gqteaMD User Manual

gqteaMD is a Python molecular dynamics program that uses velocity Verlet integration and modular force providers. It can read atomic coordinates from XYZ files, propagate atoms in an orthorhombic simulation box, and obtain forces from a simple harmonic test potential, a first classical force-field provider, a first UFF provider, or Gaussian single-point force calculations.

1. Requirements

gqteaMD works on Windows and Linux.

Required Python version:

`Python 3.11 or newer`

Python dependencies:

`numpy`
`cclib`

Development/test dependency:

`pytest`

For Gaussian calculations, Gaussian must be installed separately and available as a command such as `g16`, `g16.exe`, or another local executable name.

2. Installation

Open a terminal in the project folder:

```
cd C:\gqteaMD\venv\src
```

Install gqteaMD in editable mode:

```
python -m pip install -e .[dev]
```

On Linux or macOS shells, quote the optional dependency expression:

```
python -m pip install -e ".[dev]"
```

Updating `gqteaMD.exe` After Code Changes

On Windows, `gqteaMD.exe` is a small console-script launcher that calls the installed Python package entry point:

`gqteaMD.cli:main`

After changing files in the `gqteaMD` source code, reinstall the package from the project folder:

```
cd C:\gqteaMD\venv\src
C:\gqteaMD\venv\Scripts\python.exe -m pip install -e .
```

Editable mode means normal source-code changes under:

`C:\gqteaMD\venv\src\gqteaMD`

are picked up by `gqteaMD.exe` the next time you run it.

Run the test suite before using the changed program:

```
cd C:\gqteaMD\venv\src
C:\gqteaMD\venv\Scripts\python.exe -m pytest
```

If you changed `pyproject.toml`, package metadata, dependencies, or the console script entry point, run the editable install command again:

```
C:\gqteaMD\venv\Scripts\python.exe -m pip install -e .
```

Oracle Linux 9.5 Installation

On Oracle Linux 9.5, install gqteaMD into a fresh Linux Python environment. Do not copy a Windows `venv` folder to Linux, because virtual environments are specific to one operating system and Python installation.

Install Python 3.11 and basic tools:

```
sudo dnf install -y python3.11 python3.11-pip git
```

If you plan to build from source, run tests, or install packages that need local compilation, also install the compiler and Python headers:

```
sudo dnf install -y gcc gcc-c++ python3.11-devel
```

Create and activate a virtual environment:

```
python3.11 -m venv ~/gqteaMD-env
source ~/gqteaMD-env/bin/activate
python -m pip install --upgrade pip setuptools wheel
```

If you have the gqteaMD source folder, install it in editable mode from the project folder:

```
cd /path/to/gqteaMD
python -m pip install -e ".[dev]"
```

For a normal runtime installation without development and test tools, use:

```
python -m pip install -e .
```

If you have a built wheel file from the `dist` folder, install that wheel instead:

```
python -m pip install dist/gqteaMD-0.2.0-py3-none-any.whl
```

Check that the command is available:

```
gqteaMD --help
gqteaMD run examples/harmonic.toml
```

For Gaussian calculations on Linux, first make sure Gaussian works from the same terminal:

```
g16
```

Then use the Linux Gaussian command in the TOML file:

```
command = "g16"
```

Or run directly from an XYZ file, changing the Gaussian path if needed:

```
gqteaMD run water.xyz --time-fs 10 --dt-fs 0.5 --gaussian-home
/path/to/gaussian
```

Check that the command is available:

```
gqteaMD --help
```

3. Packaging gqteaMD For Students

If you want to distribute gqteaMD to students, build a Python wheel package. Do not distribute the `venv` folder, because virtual environments are specific to one computer and usually do not work correctly on other machines.

Build The Package

On the instructor computer, open PowerShell in the project folder:

```
cd C:\gqteaMD\venv\src
```

Install the Python build tool:

```
python -m pip install build
```

Build the package:

```
python -m build
```

This creates a `dist` folder containing files similar to:

```
dist\gqteamd-0.1.0-py3-none-any.whl
dist\gqteamd-0.1.0.tar.gz
```

These two files are Python package distribution files:

- `gqteamd-0.1.0-py3-none-any.whl` is the wheel package. This is the easiest file for students to install, because it is already built.
- `gqteamd-0.1.0.tar.gz` is the source distribution. It contains the package source code and packaging metadata. It is mostly a backup or source package for package indexes, and students usually do not need it.

For normal student installation, give students the `.whl` file and have them install that file with `pip`. You do not need to give students both files unless you specifically want them to have the source distribution too.

If the student computer does not have internet access, also provide wheel files for any required Python dependencies that are not already installed.

What To Give Students

For a class distribution, make a zip file containing:

```
gqteamd-0.1.0-py3-none-any.whl
water.xyz
water.toml
examples\methane.xyz
examples\uff_water.toml
examples\uff_methane.toml
README.md
USER_MANUAL.md
```

You can also include additional example XYZ and TOML files for different molecules or methods.

Gaussian is not included in the gqteaMD package. Each student must have Gaussian installed separately, and each TOML file must point to the correct Gaussian executable on that student's computer.

For example, if Gaussian 09 for Windows is installed in `C:\G09W`, use:

```
command = "C:/G09W/g09.exe"
```

Student Installation

Students should create their own Python environment:

```
python -m venv gqteaMD-env
.\gqteaMD-env\Scripts\Activate.ps1
```

Then install the wheel file:

```
python -m pip install .\gqteamd-0.1.0-py3-none-any.whl
```

Check that the command is available:

```
gqteaMD --help
```

Then run an example:

```
gqteaMD run water.toml
```

If PowerShell does not recognize `gqteaMD`, run it using the full path inside the student's environment:

```
& .\gqteaMD-env\Scripts\gqteaMD.exe run water.toml
```

In PowerShell, the `&` symbol is the call operator. It tells PowerShell to run the following path as a command. This is useful when running an executable by its full path, especially if the path contains spaces.

Student Quick Run Without TOML

Students can also run directly from an XYZ file:

```
gqteaMD run water.xyz --time-fs 10 --dt-fs 0.5 --gaussian-home
C:\G09W
```

This command runs a 10 fs simulation using a 0.5 fs timestep. The `--gaussian-home` folder must be changed if Gaussian is installed somewhere else.

4. Running The Example

The project includes a small harmonic-force example:

```
gqteaMD run examples/harmonic.toml
```

This creates:

```
examples/harmonic_md.log
examples/harmonic_traj.xyz
```

The log contains energies and temperature. The trajectory is an XYZ file with one frame per saved MD step.

5. Easy Gaussian Run From A Folder With An XYZ File

You can now run directly from an XYZ file without writing a TOML file first.

Suppose you are working in:

```
C:\AJCamargo\testeMD
```

and this folder contains:

```
water.xyz
```

If Gaussian 09 is installed in:

```
C:\G09
```

run:

```
cd C:\AJCamargo\testeMD
gqteaMD run water.xyz --time-fs 10 --dt-fs 0.5 --gaussian-home
C:\G09
```

This means:

- `water.xyz`: starting geometry.
- `--time-fs 10`: total MD simulation time is 10 femtoseconds.
- `--dt-fs 0.5`: each MD step is 0.5 femtoseconds.
- `--gaussian-home C:\G09`: tells gqteaMD where to find Gaussian 09.

Because:

```
10 fs / 0.5 fs = 20 steps
```

gqteaMD runs 20 integration steps. If you prefer to give the maximum number of steps directly, use `--steps`:

```
gqteaMD run water.xyz --steps 20 --dt-fs 0.5 --gaussian-home
C:\G09
```

Default output files are written in the same folder as `water.xyz`:

```
TRAJEC.xyz
water_gqteaMD.log
GEOMETRY
gaussian_steps\
```

The default Gaussian route is:

```
# B3LYP/6-31G(d) SCF=Tight
```

gqteaMD automatically adds `Force` and `NoSymm` if they are missing. To choose a different Gaussian route, pass `--route`:

```
gqteaMD run water.xyz --time-fs 10 --dt-fs 0.5 --gaussian-home
C:\G09 --route "# HF/3-21G SCF=Tight"
```

By default, direct XYZ runs use a cubic 20 angstrom cell. To change it:

```
gqteaMD run water.xyz --time-fs 10 --dt-fs 0.5 --gaussian-home
C:\G09 --cell 30 30 30
```

For a quick test without Gaussian, use the harmonic provider:

```
gqteaMD run water.xyz --time-fs 10 --dt-fs 0.5 --force-provider
harmonic
```

For a small UFF run without Gaussian or a TOML file, use:

```
gqteaMD run water.xyz --time-fs 10 --dt-fs 0.5 --force-provider
uff
```

6. Input XYZ File

The initial geometry is read from an XYZ file.

Example:

```
3
water
O 0.00000000 0.00000000 0.00000000
H 0.95720000 0.00000000 0.00000000
H -0.23998720 0.92662721 0.00000000
```

Format:

```
number_of_atoms
comment line
ElementSymbol x y z
ElementSymbol x y z
...
```

Coordinates are interpreted as angstrom.

The XYZ file does not define the simulation cell. For direct XYZ runs, use `--box-size` or `--cell`. For TOML runs, the cell is defined in the TOML configuration file.

7. Configuration File

gqteaMD simulations are controlled by a TOML file.

Minimal example:

```
[input]
xyz = "water.xyz"

[cell]
a = 20.0
b = 20.0
c = 20.0

[dynamics]
timestep_fs = 0.25
steps = 10

[force_provider]
type = "harmonic"
k_ev_per_angstrom2 = 0.1

[output]
trajectory = "harmonic_traj.xyz"
log = "harmonic_md.log"
log_interval = 1

[restart]
path = "RESTART"
interval = 5
resume_from_RESTART = false
resume_from_GEOMETRY = false
```

Run it with:

```
gqteaMD run path\to\config.toml
```

Paths inside the configuration file are interpreted relative to the location of the configuration file.

8. Configuration Sections

`[input]`

```
[input]
xyz = "initial.xyz"
```

Required fields:

- `xyz`: path to the starting XYZ geometry.

`[cell]`

```
[cell]
a = 20.0
b = 20.0
c = 20.0
```

Required fields:

- `a`: box length along x in angstrom.
- `b`: box length along y in angstrom.
- `c`: box length along z in angstrom.

The current version assumes:

`alpha = beta = gamma = 90 degrees`

That means the simulation cell is orthorhombic.

`[dynamics]`

```
[dynamics]
timestep_fs = 0.5
steps = 100
```

Required fields:

- `timestep_fs`: MD timestep in femtoseconds.
- `steps`: number of MD integration steps.

The first version uses velocity Verlet integration in the NVE style. Thermostats and barostats are planned extension points, but are not implemented yet.

`[force_provider]`

This section selects how forces are computed.

Four force providers are currently available:

```
classical
harmonic
uff
gaussian
```

[output]

```
[output]
trajectory = "TRAJEC.xyz"
log = "initial_gqteaMD.log"
log_interval = 1
```

Fields:

- `trajectory`: output XYZ trajectory path. Default: `"TRAJEC.xyz"`.
- `log`: whitespace-separated MD log path. By default, gqteaMD uses `<xyzname>_gqteaMD.log`, where `<xyzname>` is the stem of the input XYZ file.
- `log_interval`: write output every N steps.

Trajectory and log files are appended if they already exist.

Each trajectory atom line contains seven columns: symbol, x, y, z, vx, vy, vz. gqteaMD also rewrites `GEOMETRY` at every calculation step. Each `GEOMETRY` atom line contains ten columns: symbol, x, y, z, vx, vy, vz, Fx, Fy, Fz.

[restart]

This optional section controls restart file writing and resume behavior.

```
[restart]
path = "RESTART"
interval = 10
resume_from_RESTART = false
resume_from_GEOMETRY = false
```

Fields:

- `path`: restart file path. The recommended name is `"RESTART"`.
- `interval`: save a copy of the restart file every N MD steps.
- `resume_from_RESTART`: when `true`, start from the restart file instead of from the `[input]` XYZ file.
- `resume_from_GEOMETRY`: when `true`, read positions, velocities, and forces from `GEOMETRY`, and read potential and total energy from `RESTART`.

The restart file stores:

- atom symbols and masses
- cell lengths and periodic flags
- positions
- velocities

- forces
- image flags
- potential energy
- total energy
- step number
- simulation time

When a run is resumed, `steps` in `[dynamics]` means the number of additional steps to run after loading the restart state.

9. Harmonic Force Provider

The harmonic force provider is mainly for testing the program without Gaussian.

Example:

```
[force_provider]
type = "harmonic"
k_ev_per_angstrom2 = 0.1
```

It applies an independent harmonic potential to each atom:

```
E = 1/2 k r^2
F = -k r
```

The force constant `k_ev_per_angstrom2` is in eV/angstrom^2.

10. Classical Force Provider

The classical force provider is a simple classical molecular dynamics implementation in gqteaMD. This version supports:

- harmonic bond stretching
- Lennard-Jones nonbonded interactions
- minimum-image distances in the orthorhombic simulation cell
- optional exclusion of directly bonded atom pairs from Lennard-Jones

It does not include standard force-field topology reading, angle bending, dihedral torsions, Coulomb electrostatics, constraints, or full molecular mechanics force-field support.

Example:

```
[force_provider]
type = "classical"
cutoff_angstrom = 10.0
exclude_bonded = true
bonds = [
{ atoms = [0, 1], k_ev_per_angstrom2 = 45.0, r0_angstrom = 0.9572
},
{ atoms = [0, 2], k_ev_per_angstrom2 = 45.0, r0_angstrom = 0.9572
```

```
},
]

[force_provider.lennard_jones.O]
epsilon_ev = 0.0067
sigma_angstrom = 3.1507

[force_provider.lennard_jones.H]
epsilon_ev = 0.0
sigma_angstrom = 1.0
```

Fields:

- `type`: must be `"classical"`.
- `atom_types`: optional list of atom types. If omitted, gqteaMD uses the XYZ element symbols as atom types.
- `bonds`: harmonic bonds. Atom indices are zero-based, so the first atom is atom `0`.
- `k_ev_per_angstrom2`: bond force constant in eV/angstrom^2.
- `r0_angstrom`: equilibrium bond distance in angstrom.
- `lennard_jones`: Lennard-Jones parameters by atom type.
- `epsilon_ev`: Lennard-Jones well depth in eV.
- `sigma_angstrom`: Lennard-Jones sigma in angstrom.
- `cutoff_angstrom`: optional nonbonded cutoff in angstrom.
- `exclude_bonded`: when `true`, directly bonded atom pairs are skipped in the Lennard-Jones calculation.

Run the included classical example with:

```
gqteaMD run examples/classical_water.toml
```

11. UFF Force Provider

The UFF provider is a first implementation of the Universal Force Field for small teaching systems. It does not need external force-field files. It reads the XYZ coordinates, detects bonds from covalent radii, assigns simple UFF atom types, builds a reusable topology, and computes:

- harmonic bond stretching
- harmonic angle bending
- Lennard-Jones nonbonded interactions
- exclusion of directly bonded pairs and angle-neighbor pairs from Lennard-Jones interactions

The UFF topology now stores additional derived information for future force terms:

- bond orders, either inferred from the starting geometry or supplied in TOML

- torsion lists generated from four-atom bonded paths
- inversion lists generated for three-coordinate central atoms
- optional per-atom charges from the TOML file
- precomputed nonbonded exclusion pairs and 1-4 pairs

In the current force evaluation, UFF uses bond-order and electronegativity corrections for bond equilibrium distances, analytic harmonic angle forces, cosine torsion terms for supported sp2/sp3 central bonds, harmonic out-of-plane inversion terms for supported three-coordinate centers, Lennard-Jones nonbonded interactions, and optional fixed-charge Coulomb electrostatics. Charges are explicit per-atom values in units of elementary charge.

The first UFF milestone supports these elements:

```
H C N O F P S Cl Br I
```

Example:

```
[force_provider]
type = "uff"
bond_detection_scale = 1.2
cutoff_angstrom = 10.0
# Optional explicit atom types, one for each atom in the XYZ
file.
atom_types = ["O_3", "H_", "H_"]
# Optional per-atom charges, stored for future electrostatics
support.
charges = [-0.834, 0.417, 0.417]
# Fixed-charge Coulomb electrostatics. If omitted, electrostatics
turns on
# automatically when charges are provided.
electrostatics = true
# Nonbonded exclusions: "exclude_12_13" (default), "exclude_12",
or "none".
nonbonded_exclusions = "exclude_12_13"
# Scale 1-4 nonbonded pairs generated from torsions.
lj_14_scale = 1.0
electrostatic_14_scale = 1.0
# Lennard-Jones cutoff mode: "plain" or energy-shifted "shift".
lj_cutoff_mode = "plain"
# Use a Verlet neighbor list for cutoff nonbonded interactions.
use_neighbor_list = true
neighbor_skin_angstrom = 2.0
# Optional explicit bond orders. If omitted, gqteaMD infers
simple orders
# from the starting geometry and UFF atom types.
bond_orders = [
{ atoms = [0, 1], order = 1.0 },
```

```
{ atoms = [0, 2], order = 1.0 },
]

# Optional explicit topology. When bonds are provided, they
replace automatic
# bond detection. If angles, torsions, or inversions are omitted,
they are
# generated from the selected bond graph. If any of those lists
are present,
# the provided list is used exactly, including an empty list.
bonds = [
{ atoms = [0, 1], order = 1.0 },
{ atoms = [0, 2], order = 1.0 },
]
angles = [
{ atoms = [1, 0, 2] },
]
torsions = []
inversions = []
```

Fields:

- `type`: must be `"uff"`.
- `bond_detection_scale`: multiplier applied to covalent radii when detecting bonds from the initial XYZ geometry. The default is `1.2`.
- `cutoff_angstrom`: optional nonbonded cutoff in angstrom.
- `atom_types`: optional explicit UFF atom type list. If omitted, gqteaMD assigns simple atom types automatically.
- `charges`: optional per-atom charge list in elementary charge units. The list length must match the number of atoms.
- `electrostatics`: optional boolean. If omitted, electrostatics is enabled when `charges` are provided and disabled when they are not. When enabled, gqteaMD computes pairwise fixed-charge Coulomb energy and forces using the same cutoff and exclusion policy as the Lennard-Jones term.
- `nonbonded_exclusions`: controls excluded nonbonded pairs. `"exclude_12_13"` excludes directly bonded pairs and angle-neighbor pairs, matching the original UFF behavior. `"exclude_12"` excludes only directly bonded pairs. `"none"` applies nonbonded interactions to every pair.
- `lj_14_scale`: scale factor for Lennard-Jones interactions between 1-4 pairs generated from torsions. The default is `1.0`.
- `electrostatic_14_scale`: scale factor for Coulomb interactions between 1-4 pairs generated from torsions. The default is `1.0`.
- `lj_cutoff_mode`: `"plain"` uses a direct LJ cutoff. `"shift"` subtracts the LJ energy at the cutoff so the pair energy is zero at `cutoff_angstrom`.
- `use_neighbor_list`: when `true`, gqteaMD uses a cached Verlet neighbor list for UFF Lennard-Jones and Coulomb pairs whenever `cutoff_angstrom` is set. If no cutoff is set, gqteaMD uses the all-pairs loop.

- `neighbor_skin_angstrom`: extra distance added to the cutoff when building the neighbor list. The list is rebuilt after any atom moves more than half this skin distance from the last neighbor-list build position.
- `bond_orders`: optional list of bond-order overrides. Each entry uses zero-based `atoms = [i, j]` and a positive `order`, for example `1.0`, `1.5`, `2.0`, or `3.0`. If omitted, gqteaMD infers simple bond orders from the starting geometry.
- `bonds`: optional explicit bond list. Each entry uses zero-based `atoms = [i, j]`; it may also include `order`, which is equivalent to adding the same pair to `bond_orders`. When `bonds` is present, gqteaMD does not use radius-based bond detection.
- `angles`: optional explicit angle list with `atoms = [i, j, k]`, where `j` is the central atom. If present, this list replaces generated angles.
- `torsions`: optional explicit torsion list with `atoms = [i, j, k, l]`. If present, this list replaces generated torsions.
- `inversions`: optional explicit inversion list with `atoms = [center, i, j, k]`. If present, this list replaces generated inversions.

The current automatic UFF typing uses the detected bond graph. It is more structured than the original degree-only version, but it remains a simple typing scaffold. For chemically sensitive cases, provide explicit `atom_types`.

Topology control can be partial. For example, you may provide only `bonds` and let gqteaMD generate angles, torsions, and inversions from those bonds. If you provide `angles = []`, the angle list is intentionally empty and no angle terms are generated. Explicit angles, torsions, and inversions must be consistent with the selected bond graph, or gqteaMD raises an input error.

Current torsion and inversion support is intentionally conservative. Torsions are generated from four-atom bonded paths and are active for supported sp3-sp3, sp2-sp2, and selected sp2-sp3 central bonds. Inversions are generated for three-coordinate centers and are active for supported C, N, and P centers. Torsion and inversion forces are evaluated by local central finite differences, while angle forces are analytic.

Run the included UFF examples with:

```
gqteaMD run examples/uff_water.toml
gqteaMD run examples/uff_methane.toml
gqteaMD run examples/uff_water_charged.toml
gqteaMD run examples/uff_ethene_explicit_topology.toml
```

The additional UFF examples demonstrate:

- `examples/uff_water_charged.toml`: explicit atom types, fixed charges, Coulomb electrostatics, shifted Lennard-Jones cutoff, and neighbor lists.
- `examples/uff_ethene_explicit_topology.toml`: explicit bonds, bond orders, angles, torsions, and inversions.

You can also run UFF directly from an XYZ file:

```
gqteaMD run examples/water.xyz --time-fs 10 --dt-fs 0.5 --force-provider
uff
```

The current electrostatics implementation is a simple direct pairwise Coulomb term. It does not include Ewald/PME summation, dielectric screening, charge-equilibration, or long-range tail corrections.

For cutoff nonbonded calculations, the UFF provider caches candidate pairs in a Verlet neighbor list. This improves scaling for larger systems because Lennard-Jones and Coulomb loops no longer need to inspect every atom pair at every force call. The reported metadata includes `neighbor_pair_count` and `neighbor_list_rebuilds`, which can help diagnose whether the skin distance is too small for the chosen timestep.

This UFF implementation still does not include full aromatic or ring typing, full periodic-table atom typing, or charge equilibration. Those features are planned later.

Reference implementations and parameter sources used while extending UFF:

- CP2K/deMon UFF table: `https://sources.debian.org/src/cp2k/8.1-9/data/DFTB/uff_table/`
- OpenMD UFF source: `https://openmd.org/wp-content/docs/code/_u_f_f_8hpp_source.html`
- RDKit UFF utilities: `https://www.rdkit.org/docs/cppapi/namespaceForceFields_1_1UFF_1_1Util`

12. Gaussian Force Provider

The Gaussian force provider writes a Gaussian input file, runs Gaussian, reads the output, and extracts energy and forces.

Example:

```
[force_provider]
type = "gaussian"
command = "g16"
route = "# B3LYP/6-31G(d) Force NoSymm SCF=Tight"
charge = 0
multiplicity = 1
nproc = 4
workdir = "gaussian_steps"
```

Fields:

- `type`: must be `"gaussian"`.
- `command`: Gaussian executable command.
- `route`: Gaussian route section.
- `charge`: molecular charge.
- `multiplicity`: spin multiplicity.
- `nproc`: optional number of shared-memory processes for Gaussian. When set, gqteaMD writes `%nprocshared=<nproc>` into each Gaussian input file.
- `workdir`: folder where Gaussian input/output files are written.

On Linux, the command is often:

```
command = "g16"
```

On Windows, it may be:

```
command = "g16.exe"
```

Use the command that works in your terminal.

gqteaMD automatically ensures that the Gaussian route includes:

```
Force
NoSymm
```

if those keywords are not already present.

Gaussian forces are read from the:

```
Forces (Hartrees/Bohr)
```

table and converted internally to:

```
eV/angstrom
```

Gaussian energies are converted from Hartree to eV.

13. Important Gaussian/PBC Note

gqteaMD has periodic boundary handling for the MD coordinates, but ordinary molecular Gaussian calculations are not automatically periodic.

This means:

- gqteaMD wraps atoms inside the orthorhombic box.
- Gaussian receives the atomic coordinates.
- Gaussian does not necessarily know about the periodic simulation cell.

For isolated molecules or cluster calculations, this is acceptable.

For true periodic materials, you need a quantum chemistry backend or method that supports periodic boundary conditions, or a future force-provider adapter designed for periodic quantum calculations.

14. Output Files

MD Log

The log file is whitespace-separated text with fixed-width numeric columns, matching the alignment style used by XYZ trajectory atom lines:

```
    step           time_fs     potential_eV       kinetic_eV
total_eV    temperature_K
0       0.00000000       0.09162318       0.00000000       0.09162318
0.00000000
```

```
1       0.25000000       0.09161770       0.00000548       0.09162318
0.01413698
```

Columns:

- `step`: MD step number.
- `time_fs`: simulation time in femtoseconds.
- `potential_eV`: potential energy in eV.
- `kinetic_eV`: kinetic energy in eV.
- `total_eV`: potential plus kinetic energy in eV.
- `temperature_K`: instantaneous temperature in kelvin.

If the log file already exists, new rows are appended to the end of the file. The header is not duplicated.

XYZ Trajectory

The trajectory file contains multiple XYZ frames:

```
3
step=0 time_fs=0.00000000 energy_eV=0.09162318412378322
O       0.00000000       0.00000000       0.00000000
H       0.95720000       0.00000000       0.00000000
H      -0.23998720       0.92662721       0.00000000
```

Coordinates are written in angstrom.

If the trajectory file already exists, new frames are appended to the end of the file.

GEOMETRY, Velocity, And Force Files

gqteaMD rewrites `GEOMETRY` at every calculation step. It contains one latest snapshot with ten atom-line columns: symbol, x, y, z, vx, vy, vz, Fx, Fy, Fz. Positions are in angstrom, velocities are in angstrom/fs, and forces are in eV/angstrom.

The legacy `pos.xyz`, `vel.xyz`, and `forces.xyz` files are no longer written.

For direct XYZ runs, use the matching command-line options:

```
gqteaMD run water.xyz --time-fs 10 --dt-fs 0.5 --velocities
traj.vel --forces traj.for
```

Restart File

If a `[restart]` section is present, gqteaMD writes a restart file at the requested interval. The default recommended filename is:

```
RESTART
```

This file is a compact NumPy archive written without an extension. It is meant for gqteaMD to read, not for manual editing.

To resume a failed run, edit the TOML file:

```
[restart]
path = "RESTART"
interval = 10
resume_from_RESTART = true
```

Then run the same command again:

```
gqteaMD run your_input.toml
```

The simulation continues from the saved step, time, positions, velocities, and forces in `RESTART`.

To restart from the latest output snapshot instead, make sure `GEOMETRY` and `RESTART` are present from a previous TOML run and set:

```
[restart]
resume_from_GEOMETRY = true
```

gqteaMD reads positions, velocities, and forces from `GEOMETRY`, then reads potential energy and total energy from `RESTART`.

15. Units

Internal units:

```
length: angstrom
time: femtosecond
mass: atomic mass unit
energy: eV
force: eV/angstrom
temperature: kelvin
```

Gaussian conversion constants:

```
1 Hartree = 27.211386245988 eV
1 Bohr = 0.529177210903 angstrom
1 Hartree/Bohr = 51.422067 eV/angstrom
```

16. Testing

Run the test suite:

```
python -m pytest
```

Expected result:

```
all tests passed
```

The UFF tests are organized as a validation ladder:

- topology detection and explicit topology override tests
- atom-type and parameter-table validation tests
- force-gradient checks for bond, angle, Lennard-Jones, and Coulomb terms
- torsion and inversion finite-force smoke tests
- nonbonded exclusion, 1-4 scaling, shifted-cutoff, and neighbor-list tests
- example TOML parsing tests for the bundled UFF examples

For force-field development, run at least:

```
python -m pytest tests/test_uff.py tests/test_cli.py
```

Then run representative examples:

```
gqteaMD run examples/uff_water.toml
gqteaMD run examples/uff_water_charged.toml
gqteaMD run examples/uff_ethene_explicit_topology.toml
```

Inspect the MD log for finite energies and the trajectory in a molecular viewer. The tests check mathematical consistency and input parsing, but they do not prove chemical transferability for every UFF atom type or molecule.

17. Troubleshooting

`gqteaMD` command is not recognized

The package may not be installed in the active Python environment.

Run:

```
python -m pip install -e .[dev]
```

Then try:

```
gqteaMD --help
```

Gaussian command fails

Check that Gaussian works directly from the same terminal.

For example:

```
g16
```

or:

```
g16.exe
```

If your Gaussian command has a different name, update the TOML file:

```
command = "your_gaussian_command"
```

`No built-in atomic mass for element`

The internal mass table does not yet include every element. Add the missing element to:

`gqteaMD/core/masses.py`

Energy jumps in Gaussian MD

Possible causes:

- Timestep is too large.
- SCF convergence is inconsistent between steps.
- Gaussian route section changed between runs.
- The system is not suitable for molecular Gaussian calculations with wrapped coordinates.

Try a smaller timestep and tighter SCF settings.

Access denied when using `Get-Content`

Do not run `Get-Content` on a folder. Use it on a file:

```
Get-Content .\gqteaMD\cli.py
```

To list files:

```
Get-ChildItem .\gqteaMD -Recurse
```

18. Current Limitations

The first version does not yet include:

- Thermostats.
- Barostats.
- Initial velocity generation from temperature.
- Classical dihedral and Coulomb force-field terms.
- Full UFF aromatic/ring typing, full periodic-table typing, charge equilibration, and long-range electrostatics.
- Constraints.
- Standard force-field topology readers.
- Neighbor lists for large classical simulations.
- Native periodic quantum backend.
- Parallel force calculations.
- Advanced trajectory formats.

The code is structured so these features can be added as separate modules.

19. Recommended First Workflow

1. Prepare an XYZ file.

2. Create a TOML configuration file.
3. Start with the harmonic force provider to confirm paths and output.
4. Switch to the UFF, classical, or Gaussian force provider.
5. Use a small timestep.
6. Inspect the log file for energy behavior.
7. Inspect the XYZ trajectory in a molecular viewer.