Tinker. The file will be found following 3 strategies:
  1. If the value is the path to a file, use it. This way you can use `mycustomforce field.prm` or `mycustomparameters.key`. Both Tinker PRM and KEY files are supported.
  2. If no file is found with that path, try to find it under `garleek/data/prm`. This allows you to use `mm3.prm` even if it's not present in the working directory.
  3. Try again by adding the `.prm` extension. This allows you to use simply `mm3`.

- `--types` must be a file listing the correspondences between the atom types originally specified in the Gaussian input file, and those expected by Tinker. Since GaussView (or your favorite software) generated the types for UFF and we want to use MM3, the dictionary specified is `uff_to_mmff`, which is included in Garleek and available in `garleek/data/atom_types`. This option admits two types of values:
  1. A valid path to a file
  2. A filename under `garleek/data/atom_types`.

We will go into further details later. For now, we just need to know that this command would generate a file called `tutorial1.garleek.in` with these contents:

```
! Created with Garleek v0+untagged.68.g6711cac.dirty
%nprocshared=8
%mem=8000MB
%chk=A_1
#p opt=nomicro oniom(B3LYP/6-31G*:external="garleek-backend --qm gaussian_09d --mm␣
↪tinker --ff 'mmff'"/6-31G*) geom=connectivity

ONIOM(QM:MM) OPT FREQ with Garleek

0 1 0 1 0 1
 C-1              0    0.44534414   -2.95546554    0.00000000 H
 H-23             0    0.80199857   -3.96427554    0.00000000 H
 H-23             0    0.80201698   -2.45106735    0.87365150 L H-23 1
 H-23             0    0.80201698   -2.45106735   -0.87365150 H
 H-23             0   -0.62465586   -2.95545236    0.00000000 H

1 2 1.0 3 1.0 4 1.0 5 1.0
2
3
4
5
```

Let's see what has changed in this file.

1. A new line beginning with an exclamation mark `!` has been added. This is just a comment (ignored by Gaussian) listing the garleek version used so you can reproduce the calculations later on with the exact same version.
2. The route `#` section has grown significantly:
   - `opt=nomicro` has been added. This disables microoptimizations, which can lead to known errors when applying the `external` keywords.
   - `external` has a long string attached. This is the `garleek-backend` command that will be called in every Gaussian ONIOM iteration. It has been added automatically by `garleek-prepare` so you don't need to worry about its details.
   - The basis set configured in the QM layer has been included in the MM layer as well. This is a workaround some errors with the default basis sets in Gaussian. Only applies for *exotic* elements, but since it doesn't hurt to have it specified here, it's always included for convenience.
3. The atom types (`H_`, `C_`) has been replaced by numbers (`23`, `1`). This is a direct replacement as specified in the `--types` file and it's the key step in the whole process.

7.3 03 - Review the atom types

Since this simple molecule only includes one carbon atom with its four hydrogen atoms, the conversion is trivial. UFF only includes one (or very few) atom type(s) per element, but that's very uncommon in most force fields: they normally list several atom types per element depending on its bonded atoms and other conditions.

As a result, the conversion between UFF and other force fields is not unequivocal. An effort has been made to provide the best correspondence for most cases, but you should check the types manually! You can define your own atom types mapping by modifying the ones provided with Garleek (creating a separate copy is recommended) or writing a new one from scratch. The syntax is very simple: one correspondence per line, listing the original atom type in the first field, and the Tinker atom type in the second field, separated by one or more spaces. Comments can be inserted with # in its own line or ending a valid line.

For example, the `uff_to_mm3` file lists some correspondences between atomic numbers and default MM3 Tinker types:

```
# atomic number, mm3 type, description

1           5             # H_norm
2           51            # He
3           163           # Li
4           165           # Be
5           26            # B_sp2
6           1             # C_sp3
7           8             # N_sp3
8           6             # O_sp3
9           11            # F
10          52            # Ne
```

7.4 04 - Launch the Gaussian job

The resulting `.garleek.in` file is a valid Gaussian input file. You can run it with any standard procedures you are already using, like `g09 tutorial2.garleek.in` locally, or in a queued cluster system. Gaussian & Garleek will take care of the rest!

CHAPTER 8

Organometallic species

QM/MM studies are particularly useful in metal-containing systems. However, some metal elements are rarely present in MM force fields and custom parameters must be provided (especially if coordination bonds are considered in the MM part). Fortunately, most of the time you can provide an isolated metal ion (no explicit bonds for the MM calculation) and get away with providing the van der Waals radius.

Let's take the following Osmium compound as an example. Go to tests/data/Os and grab a copy of the `Os.in` and `Os.key` files. This file can be fed to `garleek-prepare` to provided a Garleek-ready `Os.garleek.in` file:

```
garleek-prepare --types uff_to_mm3 --ff Os.key Os.in
```

Several considerations must be done here:

- `--types` has been set to `uff_to_mm3`. This file is provided with Garleek, and contains a manual mapping listing UFF to MM3 correspondences. In most cases, it should work for your needs, but you are encouraged to review the choices made in that file so they fit your system.
- `--ff` has been set to `Os.key`. The `ff` flag can be set to either PRM or KEY files.

KEY files are important in Tinker and can help you perform a lot of calculations. We use them to load default parameters from PRM files and include additional parameters on case-by-case basis. In this example, the force field has been set to `qmmm3.prm`. This PRM file ships with Garleek. It's an extension of the original Tinker MM3 parameters to contain atom type definitions for most elements in the periodic table (transition metals included). However, it does not contain bond, angle or dihedral parametrization. Only the element masses and VdW radii are included, so you can only use ISOLATED metal ions. If you want to use bonded MM metals, you will need to provide those parameters. The KEY file includes this data below the `parameters` line:

```
parameters qmmm3.prm

# Define bond parameters
bond       7         165       0.3            1.67
bond       8         5         6.420          1.0150
# Define bond angle parameters
angle      7         165       7              0.5       90.0
angle      6         2         37             0.6       120.0
angle      5         8         5              0.605     106.40
```

(continues on next page)

(continued from previous page)

```
# Define torsion parameters
torsion    2    1     6     2          0.0   0.0 1     0.0 180.0 2      0.403 0.0 3
torsion    6    2     37    37         0.0   0.0 1    12.0 180.0 2      0.0   0.0 3
torsion    1    6     2     37         1.05  0.0 1     7.5 180.0 2     -0.2   0.0 3
```

Should you need more atom types, you can define those in your KEY file and provide that as the `--ff` value instead of a generic PRM file. For example, if you want to use the Amber99 force field with an aluminium atom, you should include two changes:

- The `-ff` should be a KEY file with the amber force field loaded with the `parameters` keyword and a new atom definition for the aluminium ion with an atom type id of your choice. Let's say `5000`. Van der Waals data should be added as well for that atom type id.
- The `--types` file should list a line with `13 5000`, where `13` is the Al atomic number and `5000` is the Tinker atom type. You an use any atom type label in the original Gaussian file (ie, `Al-ALX`), but since Garleek will try to use the atomic number if the atom type label (`ALX`) cannot be found in the `--types` file, using the atomic number (`13`) works just fine as a generic fallback.

8.1 Custom basis sets

When dealing with metals, custom basis sets might be needed. This usually accomplished with the `gen` or `genecp` options in the route section and specifying the basis sets and pseudopotentials after the connectivity matrix. When using `external` as the MM part in the ONIOM calls, the basis must be provided for the MM part as well. This means that the ONIOM call should be like this:

```
ONIOM(B3LYP/genecp:external="garleek-backend ..."/gen)
```

Instead of specifying the basis sets and pseudopotentials once, they must be specified separately for each QM and MM calculation. This means that, instead of having these lines (QM part only):

```
Os 0
LanL2DZ
****
O 0
6-31G*
****
C H N 0
6-31G
****

Os 0
LanL2DZ
```

You will need these (MM basis sets, QM basis sets & pseudopotentials, MM basis sets again):

```
! Lines starting with exclamation marks are comments and can be ignored
! MM basis sets
Os 0
LanL2DZ
****
O 0
6-31G*
****
C H N 0
```

(continues on next page)

(continued from previous page)

```
6-31G
****

! QM basis sets
Os 0
LanL2DZ
****
O 0
6-31G*
****
C H N 0
6-31G
****

! QM pseudopotentials
Os 0
LanL2DZ

! MM basis sets
Os 0
LanL2DZ
****
O 0
6-31G*
****
C H N 0
6-31G
****
```

`garleek-prepare` will try to detect the basis sets and pseudopotentials if the `ONIOM` keyword contains the `genecp` or `gen` options, and fill the MM basis sets automatically, so you don't worry about these technicalities. However, if somehow it fails, you will need to review those lines so Gaussian can find the proper basis sets.

CHAPTER 9

Specific details for biomolecules

When biomolecules are involved in a QM/MM calculation, protein-specific force fields are needed. Fortunately, Tinker provides several force fields that fall in this category:

- AMBER 94, 96, 98, 99, 99SB
- AMOEBABIO & AMOEBAPRO
- CHARMM 19, 22
- MM3PRO
- OPLS-AA

Protein-specific force fields usually parameterize atoms and groups them by residue. In Tinker, each atom in each residue would be a different atom type (but similar ones are grouped in atom classes). This can lead to some confusion, because Tinker will be expecting atom types, not atom classes, in its XYZ input file (this is generated automatically by Garleek). The `--types` dictionary will have to unequivocally map residue-atom pairs to each unique atom type. To overcome this limitation, we follow an alternative typing approach for biostructures.

Tip: To prepare a protein structure, using separate software like UCSF Chimera with our Tangram suite is recommended. This will take care of some annoying details that have to do with atom typing, like adding hydrogen atoms and terminal caps, fixing residue and atom names, and will also generate the properly formatted Gaussian input file Garleek expects.

When the protein structure is properly formatted, you should obtain a PDB file that can be loaded into GaussView. Instead of having atom lines like these:

```
C-C_3              0    0.44534414   -2.95546554    0.00000000 H
H-H_               0    0.80199857   -3.96427554    0.00000000 H
H-H_               0    0.80201698   -2.45106735    0.87365150 L H-H_ 1
H-H_               0    0.80201698   -2.45106735   -0.87365150 H
H-H_               0   -0.62465586   -2.95545236    0.00000000 H
```

You will see lines like these:

```
N-N3-0.000000(PDBName=N,ResName=NGL,ResNum=1)      -1   -0.47100000   20.52700000  -
↪13.50600000 L
H-H-0.000000(PDBName=H1,ResName=NGL,ResNum=1)      -1   -0.31300000   21.51500000  -
↪13.64700000 L
H-H-0.000000(PDBName=H2,ResName=NGL,ResNum=1)      -1    0.26700000   20.00100000  -
↪13.95200000 L
H-H-0.000000(PDBName=H3,ResName=NGL,ResNum=1)      -1   -1.36000000   20.26700000  -
↪13.90800000 L
C-CX-0.000000(PDBName=CA,ResName=NGL,ResNum=1)     -1   -0.48000000   20.22400000  -
↪12.02500000 L
H-HP-0.000000(PDBName=HA2,ResName=NGL,ResNum=1)    -1   -1.50100000   20.04900000  -
↪11.68700000 L
```

Notice the first *field* it's still an atom identifier whose subfields are separated by - characters:

- 1st subfield: Element symbol. Sometimes, atomic number.
- 2nd subfield: Atom type.
- 3rd subfield: Charge, `PDB` fields.

`PDB` fields are **important** in Garleek because when this type of line is present, the atom type (2nd field) is IGNORED and a NEW one is computed on the fly, following this template: `<ResName>_<PDBName>`. For example, the first line in the block above would generate an atom type named `NGL_N`. The original `N3` will be IGNORED.

As a result, for the `--types` dictionaries to work with biomolecules, they must include the adequate `<ResName>_<PDBName>` combination, and not the 2nd field as seen in the previous tutorials. Obviously, the originating PDB file must have atoms and residues properly named so the PDB fields are correctly written. Otherwise, it won't work.

We provide several mappings obtained automatically from Tinker force fields featuring a `biotype` section using a custom script. However, for this to work, the biomolecule must include the correct `PDBName` and `ResName` values.

Tip: A script named `biotyper.py` can be found under `garleek/data/prm`. This script can parse PRM files for `biotype` lines and generate a `.types` file automatically, which would work as a good starting point towards configuring your own atom types mapping.

Link atoms

Link atoms are also affected by this special treatment. If PDB fields are present, the link atom type will be composed out of the main atom `ResName` and the atom type listed next to the link atom element. For example, in the line:

```
H-HP-0.000000(PDBName=HA2,ResName=NGL,ResNum=1) -1 -1.50100000 20.04900000 -11.
↪68700000 L H-HB 5
```

, the calculated link atom type would be `NGL_HB`.

You should choose link atoms with type according to its bonded atom to avoid parameter problems (angles and dihedrals, particularly). For example, if the main atom is `CB` the correct H link atom should be `HB`. Refer to the PRM force field to locate the proper type (PDBName).

9.1 Custom residues

When custom residues are present in the structure, even in the QM region, they must be included for the MM calculation anyways. Using them is no harder than normal residues, but parameters must be present either in the PRM file or in a custom KEY file. Then, the normal atom type conversion rules will be followed to locate the proper Tinker atom type from the PDB fields.

Toy example for a NH3 residue in the Amber format:

The PDB file would be something like this:

```
HETATM    1  N1  NH3     1       0.000   0.000   0.000  1.00  0.00           N
HETATM    2  H1  NH3     1       1.010   0.000   0.000  1.00  0.00           H
HETATM    3  H2  NH3     1      -0.337   0.952   0.000  1.00  0.00           H
HETATM    4  H3  NH3     1      -0.336  -0.476  -0.825  1.00  0.00           H
```

The Gaussian input file would end up like:

```
N-N3-0.000000(PDBName=N1,ResName=NH3,ResNum=1)      -1   0.000   0.000   0.000 L
H-HN-0.000000(PDBName=H1,ResName=NH3,ResNum=1)      -1   1.010   0.000   0.000 L
H-HN-0.000000(PDBName=H2,ResName=NH3,ResNum=1)      -1  -0.337   0.952   0.000 L
H-HN-0.000000(PDBName=H3,ResName=NH3,ResNum=1)      -1  -0.336  -0.476  -0.825 L
```

The PRM file should contain:

```
atom    5000   14    N     "Custom Residue NH3 N1"      7    14.010    3
atom    5001   29    H     "Custom Residue NH3 H1"      1     1.008    1
atom    5002   29    H     "Custom Residue NH3 H2"      1     1.008    1
atom    5003   29    H     "Custom Residue NH3 H3"      1     1.008    1

# bonds, dihedrals, vdw and so on should be needed as well
# You would probably use something like Antechamber for these data
```

The `--types` dictionary should list:

```
NH3_N1 5000
NH3_H1 5001
NH3_H2 5002
NH3_H3 5003
```

9.2 Electronic embedding

For biological systems, electronic embedding is usually recommended. Garleek supports this mode (*ONIOM=EmbedCharge*) out of the box, so no additional configuration is needed. That said, read Gaussian documentation on ONIOM for notes on possible convergence problems.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index