ACPYPE

Release 2022.8.25

Alan Silva

Nov 10, 2022

CONTENTS

ACPYPE - AnteChamber PYthon Parser interfacE

A tool based in **Python** to use **Antechamber** to generate topologies for chemical compounds and to interface with others python applications like CCPN and ARIA. Topologies files to be generated so far: CNS/XPLOR, GROMACS, CHARMM and AMBER.

acpype	The Package

CHAPTER

ONE

ACPYPE

The Package

Requirements:

- `Python 3.6` or higher
- `Antechamber` (from `AmberTools` preferably)
- `OpenBabel` (optional, but strongly recommended)

This code is released under **GNU General Public License V3**.

<<< NO WARRANTY AT ALL!!! >>>

It was inspired by:

- `amb2gmx.pl` (Eric Sorin, David Mobley and John Chodera) and depends on Antechamber and Openbabel
- YASARA Autosmiles (Elmar Krieger)
- `topolbuild` (Bruce Ray)
- `xplo2d` (G.J. Kleywegt)

For Non-uniform 1-4 scale factor conversion (e.g. if using `GLYCAM06`), please cite:

> BERNARDI, A., FALLER, R., REITH, D., and KIRSCHNER, K. N. ACPYPE update for nonuniform 1-4 scale factors: Conversion of the GLYCAM06 force field from AMBER to GROMACS. SoftwareX 10 (2019), 100241. doi: 10.1016/j.softx.2019.100241

For Antechamber, please cite:

1. WANG, J., WANG, W., KOLLMAN, P. A., and CASE, D. A. Automatic atom type and bond type perception in molecular mechanical calculations. Journal of Molecular Graphics and Modelling 25, 2 (2006), 247-260. doi: 10.1016/j.jmgm.2005.12.005
2. WANG, J., WOLF, R. M., CALDWELL, J. W., KOLLMAN, P. A., and CASE, D. A. Development and testing of a General Amber Force Field. Journal of Computational Chemistry 25, 9 (2004), 1157-1174. doi: 10.1002/jcc.20035

If you use this code, I am glad if you cite:

> SOUSA DA SILVA, A. W. & VRANKEN, W. F. ACPYPE - AnteChamber PYthon Parser interfacE. BMC Research Notes 5 (2012), 367 doi: 10.1186/1756-0500-5-367

and (optionally)

> BATISTA, P. R.; WILTER, A.; DURHAM, E. H. A. B. & PASCUTTI, P. G. Molecular Dynamics Simulations Applied to the Study of Subtypes of HIV-1 Protease. Cell Biochemistry and Biophysics 44 (2006), 395-404. doi: 10.1385/CBB:44:3:395

Alan Silva, D.Sc. <alanwilter _at_ gmail _dot_ com>

Modules

acpype.acs_api	
acpype.cli	
acpype.logger	
acpype.mol	Constructors to define and store the system's topology
acpype.params	
acpype.parser_args	
acpype.topol	
acpype.utils	

1.1 acpype.acs_api

Functions

acpype_api(inputFile[, chargeType, ...])	
clearFileInMemory()	
readFiles(basename, chargeType, atomType)	

1.1.1 acpype.acs_api.acpype_api

acpype.acs_api.**acpype_api**(*inputFile*, *chargeType='bcc'*, *chargeVal=None*, *multiplicity='1'*, *atomType='gaff2'*, *force=False*, *basename=None*, *debug=False*, *outTopol='all'*, *engine='tleap'*, *allhdg=False*, *timeTol=10800*, *qprog='sqm'*, *ekFlag=None*, *verbose=True*, *gmx4=False*, *merge=False*, *direct=False*, *is_sorted=False*, *chiral=False*, *is_smiles=False*)

1.1.2 acpype.acs_api.clearFileInMemory

acpype.acs_api.**clearFileInMemory**()

1.1.3 acpype.acs_api.readFiles

acpype.acs_api.**readFiles**(*basename*, *chargeType*, *atomType*)

1.2 acpype.cli

Functions

init_main([binaries, argv])	Orchestrate the command line usage for ACPYPE with its all input arguments.

1.2.1 acpype.cli.init_main

acpype.cli.**init_main**(*binaries: Dict[str, str] = {'ac_bin': 'antechamber', 'obabel_bin': 'obabel'}*, *argv: Optional[List[str]] = None*)

Orchestrate the command line usage for ACPYPE with its all input arguments.

Parameters

- **binaries** (*Dict[str, str], optional*) – Mostly used for debug and testing. Defaults to acpype.params.binaries.
- **argv** (*Optional[List[str]], optional*) – Mostly used for debug and testing. Defaults to None.

Returns
0 or 19 (failed)

Return type
SystemExit(status)

1.3 acpype.logger

Functions

copy_log(molecule)	
set_logging_conf([level])	

1.3.1 acpype.logger.copy_log

acpype.logger.**copy_log**(*molecule*)

1.3.2 acpype.logger.set_logging_conf

acpype.logger.**set_logging_conf**(*level=20*)

Classes

LogFormatter()	Define log formatter

1.3.3 acpype.logger.LogFormatter

class acpype.logger.**LogFormatter**

Bases: Formatter

Define log formatter

__init__()

Initialize the formatter with specified format strings.

Initialize the formatter either with the specified format string, or a default as described above. Allow for specialized date formatting with the optional datefmt argument. If datefmt is omitted, you get an ISO8601-like (or RFC 3339-like) format.

Use a style parameter of '%', '{' or '$' to specify that you want to use one of %-formatting, `str.format()` ({}) formatting or `string.Template` formatting in your format string.

Changed in version 3.2: Added the `style` parameter.

Methods

__init__()	Initialize the formatter with specified format strings.
converter	localtime([seconds]) -> (tm_year,tm_mon,tm_mday,tm_hour,tm_min,
format(record)	Format the specified record as text.
formatException(ei)	Format and return the specified exception information as a string.
formatMessage(record)	
formatStack(stack_info)	This method is provided as an extension point for specialized formatting of stack information.
formatTime(record[, datefmt])	Return the creation time of the specified LogRecord as formatted text.
usesTime()	Check if the format uses the creation time of the record.

Attributes

dbg_fmt
default_msec_format
default_time_format
err_fmt
info_fmt
warn_fmt

converter()

localtime([seconds]) -> (tm_year,tm_mon,tm_mday,tm_hour,tm_min,
tm_sec,tm_wday,tm_yday,tm_isdst)

Convert seconds since the Epoch to a time tuple expressing local time. When 'seconds' is not passed in, convert the current time instead.

format(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

formatException(*ei*)

Format and return the specified exception information as a string.

This default implementation just uses traceback.print_exception()

formatStack(*stack_info*)

This method is provided as an extension point for specialized formatting of stack information.

The input data is a string as returned from a call to `traceback.print_stack()`, but with the last trailing newline removed.

The base implementation just returns the value passed in.

formatTime(*record*, *datefmt=None*)

Return the creation time of the specified LogRecord as formatted text.

This method should be called from format() by a formatter which wants to make use of a formatted time. This method can be overridden in formatters to provide for any specific requirement, but the basic behaviour is as follows: if datefmt (a string) is specified, it is used with time.strftime() to format the creation time of the record. Otherwise, an ISO8601-like (or RFC 3339-like) format is used. The resulting string is returned. This function uses a user-configurable function to convert the creation time to a tuple. By default, time.localtime() is used; to change this for a particular formatter instance, set the 'converter' attribute to a function with the same signature as time.localtime() or time.gmtime(). To change it for all formatters, for example if you want all logging times to be shown in GMT, set the 'converter' attribute in the Formatter class.

usesTime()

Check if the format uses the creation time of the record.

1.4 acpype.mol

Constructors to define and store the system's topology

It will create instances for Atoms, AtomTypes, Bonds, Angles and Dihedrals where the topology (the relationships between atoms) is defined and paramenters are stored.

Example

```
>>> atom = acpype.mol.Atom(...) # to be improved
```

acpype.mol.**Atom**

define Atom

acpype.mol.**AtomType**

define AtomType

Classes

Angle(atoms, kTheta, thetaEq)	attributes: 3 Atoms, spring constant (kcal/mol/rad^2), angle eq.
Atom(atomName, atomType, id_, resid, mass, ...)	Atom Object Definition
AtomType(atomTypeName, mass, ACOEF, BCOEF)	AtomType per atom in gaff or amber.
Bond(atoms, kBond, rEq)	attributes: pair of Atoms, spring constant (kcal/mol), dist.
Dihedral(atoms, kPhi, period, phase)	attributes: 4 Atoms, spring constant (kcal/mol), periodicity, phase (rad)

1.4.1 acpype.mol.Angle

class acpype.mol.**Angle**(*atoms*, *kTheta*, *thetaEq*)

Bases: object

attributes: 3 Atoms, spring constant (kcal/mol/rad^2), angle eq. (rad)

__init__(*atoms*, *kTheta*, *thetaEq*)

Methods

`__init__`(atoms, kTheta, thetaEq)	

1.4.2 acpype.mol.Atom

class acpype.mol.**Atom**(*atomName: str*, *atomType:* AtomType, *id_: int*, *resid: int*, *mass: float*, *charge: float*, *coord: List[float]*)

Bases: `object`

Atom Object Definition

Charges in *prmtop* file are divided by `18.2223` to be converted in units of the electron charge.

To convert `ACOEF` and `BCOEF` to `r0` (Å) and `epsilon` (: kcal/mol), as seen in `gaff.dat` for example, for a same atom type (`i = j`):

$$r_0 = 1/2 * (2 * A_{coef}/B_{coef})^{1/6}$$
$$\epsilon = 1/(4 * A_{coef}) * B_{coef}^2$$

To convert `r0` and `epsilon` to `ACOEF` and `BCOEF`:

$$A_{coef} = \sqrt{\epsilon_i * \epsilon_j} * (r_{0i} + r_{0j})^{12}$$
$$B_{coef} = 2 * \sqrt{\epsilon_i * \epsilon_j} * (r_{0i} + r_{0j})^6$$
$$= 2 * A_{coef}/(r_{0i} + r_{0j})^6$$

where index `i` and `j` for atom types. Coordinates are given in Å and masses in Atomic Mass Unit.

Returns
atom object

Return type
acpype.mol.Atom

__init__(*atomName: str*, *atomType:* AtomType, *id_: int*, *resid: int*, *mass: float*, *charge: float*, *coord: List[float]*)

Parameters

- **atomName** (*str*) – atom name
- **atomType** (AtomType) – atomType object
- **id** (*int*) – atom number index
- **resid** (*int*) – residues number index
- **mass** (*float*) – atom mass
- **charge** (*float*) – atom charge
- **coord** (*List[float]*) – atom (x,y,z) coordinates

Methods

__init__(atomName, atomType, id_, resid, ...)

param atomName
atom name

1.4.3 acpype.mol.AtomType

class acpype.mol.**AtomType**(*atomTypeName*, *mass*, *ACOEF*, *BCOEF*)

Bases: object

AtomType per atom in gaff or amber.

__init__(*atomTypeName*, *mass*, *ACOEF*, *BCOEF*)

Methods

__init__(atomTypeName, mass, ACOEF, BCOEF)

1.4.4 acpype.mol.Bond

class acpype.mol.**Bond**(*atoms*, *kBond*, *rEq*)

Bases: object

attributes: pair of Atoms, spring constant (kcal/mol), dist. eq. (Ang)

__init__(*atoms*, *kBond*, *rEq*)

Methods

__init__(atoms, kBond, rEq)

1.4.5 acpype.mol.Dihedral

class acpype.mol.**Dihedral**(*atoms*, *kPhi*, *period*, *phase*)

Bases: object

attributes: 4 Atoms, spring constant (kcal/mol), periodicity, phase (rad)

__init__(*atoms*, *kPhi*, *period*, *phase*)

Methods

__init__(atoms, kPhi, period, phase)	

1.5 acpype.params

1.6 acpype.parser_args

Functions

get_option_parser()	

1.6.1 acpype.parser_args.get_option_parser

acpype.parser_args.**get_option_parser**()

1.7 acpype.topol

Classes

ACTopol(inputFile[, binaries, chargeType, ...])	Class to build the AC topologies (Antechamber AmberTools).
AbstractTopol()	Abstract super class to build topologies
MolTopol([acTopolObj, acFileXyz, acFileTop, ...])	Class to write topologies and parameters files for several applications.
Topology_14()	Amber topology abstraction for non-uniform 1-4 scale factors.

1.7.1 acpype.topol.ACTopol

class acpype.topol.**ACTopol**(*inputFile*, *binaries={'ac_bin': 'antechamber', 'obabel_bin': 'obabel'}*, *chargeType='bcc'*, *chargeVal=None*, *multiplicity='1'*, *atomType='gaff2'*, *force=False*, *basename=None*, *debug=False*, *outTopol='all'*, *allhdg=False*, *timeTol=10800*, *qprog='sqm'*, *ekFlag=None*, *verbose=True*, *gmx4=False*, *merge=False*, *direct=False*, *is_sorted=False*, *chiral=False*, *amb2gmx=False*, *level=20*)

Bases: *AbstractTopol*

Class to build the AC topologies (Antechamber AmberTools).

__init__(*inputFile*, *binaries={'ac_bin': 'antechamber', 'obabel_bin': 'obabel'}*, *chargeType='bcc'*, *chargeVal=None*, *multiplicity='1'*, *atomType='gaff2'*, *force=False*, *basename=None*, *debug=False*, *outTopol='all'*, *allhdg=False*, *timeTol=10800*, *qprog='sqm'*, *ekFlag=None*, *verbose=True*, *gmx4=False*, *merge=False*, *direct=False*, *is_sorted=False*, *chiral=False*, *amb2gmx=False*, *level=20*)

Methods

__init__(inputFile[, binaries, chargeType, ...])	
balanceCharges(chargeList[, FirstNonSoluteId])	Spread charges fractions among atoms to balance system's total charge.
checkFrcmod()	Check FRCMOD file.
checkLeapLog(log)	Check Leap log.
checkSmiles()	Check if input arg is a SMILES string.
checkXyzAndTopFiles()	Check XYZ and TOP files.
convertPdbToMol2()	Convert PDB to MOL2 by using obabel.
convertSmilesToMol2()	Convert Smiles to MOL2 by using obabel.
createACTopol()	If successful, Amber Top and Xyz files will be generated.
createMolTopol()	Create MolTopol obj.
delOutputFiles()	Delete temporary output files.
execAntechamber([chargeType, atomType])	To call Antechamber and execute it.
execObabel()	Execute obabel.
execParmchk()	Execute parmchk.
execTleap()	Execute tleap.
getABCOEFs()	Get non-bonded coefficients.
getAngles()	Get Angles.
getAtoms()	Set a list with all atoms objects built from dat in acFileTop.
getBonds()	Get Bonds.
getChirals()	Get chiral atoms (for CNS only!).
getCoords()	For a given acFileXyz file, return a list of coords as.
getDihedrals()	Get dihedrals (proper and imp), condensed list of prop dih and atomPairs.
getFlagData(flag)	For a given acFileTop flag, return a list of the data related.
getResidueLabel()	Get a 3 capital letters code from acFileTop.
guessCharge()	Guess the charge of a system based on antechamber.
locateDat(aFile)	Locate a file pertinent to $AMBERHOME/dat/leap/parm/.
makeDir()	Make Dir.
pickleSave()	Create portable serialized representations of System's Python objects.
printDebug([text])	Debug log level.
printDebugQuoted([text])	Print quoted messages.
printError([text])	Error log level.
printErrorQuoted([text])	Print quoted messages.
printMess([text])	Info log level.
printWarn([text])	Warn log level.

continues on next page

Table 1 – continued from previous page

readMol2TotalCharge(mol2File)	Reads the charges in given mol2 file and returns the total.
search([name, alist])	Returns a list with all atomName matching 'name' or just the first case.
setAtomType4Gromacs()	Set atom types for Gromacs.
setProperDihedralsCoef()	Create proper dihedrals list with Ryckaert-Bellemans coefficients.
setResNameCheckCoords()	Set a 3 letter residue name and check coords for issues like duplication, atoms too close or too sparse.
signal_handler(_signum, _frame)	Signal handler.
sortAtomsForGromacs()	Re-sort atoms for gromacs, which expects all hydrogens to immediately follow the heavy atom they are bonded to and belong to the same charge group.
writeCharmmTopolFiles()	Write CHARMM topology files.
writeCnsTopolFiles()	Write CNS topology files.
writeGroFile()	Write GRO files.
writeGromacsTop()	Write GMX topology file.
writeGromacsTopolFiles()	Write GMX topology Files.
writeMdpFiles()	Write MDP for test with GROMACS.
writePdb(afile)	Write a new PDB file with the atom names defined by Antechamber.
writePosreFile([fc])	Write file with positional restraints for heavy atoms.

balanceCharges(*chargeList*, *FirstNonSoluteId=None*)

Spread charges fractions among atoms to balance system's total charge.

Note that python is very annoying about floating points. Even after balance, there will always be some residue of order e^{-12} to e^{-16}, which is believed to vanished once one writes a topology file, say, for CNS or GMX, where floats are represented with 4 or 5 maximum decimals.

checkFrcmod()

Check FRCMOD file.

checkLeapLog(*log*)

Check Leap log.

checkSmiles()

Check if input arg is a SMILES string.

Returns
True/False

Return type
bool

checkXyzAndTopFiles()

Check XYZ and TOP files.

convertPdbToMol2()

Convert PDB to MOL2 by using obabel.

convertSmilesToMol2()

Convert Smiles to MOL2 by using obabel.

createACTopol()

If successful, Amber Top and Xyz files will be generated.

createMolTopol()

Create MolTopol obj.

delOutputFiles()

Delete temporary output files.

execAntechamber(*chargeType=None*, *atomType=None*) → bool

To call Antechamber and execute it.

Parameters

- **chargeType** (*[str], optional*) – bcc, gas or user. Defaults to None/bcc.
- **atomType** (*[str], optional*) – gaff, amber, gaff2, amber2. Defaults to None/gaff2.

Returns

True if failed.

Return type

bool

```
Welcome to antechamber 21.0: molecular input file processor.

Usage: antechamber \
        -i     input file name
        -fi    input file format
        -o     output file name
        -fo    output file format
        -c     charge method
        -cf    charge file name
        -nc    net molecular charge (int)
        -a     additional file name
        -fa    additional file format
        -ao    additional file operation
                crd    : only read in coordinate
                crg    : only read in charge
                radius: only read in radius
                name   : only read in atom name
                type   : only read in atom type
                bond   : only read in bond type
        -m     multiplicity (2S+1), default is 1
        -rn    residue name, overrides input file, default is MOL
        -rf    residue topology file name in prep input file,
                default is molecule.res
        -ch    check file name for gaussian, default is 'molecule'
        -ek    mopac or sqm keyword, inside quotes; overwrites previous ones
        -gk    gaussian job keyword, inside quotes, is ignored when both -gopt␣
↪and -gsp are used
        -gopt  gaussian job keyword for optimization, inside quotes
        -gsp   gaussian job keyword for single point calculation, inside quotes
        -gm    gaussian memory keyword, inside quotes, such as "%mem=1000MB"
        -gn    gaussian number of processors keyword, inside quotes, such as "
↪%nproc=8"
```

(continues on next page)

(continued from previous page)

```
       -gdsk  gaussian maximum disk usage keyword, inside quotes, such as "
↪%maxdisk=50GB"
       -gv    add keyword to generate gesp file (for Gaussian 09 only)
                1    : yes
                0    : no, the default
       -ge    gaussian esp file generated by iop(6/50=1), default is g09.gesp
       -tor   torsional angle list, inside a pair of quotes, such as "1-2-3-
↪4:0,5-6-7-8"
                ':1' or ':0' indicates the torsional angle is frozen or not
       -df    am1-bcc precharge flag, 2 - use sqm(default); 0 - use mopac
       -at    atom type
                gaff : the default
                gaff2: for gaff2 (beta-version)
                amber: for PARM94/99/99SB
                bcc  : bcc
                sybyl: sybyl
       -du    fix duplicate atom names: yes(y)[default] or no(n)
       -bk    component/block Id, for ccif
       -an    adjust atom names: yes(y) or no(n)
                the default is 'y' for 'mol2' and 'ac' and 'n' for the other␣
↪formats
       -j     atom type and bond type prediction index, default is 4
                0    : no assignment
                1    : atom type
                2    : full  bond types
                3    : part  bond types
                4    : atom and full bond type
                5    : atom and part bond type
       -s     status information: 0(brief), 1(default) or 2(verbose)
       -eq    equalizing atomic charge, default is 1 for '-c resp' and '-c bcc
↪' and 0
                for the other charge methods
                0    : no use
                1    : by atomic paths
                2    : by atomic paths and structural information, i.e. E/Z␣
↪configurations
       -pf    remove intermediate files: yes(y) or no(n)[default]
       -pl    maximum path length to determin equivalence of atomic charges␣
↪for resp and bcc,
                the smaller the value, the faster the algorithm, default is -1␣
↪(use full length),
                set this parameter to 10 to 30 if your molecule is big (# atoms␣
↪>= 100)
       -seq   atomic sequence order changable: yes(y)[default] or no(n)
       -dr    acdoctor mode: yes(y)[default] or no(n)
       -i -o -fi and -fo must appear; others are optional
       Use 'antechamber -L' to list the supported file formats and charge␣
↪methods

                    List of the File Formats

       file format type  abbre. index | file format type abbre. index
```

(continues on next page)

(continued from previous page)

```
-------------------------------------------------------------------
Antechamber         ac       1  | Sybyl Mol2          mol2     2
PDB                 pdb      3  | Modified PDB        mpdb     4
AMBER PREP (int)    prepi    5  | AMBER PREP (car)    prepc    6
Gaussian Z-Matrix   gzmat    7  | Gaussian Cartesian  gcrt     8
Mopac Internal      mopint   9  | Mopac Cartesian     mopcrt  10
Gaussian Output     gout    11  | Mopac Output        mopout  12
Alchemy             alc     13  | CSD                 csd     14
MDL                 mdl     15  | Hyper               hin     16
AMBER Restart       rst     17  | Jaguar Cartesian    jcrt    18
Jaguar Z-Matrix     jzmat   19  | Jaguar Output       jout    20
Divcon Input        divcrt  21  | Divcon Output       divout  22
SQM Input           sqmcrt  23  | SQM Output          sqmout  24
Charmm              charmm  25  | Gaussian ESP        gesp    26
Component cif       ccif    27  | GAMESS dat          gamess  28
Orca input          orcinp  29  | Orca output         orcout  30
-------------------------------------------------------------------
AMBER restart file can only be read in as additional file.

            List of the Charge Methods

charge method     abbre. index | charge method    abbre. index
-------------------------------------------------------------------
RESP                resp     1  |  AM1-BCC           bcc      2
CM1                 cm1      3  |  CM2               cm2      4
ESP (Kollman)       esp      5  |  Mulliken          mul      6
Gasteiger           gas      7  |  Read in charge    rc       8
Write out charge    wc       9  |  Delete Charge     dc      10
-------------------------------------------------------------------
```

execObabel()

Execute obabel.

execParmchk()

Execute parmchk.

execTleap()

Execute tleap.

getABCOEFs()

Get non-bonded coefficients.

getAngles()

Get Angles.

getAtoms()

Set a list with all atoms objects built from dat in acFileTop.

Set also atomType system is gaff or amber, list of atomTypes, resid and system's total charge.

getBonds()

Get Bonds.

getChirals()

Get chiral atoms (for CNS only!).

Plus its 4 neighbours and improper dihedral angles to store non-planar improper dihedrals.

getCoords()

For a given acFileXyz file, return a list of coords as:

```
[[x1,y1,z1],[x2,y2,z2], etc.]
```

getDihedrals()

Get dihedrals (proper and imp), condensed list of prop dih and atomPairs.

getFlagData(*flag*)

For a given acFileTop flag, return a list of the data related.

getResidueLabel()

Get a 3 capital letters code from acFileTop.

Returns a list.

guessCharge()

Guess the charge of a system based on antechamber.

Returns None in case of error.

locateDat(*aFile*)

Locate a file pertinent to $AMBERHOME/dat/leap/parm/.

makeDir()

Make Dir.

pickleSave()

Create portable serialized representations of System's Python objects.

Example

to restore:

```
from acpype import *
# import cPickle as pickle
import pickle
mol = pickle.load(open('DDD.pkl','rb'))
```

printDebug(*text=''*)

Debug log level.

printDebugQuoted(*text=''*)

Print quoted messages.

printError(*text=''*)

Error log level.

printErrorQuoted(*text=''*)

Print quoted messages.

printMess(*text=''*)

Info log level.

printWarn(*text=''*)

Warn log level.

readMol2TotalCharge(*mol2File*)

Reads the charges in given mol2 file and returns the total.

search(*name=None*, *alist=False*)

Returns a list with all atomName matching 'name' or just the first case.

setAtomType4Gromacs()

Set atom types for Gromacs.

Atom types names in Gromacs TOP file are not case sensitive; this routine will append a '_' to lower case atom type.

Example

```
>>> CA and ca -> CA and ca_
```

setProperDihedralsCoef()

Create proper dihedrals list with Ryckaert-Bellemans coefficients.

It takes self.condensedProperDihedrals and returns self.properDihedralsCoefRB, a reduced list of quartet atoms + RB. Coefficients ready for GMX (multiplied by 4.184):

```
self.properDihedralsCoefRB = [ [atom1,..., atom4], C[0:5] ]
```

For proper dihedrals: a quartet of atoms may appear with more than one set of parameters and to convert to GMX they are treated as RBs.

The resulting coefficients calculated here may look slighted different from the ones calculated by amb2gmx.pl because python is taken full float number from prmtop and not rounded numbers from rd-parm.out as amb2gmx.pl does.

setResNameCheckCoords()

Set a 3 letter residue name and check coords for issues like duplication, atoms too close or too sparse.

signal_handler(*_signum*, *_frame*)

Signal handler.

sortAtomsForGromacs()

Re-sort atoms for gromacs, which expects all hydrogens to immediately follow the heavy atom they are bonded to and belong to the same charge group.

Currently, atom mass < 1.2 is taken to denote a proton. This behaviour may be changed by modifying the 'is_hydrogen' function within.

JDC 2011-02-03

writeCharmmTopolFiles()

Write CHARMM topology files.

writeCnsTopolFiles()

Write CNS topology files.

writeGroFile()

Write GRO files.

writeGromacsTop()

Write GMX topology file.

writeGromacsTopolFiles()

Write GMX topology Files.

```
# from ~/Programmes/amber10/dat/leap/parm/gaff.dat
#atom type        atomic mass        atomic polarizability        comments
ca                12.01                 0.360                    Sp2 C in pure␣
↪aromatic systems
ha                1.008                 0.135                    H bonded to␣
↪aromatic carbon

#bonded atoms        harmonic force kcal/mol/A^2       eq. dist. Ang.  comments
ca-ha                  344.3*                           1.087**         SOURCE3␣
↪ 1496    0.0024    0.0045
* for gmx: 344.3 * 4.184 * 100 * 2 = 288110 kJ/mol/nm^2 (why factor 2?)
** convert Ang to nm ( div by 10) for gmx: 1.087 A = 0.1087 nm
# CA HA      1    0.10800   307105.6 ; ged from 340. bsd on C6H6 nmodes; PHE,
↪TRP,TYR (from ffamber99bon.itp)
# CA-HA  367.0    1.080       changed from 340. bsd on C6H6 nmodes; PHE,TRP,TYR␣
↪(from parm99.dat)

# angle        HF kcal/mol/rad^2    eq angle degrees     comments
ca-ca-ha        48.5*             120.01                SOURCE3 2980   0.1509  ␣
↪0.2511
* to convert to gmx: 48.5 * 4.184 * 2 = 405.848 kJ/mol/rad^2 (why factor 2?)
# CA  CA  HA           1   120.000    418.400 ; new99 (from ffamber99bon.itp)
# CA-CA-HA    50.0      120.00 (from parm99.dat)

# dihedral    idivf        barrier hight/2 kcal/mol  phase degrees      ␣
↪periodicity     comments
X -ca-ca-X    4           14.500*                    180.000             2.000 ␣
↪         intrpol.bsd.on C6H6
*convert 2 gmx: 14.5/4 * 4.184 * 2 (?) (yes in amb2gmx, not in topolbuild, why?
↪) = 30.334 or 15.167 kJ/mol
# X -CA-CA-X    4   14.50        180.0     2.         intrpol.bsd.on C6H6 (from␣
↪parm99.dat)
# X CA CA  X    3   30.334       0.000   -30.33400     0.000     0.000     0.
↪000   ; intrpol.bsd.on C6H6
;propers treated as RBs in GMX to use combine multiple AMBER torsions per␣
↪quartet (from ffamber99bon.itp)

# impr. dihedral        barrier hight/2      phase degrees       periodicity   ␣
↪ comments
X -X -ca-ha             1.1*                  180.                      2.     ␣
↪ bsd.on C6H6 nmodes
* to convert to gmx: 1.1 * 4.184 = 4.6024 kJ/mol/rad^2
# X -X -CA-HA         1.1          180.          2.           bsd.on C6H6␣
↪nmodes (from parm99.dat)
# X   X   CA  HA       1      180.00     4.60240     2      ; bsd.on C6H6 nmodes
;impropers treated as propers in GROMACS to use correct AMBER analytical␣
↪function (from ffamber99bon.itp)
```

(continues on next page)

(continued from previous page)

```
# 6-12 parms     sigma = 2 * r * 2^(-1/6)    epsilon
# atomtype        radius Ang.                    pot. well depth kcal/mol
↪comments
ha                  1.4590*                     0.0150**
↪  Spellmeyer
ca                  1.9080                     0.0860
↪OPLS
* to convert to gmx:
    sigma = 1.4590 * 2^(-1/6) * 2 = 2 * 1.29982 Ang. = 2 * 0.129982 nm  = 1.
↪4590 * 2^(5/6)/10 =  0.259964 nm
** to convert to gmx: 0.0150 * 4.184 = 0.06276 kJ/mol
# amber99_3    CA     0.0000  0.0000  A   3.39967e-01  3.59824e-01 (from
↪ffamber99nb.itp)
# amber99_22   HA     0.0000  0.0000  A   2.59964e-01  6.27600e-02 (from
↪ffamber99nb.itp)
# C*          1.9080  0.0860             Spellmeyer
# HA          1.4590  0.0150             Spellmeyer (from parm99.dat)
# to convert r and epsilon to ACOEF and BCOEF
# ACOEF = sqrt(e1*e2) * (r1 + r2)^12 ; BCOEF = 2 * sqrt(e1*e2) * (r1 + r2)^6 =
↪2 * ACOEF/(r1+r2)^6
# to convert ACOEF and BCOEF to r and epsilon
# r = 0.5 * (2*ACOEF/BCOEF)^(1/6); ep = BCOEF^2/(4*ACOEF)
# to convert ACOEF and BCOEF to sigma and epsilon (GMX)
# sigma = (ACOEF/BCOEF)^(1/6) * 0.1 ; epsilon = 4.184 * BCOEF^2/(4*ACOEF)
#   ca    ca       819971.66        531.10
#   ca    ha        76245.15        104.66
#   ha    ha         5716.30         18.52
```

For proper dihedrals: a quartet of atoms may appear with more than one set of parameters and to convert to GMX they are treated as RBs; use the algorithm:

```
for(my $j=$i;$j<=$lines;$j++){
    my $period = $pn{$j};
    if($pk{$j}>0) {
    $V[$period] = 2*$pk{$j}*$cal;
    }
    # assign V values to C values as predefined #
    if($period==1){
    $C[0]+=0.5*$V[$period];
    if($phase{$j}==0){
        $C[1]-=0.5*$V[$period];
    }else{
        $C[1]+=0.5*$V[$period];
    }
    }elsif($period==2){
    if(($phase{$j}==180)||($phase{$j}==3.14)){
        $C[0]+=$V[$period];
        $C[2]-=$V[$period];
    }else{
        $C[2]+=$V[$period];
    }
```

(continues on next page)

(continued from previous page)

```
    }elsif($period==3){
    $C[0]+=0.5*$V[$period];
    if($phase{$j}==0){
        $C[1]+=1.5*$V[$period];
        $C[3]-=2*$V[$period];
    }else{
        $C[1]-=1.5*$V[$period];
        $C[3]+=2*$V[$period];
    }
    }elsif($period==4){
    if(($phase{$j}==180)||($phase{$j}==3.14)){
        $C[2]+=4*$V[$period];
        $C[4]-=4*$V[$period];
    }else{
        $C[0]+=$V[$period];
        $C[2]-=4*$V[$period];
        $C[4]+=4*$V[$period];
    }
    }
}
```

writeMdpFiles()

Write MDP for test with GROMACS.

writePdb(*afile*)

Write a new PDB file with the atom names defined by Antechamber.

The format generated here use is slightly different from:

> old: http://www.wwpdb.org/documentation/file-format-content/format23/sect9.html latest: http://www.wwpdb.org/documentation/file-format-content/format33/sect9.html

respected to atom name. Using GAFF2 atom types:

```
CU/Cu Copper, CL/cl Chlorine, BR/br Bromine
```

Parameters

afile (*[str]*) – file path name

writePosreFile(*fc=1000*)

Write file with positional restraints for heavy atoms.

http://www.mdtutorials.com/gmx/complex/06_equil.html

1.7.2 acpype.topol.AbstractTopol

class acpype.topol.**AbstractTopol**

Bases: ABC

Abstract super class to build topologies

abstract __init__()

Methods

__init__()	
balanceCharges(chargeList[, FirstNonSoluteId])	Spread charges fractions among atoms to balance system's total charge.
checkFrcmod()	Check FRCMOD file.
checkLeapLog(log)	Check Leap log.
checkSmiles()	Check if input arg is a SMILES string.
checkXyzAndTopFiles()	Check XYZ and TOP files.
convertPdbToMol2()	Convert PDB to MOL2 by using obabel.
convertSmilesToMol2()	Convert Smiles to MOL2 by using obabel.
createACTopol()	If successful, Amber Top and Xyz files will be generated.
createMolTopol()	Create MolTopol obj.
delOutputFiles()	Delete temporary output files.
execAntechamber([chargeType, atomType])	To call Antechamber and execute it.
execObabel()	Execute obabel.
execParmchk()	Execute parmchk.
execTleap()	Execute tleap.
getABCOEFs()	Get non-bonded coefficients.
getAngles()	Get Angles.
getAtoms()	Set a list with all atoms objects built from dat in acFileTop.
getBonds()	Get Bonds.
getChirals()	Get chiral atoms (for CNS only!).
getCoords()	For a given acFileXyz file, return a list of coords as.
getDihedrals()	Get dihedrals (proper and imp), condensed list of prop dih and atomPairs.
getFlagData(flag)	For a given acFileTop flag, return a list of the data related.
getResidueLabel()	Get a 3 capital letters code from acFileTop.
guessCharge()	Guess the charge of a system based on antechamber.
locateDat(aFile)	Locate a file pertinent to $AMBERHOME/dat/leap/parm/.
makeDir()	Make Dir.
pickleSave()	Create portable serialized representations of System's Python objects.
printDebug([text])	Debug log level.
printDebugQuoted([text])	Print quoted messages.
printError([text])	Error log level.
printErrorQuoted([text])	Print quoted messages.

continues on next page

Table 2 – continued from previous page

printMess([text])	Info log level.
printWarn([text])	Warn log level.
readMol2TotalCharge(mol2File)	Reads the charges in given mol2 file and returns the total.
search([name, alist])	Returns a list with all atomName matching 'name' or just the first case.
setAtomType4Gromacs()	Set atom types for Gromacs.
setProperDihedralsCoef()	Create proper dihedrals list with Ryckaert-Bellemans coefficients.
setResNameCheckCoords()	Set a 3 letter residue name and check coords for issues like duplication, atoms too close or too sparse.
signal_handler(_signum, _frame)	Signal handler.
sortAtomsForGromacs()	Re-sort atoms for gromacs, which expects all hydrogens to immediately follow the heavy atom they are bonded to and belong to the same charge group.
writeCharmmTopolFiles()	Write CHARMM topology files.
writeCnsTopolFiles()	Write CNS topology files.
writeGroFile()	Write GRO files.
writeGromacsTop()	Write GMX topology file.
writeGromacsTopolFiles()	Write GMX topology Files.
writeMdpFiles()	Write MDP for test with GROMACS.
writePdb(afile)	Write a new PDB file with the atom names defined by Antechamber.
writePosreFile([fc])	Write file with positional restraints for heavy atoms.

balanceCharges(*chargeList*, *FirstNonSoluteId=None*)

Spread charges fractions among atoms to balance system's total charge.

Note that python is very annoying about floating points. Even after balance, there will always be some residue of order e^{-12} to e^{-16}, which is believed to vanished once one writes a topology file, say, for CNS or GMX, where floats are represented with 4 or 5 maximum decimals.

checkFrcmod()

Check FRCMOD file.

checkLeapLog(*log*)

Check Leap log.

checkSmiles()

Check if input arg is a SMILES string.

Returns
True/False

Return type
bool

checkXyzAndTopFiles()

Check XYZ and TOP files.

convertPdbToMol2()

Convert PDB to MOL2 by using obabel.

convertSmilesToMol2()

Convert Smiles to MOL2 by using obabel.

createACTopol()

If successful, Amber Top and Xyz files will be generated.

createMolTopol()

Create MolTopol obj.

delOutputFiles()

Delete temporary output files.

execAntechamber(*chargeType=None*, *atomType=None*) → bool

To call Antechamber and execute it.

Parameters

- **chargeType** (*[str], optional*) – bcc, gas or user. Defaults to None/bcc.
- **atomType** (*[str], optional*) – gaff, amber, gaff2, amber2. Defaults to None/gaff2.

Returns

True if failed.

Return type

bool

```
Welcome to antechamber 21.0: molecular input file processor.

Usage: antechamber \
        -i     input file name
        -fi    input file format
        -o     output file name
        -fo    output file format
        -c     charge method
        -cf    charge file name
        -nc    net molecular charge (int)
        -a     additional file name
        -fa    additional file format
        -ao    additional file operation
                crd    : only read in coordinate
                crg    : only read in charge
                radius: only read in radius
                name   : only read in atom name
                type   : only read in atom type
                bond   : only read in bond type
        -m     multiplicity (2S+1), default is 1
        -rn    residue name, overrides input file, default is MOL
        -rf    residue topology file name in prep input file,
                default is molecule.res
        -ch    check file name for gaussian, default is 'molecule'
        -ek    mopac or sqm keyword, inside quotes; overwrites previous ones
        -gk    gaussian job keyword, inside quotes, is ignored when both -gopt␣
↪and -gsp are used
        -gopt  gaussian job keyword for optimization, inside quotes
        -gsp   gaussian job keyword for single point calculation, inside quotes
        -gm    gaussian memory keyword, inside quotes, such as "%mem=1000MB"
        -gn    gaussian number of processors keyword, inside quotes, such as "
↪%nproc=8"
```

(continues on next page)

(continued from previous page)

```
       -gdsk  gaussian maximum disk usage keyword, inside quotes, such as "
↪%maxdisk=50GB"
       -gv    add keyword to generate gesp file (for Gaussian 09 only)
                1    : yes
                0    : no, the default
       -ge    gaussian esp file generated by iop(6/50=1), default is g09.gesp
       -tor   torsional angle list, inside a pair of quotes, such as "1-2-3-
↪4:0,5-6-7-8"
                ':1' or ':0' indicates the torsional angle is frozen or not
       -df    am1-bcc precharge flag, 2 - use sqm(default); 0 - use mopac
       -at    atom type
                gaff : the default
                gaff2: for gaff2 (beta-version)
                amber: for PARM94/99/99SB
                bcc  : bcc
                sybyl: sybyl
       -du    fix duplicate atom names: yes(y)[default] or no(n)
       -bk    component/block Id, for ccif
       -an    adjust atom names: yes(y) or no(n)
                the default is 'y' for 'mol2' and 'ac' and 'n' for the other␣
↪formats
       -j     atom type and bond type prediction index, default is 4
                0    : no assignment
                1    : atom type
                2    : full  bond types
                3    : part  bond types
                4    : atom and full bond type
                5    : atom and part bond type
       -s     status information: 0(brief), 1(default) or 2(verbose)
       -eq    equalizing atomic charge, default is 1 for '-c resp' and '-c bcc
↪' and 0
                for the other charge methods
                0    : no use
                1    : by atomic paths
                2    : by atomic paths and structural information, i.e. E/Z␣
↪configurations
       -pf    remove intermediate files: yes(y) or no(n)[default]
       -pl    maximum path length to determin equivalence of atomic charges␣
↪for resp and bcc,
                the smaller the value, the faster the algorithm, default is -1␣
↪(use full length),
                set this parameter to 10 to 30 if your molecule is big (# atoms␣
↪>= 100)
       -seq   atomic sequence order changable: yes(y)[default] or no(n)
       -dr    acdoctor mode: yes(y)[default] or no(n)
       -i -o -fi and -fo must appear; others are optional
       Use 'antechamber -L' to list the supported file formats and charge␣
↪methods

                    List of the File Formats

       file format type  abbre. index | file format type abbre. index
```

(continues on next page)

(continued from previous page)

```
-----------------------------------------------------------------
Antechamber        ac       1  | Sybyl Mol2          mol2    2
PDB                pdb      3  | Modified PDB        mpdb    4
AMBER PREP (int)   prepi    5  | AMBER PREP (car)    prepc   6
Gaussian Z-Matrix  gzmat    7  | Gaussian Cartesian  gcrt    8
Mopac Internal     mopint   9  | Mopac Cartesian     mopcrt 10
Gaussian Output    gout    11  | Mopac Output        mopout 12
Alchemy            alc     13  | CSD                 csd    14
MDL                mdl     15  | Hyper               hin    16
AMBER Restart      rst     17  | Jaguar Cartesian    jcrt   18
Jaguar Z-Matrix    jzmat   19  | Jaguar Output       jout   20
Divcon Input       divcrt  21  | Divcon Output       divout 22
SQM Input          sqmcrt  23  | SQM Output          sqmout 24
Charmm             charmm  25  | Gaussian ESP        gesp   26
Component cif      ccif    27  | GAMESS dat          gamess 28
Orca input         orcinp  29  | Orca output         orcout 30
-----------------------------------------------------------------
AMBER restart file can only be read in as additional file.

            List of the Charge Methods

charge method     abbre. index | charge method    abbre. index
-----------------------------------------------------------------
RESP              resp     1  |  AM1-BCC           bcc     2
CM1               cm1      3  |  CM2               cm2     4
ESP (Kollman)     esp      5  |  Mulliken          mul     6
Gasteiger         gas      7  |  Read in charge    rc      8
Write out charge  wc       9  |  Delete Charge     dc     10
-----------------------------------------------------------------
```

execObabel()

Execute obabel.

execParmchk()

Execute parmchk.

execTleap()

Execute tleap.

getABCOEFs()

Get non-bonded coefficients.

getAngles()

Get Angles.

getAtoms()

Set a list with all atoms objects built from dat in acFileTop.

Set also atomType system is gaff or amber, list of atomTypes, resid and system's total charge.

getBonds()

Get Bonds.

getChirals()

Get chiral atoms (for CNS only!).

Plus its 4 neighbours and improper dihedral angles to store non-planar improper dihedrals.

getCoords()

For a given acFileXyz file, return a list of coords as:

```
[[x1,y1,z1],[x2,y2,z2], etc.]
```

getDihedrals()

Get dihedrals (proper and imp), condensed list of prop dih and atomPairs.

getFlagData(*flag*)

For a given acFileTop flag, return a list of the data related.

getResidueLabel()

Get a 3 capital letters code from acFileTop.

Returns a list.

guessCharge()

Guess the charge of a system based on antechamber.

Returns None in case of error.

locateDat(*aFile*)

Locate a file pertinent to $AMBERHOME/dat/leap/parm/.

makeDir()

Make Dir.

pickleSave()

Create portable serialized representations of System's Python objects.

Example

to restore:

```
from acpype import *
# import cPickle as pickle
import pickle
mol = pickle.load(open('DDD.pkl','rb'))
```

printDebug(*text=''*)

Debug log level.

printDebugQuoted(*text=''*)

Print quoted messages.

printError(*text=''*)

Error log level.

printErrorQuoted(*text=''*)

Print quoted messages.

printMess(*text=''*)

Info log level.

printWarn(*text=''*)

Warn log level.

readMol2TotalCharge(*mol2File*)

Reads the charges in given mol2 file and returns the total.

search(*name=None*, *alist=False*)

Returns a list with all atomName matching 'name' or just the first case.

setAtomType4Gromacs()

Set atom types for Gromacs.

Atom types names in Gromacs TOP file are not case sensitive; this routine will append a '_' to lower case atom type.

Example

```
>>> CA and ca -> CA and ca_
```

setProperDihedralsCoef()

Create proper dihedrals list with Ryckaert-Bellemans coefficients.

It takes self.condensedProperDihedrals and returns self.properDihedralsCoefRB, a reduced list of quartet atoms + RB. Coefficients ready for GMX (multiplied by 4.184):

```
self.properDihedralsCoefRB = [ [atom1,..., atom4], C[0:5] ]
```

For proper dihedrals: a quartet of atoms may appear with more than one set of parameters and to convert to GMX they are treated as RBs.

The resulting coefficients calculated here may look slighted different from the ones calculated by amb2gmx.pl because python is taken full float number from prmtop and not rounded numbers from rd-parm.out as amb2gmx.pl does.

setResNameCheckCoords()

Set a 3 letter residue name and check coords for issues like duplication, atoms too close or too sparse.

signal_handler(*_signum*, *_frame*)

Signal handler.

sortAtomsForGromacs()

Re-sort atoms for gromacs, which expects all hydrogens to immediately follow the heavy atom they are bonded to and belong to the same charge group.

Currently, atom mass < 1.2 is taken to denote a proton. This behaviour may be changed by modifying the 'is_hydrogen' function within.

JDC 2011-02-03

writeCharmmTopolFiles()

Write CHARMM topology files.

writeCnsTopolFiles()

Write CNS topology files.

writeGroFile()

Write GRO files.

writeGromacsTop()

Write GMX topology file.

writeGromacsTopolFiles()

Write GMX topology Files.

```
# from ~/Programmes/amber10/dat/leap/parm/gaff.dat
#atom type        atomic mass        atomic polarizability        comments
ca                12.01                 0.360                    Sp2 C in pure␣
↪aromatic systems
ha                1.008                 0.135                    H bonded to␣
↪aromatic carbon

#bonded atoms        harmonic force kcal/mol/A^2       eq. dist. Ang.  comments
ca-ha                  344.3*                           1.087**         SOURCE3␣
↪ 1496    0.0024    0.0045
* for gmx: 344.3 * 4.184 * 100 * 2 = 288110 kJ/mol/nm^2 (why factor 2?)
** convert Ang to nm ( div by 10) for gmx: 1.087 A = 0.1087 nm
# CA HA      1    0.10800   307105.6 ; ged from 340. bsd on C6H6 nmodes; PHE,
↪TRP,TYR (from ffamber99bon.itp)
# CA-HA  367.0    1.080       changed from 340. bsd on C6H6 nmodes; PHE,TRP,TYR␣
↪(from parm99.dat)

# angle        HF kcal/mol/rad^2    eq angle degrees     comments
ca-ca-ha        48.5*             120.01                SOURCE3 2980   0.1509  ␣
↪0.2511
* to convert to gmx: 48.5 * 4.184 * 2 = 405.848 kJ/mol/rad^2 (why factor 2?)
# CA  CA  HA           1   120.000    418.400 ; new99 (from ffamber99bon.itp)
# CA-CA-HA    50.0      120.00 (from parm99.dat)

# dihedral    idivf        barrier hight/2 kcal/mol  phase degrees      ␣
↪periodicity     comments
X -ca-ca-X    4           14.500*                    180.000             2.000 ␣
↪         intrpol.bsd.on C6H6
*convert 2 gmx: 14.5/4 * 4.184 * 2 (?) (yes in amb2gmx, not in topolbuild, why?
↪) = 30.334 or 15.167 kJ/mol
# X -CA-CA-X    4   14.50        180.0     2.         intrpol.bsd.on C6H6 (from␣
↪parm99.dat)
# X CA CA  X    3   30.334       0.000   -30.33400     0.000     0.000     0.
↪000   ; intrpol.bsd.on C6H6
;propers treated as RBs in GMX to use combine multiple AMBER torsions per␣
↪quartet (from ffamber99bon.itp)

# impr. dihedral        barrier hight/2      phase degrees       periodicity   ␣
↪ comments
X -X -ca-ha             1.1*                  180.                      2.     ␣
↪ bsd.on C6H6 nmodes
* to convert to gmx: 1.1 * 4.184 = 4.6024 kJ/mol/rad^2
# X -X -CA-HA         1.1          180.          2.           bsd.on C6H6␣
↪nmodes (from parm99.dat)
# X   X   CA  HA       1      180.00     4.60240     2      ; bsd.on C6H6 nmodes
;impropers treated as propers in GROMACS to use correct AMBER analytical␣
↪function (from ffamber99bon.itp)
```

(continues on next page)

(continued from previous page)

```
# 6-12 parms     sigma = 2 * r * 2^(-1/6)    epsilon
# atomtype        radius Ang.                    pot. well depth kcal/mol
↪comments
ha                   1.4590*                     0.0150**
↪  Spellmeyer
ca                   1.9080                     0.0860
↪OPLS
* to convert to gmx:
    sigma = 1.4590 * 2^(-1/6) * 2 = 2 * 1.29982 Ang. = 2 * 0.129982 nm  = 1.
↪4590 * 2^(5/6)/10 =  0.259964 nm
** to convert to gmx: 0.0150 * 4.184 = 0.06276 kJ/mol
# amber99_3    CA     0.0000  0.0000  A   3.39967e-01  3.59824e-01 (from
↪ffamber99nb.itp)
# amber99_22   HA     0.0000  0.0000  A   2.59964e-01  6.27600e-02 (from
↪ffamber99nb.itp)
# C*          1.9080  0.0860             Spellmeyer
# HA          1.4590  0.0150             Spellmeyer (from parm99.dat)
# to convert r and epsilon to ACOEF and BCOEF
# ACOEF = sqrt(e1*e2) * (r1 + r2)^12 ; BCOEF = 2 * sqrt(e1*e2) * (r1 + r2)^6 =
↪2 * ACOEF/(r1+r2)^6
# to convert ACOEF and BCOEF to r and epsilon
# r = 0.5 * (2*ACOEF/BCOEF)^(1/6); ep = BCOEF^2/(4*ACOEF)
# to convert ACOEF and BCOEF to sigma and epsilon (GMX)
# sigma = (ACOEF/BCOEF)^(1/6) * 0.1 ; epsilon = 4.184 * BCOEF^2/(4*ACOEF)
#   ca    ca       819971.66        531.10
#   ca    ha        76245.15        104.66
#   ha    ha         5716.30         18.52
```

For proper dihedrals: a quartet of atoms may appear with more than one set of parameters and to convert to GMX they are treated as RBs; use the algorithm:

```
for(my $j=$i;$j<=$lines;$j++){
    my $period = $pn{$j};
    if($pk{$j}>0) {
    $V[$period] = 2*$pk{$j}*$cal;
    }
    # assign V values to C values as predefined #
    if($period==1){
    $C[0]+=0.5*$V[$period];
    if($phase{$j}==0){
        $C[1]-=0.5*$V[$period];
    }else{
        $C[1]+=0.5*$V[$period];
    }
    }elsif($period==2){
    if(($phase{$j}==180)||($phase{$j}==3.14)){
        $C[0]+=$V[$period];
        $C[2]-=$V[$period];
    }else{
        $C[2]+=$V[$period];
    }
```

(continues on next page)

(continued from previous page)

```
    }elsif($period==3){
    $C[0]+=0.5*$V[$period];
    if($phase{$j}==0){
        $C[1]+=1.5*$V[$period];
        $C[3]-=2*$V[$period];
    }else{
        $C[1]-=1.5*$V[$period];
        $C[3]+=2*$V[$period];
    }
    }elsif($period==4){
    if(($phase{$j}==180)||($phase{$j}==3.14)){
        $C[2]+=4*$V[$period];
        $C[4]-=4*$V[$period];
    }else{
        $C[0]+=$V[$period];
        $C[2]-=4*$V[$period];
        $C[4]+=4*$V[$period];
    }
    }
}
```

writeMdpFiles()

Write MDP for test with GROMACS.

writePdb(*afile*)

Write a new PDB file with the atom names defined by Antechamber.

The format generated here use is slightly different from:

> old: http://www.wwpdb.org/documentation/file-format-content/format23/sect9.html latest: http://www.wwpdb.org/documentation/file-format-content/format33/sect9.html

respected to atom name. Using GAFF2 atom types:

```
CU/Cu Copper, CL/cl Chlorine, BR/br Bromine
```

Parameters

afile (*[str]*) – file path name

writePosreFile(*fc=1000*)

Write file with positional restraints for heavy atoms.

http://www.mdtutorials.com/gmx/complex/06_equil.html

1.7.3 acpype.topol.MolTopol

class acpype.topol.**MolTopol**(*acTopolObj=None*, *acFileXyz=None*, *acFileTop=None*, *debug=False*, *basename=None*, *verbose=True*, *gmx4=False*, *merge=False*, *direct=False*, *is_sorted=False*, *chiral=False*, *amb2gmx=False*, *level=20*)

Bases: *AbstractTopol*

Class to write topologies and parameters files for several applications.

https://ambermd.org/FileFormats.php

Parser, take information in AC xyz and top files and convert to objects.

Parameters

- **acFileXyz** –
- **acFileTop** –

Returns

molTopol obj or None

__init__(*acTopolObj=None*, *acFileXyz=None*, *acFileTop=None*, *debug=False*, *basename=None*, *verbose=True*, *gmx4=False*, *merge=False*, *direct=False*, *is_sorted=False*, *chiral=False*, *amb2gmx=False*, *level=20*)

Methods

__init__([acTopolObj, acFileXyz, acFileTop, ...])	
balanceCharges(chargeList[, FirstNonSoluteId])	Spread charges fractions among atoms to balance system's total charge.
checkFrcmod()	Check FRCMOD file.
checkLeapLog(log)	Check Leap log.
checkSmiles()	Check if input arg is a SMILES string.
checkXyzAndTopFiles()	Check XYZ and TOP files.
convertPdbToMol2()	Convert PDB to MOL2 by using obabel.
convertSmilesToMol2()	Convert Smiles to MOL2 by using obabel.
createACTopol()	If successful, Amber Top and Xyz files will be generated.
createMolTopol()	Create MolTopol obj.
delOutputFiles()	Delete temporary output files.
execAntechamber([chargeType, atomType])	To call Antechamber and execute it.
execObabel()	Execute obabel.
execParmchk()	Execute parmchk.
execTleap()	Execute tleap.
getABCOEFs()	Get non-bonded coefficients.
getAngles()	Get Angles.
getAtoms()	Set a list with all atoms objects built from dat in acFileTop.
getBonds()	Get Bonds.
getChirals()	Get chiral atoms (for CNS only!).
getCoords()	For a given acFileXyz file, return a list of coords as.

continues on next page

Table 3 – continued from previous page

getDihedrals()	Get dihedrals (proper and imp), condensed list of prop dih and atomPairs.
getFlagData(flag)	For a given acFileTop flag, return a list of the data related.
getResidueLabel()	Get a 3 capital letters code from acFileTop.
guessCharge()	Guess the charge of a system based on antechamber.
locateDat(aFile)	Locate a file pertinent to $AMBERHOME/dat/leap/parm/.
makeDir()	Make Dir.
pickleSave()	Create portable serialized representations of System's Python objects.
printDebug([text])	Debug log level.
printDebugQuoted([text])	Print quoted messages.
printError([text])	Error log level.
printErrorQuoted([text])	Print quoted messages.
printMess([text])	Info log level.
printWarn([text])	Warn log level.
readMol2TotalCharge(mol2File)	Reads the charges in given mol2 file and returns the total.
search([name, alist])	Returns a list with all atomName matching 'name' or just the first case.
setAtomType4Gromacs()	Set atom types for Gromacs.
setProperDihedralsCoef()	Create proper dihedrals list with Ryckaert-Bellemans coefficients.
setResNameCheckCoords()	Set a 3 letter residue name and check coords for issues like duplication, atoms too close or too sparse.
signal_handler(_signum, _frame)	Signal handler.
sortAtomsForGromacs()	Re-sort atoms for gromacs, which expects all hydrogens to immediately follow the heavy atom they are bonded to and belong to the same charge group.
writeCharmmTopolFiles()	Write CHARMM topology files.
writeCnsTopolFiles()	Write CNS topology files.
writeGroFile()	Write GRO files.
writeGromacsTop()	Write GMX topology file.
writeGromacsTopolFiles()	Write GMX topology Files.
writeMdpFiles()	Write MDP for test with GROMACS.
writePdb(afile)	Write a new PDB file with the atom names defined by Antechamber.
writePosreFile([fc])	Write file with positional restraints for heavy atoms.

balanceCharges(*chargeList*, *FirstNonSoluteId=None*)

Spread charges fractions among atoms to balance system's total charge.

Note that python is very annoying about floating points. Even after balance, there will always be some residue of order e^{-12} to e^{-16}, which is believed to vanished once one writes a topology file, say, for CNS or GMX, where floats are represented with 4 or 5 maximum decimals.

checkFrcmod()

Check FRCMOD file.

checkLeapLog(*log*)

Check Leap log.

checkSmiles()

Check if input arg is a SMILES string.

Returns
True/False

Return type
bool

checkXyzAndTopFiles()

Check XYZ and TOP files.

convertPdbToMol2()

Convert PDB to MOL2 by using obabel.

convertSmilesToMol2()

Convert Smiles to MOL2 by using obabel.

createACTopol()

If successful, Amber Top and Xyz files will be generated.

createMolTopol()

Create MolTopol obj.

delOutputFiles()

Delete temporary output files.

execAntechamber(*chargeType=None*, *atomType=None*) → bool

To call Antechamber and execute it.

Parameters

- **chargeType** (*[str], optional*) – bcc, gas or user. Defaults to None/bcc.
- **atomType** (*[str], optional*) – gaff, amber, gaff2, amber2. Defaults to None/gaff2.

Returns
True if failed.

Return type
bool

```
Welcome to antechamber 21.0: molecular input file processor.

Usage: antechamber \
        -i     input file name
        -fi    input file format
        -o     output file name
        -fo    output file format
        -c     charge method
        -cf    charge file name
        -nc    net molecular charge (int)
        -a     additional file name
        -fa    additional file format
        -ao    additional file operation
                crd    : only read in coordinate
                crg    : only read in charge
                radius: only read in radius
```

(continues on next page)

(continued from previous page)

```
                name  : only read in atom name
                type  : only read in atom type
                bond  : only read in bond type
        -m      multiplicity (2S+1), default is 1
        -rn     residue name, overrides input file, default is MOL
        -rf     residue topology file name in prep input file,
                default is molecule.res
        -ch     check file name for gaussian, default is 'molecule'
        -ek     mopac or sqm keyword, inside quotes; overwrites previous ones
        -gk     gaussian job keyword, inside quotes, is ignored when both -gopt␣
↪and -gsp are used
        -gopt   gaussian job keyword for optimization, inside quotes
        -gsp    gaussian job keyword for single point calculation, inside quotes
        -gm     gaussian memory keyword, inside quotes, such as "%mem=1000MB"
        -gn     gaussian number of processors keyword, inside quotes, such as "
↪%nproc=8"
        -gdsk   gaussian maximum disk usage keyword, inside quotes, such as "
↪%maxdisk=50GB"
        -gv     add keyword to generate gesp file (for Gaussian 09 only)
                1     : yes
                0     : no, the default
        -ge     gaussian esp file generated by iop(6/50=1), default is g09.gesp
        -tor    torsional angle list, inside a pair of quotes, such as "1-2-3-
↪4:0,5-6-7-8"
                ':1' or ':0' indicates the torsional angle is frozen or not
        -df     am1-bcc precharge flag, 2 - use sqm(default); 0 - use mopac
        -at     atom type
                gaff : the default
                gaff2: for gaff2 (beta-version)
                amber: for PARM94/99/99SB
                bcc  : bcc
                sybyl: sybyl
        -du     fix duplicate atom names: yes(y)[default] or no(n)
        -bk     component/block Id, for ccif
        -an     adjust atom names: yes(y) or no(n)
                the default is 'y' for 'mol2' and 'ac' and 'n' for the other␣
↪formats
        -j      atom type and bond type prediction index, default is 4
                0     : no assignment
                1     : atom type
                2     : full  bond types
                3     : part  bond types
                4     : atom and full bond type
                5     : atom and part bond type
        -s      status information: 0(brief), 1(default) or 2(verbose)
        -eq     equalizing atomic charge, default is 1 for '-c resp' and '-c bcc
↪' and 0
                for the other charge methods
                0     : no use
                1     : by atomic paths
                2     : by atomic paths and structural information, i.e. E/Z␣
↪configurations
```

(continues on next page)

(continued from previous page)

```
       -pf    remove intermediate files: yes(y) or no(n)[default]
       -pl    maximum path length to determin equivalence of atomic charges␣
↪for resp and bcc,
               the smaller the value, the faster the algorithm, default is -1␣
↪(use full length),
               set this parameter to 10 to 30 if your molecule is big (# atoms␣
↪>= 100)
       -seq   atomic sequence order changable: yes(y)[default] or no(n)
       -dr    acdoctor mode: yes(y)[default] or no(n)
       -i -o -fi and -fo must appear; others are optional
       Use 'antechamber -L' to list the supported file formats and charge␣
↪methods

                  List of the File Formats

       file format type  abbre. index | file format type abbre. index
       ------------------------------------------------------------------
       Antechamber       ac       1  | Sybyl Mol2         mol2    2
       PDB               pdb      3  | Modified PDB       mpdb    4
       AMBER PREP (int)  prepi    5  | AMBER PREP (car)   prepc   6
       Gaussian Z-Matrix gzmat    7  | Gaussian Cartesian gcrt    8
       Mopac Internal    mopint   9  | Mopac Cartesian    mopcrt 10
       Gaussian Output   gout    11  | Mopac Output       mopout 12
       Alchemy           alc     13  | CSD                csd    14
       MDL               mdl     15  | Hyper              hin    16
       AMBER Restart     rst     17  | Jaguar Cartesian   jcrt   18
       Jaguar Z-Matrix   jzmat   19  | Jaguar Output      jout   20
       Divcon Input      divcrt  21  | Divcon Output      divout 22
       SQM Input         sqmcrt  23  | SQM Output         sqmout 24
       Charmm            charmm  25  | Gaussian ESP       gesp   26
       Component cif     ccif    27  | GAMESS dat         gamess 28
       Orca input        orcinp  29  | Orca output        orcout 30
       ------------------------------------------------------------------
       AMBER restart file can only be read in as additional file.

                  List of the Charge Methods

       charge method     abbre. index | charge method    abbre. index
       ------------------------------------------------------------------
       RESP              resp     1  |  AM1-BCC          bcc     2
       CM1               cm1      3  |  CM2              cm2     4
       ESP (Kollman)     esp      5  |  Mulliken         mul     6
       Gasteiger         gas      7  |  Read in charge   rc      8
       Write out charge  wc       9  |  Delete Charge    dc     10
       ------------------------------------------------------------------
```

execObabel()

Execute obabel.

execParmchk()

Execute parmchk.

execTleap()

Execute tleap.

getABCOEFs()

Get non-bonded coefficients.

getAngles()

Get Angles.

getAtoms()

Set a list with all atoms objects built from dat in acFileTop.

Set also atomType system is gaff or amber, list of atomTypes, resid and system's total charge.

getBonds()

Get Bonds.

getChirals()

Get chiral atoms (for CNS only!).

Plus its 4 neighbours and improper dihedral angles to store non-planar improper dihedrals.

getCoords()

For a given acFileXyz file, return a list of coords as:

```
[[x1,y1,z1],[x2,y2,z2], etc.]
```

getDihedrals()

Get dihedrals (proper and imp), condensed list of prop dih and atomPairs.

getFlagData(*flag*)

For a given acFileTop flag, return a list of the data related.

getResidueLabel()

Get a 3 capital letters code from acFileTop.

Returns a list.

guessCharge()

Guess the charge of a system based on antechamber.

Returns None in case of error.

locateDat(*aFile*)

Locate a file pertinent to $AMBERHOME/dat/leap/parm/.

makeDir()

Make Dir.

pickleSave()

Create portable serialized representations of System's Python objects.

Example

to restore:

```
from acpype import *
# import cPickle as pickle
import pickle
mol = pickle.load(open('DDD.pkl','rb'))
```

printDebug(*text=''*)

Debug log level.

printDebugQuoted(*text=''*)

Print quoted messages.

printError(*text=''*)

Error log level.

printErrorQuoted(*text=''*)

Print quoted messages.

printMess(*text=''*)

Info log level.

printWarn(*text=''*)

Warn log level.

readMol2TotalCharge(*mol2File*)

Reads the charges in given mol2 file and returns the total.

search(*name=None*, *alist=False*)

Returns a list with all atomName matching 'name' or just the first case.

setAtomType4Gromacs()

Set atom types for Gromacs.

Atom types names in Gromacs TOP file are not case sensitive; this routine will append a '_' to lower case atom type.

Example

```
>>> CA and ca -> CA and ca_
```

setProperDihedralsCoef()

Create proper dihedrals list with Ryckaert-Bellemans coefficients.

It takes self.condensedProperDihedrals and returns self.properDihedralsCoefRB, a reduced list of quartet atoms + RB. Coefficients ready for GMX (multiplied by 4.184):

```
self.properDihedralsCoefRB = [ [atom1,..., atom4], C[0:5] ]
```

For proper dihedrals: a quartet of atoms may appear with more than one set of parameters and to convert to GMX they are treated as RBs.

The resulting coefficients calculated here may look slighted different from the ones calculated by amb2gmx.pl because python is taken full float number from prmtop and not rounded numbers from rdparm.out as amb2gmx.pl does.

setResNameCheckCoords()

Set a 3 letter residue name and check coords for issues like duplication, atoms too close or too sparse.

signal_handler(*_signum*, *_frame*)

Signal handler.

sortAtomsForGromacs()

Re-sort atoms for gromacs, which expects all hydrogens to immediately follow the heavy atom they are bonded to and belong to the same charge group.

Currently, atom mass < 1.2 is taken to denote a proton. This behaviour may be changed by modifying the 'is_hydrogen' function within.

JDC 2011-02-03

writeCharmmTopolFiles()

Write CHARMM topology files.

writeCnsTopolFiles()

Write CNS topology files.

writeGroFile()

Write GRO files.

writeGromacsTop()

Write GMX topology file.

writeGromacsTopolFiles()

Write GMX topology Files.

```
# from ~/Programmes/amber10/dat/leap/parm/gaff.dat
#atom type        atomic mass        atomic polarizability        comments
ca                12.01                 0.360                   Sp2 C in pure␣
↪aromatic systems
ha                1.008                 0.135                   H bonded to␣
↪aromatic carbon

#bonded atoms        harmonic force kcal/mol/A^2       eq. dist. Ang.  comments
ca-ha                  344.3*                           1.087**         SOURCE3␣
↪ 1496    0.0024    0.0045
* for gmx: 344.3 * 4.184 * 100 * 2 = 288110 kJ/mol/nm^2 (why factor 2?)
** convert Ang to nm ( div by 10) for gmx: 1.087 A = 0.1087 nm
# CA HA      1    0.10800   307105.6 ; ged from 340. bsd on C6H6 nmodes; PHE,
↪TRP,TYR (from ffamber99bon.itp)
# CA-HA  367.0    1.080       changed from 340. bsd on C6H6 nmodes; PHE,TRP,TYR␣
↪(from parm99.dat)

# angle        HF kcal/mol/rad^2    eq angle degrees     comments
ca-ca-ha        48.5*             120.01                SOURCE3 2980   0.1509  ␣
↪0.2511
* to convert to gmx: 48.5 * 4.184 * 2 = 405.848 kJ/mol/rad^2 (why factor 2?)
# CA  CA  HA           1   120.000    418.400 ; new99 (from ffamber99bon.itp)
# CA-CA-HA    50.0      120.00 (from parm99.dat)

# dihedral    idivf        barrier hight/2 kcal/mol  phase degrees      ␣
↪periodicity     comments
```

(continues on next page)

(continued from previous page)

```
X -ca-ca-X    4           14.500*                    180.000             2.000 ␣
↪         intrpol.bsd.on C6H6
*convert 2 gmx: 14.5/4 * 4.184 * 2 (?) (yes in amb2gmx, not in topolbuild, why?
↪) = 30.334 or 15.167 kJ/mol
# X -CA-CA-X    4   14.50        180.0     2.         intrpol.bsd.on C6H6 (from␣
↪parm99.dat)
# X CA CA  X    3   30.334       0.000   -30.33400     0.000     0.000     0.
↪000   ; intrpol.bsd.on C6H6
;propers treated as RBs in GMX to use combine multiple AMBER torsions per␣
↪quartet (from ffamber99bon.itp)

# impr. dihedral        barrier hight/2      phase degrees       periodicity    ␣
↪ comments
X -X -ca-ha             1.1*                  180.                        2.      ␣
↪ bsd.on C6H6 nmodes
* to convert to gmx: 1.1 * 4.184 = 4.6024 kJ/mol/rad^2
# X -X -CA-HA         1.1          180.          2.           bsd.on C6H6␣
↪nmodes (from parm99.dat)
# X   X   CA  HA       1      180.00     4.60240     2      ; bsd.on C6H6 nmodes
;impropers treated as propers in GROMACS to use correct AMBER analytical␣
↪function (from ffamber99bon.itp)

# 6-12 parms     sigma = 2 * r * 2^(-1/6)    epsilon
# atomtype        radius Ang.                    pot. well depth kcal/mol     ␣
↪comments
ha                  1.4590*                     0.0150**                      ␣
↪  Spellmeyer
ca                  1.9080                     0.0860                         ␣
↪OPLS
* to convert to gmx:
    sigma = 1.4590 * 2^(-1/6) * 2 = 2 * 1.29982 Ang. = 2 * 0.129982 nm  = 1.
↪4590 * 2^(5/6)/10 =  0.259964 nm
** to convert to gmx: 0.0150 * 4.184 = 0.06276 kJ/mol
# amber99_3    CA     0.0000  0.0000  A   3.39967e-01  3.59824e-01 (from␣
↪ffamber99nb.itp)
# amber99_22   HA     0.0000  0.0000  A   2.59964e-01  6.27600e-02 (from␣
↪ffamber99nb.itp)
# C*          1.9080  0.0860             Spellmeyer
# HA          1.4590  0.0150             Spellmeyer (from parm99.dat)
# to convert r and epsilon to ACOEF and BCOEF
# ACOEF = sqrt(e1*e2) * (r1 + r2)^12 ; BCOEF = 2 * sqrt(e1*e2) * (r1 + r2)^6 =␣
↪2 * ACOEF/(r1+r2)^6
# to convert ACOEF and BCOEF to r and epsilon
# r = 0.5 * (2*ACOEF/BCOEF)^(1/6); ep = BCOEF^2/(4*ACOEF)
# to convert ACOEF and BCOEF to sigma and epsilon (GMX)
# sigma = (ACOEF/BCOEF)^(1/6) * 0.1 ; epsilon = 4.184 * BCOEF^2/(4*ACOEF)
#   ca   ca       819971.66        531.10
#   ca   ha        76245.15        104.66
#   ha   ha         5716.30         18.52
```

For proper dihedrals: a quartet of atoms may appear with more than one set of parameters and to convert to GMX they are treated as RBs; use the algorithm:

```
for(my $j=$i;$j<=$lines;$j++){
    my $period = $pn{$j};
    if($pk{$j}>0) {
    $V[$period] = 2*$pk{$j}*$cal;
    }
    # assign V values to C values as predefined #
    if($period==1){
    $C[0]+=0.5*$V[$period];
    if($phase{$j}==0){
        $C[1]-=0.5*$V[$period];
    }else{
        $C[1]+=0.5*$V[$period];
    }
    }elsif($period==2){
    if(($phase{$j}==180)||($phase{$j}==3.14)){
        $C[0]+=$V[$period];
        $C[2]-=$V[$period];
    }else{
        $C[2]+=$V[$period];
    }
    }elsif($period==3){
    $C[0]+=0.5*$V[$period];
    if($phase{$j}==0){
        $C[1]+=1.5*$V[$period];
        $C[3]-=2*$V[$period];
    }else{
        $C[1]-=1.5*$V[$period];
        $C[3]+=2*$V[$period];
    }
    }elsif($period==4){
    if(($phase{$j}==180)||($phase{$j}==3.14)){
        $C[2]+=4*$V[$period];
        $C[4]-=4*$V[$period];
    }else{
        $C[0]+=$V[$period];
        $C[2]-=4*$V[$period];
        $C[4]+=4*$V[$period];
    }
    }
}
```

writeMdpFiles()

Write MDP for test with GROMACS.

writePdb(*afile*)

Write a new PDB file with the atom names defined by Antechamber.

The format generated here use is slightly different from:

old: http://www.wwpdb.org/documentation/file-format-content/format23/sect9.html latest: http://www.wwpdb.org/documentation/file-format-content/format33/sect9.html

respected to atom name. Using GAFF2 atom types:

```
CU/Cu Copper, CL/cl Chlorine, BR/br Bromine
```

Parameters
afile (*[str]*) – file path name

writePosreFile(*fc=1000*)

Write file with positional restraints for heavy atoms.

http://www.mdtutorials.com/gmx/complex/06_equil.html

1.7.4 acpype.topol.Topology_14

class acpype.topol.**Topology_14**

Bases: object

Amber topology abstraction for non-uniform 1-4 scale factors.

__init__() → None

Methods

__init__()

hasNondefault14()	Check non-uniform 1-4 scale factor.
p7_array_read(buff, flag_string)	Convert AMBER topology data to python array.
patch_gmx_topol14(gmx_init_top)	Patch GMX topology file for non-uniform 1-4 scale factor.
print_gmx_pairs()	Generate non-bonded pairs list.
read_amber_topology(buff)	Read AMBER topology file.
skipline(buff, index)	skip line.

hasNondefault14()

Check non-uniform 1-4 scale factor.

p7_array_read(*buff*, *flag_string*)

Convert AMBER topology data to python array.

patch_gmx_topol14(*gmx_init_top*)

Patch GMX topology file for non-uniform 1-4 scale factor.

print_gmx_pairs()

Generate non-bonded pairs list.

read_amber_topology(*buff*)

Read AMBER topology file.

static skipline(*buff*, *index*)

skip line.

1.8 acpype.utils

Functions

checkOpenBabelVersion()	check openbabel version
cross_product(a, b)	cross product
distanceAA(c1, c2)	Distance between two atoms
dotproduct(aa, bb)	scalar product
elapsedTime(seconds[, add_s, separator])	Takes an amount of seconds and turns it into a human-readable amount of time.
find_bin(abin)	
imprDihAngle(a, b, c, d)	calculate improper dihedral angle
job_pids_family(jpid)	INTERNAL: Return all job processes (PIDs)
length(v)	distance between 2 vectors
parmMerge(fdat1, fdat2[, frcmod])	merge two amber parm dat/frcmod files and save in /tmp
parseFrcmod(lista)	Parse FRCMOD file
set_for_pip(binaries)	
splitBlock(dat)	split a amber parm dat file in blocks 0 = mass, 1 = extra + bond, 2 = angle, 3 = dihedral, 4 = improp, 5 = hbond 6 = equiv nbon, 7 = nbon, 8 = END, 9 = etc.
vec_sub(aa, bb)	vector A - B
while_replace(string)	

1.8.1 acpype.utils.checkOpenBabelVersion

acpype.utils.**checkOpenBabelVersion**()

check openbabel version

1.8.2 acpype.utils.cross_product

acpype.utils.**cross_product**(*a*, *b*)

cross product

1.8.3 acpype.utils.distanceAA

acpype.utils.**distanceAA**(*c1*, *c2*)

Distance between two atoms

1.8.4 acpype.utils.dotproduct

acpype.utils.**dotproduct**(*aa*, *bb*)

scalar product

1.8.5 acpype.utils.elapsedTime

acpype.utils.**elapsedTime**(*seconds*, *add_s=False*, *separator=' '*)

Takes an amount of seconds and turns it into a human-readable amount of time.

1.8.6 acpype.utils.find_bin

acpype.utils.**find_bin**(*abin*)

1.8.7 acpype.utils.imprDihAngle

acpype.utils.**imprDihAngle**(*a*, *b*, *c*, *d*)

calculate improper dihedral angle

1.8.8 acpype.utils.job_pids_family

acpype.utils.**job_pids_family**(*jpid*)

INTERNAL: Return all job processes (PIDs)

1.8.9 acpype.utils.length

acpype.utils.**length**(*v*)

distance between 2 vectors

1.8.10 acpype.utils.parmMerge

acpype.utils.**parmMerge**(*fdat1*, *fdat2*, *frcmod=False*)

merge two amber parm dat/frcmod files and save in /tmp

1.8.11 acpype.utils.parseFrcmod

acpype.utils.**parseFrcmod**(*lista*)

Parse FRCMOD file

1.8.12 acpype.utils.set_for_pip

acpype.utils.**set_for_pip**(*binaries*)

1.8.13 acpype.utils.splitBlock

acpype.utils.**splitBlock**(*dat*)

split a amber parm dat file in blocks 0 = mass, 1 = extra + bond, 2 = angle, 3 = dihedral, 4 = improp, 5 = hbond 6 = equiv nbon, 7 = nbon, 8 = END, 9 = etc.

1.8.14 acpype.utils.vec_sub

acpype.utils.**vec_sub**(*aa*, *bb*)

vector A - B

1.8.15 acpype.utils.while_replace

acpype.utils.**while_replace**(*string*)

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

P

R

S

T

U

V

W