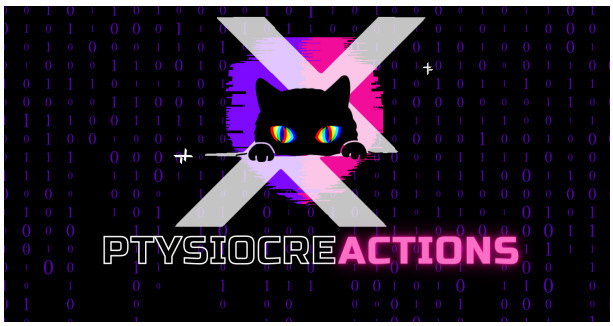
# GaussianMaster Package, ver. 0.1

by Michał Pocheć, PtysioCreACTIONS

updated on 04.08.2022




## Introduction

GaussianMaster Package is a modest Python3.X package allowing faster and more automated extraction and manipulation on logdata from Gaussian calculations. It currently supports extraction from results of optimalization and modredundant (scan) runs. The detailed description of the contents is presented below. With any questions, request or bug reporting please contact me on following:

email: lysy.pochec@gmail.com
LinkedIn: https://www.linkedin.com/in/micha%C5%82-poche%C4%87-b03a8a246/
GitHub: https://github.com/panpochec

## 1 Requirements

The program was tested under Python3.10 environment, but I have no data on other Python version compatibility. As it uses rather simple approach, it should be highly cross-version compatible, but I will try to maintain it for new Python releases.
External packages used:

- pandas - tested for 1.4.3
- numpy - tested for 1.23.1

Additionally, it uses *math* and *decimal*, contained within main Python3.10 distribution.

## 2 Getting started

First, make sure that you have satisfied requirements.
To install this package you can simply copy the folder **gaussianmaster** into either your workspace directory or follow:
https://www.geeksforgeeks.org/how-to-install-python-libraries-without-using-the-pip-command/
for the system-wide solution.
But the simplest solution is using PIP for installing. If you have PIP installed simply type in your command line:

```
pip install gaussianmaster
```

# 3 How to use the package

GaussianMaster package is branched into several different functions. Some functions rely on data from other functions for calculations, so it is best to feed data from one to the other. You can also manually input most of the data (prepare tables etc), but it's less convenient - so I recommend to use this method mostly for troubleshooting.
To use the package you have to import it using:

```
import gaussianmaster as gm
```

With the package imported, you can use several different commands. Below you can find the description of functions, but they are also available from docstrings within package.

## 3.1 full_matrix

This function opens the logfile from Gaussian and extracts the last optimized (standard optimalization) or input (scans) xyz matrix, as well as Gaussian-provided parameter tables. Example:

```
a, b, c, d, e, f, g = gm.full_matrix('file.log', 'opt')
```

The input parameters are (in order, example label given in parentheses):

- File name given as string ('file.log');
- Type of logfile - scan or standard optimalization ('opt').

You can only use 'scan' or 'opt' parameters as type!

The results are given as pd.DataFrames (all indexed from 1), described below (in order, example label given in parentheses).

- Atom Matrix - 4 columns: atom type, and XYZ coordinates ('a');
- Distance NumMatrix - 3 columns: atom1 number, atom2 number and value ('b');
- Distance AtomMatrix - 3 columns: atom1 type, atom2 type and value ('c');
- Angle NumMatrix - 4 columns: atom1 number, atom2 number, atom3 number and value ('d');
- Angle AtomMatrix - 4 columns: atom1 type, atom2 type, atom3 type and value ('e');
- Dihedral NumMatrix - 5 columns: atom1 number, atom2 number, atom3 number, atom4 number and value ('f');
- Dihedral AtomMatrix - 5 columns: atom1 type, atom2 type, atom3 type, atom4 type and value ('g').

## 3.2 bridge_atom_selection

This function searches the atom matrix for possible donors and protons of the specified type, as well as protons connected to the donors. Example:

```
a, b, c = gm.bridge_atom_selection(am, df, dfc, donor='O', acceptor='N')
```

The input parameters are (in order, example label given in parentheses):

- Atom matrix - pd.DataFrame with 4 columns: atom type, and XYZ coordinates ((am);
- Distance NumMatrix - pd.DataFrame with 3 columns: atom1 type, atom2 type and value ((df);
- Distance AtomMatrix - pd.DataFrame with 3 columns: atom1 type, atom2 type and value ((dfc);
- (Optional) Type for donor - by default 'O' (donor='O');
- (Optional) Type for acceptor - by default 'N' (acceptor='N').

The output parameters are given as(in order, example label given in parentheses):

- Donor atoms - list of atom numbers from XYZ atom matrix (a);
- Proton atoms - list of atom numbers from XYZ atom matrix (b);
- Acceptor atoms - list of atom numbers from XYZ atom matrix (c).

### 3.3 bridge_parameters

This function assigns acceptor and donor to a single proton, describing a single hydrogen bridge.

```
a, p, d= gm.bridge_parameters(don, prot, acc, am, which=0)
```

The input parameters are (in order, example label given in parentheses):

- Donor atoms - list of atom numbers from XYZ atom matrix (don)
- Proton atoms - list of atom numbers from XYZ atom matrix (prot)
- Acceptor atoms - list of atom numbers from XYZ atom matrix (acc)
- Atom matrix - pd.DataFrame with 4 columns: atom type, and XYZ coordinates (am);
- (Optional) Which proton from list is used to assign bridge, by default 0 (which=0)

The output parameters are given as(in order, example label given in parentheses):

- Bridge acceptor - position in Atom Matrix (a)
- Bridge proton - position in Atom Matrix (p)
- Bridge donor - position in Atom Matrix (d)

### 3.4 com_builder

Function creating text for input com files of different using given parameters.

```
xyz = com_builder(am, 'fpb', 'scan', 'meta', steps=20, dist=0.05, a=0, p=0, d=0, charge=0,
multiplicity=1)
```

The input parameters are (in order, example label given in parentheses):

- Atom matrix - pd.DataFrame with 4 columns: atom type, and XYZ coordinates (am)
- String of functional and basis, i.e **# wB97xD/6-311++G(2d,2p)** ('fpb')
- Type of comfile, string ('scan')
- Name - string of characters that will bu used as output name by Gaussian (meta);
- (Optional) Number of steps for scan, by default 20 (steps=20)
- (Optional) Distance for one step for scan, by default 0.05 (distance=20)
- (Optional) Acceptor atom number for scan (a=0)
- (Optional) Proton atom number for scan (p=0)
- (Optional) Donor atom number for scan (d=0)
- (Optional) Charge, by default 0 (charge=0)
- (Optional) Multiplicity, by default 1 (distance=1)

Function produces xyz string, which has the formatting of Gaussian COM file.

## 3.5 energies_scan

Function creating DataFrame with energies for each step of the scan file.

```
tab, list = gm.energies_scan('meta_BridgeOne.log', conv=True)
```

The input parameters are (in order, example label given in parentheses):

- Filename, given as string ('meta_BridgeOne.log')
- (Optional) Convertion from values in Hartree to differences from first in kcal/mol, by deafault true (conv=True)

Function produces DataFrame with columns 'Bond length' and 'Energy' or Energy difference'. It also produces list of atom numbers from the scanned angle, which then can be cross-referenced with bridge atoms from **bridge_parameters**.

## 3.6 Minor functions

### 3.6.1 extraction_atom_dataframe

This function is a part of **full_matrix**, so description and parameters are the same as in **3.1**

```
import gaussianmaster.core.extractor as ma
```

then

```
atom_matrix = ma.extraction_atom_dataframe(name, filetype)
```

### 3.6.2 distance_calc

Function calculating distance between two chosen atoms.

```
gm.distance_calc(where, atom1_number, atom2_number)
```

The input parameters are (in order, example label given in parentheses):

- Atom matrix as given earlier (where);
- Atom 1 - integer of atom 1 position in atom matrix (atom1_number);
- Atom 2 - integer of atom 2 position in atom matrix (atom2_number);

Function returns distance in 3D space as float.

# 4 Examples

## 4.1 Building COM file for scan from Gaussian output logfile

The following is a use case of GaussianMaster, which acquires data from Gaussian logfile to create a complete input for scan.

```python
import gaussianmaster as gm

# Acquiring and parametric data
atom_matrix, distance_frame, distance_frame_clean,\
    angle_frame, angle_frame_clean,\
    dihedral_frame, dihedral_frame_clean = gm.full_matrix('meta.log', 'opt')

# Assigning potential bridge atoms
don, prot, acc = gm.bridge_atom_selection(atom_matrix, distance_frame, distance_frame_clean, donor='O', acceptor='N')

# Building hydrogen bridges
a, p, d = gm.bridge_parameters(don, prot, acc, atom_matrix)
a2, p2, d2 = gm.bridge_parameters(don, prot, acc, atom_matrix, which=2)

# Building COM contents
xyz = gm.com_builder(atom_matrix, '# PBEPBE/6-311++G(2d,2p)',
                        'scan', 'meta_BridgeOne', a_bridge=a, p_bridge=p, d_bridge=d)

# Writing COM contents to file
comfile = open('meta.com', "w")
comfile.write(xyz)
comfile.close()

```

Examples of how the DataFrames look:

atom_matrix

| | Atom type | x | y | z |
|---|---|---|---|---|
| 1 | C | 0.00009 | 0.30716 | -0.41469 |
| 2 | H | 0.00008 | -0.76670 | -0.29001 |
| 3 | C | -1.20544 | 0.99476 | -0.48131 |
| 4 | C | -1.20440 | 2.37572 | -0.67918 |
| 5 | H | -2.14010 | 2.90488 | -0.79691 |
| 6 | C | 0.00011 | 3.05293 | -0.78203 |
| 7 | H | 0.00011 | 4.11900 | -0.96448 |
| 8 | C | 1.20461 | 2.37569 | -0.67922 |
| 9 | H | 2.14034 | 2.90477 | -0.79695 |
| 10 | C | 1.20564 | 0.99473 | -0.48132 |
| 11 | N | 2.39615 | 0.24881 | -0.39429 |
| 12 | C | 3.37635 | 0.68791 | 0.29847 |
| 13 | H | 3.28989 | 1.61891 | 0.86730 |
| 14 | C | 4.64246 | -0.01265 | 0.39085 |
| 15 | C | 5.66990 | 0.52284 | 1.17619 |
| 16 | H | 5.49023 | 1.45413 | 1.69976 |
| 17 | C | 6.88873 | -0.10952 | 1.28979 |
| 18 | H | 7.67316 | 0.31542 | 1.88895 |
| 19 | C | 7.09401 | -1.30821 | 0.60604 |
| 20 | H | 8.04632 | -1.81523 | 0.68895 |
| 21 | C | 6.10028 | -1.86249 | -0.17462 |

angle_frame_clean

| | Atom 1 | Atom 2 | Atom 3 | Angle |
|---|---|---|---|---|
| 1 | H | C | C | 119.80780 |
| 2 | H | C | C | 119.80810 |
| 3 | C | C | C | 120.37230 |
| 4 | C | C | C | 119.73750 |
| 5 | C | C | N | 117.94020 |
| 6 | C | C | N | 122.27900 |
| 7 | C | C | H | 119.94590 |
| 8 | C | C | C | 119.66740 |
| 9 | H | C | C | 120.32650 |
| 10 | C | C | H | 119.62010 |
| 11 | C | C | C | 120.74730 |
| 12 | H | C | C | 119.62100 |
| 13 | C | C | H | 120.33010 |
| 14 | C | C | C | 119.66820 |
| 15 | H | C | C | 119.94170 |
| 16 | C | C | C | 119.73640 |
| 17 | C | C | N | 117.94630 |
| 18 | C | C | N | 122.27380 |
| 19 | C | N | C | 120.00680 |
| 20 | N | C | H | 120.88980 |
| 21 | N | C | C | 122.56300 |

The resulting comfile:

```
# PBEPBE/6-311++G(2d,2p) opt=modredundant

meta_BridgeOne

0 1
C              0.000093   0.307161  -0.414687
H              0.000078  -0.766696  -0.290005
C             -1.205442   0.994762  -0.481308
C             -1.204402   2.37572   -0.67918
H             -2.140097   2.904875  -0.796913
C              0.000106   3.052929  -0.782032
H              0.000106   4.119     -0.964483
C              1.204606   2.37569   -0.679224
--------------------------------------------------------
H              2.140344   2.904773  -0.796953
C             -6.888983  -0.108735   1.289051
H             -7.673529   0.31672    1.897696
C             -7.094184  -1.307936   0.606169
H             -8.046532  -1.814862   0.68729
C             -6.100339  -1.862857  -0.173869
H             -6.252782  -2.792003  -0.704321
C             -4.864382  -1.226327  -0.293132
O             -3.927465  -1.793382  -1.054803
H             -3.127184  -1.222366  -1.021336

24 25 11 F
24 25 S 20 0.05
```

This can also be automated for multiple files - by using glob and iterating through list of files. All one needs to do is create variable with name (i.e name), and then use it as parameter for functions (use name + '.log', name + '_BridgeOne', name + '_BridgeOne.com'):

```
from pathlib import Path
result = [f.name for f in dirPath.glob('*.log') if f.is_file()]
for x in result:
    name = x
```

## 4.2 Acquiring scan energy profile

For scan runs, code can be appended by additional part - acquisition of energy profile for each step of the scanning run. This is achieved using following lines:

```
z, scaned_bridge = gm.energies_scan('meta_BridgeOne.log', conv=True)

if p in scaned_bridge:
    bridge = [a, p, d]
elif p2 in scaned_bridge:
    bridge = [a2, p2, d2]
```

Code above allows two things:

- it reads what atoms were subjected to modredundant (this will usually be hydrogen bridge, stored as scaned bridge in the code snippet) and then checks if any of the protons from bridge_parameters are in the resulting list;
- generates DataFrame of energies for each step of the scan

An example of energies table (stored as **z** in the code snippet) is given below:

z

| | Bond Lenght | Energy difference [kcal/mol] |
|---|---|---|
| 0 | 0.00000 | 0.00000 |
| 1 | 0.05000 | 0.89896 |
| 2 | 0.10000 | 2.95423 |
| 3 | 0.15000 | 5.27046 |
| 4 | 0.20000 | 7.09957 |
| 5 | 0.25000 | 8.08997 |
| 6 | 0.30000 | 8.34249 |
| 7 | 0.35000 | 8.12632 |
| 8 | 0.40000 | 7.69426 |
| 9 | 0.45000 | 7.22730 |
| 10 | 0.50000 | 6.83183 |
| 11 | 0.55000 | 6.57793 |
| 12 | 0.60000 | 6.49496 |
| 13 | 0.65000 | 6.58835 |
| 14 | 0.70000 | 6.87376 |
| 15 | 0.75000 | 7.31985 |
| 16 | 0.80000 | 7.92639 |
| 17 | 0.85000 | 8.67976 |
| 18 | 0.90000 | 9.57165 |
| 19 | 0.95000 | 10.58773 |
| 20 | 1.00000 | 11.72029 |

## Parting words

I hope, that this small package will allow a smoother and faster experience of your Gaussian logfile manipulation. If you encounter any bugs or want to request additional functionality - message me using contact info from the beginning of this tutorial.

I wish you smooth sailing and may the odds be ever in your favour

Michał Pocheć
PtysioCreACTIONS