PyFrag Documentation

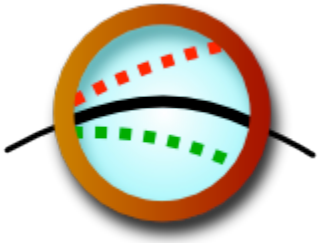
Release 0.1.0

X. Sun

Apr 13, 2023

CONTENTS

Contents:



CHAPTER

ONE

PYFRAG 2019

See documentation for tutorials and documentation.

1.1 NOTICE(October 28 2021)

Since ADF2019, the ADF has been reconfigured and renamed as AMS2020 and AMS2021. Accordingly the format of input has also changed a lot. PyFrag has now been updated as well to be compatible with these changes. Old users can delete old version and reinstall the new version. If user still chooses ADF2019 or older version of ADF as the computational engine, one can use the command `pyfrag -x adfold job.in` to invoke PyFrag.

1.2 Motivation

The PyFrag 2019 program was specially designed to facilitate the analysis of reaction mechanism in a more efficient and user-friendly way. The original PyFrag 2008 workflow facilitated the characterization of reaction mechanisms in terms of the intrinsic properties, such as strain and interaction, of the reactants. This approach is routinely applied in the Bickelhaupt Group to understand numerous organic, inorganic, and biomolecular reactions/processes. The new PyFrag 2019 program has automated and reduced the time-consuming and laborious task of setting up, running, analyzing, and visualizing computational data from reaction mechanism studies to a single job. PyFrag 2019 resolves three main challenges associated with the automatized computational exploration of reaction mechanisms: 1) the management of multiple parallel calculations to automatically find a reaction path; 2) the monitoring of the entire computational process along with the extraction and plotting of relevant information from large amounts of data; and 3) the analysis and presentation of these data in a clear and informative way. The activation strain and canonical energy decomposition results that are generated, relate the characteristics of the reaction profile in terms of intrinsic properties (strain, interaction, orbital overlaps, orbital energies, populations) of the reactant species.

1.3 Description

1.3.1 Usage

In order to see all the commands that can be used in this program, the user can type pyfrag -h, which will show:

```
Usage: pyfrag [-h] [-s] [-x command]  [...]
-h          : print this information
-s          : run job quietly
-x          : start the executable named command
            : command include restart, which restart job
```

(continues on next page)

(continued from previous page)

```
            : restart, which restart a job after it is stoped
            : summary, which summarize all job result after jobs finished
            : default command is pyfrag itself
The example command is like as follow, in which job.in is job input
pyfrag job.in
or
pyfrag -x restart job.in
or
pyfrag -s -x summary job.in
```

1.3.2 Input example

A simple job input is provided below. The input script can be roughly divided into four section: the required submit information for a job scheduling system (Slurm in this example), ADF parameters, pyfrag parameters, and geometry parameters. Additional information about the input file can be found in input explanation and main specifications in the following webpages.

```
JOBSUB
#!/bin/bash
#SBATCH -J frag_1
#SBATCH -N 1
#SBATCH -t 50:00
#SBATCH --ntasks-per-node=24
#SBATCH --partition=short
#SBATCH --output=%job.stdout
#SBATCH --error=%job.stdout
export NSCM=24

JOBSUB END

ADF

basis
type TZ2P
core Small
end

xc
gga OPBE
end

relativistic SCALAR ZORA

scf
iterations 299
converge 0.00001
mixing 0.20
end

numericalquality verygood
```

(continues on next page)

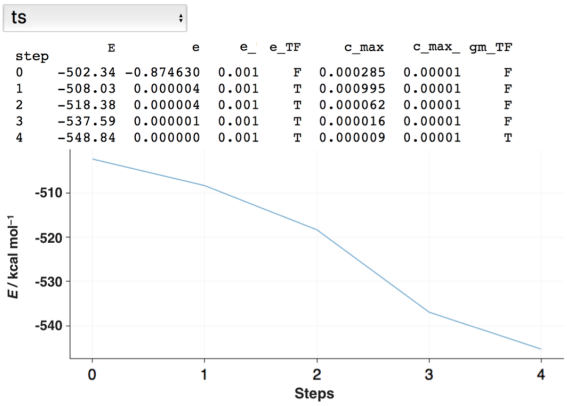
(continued from previous page)

```
charge 0 0
symmetry auto

ADF END

PyFrag

fragment  2
fragment  1 3 4 5 6
strain    0
strain   -554.09
bondlength 1 6  1.09

PyFrag END

Geometrycoor

R1: Fe-II(CO)4 + CH4
Pd       0.00000000       0.00000000       0.32205546

R2: CH4
C        0.00000000       0.00000000      -1.93543634
H       -0.96181082       0.00000000      -1.33610429
H        0.00000000      -0.90063254      -2.55201285
H        0.00000000       0.90063254      -2.55201285
H        0.96181082       0.00000000      -1.33610429

RC: Fe-II(CO)4 + CH4
C        0.00000000       0.00000000      -1.93543615
Pd       0.00000000       0.00000000       0.322055
H       -0.96181082       0.00000000      -1.33610429
H        0.00000000      -0.90063254      -2.55201285
H        0.00000000       0.90063254      -2.55201285
H        0.96181082       0.00000000      -1.33610429

TS: Fe-II(CO)4 + CH4
C       -1.74196777      -2.22087997       0.00000000
Pd      -2.13750904      -0.23784341       0.00000000
H       -2.80956968      -2.49954731       0.00000000
H       -1.26528821      -2.62993236       0.8956767
H       -1.26528821      -2.62993236      -0.895676
H       -0.75509932      -0.88569836       0.00000000

P: Fe-II(CO)4 + CH4
C       -2.10134690      -2.41901732       0.1862099
Pd       -2.73145901      -0.57025833       0.419766
H       -3.88639130      -1.04648079      -0.43099501
H       -2.78392696      -3.12497645       0.66994616
H       -1.97386865      -2.66955518      -0.87144525
H       -1.12556673      -2.41201402       0.698583

Geometrycoor END
```

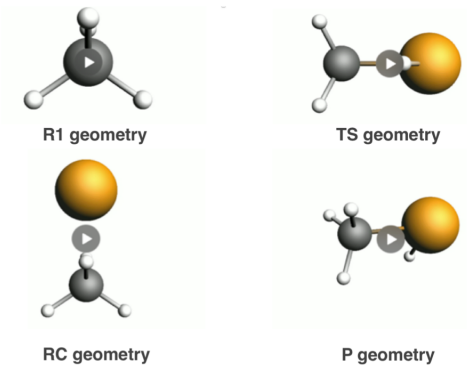
1.3.3 Result example

After the job has been submitted, a website as provided in the figure below will be launched that summarizes all relevant information, including: a) the convergence information, b) the latest structure from the optimization in the form of movie, c) the latest energy and coordinates, and d) the activation strain analysis (if a job is finished). The user can decide if the trend of optimization is right or wrong, and if necessary, the job can be stopped. If the input file has been modified, the job will be resubmitted and the overall workflow will resume from where it stopped before.

a) Current Job Status



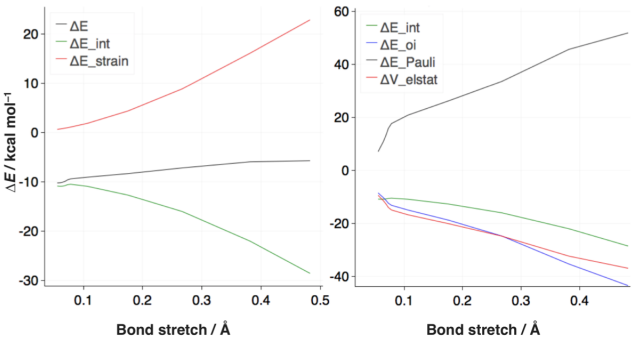
b) Movies of Job Process



c) Summary of latest coordinates and other information

Geometry (cartesian) and Energy (kcal/mol)
R1.xyz, -554.0858
1.C 0.00000000 0.00000000 0.00000000
2.H 0.63276400 -0.63276400 0.63276400
3.H -0.63276400 -0.63276400 -0.63276400
4.H -0.63276400 0.63276400 0.63276400
5.H 0.63276400 0.63276400 -0.63276400
P.xyz, -552.4892
1.C -2.10134690 -2.41901732 0.18620613
2.Pd -2.73145901 -0.57025833 0.41934115

d) Activation Strain Analysis



1.3.4 Installation

For installation, please read installation.

CHAPTER

TWO

INSTALLATION

2.1 Activation Strain Analysis (ASA) Module of PyFrag 2019

The user may choose to only install the part of the program needed to perform the Activation Strain Analysis (ASA) based on Activation Strain Model (ASM). Note that Python3 is needed to run this program. The ASA can be performed using a variety of quantum chemical software packages, including: ADF, Gaussian, Orca and Turbomole, given a series of coordinate from the potential energy surface is provided.

To install the ASA module of PyFrag 2019, the user must complete the following step. Go to your host machine (supercomputer or cluster), open a terminal and run the following command:

```
curl -L -o install_alone.sh https://raw.githubusercontent.com/TheoChem-VU/PyFrag/master/
install_alone.sh
```

```
bash install_alone.sh
```

To run a simple test, open a terminal window on your host machine, make a directory, enter into that directory and run the following command to download the job input file (job.in) and coordinate file (molecule.xyz):

```
curl -L -o job.in       https://raw.githubusercontent.com/TheoChem-VU/PyFrag/master/host/
standalone/adf_new/example/job.in
```

```
curl -L -o molecule.xyz  https://raw.githubusercontent.com/TheoChem-VU/PyFrag/master/host/
standalone/adf_new/example/molecule.xyz
```

Change the ircpath and the submit information, such as the number of nodes and wall time, located in job.in using vim or any other text editor according to your situation, and run:

```
pyfrag job.in
```

The user can also download the module for either ADF, Gaussian, Orca, and Turbomole separately from PyFrag standalone and run it as a normal python code in your laptop or desktop. An input sample is provided in the example folder along with the source code file.

2.2 The Complete PyFrag 2019 Package

The entire PyFrag 2019 package is only compatible with ADF at the moment. . For optimal use of PyFrag 2019, one part of the program is installed on the users' local machine and the second part is installed on the users' host machine (supercomputer or cluster) where the heavy computational jobs is running. The user must ensure to transport their public key to your host machine to allow the communication between your local and host machine. The following installation bash script (install_local.sh, install_host.sh) is was designed to make the installation process as simple as possible. However, for the advanced user, if a different configuration of the program is desired, please read the detailed comments in the installation bash script and set up the program accordingly. To install and test PyFrag 2019, the user must perform the following three steps:

1) Go to your local machine (your laptop or desktop), open a terminal window and run the following command on your terminal:

```
xcode-select --install
```

```
curl -L -o install_local.sh https://raw.githubusercontent.com/TheoChem-VU/PyFrag/master/
install_local.sh
```

```
bash install_local.sh
```

2) Go to your host machine (supercomputer or cluster), open a terminal window and run the following command:

```
curl -L -o install_host.sh https://raw.githubusercontent.com/TheoChem-VU/PyFrag/master/
install_host.sh
```

```
bash install_host.sh
```

3) Open a terminal window on your local machine, make a directory, enter into that directory and run the following command:

```
curl -L -o job.in https://raw.githubusercontent.com/TheoChem-VU/PyFrag/master/example/
job.in
```

Change the submit information, such as the number of nodes and wall time, located in job.in using vim or any other text editor, and run:

```
pyfrag job.in
```

To obtain the latest information about your job, the user can run:

```
pyfrag -x summary job.in
```

CHAPTER

THREE

BASIC USAGE TUTORIAL

3.1 Usage

The user can type pyfrag -h to see all the commands that can be used in this program, which will show:

```
Usage: pyfrag [-h] [-s] [-x command]  [...]
-h          : print this information
-s          : run job quietly
-x          : start the executable named command
            : command include restart, which restart job
            : restart, which restart a job after it is stoped
            : summary, which summarize all job result after jobs finished
            : default command is pyfrag itself
The example command is like as follow, in which job.in is job input
pyfrag job.in
or
pyfrag -x restart job.in
or
pyfrag -s -x summary job.in
```

To submit a job, create a directory and generate a input file and run the following command to submit a job. Note for each job, a new directory and a new job name should be given. Note: The user should avoid running more than one job in a single directory.

```
pyfrag job.in
```

To obtain the latest information about your job, the user can run:

```
pyfrag -x summary job.in
```

If a change in the input file is required, make the change and the resubmit the job using:

```
pyfrag -x restart job.in
```

3.2 Sample Input Example

A simple job input is provided below. The input script can be roughly divided into four section: the required submit information for a job scheduling system (Slurm in this example), ADF parameters, PyFrag 2019 parameters, and geometry parameters.

```
""
JOBSUB section is for the information passed to the remote host machine
where the heavy computational job is done! It is written in the fashion of Slurm.
"'
JOBSUB

#!/bin/bash
#SBATCH -J frag_1
#SBATCH -N 1
#SBATCH -t 50:00
#SBATCH --ntasks-per-node=24
#SBATCH --partition=short
#SBATCH --output=%job.stdout
#SBATCH --error=%job.stdout
export NSCM=24

JOBSUB END

""
Provide the parameters for a DFT calculation using ADF software.
"'
ADF

basis
type TZ2P
core Small
end

xc
gga OPBE
end

relativistic SCALAR ZORA

scf
iterations 299
converge 0.00001
mixing 0.20
end

numericalquality verygood

charge 0 0
symmetry auto


ADF END
```

(continues on next page)

(continued from previous page)

```
""
Provide the parameters for an activation strain analysis.
Noted a bondlength calculation is needed to provilde x axis value for ASA.
"'

PyFrag

fragment  2
fragment  1 3 4 5 6
strain    0
strain   -554.09
bondlength 1 6  1.09

PyFrag END


""
Guessed geometry coordinate for reactent1, reactent2, reactent complex,
transition state and product.
"'

Geometrycoor

R1: Fe-II(CO)4 + CH4
Pd       0.00000000       0.00000000       0.32205546

R2: CH4
C        0.00000000       0.00000000      -1.93543634
H       -0.96181082       0.00000000      -1.33610429
H        0.00000000      -0.90063254      -2.55201285
H        0.00000000       0.90063254      -2.55201285
H        0.96181082       0.00000000      -1.33610429

RC: Fe-II(CO)4 + CH4
C        0.00000000       0.00000000      -1.93543615
Pd       0.00000000       0.00000000       0.322055
H       -0.96181082       0.00000000      -1.33610429
H        0.00000000      -0.90063254      -2.55201285
H        0.00000000       0.90063254      -2.55201285
H        0.96181082       0.00000000      -1.33610429

TS: Fe-II(CO)4 + CH4
C       -1.74196777      -2.22087997       0.00000000
Pd      -2.13750904      -0.23784341       0.00000000
H       -2.80956968      -2.49954731       0.00000000
H       -1.26528821      -2.62993236       0.8956767
H       -1.26528821      -2.62993236      -0.895676
H       -0.75509932      -0.88569836       0.00000000

P: Fe-II(CO)4 + CH4
C       -2.10134690      -2.41901732       0.1862099
```

(continues on next page)

(continued from previous page)

```
Pd      -2.73145901      -0.57025833       0.419766
H      -3.88639130      -1.04648079      -0.43099501
H      -2.78392696      -3.12497645       0.66994616
H      -1.97386865      -2.66955518      -0.87144525
H      -1.12556673      -2.41201402       0.698583

Geometrycoor END
```

The user might want to specify an additional input for the different sections of the overall workflow. To specify additional information for say, fragment1 and fragment2 see the syntax shown below. Additional complex insert statements for the fragment analysis calculation can be added. Similarly, the R1 EXTRA, R2 EXTRA, RC EXTRA, TS EXTRA, P EXTRA, IR EXTRA insert statements for R1, R2, RC, TS, P, IRC calculation.

```
fragment1 EXTRA
charge 1
fragment1 EXTRA END

fragment2 EXTRA
charge -1
fragment2 EXTRA END

complex EXTRA
charge 2
complex EXTRA END

R1 EXTRA
charge 0
R1 EXTRA END

R2 EXTRA
charge 0
R2 EXTRA END

RC EXTRA
charge 0
RC EXTRA END

TS EXTRA
charge 0
tsrc
Bond 1 2 -1
end
TS EXTRA END

P EXTRA
charge 0
P EXTRA END

IR EXTRA
Geometry
 IRC Backward POINTS=20 STEP=1
ITERATIONS 300
```

(continues on next page)

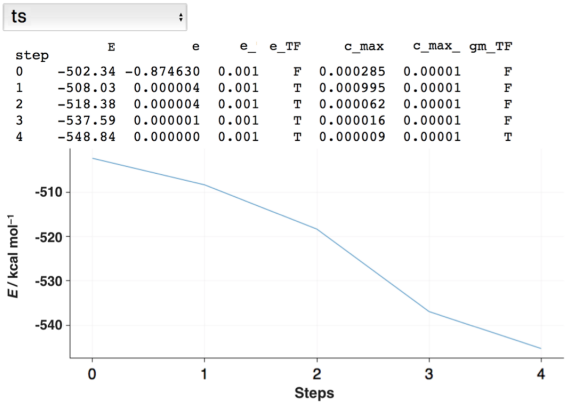
(continued from previous page)

```
CONVERGE 0.000001
End
IR EXTRA END
```

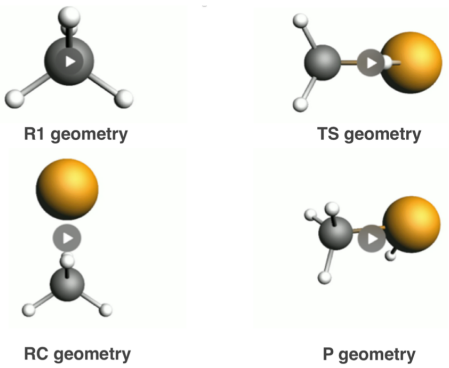
3.3 Result example

After the job has been submitted, a website as shown in the figure below will be launched. The website summarizes all relevant information, including: a) the convergence criteria, b) the latest structure from the optimization in the form of movie, c) the latest energy and coordinates, and d) the activation strain analysis (once the complete workflow has finished). The user can decide if the optimization process is correct or incorrect, and if necessary, can stop the job. If the input file is then modified or updated, the job will be resubmitted and the overall workflow will resume from where it left off.

a) Current Job Status



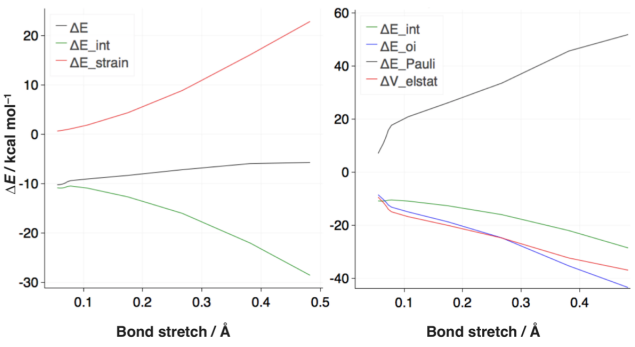
b) Movies of Job Process



c) Summary of latest coordinates and other information

Geometry (cartesian) and Energy (kcal/mol)
R1.xyz, -554.0858
1.C 0.00000000 0.00000000 0.00000000
2.H 0.63276400 -0.63276400 0.63276400
3.H -0.63276400 -0.63276400 -0.63276400
4.H -0.63276400 0.63276400 0.63276400
5.H 0.63276400 0.63276400 -0.63276400
P.xyz, -552.4892
1.C -2.10134690 -2.41901732 0.18620613
2.Pd -2.73145901 -0.57025833 0.41934115

d) Activation Strain Analysis



CHAPTER

FOUR

MAIN SPECIFICATIONS

The user can print additional information in the final activation strain analysis by adding the following specifications between the PyFrag and PyFrag END in the job input.

These following statements allow the user to define the fragments in the analysis. For each of the fragments you supply a list of the numbers for the atoms as they exist in the supplied XYZ coordinate file of the reaction path from an IRC or LT calculation. The program will check if they match with the order in the supplied XYZ coordinate file of the reaction path from an IRC or LT calculation. If the atom ordering is incorrect, a statement will be printed in the error log.

```
fragment atomnrs

for example:

fragment 2
fragment 1 3 4 5 6
```

The following possibilities are optional. The user can choose what information to print during the Activation Strain Analysis (ASA). For instance, the user can specify to print the strain energy for the fragments. Or one can specify the equilibrium energies (in kcal/mol) for the fragments. Beware that the order of this specification should correspond to the order of the fragment definition. This value will then simply be subtracted from the energy of the fragment in question. The program will print the individual strain values for each fragment, plus the total strain and total energy.

```
strain fragenergy

for example:

strain -301.01
strain -19.02
```

To specify the bond length to be printed for each geometry step, the user needs to indicate the atom numbers as they appear in the input order of the total molecule. Specifying bond_diff as well subtracts this value from the actual bond length.

```
bondlength atomnr1 atomnr2 bond-diff
```

To specify the angle between atoms 1, 2, and 3 to be printed for each geometry step, just indicate the atom numbers as they appear in the input order of the total molecule. Specifying angle_diff as well subtracts this value from the actual angle.

```
angle atomnr1 atomnr2 atomnr3 angle-diff
```

The following statement will print the Hirshfeld charges for the fragment. Note that the order of the fragments as used internally by ADF may differ from what you would expect. Note also that Hirshfeld charges as computed in a fragment

analysis differ from those obtained in a 'normal' calculation from basic (spherical average-of configurations) ADF atoms like simple single point calculations.

```
hirshfeld frag1
```

The following statement will print the VDD charges on the atoms with numbers as given by atomnrs. Note also that VDD charges as computed in a fragment analysis differ from those obtained in a 'normal' calculation from basic (spherical average-of configurations) atoms like single point calculation.

```
VDD atomnrs

for example:

VDD 1 2
```

The following statement will print the orbital interaction energy per available irrep. The irrep symbol relates to the symmetry of the whole molecule.

```
irrepOI oi irrep

for example:

irrepOI AA
```

The following statement will print the overlap between orbital numbers orb1 on frag1 and orb2 on frag2 in irrep as they appear in the fragment analysis calculation. It can also print the overlap between the HOMOs of fragment 1 (frag1) and the LUMOs of fragment 2 (orbitals as found on the fragment calculations). If the orbitals found in this way differ in symmetry, an orbital overlap value of zero is returned. It should be noted that irrep and orbital number refers to each fragment, rather than the whole molecule. Especially, when it comes to frozen core situation, the count of orbital number should not include core orbitals. This rule also applies to the situation of printing orbital energy and population for orbitals of certain fragment.

```
overlap frag1 HOMO/LUMO frag2 HOMO/LUMO
overlap irrep1 frag1 orb1 irrp2 frag2 orb2

for example:

overlap frag1 HOMO frag2 LUMO
overlap frag1 HOMO-1 frag2 LUMO+3
overlap S frag1 5 AA frag2 4
```

The following statement will print the orbital energy for a fragment orbital per available irrep. The irrep symbol relates to the symmetry of the fragment.

```
orbitalenergy frag HOMO/LUMO
orbitalenergy irrep frag orb

for example:

orbitalenergy frag1 HOMO
orbitalenergy frag1 HOMO-2
orbitalenergy AA frag2 5
```

The following statement will print the gross Mulliken population for a fragment orbital.

```
population frag HOMO/LUMO
population irrep frag orb

for example:

population frag1 HOMO
population frag2 HOMO-1
population AA frag2 5
```

CHAPTER

FIVE

SIMPLE PYFRAG CALCULATION

The user may choose to only install the part of the program needed to perform the Activation Strain Analysis (ASA) based on Activation Strain Model (ASM). The ASA can be performed using a variety of quantum chemical software packages, including: ADF, Gaussian, Orca, and Turbomole. The user must only provide a series of coordinate from the reaction path. An input sample is provided in the standalone example folder for either ADF, Gaussian, Orca and Turbomole.

5.1 ADF

The basic PyFrag 2019 input for the Activation Strain Analysis (ASA) using ADF is similar to the previous input example, except the coordinate section:

```
JOBSUB

#!/bin/bash
#SBATCH -J NNC
#SBATCH -N 1
#SBATCH -t 24:00:00
#SBATCH --ntasks-per-node=24
#SBATCH --partition=normal
#SBATCH --output=%job.stdout
#SBATCH --error=%job.stdout
export NSCM=24

JOBSUB END

PyFrag

ircpath /home/x2sun/pyfragnew/test/molecule.xyz
fragment  2
fragment  1 3 4 5 6
strain    0
strain   -554.09
bondlength 1 6  1.09

PyFrag END

ADF

basis
```

(continues on next page)

(continued from previous page)

```
type TZ2P
core Small
end

xc
gga OPBE
end

relativistic SCALAR ZORA

scf
iterations 299
converge 0.00001
mixing 0.20
end

numericalquality verygood

charge 0
symmetry auto

ADF END
```

The only difference compared with the previous job input is that the user must provide the path to a series of coordinates. For additional specifications, the user can read the previous page. The user can specify the filename of a linear transit (LT), scan, IRC, or a simple text output file you wish to analyze. Note: When using a TAPE21, the file should have the .t21 extension. The user can specify two output files for an IRC TAPE21 (backward and forward). A text file containing coordinates of an IRC calculation is also acceptable, for example, a text file generated by ADFmovie or Gaussian.

```
ircpath path/filename.xyz
irct21 path/filename.t21
lt path/filename.t21
```

To submit a job, create a directory and generate a input file and run the following command to submit a job:

```
pyfrag job.in
```

If more terms needed to be printed, user can simply re-run the job as long as the plams directory exists. During the process, PyFrag just extracts the new information from the previous result without actually doing new calculation. In order to do that, the path to the plams directory should be specified in the job file in the PyFrag section, such as:

```
PyFrag

restartjob /path/to/plams/directory

ircpath /home/x2sun/pyfragnew/test/molecule.xyz
fragment  2
fragment  1 3 4 5 6
strain    0
strain   -554.09
bondlength 1 6  1.09

PyFrag END
```

To submit a job, create a directory and generate a input file and run the following command to submit a job:

```
pyfrag job.in
```

5.2 Gaussian

The basic PyFrag 2019 input for the Activation Strain Analysis (ASA) using Gaussian is as follows:

```
INPUT_SPECS
type = IRC
output file = Ethylene-forward.amv

frag1 = C4H6
1
2
3
4
5
6
7
8
13
14
end frag1
frag2 = C2H4
9
10
11
12
15
16
end frag2

print bond 1 9 1.384
print strain frag1  1000
print strain frag2  2000

END INPUT_SPECS


"g09" <<eor

%nprocs=16
%mem=14000mb
#OPBE/6-31G*

Comments

0 1
END INPUT
```

The first section between INPUT_SPECS and END INPUT_SPECS is used to define fragment and provide coordinate path. The second section between END INPUT_SPECS and END INPUT is used to do Gaussian parameter set up.

User can also specify different parameters for fragment 1, fragment 2 and tatol complex in the extra section such as between EXTRA frag1 and END EXTRA frag1 using input example as below:

```
INPUT_SPECS
type = IRC
output file = Ethylene-forward.amv
fa1_name = complex
frag1 = C4H6
1.C
2.H
3.C
4.H
5.C
6.H
7.C
8.H
13.H
14.H
end frag1
frag2 = C2H4
9.C
10.C
11.H
12.H
15.H
16.H
end frag2

print bond 1 9 1.384
print strain frag1  1000
print strain frag2  2000

END INPUT_SPECS

END INPUT


EXTRA frag1
"g09" <<eor

%nprocs=16
%mem=14000mb
#OPBE/6-31G* -1

Comments

0 1
END EXTRA frag1


EXTRA frag2
"g09" <<eor
```

(continues on next page)

(continued from previous page)

```
%nprocs=16
%mem=14000mb
#OPBE/6-31G* 0

Comments

0 1
END EXTRA frag2


EXTRA fa

"g09" <<eor

%nprocs=16
%mem=14000mb
#OPBE/6-31G* -1

Comments

0 1
END EXTRA fa
```

To submit a job, create a directory and generate a input file and run the following command to submit a job:

```
pyfrag -x gaussian job.in
```

5.3 Orca

The basic PyFrag 2019 input for the Activation Strain Analysis (ASA) using Orca is as follows:

```
INPUT_SPECS
type = IRC
output file = irc.amv
frag1 = H2
1
2
end frag1
frag2 = H2
3
4
end frag2

print bond 1 3 1.00
print strain frag1  100
print strain frag2  200

END INPUT_SPECS

! SP B3LYP 6-31G(d)
```

(continues on next page)

(continued from previous page)

```
* xyz 0 1
END INPUT
```

The first section between INPUT_SPECS and END INPUT_SPECS is used to define fragment and provide coordinate path. The second section between END INPUT_SPECS and END INPUT is used to do Orca parameter set up. To submit a job, create a directory and generate a input file and run the following command to submit a job:

```
pyfrag -x orca job.in
```

5.4 Turbomole

The basic PyFrag 2019 input for the Activation Strain Analysis (ASA) using Turbomole is as follows:

```
INPUT_SPECS
type = IRC
output file = irc.amv
frag1 = pd
1
2
end frag1
frag2 = cc
3
4
end frag2

print bond 1 3 1.00
print strain frag1  100
print strain frag2  200

END INPUT_SPECS
%method
ENRGY :: b-p/SVP [gen_stat=1,scf_msil=99,&
                  scf_grid=m4]
%charge
0
%coord
%end
END INPUT
```

The first section between INPUT_SPECS and END INPUT_SPECS is used to define fragment and provide coordinate path. The second section between END INPUT_SPECS and END INPUT is used to do Turbomole parameter set up. To submit a job, create a directory and generate a input file and run the following command to submit a job:

```
pyfrag -x turbomole job.in
```

CHAPTER

SIX

SPECIAL PYFRAG CALCULATION

Besides the above simple calculations, it is more complicated to perform an open shell Activation Strain Analysis (ASA) using PyFrag 2019 for the technical reasons. For more information please check the example consisting of an analysis of the C-C single bond between two CP radicals in the four-atomic molecule PCCP.

6.1 Open Shell ASA

The basic PyFrag 2019 input for the Activation Strain Analysis (ASA) using ADF to perform an open shell Activation Strain Analysis is as follow:

```
JOBSUB

#!/bin/bash
#SBATCH -J NNC
#SBATCH -N 1
#SBATCH -t 24:00:00
#SBATCH --ntasks-per-node=24
#SBATCH --partition=normal
#SBATCH --output=%job.stdout
#SBATCH --error=%job.stdout
export NSCM=24

JOBSUB END


PyFrag

ircpath /Users/xiaobo/Desktop/test/molecule.xyz
fragment  1 3 4 5
fragment  2 6 7 8
strain    0
strain    0
bondlength 1 2  1.52

PyFrag END

fragment1 EXTRA

SYMMETRY C(3V)
CHARGE    0 0
```

(continues on next page)

(continued from previous page)

```
OCCUPATIONS
E1 4
A1 5
END

fragment1 EXTRA END

fragment2 EXTRA

SYMMETRY C(3V)
CHARGE    0 0

OCCUPATIONS
E1 4
A1 5
END

fragment2 EXTRA END

complex EXTRA

FRAGOCCUPATIONS

frag1
E1 2//2
A1 3//2
SUBEND

frag2
E1 2//2
A1 2//3
SUBEND

END

complex EXTRA END

fragment1 open EXTRA
charge 0 1
unrestricted
fragment1 open EXTRA END

fragment2 open EXTRA
charge 0 1
unrestricted
fragment2 open EXTRA END

complex open EXTRA
charge 0 0
complex open EXTRA END
```

(continues on next page)

(continued from previous page)

```
ADF

basis
type DZ
core None
end

xc
gga OPBE
end


scf
iterations 99
converge 0.0001
mixing 0.20
end

numericalquality good


ADF END
```

To submit a job, create a directory and generate a input file and run the following command to submit a job:

```
pyfrag -x open job.in
```

In order to perform a successful open shell fragment analysis, additional information should be provided in the following input blocks:

```
fragment1 EXTRA

SYMMETRY C(3V)
CHARGE    0 0

OCCUPATIONS
E1 4
A1 5
END

fragment1 EXTRA END

fragment2 EXTRA

SYMMETRY C(3V)
CHARGE    0 0

OCCUPATIONS
E1 4
A1 5
END
```

(continues on next page)

(continued from previous page)

```
fragment2 EXTRA END

complex EXTRA

FRAGOCCUPATIONS

frag1
E1 2//2
A1 3//2
SUBEND

frag2
E1 2//2
A1 2//3
SUBEND

END
complex EXTRA END
```

The fragment calculations used to provide the TAPE21 for the overall complex calculation must be done, for technical reasons, in the restricted mode. The proper spins are then specified in the calculation of the overall molecule using the FragOccupations key. Noted a proper decomposition of an electron-pair bond energy requires specifying opposite spins for the unpaired electrons of the respective radical fragments, which can be done with the input key FragOccupations. For the convenience of the analysis, it is suggested to specify the electronic configuration according to the symmtry of the molecule.

Please note that if one neglects explicitly specifying opposite spins for the unpaired electrons of the fragments, each of them is treated as being half an alpha and half a beta electron and consequently, they enter into a spurious Pauli repulsive interaction. This results, among others, into the Pauli repulsion term being too repulsive and the orbital interaction term being too much stabilizing.

Note that this implies a slight approximation because the bond energy computed in this way refers to the energy difference between complex and two fragment radicals that are described by orbitals from a spin-restricted SCF calculation, which have been given an unrestricted occupation. In other words, the set of alpha- and beta-spin orbitals are identical and the effect of spin polarization is missing. In practice, this leads to minor energy differences with respect to the correct bond energy, that is, the energy difference between complex and two fragment radicals treated in the unrestricted mode, i.e., for which the set of alpha- and beta-spin orbitals are allowed to relax toward different solutions in the SCF procedure.

This correction term can be computed directly by carrying out an unrestricted computation of the fragment radical using the following block:

```
fragment1 open EXTRA
charge 0 1
unrestricted
fragment1 open EXTRA END

fragment2 open EXTRA
charge 0 1
unrestricted
fragment2 open EXTRA END

complex open EXTRA
charge 0 0
```

(continues on next page)

(continued from previous page)

```
complex open EXTRA END
```

After the calculation, all results will be summarized in two text files. One file with the name started with pyfrag1 include all terms obtained from the above open shell ASA.

The second file with the name started with pyfrag2 include the correction energy terms from the correction procedure later.

6.2 New Open Shell ASA (Since ADF 2019)

Since ADF 2019, new method to do open-shell fragment analysis has been included. For details, please refer to the ADF website.

Based on this method, a new module to do the activation strain analysis has been developed by Xiaobo Sun and Eva Blokker. The specification for the print options is similar with the previous one, except to has to specify the spin state of orbital, such as 1_A, which means spin-A orbital 1. All the following options are acceptable:

```
overlap frag1 HOMO frag2 HOMO
overlap A1 frag1 3_B S frag2 1_B

orbitalenergy frag1 HOMO-2
orbitalenergy frag1 HOMO-1
orbitalenergy frag1 LUMO
orbitalenergy frag2 LUMO
orbitalenergy A1 frag1 3_A

population frag1 HOMO
population frag2 HOMO
population frag2 LUMO
population A1 frag1 3_A
population S frag2 1_B
```

The basic PyFrag 2019 input for the Activation Strain Analysis (ASA) using ADF 2019 to perform a new open shell Activation Strain Analysis is as follow:

```
JOBSUB
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16
#SBATCH --partition=tc
#SBATCH --time=24:00:00
#SBATCH --job-name=methane
#SBATCH --output=methane.out
#SBATCH --error=methane.err
module load adf/2019.301
JOBSUB END

ADF
XC
  GGA BLYP
  DISPERSION Grimme3 BJDAMP
END
```

(continues on next page)

(continued from previous page)

```
NumericalQuality Excellent

BASIS
  TYPE TZ2P
  CORE None
END

SCF
  ITERATIONS 300
END

SYMMETRY AUTO
CHARGE 0
ADF END

PyFrag
ircpath /home/x2sun/methane.amv
fragment  1 2 3 4
fragment  5
strain    0
strain    0
bondlength 1 5
overlap A1 frag1 3_A S frag2 1_A
overlap A1 frag1 3_B S frag2 1_B
overlap A1 frag1 2_B S frag2 1_B
population frag1 HOMO
population frag2 HOMO
PyFrag END


fragment1 EXTRA
SYMMETRY C(3V)
CHARGE 0 1
unrestricted

IrrepOccupations
E1 2//2
A1 3//2
END
fragment1 EXTRA END


fragment2 EXTRA
SYMMETRY AUTO
CHARGE 0 -1
Unrestricted

IrrepOccupations
S 0//1
END
fragment2 EXTRA END
```

(continues on next page)

(continued from previous page)

```
complex EXTRA
UnrestrictedFragments
unrestricted
complex EXTRA END
```

The molecule is methane:

```
C      -0.88533700      -1.60854000       0.00000000
H      -0.50220300      -2.11092900       0.89352900
H      -0.50220300      -2.11092900      -0.89352900
H      -1.97897100      -1.64799500       0.00000000
H      -0.55799500      -0.56431300       0.00000000
```

To submit a job, create a directory and generate a input file and run the following command to submit a job:

```
pyfrag -x newopen job.in
```

6.3 Open Shell ASA Orbital Energy

Because the above open shell Activation Strain Analysis will not give the correct orbital energy of fragment, thus, in order to extract the correct orbital energy, the following small calculation can be performed:

```
PyFrag

ircpath /Users/xiaobo/Desktop/test/plams.0001
fragment  frag1open
orbitalenergy  HOMO
orbitalenergy  HOMO-1
orbitalenergy  LUMO
orbitalenergy  LUMO+1
orbitalenergy  AA 5

PyFrag END
```

The ircpath refer to the plams directory that contains all the open shell calculation results. Besides, the fragment term specifies from which (fragment1open or fragment1open) orbital energy will be extracted. Noted only one fragment informatin can be readed for one calculation.

To submit a job, create a directory and generate a input file and run the following command to run a job:

```
pyfrag -x openorb job.in
```

6.4 Single Points

The basic PyFrag 2019 input for the Activation Strain Analysis (ASA) using ADF to do single point calculation for a series of coordinates is as follows:

```
JOBSUB

#!/bin/bash
#SBATCH -J NNC
#SBATCH -N 1
#SBATCH -t 1:00:00
#SBATCH --ntasks-per-node=24
#SBATCH --partition=short
#SBATCH --output=%job.stdout
#SBATCH --error=%job.stdout
export NSCM=24

JOBSUB END

PyFrag

ircpath /Users/xiaobo/Desktop/test1/molecule.xyz

VDD 1 2 3
angle 1 2 3 90
bondlength 1 2 5

PyFrag END


ADF

basis
type DZ
core None
end

xc
gga OPBE
end

scf
iterations 99
converge 0.0001
mixing 0.20
end

numericalquality good

ADF END
```

Note that the fragment definations are not needed. This functionality provide an easy way to do a simple single point calculation for a series of different molecular coordinates and get the computational results like VDD charges, total

energy, bond length and angles. Use the following command to run this calculation:

```
pyfrag -x single job.in
```

CHAPTER

SEVEN

FURTHER READING

7.1 Whole Time Monitor

The user can monitor the entire calculation process by using this the following command:

```
pyfrag -x consist job.in
```

In this mode, periodically (default set is 20 seconds), new data will be collected and updated in the form of webpage. In the meantime, if the original input is changed, a window will pop up to ask the user if they want to resubmit the job. If the user agrees to restart the job, a new input file will be submitted and started again and all other previous data will remain unchanged.

In this case, you need specify the time interval to check the new result in /pyfrag/.pyfragrc

```
export JOBCHECK="20"
```

and in the /pyfrag/util/configure.py

```
RESULTCHECK="20"
```

CHAPTER

EIGHT

FURTHER INFORMATION

8.1 History of PyFrag

PyFrag 2008

The original version of PyFrag 2008 was developed by Willem-Jan van Zeist, Lando P. Wolters, F. Matthias Bickelhaupt, and Célia Fonseca Guerra at the Theoretical Chemistry Department at the Vrije Universiteit Amsterdam. PyFrag is mainly used to enable a user-friendly analysis of reaction paths in terms of the Extended Activation Strain model of chemical reactivity(ASM). The explanation and application can be found in the references below. For users still using this version, additional useful information can be found on here or here for a more concise version. This version is no longer maintained.

PyFrag 2016

The PyFrag 2016 program was rewritten by Xiaobo Sun and Thomas Soini using the PLAMS library in the ADF package and has been included in the script collection in ADF 2017 and later version. Compared to the old PyFrag, the new version is more compact and easy to be maintained, expanded and upgraded. Also, due to its high compatibility with other python library tools developed by SCM, such as, PLAMS and QMworks, it can be used as a module in line with these computational chemistry job management tools to streamline a large flow of job. For this version, description can be found using this site.

PyFrag 2019

The PyFrag 2019 program was specially designed to facilitate the analysis of reaction mechanism in a more efficient and user-friendly way. PyFrag 2019 has automated and reduced the time-consuming and laborious task of setting up, running, analyzing, and visualizing computational data from reaction mechanism studies to a single job. PyFrag 2019 resolves three main challenges associated with the automatized computational exploration of reaction mechanisms: 1) the management of multiple parallel calculations to automatically find a reaction path; 2) the monitoring of the entire computational process along with the extraction and plotting of relevant information from large amounts of data; and 3) the analysis and presentation of these data in a clear and informative way. The activation strain and canonical energy decomposition results that are generated, relate the characteristics of the reaction profile in terms of intrinsic properties (strain, interaction, orbital overlaps, orbital energies, populations) of the reactant species.

8.2 Activation Strain Model

For more information on the Activation Strain Model (ASM) of chemical reactivity, the user is directed to the references provided below. An easy exercise for activation strain analysis of reaction mechanism using ADF is also included.

Literature


```
1 W.-J. van Zeist, C. Fonseca Guerra, F. M. Bickelhaupt, J. Comput. Chem. 2008, 29, 312-
↪315.
2 I. Fernandez, F. M. Bickelhaupt, Chem. Soc. Rev. 2014, 43, 4953-4967.
3 L. P. Wolters, F. M. Bickelhaupt, WIRES Comput. Mol. Sci. 2015, 5, 324-343.
4 F. M. Bickelhaupt, K. N. Houk Angew. Chem. 2017, 129, 10204-10221; Angew. Chem. Int.␣
↪Ed. 2017, 56, 10070-10086.
```

CHAPTER

NINE

CODE STRUCTURE

For the advanced users who are interested and want to contribute to the code,

please check Code Structure

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search