CDE

0.1

Generated by Doxygen 1.8.17

Chapter 1

CDE: Chemical discovery engine

CDE is a set of fortran routines which implement several different calculation types associated with chemical reaction-path analysis.

This is a legacy version of our reaction discovery code - an updated python version is in progress.

CDE can perform several types of calculations, including:

- Single-ended Graph-driven sampling for chemical reactions (CURRENTLY UNAVAILABLE),
- Double-ended mechanism searching,
- Generation of initial approximate MEPs connecting user-defined reactant and product structures using a variety of methods, including linear interpolation and image-dependent pair potential (IDPP) method.
- Nudged elastic band calculations for refinement of chemical reaction paths.

CDE interfaces with several external programs to perform energy evaluations and geometry optimization, including:

- ORCA
- Psi-4
- LAMMPS
- DFTB+
- Molpro

See the **references** section for further references to methods employed.

Chapter 2

Setting up CDE

2.1 Organization of CDE

The main CDE directory (referred to hereafter as ∼/cde) has a series of subdirectories containing the following:

- ./src: contains the main source files
- ./docs: contains the documentation (generated using doxygen)
- ./bin: a directory where the cde.x executable is stored
- ./examples: a directory containing the CDE tutorials
- ./utils: a directory containining a few (possibly) useful plotting and analysis scripts, mostly written in python and using matplotlib.

2.2 Compiling CDE

CDE is generally written in pretty standard Fortran90 throughout - it should be relatively easy to compile, and does not require any fancy libraries beyond LAPACK.

To compile CDE, just do the following:

- Go to the *src* directory.
- Edit the *Makefile* so that your Fortran compiler and LAPACK libraries are picked up. You can also edit the *Makefile* (if you want) to change the executable name and the location of the compiled executable (the default is ∗∼/cde/bin/cde.x∗).
- Type *make clean*
- Type *make*
- After compilation, you should now have an executable labelled *cde.x* in the specified location (probably ∗∼/cde/bin/).
- At this point, it's a good idea (although not essential) to make sure that either ∗∼/bin/∗ is in your environment's *PATH* variable, or you add an alias to run cde.x. From now on, the rest of the documentation will assume that typing *cde.x* will run the compiled CDE executable.

2.3 Running CDE

To run the code, you'll need some input files. First, you should go and read the sections about the input files that CDE requires.

Once your input files are ready, you simply type:

```
cde.x input
```

where *input* is the name of your input file.

See the sections Annotated input file description and Example input file for CDE for descriptions of input files.

2.4 Adding an alias to your environment

To make things easier, you can add an alias to your environment's configuration file in your home directory. To do so (assuming a linux/unix system using BASH shell):

- Type *cd* to change to your home directory;
- Type *vi .bashrc*
- Add the line

```
alias cde='YYY/cde.x'
```

where *YYY* is the full directory path to the *cde.x* executable.

- Save the ∗.bashrc∗ file, then type

```
source .bashrc
```

- If successful, you should then be able to type *cde.x* to run the CDE code.

Chapter 3

Example input file for CDE

This is an example input file for CDE.

```
# Example input file - note that all possible options are included here, but are NOT required in the input fi

nimage 8
temperature 10
calctype pathfind
startfile start.xyz
endfile end.xyz
pathfile path.xyz
ranseed  1
startfrompath   .FALSE.
pathinit  linear

# path optimization

pathoptmethod cineb
nebmethod quickmin
nebiter 500
cithresh 5d-4
nebspring 0.05
nebstep  10.1
neboutfreq 10
stripinactive .TRUE.
optendsduring .FALSE.
optendsbefore .TRUE.
reconnect .TRUE.
idppguess .TRUE.
projforcetype 3
vsthresh 1d-3
nebconv 1d-3
nebmaxconv 5d-3

#constraints

dofconstraints 6
1 2 3 5 6 9

atomconstraints 0

alignedatoms
1 2 3

#PES input

pesfull .TRUE.
pestype  null
pesfile   orca.head
pesopttype  null
pesoptfile orca.min
```

```
pesexecutable orca
pesoptexecutable orca

# Graph-driven sampling (GDS) control - by default, everything inactive!

movefile moves.in
gdsrestspring 0.1
nbstrength 0.02
nbrange 2.0
kradius 0.1
ngdsiter 500
ngdsrelax 2000
gdsdtrelax 0.05
gdsoutfreq 10

valencerange{
C 1 4
O 1 2
H 0 1
}

reactiveatomtypes{
C
O
H
}

reactiveatoms{
range 1 4
}

reactivevalence{
Fe Fe 6 8
}

fixedbonds{
C O
3 4
}

essentialatoms{
range 1 5
}

essentialmoveatoms{
id 1
}

reactiveatomtypes{
C
H
}

forbidgraphs .TRUE.
forbidfile forbid.in

# Pathfinder calculation
nrxn 15
nmcrxn 2000000
mcrxntemp 1000000.0
graphfunctype 4
nmechmove 1
minmolcharge 0
maxmolcharge 0
nchargemol 0
maxstepcharge 0
maxtotalcharge 0
```

Chapter 4

Annotated input file description

This page describes the allowed input parameters, and a description of what they do. Please note the following:

- Note that input-file keywords are case-sensitive - everything should be lower-case!
- In the input file, lines beginning with '#' are comments.
- Blank lines are ignored.
- Keywords can appear in any order.

4.1 Main input parameters

These parameters are used to control input, number of images, and other general aspects of the calculation.

nimage 10 :: Number of images.

calctype optpath :: Calculation type (*optpath* for path optimization, and *pathfind* for double-ended reaction-path finding calculations.)

startfile start.xyz :: Start-point coordinate file (used when startfrompath = .FALSE.)

endfile end.xyz :: End-point coordinate file (used when startfrompath = .FALSE.)

pathfile path.xyz :: Initial path file (used when startfrompath = .TRUE.)

ranseed 13 :: Random number seed (positive integer)

startfrompath .FALSE. :: Flag indicating start from end-point files only (.FALSE.) or full path (.TRUE.)

pathinit linear :: If startfrompath = .FALSE., pathinit indicates initial interpolation (must be *linear* for now ...)

4.2 Path optimization control

The following parameters are used to control the calculation when performing a path-optimization simulation (**calctype optpath**), for example using climbing-image nudged elastic band.

pathoptmethod cineb :: Path-optimization method (currently cineb only)

nebmethod quickmin :: (CI)NEB optimization method (*fire* for fire algorithm (Bitzek *et al* Phys. Rev. Lett. 97, 170201 (2006)), *quickmin* for velocity verlet method (competitive), *steepest* for steepest-descents (SLOW))

nebiter 500 :: Number of (CI)NEB optimization iterations.

cithresh 5d-4 :: RMS force threshold at which climbing-image is activated.

nebspring 0.05 :: (CI)NEB spring constant in au (Eh/bohr∗∗2)

nebstep 10.1 :: Step-size for steepest descent or quickmin (au)

neboutfreq 10 :: Output frequency during (CI)NEB optimization.

optendsbefore .FALSE. :: Flag indicating whether to optimize end-points BEFORE NEB refinement.

optendsduring .TRUE. :: Flag indicating whether to optimize end-point SIMULTANEOUSLY with path.

nebrestrend .FALSE. :: Flag indicating whether to apply graph-restrain potential during NEB (also used in GDS).

vsthresh 1d-3 :: RMS force threshold at which the variable spring strength is switched on (see Henkelman 00).

reconnect .true. :: For 2->N-1 images, linearly interpolates the coordinates of beads 1 and N, AFTER a minimization has been performed

idppguess .true. :: performs NEB under a Image-Dependent Pair Potential (IDPP) starting from a linear interpolation to generate a better initial guess for NEB. See Article Smidstrup *et al* J. Chem. Phys. 140, 214106 (2016)

projforcetype 2 :: Selects which type of NEB projected force to use (1: original, 2: Henkelman *et al*, J. Chem. Phys. 113, 9901 (2000), 3: Kolsbjerg *et al* J. Chem. Phys. 145, 94107 (2016))

stripinactive .TRUE. :: This is a logical flag which controls whether or not "inactive" molecules are removed from the initial path of a NEB calculation before NEB refinement.

nebconv 1d-3 ::

nebmaxconv 5d-3 ::

4.3 Constraints control

This block controls the constrained atoms and degrees-of-freedom in GDS calculations and geometry-optimization calculations.

The *dofconstraints* input gives the number of fixed DOFs (in this case, 6), with following line in the input file listing the integer-values of the fixed DOFS. Here, counting starts at 1 for the *x*-coordinate of atom 1. The *z*-coordinate of atom 2 has the integer value of 6, and so on...

```
dofconstraints 6
1 2 3 5 6 9
```

The *atomconstraints* has the same format as *dofconstraints*, but the integers now refer to the number of fixed atoms, and the indices of the fixed atoms. In the following, 1 atom is fixed - atom number 13.

```
atomconstraints 1
13
```

Finally, the *alignedatoms* input allows one to define three atoms which will be used to position and orient any input molecular structure in space. This is very useful when performing NEB calculations as it helps remove overall translations and orientations of the molecules in the optimized string. Note that the NEB routines will automatically select 3 atoms to position and orient if you don't input anything as *alignedatoms.* In the following example, atoms 1, 2 and 3 are chosen to define the relative position and orientation.

```
alignedatoms
1 2 3
```

4.4 PES control

This block controls potential energy evaluations during GDS and NEB, and also controls geometry-optimization.

pesfull .TRUE. :: If .TRUE., we perform PES evaluations and geometry optimizations for the full system in one go. If .FALSE., we separately calculate the energy for each independent molecule.

pestype orca :: Defines single-point PES evaluation code to use (*orca*, *dftb*, *null* or *lammps*)

pesfile orca.head :: Defines the inputfile containing the header for pes single-point calculation.

pesopttype orca :: Defines geometry optimization PES code being used (*orca*, *dftb*, *null*, *uff*)

pesoptfile orca.min :: Defines the inputfile containing the header for geometry-optimization calculation.

pesexecutable orca :: Defines the executable to use to for single-point PES evaluation (full path or alias)

pesoptexecutable orca :: Defines the executable to use to for geometry optimization (full path or alias)

4.5 Graph-restraint potential control

Controls GDS simulations.

movefile moves.in :: Identifies the movefile used to input allowed graph moves.

gdsspring 0.025 :: Inter-bead spring constant for GRP (Eh/Bohr∗∗2)

gdsrestspring 0.1 :: Spring constant for harmonic graph enforcement terms (Eh/Bohr∗∗2)

nbstrength 0.02 :: Repulsion strength for non-bonded atoms [V = nbstrength ∗ exp(-r_{ij}∗∗2 / (2 ∗ nbrange∗∗2))]

nbrange 2.0 :: Range parameter for non-bonded atoms (see above).

kradius 0.1 :: Spring constant for non-interacting molecules in GDS simulations.

ngdsrelax 2000 :: Number of steepest-descent optimization steps to minimize graph restraint potential

gdsdtrelax 0.05 :: Step-size (au) for steepest-descent relaxation above.

gdsoutfreq 10 :: Information output frequency during GDS simulation.

nebrestrend .FALSE. :: Flag indicating weather to apply to it duing the optimization of molecules during GDS, if optaftermove is .true. (also used in CINEB).

optaftermove .TRUE. :: Optimizes every molecule involved in the path formation - if the resulting optimised molecule does not conform to the graph, the proposed path is rejected. If nebrestrend .TRUE., the graph constraining potential are also included during the optimization, and then switched off during a second, in the hope to find a minima in agreement with the graphs (would recommend to set nebrestrend .TRUE. for this).

valencerange :: Defines the allowed valence ranges of each element as follows:

```
valencerange{
C 1 4
O 1 2
H 0 1
Pt 0 3 fz
}
```

The fz keyword applies only to frozen atoms, either by being defined in the fixedbonds{} input (see below), or by atomcontraints keyword. In the example, only Pt-X bonds (X= whatever) which can be formed or broken during the GDS calculation are counted into the 'valence'. If a cluster of Pt atoms are not moving during a GDS calculation, and they are bonded to each other (to varying degrees), the above keyword will allow for up to 3 more bonds, not counting the Pt-Pt bonds, to be formed on the Pt atoms.

reactiveatomtypes :: Defines which types of elements are allowed to react, as follows:

```
reactiveatomtypes{
C
O
H
}
```

reactiveatoms :: Defines which atom numbers are allowed to react. These can be given as an inclusive range (*range XX YY*), or by individual id number (*id XX*).

```
reactiveatoms{
range 1 4
id 7
}
```

reactivevalence :: Defines the valence range one element can have with another

```
reactivevalence{
Fe Fe 6 8
    H C 0 1
    H O 0 1
}
```

In the above, *Fe* must be bonded between 6 to 8 other Fe atoms to be an acceptable molecule. Similarly, the second line indicates that Hydrogen can only be bonded up to 1 carbon atom.

fixedbonds :: Defines fixed bonds, which are not allowed to change during a chemical reaction (although they can vibrate, translate, etc.)

```
fixedbonds{
C O
3 4
}
```

In the above example, *all* C-O bond orders are fixed at whatever they are in the starting structure. So, if a C-O bond is present in the starting structure, it cannot change during the GDS simulation. Similarly, the bond between atoms *3* and *4* is also fixed at the starting structure value.

essentialmoveatoms :: Defines a list of atoms of which AT LEAST ONE OF THEM MUST be included in any possible graph moves. The formatting options are the same as the *reactiveatoms* block described above, for example:

```
essentialmoveatoms{
range 1 2
}
```

In the above example, the atoms in the range 1-2 must be included amogst the moves chosen.

essentialatoms :: Defines atoms of which AT LEAST ONE OF THEM MUST be included in molecules that are involved in the reactions. The formatting options are the same as the *reactiveatoms* block described above, for example:

```
essentialatoms{
id 5
id 7
}
```

In the above example, the atoms 5 and 7 must be in any of the molecules that are involved in the reaction (from chemical species image 1 changing to nimage)

allowedbonds :: Allows definition of bond-number constraints in the structures generated by graph moves. The format is as given in the following example:

```
allowedbonds{
O O 1
}
```

Here, the constraint indicates that an oxygen atom cannot be bonded to more than 1 other oxygen atom.

forbidgraphs :: This is a logical flag which indicates whether or not to use the forbidden graph pattern file. If .TRUE., then a GDS simulation will not allow any graph-patterns to form which are input into the forbidfile defined below. By adding entries to the forbid file, users can force GDS to stop generating user-defined bonding patterns.

forbidfile forbid.in :: Identifies the file (here, forbid.in) which contains the library of forbidden bonding patterns which are used if forbidgraphs = .TRUE.

4.6 Path-finding controls

The following parameters control the double-ended reaction-path finding algorithm.

nrxn :: Total number of elementary steps allowed in mechanism.

nmcrxn :: Maximum number of Monte Carlo moves to try during simulated annealing optimization.

nmechmove :: Maximum number of mechanism updates in each MC step.

mcrxntemp :: Initial temperature (in K) for simulated annealing optimization (decreases linearly during run).

graphfunctype :: Determines the optimization function for the path-finding calculation. This should be an integer↩
: (1) Standard element-wise comparison, (2) Eigenvalue comparison, (3) Histogram comparison, (4) Eigenvalue comparison for single molecule. *You should preferably use either (2) or (4)!*

minmolcharge :: Minimum allowed molecular charge (if electron transfer moves are allowed in the movefile).

maxmolcharge :: As above, but maximum value.

nchargemol :: Maximum number of molecules which can be charged.

maxstepcharge :: Maximum number of reation steps which can involve charge changes.

maxtotalcharge :: Maximum total charge in a given reaction step.

Chapter 5

I/O structure formats

Some important things to note about structure formats used in CDE:

- Throughout CDE, we generally use simple $*$.xyz files as input/output formats.
- You can use *openbabel* (`http://openbabel.org/wiki/Main_Page`) to convert from a wide range of files to/from xyz files.
- A convenient program for creating initial structures in xyz format is *Avogadro* (`https://avogadro.cc`).
- The coordinates in an XYZ format file in units of Angstroms.
- If performing NEB calculations, the number of snapshots (or configurations) in the xyz file for the full path should be the same as that defined in *nimage* in the input file.
- Each xyz frame or file looks like this:

```
2

C 0.000   0.000 0.0000
O 1.50    1.670 5.6708
```

- The first line contains the number of atoms, the second line is blank (or a comment line) and then there is a list of atoms with their atomic labels and (x,y,z) coordinates in Angstroms.
- Different frames in an xyz file follow each other immediately, like this:

```
2

C 0.000   0.000 0.0000
O 1.50    1.670 5.6708
2

C 0.1500  0.2166 0.007
O 1.56  1.679  5.7098
```

Chapter 6

PES evaluations

CDE can call several external programs to obtain energies and forces on atoms during GDS and NEB runs. This section describes how this is managed, and how input files can be set-up to run different external codes.

6.1 PES types

- The potential energy surface (PES) type to be used during a CDE-based calculation is indicated through the variables **pestype** and **pesopttype**.
- The **pestype** is used whenever the code requires a single-point energy evaluation.
- The current allowed values of **pestype** are (see pes.f90):
  - *orca*: Performs a calculation using the quantum chemical program ORCA (`https://orcaforum.cec.mpg.de`)
  - *dftb*: Performs a calculation using the density-functional tight-binding code DFTB+ (`https://www.dftbplus.org`)
  - *psi4*: Performs a calculation using the psi-4 quantum code (`http://www.psicode.org`)
  - *lammps*: Performs a calculation of the PES using the LAMMPS simulation package (`https://lammps.sandia.gov`)
  - *null*: Does not perform a PES evaluation. Simply returns V = 0 and forces(:) = 0 for all atoms.
- The **pesopttype** is used whenever the code requires a geometry optimization.
- The allowed values of **pesopttype** are:
  - *orca*: ORCA calculation.
  - *dftb*: DFTB+ calculation.
  - *psi4*: Psi-4 calculation.
  - *lammps*: LAMMPS calculation.
  - *uff*: Performs geometry optimization under the UFF (Universal force-field) as implemented in babel/openbabel (`http://openbabel.org/wiki/Main_Page`)
  - *null*: Does not perform geometry optimization.
- To add new PES evaluation types, you need to edit the files pes.f90 and io.f90; **DO NOT DO THIS UNLESS YOU ARE SURE YOU KNOW WHAT YOU ARE DOING!**

6.2 PES and geometry optimization executables

As well as knowing which type of PES calculation you'd like, the CDE code needs to know *how* to run the external executables which are requested to calculate potential energies and forces, or to run geometry optimizations.

The executables to be used to energy evaluations or geometry optimization are defined in the variables *pesexecutable* and *pesoptexecutable*, as follows:

```
pesexecutable ~/programs/orca
pesoptexecutable ~/programs/orca
```

In the example above, we're are indicating that the external program to be used is *orca*, and this is stored in the directory ∗∼/programs/∗.

Internally, CDE runs FORTRAN *EXECUTE_COMMAND_LINE* calls to run external codes, using input files generated according to templates provided as *pesfile* and *pesoptfile* (see below).

For example, based on the above, CDE will run the following command when an *ORCA* calculation is required:

```
~/programs/orca temp.in
```

where *temp.in* is the name of a temporary input file which is auto-generated by CDE during NEB calculations.

If there is no executable called "∼/programs/orca", the CDE calculation will fail!

To make sure everything runs smoothly, there are two useful options:

(1) You can set up an alias for each executable, then provide the alias as *pesexecutable* and *pesoptexecutable*. For example, lets say you have *ORCA* installed in ∗∼/programs/stuff/orca/bin/∗. If you include an alias in your ∗.bashrc∗ (or similar config file if you're on a different system) which reads

```
alias orca='~/programs/stuff/orca/bin/orca'
```

then running the command *orca* will then execute ∗∼/programs/stuff/orca/bin/orca∗, which should indeed correspond to an executable binary.

This means that you can then use the following in your CDE input files:

```
pesexecutable orca
pesoptexecutable orca
```

without having to provide the full path to the executables.

(2) As an alternative, you can also give the full pathname to the desired executable directly in the input file, lik this:

```
  pesexecutable ~/programs/stuff/orca/bin/orca
  pesoptexecutable ~/programs/stuff/orca/bin/orca
```

6.3 PES template files

- The final pieces of information which CDE requires in order to perform energy evaluations or geometry optimization is a PES *template file*.
- The input parameters **pesfile** and **pesoptfile** are header files which are used to direct external codes which calculation to perform when they are called by CDE. These header files must have the same format as input files required by the *pesexecutable* and *pesoptexecutable*.
- The only difference is that there must be a line with *XXX* as follows:

```
XXX
```

- The *XXX* tells the CDE code where it should insert the coordinates of the configuration at which the energy evaluation is requested. After inserting the coordinates, CDE writes the input file then runs the relevant external executable to evaluate the energy. Once the energy evaluation (or geometry optimization) is complete, CDE reads in the results.
- The file pes.f90 can be consulted to see how this write/run/read process works for each external code.
- An example header file for a PES evaluation by ORCA using PM3 semi-empirical method might be as follows:

```
! PM3
* xyz 0 1
XXX
*
```

- An example header file for PES evaluation by DFTB+ might be as follows:

```
 Geometry = GenFormat {
XXX
}
Driver = {}
 Hamiltonian = DFTB {
 SCC = Yes
 SCCTolerance = 1e-3
 Mixer = Broyden{}
 Eigensolver = RelativelyRobust {}
MaxAngularMomentum {
 C = "p"
O = "p"
H = "s"
 }
SlaterKosterFiles = Type2FileNames {
Prefix = "/Users/scott/code/cde/src/SKfiles/"
 Separator = "-"
 Suffix = ".skf"
 }
}
Analysis = {
CalculateForces = Yes
 }
```

- Note in the above examples that the only part of the file which is written by CDE is in replacing the *XXX* with the coordinates at which the energy evaluation is being performed. All other aspects, such as the calculation type (e.g. DFT, Hartree-Fock, semi-empirical, DFTB+), basis sets, accuracy, and so on, are controlled by editing the *pesfile* and *pesoptfile* BEFORE the calculation starts.

Chapter 7

PES templates

This page gives several examples of template files for PES evaluation or geometry evaluation.

In these examples, the only part of the file which is written by CDE is in replacing the *XXX* with the coordinates at which the energy evaluation is being performed. All other aspects, such as the calculation type (e.g. DFT, Hartree-Fock, semi-empirical, DFTB+), basis sets, accuracy, and so on, are controlled by editing the *pesfile* and *pesoptfile* BEFORE the CDE calculation starts.

Note that a single PES template file is used for all energy evaluations in CDE. There is no option to change the PES evaluation-type "on the fly".

7.1 ORCA examples

- Example header file for a PES evaluation by ORCA using PM3 semi-empirical method:

```
! PM3
* xyz 0 1
XXX
*
```

- Example header file for a PES evaluation by ORCA using DFT (B3LYP):

```
! DFT B3LYP aug-cc-pvtz
    * xyz 0 1
    XXX
    *
```

- Example header file for geometry optimization using PM3:

```
! PM3 OPT
* xyz 0 1
XXX
*
```

7.2 DFTB+ example

In the case of DFTB+, note that the input file must correctly specify the location of any Slater-Koster files required by the DFTB+ code.

- Example header file for PES evaluation by DFTB+:

```
 Geometry = GenFormat {
XXX
}
Driver = {}
 Hamiltonian = DFTB {
 SCC = Yes
 SCCTolerance = 1e-3
 Mixer = Broyden{}
 Eigensolver = RelativelyRobust {}
MaxAngularMomentum {
 C = "p"
O = "p"
H = "s"
 }
SlaterKosterFiles = Type2FileNames {
Prefix = "./SKfiles/"
 Separator = "-"
 Suffix = ".skf"
 }
}
Analysis = {
CalculateForces = Yes
 }
```

- DFTB+ geometry optimization example (with constraints):

```
Geometry = GenFormat {
XXX
}
 Driver = ConjugateGradient{
 MaxForceComponent = 1d-5
 MaxSteps = 1000
 Constraints = {
 1 1.0 0.0 0.0
 1 0.0 1.0 0.0
 1 0.0 0.0 1.0
 2 0.0 1.0 0.0
 2 0.0 0.0 1.0
 3 0.0 0.0 1.0
 }
 }
 Hamiltonian = DFTB {
SCC = Yes
 SCCTolerance = 1e-3
 Mixer = Broyden{}
 Eigensolver = RelativelyRobust {}
MaxAngularMomentum {
 C = "p"
O = "p"
 H = "s"
 }
 SlaterKosterFiles = Type2FileNames {
 Prefix = "./SKfiles/"
 Separator = "-"
 Suffix = ".skf"
 }
 }
 Analysis = {
 CalculateForces = Yes
 }
```

7.3 Psi-4 example

- Hartree-Fock example using Psi-4

```
################
# INPUT SETUP #
################

molecule x {
0 1
XXX
}

set {
    basis def2-SVP
    fail_on_maxiter false
}


################
# OUTPUT SETUP #
################

energy = energy('scf')
grad = np.array(gradient('scf'))

e = open('e.out','w')
e.write(str(energy))
e.close()

f = open('f.out','w')
for i in grad:
    for j in i:
        f.write(str(j)+'\n')
f.close
```

7.4 LAMMPS/ReaxFF example

- The following is an example of a LAMMPS input file (*e.g. lmp.head*) for running a ReaxFF calculation.

```
#dimension      3
units          real
boundary       f f f
atom_style     charge
atom_modify    map array sort 0 0.0

# XXX replaced by mass labels and atom creation

XXX

pair_style     reax/c lmp_control
# Force file to be used, types added automatically
pair_coeff     * * CHOPtNiX.ff
fix            1 all qeq/reax 1 0.0 10.0 1.0e-8 reax/c  # new addition

neighbor       1.0 bin
neigh_modify   every 1 delay 0 check no

compute        reax all pair reax/c

dump           1 all custom 1 temp.force fx fy fz
dump_modify    1 format float "%14.8f"

thermo         0
thermo_style   custom pe

run            0
```

- The following is an example of a geometry optimization calculation performed using ReaxFF through LAM↩ MPS.

```
# Intialization
dimension      3
units          real
boundary       f f f
atom_style     charge
atom_modify    map array sort 0 0.0

# XXX replaced by mass labels and atom creation
XXX

pair_style     reax/c lmp_control
# Force file to be used, atom types are added automatically for reax/c
pair_coeff     * * CHOPtNiX.ff
fix            1 all qeq/reax 1 0.0 10.0 1.0e-8 reax/c

neighbor       2.5 bin
neigh_modify   every 1 delay 0 check no

compute        reax all pair reax/c

thermo         1
thermo_style   custom pe

min_style quickmin # new addition
 minimize 1.00e-10 1.00e-8 50000 50000

timestep 1

# output forces
dump           2 all custom 1 temp.force fx fy fz
dump_modify    2 format float "%14.8f"
```

- Both of the files above also require two additional files to be present in the run directory. The first of these, *lmp_control* is an additional LAMMPS control file, which usually looks like the following:

```
nbrhood_cutoff          5.0  ! near neighbors cutoff for bond calculations in A
hbond_cutoff            6.0  ! cutoff distance for hydrogen bond interactions
bond_graph_cutoff       0.3  ! bond strength cutoff for bond graphs
thb_cutoff              0.001 ! cutoff value for three body interactions
atom_forces 1
```

- The second additional file required is the ReaxFF parameter file. In this case, it is referred to as *CHOPtNiX.ff*, but this could be changed depending on the system under investigation and the parameters available. An example of these *ff* files can be found in *Tutorial 2* (Tutorial 2: CINEB using LAMMPS).

Chapter 8

Forbid-file example

The forbid-file is used in GDS to define which graph-patterns cannot be generated as new reaction-path end-points during each GDS graph-change step.

A typical forbid-file (usually called *forbid.in*, although can be called anything) looks like this:

```
forbid
natom 3
-
0 1 0
1 0 1
0 1 0
-
labels C O O

forbid
natom 3
-
0 1 0
1 0 1
0 1 0
-
labels C O C
```

Note that the format is very similar to that of the moves files.

Each forbidden graph-patter is defined in a block, which looks like this.

```
forbid
natom 3
-
0 1 0
1 0 1
    0 1 0
-
labels C O O
```

In the example above, the bonding pattern specified by the graph (e.g. bond between atoms 1 and 2, bond between atoms 2 and 3) is NOT allowed to be generated for the atom-labels C,O,O. In othert words, this pattern would stop C-O-O, and any molecules containing this pattern, from forming.

Note that, in contrast to the move-file, the atom labels **must** be defined in the forbid-file.

Chapter 9

Movefile example

The move-file is used in GDS to define which graph-moves can be used to generate new reaction-path end-points during each GDS graph-change step.

A typical move-file (usually called *moves.in*, although can be called anything) looks like this:

```
move
natom 2
-
0 1
1 0
-
0 0
0 0
-
labels * *
prob 0.4

move
natom 2
-
0 0
0 0
-
0 1
1 0
-
labels * *
prob 0.4

move
natom 3
-
0 1 0
1 0 1
0 1 0
-
0 0 1
0 0 1
1 1 0
-
labels * * *
prob 0.1

move
natom 3
-
0 1 0
1 0 1
0 1 0
```

```
-
0 0 1
0 0 0
1 0 0
-
labels * * *
prob 0.1
```

Each move is defined in a block, which looks like this.

```
move
natom 2
-
0 1
1 0
-
0 0
0 0
-
labels * *
prob 0.4
```

In the example above, the proposed graphmove involves two atoms; these are randomly selected in the code. The first 2x2 matrix is the required bonding pattern of the two atoms; in this case, the moves requires that the two atoms are bonded (*i.e.* there is a 1, indicating bonding, on the off-diagonal element). The second 2x2 matrix indicates the target final graph; in thise case, there is a *0* on the off-diagonal, indicating that there is no bond between the two atoms in the final structure. Overall, therefore, this is a simple bond-breaking reaction.

The *labels* line indicates the required atom labels for this move; in this case, we have indicated " $*$ ", which means that **any** atom can participate. An alternative would be

```
labels C O
```

which would imply that only C and O can be involved for this graph move.

The *prob* value gives the relative probability of this move taking place during a given GDS run; not that the probabilities of all moves are all normalised in the code (*i.e.* they are re-scaled so that they sum to 1), so they do not have to add to 1 in the movefile.

A second example, this time involving 3 atoms, is as follows:

```
move
natom 3
-
0 1 0
1 0 1
0 1 0
-
0 0 1
0 0 0
1 0 0
-
labels * * *
prob 0.1
```

The above reaction is A-B-C --> A-C + B.

Chapter 10

Hints and tips

10.0.1 Hints on input files.

- The main CDE input file is CASE SENSITIVE. *Everything should be lower-case, except for chemical element symbols.*
- If you are running a calculation which does not use some of the input parameters, they will be read in but then ignored. However, their formatting still needs to be correct.

10.0.2 Hints on bonding definitions.

- In CDE, bonds between atoms are based on a cutoff distance given by *R = (CovRad(i) + CovRad(j))* $*$ *bondingsf*. The factors CovRad(:) are the covalent radii of each species, and can be found in *constants.f90*. The factor **bondingsf** is also found in constants.f90 - it is a scale-factor, usually with a value of 1.1.
- The CovRad(:) and bondingsf are defined as Fortran constants - if you want to change them, you need to recompile the code.
- The Covrad(:) and bondingsf values do not need to be extremely accurate - they are used to define what is bonded to what, so as long as they adequately capture typical bonds, they should be fine.

10.0.3 Hints on external executables

- The input parameters **pesexecutable** and **pesoptexecutable** define the external executables which are run when evaluating energy or performing geometry optimizations.
- The CDE code uses the Fortran implicit *EXECUTE_COMMAND_LINE* to run these executables.
- It is often easiest to define an alias for executables like ORCA and DFTB+. For example, if the ORCA executable is in a directory $*$/user/test/code/orca/bin$*$, then setting an alias so that the command *orca* actually executes the command $*$/user/test/code/orca/bin/orca$*$ is a useful way of simplifying the input file.
- If the executables can't be found when run by *EXECUTE_COMMAND_LINE*, the code will just give up and crash!
- If you want to use **pesopttype UFF** (that is, the Universal Force-Field), you need to install *OpenBabel*.
- When using **pesopttype UFF**, the executables *babel* and *obminimize* are called directly. As a result, both of these executables need to be in your PATH variable before running CDE.

10.0.4 Common problems

- In several types of calculations (e.g. double-ended reaction-path finding, NEB), the first step that the C↩
DE code performs is to evaluate the connectivity matrix of the input molecular structures. Depending on the calculation type, this connectivity matrix is then used to define the graph-restraining potential (GRP) for structure optimization and checking. However, if the connectivity matrix calculated using the input structure does not correspond to the connectivity matrix that you wanted (for example, due to bond-lengths being too long in the input sutrcture), then you might end up with some odd results. **In short, TAKE CARE with your input structures!**

Chapter 11

Things to do

Features requested as of July 2021:

- Connect to kinetic Monte Carlo simulations.
- Remove overall translation and rotation from NEB calculations.

Chapter 12

References

12.1 Graph-driven sampling methodology

These papers describe the core principles and ideas of the main reaction-discovery methods implemented in CDE.

- Sampling reactive pathways with random walks in chemical space: Applications to molecular dissociation and catalysis, S Habershon, *Journal of Chemical Physics*, **143**, 094106 (2015)
- Automated prediction of catalytic mechanism and rate law using graph-based reaction path sampling, S Habershon, *Journal of chemical theory and computation*, **12**, 1786 (2016)
- Automatic Proposal of Multistep Reaction Mechanisms using a Graph-Driven Search, I. Ismail, H. B. V. A. Stuttaford-Fowler, C. Ochan Ashok, C. Robertson and S. Habershon, *J. Phys. Chem. A*, **123**, 3407 (2019)
- Fast screening of homogeneous catalysis mechanisms using graph-driven searches and approximate quantum chemistry, C. Robertson and S. Habershon, *Cat. Sci. Tech.*, dx.doi.org/10.1039/C9CY01997A (in press, 2019)

12.2 Minimum energy path refinement.

These papers describe the MEP finding algorithms which are implemented in CDE.

- Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points, G. Henkelman and H. Jonsson, *Journal of Chemical Physics*, **113**, 9978 (2000)
- A climbing image nudged elastic band method for finding saddle points and minimum energy paths, G. Henkelman, B. P. Uberuaga and H. Jonsson, *Journal of Chemical Physics*, **113**, 9901 (2000)
- An automated nudged elastic band method, E. L. Kolsbjerg, M. N. Groves, and B. Hammer *Journal of Chemical Physics*, **145**, 094107 (2016)

12.3 ReaxFF and LAMMPS

- ReaxFF: A Reactive Force Field for Hydrocarbons, A. C. T. van Duin and S. Dasgupta and F. Lorant and W. A. Goddard, *J. Phys. Chem. A*, **105**, 9396-9409 (2001) (https://doi.org/10.1021/jp004368u)
- Information and full documentation on the LAMMPS molecular dynamics code can be found at: https↩ ://lammps.sandia.gov

Chapter 13

Useful scripts

The directory ∗∼/utils/∗ in the main CDE installation directory contains a series of useful *bash* and *python* scripts as follows:

- **CompareProfiles.py**: Uses python and matplotlib to compare two energy profiles resulting from NEB calculations.
- **WholeEnergyProfile.py**: Plots a sequence of NEB results as a single continuous energy profile using python and matplotlib.
- **maker.sc**: A bash script for creating multiple directories with identicial copies of a set of input files, except with different random number seeds.
- **runner.sc**: A bash script to submit multiple jobs to the HPC queue. This can be used after using *maker.sc* to generate the input directories and files.
- **neb.sc**: A bash script to perform multiple NEB runs. This can be used after a double-ended GDS calculation to calculate the energy along the minimum energy path for a whole sequence of reactions. The final result of using *neb.sc* can then be plotted using *WholeEnergyProfile.py*.
- **job_script.sc**: An example queue script for submission of a job to the Warwick COW.

Chapter 14

Tutorial 1: CINEB

This section gives example input files and instructions for a typical CINEB calculation run through CDE.

In this case, we are going to run a CINEB optimization of a reaction path representing dissociation of molecular hydrogen (H2) from formaldehyde (H2CO). We will using the semi-empirical PM3 method *via* the *ab initio* code *ORCA*, to calculate potential energies and forces.

The actual files to run this example can be found in the ∗∼/cde/examples/Tutorial_1 directory.

In this directory, you will find several input files:

```
  input
  initial_path.xyz
  orca.head
  orca.min
```

All of these input filenames are arbitrary; we could have called them alice, bob, clive and ralph and the program would still read them in correctly. Note that the extensions have no effect for input files read in by CDE, but they can be used to remind you what the different input files are.

Let's have a look at each input file individually:

14.0.1 input

The *input* file looks like this:

```
# This input file demonstrates a CINEB refinement.
# We use the quickmin method here, and orca
# for PES calculations.

nimage 8
temperature 10
calctype optpath
startfile start.xyz
endfile end.xyz
pathfile path.xyz
ranseed  5
startfrompath  .FALSE.
pathinit  linear

# path optimization

pathoptmethod cineb
```

```
nebmethod quickmin
nebiter 250
vsthresh 1d-2
cithresh 1d-3
nebspring 0.05
nebconv 2d-3
nebstep  10.0
neboutfreq 5
stripinactive .FALSE.
optendsbefore .TRUE.
optendsduring .false.
nebrestrend .FALSE.
alignedatoms
1 2 3

# constraints

dofconstraints 0
atomconstraints 0

# PES input

pesfull .true.
pestype  orca
pesfile   orca.head
pesopttype  orca
pesoptfile orca.min
pesexecutable orca
pesoptexecutable orca
```

The input file in this case only contains those options which are relevant to the CINEB simulation; other input directives have been removed and will be ignored internally in CDE.

The first input block (beginning with the comment line "# control parameters") identifies the following input parameters:

- **nimage**: Number of images (in total) in the reaction-path which is going to be optimized. In this case, we have 8; note that the number of images indicated here must be the same as in the input pathfile if *startfrompath = .TRUE.* (see below).
- **temperature**: We're going to use the *quickmin* optimization algorithm to refine our path (see below). In *quickmin*, each atom in each string is given a velocity to move it towards the MEP; these initial velocities are drawn from a Boltzmann distribution at the input temperature (in Kelvin) indicated here. For *quickmin* optimization, this temperature should be relatively low.
- **calctype**: Here, we're using *optpath* to indicate that we want to do a path optimization calculation.
- **pathfile**: In nudged elastic band path refinement calculations, we often have a good initial guess of the reaction path we'd like to refine. The *pathfile* variable defines the input file where our initial reaction-path guess is stored. In this case, we're saying that our initial reaction-path is in *path.xyz*; this is an *xyz* format file (with coordinates given in Angstroms) which MUST contain the same number of snapshots as the number *nimage* defined above.

IMPORTANT: In this example, although we're showing how to define a starting *pathfile*, we're note actually going to use it! Instead, we've set *startfrompath .FALSE.*, which means that the initial path is instead going to be generated by linear interpolation of coordinates between the *startfile* and *endfile*.

- **ranseed**: This is an integer random number seed, used to generate random initial velocities drawn from the Boltzmann distribution for *quickmin* (see above)
- **startfile**: This is an xyz file containing the reactant structure for our target system.
- **endfile**: This is an xyz file containing the product structure for out target system. **Note that the *startfile* and *endfile* MUST have the same number of atoms in the same order!**

- **startfrompath**: Here, by inputting ".FALSE.", we're indicating that CDE should instead start by generating its own initial path by linear interpolation between *startfile* and *endfile*.

The second input block (beginning with "# path optimization") defines parameters which control the NEB path refinement calculation:

- **pathoptmethod**: We're setting this to *cineb*, to indicate that we'd like to run a CINEB calculation.
- **nebmethod**: This defines the optimization method we're using to find the minimum-energy path. Options are *quickmin*, which uses a velocity Verlet-based method, or *steepest* for a simple steepest-descent method. The *quickmin* method is preferred - it's usually much faster to converge.
- **nebiter**: This defines the mamximum number of CINEB iterations to perform before stopping. Note that if the calculation reaches a force convergence value lower than *nebconv* (see below), it stops anyway.
- **cithresh**: This is the force threshold at which the climbing-image variant of NEB (see references for details) is activated. In this case, once the current reaction-path forces are less than 0.008 Eh/a0, climbing-image is activated.
- **nebconv**: Target force convergence threshold for CINEB calculation.
- **nebspring**: CINEB sprin parameter (in atomic units, Eh/a0∗∗2).
- **nebstep**: Step-length parameter for optimization. In the case of *quickmin*, this is an effective velocity Verlet time-step taken at each CINEB iteration (in atomic units, so a value of $\sim$41 au corresponds to about a 1 fs time-step). In the case of steepest-descent (*nebmethod steepest*), the *nebstep* parameter indicates the fraction of the current force used to update the atomic coordinates at each step.
- **neboutfreq**: Frequency with which the NEB calculation will output the current reaction path and energy profile. The energy profiles are input to the file ending with ∗.nebprofile∗ and the paths are output to *input↩ _YYYY.xyz*, where *YYYY* is the current NEB iteration.
- **stripinactive**: If set toe ∗.TRUE.∗, then any molecules which are simply acting as "spectators" to the reaction (e.g. they are far away from the reaction sites and do not participate) are "stripped" (removed) from the reaction path BEFORE the NEB calculation. This option can make calculations a bit faster!
- **optendsbefore**: If set to ∗.TRUE.∗, then CDE first performs a geometry optimization of the two end-point structures *before* procedding with CINEB.
- **optendsduring**: If set to ∗.TRUE.∗, the two end-points are optimized under the action of the PES *during* the NEB optimization. Here, the end-points only feel the force due to the PES, and do not feel the NEB springs.
- **nebrestrend**: If *optendsduring = .TRUE.*, this options indicates whether or not to additionally include the graph-restraining potential into the forces felt by the two end-points during NEB optimization. This option can prevent the end-point molecules changing the connectivity during NEB optimization. Note that this option is only important if *optendsduring = .TRUE.*.
- **alignedatoms**: As described in the *annotated input file description* (Annotated input file description), the *alignedatoms* directive tells CDE which triple of atoms should serve as coordinates to align and orient the NEB string in space, to avoid overall translation and rotation of the system during NEB refinement.

The next input block (starting with "# constraints") indicate any constraints we'd like to impose on each image in the reaction-path during NEB optimization.

- **atomconstraints**: In this example, we've indicated that there are no atom constraints (beyond the *alignedatoms* directive noted above).
- **dofconstraints**: Again, in this case, we don't have any DOF constraints other than those noted above.

The final block (starting with "# PES input") defines options to enable potential energy evaluations and geometry optimization during the NEB calculation. These input parameters are discussed elsewhere, such as in *Annotated input file description* (Annotated input file description).

14.0.2 start.xyz and end.xyz

Here, the file *start.xyz* and *end.xyz* files contain the reactant and product structures for the initial reaction-path. These are xyz files, as described in *I/O formats* (I/O structure formats).

14.0.3 orca.head

In the current calculation, we're going to use the *ab initio* code *ORCA* to perform potential energy and force evaluations.

Again, note that we could have called this file anything we wanted to - it doesn't have to be called *orca.head*. However, we must make sure that this is the same filename as defined in the *pesfil* variable in the input file above.

The *orca.head* file looks like this.

```
! PM3 ENGRAD
* xyz 0 1
XXX
*
```

Note that this file is exactly the same format as a usual *ORCA* input file:

- The first line tells *ORCA* to run an *energy and gradient* evaluation using the *PM3* semi-empirical method.
- The second line begins the geometry specification; here, we're indicating that the geometry will be specified as *xyz* Cartesian coordinates (in Angstroms), the total charge on the system is *0*, and the total spin multiplicity is *1* (i.e. closed-shell species).
- In a normal *ORCA* file, we would then input the geometry on the following lines. For example, an *ORCA* input file would look like this:

  ```
  ! PM3 ENGRAD
  * xyz 0 1
  C 0.1500  0.2166 0.007
  O 1.56  1.679  5.7098
  *
  ```

- However, in an input file for CDE, the coordinate block is simply replaced by the string *XXX*. This indicates that this is the point at which *CDE* should insert coordinates for molecules it would like to run through *ORCA*.
- Finally, there is a "$*$" symbol on the last line to indicate that this is the end of the *ORCA* input file.

14.0.4 orca.min

Like the *orca.head* file, the file defined as *pesoptfile* should also be present in the directory.

This file indicates how to run a geometry optimization calculation; note that this calculation is only used if *optendsbefore = .TRUE.*.

As in the case of the files above, the *pesoptfile* can be called anything you'd like!

The format of the *pesoptfile* (here called *orca.min*) is the same as an *ORCA* file for a geometry optimization, but again with the condition that the coordinate input block is replaced by *XXX*. An example is:

```
! PM3 OPT
* xyz 0 1
XXX
*
```

Note that everything in the file above is the normal *ORCA* format; the only difference is the *XXX* line required by CDE.

14.0.5 A note on executables for external programs

Note that our input file above contains two variables defined as:

```
  pesexecutable orca
  pesoptexecutable orca
```

The *pesexecutable* and *pesoptexecutable* variables indicate which code to run to calculate energies and forces; in this case, we're using *ORCA*.

Note that, in each case, the input command is run directly. For example, based on the above, CDE will run the following command when an *ORCA* calculation is required:

```
orca temp.in
```

where *temp.in* is the name of a temporary input file which is auto-generated by CDE during NEB calculations.

If there is no executable called "orca", the CDE calculation will fail!

If you see error messages such as "No such file orca..." or similar, you should give the full path to your executable in the input file.

To make sure everything runs smoothly, there are two options:

(1) You can set up an alias so that, when CDE executes the command *orca* given as *pesexecutable* and *pesoptexecutable*, everything runs as expected. For example, lets say you have *ORCA* installed in ∗∼/programs/stuff/orca/bin/∗. If you include an alias in your ∗.bashrc∗ (or similar config file if you're on a different system) which reads

```
alias orca='~/programs/stuff/orca/bin/orca'
```

then running the command *orca* will then execute ∗∼/programs/stuff/orca/bin/orca∗, which should indeed correspond to an executable binary.

(2) As an alternative, you can also give the full pathname to the desired executable directly in the input file, lik this:

```
  pesexecutable ~/programs/stuff/orca/bin/orca
  pesoptexecutable ~/programs/stuff/orca/bin/orca
```

14.0.6 Running the calculation

With these input files, we are now able to run the CINEB optimization. To do so, go into the ∗∼/cde/examples/cineb/∗ directory and type:

```
cde.x input
```

The above assumes that you have already made sure that CDE can be run by simply typing *cde.x*. See the setup section for more details.

As the calculation runs, you will find lots of output files generated in the run directory. Many of these files will be useless outputs generated by *ORCA*, typically ocntaining wavefunction and integration grid information.

The interesting output files are as follows:

- **input.nebconv**: This file shows the total norm of the forces, averaged over the images. These values are given for calculations performed with and without the NEB spring terms. This file can be monitored during the calculation to find out how close to convergence the NEB calculation is. This file is written after every NEB iteration.
- **input.nebprofile**: This file contains the energy profile along the reaction-string at the different iterations of the NEB calculation; both absolute energy and relative energy are given. This file is output every ∗∗neboutfreq∗ steps of the NEB optimization.
- **input_YYYY.xyz**: These files contain the refined path at NEB iteration *YYYY*. These files are output every *neboutfreq* steps of the NEB optimization, and are in standard XYZ format. They can be visualized in *VMD* or *Avogadro*.
- **input.energy-neb-start**: This contains the energy profile of the reaction string at the start of the calculation.
- **input.energy-neb-end**: This contains the energy profile of the reaction string at the end of the calculation.

In addition to the above ouptut files, the ∗.log file produced by all CDE calculations will also contain some useful information about the NEB run.

14.0.7 Next steps

Possible further steps include:

- Performing geometry optimization, followed by frequency calculations, for the reaction-path end-points in order to calculate the free energies using the harmonic oscillator/rigid rotor approximation.
- Finding the transition state for the reaction, using the image nearest to the top of the reaction-barrier obtained by NEB.
- Using all of the above information to calculte the transition-state theory rate constant.

Chapter 15

Tutorial 2: CINEB using LAMMPS

This tutorial gives a set of example input files and instructions for a CINEB calculation which uses the ReaxFF forcefield through LAMMPS.

In this case, we are going to run a CINEB optimization of a reaction path representing association of water H2O onto a Pt atom which also has a pre-bound CO molecule. We will use the ReaxFF reactive force-field to model the potential energy surface, accessed through the LAMMPS molecular dynamics code.

The actual files to run this example can be found in the $*\sim$/cde/examples/Tutorial_2 directory.

In this directory, you will find several input files:

```
  input
  path.xyz
  CHOPtNiX.ff
  lmp.head
  lmp.min
  lmp_control
```

These files are all required in order to the run the CINEB/LAMMPS calculation defined in the input file.

In general, the *input* is similar to that given in *Tutorial 1* (Tutorial 1: CINEB). The key difference is that the setup of the PES evaluations is different - in *Tutorial 1* we used *ORCA*, but here we use ReaxFF via LAMMPS. As a result, the *PES* input block of the input file looks like the following in this case:

```
pestype  lammps
pesfile   lmp.head
pesopttype  lammps
pesoptfile lmp.min
pesexecutable '/Users/scott/code/lammps-22Aug18/src/lmp_serial'
pesoptexecutable '/Users/scott/code/lammps-22Aug18/src/lmp_serial'
```

There are a few important things to note about running calculations with ReaxFF/LAMMPS:

- Here, the *pestype* and *pestoptype* have both been set to *lammps*, and the paths to the relevant LAMMPS executable have also been specified.
- **To get this example running, you MUST modify the executables to point to the LAMMPS executable on your own system.**
- Another difference with *Tutorial 1* is that the PES template files are different. In this case, the template files *lmp.head* and *lmp.min* are LAMMPS template files. More information can be found in the *PES templates* (PES templates) section of the documentation.
- Finally, as also noted in the *PES templates* (PES templates) section, the LAMMPS code requires two additional files to be present in the run directory. These are: (1) A $*$.ff file, containing the ReaxFF parameters which should be used. In our case, this is called *CHOPtNiX.ff*. (2) A *lmp_control* file, which contains additional control parameters used by LAMMPS!

Chapter 16

Tutorial 3: Reaction path-finding

This section gives example input files and instructions for a typical double-ended reaction path-finding calculation run through CDE.

This calculation will perform a simulated annealing (SA) optimization of the mechanism error-function, seeking out a reaction mechanism which connects the reactant graph (as calculated from the input reactant xyz file) to the product graph (as determined from the input product xyz file).

Important: Note that the target reactant and prosuct graphs are calculated directly from the input reactant and product structures. So, make sure that the bond-lengths in your reactants and products are correct so that the calculated connectivity matrices are also correct!

The allowed moves (i.e. chemical reactions) which are used in the search for a mechanism connecting the reactant and product are provided in the move file.

In this example, we're going to look at the oxidation of carbon monoxide to carbon dioxide, occuring on a platinum cluster.

The actual files to run this example can be found in the ∗∼/cde/examples/Tutorial_5/∗ directory.

In this directory, you will find several input files:

```
input
start.xyz
end.xyz
moves.in
```

All of these input filenames are arbitrary; we could have called them alice, bob, clive and ralph and the program would still read them in correctly. Note that the extensions have no effect for input files read in by CDE, but they can be used to remind you what the different input files are.

Let's have a look at each input file individually:

16.0.1 input

The *input* file for this GDS run looks like this:

```
# Test input file

optaftermove .true.
calctype pathfind
nimage 10
startfile start.xyz
endfile end.xyz
ranseed 1

# path optimization
projforcetype 3
nebmethod quickmin
nebiter 1000
cithresh 1d-4
nebspring 0.1
nebstep  10.0
neboutfreq 5
nebconv 1d-4

# constraints
dofconstraints 0
atomconstraints 7
1 2 3 4 5 6 7

# PES input
pestype  lammps
pesfile   lmp.head
pesopttype  lammps
pesoptfile lmp.min
pesexecutable '/Users/scott/code/lammps-22Aug18/src/lmp_serial'
pesoptexecutable '/Users/scott/code/lammps-22Aug18/src/lmp_serial'

# Graph-driven sampling (GDS) control.
movefile moves.in
gdsthresh 0.5
gdsspring 0.02
gdsrestspring 0.05
nbstrength 0.03
nbrange 2.2
kradius 0.05
ngdsiter 500
ngdsrelax 7000
gdsdtrelax 0.15

# Chemical constraints.
valencerange{
Pt 2 12
O 1 2
C 1 4
}

reactiveatomtypes{
Pt
C
O
}

reactiveatoms{
all
}

reactivevalence{
}

fixedbonds{
Pt Pt
}
```

```
allowedbonds{
}

# Pathfinder calculation setup parameters.
nrxn 10
nmcrxn 1000000
mcrxntemp 1000000.0
nmechmove 1
graphfunctype 2
```

The input file in this case only contains (mostly) those options which are relevant to the pathfinder simulation; other input directives have been removed and will be ignored internally in CDE.

The parameter input blocks in the input file have already been covered in other tutorials for GDS and for CINEB (see, for example, Annotated input file description). In the case of a pathfinding calculation, the following are the new parameters which have an impact on the algorithm:

- **calctype**: Note that the *calctype* parameter must be set to *pathfind*.
- **nrxn**: This is the total number of reactions used in the reaction mechanisms being searched and generated. Note that a "null" reaction is automatically included as a possible graph-move (reaction) during the simulated-annealing search, so *nrxn* is the maximum number of active chemical reactions used in each proposed reaction mechanism.
- **nmcrxn**: This is the total number of simulated annealing iterations to search for; if a reaction is not found within this number of steps, the simulation simply stops without producing final structures.
- **mcrxntemp**: This is the *initial* temperature for the simulated-annealing search; note that this temperature is linearly-scaled to zero over the course of the *nmcrxn* iterations to drive the system to local minima with lower error functions. The actual initial temperature must be chosen to suit the graphfunctype (see below).
- **nmechmove**: This integer identifies how many of the reactions the simulated-annealing algorithm should attempt to change in *each* iteration. We haven't yet systematically investigated the impact of this number; 1 or 2 seems like sensible values.
- **graphfunctype**: This integer (0, 1, 2 or 3) defines the function-type which is optimized during the SA run. All of the graph-functions are zero when one has found a mechanisms which connects the reactants and products in *nrxn* steps or less. The difference arises due to the treatment of permutational invariance, as follows:

1. *graphfunctype 0* does not consider permutational invariance. The reactant and product structures must match in their atomic ordering, and one must also make a choice of which reactant atom ends up in which position in the prodoct structure. This is obviously not that useful for automatic reaction discovery.
2. *graphfunctype 1* accounts for permutational invariance by calculating a graph-error function based on eigen-values of a matrix defined as $M_{ij} = (m_i m_j)/d_{ij}$, where m_i is the atomic mass of atom i and d_{ij} is the shortest distance between any two atoms calculated in terms of bond connections.
3. *graphfunctype 2* accounts for permutational invariance by calculating a valence histogram for each pair of element types.
4. *graphfunctype 3* accounts for permutational invariance by calculating a graph-error function based on eigen-values of a matrix which is identical to the connectivity matrix except with the atomic masses on the diagonal elements.

In the calculation setup for this example, we're allowing a maximum of 10 active reactions in the proposed reaction mechanisms, we're running for a maximum of 1000000 iterations, and we're starting at a temperature of 1,000,000 K. Note that this is not a "real" temperature, but is instead related to the graph error function type.

16.0.2 start.xyz and end.xyz

The *start.xyz* and *end.xyz* files are standard XYZ format files (Cartesian coordinates in Angstroms).

In a reaction pathfinding simulation, *start.xyz* contains the coordinates of the **reactant** structure and *end.xyz* contains the coordinates of the **product** structure. These initial stuctures are used to calculate the reactant connectivity matrix and the target (product) connectivity matrix.

16.0.3 moves.in

The moves.in file describes the allowed chemical reaction classes which are allowed to take place; these allowed moves are applied to configurations during a pathfinding calculation in order to generate new reaction end-points.

The format of the move-file is discussed in Movefile example.

16.0.4 Running the calculation

With these input files, we are now able to run the pathfinding calculation. To do so, go into the $*\sim$/cde/examples/pathfind/$*$ directory and type:

```
cde.x input
```

As in the other tutorials, the above assumes that you have already made sure that CDE can be run by simply typing *cde.x*. See the setup section for more details.

As the calculation runs, you will find lots of output files generated in the run directory.

The interesting output files are as follows:

- **mcopt.dat**: This file contains a running value of the total graph-error function as a function of the number of simulated annealing iterations.
- **input.log**: Once the simulated annealing calculation is complete, the *log* file will contain lots of useful information about the final reaction mechanism (if successful).
- **final_path.xyz**: Contains an *xyz* file with molecular snapshots of the intermediates generated along the final reaction-path.
- **final_path_rx_ZZZ.xyz**: Contains an approximation to the reaction-path for reaction-step *ZZZ* in the final mechanism.

The *xyz* files can obviously be plotted in VMD in order to visualize the reaction.

16.0.5 Optimizing structures

If one sets the input parameter

```
    optaftermove .true.
```

then this means that each of the intemediate structures generated in the *final* reaction mechanism will be optimized, using the *pesopttype* and defined in the input file. In addition, the energy of each intermediate, as calculated according to *pestype*, will also be output to the *log* file.

Using *optaftermove .true.* is advised if one is intending to perform further calculations on the final reaction mechanism, such as NEB calculations for each reaction-path file (*final_path_rx_ZZZ.xyz*). In this case, the initial reaction-paths created for each reaction are generated such that they connect the optimized geomtries.

16.0.6 Final output and next steps.

- Each of the reaction-path files ((*final_path_rx_ZZZ.xyz*) generated by a pathfinding could, in principle, be used as the starting-point of a CINEB refinement calculation.
- By comparing CINEB calculations for a number of different pathfinding simulation outputs, it should also be possible to identify the "most likely" reaction mechanism (for example, with most favourable thermodynamic and kinetic properties).

Chapter 17

Tutorial 4: Another reaction path-finding example

This section gives a second example input file set for a typical double-ended reaction path-finding calculation run through CDE.

The calculation set-up is similar to Tutorial 3, but uses a different optimization function (type 4) instead. So, in this example, the target product is a single molecule, whereas the reactants contain a set of different molecular species.

This example can be run in the same way as tutorial 3, and similar input files are output.

The interesting output files are as follows:

- **mcopt.dat**: This file contains a running value of the total graph-error function as a function of the number of simulated annealing iterations.
- **input.log**: Once the simulated annealing calculation is complete, the *log* file will contain lots of useful information about the final reaction mechanism (if successful).
- **final_path.xyz**: Contains an *xyz* file with molecular snapshots of the intermediates generated along the final reaction-path.
- **final_path_rx_ZZZ.xyz**: Contains an approximation to the reaction-path for reaction-step *ZZZ* in the final mechanism.
- **adjusted_path_ZZZ.xyz**: These adjusted xyz files contain only those molecules which are relevant to the formation of the product of interest.

The *xyz* files can obviously be plotted in VMD in order to visualize the reaction.

Chapter 18

Modules Index

18.1 Modules List

Here is a list of all documented modules with brief descriptions:

chemstr
Chemical structure object definition . . . . **??**
constants
Constants definition file, containing importance universal values used throughout code . . . . **??**
functions
Functions . . . . **??**
globaldata
Contains global data definitions for CDE . . . . **??**
io
Main input/output functions for CDE . . . . **??**
pathfinder
Contains routines for automatic determination of a reaction-mechanism connecting two distinct (user-input) structures . . . . **??**
pathopt
Contains routines for path optimization, including CINEB . . . . **??**
pes
Potential energy surface calculations . . . . **??**
rpath
Reaction-path object definition . . . . **??**

Chapter 19

Data Type Index

19.1 Data Types List

Here are the data types with brief descriptions:

chemstr::cxs
Chemical structure object definition . . . . ??
globaldata::fingerprint
Type definition for the fingerprint of a reaction-path . . . . ??
globaldata::graphpar . . . . ??
globaldata::molgm . . . . ??
globaldata::refresh_template . . . . ??
rpath::rxp
Type definition for the reaction-path object . . . . ??
globaldata::vtbashop . . . . ??

Chapter 20

Module Documentation

20.1 chemstr Module Reference

Chemical structure object definition.

Data Types

- type cxs

  Chemical structure object definition.

Functions/Subroutines

- subroutine createcxs (cx, na, label, x, y, z)

  CreateCXS.
- subroutine deletecxs (cx)

  DeleteCXS.
- subroutine copytonewcxs (cx1, cx2)

  CopyToNewCXS.
- subroutine copycxs (cx1, cx2)

  CopyCXS.
- subroutine setmass (cx)

  SetMass.
- subroutine createcxsfromxyz (cx, ifile)

  CreateCXSFromXYZ.
- subroutine readxyztocxs (cx, ifile)

  ReadXYZtoCXS.
- subroutine readcxs (cx, ifile)

  ReadCXS.
- subroutine setcxsconstraints (cx, NDOFconstr, FixedDOF, Natomconstr, FixedAtom)

  SetCXSConstraints.
- subroutine getprojectedmomenta (cx)

  GetProjectedMomenta.
- subroutine getgraph (cx)

  GetGraph.

- subroutine getmols (cx)

  GetMols.
- subroutine getshortestpaths (N, dg, dsp)

  GetShortestPaths.
- subroutine graphconstraints (cx, kspring, nbstrength, nbrange, kradius)

  GraphConstraints.
- subroutine graphconstraints_doubleended (cx, cxstart, kspring, nbstrength, nbrange, kradius)

  GraphConstraints_DoubleEnded.
- subroutine projactmolrottransdvdr2 (cx, nactmol, MolecAct, frozlistin)

  ProjActMolRotTransDVDR2 projects the rotational and translational degrees of freedom of each nactmol active molecule in molecact from the dcerivative vector dvdr. WARNING - THE PROJECTION OF OF DVDR ONLY WORKS PARTIALLY FOR ROTATIONAL DOF, IF DVDR IS VERY LARGE, IT NO LONGER WORKS APPROPRIATELY..
- subroutine projoutactmolrottransdvdr (cx, nactmol, MolecAct, mcx)

  ProjOutActMolRotTransDVDR projects out the rotational and translational degrees of freedom of the system of active molecules from the dcerivative vector dvdr.
- subroutine projoutactmolrottransdvdrpair (cx, MolecAct, nactmol, ic)

  ProjOutActMolRotTransDVDRPair.
- subroutine projoutactmolrottransdvdr2 (cx, MolecAct, nactmol, ic)

  ProjOutActMolRotTransDVDR2.
- subroutine projmolrottransdvdr2 (cx)

  ProjMolRotTransDVDR2 projects the rotational and translational degrees of freedom of each molecule from the dcerivative vector dvdr.
- subroutine projmolrottransdvdr (cx)

  ProjMolRotTransDVDR projects the rotational and translational degrees of freedom of each molecule from the dcerivative vector dvdr not sure if this one works...
- integer function molidofatom (atid, cx)

  MolIdFromAtom.
- double precision function, dimension(3, 3) molmomentofinertiatensor (cx, mi)

  MoleculeMomentOfInertiaTensor.
- real(8) function, dimension(3) molecularcom (cx, m)

  MolecularCOM.
- subroutine accumulatederivatives (cx, t1, i, j)

  AccumulateDerivatives.
- subroutine get3rings (cx, nrings)

  Get3rings.
- subroutine get4rings (cx, nrings)

  Get4rings.
- subroutine printcxstofile (cx, filename, value)

  PrintCXSToFile.
- subroutine createmolecularcx (cx, mcx, m)

  CreateMolecularCX.
- subroutine **createsubstructurecx** (cx, mcx, id, na)
- integer function, dimension(cx%namol(m), cx%namol(m)) moleculargraph (cx, m)

  MolecularGraph.
- character(len=2) function, dimension(cx%namol(m)) molecularalabel (cx, m)

  MolecularAtomicLabel.
- subroutine removehydrogens (cxin, cxout)

  RemoveHydrogens.
- subroutine setcxslattice (cx)

  SetCXSLattice.
- subroutine optcxsagainstgraph (cx, gdsrestspring, nbstrength, nbrange, kradius, nrelax, step)

  OptCXSAgainstGraph.

- subroutine optimizegrp (cx, success, gdsrestspring, nbstrength, nbrange, kradius, ngdsrelax, gdsdtrelax)

  OptimizeGRP.
- subroutine optimizegrp2 (cx, cxstart, success, gdsrestspring, nbstrength, nbrange, kradius, ngdsrelax, gdsdtrelax)

  OptimizeGRP2.
- subroutine optimizegrp_doubleended (cx, cxstart, success, gdsrestspring, nbstrength, nbrange, kradius, ngdsrelax, gdsdtrelax)

  OptimizeGRP_DoubleEnded.
- character(:) function, allocatable molecularformula (cx, molid)

  MolecularFormula.
- integer function, dimension(2, cx%na) getatomvalency (cx, fixedbonds)

  GetAtomValency.
- integer function, dimension(cx%na, cx%na) getelementpairvalency (cx)

  GetElementPairValency.
- logical function allowedcxsvalencerange (cx)

  AllowedCXSValenceRange.
- logical function allowedcxsreactivevalence (cx)

  AllowedCXSReactiveValence.
- logical function allowedcxsbondsmax (cx)

  AllowedCXSBondsMax.
- subroutine setreactiveindices (bondchange, atomchange, naa, cx, rxindex, nrx)

  SetReactiveIndices.
- subroutine checkforbidden (cx, nforbid, naforbid, gforbid, forbidlabel, iflag)

  CheckForbidden.
- subroutine printcxsgraphinfo (cx, iunit, message)

  PrintCXSGraphInfo()
- subroutine setvalencecoords (cx)

  SetValenceCoords()
- subroutine getbonds (cx)

  GetBonds()
- subroutine getangles (cx)

  GetAngles()
- integer function, dimension(cx%namol(mi) ∗(cx%namol(mi) -1)/2) getmoleculargraphvector (cx, mi)

  GetMolecularGraphVector.
- character(250) function getmolecularsmiles (cx, mi)

  GetMolecularSmiles.
- double precision function molecularmass (cx, molid)

  MolecularMass.

20.1.1 Detailed Description

Chemical structure object definition.

Defines the chemical structure (cxs) object type.

20.1.2 Function/Subroutine Documentation

20.1.2.1 createcxs()

```
subroutine chemstr::createcxs (
            type(cxs) cx,
            integer na,
            character*2, dimension(*) label,
            real*8, dimension(*) x,
            real*8, dimension(*) y,
            real*8, dimension(*) z )
```

CreateCXS.

Creates a CXS object directly using labels and x,y,z coordinates which are passed in.

cx - the created CXS object. na - number of atoms. label - atomic labels. x,y,z, - atomic coordinates in atomic units.

Definition at line 81 of file structure.f90.

20.1.2.2 deletecxs()

```
subroutine chemstr::deletecxs (
            type(cxs) cx )
```

DeleteCXS.

Deletes memory associated with a CXS object.

cx - the created CXS object.

Definition at line 180 of file structure.f90.

20.1.2.3 copytonewcxs()

```
subroutine chemstr::copytonewcxs (
            type(cxs) cx1,
            type(cxs) cx2 )
```

CopyToNewCXS.

Creates a new copy of cx1 and puts it in cx2. Note that this routine allocates new space for cx2 - cx2 must not have already been allocated.

- cx1: Input chemical structure object.
- cx2: Nex chemical structure object.

Definition at line 221 of file structure.f90.

20.1.2.4 copycxs()

```
subroutine chemstr::copycxs (
             type(cxs) cx1,
             type(cxs) cx2 )
```

CopyCXS.

Creates a copy of cx1 and in cx2. Note that cx2 must already exist as a cxs object.

- cx1: Input chemical structure object.
- cx2: Nex chemical structure object.

Definition at line 282 of file structure.f90.

20.1.2.5 setmass()

```
subroutine chemstr::setmass (
             type (cxs) cx )
```

SetMass.

Sets the atomic masses in a cxs object based on atomic labels. The mass values are stored in 'constants.f90'.

- cx: The chemical structure object.

Definition at line 332 of file structure.f90.

20.1.2.6 createcxsfromxyz()

```
subroutine chemstr::createcxsfromxyz (
             type(cxs) cx,
             character, dimension(*), intent(in) ifile )
```

CreateCXSFromXYZ.

Creates empty CXS from XYZ

- cx: the created CXS object.
- ifile: the input file to read the coordinates from. Note that this must be an xyz file format.

Definition at line 362 of file structure.f90.

20.1.2.7 readxyztocxs()

```
subroutine chemstr::readxyztocxs (
            type(cxs) cx,
            character, dimension(*), intent(in) ifile )
```

ReadXYZtoCXS.

Reads in a set of coordinates from xyz and puts it in CXS beware: the numer of atoms and indecies must be the same in the XYZ as the CXS allocated arrays

- cx: the created CXS object.
- ifile: the input file to read the coordinates from. Note that this must be an xyz file format. seb

Definition at line 407 of file structure.f90.

20.1.2.8 readcxs()

```
subroutine chemstr::readcxs (
            type(cxs) cx,
            character, dimension(*), intent(in) ifile )
```

ReadCXS.

Reads in a set of coordinates to a CXS object from an xyz file.

- cx: the created CXS object.
- ifile: the input file to read the coordinates from. Note that this must be an xyz file format.

Definition at line 464 of file structure.f90.

20.1.2.9 setcxsconstraints()

```
subroutine chemstr::setcxsconstraints (
            type(cxs) cx,
            integer, intent(in) NDOFconstr,
            integer, dimension(*), intent(in) FixedDOF,
            integer, intent(in) Natomconstr,
            integer, dimension(*), intent(in) FixedAtom )
```

SetCXSConstraints.

sets constraints on DOFs and atoms in a structure.

- cx: the created CXS object.
- NDOFconstr: Number of DOF constraints.
- FixedDOF: Array containing integer number of constrained DOFs
- Natomconstr: Number of atom constraints
- Fixedatom: Array containing integer ids of fixed atoms.

Definition at line 569 of file structure.f90.

20.1.2.10 getprojectedmomenta()

```
subroutine chemstr::getprojectedmomenta (
            type(cxs) cx )
```

GetProjectedMomenta.

Calculates the momenta projected along the direction of the force array - this is usually the forces from (CI)NEB.

- cx: A chemical structure object.

Definition at line 610 of file structure.f90.

20.1.2.11 getgraph()

```
subroutine chemstr::getgraph (
            type(cxs) cx )
```

GetGraph.

Calculates the bonding graph for the system. The values which define whether or not atoms are bonded are based on covalent radii, and are stored in constants.f90 in an atomic-number-labelled array called BondingCutoff(:,:).

- cx: A chemical structure object.

Definition at line 698 of file structure.f90.

20.1.2.12 getmols()

```
subroutine chemstr::getmols (
            type(cxs) cx )
```

GetMols.

Uses the connectivity graph to determine the molecules defined by the chemical structure cx. To do so, we use Floyd-Warshall shortest-path walks on the chemical structure graph. Molen/Molcharge arrays will be reset, so beware!

- cx: Input chemical structure object.

Definition at line 766 of file structure.f90.

20.1.2.13 getshortestpaths()

```
subroutine chemstr::getshortestpaths (
            integer N,
            real(8), dimension(n,n) dg,
            real(8), dimension(n,n) dsp )
```

GetShortestPaths.

Calculates the matrix dsp(i,j) which contains the shortest path between points i and j as determined by Floyd-↩ Warshall algorithm.

- n: number of points
- dg(:,:): distance between points.
- dsp(:,:): Shortest-path between points.

Definition at line 882 of file structure.f90.

20.1.2.14 graphconstraints()

```
subroutine chemstr::graphconstraints (
            type(cxs) cx,
            real(8) kspring,
            real(8) nbstrength,
            real(8) nbrange,
            real(8) kradius )
```

GraphConstraints.

Calculates the potential energy and forces arising due to graph constraints.

- cx: A chemical structure object.
- kspring: Spring constant for the bonding restraint term.
- nbstrength: Repulsive exponential interaction strength in au for atom pairs with g(i,j) = 0
- nbrange: Repulsive exponential interaction range in au for atom pairs with g(i,j) = 0
- kradius: Harmonic restraint term strength for different molecules.

Definition at line 924 of file structure.f90.

20.1.2.15 graphconstraints_doubleended()

```
subroutine chemstr::graphconstraints_doubleended (
            type(cxs) cx,
            type(cxs) cxstart,
            real(8) kspring,
            real(8) nbstrength,
            real(8) nbrange,
            real(8) kradius )
```

GraphConstraints_DoubleEnded.

Calculates the potential energy and forces arising due to graph constraints.

- cx: A chemical structure object.
- kspring: Spring constant for the bonding restraint term.
- nbstrength: Repulsive exponential interaction strength in au for atom pairs with g(i,j) = 0
- nbrange: Repulsive exponential interaction range in au for atom pairs with g(i,j) = 0
- kradius: Harmonic restraint term strength for different molecules.

Definition at line 1193 of file structure.f90.

20.1.2.16 projactmolrottransdvdr2()

```
subroutine chemstr::projactmolrottransdvdr2 (
            type(cxs) cx,
            integer nactmol,
            integer, dimension(nactmol) MolecAct,
            logical, dimension(nactmol), optional frozlistin )
```

ProjActMolRotTransDVDR2 projects the rotational and translational degrees of freedom of each nactmol active molecule in molecact from the dcerivative vector dvdr. WARNING - THE PROJECTION OF OF DVDR ONLY WORKS PARTIALLY FOR ROTATIONAL DOF, IF DVDR IS VERY LARGE, IT NO LONGER WORKS APPROP↩ RIATELY..

- cx: Chemical structure
- bx: Chemical structure object
- rad_min: distance between two molecules hard spheres
- nactmol: number of active molecules in cx and bx
- MolecAct: the ID of the molecules that are acive in cx and bx (size of array = nactmol)
- gm: matrix of molecules with atoms bonding/ breaking during a reaction

Definition at line 1402 of file structure.f90.

20.1.2.17 projoutactmolrottransdvdr()

```
subroutine chemstr::projoutactmolrottransdvdr (
            type(cxs) cx,
            integer nactmol,
            integer, dimension(nactmol) MolecAct,
            type(cxs), optional mcx )
```

ProjOutActMolRotTransDVDR projects out the rotational and translational degrees of freedom of the system of active molecules from the dcerivative vector dvdr.

- cx: Chemical structure
- bx: Chemical structure object
- rad_min: distance between two molecules hard spheres
- nactmol: number of active molecules in cx and bx
- MolecAct: the ID of the molecules that are acive in cx and bx (size of array = nactmol)
- mcx (optional): if another cx is provided, then it uses the molid and namol arrays from that one

Definition at line 1501 of file structure.f90.

20.1.2.18 projoutactmolrottransdvdrpair()

```
subroutine chemstr::projoutactmolrottransdvdrpair (
            type(cxs), dimension(2) cx,
            integer, dimension(cx(ic)%nmol) MolecAct,
            integer nactmol,
            integer ic )
```

ProjOutActMolRotTransDVDRPair.

removes the overal translation and rotation DOF of the superset of a pair of cx(1:2) from the DVDR, for molecules that are "active" (break/form bonds based on graphs in images at endpoints) it uses the molecule id's of the image ic note that it does not check weather the molecules are frozen or not...

- rp: Reaction Path Object
- if:

Definition at line 1605 of file structure.f90.

20.1.2.19 projoutactmolrottransdvdr2()

```
subroutine chemstr::projoutactmolrottransdvdr2 (
            type(cxs), dimension(2) cx,
            integer, dimension(nactmol) MolecAct,
            integer nactmol,
            integer ic )
```

ProjOutActMolRotTransDVDR2.

removes the overal translation and rotation DOF of the entire rp coordinates from the DVDR, for molecules that are "active" (break/form bonds based on graphs in images at endpoints) it uses the molecule id's of the image ic note that it does not check weather the molecules are frozen or not...

- rp: Reaction Path Object
- if:

Definition at line 1722 of file structure.f90.

20.1.2.20 projmolrottransdvdr2()

```
subroutine chemstr::projmolrottransdvdr2 (
            type(cxs) cx )
```

ProjMolRotTransDVDR2 projects the rotational and translational degrees of freedom of each molecule from the dcerivative vector dvdr.

- cx: Chemical structure
- bx: Chemical structure object
- rad_min: distance between two molecules hard spheres
- nactmol: number of active molecules in cx and bx
- MolecAct: the ID of the molecules that are acive in cx and bx (size of array = nactmol)
- gm: matrix of molecules with atoms bonding/ breaking during a reaction

Definition at line 1841 of file structure.f90.

20.1.2.21 projmolrottransdvdr()

```
subroutine chemstr::projmolrottransdvdr (
            type(cxs) cx )
```

ProjMolRotTransDVDR projects the rotational and translational degrees of freedom of each molecule from the dcerivative vector dvdr not sure if this one works...

- cx: Chemical structure
- bx: Chemical structure object
- rad_min: distance between two molecules hard spheres
- nactmol: number of active molecules in cx and bx
- MolecAct: the ID of the molecules that are acive in cx and bx (size of array = nactmol)
- gm: matrix of molecules with atoms bonding/ breaking during a reaction

Definition at line 1929 of file structure.f90.

20.1.2.22 molidofatom()

```
integer function chemstr::molidofatom (
            integer atid,
            type(cxs) cx )
```

MolIdFromAtom.

Tells you which molecule the atom belogs to. Returns the ID of that molecule

- cx: chemical structure
- atid: atom index

Definition at line 2066 of file structure.f90.

20.1.2.23 molmomentofinertiatensor()

```
double precision function, dimension(3,3) chemstr::molmomentofinertiatensor (
            type(cxs) cx,
            integer mi )
```

MoleculeMomentOfInertiaTensor.

Returns the Moment of inertia tensor matrix of a molecule mi in chemical structure cx

mi - the index of the molecule mit - returns moment of inertia tensor

Definition at line 2108 of file structure.f90.

20.1.2.24 molecularcom()

```
real(8) function, dimension(3) chemstr::molecularcom (
            type(cxs) cx,
            integer m )
```

MolecularCOM.

returns the centre of mass of molecule m from reaction strucuture cx

- cx: Chemical structure object.
- m: molecule in cxs

Definition at line 2154 of file structure.f90.

20.1.2.25 accumulatederivatives()

```
subroutine chemstr::accumulatederivatives (
            type(cxs) cx,
            real(8) t1,
            integer i,
            integer j )
```

AccumulateDerivatives.

Updates dvdr with additional forces.

- cx: Chemical structure object.
- t1: Force term to be applied.
- i, j: Atom indexes.

Definition at line 2183 of file structure.f90.

20.1.2.26 get3rings()

```
subroutine chemstr::get3rings (
            type(cxs) cx,
            integer nrings )
```

Get3rings.

Counts the number of 3-rings in the cx graph.

- cx: Chemical structure object.
- nrings: Number of 3-rings identified.

Definition at line 2225 of file structure.f90.

20.1.2.27 get4rings()

```
subroutine chemstr::get4rings (
            type(cxs) cx,
            integer nrings )
```

Get4rings.

Counts the number of 4-rings in the cx graph.

- cx: Chemical structure object.
- nrings: Number of 4-rings identified.

Definition at line 2270 of file structure.f90.

20.1.2.28 printcxstofile()

```
subroutine chemstr::printcxstofile (
            type(cxs) cx,
            character, dimension(*) filename,
            real*8 value )
```

PrintCXSToFile.

Outputs cxs coordinates to an xyz file.

- cx: The CXS object.
- filename: the output xyz file.
- value: A real value which can be output as the comment line. For example, this could be the energy or fitness.

Definition at line 2329 of file structure.f90.

20.1.2.29 createmolecularcx()

```
subroutine chemstr::createmolecularcx (
            type(cxs) cx,
            type(cxs) mcx,
            integer m )
```

CreateMolecularCX.

Creates a CXS mcx for a single molecule, indexed m in CX

- cx: the parent chemical structure from which mcx is generated
- mcx: the CXS of a the molecule m in cx
- m: the molecule's index in cx, which we wish to make mcx from..

Definition at line 2362 of file structure.f90.

20.1.2.30 moleculargraph()

```
integer function, dimension(cx%namol(m),cx%namol(m)) chemstr::moleculargraph (
            type(cxs) cx,
            integer, intent(in) m )
```

MolecularGraph.

Returns graph of molecule in cx

- cx: the parent chemical structure from which mcx is generated
- m: the molecule's index in cx, which we wish to make the graph from..

Definition at line 2509 of file structure.f90.

20.1.2.31 molecularalabel()

```
character(len=2) function, dimension(cx%namol(m)) chemstr::molecularalabel (
            type(cxs) cx,
            integer, intent(in) m )
```

MolecularAtomicLabel.

Returns the atomic labels for the molecle

- cx: the parent chemical structure from which the atomic labels is generated
- m: the molecule's index in cx, which we wish to make the atomic labels from..

Definition at line 2535 of file structure.f90.

20.1.2.32 removehydrogens()

```
subroutine chemstr::removehydrogens (
            type(cxs) cxin,
            type(cxs) cxout )
```

RemoveHydrogens.

Removes all hydrogen atoms from a cxs object, and returns a new cxs object with only heteroatoms.

Definition at line 2560 of file structure.f90.

20.1.2.33 setcxslattice()

```
subroutine chemstr::setcxslattice (
            type(cxs) cx )
```

SetCXSLattice.

Initializes the coordinates of the chemical structure object cxs on a simple cubic lattice, with lattice spacing determined by the parameter LATTICESTEP.

- cx: Input chemical structure object.

Definition at line 2611 of file structure.f90.

20.1.2.34 optcxsagainstgraph()

```
subroutine chemstr::optcxsagainstgraph (
            type(cxs) cx,
            real(8) gdsrestspring,
            real(8) nbstrength,
            real(8) nbrange,
            real(8) kradius,
            integer nrelax,
            real(8) step )
```

OptCXSAgainstGraph.

Optimizes the coordinates of a chemical structure object under the action of the graph-restraining potential only.

- cx: Input chemical structure object.
- gdsrestspring: GDS spring restraint strength (au)
- nbstrength: Repulsion interaction strength for non-bonded atoms.
- nbrange: Repulsion interaction range for non-bonded atoms.
- kradius: Spring constant for repulsion between deiscrete molecules (au)

Definition at line 2658 of file structure.f90.

20.1.2.35 optimizegrp()

```
subroutine chemstr::optimizegrp (
            type(cxs) cx,
            logical success,
            real(8) gdsrestspring,
            real(8) nbstrength,
            real(8) nbrange,
            real(8) kradius,
            integer ngdsrelax,
            real(8) gdsdtrelax )
```

OptimizeGRP.

Minimises the molecules according to the Graph restraining potential

- cx: The input chemical structure
- success: weather we satisfied the contraints after the monimization or not
- rest: parameters for the GRP and minimization

Definition at line 2714 of file structure.f90.

20.1.2.36 optimizegrp2()

```
subroutine chemstr::optimizegrp2 (
            type(cxs) cx,
            type(cxs) cxstart,
            logical success,
            real(8) gdsrestspring,
            real(8) nbstrength,
            real(8) nbrange,
            real(8) kradius,
            integer ngdsrelax,
            real(8) gdsdtrelax )
```

OptimizeGRP2.

Minimises the molecules according to the Graph restraining potential

- cx: The input chemical structure
- success: weather we satisfied the contraints after the monimization or not
- rest: parameters for the GRP and minimization

Definition at line 2820 of file structure.f90.

20.1.2.37 optimizegrp_doubleended()

```
subroutine chemstr::optimizegrp_doubleended (
            type(cxs) cx,
            type(cxs) cxstart,
            logical success,
            real(8) gdsrestspring,
            real(8) nbstrength,
            real(8) nbrange,
            real(8) kradius,
            integer ngdsrelax,
            real(8) gdsdtrelax )
```

OptimizeGRP_DoubleEnded.

Optimizes the geometry of cx according to a modified Graph restraining potential which tries to keep geometry close to a starting point cxstart.

- cx: The input chemical structure
- success: weather we satisfied the contraints after the monimization or not
- rest: parameters for the GRP and minimization

Definition at line 2929 of file structure.f90.

20.1.2.38 molecularformula()

```
character(:) function, allocatable chemstr::molecularformula (
            type(cxs) cx,
            integer molid )
```

MolecularFormula.

Returns a variable length string with the molecular formula of molecule molid, from chemical structure cx.

- cx: chemical structure
- molecule index

Definition at line 3019 of file structure.f90.

20.1.2.39 getatomvalency()

```
integer function, dimension(2,cx%na) chemstr::getatomvalency (
            type(cxs) cx,
            logical, dimension(:,:)  fixedbonds )
```

GetAtomValency.

returns the valency of every atom in cx, the first index (2,:) only counts bonds which can actually change during the gds calculation

- cx: The input chemical structure

Definition at line 3077 of file structure.f90.

20.1.2.40 getelementpairvalency()

```
integer function, dimension(cx%na,cx%na) chemstr::getelementpairvalency (
            type(cxs) cx )
```

GetElementPairValency.

returns a matrix of size val(na,na) with the number of bonds of a particular pair of elements (eg H-C) that corresponds to that pair of atoms va(i,j), where the first index says how many of those there are bonded to atom i of type j used for rxnvalatom tests

- cx: The input chemical structure

Definition at line 3112 of file structure.f90.

20.1.2.41 allowedcxsvalencerange()

```
logical function chemstr::allowedcxsvalencerange (
            type(cxs) cx )
```

AllowedCXSValenceRange.

Checks wheather the valence of elements falls withtin the allowed range

- cx: The input chemical structure

Definition at line 3148 of file structure.f90.

20.1.2.42 allowedcxsreactivevalence()

```
logical function chemstr::allowedcxsreactivevalence (
            type(cxs) cx )
```

AllowedCXSReactiveValence.

Checks wheather the valence of an element to some other falls withing the allowed values

- cx: The input chemical structure

Definition at line 3198 of file structure.f90.

20.1.2.43 allowedcxsbondsmax()

```
logical function chemstr::allowedcxsbondsmax (
            type(cxs) cx )
```

AllowedCXSBondsMax.

Checks wheather the bonds present are allowed as prescibed by allowedbonds keyword

- cx: The input chemical structure

Definition at line 3239 of file structure.f90.

20.1.2.44 setreactiveindices()

```
subroutine chemstr::setreactiveindices (
            logical, dimension(naa,naa) bondchange,
            logical, dimension(naa) atomchange,
            integer naa,
            type(cxs) cx,
            integer, dimension(namax) rxindex,
            integer nrx )
```

SetReactiveIndices.

Sets the bondchange(:,:) and atomchange(:) indices, which indicate whether the specified bonds or atoms are allowed to change as a result of a proposed graph move.

Returns the bondchange(:,:) and atomchange(:) variables, as well as the number of reactive atoms (nrx) and a list of the reactive atoms (in rxindex(1:nrx)).

bondchange(:,:) - Logical array indicating whether bond (i,j) is allowed to change or not. atomchange(:) - Logical array indicating whether or not atom i is allowed to change its bonding pattern. na - Number of atoms cx - A cxs object, primarily used to provide atom labels. rxindex - List of reactive atoms. nrx - Number of reactive atoms.

Definition at line 3289 of file structure.f90.

20.1.2.45 checkforbidden()

```
subroutine chemstr::checkforbidden (
            type(cxs) cx,
            integer nforbid,
            integer, dimension(nforbidmax) naforbid,
            integer, dimension(nforbidmax,namovemax,namovemax) gforbid,
            character (len=4), dimension(nforbidmax,namovemax) forbidlabel,
            integer iflag )
```

CheckForbidden.

Checks whether the current graph, generated during the GraphMoves, routine, matches one of the forbidden patterns given in the fobidfile.

This subroutine is a nightmare-ish hellscape of enddo and endif... nobody should have to deal with this...

Returns iflag = 0 if the graph is forbidden.

Definition at line 3406 of file structure.f90.

20.1.2.46 printcxsgraphinfo()

```
subroutine chemstr::printcxsgraphinfo (
            type(cxs) cx,
            integer iunit,
            character(len=*) message )
```

PrintCXSGraphInfo()

Outputs the bonding graph information for a chemical structure object.

- cx: The chemical structure object
- iunit: Integer ID of the output destination.
- message: Character string (80) identifying CXS details.

Parameters

cx	Chemical structure object
iunit	Output unit ID
message	A message identifying this CXS.

Definition at line 3592 of file structure.f90.

20.1.2.47 setvalencecoords()

```
subroutine chemstr::setvalencecoords (
            type(cxs) cx )
```

SetValenceCoords()

Calculates the set of internal coordinates (bond-lengths, bond-angles and torsion angles) for cx. Note that the derivatives of each internal coordinates with respect to the atomic coordinates is also calculated.

Important: The identification of bonds, angle and torsion angles is based on the cxgraph(:,:).

- cx: The chemical structure object

Parameters

cx	Chemical structure object

Definition at line 3647 of file structure.f90.

20.1.2.48 getbonds()

```
subroutine chemstr::getbonds (
            type(cxs) cx )
```

GetBonds()

Calculates the set of bond-lengths for cx. Note that the derivatives of each bond-length with respect to the atomic coordinates is also calculated.

Important: The identification of bonds is based on the cxgraph(:,:).

- cx: The chemical structure object

Definition at line 3697 of file structure.f90.

20.1.2.49 getangles()

```
subroutine chemstr::getangles (
            type(cxs) cx )
```

GetAngles()

Calculates the set of bond-angles for cx. Note that the derivatives of each bond-angle with respect to the atomic coordinates is also calculated.

Important: The identification of bonds is based on the cxgraph(:,:).

- cx: The chemical structure object

Definition at line 3751 of file structure.f90.

20.1.2.50 getmoleculargraphvector()

```
integer function, dimension(cx%namol(mi)*(cx%namol(mi)-1)/2) chemstr::getmoleculargraphvector
(
            type(cxs) cx,
            integer mi )
```

GetMolecularGraphVector.

fobidfile.

Returns iflag = 0 if the graph is forbidden.

Definition at line 3831 of file structure.f90.

20.1.2.51 getmolecularsmiles()

```
character(250) function chemstr::getmolecularsmiles (
            type(cxs) cx,
            integer mi )
```

GetMolecularSmiles.

Returns MolSmiles, an array of characters with the Smiles strip for each molecule in MolecAct

- cx: Chemical structure
- nactmol: number of active molecules in cx
- MolecAct: array with indecies of the nactmol molecules that are active in this problem
- MolSmiles: array of nactmol Smiles strips

Definition at line 3864 of file structure.f90.

20.1.2.52 molecularmass()

```
double precision function chemstr::molecularmass (
              type(cxs) cx,
              integer molid )
```

MolecularMass.

Returns the mass of a molecule molid in cx

- cx: Chemical structure
- molid: id of the molecule in cx

Definition at line 3907 of file structure.f90.

20.2 constants Module Reference

constants definition file, containing importance universal values used throughout code.

Variables

- integer, parameter namax = 200

  Maximum number of atoms.
- integer, parameter nfmax = 3 ∗ NAMAX

  Maximum number of DOFs.
- integer, parameter nbmax = 20

  Maximum number of images.
- integer, parameter nmolmax = 50

  Maximum number of molecules.
- integer, parameter ndirmax = 50

  Maximum number of directories to create.
- integer, parameter ntypemax = 100

  Maximum number of atom types.
- integer, parameter ntrymax = 1000

  Maximum number of outer graph-move attempts.
- integer, parameter ntrymax2 = 5000

  Maximum number of inner graph-move attempts.
- integer, parameter nmovemax = 20

  Maximum number of graph-moves.
- integer, parameter nforbidmax = 50

  Maximum number of forbidden graphs.
- integer, parameter namovemax = 6

  Maximum number of atoms which can be involved in graph moves.
- integer, parameter maxmol = 4000

  Maximum number of molecules stored in MolData datatype.
- integer, parameter nmechmax = 250

  Maximum number of stored mechanisms.
- integer, parameter nrxnmax = 25

Maximum number of stored mechanisms.

- integer, parameter **nvalmax** = 12
- integer, parameter **nelmax** = 10
- integer, parameter **nbondmax** = 100
- integer, parameter **nanglemax** = 200
- integer, parameter **ntorsmax** = 300
- real(8), parameter bohr_to_ang = 0.5291772108d0

  Bohr to Angstrom conversion.
- real(8), parameter ang_to_bohr = 1.889726128d0

  Angstrom to Bohr conversion.
- real(8), parameter kcalmol_to_au = 0.001593601d0

  Convert kcal/mol to au.
- real(8), parameter au_to_kjmol = 2625.49963d0

  convert au to kJ/mol
- real(8), parameter **au_to_ev** = 27.0d0
- real(8), parameter pivalue = 3.141592654d0

  PI!
- real(8), parameter kboltz = 3.166829d-6

  Boltzmann constant in atomic units.
- real(8), parameter hbar = 1.d0

  hbar = 1 in atomic units
- real(8), parameter fs_to_au = 41.341373d0

  Convert femtosecond to atomic units.
- real(8), parameter eh_to_kelvin = 3.157733d+5

  Convert Hartrees to Kelvin.
- real(8), parameter eh_to_wavenumber = 219474.63

  Convert Hartrees to wavenumbers.
- real(8), parameter big = 10000

  A 'BIG' number which is occasionally useful.
- real(8), parameter small = 1d-5

  A 'SMALL' number which is occasionally useful.
- real(8), parameter me_to_amu = 5.485799090d-4

  Convert electron mass units to atomic mass units.
- real(8), parameter **epsil** = 10.0d0∗∗(-20)
- real(8), parameter **gconv** = 1d-7
- real(8), dimension(87), parameter mass = (/1837.1527d0, 0.d0, 0.d0, 0.d0, 19852.3d0, 21894.16747d0, 25533.1990d0, 29156.9471d0, 34905.9013d0, 0.d0, 0.d0, 0.d0, 0.d0, 51594.6d0, 56903.5d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 102595.8d0, 107524.4367d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 195509.8d0, 195509.8d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 358399.0973d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 50000.0d0/)

  Set atomic masses. The MASS index can be referenced by atomic number; For example, MASS(1) = mass of hydrogen.
- real(8), dimension(87), parameter covrad = (/0.4d0 ∗ ang_to_bohr, 0.d0, 0.d0, 0.d0, 0.84d0 ∗ ang_to_bohr, 0.72d0 ∗ ang_to_bohr, 0.72d0 ∗ ang_to_bohr, 0.62d0 ∗ ang_to_bohr, 0.6d0 ∗ ang_to_bohr, 0.d0, 0.d0, 0.d0, 0.d0, 1.15d0 ∗ ang_to_bohr, 1.07d0 ∗ ang_to_bohr, 1.07d0∗ang_to_bohr, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 1.52d0 ∗ ang_to_bohr, 1.52d0 ∗ ang_to_bohr, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 1.42d0 ∗ ang_to_bohr, 1.39d0 ∗ ang_to_bohr, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 1.46d0 ∗ ang_to_bohr, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 0.d0, 1.50d0/)

  Set covalent radii. The CovRad(:) array is indexed by atomic number, so CovRad(1) is the covalent radius of hydrogen. Note that the values are given in Angstroms and converted into Bohr using the ang_to_bohr conversion.

- integer, dimension(87), parameter **avalency** = (/1, 2, 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10 /)
- real(8), parameter bondingsf = 1.10d0

  Scale factor applied to covalent radii to define bonding. Atoms are bonded if r(i,j) <= (covrad(i) + covrad(j)) ∗ bondingsf.
- real(8), parameter bondingrange1 = 0.25d0 ∗ ang_to_bohr

  Shift range over which atoms are restrained in GDS simulations.
- real(8), parameter bondingrange2 = -0.10d0 ∗ ang_to_bohr

  Shift range over which atoms are restrained in GDS simulations.
- real(8), parameter **radius_max** = 50.d0 ∗ ang_to_bohr
- real(8), parameter **radius_min** = 12.d0 ∗ ang_to_bohr
- real(8), parameter **latticestep** = 4.d0 ∗ ang_to_bohr
- real(8), parameter **hidpp** = 2.750
- real(8), dimension(100, 100) **sbstrength** = 0.0d0
- real(8), dimension(10, 10) **dbstrength** = 0.0d0
- real(8), dimension(10, 10) **tbstrength** = 0.0d0
- real(8), parameter grpminthresh = 1d-3

  Force convergence target (in au) for minimization under GRP.
- real(8), parameter grpmaxthresh = 1d-3

  Force convergence target (in au) for minimization under GRP.

20.2.1 Detailed Description

constants definition file, containing importance universal values used throughout code.

You should avoid changing these values unless you are absolutely sure you know what you're doing.

20.3 functions Module Reference

Functions.

Functions/Subroutines

- logical function is_numeric (string)

  is_numeric
- double precision function, dimension(n, n) oprod (v1, v2, n)

  OProd.
- double precision function sigmoid (x, alp, xsh, dir)

  Sigmoid.
- double precision function, dimension(3) xproduct (a, b)

  xproduct
- integer(8) function factorial (n)

  factorial
- integer(8) function choose (n, k)

  choose
- logical function findidx (array, num, siz)

  findidx

- integer function labeltonumber (label)

  LabelToNumber.
- character(len=2) function numbertolabel (num)

  NumberToLabel.
- integer function **setranseed** (irun)
- double precision function, dimension(:), allocatable massfromlabels (labels)

  MassFromLabels.

20.3.1 Detailed Description

Functions.

Library of Generic Function

20.3.2 Function/Subroutine Documentation

20.3.2.1 is_numeric()

```
logical function functions::is_numeric (
             character(len=*), intent(in) string )
```

is_numeric

checkst to see if its number

Definition at line 25 of file functions.f90.

20.3.2.2 oprod()

```
double precision function, dimension(n,n) functions::oprod (
             double precision, dimension(n) v1,
             double precision, dimension(n) v2,
             integer n )
```

OProd.

Calculats the outer product of two vectors with dimensions n

Definition at line 41 of file functions.f90.

20.3.2.3 sigmoid()

```
double precision function functions::sigmoid (
            double precision, intent(in) x,
            double precision, intent(in) alp,
            double precision, intent(in) xsh,
            integer, intent(in) dir )
```

Sigmoid.

returns sigmoid function with exponential parameter alp, shift xsh and (in) direction +1/-1

Definition at line 54 of file functions.f90.

20.3.2.4 xproduct()

```
double precision function, dimension(3) functions::xproduct (
            double precision, dimension(3) a,
            double precision, dimension(3) b )
```

xproduct

cross product between 3d vectors, not for general space. returns vector

- a,b: vectors

Definition at line 78 of file functions.f90.

20.3.2.5 factorial()

```
integer(8) function functions::factorial (
            integer, intent(in) n )
```

factorial

Factorial calculator

Definition at line 94 of file functions.f90.

20.3.2.6 choose()

```
integer(8) function functions::choose (
            integer, intent(in) n,
            integer, intent(in) k )
```

choose

Statistical choose function

Definition at line 110 of file functions.f90.

20.3.2.7 findidx()

```
logical function functions::findidx (
            integer, dimension(siz) array,
            integer num,
            integer siz )
```

findidx

flags if we found integer num in array "array" of size siz

Definition at line 129 of file functions.f90.

20.3.2.8 labeltonumber()

```
integer function functions::labeltonumber (
            character (len = 2) label )
```

LabelToNumber.

Converts an input atomic label into a number.

- label: the input atomic label.

Definition at line 154 of file functions.f90.

20.3.2.9 numbertolabel()

```
character (len = 2) function functions::numbertolabel (
            integer num )
```

NumberToLabel.

Converts an input atomic number into label.

- num: the input atomic number.

Definition at line 440 of file functions.f90.

20.3.2.10 massfromlabels()

```
double precision function, dimension(:), allocatable functions::massfromlabels (
            character(2), dimension(:)  labels )
```

MassFromLabels.

gets the mass array from labels

- labels : label array

Definition at line 752 of file functions.f90.

20.4 globaldata Module Reference

Contains global data definitions for CDE.

Data Types

- type fingerprint

  Type definition for the fingerprint of a reaction-path.
- type graphpar
- type molgm
- type refresh_template
- type vtbashop

Variables

- integer **logfile** = 11
- character(len=50) inputfile

  Input filename.
- character(len=50) **startfile**
- character(len=50) endfile

  Filenames for start/end-points of new reaction path.
- character(len=50) pathfile

  Filename for full reaction-path for reading.
- character(len=50) movefile

  Filename for graph moves.
- character(len=50) forbidfile

  Filename for forbidden graphs.
- integer **evbtype**
- integer **evbiter**
- integer **nevbl**
- integer **nevbdist**
- integer **nevbqm**
- logical **evbvrep**
- real(8) **evbalpha1**
- real(8) **evbalpha2**

- real(8) **evbstep**
- real(8) **evbmaxdl**
- integer **idum**
- integer nimage

  Number of images in path.
- integer f

  Number of DOFs.
- integer na

  number of atoms
- integer irun

  Random number seed.
- real(8) **ran2**
- real(8) gaussian

  Randum number functions.
- integer nebiter

  Total NEB/CINEB iterations.
- integer neboutfreq

  Output frequency during NEB.
- real(8) nebconv

  NEB convergence parameter for average force modulus.
- real(8) nebmaxconv

  NEB MAXIMUM convergence parameter for average force modulus.
- real(8) cithresh

  Force threshold to turn on climbing-image NEB.
- real(8) **temperature**
- real(8) beta

  Temperature and inverse temperature (1/kT)
- real(8) nebspring

  Spring constant for NEB (au)
- real(8) nebstep

  Step-length for steepest-descent NEB.
- character(len=8) calctype

  Calculation type, NEB or GDS.
- character(len=6) pathinit

  Path initialization type, LINEAR.
- character(len=6) pathoptmethod

  Path optimization method, CINEB.
- character(len=9) nebmethod

  (CI)NEB optimization method, CINEB
- logical startfrompath

  Logical flag to signal whether or not we're starting from full path.
- logical stripinactive

  Logical flag to signal that spectator molecules should be removed before optimization.
- logical optendsbefore

  Logical flag indicating whether to geometry-optimize path end-points before path optimization.
- logical optendsduring

  Logical flag indicating whether to optimize path end-points during path optimization.
- logical reconnect

  Logical flag indicating whether to generate new linear path after end-point optimization.
- logical idpppath

  Logical flag to turn the image-dependent pair potential NEB for a improved guess path.

- logical fraginterpol

  Logical flag to turn the fragment interpolation procedure - where fragments are first brought together before interpolating.
- logical optfragorient

  Logical flag to turn the fragment orientation optimization procedure - where fragments are oriented appropriately to enable the exchange of atoms.
- integer ndofconstr

  Number of constrained DOFs.
- integer, dimension(3 ∗namax) **fixeddof**
- integer natomconstr

  Number of constrained DOFs.
- integer, dimension(namax) **fixedatom**
- character(len=6) pestype

  Identifies PES calculation type, 'ORCA'...
- character(len=25) pesfile

  Identifies PES calculation template file.
- character(len=6) pesopttype

  Identifies PES type for geometry optimisation.
- character(len=25) pesoptfile

  Identifies geometry optimisation template file.
- character(len=100) pesexecutable

  Identifies PES calculation executable.
- character(len=100) pesoptexecutable

  Identifies geometry optimisation executable.
- logical pesfull

  Logical flag indicating whether we're running PES evaluations for the full system or independent molecules.
- character(len=8), parameter **fmt4** = '(I4.4)'
- integer nreactivetypes

  Number of reactive atom-types.
- integer nfixtype

  Number of types of fixed-bonds.
- character(len=2), dimension(ntypemax) reactivetype

  Atom labels for reactive atoms.
- character(len=2), dimension(ntypemax, 2) fixedbondtype

  Atom labels for each type of fixed bonds.
- logical, dimension(namax) reactive

  Logical flag indicating whether or not atoms are reactive.
- logical, dimension(namax) essentialmoves

  Logical flag indicating whether or not atoms are essential to moves.
- logical, dimension(namax) essential

  Logical flag indicating whether or not atoms are essential.
- integer nessentialatoms

  Number of essential atoms labelled .TRUE. in essentialmoves(:)
- double precision intra_cutoff

  when performing a bond forming INTRA-molecular rxn move between atoms, set a cutoff distance
- integer **nessentialatomsinmols**
- integer, dimension(namax) **essentialatomsinmols**
- logical, dimension(namax, namax) fixedbonds

  Logical flag indicating fixed bonds.
- integer ngdsiter

  Number of GDS iterations.

- real(8) gdsspring

  Spring constant for inter-bead spring terms.
- real(8) gdsrestspring

  Spring constant for bonded graph restraints.
- real(8) nbstrength

  Strength of non-bonded exponential graph-restraints.
- real(8) nbrange

  Range of non-bonded exponential graph-restraints.
- real(8) kradius

  Spring constant for intermolecular repulsion restraints.
- real(8) gdstemperature

  End-point temperature for GDS simulation.
- real(8) gdscoefftemp

  Temperature for GDS coefficients.
- real(8) gdscoeffmass

  Mass for GDS coefficients.
- real(8) timestep

  GDS timestep.
- real(8) vsthresh

  Variable spring strength threshold (used suring CINEB)
- integer anebb

  AutoNEB number of extra beads (also works as a logical switching Autoneb on)
- integer ngmove

  Number of defined graph moves.
- integer, dimension(nmovemax) namove

  Number of atoms in each graph move.
- integer, dimension(nmovemax, namovemax, namovemax) gmstart

  Start graph for each move.
- integer, dimension(nmovemax, namovemax, namovemax) gmend

  End graph for each move.
- character(len=4), dimension(nmovemax, namovemax) movelabel

  Labels for atoms involved in graph moves.
- character(len=7), dimension(nmovemax) movetype

  For labelling as 'ionize' or 'eattach'.
- real(8), dimension(nmovemax) moveprob

  Probability of graph-move (not that these are normalized internally)
- integer gdsoutfreq

  GDS output frequency.
- real(8) gdsthresh

  GDS graph-move attempt probability.
- integer ngdsrelax

  Number of SD relaxation steps under graph restraints after graph moves.
- real(8) gdsdtrelax

  Step size for SD relaxation after graph moves.
- logical optaftermove

  Flag indicating whether to optimize on PES after each graph move.
- logical skiprepeats

  Flag indicating whether to skip repeat graph moves.
- logical gatherreactivemol

  Flag indicating whether to make the reactive molecles gather around into a "circle".
- logical simpleopt

optimises end images in NEB, all molecules in image at once...
- logical nebrestrend

  Flag indicating whether or not graph-restraint potenial is applied to end-points during NEB optimization.
- logical removecollisions

  Flag indicating whether to refine path after move under purely repulsive potential for non-bonded atoms.
- logical shimmybeads

  Flag determining if we want flat PES beads at the ends to shimmy (center the path)
- logical restartgds

  Flag determining weather user wishes to read a ChemReacList file to restart the calculation (xyz needed)
- integer nallowbonds

  Number of defined bonding constraints for move-generated structures.
- character(len=2), dimension(ntypemax, 2) allowbondsatom

  Atom labels for nallowbonds bonding constraints.
- integer, dimension(ntypemax) allowbondsmax

  Maximum number of bonds associated with nallowbonds constraints.
- logical forbidgraphs

  Logical flag indicating whether to use the forbidden graphs file in allowing.
- integer nforbid

  Number of forbidden graph definitions.
- integer, dimension(nforbidmax) naforbid

  Number of atoms in each forbidden graph.
- integer, dimension(nforbidmax, namovemax, namovemax) gforbid

  Forbidden graph definitions.
- character(len=4), dimension(nforbidmax, namovemax) forbidlabel

  Labels for atoms involved in forbidden graphs.
- integer maxmolcharge

  Maximum allowed charged on a single molecule.
- integer minmolcharge

  Minimum allowed charged on a single molecule.
- integer nchargemol

  Maximum total charge at any reaction step.
- integer maxstepcharge

  Maximum number of steps inolving charge changes.
- integer maxtotalcharge

  Maximum number of steps inolving charge changes.
- integer projforcetype

  The type of projected Force used in NEB.
- logical lmoldata

  logical flag to switch the instructions to print the molecular database on
- character(len=100) pathwaysfile

  hold the file name for the pathway to the molecular database files (NEB+GDS DATA)
- integer nvalcon

  Number of atomic valence constraints for products.
- character(len=2), dimension(ntypemax) valatom

  Atom label corresponding to each set of valence constraints.
- integer, dimension(ntypemax, 2) valrange

  Valence range for each valence constraint.
- logical, dimension(ntypemax) valfz

  flag determining if this rule applies to frozen atoms bonding to non-frozen ones
- integer nrxval

  Number of atomic valence constraints for reactants.

- character(len=2), dimension(ntypemax, 2) rxvalatom

  Atom label corresponding to each set of reactant valence constraints.
- integer, dimension(ntypemax, 2) rxvalrange

  Valence ranges for each reactant valence constraint.
- logical **molecularvalency** = .false.
- integer nmv = 0

  the number of sets of molecular valence constraints
- character(2), dimension(30, 20) mvconel

  Elements in the molecular velency sets according to user dimensions (setindex,atomic number)
- integer, dimension(30, 20) mvconva

  Valency of element in in the molecular velency sets associated with mvconel dimensions (setindex,valency assiciated with that element)
- integer, dimension(30) mvconne

  number of elements in each molecular valency set
- integer, dimension(30) mxvlel

  the element with the largers number of bonds allowed, used for knowing how many times to loop for counting valences...see routine
- integer, dimension(30) ncharcon

  the maximun number of charged atoms (non-matched valency) this constrain allows for ("charged" meaning radicals dont exist in this context, ie every atom has the full shell occupation)
- logical **lewiscon** = .false.
- integer nlc = 0

  the number of sets of lewis constraints
- character(2), dimension(30, 20) lcconel

  Elements in the lewis constraint sets according to user dimensions (setindex,atomic number)
- integer, dimension(30) lcconne

  number of elements in each molecular valency set
- integer, dimension(30) nlcchar

  the maximun number of charged atoms (non-matched valency) this constrain allows for ("charged" meaning radicals dont exist in this context, ie every atom has the full shell occupation)
- logical, dimension(30, 20) lcshift = .false.

  flag determinin if this element should be shifted out of the molecule before calculating constraints
- integer ngen

  Number of GA generations.
- integer npop

  Population for GA.
- integer ncross

  Crossover events per generation.
- integer nmut

  Number of mutation events per generation.
- logical **readcore**
- integer **nacore**
- integer **nheteromax**
- integer nheteromin

  Max / min number of heteroatoms in generated structures.
- integer nelprob

  Number of element probabilities defined in input.
- character(len=2), dimension(ntypemax) elprob_label

  Atomic label for element probability.
- real(8), dimension(ntypemax) elprob

  Appearance probability for each element in structure generation.
- integer nmccxs

Maximum number of MC moves in structure generation.
- real(8) mctemperature

  Temperature for MC structure generation.
- real(8) mcbondprob

  Probability of bond creation in initial structure for MC optimization.
- real(8) nmolpenalty

  Penalty factor for having more than 1 molecule in graph during MC generation.
- real(8) ringpenalty3

  Penalty factor for having 3-rings in graph during MC generation.
- real(8) ringpenalty4

  Penalty factor for having 4-rings in graph during MC generation.
- integer nheterolimit

  Maximum number of heteroatoms in structures generated during GA.
- integer nrxn

  Maximum number of reaction steps to locate in pathfinder calculation.
- integer nmcrxn

  Number of MC annealing moves to attempt in initial graph search.
- integer nmcsearch

  Number of MC moves during search phase.
- integer nmechmove

  Maximum number of simultanerous changes in mechanism search.
- real(8) mcrxntemp

  Temperature for MC reaction-string searching.
- integer igfunc

  Graph function type.
- integer, dimension(3) atomidx

  Atoms which are aligned before NEB.
- real(8) alphavbe

  Scaling factor for bond-energy based contribution to graph-error function.
- type(graphpar) gpar

  variable type of parameters used during graphfinder...
- integer moleculemax

  Maximum number of fragments/molecules allowed to form ! seB.
- type(refresh_template) **rtmpl**
- type(vtbashop) **bashop**

20.4.1 Detailed Description

Contains global data definitions for CDE.

20.4.2 Variable Documentation

20.4.2.1 alphavbe

```
real(8) globaldata::alphavbe
```

Scaling factor for bond-energy based contribution to graph-error function.

Type definition for the parameteres used in the graphfinder algorithm

Definition at line 236 of file globaldata.f90.

20.5 io Module Reference

Main input/output functions for CDE.

Functions/Subroutines

- subroutine readinput ()

    ReadInput.
- subroutine readgraphmoves (movefile)

    ReadGraphMoves.
- subroutine readforbiddengraphs (forbidfile)

    ReadForbiddenGraphs.
- subroutine setiodefaults ()

    SetIODefaults.

20.5.1 Detailed Description

Main input/output functions for CDE.

20.5.2 Function/Subroutine Documentation

20.5.2.1 readinput()

```
subroutine io::readinput
```

ReadInput.

Reads the main CDE input file, and fills in the data in globaldata.f90.

The input filename is read as the first argument from the command-line.

Definition at line 29 of file io.f90.

20.5.2.2 readgraphmoves()

```
subroutine io::readgraphmoves (
            character(len=50) movefile )
```

ReadGraphMoves.

Reads the set of graph moves used during GDS simulation from the specified file 'movefile'.

- movefile: The file containing the graph-move information.

Definition at line 839 of file io.f90.

20.5.2.3 readforbiddengraphs()

```
subroutine io::readforbiddengraphs (
            character(len=50) forbidfile )
```

ReadForbiddenGraphs.

Reads the set of graphs which are not allowed to exist.

- forbidfile: The file containing the forbidden graph information.

Definition at line 1007 of file io.f90.

20.5.2.4 setiodefaults()

```
subroutine io::setiodefaults
```

SetIODefaults.

Sets default input parameters to "sensible" values.

Definition at line 1118 of file io.f90.

20.6 pathfinder Module Reference

Contains routines for automatic determination of a reaction-mechanism connecting two distinct (user-input) structures.

Functions/Subroutines

- subroutine runpathfinder ()

  RunPathFinder.
- subroutine adjustpaths (cx_start, cx_end, cx, nrxn, movenum, moveatoms, ifound)

  AdjustPaths.
- logical function containstargetatom (cx1, mol1, cx2, itarget)

  ContainsTargetAtom.
- subroutine evaluategrapherror (cx, cx_end, error, errflag)

  EvaluateGraphError.
- subroutine comparegraphs (cx1, cx2, error)

  CompareGraphs.
- subroutine propagategraphs (cx_start, cx, nrxn, movenum, moveatoms, errflag, error)

  PropagateGraphs.
- subroutine selectmoveatoms (imove, moveatoms, nrxn, irx, rxindex, nrx, fail, atomchange, bondchange, na, cx, cx_start)

  SelectMoveAtoms.
- subroutine pathbondenergy (cx_start, cx, nrxn, vbe, movenum, errflag)

  PathBondEnergy.
- subroutine getpathfitness (cx_start, cx_end, cx, nrxn, movenum, moveatoms, errflag, GraphError, TotalError, vbe)

  GetPathFitness.
- subroutine trimpath (cx_start, cx_end, cx, nrxn, movenum, moveatoms, iend)

  TrimPath.
- subroutine printmolsalongpath (nrxn, cx_start, cx_end, cx, movenum, chargemove, changecharges, moveatoms)

  PrintMolsAlongPath.
- subroutine graphstocoords (cx_start, cx, nrxn, printflag, printfile)

  GraphsToCoords.
- subroutine printmechanismpaths (nrxn, cx_start, cx, file_root, maxbarrier, bsum, movenum)

  PrintMechanismPaths.
- subroutine updatemechanism (nrxn, movenum, moveatoms, bondchange, atomchange, na, cx_start, cx, rxindex, cyc, movenum_store, moveatoms_store)

  UpdateMechanism.
- subroutine reactionerrorcorrection (cx_start, cx, nrxn, movenum, moveatoms, errx)

  ReactionErrorCorrection.
- subroutine updatecharges (nrxn, cx, chargemove, chargemove_store, errflag)

  UpdateCharges.

20.6.1 Detailed Description

Contains routines for automatic determination of a reaction-mechanism connecting two distinct (user-input) structures.

The method employed here has been reported in:

Automatic Proposal of Multistep Reaction Mechanisms using a Graph-Driven Search, I. Ismail and H. B. V. A. Stuttaford-Fowler and C. Ochan Ashok and C. Robertson and S. Habershon, J Phys Chem A 123 3407-3417 (2019)

Fast screening of homogeneous catalysis mechanisms using graph-driven searches and approximate quantum chemistry, C Robertson, S Habershon, Catalysis Science & Technology, 9, 6357 (2019)

Bibtex records:

@article{Habershon:2019ab, Author = {Ismail, Idil and Stuttaford-Fowler, Holly B V A and Ochan Ashok, Curtis and Robertson, Christopher and Habershon, Scott}, Journal = {J Phys Chem A}, Month = {Apr}, Number = {15}, Pages = {3407-3417}, Title = {Automatic Proposal of Multistep Reaction Mechanisms using a Graph-Driven Search}, Volume = {123}, Year = {2019}}

@article{Habershon:2019ee, Author = {Robertson, Christopher and Habershon, Scott}, Date-Added = {2019-12-03 15:41:20 +0000}, Date-Modified = {2019-12-03 15:41:29 +0000}, Doi = {10.1039/C9CY01997A}, Issue = {22}, Journal = {Catal. Sci. Technol.}, Pages = {6357-6369}, Publisher = {The Royal Society of Chemistry}, Title = {Fast screening of homogeneous catalysis mechanisms using graph-driven searches and approximate quantum chemistry}, Url = { http://dx.doi.org/10.1039/C9CY01997A}, Volume = {9}, Year = {2019}, }

20.6.1.1 DESCRIPTION

Given input xyz files for target reactants and products, this routine will try to find a reaction mechanism connecting the two by using graph-moves defined in the "movefile".
After determination of a path which definitively connects reactants and products, we then generate xyz-files for each reaction-path along the reaction-mechanism using the idea of optimization under the graph-restraining potential.
If optaftermove = .TRUE. in the input file, this code will also perform geometry optimization for all of the intermediate structures and report the results in the .log file.

20.6.1.2 OUTPUTS

After completion, the following outputs are worth looking at:

- mcopt.dat: contains the graph-error function as a function of number of MC iterations in the calculation. This should reach zero for a successful calculation.
- final_path.xyz: Contains the intermediate structures (xyz format) along the located reaction mechanism. If optaftermove = .true., the structures are geometry-optimized using whichever electronic structure methods is specified in the input file.
- final_path_rx_$*$.xyz: This series of xyz files, one for each reaction-step in the mechanism, contain initial nimage-long paths for each intermediate reaction.
- .log: The .log file will contain useful information about the reaction, including the calculated Q-value and the maximum dE value.

In addition, if igfunc = 4 (so we are targetting formation of a single molecule or subset of molecules, rather than an entire structure), the further outputs are produced which have the prefix 'adjusted_'. This are files which have been determined for the reaction path after removing molecules and reactions which are not relevant to the formation of the product.

20.6.2 Function/Subroutine Documentation

20.6.2.1 runpathfinder()

```
subroutine pathfinder::runpathfinder
```

RunPathFinder.
Runs automatic determination of a reaction-path connecting two structures.
Definition at line 108 of file pathfinder.f90.

20.6.2.2 adjustpaths()

```
subroutine pathfinder::adjustpaths (
              type(cxs) cx_start,
              type(cxs) cx_end,
              type(cxs), dimension(nrxn) cx,
              integer nrxn,
              integer, dimension(nrxn) movenum,
              integer, dimension(nrxn,namovemax) moveatoms,
              integer ifound )
```

AdjustPaths.
This routine performs a series of functions to simplify the post-analysis of reaction-path simulations. (i) If pesfull = .FALSE., we print out the energies and Q-values calculated only for those molecules involved in formation of the product. (ii) We print out a new set of reaction paths which only involve the molecules containing atoms ending up in the final structure. (iii) We strip the adjusted paths of non-reactive molecules.
The final set of xyz path files should then be most appropriate for post-analysis.
Input parameters:

- cx_start: CXS object for reactants.
- cx_end: CXS object for target product.
- cx(nrxn): Intermediate CXS objects along mechanism path.
- nrxn: Maximum number of active reactions.
- movenum(:): Graph-move number at each reaction step.
- moveatoms(:,:): Atoms involved in graph-moves at each step.
- ifound: Reaction step-number at which target products are first found.

Definition at line 507 of file pathfinder.f90.

20.6.2.3 containstargetatom()

```
logical function pathfinder::containstargetatom (
              type(cxs) cx1,
              integer mol1,
              type(cxs) cx2,
              integer itarget )
```

ContainsTargetAtom.
Logical function to check whether cx1 contains any of the atoms in molecule number "itarget" in cx2.
Definition at line 997 of file pathfinder.f90.

20.6.2.4 evaluategrapherror()

```
subroutine pathfinder::evaluategrapherror (
              type(cxs) cx,
              type(cxs) cx_end,
              real(8) error,
              logical errflag )
```

EvaluateGraphError.
Calculates the error measure between the final graph cx and the target graph cx_end. The error is defines as the sum of squared differences for each graph element. If errflag = .TRUE., it means that the reaction path itself is not valid and we assign it and arbitrarily large error.
cx - Final graph in reaction string. cx_end - Target graph for reaction string. error - Calculated graph error. errflag - Logical flag indicating whether the reaction string is valid or not.
Definition at line 1032 of file pathfinder.f90.

20.6.2.5 comparegraphs()

```
subroutine pathfinder::comparegraphs (
            type(cxs) cx1,
            type(cxs) cx2,
            real(8) error )
```

CompareGraphs.

Compare two separate structures via their graphs. In particular, we compare the eigenvalues of the atomic-weighted graph Coulomb matrix, with the distances between atoms given by the shortest-path distances as calculated from the graph.

cx1, cx2 - Two structures to compare. Note that the graphs MUST have been already calculated. error - Calculate mean-square error between eigenvalues.

Definition at line 1075 of file pathfinder.f90.

20.6.2.6 propagategraphs()

```
subroutine pathfinder::propagategraphs (
            type(cxs) cx_start,
            type(cxs), dimension(nrxn) cx,
            integer nrxn,
            integer, dimension(nrxn) movenum,
            integer, dimension(nrxn,namovemax) moveatoms,
            logical errflag,
            real(8) error )
```

PropagateGraphs.

Using the string of nrxn graph moves and their atom labels, this routine starts from the connectivity graph of cx_start and generates the sequence of nrxn graphs.

cx_start - Chemical structure object, contains the starting graph. cx(1:nrxn) - Sequence of chemical structure objects generated by graph moves. nrxn - Number of moves in the reaction string. movenum(:) - Integer move number for reaction i. moveatoms(:,:) - Indices of atoms involved in move i.

Definition at line 1441 of file pathfinder.f90.

20.6.2.7 selectmoveatoms()

```
subroutine pathfinder::selectmoveatoms (
            integer imove,
            integer, dimension(nrxn,namovemax) moveatoms,
            integer nrxn,
            integer irx,
            integer, dimension(namax) rxindex,
            integer nrx,
            logical fail,
            logical, dimension(na) atomchange,
            logical, dimension(na,na) bondchange,
            integer na,
            type(cxs), dimension(nrxn) cx,
            type(cxs) cx_start )
```

SelectMoveAtoms.

For move number imove, this subroutine selects new atoms for the move using the pre-defined list of reactive atom indices in rxindex.

imove - A given movenumber. This corresponds to the index of a move in the movefile. moveatoms(nrx,NAMOVE↩
MAX) - Indices of atoms which are moved during a given graph-move. nrxn - Number of reactions in string. irxn - The reaction number which we're trying to change. rxindex(MAMAX) - Indices of reactive atoms in current graph. This array must be previously determined using SetReacctiveIndices() from module gds. nrx - Number of reactive atoms stored in rxindex(:). fail - logical flag indicating whether or not this subroutine has successfully managed to find new atoms for the move imove. atomchange(na) - Logical array indicating which atoms can react. bondchange(na,na) - Logical array indicating which bonds can change. na - Number of atoms. cx(nrxn) - Chemical structure objects for

each node in reaction string. cx_start - Chemical structure object for start configuration.
Definition at line 1703 of file pathfinder.f90.

20.6.2.8 pathbondenergy()

```
subroutine pathfinder::pathbondenergy (
            type(cxs) cx_start,
            type(cxs), dimension(nrxn) cx,
            integer nrxn,
            real(8) vbe,
            integer, dimension(nrxn) movenum,
            logical errflag )
```

PathBondEnergy.
Using a simple additive model, calculate the bonding energy along the entire path, using typical bond energies.... cx_start - Starting chemical structure. cx(:) - The set of structures along the path. nrxn - Number of reactions in string. vbe - Returned additive energy.
Definition at line 1926 of file pathfinder.f90.

20.6.2.9 getpathfitness()

```
subroutine pathfinder::getpathfitness (
            type(cxs) cx_start,
            type(cxs) cx_end,
            type(cxs), dimension(nrxn) cx,
            integer nrxn,
            integer, dimension(nrxn) movenum,
            integer, dimension(nrxn,namovemax) moveatoms,
            logical errflag,
            real(8) GraphError,
            real(8) TotalError,
            real(8) vbe )
```

GetPathFitness.
Calculate the fitness value associated with a given reaction mechanism.
This TotalError is defined as the sum of a GraphError, describing how far away a propagated graph is from a target graph, as well as a vbe contribution which includes energy contributions to the path fitness.
Note that the energy contribution is only calculated when alphavbe $>$ 1d-3.
cx_start - The chemical structure object for the reactants. cx_end - The chemical structure object for the products. cx(:) - Chemical structure objects for intermediates along path. nrxn - Number of reactions in path. novenum(nrxn) - Move numbers for each reaction in the path. moveatomos(nrxn,NAMOVEMAX) - Atoms which are acted upon in each reaction. errflag - Logical flag indicating whether or not a chemically-sensible reaction path has been generated. GraphError - Error term arising due to differences between propagated graph and target graph. Total↩
Error - GraphError + alphavbe $*$ vbe. vbe - Calculated bonding energy.
Definition at line 2068 of file pathfinder.f90.

20.6.2.10 trimpath()

```
subroutine pathfinder::trimpath (
            type(cxs) cx_start,
            type(cxs) cx_end,
            type(cxs), dimension(nrxn) cx,
            integer nrxn,
            integer, dimension(nrxn) movenum,
            integer, dimension(nrxn,namovemax) moveatoms,
            integer iend )
```

TrimPath.

When the target molecule is detected in a reaction, the rest of the movenums are set to zero. In other words, if the target is found at some reaction step before the end, we then set the rest of the reactions to zero because we don't care what they are any more..
cx_start - The chemical structure object for the reactants. cx_end - The chemical structure object for the products. cx(:) - Chemical structure objects for intermediates along path. nrxn - Number of reactions in path. novenum(nrxn) - Move numbers for each reaction in the path. moveatoms(nrxn,NAMOVEMAX) - Atoms which are acted upon in each reaction.
Definition at line 2118 of file pathfinder.f90.

20.6.2.11 printmolsalongpath()

```
subroutine pathfinder::printmolsalongpath (
            integer nrxn,
            type(cxs) cx_start,
            type(cxs) cx_end,
            type(cxs), dimension(nrxn) cx,
            integer, dimension(nrxn) movenum,
            integer, dimension(nrxn,nmolmax) chargemove,
            logical changecharges,
            integer, dimension(nrxn,namovemax) moveatoms )
```

PrintMolsAlongPath.
Outputs (to logfile) the molecular structure present at each step along a reaction-mechanism,so that the user can get a feel for what is going on in the path.
nrxn - Number of reactions in path. cx_start - The chemical structure object for the reactants. cx_end - The chemical structure object for the products. cx(:) - Chemical structure objects for intermediates along path. movenum(nrxn) - Move numbers for each reaction in the path.
Definition at line 2176 of file pathfinder.f90.

20.6.2.12 graphstocoords()

```
subroutine pathfinder::graphstocoords (
            type (cxs) cx_start,
            type (cxs), dimension(nrxn) cx,
            integer nrxn,
            logical printflag,
            character(len=*) printfile )
```

GraphsToCoords.
Converts a sequence of connectivity graphs to xyz coordinates by optimization under the action of the graph-restraining potential.
The coordinates of the reactant structure are contained in cx_start; these are used as the starting point for a sequence of geometry optimizations. The resulting structures obey the target bonding graph at each step.
If optaftermove = .TRUE. in the input file, then geometry optimization of the structures is also performed AFTER graph-potential optimization.
cx_start - Chemical structure object for reactants. cx(nrxn) - Chemical structures along the path. nrxn - Number of reactions along the path. printflag - Logical flag indicating whether to print out xyz files or not. printfile - If printflag = .TRUE., printfile indicates the file to print to.
Definition at line 2244 of file pathfinder.f90.

20.6.2.13 printmechanismpaths()

```
subroutine pathfinder::printmechanismpaths (
            integer nrxn,
            type(cxs) cx_start,
            type(cxs), dimension(nrxn) cx,
            character (len=20) file_root,
            real(8) maxbarrier,
            real(8) bsum,
            integer, dimension(nrxnmax) movenum )
```

PrintMechanismPaths.
Prints out nimage intermediate snapshots for each of the generated reactions in the final reaction-path.
Note that the end-point coordinates for each reaction are assumed to be stored in cx_start and cx(nrxn).
We use linear interpolation between these end-points to generate internal images.
The results are stored to the files final_path_rx_∗.xyz.
nrxn - Number of reactions in mechanism. cx_start - Chemical structure object for reactants. cx(nrxn) - Chemical structure objects for products of each reaction. These come from the subroutine GraphsToCoords(). file_root - Root name of printing file.
Definition at line 2433 of file pathfinder.f90.

20.6.2.14 updatemechanism()

```
subroutine pathfinder::updatemechanism (
            integer nrxn,
            integer, dimension(nrxn) movenum,
            integer, dimension(nrxn,namovemax) moveatoms,
            logical, dimension(na,na) bondchange,
            logical, dimension(na) atomchange,
            integer na,
            type(cxs) cx_start,
            type(cxs), dimension(nrxn) cx,
            integer, dimension(namax) rxindex,
            logical cyc,
            integer, dimension(nrxn) movenum_store,
            integer, dimension(nrxn,namovemax) moveatoms_store )
```

UpdateMechanism.
Performs a trial move by modifying a current mechanism string by either changing a move and its atoms, or just changing the atoms of an exisiting move.
Note that the end-point coordinates for each reaction are assumed to be stored in cx_start and cx(nrxn).
nrxn - Number of reactions in mechanism. cx_start - Chemical structure object for reactants. cx(nrxn) - Chemical structure objects for products of each reaction. These come from the subroutine GraphsToCoords().
Definition at line 2538 of file pathfinder.f90.

20.6.2.15 reactionerrorcorrection()

```
subroutine pathfinder::reactionerrorcorrection (
            type(cxs) cx_start,
            type(cxs), dimension(nrxn) cx,
            integer nrxn,
            integer, dimension(nrxn) movenum,
            integer, dimension(nrxn,namovemax) moveatoms,
            logical errx )
```

ReactionErrorCorrection.
Performs error correction using geometry-optimized structures, making sure that the reactions listed in movenum and moveatoms are actually correct for a given reaction string.
Note that the end-point coordinates for each reaction are assumed to be stored in cx_start and cx(nrxn).
Definition at line 2704 of file pathfinder.f90.

20.6.2.16 updatecharges()

```
subroutine pathfinder::updatecharges (
            integer nrxn,
            type(cxs), dimension(nrxn) cx,
            integer, dimension(nrxn,nmolmax) chargemove,
            integer, dimension(nrxn,nmolmax) chargemove_store,
            logical errflag )
```

UpdateCharges.
Update the charges on molecules along the reaction path. These moves are performed in addition to reaction changes.
Definition at line 3014 of file pathfinder.f90.

20.7 pathopt Module Reference

Contains routines for path optimization, including CINEB.

Functions/Subroutines

- subroutine optimizepath (rp)
  OptimizePath.
- subroutine cineb (rp)
  CINEB.

20.7.1 Detailed Description

Contains routines for path optimization, including CINEB.

20.7.2 Function/Subroutine Documentation

20.7.2.1 optimizepath()

```
subroutine pathopt::optimizepath (
            type(rxp) rp )
```

OptimizePath.
Initiates a path optimization calculation.

- rp: Initial reaction path.
- nebiter: number of NEB iterations
- neboutfreq: number of NEB iterations between outputs
- nebspring: NEB spring parameter in atomic units
- pathoptmethod: Method used to optimize reaction path. Allowed values are 'NEB'....
- NEBconv - Convergence threshold on average force for NEB calculations.
- CIthresh - Average force modulus when climbing-image should be activated.
- inputfile: The input filename, used to generate further output files.
- logfile: Log file unit nunmber for output

Definition at line 41 of file pathopt.f90.

20.7.2.2 cineb()

```
subroutine pathopt::cineb (
            type(rxp) rp )
```

CINEB.
Runs a climbing-image nudged elastic band calculation starting from an initial reaction-path defined in rp.

- rp: The reaction-path object to be refined.

Definition at line 77 of file pathopt.f90.

20.8 pes Module Reference

Potential energy surface calculations.

Functions/Subroutines

- subroutine setupenergycalc (PEStype, PESfile, PESExecutable)

    SetupEnergyCalc.
- subroutine setupgeomopt (PESOpttype, PESOptfile, PESOptExecutable)

    SetupGeomOpt.
- subroutine abinitio (cx, abtypein, success)

    AbInitio.
- subroutine molprocalc (cx, abtype, success)

    MOLPROcalc.
- subroutine orcacalc (cx, abtype, success)

    ORCAcalc.
- subroutine dftbcalc (cx, minimize, success)

    DFTBcalc.
- subroutine uffcalc (cx, minimize, success)

    UFFcalc.
- subroutine lammpscalc (cx, abtype, success)

    LAMMPScalc.
- subroutine psi4calc (cx, minimize, success)

    PSI4calc.
- subroutine **readforcesindividually** (cx, dummyLines)
- subroutine **readforcestogether** (cx, dummyLines)
- subroutine **readenergy** (cx, dummyLines)
- subroutine **readhessian** (cx, fileName)
- subroutine **readoptimizedcoordinates** (cx, fileName)
- subroutine **createfiletemplate** (minimize, exec, n, buffer, iline)
- subroutine **executecalculation** (str, success)
- subroutine **extractdata** (str)
- subroutine fireminimise (cx, iterin, gdsrestspring_in, nbstrength_in, nbrange, kradius, nebrestrend)

    FireMinimise.
- subroutine **mincxsenergynebrest** (cx, success)
- subroutine **mincxsenergy** (cx, success)
- subroutine matchcxconstraint (cx, mcx, kspring)

    MatchCXContraints.
- subroutine getactmolecenergies (cx, MolecAct, nactmol, molen)

    GetActMolecularEnergies.
- subroutine getmolecularenergies (cx)

    GetMolecularEnergies.

Variables

- integer, parameter **nlinemax** = 200
- character(len=6) **vtype**
- character(len=25) **vfile**
- character(len=6) **vopttype**
- character(len=25) **voptfile**
- integer **nline**
- integer **nlineopt**
- character(len=100), dimension(nlinemax) **peslines**

- character(len=100) **pesexec**
- character(len=100) **pesoptexec**
- character(len=100), dimension(nlinemax) **pesoptlines**
- integer **coordsline**
- integer **optcoordsline**

20.8.1 Detailed Description

Potential energy surface calculations.
Defines subroutines for evaluating energies and forces using various external codes.

20.8.2 Function/Subroutine Documentation

20.8.2.1 setupenergycalc()

```
subroutine pes::setupenergycalc (
            character PEStype,
            character PESfile,
            character PESExecutable )
```

SetupEnergyCalc.
Initializes the PES calculation.

- PEStype: Identifies the type of PES calculation, 'ORCA'...
- PESfile: Identifies the file containing the template input file for PES evaluation. This should have the characters 'XXX' in the line where the coordinates will be input.

Definition at line 47 of file pes.f90.

20.8.2.2 setupgeomopt()

```
subroutine pes::setupgeomopt (
            character PESOpttype,
            character PESOptfile,
            character PESOptExecutable )
```

SetupGeomOpt.
Initializes the geometry optimisation calculation.

- PESopttype: Identifies the PES for geometry optimisation, 'ORCA'...
- PESoptfile: Identifies the file containing the template input file for PES evaluation. This should have the characters 'XXX' in the line where the coordinates will be input.

Definition at line 104 of file pes.f90.

20.8.2.3 abinitio()

```
subroutine pes::abinitio (
            type(cxs) cx,
            character(len=4), intent(in) abtypein,
            logical success )
```

AbInitio.
Calculates Ab-initio properties for the chemical structure in cx.

- cx: Chemical structure object.

- abtype: 'ener' = energy calculation ; 'grad' = energy+gradient 'hess' = energy+gradient+hessian ; 'optg' = optimization
- success: A logical flag indicating whether or not the PES evaluation was successful. For example, might not have converged the SCF cycles in DFT/HF, etc.

Definition at line 171 of file pes.f90.

20.8.2.4 molprocalc()

```
subroutine pes::molprocalc (
             type(cxs) cx,
             character (len=4) abtype,
             logical success )
```

MOLPROcalc.
Performs a single-point energy and force calculation using MOLPRO.

- cx: Chemical structure object.
- abtype: indicates what time of calculation we shold perform

Definition at line 349 of file pes.f90.

20.8.2.5 orcacalc()

```
subroutine pes::orcacalc (
             type(cxs) cx,
             character(len=4) abtype,
             logical success )
```

ORCAcalc.
Performs a single-point energy and force calculation using ORCA.

- cx: Chemical structure object.
- abtype: indicates what type of calculation we shold perform
- success: return flag indicating non-convergence of calculation if success = .FALSE.

Definition at line 521 of file pes.f90.

20.8.2.6 dftbcalc()

```
subroutine pes::dftbcalc (
             type(cxs) cx,
             logical minimize,
             logical success )
```

DFTBcalc.
Performs a single-point energy and force calculation using DFTB. Note that the appropriate Slater-Koster files should be present in the run directory.

- cx: Chemical structure object.
- minimize: logical flag indicating whether or not to run a geometry optimization calculation.
- success: Flag indicating success of calculation.

Definition at line 816 of file pes.f90.

20.8.2.7 uffcalc()

```
subroutine pes::uffcalc (
             type(cxs) cx,
             logical minimize,
             logical success )
```

UFFcalc.
Performs a single-point energy and force calculation using UFF, as implemented in openbabel (obabel).

- cx: Chemical structure object.
- minimize: logical flag indicating whether or not to run a geometry optimization calculation.

  For openbabel, we must have minimize = .TRUE.
- success: Flag indicating success of calculation.

DEPENDENCIES: Requires babel and obminimize.
Definition at line 974 of file pes.f90.

20.8.2.8 lammpscalc()

```
subroutine pes::lammpscalc (
             type(cxs) cx,
             character (len=4) abtype,
             logical success )
```

LAMMPScalc.
Performs a single-point energy and force calculation using LAMMPS.

- cx: Chemical structure object.
- abtype: indicates what time of calculation we shold perform

Definition at line 1065 of file pes.f90.

20.8.2.9 psi4calc()

```
subroutine pes::psi4calc (
             type(cxs) cx,
             logical minimize,
             logical success )
```

PSI4calc.
Performs a single-point energy and force calculation using psi4.

- cx: Chemical structure object.
- minimize: logical flag indicating whether or not to run a geometry optimization calculation.

Definition at line 1308 of file pes.f90.

20.8.2.10 fireminimise()

```
subroutine pes::fireminimise (
             type(cxs) cx,
             integer iterin,
             real(8) gdsrestspring_in,
             real(8) nbstrength_in,
             real(8) nbrange,
             real(8) kradius,
             logical nebrestrend )
```

FireMinimise.
minimises the energy using the FIRE algorithm, incorporating the graph constraining ppotentials if nebrestend = .true. the last bit of the minimization, we switch to quickmin, optional

- cx: Chemical structure object to be optimized
- iter: returns the number of iterations it took to optimize
- all other: parameteres for the graph restraining potential.

Definition at line 1651 of file pes.f90.

20.8.2.11 matchcxconstraint()

```
subroutine pes::matchcxconstraint (
            type(cxs) cx,
            type(cxs) mcx,
            real(8) kspring )
```

MatchCXContraints.
provides a force proportional to the difference btween the coordinates of the two structures cx and mcx

- cx: A chemical structure object.
- kspring: Spring constant for the bonding restraint term.

Definition at line 1791 of file pes.f90.

20.8.2.12 getactmolecenergies()

```
subroutine pes::getactmolecenergies (
            type(cxs) cx,
            integer, dimension(nactmol) MolecAct,
            integer nactmol,
            double precision, dimension(nactmol), optional molen )
```

GetActMolecularEnergies.
Get the energies of each molecule in the reactants and produts of rp for molecules that are "active"

- cx: chemical structure.
- MolecAct: array with the molecue indecies
- nactmol: number of molecules in MolecAct array
- molen: array with energies
- filenam: The filename to be outputed

Definition at line 1821 of file pes.f90.

20.8.2.13 getmolecularenergies()

```
subroutine pes::getmolecularenergies (
            type(cxs) cx )
```

GetMolecularEnergies.
Get the energies of each molecule in the reactants and produts of rp

- cx: chemical structure.
- filenam: The filename to be outputed

Definition at line 1851 of file pes.f90.

20.9 rpath Module Reference

Reaction-path object definition.

Data Types

- type rxp

    Type definition for the reaction-path object.

Functions/Subroutines

- subroutine newpath (rp, startfrompath, startfile, endfile, pathfile, nimage, pathinit, emptypath, natom)

    NewPath.
- subroutine copypath (rpin, rpout)

    CopyPath.
- subroutine deletepath (rp)

    DeletePath.
- subroutine setinternalcoords (rp, pathinit)

    SetInternalCoords.
- subroutine printpathtofile (rp, filename)

    PrintPathToFile.
- subroutine findidpppath (rp, IDPPIter, IDPPConv, IDPPStep, IDPPspring)

    FindIDPPPath.
- subroutine **getidppforces** (rp, rtarg)
- subroutine getforcenorm (rp, forcenorm, fmax, n1, n2)

    GetForceNorm.
- subroutine readxyzframe (fileid, na, label, x, y, z)

    ReadXYZFrame.
- subroutine startfromendpoints (rp, startfile, endfile, pathinit, nimage)

    StartFromEndPoints.
- subroutine readpathfromfile (rp, pathfile)

    ReadPathFromFile.
- subroutine setpathconstraints (rp, NDOFconstr, FixedDOF, Natomconstr, Fixedatom)

    SetPathConstraints.
- subroutine getpathenergy (rp, success)

    GetPathEnergy.
- subroutine getpathgradients (rp, success)

    GetPathGradients.
- subroutine getprojforces1 (rp, kon, ci_flag, endflag)

    GetProjForces1.
- subroutine getprojforces2 (rp, kon, ci_flag, endflag)

    GetProjForces2.
- subroutine **variablesprings** (rp)
- subroutine getprojforces3 (rp, kon, ci_flag, endflag)

    GetProjForces3.
- subroutine shimmyendbeads (rp, shimmied, beadcount, pend)

    ShimmyEndBeads.
- subroutine getfourierforces (rp)

    GetFourierForces.
- subroutine fouriertopath (rp)

    FourierToPath.
- subroutine pathtofourier (rp)

PathToFourier.

- subroutine firstvvupdate (rp, timestep, mass)

  FirstVVUpdate.

- subroutine secondvvupdate (rp, timestep, mass)

  SecondVVUpdate.

- subroutine getspringforces (rp, kspring)

  GetSpringForces.

- subroutine stripinactivefrompath (rp, stripfile, FixedDOF, FixedAtom, NDOFconstr, Natomconstr, Update↵Cons)

  StripInactiveFromPath.

- real(8) function errorr (one, two, na)

  errorr calculates the absolute difference, elementwise between two matrices.

20.9.1 Detailed Description

Reaction-path object definition.
Defines the reaction-path object type. This object is a string of nimage snapshots along a reaction-path.

20.9.2 Function/Subroutine Documentation

20.9.2.1 newpath()

```
subroutine rpath::newpath (
             type(rxp) rp,
             logical, intent(in) startfrompath,
             character, dimension(*), intent(in) startfile,
             character, dimension(*), intent(in) endfile,
             character, dimension(*), intent(in) pathfile,
             integer nimage,
             character, dimension(*), intent(in) pathinit,
             logical, intent(in) emptypath,
             integer natom )
```

NewPath.
Initializes a path object.

- rp: Initialized path object.
- startfrompath: Flag to indicate that we're starting from a full path (logical)
- startfile: filename for starting point structure (xyz file)
- endfile: filename for end point structure (xyz file)
- pathfile: If startfrompath == .TRUE., read the full path from pathfile
- nimage: number of images along path in total (read from input file OR from pathfile)
- pathinit: path initialization type when startfrompath = .FALSE.. Allowed values are 'LINEAR' and ....?
- emptypath: Flag indicating whether to start with an empty path.
- natom: number of atoms.

Definition at line 59 of file rpath.f90.

20.9.2.2 copypath()

```
subroutine rpath::copypath (
            type(rxp) rpin,
            type(rxp) rpout )
```

CopyPath.
Creates an empty reaction path, rpout, using parameter sizes from rpin

- rpin: The reaction path object used to get size information.
- rpout: The new reaction path object.

Definition at line 127 of file rpath.f90.

20.9.2.3 deletepath()

```
subroutine rpath::deletepath (
            type(rxp) rp )
```

DeletePath.
Deletes memory associated with a path.

- rp: The reaction path object to be deleted.

Definition at line 154 of file rpath.f90.

20.9.2.4 setinternalcoords()

```
subroutine rpath::setinternalcoords (
            type(rxp) rp,
            character, dimension(*) pathinit )
```

SetInternalCoords.
When building a path from end-points only, this subroutine sets the coordinates of the nimage-2 internal beads according to the method defined in pathinit.

- rp: The reaction path object.
- pathinit: string defining interpolation type. Allowed values are 'LINEAR' for linear interpolation..

Definition at line 195 of file rpath.f90.

20.9.2.5 printpathtofile()

```
subroutine rpath::printpathtofile (
            type(rxp) rp,
            character, dimension(*) filename )
```

PrintPathToFile.
Outputs an entire reaction path to an xyz file.

- rp: The reaction path object.
- filename: the output xyz file.

Definition at line 237 of file rpath.f90.

20.9.2.6 findidpppath()

```
subroutine rpath::findidpppath (
            type(rxp) rp,
            integer IDPPIter,
            real(8) IDPPConv,
            real(8) IDPPStep,
            real(8) IDPPspring )
```

FindIDPPPath.
finds an interpolated path using NEB on a Image-dependent PairPotential, as described in Smidstrup et al J Chem phys 2014, 140

- rp: Reaction path

Definition at line 274 of file rpath.f90.

20.9.2.7 getforcenorm()

```
subroutine rpath::getforcenorm (
            type(rxp) rp,
            real(8) forcenorm,
            real(8) fmax,
            integer n1,
            integer n2 )
```

GetForceNorm.
Calculate the norm of the projected forces for the internal beads.

- rp: The reaction-path object to be refined.
- forcenorm: Returned force norm
- n1, n2: Integer range of images to include in calculation.

Definition at line 430 of file rpath.f90.

20.9.2.8 readxyzframe()

```
subroutine rpath::readxyzframe (
            integer fileid,
            integer na,
            character*2, dimension(*) label,
            real(8), dimension(*) x,
            real(8), dimension(*) y,
            real(8), dimension(*) z )
```

ReadXYZFrame.
reads a single frame from an xyz file.

- fileid: Integer file label.
- na: Number of atoms
- label: atom labes read from file.
- x,y,z: Coordinates of each atom read from file

Definition at line 482 of file rpath.f90.

20.9.2.9 startfromendpoints()

```
subroutine rpath::startfromendpoints (
            type(rxp) rp,
            character, dimension(*) startfile,
            character, dimension(*) endfile,
            character, dimension(*) pathinit,
            integer nimage )
```

StartFromEndPoints.
Initializes a reaction path using the path end-points.

- rp: The reaction path object.
- startfile: the xyz file containing the startpoint.
- endfile: the xyz file containing the endpoint.
- pathinit: character∗5 string containing identifier for method used to interpolate internal image coordinates.
- nimage: Number of images in path.

Definition at line 519 of file rpath.f90.

20.9.2.10 readpathfromfile()

```
subroutine rpath::readpathfromfile (
            type(rxp) rp,
            character, dimension(*) pathfile )
```

ReadPathFromFile.
Reads a file reaction path from a xyz file with nimage frames. First, we read the number of atoms and images, then create the path object. Finally, we read in the coordinates along the path.

- rp: Reaction path object.
- pathfile: filename for path.

Definition at line 584 of file rpath.f90.

20.9.2.11 setpathconstraints()

```
subroutine rpath::setpathconstraints (
            type(rxp) rp,
            integer, intent(in) NDOFconstr,
            integer, dimension(*), intent(in) FixedDOF,
            integer, intent(in) Natomconstr,
            integer, dimension(*), intent(in) Fixedatom )
```

SetPathConstraints.
Identify constrained atoms and DOFs in the path.

- rp: Initialized path object.
- NDOFconstr: Number of DOF constraints.
- FixedDOF: Array containing integer number of constrained DOFs
- Natomconstr: Number of atom constraints
- Fixedatom: Array containing integer ids of fixed atoms.

Definition at line 683 of file rpath.f90.

20.9.2.12 getpathenergy()

```
subroutine rpath::getpathenergy (
            type(rxp) rp,
            logical success )
```

GetPathEnergy.
Calculates the energy for all structures along a reaction path. The pes.f90 module contains details of the calculation to perform.

- rp: Path object.

Definition at line 710 of file rpath.f90.

20.9.2.13 getpathgradients()

```
subroutine rpath::getpathgradients (
            type(rxp) rp,
            logical success )
```

GetPathGradients.
Calculates the energy for all structures along a reaction path. The pes.f90 module contains details of the calculation to perform.

- rp: Path object.

Definition at line 740 of file rpath.f90.

20.9.2.14 getprojforces1()

```
subroutine rpath::getprojforces1 (
            type (rxp) rp,
            logical kon,
            logical ci_flag,
            logical endflag )
```

GetProjForces1.
Calculates the forces projected along the reaction path, as required in NEB or CINEB calculations. A previous PES calculation MUST have been performed for the reaction path.
Note: By setting ci_flag = .true., this routine returns forces for CI-NEB.

- rp: Reaction path object.
- kon: Flag determining if the spring forces should be calculated or not
- ci_flag: Logical flag indicating whether to return climbing-image forces.
- endflag: Logical flag indicatin whether or not the forces on the end-point beads should be returned as the "bare" (non-NEB) forces.

Definition at line 783 of file rpath.f90.

20.9.2.15 getprojforces2()

```
subroutine rpath::getprojforces2 (
            type (rxp) rp,
            logical kon,
            logical ci_flag,
            logical endflag )
```

GetProjForces2.
Calculates the forces projected along the reaction path, as required in NEB or CINEB calculations. A previous PES calculation MUST have been performed for the reaction path. This routine uses a slightly improved version of tangents and forces as described by henkelman 00
Note: By setting ci_flag = .true., this routine returns forces for CI-NEB.

- rp: Reaction path object. size is made.
- ci_flag: Logical flag indicating whether to return climbing-image forces.
- endflag: Logical flag indicatin whether or not the forces on the end-point beads should be returned as the "bare" (non-NEB) forces.

Definition at line 1166 of file rpath.f90.

20.9.2.16 getprojforces3()

```
subroutine rpath::getprojforces3 (
            type (rxp) rp,
            logical kon,
            logical ci_flag,
            logical endflag )
```

GetProjForces3.
Calculates the forces projected along the reaction path, as required in NEB or CINEB calculations. This routine uses improves the improvement in henkelmans two 00 papers as described in Kolsbjerg and Hammer 16
Note: By setting ci_flag = .true., this routine returns forces for CI-NEB.

- rp: Reaction path object. size is made.
- ci_flag: Logical flag indicating whether to return climbing-image forces.
- endflag: Logical flag indicatin whether or not the forces on the end-point beads should be returned as the "bare" (non-NEB) forces.

Definition at line 1340 of file rpath.f90.

20.9.2.17 shimmyendbeads()

```
subroutine rpath::shimmyendbeads (
            type (rxp) rp,
            logical shimmied,
            integer beadcount,
            integer pend )
```

ShimmyEndBeads.

- rp: Reaction path object.
- kspring: NEB spring constant (in au), used as a reference for variable spring size is made.

Definition at line 1455 of file rpath.f90.

20.9.2.18 getfourierforces()

```
subroutine rpath::getfourierforces (
            type(rxp) rp )
```

GetFourierForces.
Calculates the Fourier expansion coefficients for the path rp.

- rp: Input reaction-path object.

Definition at line 1573 of file rpath.f90.

20.9.2.19 fouriertopath()

```
subroutine rpath::fouriertopath (
            type(rxp) rp )
```

FourierToPath.
Calculates the path positions using the path Fourier coefficients.

- rp: Input reaction-path object.

Definition at line 1638 of file rpath.f90.

20.9.2.20 pathtofourier()

```
subroutine rpath::pathtofourier (
            type(rxp) rp )
```

PathToFourier.
Calculates the Fourier coefficiencts from path positions.

- rp: Input reaction-path object.

Definition at line 1685 of file rpath.f90.

20.9.2.21 firstvvupdate()

```
subroutine rpath::firstvvupdate (
            type(rxp) rp,
            real(8) timestep,
            real(8) mass )
```

FirstVVUpdate.
Evolves the Fourier coefficients and momenta through first half of a Velocity-Verlet time-step.

- rp: Reaction path object
- timestep: obvs...
- mass: Fourier coefficient mass.

Definition at line 1731 of file rpath.f90.

20.9.2.22 secondvvupdate()

```
subroutine rpath::secondvvupdate (
            type(rxp) rp,
            real(8) timestep,
            real(8) mass )
```

SecondVVUpdate.
Evolves the Fourier coefficients and momenta through second half of a Velocity-Verlet time-step.

- rp: Reaction path object
- timestep: obvs...
- mass: Fourier coefficient mass.

Definition at line 1797 of file rpath.f90.

20.9.2.23 getspringforces()

```
subroutine rpath::getspringforces (
            type(rxp) rp,
            real(8) kspring )
```

GetSpringForces.
Calculates the inter-bead spring potential and forces.

- rp: Reaction path object.
- kspring: Spring constant

Definition at line 1857 of file rpath.f90.

20.9.2.24 stripinactivefrompath()

```
subroutine rpath::stripinactivefrompath (
            type(rxp) rp,
            character*(*) stripfile,
            integer, dimension(3*namax) FixedDOF,
            integer, dimension(namax) FixedAtom,
            integer NDOFconstr,
            integer Natomconstr,
            logical UpdateCons )
```

StripInactiveFromPath.
Removes inactive molecules from the reaction path. Here, inactive means that the molecules are not involved in any bond changes.

- rp: Reaction path object.
- UpdateCons: flag determining if the FixedAtom and FixedDof should be updated to match the stripped CXS
- stripfile: The filename where the stripped path is output.

Definition at line 1901 of file rpath.f90.

20.9.2.25 errorr()

```
real(8) function rpath::errorr (
            real(8), dimension(3,na) one,
            real(8), dimension(3,na) two,
            integer na )
```

errorr calculates the absolute difference, elementwise between two matrices.

- one, two: the matrices
- na: ...

Definition at line 2102 of file rpath.f90.

Chapter 21

Data Type Documentation

21.1 chemstr::cxs Type Reference

Chemical structure object definition.

Public Attributes

- integer **na**
- real(8), dimension(:,:), allocatable r

  Coordinates in atomic units (bohr)
- real(8), dimension(:,:), allocatable p

  Momenta in atomic units (bohr / au)
- real(8), dimension(:,:), allocatable dvdr

  PES derivatives wrt coordinates in atomic units (Eh/bohr)
- real(8), dimension(:,:), allocatable force

  Forces wrt coordinates in atomic units (Eh/bohr)
- real(8), dimension(:,:), allocatable hessian

  Hessian matrix.
- character ∗2, dimension(:), allocatable atomlabel

  Atom labels.
- real(8), dimension(:), allocatable mass

  Atomic masses in au.
- integer ndofconstr

  Number of DOF constraints.
- integer natomconstr

  Number of atom constraints.
- logical, dimension(:), allocatable fixeddof

  Fixed DOF ids.
- logical, dimension(:), allocatable fixedatom

  Fixed atom ids.
- real(8) vcalc

  Calculate potential energy of structure.
- integer, dimension(:,:), allocatable graph

  Bonding graph for structure.
- real(8), dimension(:), allocatable sprintcoords

  SPRINT coordinates of the structure.
- integer, dimension(:,:), allocatable **molid**
- integer, dimension(:), allocatable namol

  Atom indices and numbers for each molecule in graph.

- integer nmol

  Number of molecules defined by structure.
- real(8) vcon

  Constraint potential energy.
- real(8) fitness

  Fitness for molecular optimization.
- real(8) fitness_scaled

  Scaled fitness for molecular optimization.
- character(len=10) method = ''

  abinitio method used to calculate properties
- integer nbonds

  Number of bonds.
- integer nangles

  Number of angles.
- integer ntors

  Number of torsions.
- real(8), dimension(nbondmax) bondl

  Calculated bond lengths.
- real(8), dimension(nanglemax) angle

  Calculated angles.
- real(8), dimension(ntorsmax) torsion

  Calculated torsion angles.
- integer, dimension(nbondmax, 2) bondid

  IDs of atoms in bonds.
- integer, dimension(nanglemax, 3) angleid

  IDs of atoms in angles.
- integer, dimension(ntorsmax, 4) torsid

  IDs of atoms in torsions.
- real(8), dimension(nbondmax, 2, 3) dbonddr

  Derivatives of bond-lengths wrt xyz of atoms.
- real(8), dimension(nanglemax, 3, 3) dangdr

  Derivatives of angles wrt xyz of atoms.
- real(8), dimension(ntorsmax, 4, 3) dtorsdr

  Derivatives of torsions wrt xyz of atoms.
- real(8), dimension(:), allocatable molen

  energy per molecule
- integer, dimension(:), allocatable molspin

  spin on each molecule
- integer, dimension(:), allocatable molcharge

  Total charge on each molecule.
- integer **itargetmol**

21.1.1 Detailed Description

Chemical structure object definition.
Definition at line 21 of file structure.f90.
The documentation for this type was generated from the following file:

- /Users/scott/code/cde/src/structure.f90

21.2 globaldata::fingerprint Type Reference

Type definition for the fingerprint of a reaction-path.

Public Attributes

- double precision, dimension(:,:,:), allocatable sprint

    sprint vector for beggining and end chemical structures (N elements)
- double precision, dimension(:,:,:), allocatable piv

    as above, but piv (N∗(N-1)/2 elements)
- double precision, dimension(:,:), allocatable menergy

    molecular energy per image, per active molecule i think
- double precision, dimension(:,:,:), allocatable mhash

    the molecular eigenvalues that characterises the graph
- integer graphmoveid

    graph ID that was used on chemical structure IC
- integer ic

    the start or end of the Path that was subject to the graph move
- integer, dimension(2) nactmol

    number of active molecules in the start and end images
- character(100) makebreaklab

    label showing which elements are breaking/forming (symmetric)
- character(250), dimension(:,:), allocatable smiles

    an array of nactmol Smile strings in a unique order
- character(100), dimension(:,:), allocatable mformula

    an array of nactmol molecular Formula strings in a unique order
- integer, dimension(:,:), allocatable actnamol

    number of atoms per active molecules
- double precision nbb

    (number of bonds formed+broken)/2
- integer id

    fingerprint ID
- type(fingerprint), pointer **next** => null()
- type(fingerprint), pointer **prev** => null()

21.2.1 Detailed Description

Type definition for the fingerprint of a reaction-path.
Definition at line 280 of file globaldata.f90.
The documentation for this type was generated from the following file:

- /Users/scott/code/cde/src/globaldata.f90

21.3 globaldata::graphpar Type Reference

Public Attributes

- integer, dimension(10) **react**
- integer, dimension(10) prod

    user supplied molecule ID from the Mol.dat file that will form the ends (products and reactants) of the reaction netowrk tree that they wish the code to calculate
- integer **nprod**
- integer nreact

    the number of molecules of products and reactants the user has supplied
- integer maxtreesolutions

    the number of tree solutions the user requested to generate...
- integer **nni**

- integer nri

  number of nodes and reactions the user want us to ignore during the graph generation algorithm...
- integer maxcomb

  the cutoff for the number of trees we keep from levels below the tree, to stop the combinatorial explosion
- integer maxb

  the maximum number of nodes-with-more-than-one-reaction we are allowed to visit when traversing the tree network
- integer mxevp

  the maximum number of trees per eigenvector cluster the user wants to print out
- integer treefav

  -1: the tree should be unfavourable; 1: tree should be favourable; 0: it can be either (speaking about energetics)
- integer maxisosteps

  the maximum number of ismerization reactions per isomer group that can be found in the reaction trees
- double precision rdiskstore

  if a particuar branch has more than maxcomb∗rdiskstore trees comming off it, those trees will be stored in disk, to avoid memory consumption
- double precision mxndexpo

  The order of magnitude of the maximum total number of nodes explored during the tree graph search prodedure.
- double precision absthresh0

  The initial sum of absolute energy of reactions threshold used to estimate the size of the network search... can be provided by the user if it has previously perfomed a calculation and already known a good number (printed in the logfile)
- integer, dimension(:), allocatable nignore

  an array of node ids to ignore when constructing the network
- integer, dimension(:), allocatable rignore

  an array of reaction ids to ignore when constructing the network
- logical enforcefullrx

  flag determining weather the user wants only trees that have all the reactants in the react array
- logical readnds

  reads the nds information from a restart binary file inputfilename_nds.bin if react, prod, nignore,rignore,Md are the same
- character(10) threshmeth

  weather to use absolute energies or lifetimes as the threshold of exploration of the network

21.3.1 Detailed Description

Definition at line 239 of file globaldata.f90.
The documentation for this type was generated from the following file:

- /Users/scott/code/cde/src/globaldata.f90

21.4 globaldata::molgm Type Reference

Public Attributes

- integer, dimension(:,:,:,:), allocatable **id**
- integer, dimension(:,:), allocatable **naid**
- integer, dimension(:,:,:), allocatable **aid**
- integer **nid**
- integer **na**
- integer, dimension(:,:,:), allocatable **ic**
- integer, dimension(:), allocatable **naic**
- integer, dimension(:,:), allocatable **aic**
- logical, dimension(:,:,:), allocatable **arm**

21.4.1 Detailed Description

Definition at line 266 of file globaldata.f90.
The documentation for this type was generated from the following file:

- /Users/scott/code/cde/src/globaldata.f90

21.5 globaldata::refresh_template Type Reference

Public Attributes

- integer desirednmoves

  Number of steps before reset to a template, 80% of the time !seb.
- integer ntemplate

  Number of defined templates seb].
- double precision desirednmoveprob

  the probability that we reach the desirednmoves away from templates
- double precision, dimension(:), allocatable templateprob

  the probability that
- character(len=50), dimension(:), allocatable gdscxstemplate

  the filename of that template

21.5.1 Detailed Description

Definition at line 303 of file globaldata.f90.
The documentation for this type was generated from the following file:

- /Users/scott/code/cde/src/globaldata.f90

21.6 rpath::rxp Type Reference

Type definition for the reaction-path object.

Public Attributes

- integer nimage

  Number of images.
- integer na

  Number of atoms per image.
- type(cxs), dimension(:), allocatable cx

  Allocated chemical structure objects in the path.
- real(8), dimension(:,:,:), allocatable coeff

  Fourier coefficients of path (3,na,nb)
- real(8), dimension(:,:,:), allocatable pcoeff

  Momentum of Fourier coefficients of path (3,na,nb)
- real(8), dimension(:,:,:), allocatable dcoeff

  Derivatives of Fourier coefficients of path (3,na,nb)
- real(8), dimension(:), allocatable ks

  intra bead force constants
- real(8) vspring

  Spring potential term.

21.6.1 Detailed Description

Type definition for the reaction-path object.
Definition at line 23 of file rpath.f90.
The documentation for this type was generated from the following file:

- /Users/scott/code/cde/src/rpath.f90

21.7 globaldata::vtbashop Type Reference

Public Attributes

- logical **mcaccepted** = .false.
- logical **lbasinhopping** = .false.
- double precision **t** = 298.0d0

21.7.1 Detailed Description

Definition at line 311 of file globaldata.f90.
The documentation for this type was generated from the following file:

- /Users/scott/code/cde/src/globaldata.f90