# CONTENTS

## 1 Installation
1.1 Required Software .................................................. 1
1.2 Quick Install ........................................................ 1
1.3 Download & Install .................................................. 1
1.4 Recommended Software .............................................. 2
1.5 Included in ProDy .................................................... 2
1.6 Source Code ........................................................ 2

## 2 Applications
2.1 ProDy Applications .................................................. 3
2.2 Evol Applications ................................................... 16

## 3 Reference Manual
3.1 Atomic Data ........................................................ 31
3.2 Database Support .................................................. 94
3.3 Dynamics Analysis ................................................ 96
3.4 Ensemble Analysis ................................................ 130
3.5 KDTree ............................................................... 137
3.6 Measurement Tools ................................................ 138
3.7 Protein Structure .................................................. 146
3.8 Sequence Analysis ................................................ 166
3.9 Trajectory I/O ...................................................... 174
3.10 ProDy Utilities ...................................................... 185
3.11 Applications API .................................................. 192
3.12 Configuration & Logging ......................................... 204

## 4 Developer’s Guide
4.1 Contributing to ProDy ................................................. 206
4.2 Documenting ProDy ................................................ 208
4.3 How to Make a Release ............................................. 210
4.4 Style Guide for ProDy .............................................. 211
4.5 Testing ProDy ....................................................... 214
4.6 Writing Tutorials .................................................... 214
4.7 Making Windows Installers ....................................... 218
4.8 Cross-platform Issues .............................................. 218

## 5 Release Notes
5.1 ProDy 1.5 Series ................................................... 219
5.2 ProDy 1.4 Series ................................................... 220
5.3  ProDy 1.3 Series ................................................................. 225
5.4  ProDy 1.2 Series ................................................................. 226
5.5  ProDy 1.1 Series ................................................................. 229
5.6  ProDy 1.0 Series ................................................................. 230
5.7  ProDy 0.9 Series ................................................................. 234
5.8  ProDy 0.8 Series ................................................................. 241
5.9  ProDy 0.7 Series ................................................................. 245
5.10 ProDy 0.6 Series ................................................................. 248
5.11 ProDy 0.5 Series ................................................................. 250
5.12 ProDy 0.2 Series ................................................................. 253
5.13 ProDy 0.1 Series ................................................................. 254

6  About ProDy ................................................................. 256
6.1  People ................................................................................. 256
6.2  Citing ................................................................................. 257
6.3  Credits ................................................................................. 257
6.4  Funding ............................................................................... 257
6.5  License ............................................................................... 258

Bibliography ................................................................. 260
Python Module Index ................................................................. 262
Index ....................................................................................... 264
CHAPTER ONE

INSTALLATION

1.1 Required Software

- **Python**\(^1\) 2.6, 2.7, 3.2 or later
  
  *Windows*: You need to use **32-bit** Python on Windows to be able to install NumPy and ProDy.

- **NumPy**\(^2\) 1.7 or later

When compiling from source, on Linux for example, you will need a C compiler (e.g. gcc) and Python developer libraries (i.e. python.h). If you don’t have Python developer libraries installed on your machine, use your package manager to install `python-dev` package.

In addition, `matplotlib`\(^3\) is required for using plotting functions. ProDy, *ProDy Applications* (page 3), and *Evol Applications* (page 16) can be operated without this package.

1.2 Quick Install

If you have `pip`\(^4\) installed, type the following:

```
pip install -U ProDy
```

If you don’t have `pip`\(^5\), please download an installation file and follow the instructions.

1.3 Download & Install

After installing the required packages, you will need to download a suitable ProDy source or installation file from [http://python.org/pypi/ProDy](http://python.org/pypi/ProDy). For changes and list of new features see *Release Notes* (page 219).

*Linux*

Download `ProDy-x.y.z.tar.gz`. Extract tarball contents and run `setup.py` as follows:

---

\(^1\) [http://www.python.org](http://www.python.org)

\(^2\) [http://www.numpy.org](http://www.numpy.org)

\(^3\) [http://matplotlib.org](http://matplotlib.org)

\(^4\) [http://www.pip-installer.org](http://www.pip-installer.org)

\(^5\) [http://www.pip-installer.org](http://www.pip-installer.org)
$ tar -xzf ProDy-x.y.z.tar.gz
$ cd ProDy-x.y.z
$ python setup.py build
$ python setup.py install

If you need root access for installation, try `sudo python setup.py install`. If you don’t have root access, please consult alternate and custom installation schemes in Installing Python Modules⁶.

### Mac OS

For installing ProDy, please follow the Linux installation instructions.

### Windows

Remove previously installed ProDy release from Uninstall a program in Control Panel. Download ProDy-0.x.y.win32-py2.z.exe and run to install ProDy.

To be able use ProDy Applications (page 3) and Evol Applications (page 16) in command prompt (cmd.exe), append Python and scripts folders (e.g. `C:\Python27` and `C:\Python27\Scripts`) to PATH⁷ environment variable.

## 1.4 Recommended Software

- Scipy⁸, when installed, replaces linear algebra module of Numpy. Scipy linear algebra module is more flexible and can be faster.
- IPython⁹ is a must have for interactive ProDy sessions.
- PyReadline¹⁰ for colorful IPython sessions on Windows.
- MDAnalysis¹¹ for reading molecular dynamics trajectories.

## 1.5 Included in ProDy

Following software is included in the ProDy installation packages:

- pyparsing¹² is used to define the atom selection grammar.
- Biopython¹³ KDTree package and pairwise2 module are used for distance based atom selections and pairwise sequence alignment, respectively.
- argparse¹⁴ is used to implement applications and provided for compatibility with Python 2.6.

## 1.6 Source Code

Source code is available at https://github.com/prody/ProDy.

---

⁶http://docs.python.org/install/index.html
⁷http://matplotlib.sourceforge.net/faq/environment_variables_faq.html#envvar-PATH
⁸http://www.scipy.org
⁹http://ipython.org
¹⁰http://ipython.org/pyreadline.html
¹¹http://code.google.com/p/mdanalysis
¹²http://pyparsing.wikispaces.com
¹³http://biopython.org
¹⁴http://code.google.com/p/argparse/
ProDy comes with two sets of applications that automate structural dynamics and sequence coevolution analysis:

## 2.1 ProDy Applications

ProDy applications are command line programs that automates structure processing and structural dynamics analysis:

### 2.1.1 prody align

**Usage**

Running **prody align -h** displays:

```
```

```
positional arguments:
pdb PDB identifier(s) or filename(s)
```

```
optional arguments:
-h, --help show this help message and exit
--quiet suppress info messages to stderr
--examples show usage examples and exit
```

```
atom/model selection:
-s SEL, --select SEL reference structure atom selection (default: calpha)
-m INT, --model INT for NMR files, reference model index (default: 1)
```

```
chain matching options:
-i INT, --seqid INT percent sequence identity (default: 90)
-o INT, --overlap INT percent sequence overlap (default: 90)
```

```
output options:
-p STR, --prefix STR output filename prefix (default: PDB filename)
-x STR, --suffix STR output filename suffix (default: _aligned)
```
## Examples

**Running prody align --examples displays:**

Align models in a PDB structure or multiple PDB structures and save aligned coordinate sets. When multiple structures are aligned, ProDy will match chains based on sequence alignment and use best match for aligning the structures.

Fetch PDB structure 2k39 and align models (reference model is the first model):

```bash
$ prody align 2k39
```

Fetch PDB structure 2k39 and align models using backbone of residues with number less than 71:

```bash
$ prody align 2k39 --select "backbone and resnum < 71"
```

Align 1r39 and 1zz2 onto 1p38 using residues with number less than 300:

```bash
$ prody align --select "resnum < 300" 1p38 1r39 1zz2
```

Align all models of 2k39 onto laar using residues 1 to 70 (inclusive):

```bash
$ prody align --select "resnum 1 to 70" laar 2k39
```

Align 1fi7 onto 1hrc using heme atoms:

```bash
$ prody align --select "noh heme and chain A" 1hrc 1fi7
```

### 2.1.2 prody anm

**Usage**

**Running prody anm -h displays:**

```bash
```

**Positional arguments:**

`pdb` PDB identifier or filename

**Optional arguments:**

- `-h`, `--help`: show this help message and exit
- `--quiet`: suppress info messages to stderr
- `--examples`: show usage examples and exit

**Parameters:**

- `-n INT`, `--number-of-modes INT`: number of non-zero eigenvectors (modes) to calculate (default: 10)
-s SEL, --select SEL atom selection (default: "protein and name CA or nucleic and name P C4' C2")
-c FLOAT, --cutoff FLOAT
cutoff distance (Å) (default: 15.0)
-g FLOAT, --gamma FLOAT
spring constant (default: 1.0)
-m INT, --model INT index of model that will be used in the calculations

output:
-a, --all-output write all outputs
-o PATH, --output-dir PATH
output directory (default: .)
-e, --eigenvs write eigenvalues/vectors
-r, --cross-correlations
write cross-correlations
-u, --heatmap write cross-correlations heatmap file
-q, --square-fluctuations
write square-fluctuations
-v, --covariance write covariance matrix
-z, --npz write compressed ProDy data file
-t STR, --extend STR write NMD file for the model extended to "backbone" ("bb") or "all" atoms of the residue, model must have one node per residue
-b, --beta-factors write beta-factors calculated from GNM modes
-l, --hessian write Hessian matrix
-k, --kirchhoff write Kirchhoff matrix

output options:
-p STR, --file-prefix STR
output file prefix (default: pdb_anm)
-f STR, --number-format STR
number output format (default: %12g)
-d STR, --delimiter STR
number delimiter (default: " ")
-x STR, --extension STR
numeric file extension (default: .txt)

figures:
-A, --all-figures save all figures
-R, --cross-correlations-figure save cross-correlations figure
-Q, --square-fluctuations-figure save square-fluctuations figure
-B, --beta-factors-figure save beta-factors figure
-K, --contact-map save contact map (Kirchhoff matrix) figure

figure options:
-F STR, --figure-format STR
pdf (default: pdf)
-D INT, --dpi INT figure resolution (dpi) (default: 300)
-W FLOAT, --width FLOAT
figure width (inch) (default: 8.0)
-H FLOAT, --height FLOAT
figure height (inch) (default: 6.0)
Examples

Running `prody anm --examples` displays:

Perform ANM calculations for given PDB structure and output results in NMD format. If an identifier is passed, structure file will be downloaded from the PDB FTP server.

Fetch PDB 1p38, run ANM calculations using default parameters, and write NMD file:

```bash
$ prody anm 1p38
```

Fetch PDB 1aar, run ANM calculations using default parameters for chain A carbon alpha atoms with residue numbers less than 70, and save all of the graphical output files:

```bash
$ prody anm 1aar -s "calpha and chain A and resnum < 70" -A
```

2.1.3 prody biomol

Usage

Running `prody biomol -h` displays:


positional arguments:
  pdb    PDB identifier or filename

optional arguments:
  -h, --help            show this help message and exit
  --quiet              suppress info messages to stderr
  --examples           show usage examples and exit
  -p STR, --prefix STR  prefix for output files (default: pdb_biomol_)
  -b INT, --biomol INT  index of the biomolecule, by default all are generated

Examples

Running `prody biomol --examples` displays:

Generate biomolecule coordinates:

```bash
$ prody biomol 2bfu
```

2.1.4 prody blast

Usage

Running `prody blast -h` displays:

positional arguments:
  sequence sequence or file in fasta format

optional arguments:
  -h, --help show this help message and exit
  --quiet suppress info messages to stderr
  --examples show usage examples and exit
  -i FLOAT, --identity FLOAT percent sequence identity (default: 90.0)
  -o FLOAT, --overlap FLOAT percent sequence overlap (default: 90.0)
  -d PATH, --output-dir PATH download uncompressed PDB files to given directory
  -z, --gzip write compressed PDB file

Blast Parameters:
  -f STR, --filename STR a filename to save the results in XML format
  -e FLOAT, --expect FLOAT blast search parameter
  -l INT, --hit-list-size INT blast search parameter
  -s INT, --sleep-time INT how long to wait to reconnect for results (sleep time is doubled when results are not ready)
  -t INT, --timeout INT when to give up waiting for results

Examples

Running prody blast --examples displays:

Blast search PDB for the first sequence in a fasta file:

$ prody blast seq.fasta -i 70

Blast search PDB for the sequence argument:

$ prody blast MQIFVKLTGKTLTVEPSTIKVKAKI_QDFIPDQQLFAGEKRQLEDGRDNYI квартира:

Blast search PDB for avidin structures, download files, and align all files onto the 2avi structure:

$ prody blast -d . ARKCSLTGKWNDLGNNAMTSNIEFTGTYTAVTSNIEIKESPLHGQTQNTINKRTQPTFTVNVWKFSESTVFT

$ prody align 2avi.pdb *pdb

2.1.5 prody catdcd

Usage

Running prody catdcd -h displays:

  [-psf PSF] [--pdb PDB] [--first INT] [--last INT]

2.1. ProDy Applications
positional arguments:
  dcd  DCD filename(s) (all must have same number of atoms)

optional arguments:
  -h, --help      show this help message and exit
  --quiet         suppress info messages to stderr
  --examples      show usage examples and exit
  -s SEL, --select SEL  atom selection (default: all)
  -o FILE, --output FILE  output filename (default: trajectory.dcd)
  -n, --num       print the number of frames in each file and exit
  --psf PSF       PSF filename (must have same number of atoms as DCDs)
  --pdb PDB       PDB filename (must have same number of atoms as DCDs)
  --first INT     index of the first output frame, default: 0
  --last INT      index of the last output frame, default: -1
  --stride INT    number of steps between output frames, default: 1
  --align SEL     atom selection for aligning frames, a PSF or PDB file
                  must be provided, if a PDB is provided frames will be
                  superposed onto PDB coordinates

Examples

Running prody catdcd --examples displays:

Concatenate two DCD files and output all atoms:

  $ prody catdcd mdm2.dcd mdm2sim2.dcd

Concatenate two DCD files and output backbone atoms:

  $ prody catdcd mdm2.dcd mdm2sim2.dcd --pdb mdm2.pdb -s bb

2.1.6 prody contacts

Usage

Running prody contacts -h displays:

usage: prody contacts [-h] [--quiet] [--examples] [-s SELSTR] [-r FLOAT]
                   [-t STR] [-p STR] [-x STR]
                   target ligand [ligand ...]

positional arguments:
  target          target PDB identifier or filename
  ligand          ligand PDB identifier(s) or filename(s)

optional arguments:
  -h, --help      show this help message and exit
  --quiet         suppress info messages to stderr
  --examples      show usage examples and exit
  -s SELSTR, --select SELSTR  selection string for target
Running `prody contacts` --examples displays:

Identify contacts of a target structure with one or more ligands.

Fetch PDB structure 1zz2, save PDB files for individual ligands, and identify contacting residues of the target protein:

```bash
$ prody select -o B11 "resname B11" 1zz2
$ prody select -o BOG "resname BOG" 1zz2
$ prody contacts -r 4.0 -t residue -s protein 1zz2 B11.pdb BOG.pdb
```

### 2.1.7 prody eda

**Usage**

Running `prody eda` -h displays:

```bash
```

Positional arguments:
- `dcd` file in DCD or PDB format

Optional arguments:
- `-h, --help` show this help message and exit
- `--quiet` suppress info messages to stderr
- `--examples` show usage examples and exit
- `--psf PSF` PSF filename
- `--pdb PDB` PDB filename
- `--aligned` trajectory is already aligned

Parameters:
- `-n INT, --number-of-modes INT` number of non-zero eigenvectors (modes) to calculate (default: 10)
- `-s SEL, --select SEL` atom selection (default: "protein and name CA or nucleic and name P C4’ C2")

Output:
- `-a, --all-output` write all outputs
- `-o PATH, --output-dir PATH`
output directory (default: .)
-e, --eigenvs write eigenvalues/vectors
-r, --cross-correlations write cross-correlations
-u, --heatmap write cross-correlations heatmap file
-q, --square-fluctuations write square-fluctuations
-v, --covariance write covariance matrix
-z, --npz write compressed ProDy data file
-t STR, --extend STR write NMD file for the model extended to "backbone" ("bb") or "all" atoms of the residue, model must have one node per residue
-j, --projection write projections onto PCs

output options:
-p STR, --file-prefix STR output file prefix (default: pdb_pca)
-f STR, --number-format STR number output format (default: %12g)
-d STR, --delimiter STR number delimiter (default: " ")
-x STR, --extension STR numeric file extension (default: .txt)

figures:
-A, --all-figures save all figures
-R, --cross-correlations-figure save cross-correlations figure
-Q, --square-fluctuations-figure save square-fluctuations figure
-J STR, --projection-figure STR save projections onto specified subspaces, e.g. "1,2" for projections onto PCs 1 and 2; "1,2 1,3" for projections onto PCs 1,2 and 1, 3; "1 1,2,3" for projections onto PCs 1 and 1, 2, 3

figure options:
-F STR, --figure-format STR pdf (default: pdf)
-D INT, --dpi INT figure resolution (dpi) (default: 300)
-W FLOAT, --width FLOAT figure width (inch) (default: 8.0)
-H FLOAT, --height FLOAT figure height (inch) (default: 6.0)

Examples

Running prody eda --examples displays:

This command performs PCA (or EDA) calculations for given multi-model PDB structure or DCD format trajectory file and outputs results in NMD format. If a PDB identifier is given, structure file will be downloaded from the PDB FTP server. DCD files may be accompanied with PDB or PSF files to enable atoms selections.

Fetch pdb 2k39, perform PCA calculations, and output NMD file:
$ prody pca 2k39

Fetch pdb 2k39 and perform calculations for backbone of residues up to 71, and save all output and figure files:

$ prody pca 2k39 --select "backbone and resnum < 71" -a -A

Perform EDA of MDM2 trajectory:

$ prody eda mdm2.dcd

Perform EDA for backbone atoms:

$ prody eda mdm2.dcd --pdb mdm2.pdb --select backbone

### 2.1.8 prody fetch

#### Usage

Running **prody fetch -h** displays:

```
```

positional arguments:

pdb

PDB identifier(s) or a file that contains them

optional arguments:

-h, --help

show this help message and exit

--quiet

suppress info messages to stderr

--examples

show usage examples and exit

-d PATH, --dir PATH

target directory for saving PDB file(s)

-z, --gzip

write compressed PDB file(S)

#### Examples

Running **prody fetch --examples** displays:

Download PDB file(s) by specifying identifiers:

```
$ prody fetch 1mkp 1p38
```

### 2.1.9 prody gnm

#### Usage

Running **prody gnm -h** displays:

```
[-W FLOAT] [-H FLOAT] pdb
```

2.1. ProDy Applications
positional arguments:
  pdb                  PDB identifier or filename

optional arguments:
  -h, --help           show this help message and exit
  --quiet              suppress info messages to stderr
  --examples           show usage examples and exit

parameters:
  -n INT, --number-of-modes INT
    number of non-zero eigenvectors (modes) to calculate
    (default: 10)
  -s SEL, --select SEL
    atom selection (default: "protein and name CA or nucleic and name P C4' C2")
  -c FLOAT, --cutoff FLOAT
    cutoff distance (Å) (default: 10.0)
  -g FLOAT, --gamma FLOAT
    spring constant (default: 1.0)
  -m INT, --model INT
    index of model that will be used in the calculations

output:
  -a, --all-output     write all outputs
  -o PATH, --output-dir PATH
    output directory (default: .)
  -e, --eigenvs        write eigenvalues/vectors
  -r, --cross-correlations
    write cross-correlations
  -u, --heatmap        write cross-correlations heatmap file
  -q, --square-fluctuations
    write square-fluctuations
  -v, --covariance     write covariance matrix
  -z, --npz            write compressed ProDy data file
  -t STR, --extend STR
    write NMD file for the model extended to "backbone" ("bb") or "all" atoms of the residue, model must have one node per residue
  -b, --beta-factors
    write beta-factors calculated from GNM modes
  -k, --kirchhoff      write Kirchhoff matrix

output options:
  -p STR, --file-prefix STR
    output file prefix (default: pdb_gnm)
  -f STR, --number-format STR
    number output format (default: %12g)
  -d STR, --delimiter STR
    number delimiter (default: " ")
  -x STR, --extension STR
    numeric file extension (default: .txt)

figures:
  -A, --all-figures    save all figures
  -R, --cross-correlations-figure
    save cross-correlations figure
  -Q, --square-fluctuations-figure
    save square-fluctuations figure
  -B, --beta-factors-figure
    save beta-factors figure
  -K, --contact-map    save contact map (Kirchhoff matrix) figure
  -M STR, --mode-shape-figure STR
save mode shape figures for specified modes, e.g. "1-3 5" for modes 1, 2, 3 and 5

figure options:
-F STR, --figure-format STR
    pdf (default: pdf)
-D INT, --dpi INT
    figure resolution (dpi) (default: 300)
-W FLOAT, --width FLOAT
    figure width (inch) (default: 8.0)
-H FLOAT, --height FLOAT
    figure height (inch) (default: 6.0)

Examples

Running prody gnm --examples displays:

This command performs GNM calculations for given PDB structure and outputs results in NMD format. If an identifier is passed, structure file will be downloaded from the PDB FTP server.

Fetch PDB 1p38, run GNM calculations using default parameters, and results:

    $ prody gnm 1p38

Fetch PDB 1aar, run GNM calculations with cutoff distance 7 angstrom for chain A carbon alpha atoms with residue numbers less than 70, and save all of the graphical output files:

    $ prody gnm 1aar -c 7 -s "calpha and chain A and resnum < 70" -A

2.1.10 prody pca

Usage

Running prody pca -h displays:

    [-W FLOAT] [-H FLOAT] [--psf PSF | --pdb PDB] [--aligned]
dcd

dcd file in DCD or PDB format

optional arguments:
    -h, --help          show this help message and exit
    --quiet             suppress info messages to stderr
    --examples          show usage examples and exit
    --psf PSF           PSF filename
    --pdb PDB           PDB filename
    --aligned           trajectory is already aligned

parameters:
-n INT, --number-of-modes INT  
  number of non-zero eigenvectors (modes) to calculate  
  (default: 10)
-s SEL, --select SEL  
  atom selection (default: "protein and name CA or nucleic and name P C4’ C2")

**output:**
- a, --all-output  
  write all outputs
-o PATH, --output-dir PATH  
  output directory (default: .)
-e, --eigenvs  
  write eigenvalues/vectors
-r, --cross-correlations  
  write cross-correlations
-u, --heatmap  
  write cross-correlations heatmap file
-g, --square-fluctuations  
  write square-fluctuations
-v, --covariance  
  write covariance matrix
-z, --npz  
  write compressed ProDy data file
-t STR, --extend STR  
  write NMD file for the model extended to "backbone" ("bb") or "all" atoms of the residue, model must have one node per residue
-j, --projection  
  write projections onto PCs

**output options:**
- p STR, --file-prefix STR  
  output file prefix (default: pdb_pca)
-f STR, --number-format STR  
  number output format (default: %12g)
-d STR, --delimiter STR  
  number delimiter (default: " ")
-x STR, --extension STR  
  numeric file extension (default: .txt)

**figures:**
- A, --all-figures  
  save all figures
-R, --cross-correlations-figure  
  save cross-correlations figure
-Q, --square-fluctuations-figure  
  save square-fluctuations figure
-J STR, --projection-figure STR  
  save projections onto specified subspaces, e.g. "1,2" for projections onto PCs 1 and 2; "1,2 1,3" for projections onto PCs 1,2 and 1, 3; "1 1,2,3" for projections onto PCs 1 and 1, 2, 3

**figure options:**
- F STR, --figure-format STR  
  pdf (default: pdf)
-D INT, --dpi INT  
  figure resolution (dpi) (default: 300)
-W FLOAT, --width FLOAT  
  figure width (inch) (default: 8.0)
-H FLOAT, --height FLOAT  
  figure height (inch) (default: 6.0)

**Examples**

Running `prody pca --examples` displays:
This command performs PCA (or EDA) calculations for given multi-model PDB structure or DCD format trajectory file and outputs results in NMD format. If a PDB identifier is given, structure file will be downloaded from the PDB FTP server. DCD files may be accompanied with PDB or PSF files to enable atoms selections.

Fetch pdb 2k39, perform PCA calculations, and output NMD file:

```bash
$ prody pca 2k39
```

Fetch pdb 2k39 and perform calculations for backbone of residues up to 71, and save all output and figure files:

```bash
$ prody pca 2k39 --select "backbone and resnum < 71" -a -A
```

Perform EDA of MDM2 trajectory:

```bash
$ prody eda mdm2.dcd
```

Perform EDA for backbone atoms:

```bash
$ prody eda mdm2.dcd --pdb mdm2.pdb --select backbone
```

### 2.1.11 prody select

**Usage**

Running `prody select -h` displays:

```
                 select pdb [pdb ...]

positional arguments:
  select            atom selection string
  pdb              PDB identifier(s) or filename(s)

optional arguments:
  -h, --help        show this help message and exit
  --quiet           suppress info messages to stderr
  --examples        show usage examples and exit

output options:
  -o STR, --output STR  output PDB filename (default: pdb_selected.pdb)
  -p STR, --prefix STR  output filename prefix (default: PDB filename)
  -x STR, --suffix STR  output filename suffix (default: _selected)
```

**Examples**

Running `prody select --examples` displays:

This command selects specified atoms and writes them in a PDB file.

Fetch PDB files 1p38 and 1r39 and write backbone atoms in a file:

```bash
$ prody select backbone 1p38 1r39
```
Running **prody** command will provide a description of applications:

```
$ prody
```

**usage:** prody [-h] [-c] [-v]
{anm,gnm,pca,eda,align,blast,biomol,catdcd,contacts,fetch,select} ...

**Optional arguments:**
- `-h`, `--help`  show this help message and exit
- `-c`, `--cite`  print citation info and exit
- `-v`, `--version`  print ProDy version and exit

**Subcommands:**
{anm,gnm,pca,eda,align,blast,biomol,catdcd,contacts,fetch,select}
- `anm`  perform anisotropic network model calculations
- `gnm`  perform Gaussian network model calculations
- `pca`  perform principal component analysis calculations
- `eda`  perform essential dynamics analysis calculations
- `align`  align models or structures
- `blast`  blast search Protein Data Bank
- `biomol`  build biomolecules
- `catdcd`  concatenate dcd files
- `contacts`  identify contacts between a target and ligand(s)
- `fetch`  fetch a PDB file
- `select`  select atoms and write a PDB file

See ‘prody <command> -h’ for more information on a specific command.

Detailed information on a specific application can be obtained by typing the command and application names as **prody anm -h**.

Running **prody anm** application as follows will perform ANM calculations for the p38 MAP kinase structure, and will write eigenvalues/vectors in plain text and **NMD Format** (page 118):

```
$ prody anm 1p38
```

In the above example, the default parameters (**cutoff=15.** and **gamma=1.**) and all of the Cα atoms of the protein structure 1p38 are used.

In the example below, the **cutoff** distance is changed to 14 Å, and the Cα atoms of residues with numbers smaller than 340 are used, the output files are prefixed with **p38_anm**:

```
$ prody anm -c 14 -s "calpha resnum < 340" -p p38_anm 1p38
```

The output file **p38_anm.nmd** can be visualized using **NMWiz**\(^1\).

### 2.2 Evol Applications

Evol applications are command line programs that automate retrieval, refinement, and analysis of multiple sequence alignments:

\(^1\)http://csb.pitt.edu/NMWiz
2.2.1 evol coevol

Usage

Running evol coevol -h displays:


positional arguments:
  msa                      refined MSA file

optional arguments:
  -h, --help               show this help message and exit
  --quiet                  suppress info messages to stderr
  --examples               show usage examples and exit

calculation options:
  -n, --no-ambiguity       treat amino acids characters B, Z, J, and X as non-
                           ambiguous
  -c STR, --correction STR  also save corrected mutual information matrix data and
                           plot, one of apc, asc
  -m STR, --normalization STR also save normalized mutual information matrix data
                           and plot, one of sument, minent, maxent, mincon,
                           maxcon, joint

output options:
  -t, --heatmap            save heatmap files for all mutual information matrices
  -p STR, --prefix STR     output filename prefix, default is msa filename with
                           _coevol suffix
  -f STR, --number-format STR number output format (default: %12g)

figure options:
  -S, --save-plot          save coevolution plot
  -L FLOAT, --cmin FLOAT   apply lower limits for figure plot
  -U FLOAT, --cmax FLOAT   apply upper limits for figure plot
  -X STR, --xlabel STR     specify xlabel, by default will be applied on ylabel
  -T STR, --title STR      figure title
  -D INT, --dpi INT        figure resolution (dpi) (default: 300)
  -H FLOAT, --height FLOAT figure height (inch) (default: 6)
  -W FLOAT, --width FLOAT  figure width (inch) (default: 8)
  -F STR, --figure-format STR figure file format, one of svgz, rgba, png, pdf, eps,
                                 svg, ps, raw (default: pdf)

Examples

Running evol coevol --examples displays:
Sequence coevolution analysis involves several steps that including retrieving data and refining it for calculations. These steps are illustrated below for RnaseA protein family.

Search Pfam database:

$ evol search 2w5i

Download Pfam MSA file:

$ evol fetch RnaseA

Refine MSA file:

$ evol refine RnaseA_full.slx -l RNAS1_BOVIN --seqid 0.98 --rowocc 0.8

Checking occupancy:

$ evol occupancy RnaseA_full.slx -l RNAS1_BOVIN -o col -S

Conservation analysis:

$ evol conserv RnaseA_full_refined.slx

Coevolution analysis:

$ evol coevol RnaseA_full_refined.slx -S -c apc

Rank order analysis:

$ evol rankorder RnaseA_full_refined_mutinfo_corr_apc.txt -p 2w5i_1-121.pdb --seq-sep 3

### 2.2.2 evol conserv

#### Usage

Running `evol conserv -h` displays:

```

msa

positional arguments:

msa refined MSA file

optional arguments:

-h, --help show this help message and exit
--quiet suppress info messages to stderr
--examples show usage examples and exit
calculation options:

-n, --no-ambiguity treat amino acids characters B, Z, J, and X as non-ambiguous
-g, --gaps do not omit gap characters

output options:
```

2.2. Evol Applications
-p STR, --prefix STR  output filename prefix, default is msa filename with
    _conserv suffix
-f STR, --number-format STR
    number output format (default: %12g)
figure options:
-S, --save-plot  save conservation plot
-H FLOAT, --height FLOAT
    figure height (inch) (default: 6)
-W FLOAT, --width FLOAT
    figure width (inch) (default: 8)
-F STR, --figure-format STR
    figure file format, one of raw, png, ps, svgz, eps,
        pdf, rgba, svg (default: pdf)
-D INT, --dpi INT  figure resolution (dpi) (default: 300)

Examples

Running evol conserv --examples displays:

Sequence coevolution analysis involves several steps that including
retrieving data and refining it for calculations. These steps are
illustrated below for RnaseA protein family.

Search Pfam database:

$ evol search 2w5i

Download Pfam MSA file:

$ evol fetch RnaseA

Refine MSA file:

$ evol refine RnaseA_full.slx -l RNAS1_BOVIN --seqid 0.98 --rowocc 0.8

Checking occupancy:

$ evol occupancy RnaseA_full.slx -l RNAS1_BOVIN -o col -S

Conservation analysis:

$ evol conserv RnaseA_full_refined.slx

Coevolution analysis:

$ evol coevol RnaseA_full_refined.slx -S -c apc

Rank order analysis:

$ evol rankorder RnaseA_full_refined_mutinfo_corr_apc.txt -p 2w5i_1-121.pdb --seq-sep 3

2.2. Evol Applications 19
2.2.3 evol fetch

Usage

Running evol fetch -h displays:


positional arguments:
  acc            Pfam accession or ID

optional arguments:
  -h, --help     show this help message and exit
  --quiet        suppress info messages to stderr
  --examples     show usage examples and exit

download options:
  -a STR, --alignment STR
              alignment type, one of full, seed, ncbi, metagenomics
              (default: full)
  -f STR, --format STR
              Pfam supported MSA format, one of selex, fasta,
              stockholm (default: selex)
  -o STR, --order STR
              ordering of sequences, one of tree, alphabetical
              (default: tree)
  -i STR, --inserts STR
              letter case for inserts, one of upper, lower (default:
              upper)
  -g STR, --gaps STR
              gap character, one of dashes, dots, mixed (default:
              dashes)
  -t INT, --timeout INT
              timeout for blocking connection attempts (default: 60)

output options:
  -d PATH, --outdir PATH
              output directory (default: .)
  -p STR, --outname STR
              output filename, default is accession and alignment
              type
  -z, --compressed
              gzip downloaded MSA file

Examples

Running evol fetch --examples displays:

Sequence coevolution analysis involves several steps that including
retrieving data and refining it for calculations. These steps are
illustrated below for RnaseA protein family.

Search Pfam database:

$ evol search 2w5i

Download Pfam MSA file:

$ evol fetch RnaseA
Refine MSA file:

$ evol refine RnaseA_full.slx -l RNAS1_BOVIN --seqid 0.98 --rowocc 0.8

Checking occupancy:

$ evol occupancy RnaseA_full.slx -l RNAS1_BOVIN -o col -S

Conservation analysis:

$ evol conserv RnaseA_full_refined.slx

Coevolution analysis:

$ evol coevol RnaseA_full_refined.slx -S -c apc

Rank order analysis:

$ evol rankorder RnaseA_full_refined_mutinfo_corr_apc.txt -p 2w5i_1-121.pdb --seq-sep 3

### 2.2.4 evol filter

**Usage**

Running `evol filter -h` displays:

```
                  msa word [word ...]
```

**Positional arguments:**
- `msa` MSA filename to be filtered
- `word` word to be compared to sequence label

**Optional arguments:**
- `-h, --help` show this help message and exit
- `--quiet` suppress info messages to stderr
- `--examples` show usage examples and exit

**Filtering method (required):**
- `-s, --startswith` sequence label starts with given words
- `-e, --endswith` sequence label ends with given words
- `-c, --contains` sequence label contains with given words

**Filter option:**
- `-F, --full-label` compare full label with word(s)

**Output options:**
- `-o STR, --outname STR` output filename, default is msa filename with _refined suffix
- `-f STR, --format STR` output MSA file format, default is same as input
- `-z, --compressed` gzip refined MSA output

---

**2.2. Evol Applications**
Examples

Running `evol filter -examples` displays:

Sequence coevolution analysis involves several steps that including
retrieving data and refining it for calculations. These steps are
illustrated below for RnaseA protein family.

Search Pfam database:

```bash
$ evol search 2w5i
```

Download Pfam MSA file:

```bash
$ evol fetch RnaseA
```

Refine MSA file:

```bash
$ evol refine RnaseA_full.slx -l RNAS1_BOVIN --seqid 0.98 --rowocc 0.8
```

Checking occupancy:

```bash
$ evol occupancy RnaseA_full.slx -l RNAS1_BOVIN -o col -S
```

Conservation analysis:

```bash
$ evol conserv RnaseA_full_refined.slx
```

Coevolution analysis:

```bash
$ evol coevol RnaseA_full_refined.slx -S -c apc
```

Rank order analysis:

```bash
$ evol rankorder RnaseA_full_refined_mutinfo_corr_apc.txt -p 2w5i_1-121.pdb --seq-sep 3
```

2.2.5 evol merge

Usage

Running `evol merge -h` displays:

```
                  msa [msa ...]
```

Positional arguments:

- `msa`: MSA filenames to be merged

Optional arguments:

- `-h`, `--help`: show this help message and exit
- `--quiet`: suppress info messages to stderr
- `--examples`: show usage examples and exit

Output options:

- `-o STR`, `--outname STR`: output filename, default is first input filename with
  _merged suffix
Examples

Running **evol merge --examples** displays:

Sequence coevolution analysis involves several steps that include retrieving data and refining it for calculations. These steps are illustrated below for RnaseA protein family.

Search Pfam database:

```bash
$ evol search 2w5i
```

Download Pfam MSA file:

```bash
$ evol fetch RnaseA
```

Refine MSA file:

```bash
$ evol refine RnaseA_full.slx -l RNAS1_BOVIN --seqid 0.98 --rowocc 0.8
```

Checking occupancy:

```bash
$ evol occupancy RnaseA_full.slx -l RNAS1_BOVIN -o col -S
```

Conservation analysis:

```bash
$ evol conserv RnaseA_full_refined.slx
```

Coevolution analysis:

```bash
$ evol coevol RnaseA_full_refined.slx -S -c apc
```

Rank order analysis:

```bash
$ evol rankorder RnaseA_full_refined_mutinfo_corr_apc.txt -p 2w5i_1-121.pdb --seq-sep 3
```

### 2.2.6 evol occupancy

**Usage**

Running **evol occupancy -h** displays:

```
[-W FLOAT] [-F STR] [-H FLOAT]
msa
```

Positional arguments:

- **msa** MSA file

Optional arguments:

- **-h** Display help message
- **--quiet** Quiet mode
- **--examples** Display examples
- **-o STR** Output MSA file format, default is same as first input MSA
- **-f STR** Output MSA file format, default is same as first input MSA
- **-S** gzip merged MSA output
- **-W FLOAT** Weight
- **-F STR** Filter
- **-H FLOAT** Threshold
Examples

Running `evol occupancy --examples` displays:

Sequence coevolution analysis involves several steps that including retrieving data and refining it for calculations. These steps are illustrated below for RnaseA protein family.

Search Pfam database:

```
$ evol search 2w5i
```

Download Pfam MSA file:

```
$ evol fetch RnaseA
```

Refine MSA file:

```
$ evol refine RnaseA_full.slx -l RNAS1_BOVIN --seqid 0.98 --rowocc 0.8
```

Checking occupancy:

```
$ evol occupancy RnaseA_full.slx -l RNAS1_BOVIN -o col -S
```

Conservation analysis:

```
$ evol conserv RnaseA_full.refined.slx
```
Coevolution analysis:

\$ evol coevol RnaseA_full_refined.slix -S -c apc

Rank order analysis:

\$ evol rankorder RnaseA_full_refined_mutinfo_corr_apc.txt -p 2w5i_1-121.pdb --seq-sep 3

2.2.7 evol rankorder

Usage

Running evol rankorder -h displays:


positional arguments:
  mutinfo mutual information matrix

optional arguments:
  -h, --help show this help message and exit
  --quiet suppress info messages to stderr
  --examples show usage examples and exit

input options:
  -z, --zscore apply zscore for identifying top ranked coevolving pairs
  -d STR, --delimiter STR delimiter used in mutual information matrix file
  -p STR, --pdb STR PDB file that contains same number of residues as the mutual information matrix, output residue numbers will be based on PDB file
  -m STR, --msa STR MSA file used for building the mutual info matrix, output residue numbers will be based on the most complete sequence in MSA if a PDB file or sequence label is not specified
  -l STR, --label STR label in MSA file for output residue numbers

output options:
  -n INT, --num-pairs INT number of top ranking residue pairs to list (default: 100)
  -q INT, --seq-sep INT report coevolution for residue pairs that are sequentially separated by input value (default: 3)
  -t FLOAT, --min-dist FLOAT report coevolution for residue pairs whose CA atoms are spatially separated by at least the input value, used when a PDB file is given and --use-dist is true (default: 10.0)
  -u, --use-dist use structural separation to report coevolving pairs
  -o STR, --outname STR output filename, default is mutinfo_rankorder.txt

2.2. Evol Applications 25
**Examples**

Running **evol rankorder --examples** displays:

Sequence coevolution analysis involves several steps that including retrieving data and refining it for calculations. These steps are illustrated below for RnaseA protein family.

Search Pfam database:

```bash
$ evol search 2w5i
```

Download Pfam MSA file:

```bash
$ evol fetch RnaseA
```

Refine MSA file:

```bash
$ evol refine RnaseA_full.slx -l RNAS1_BOVIN --seqid 0.98 --rowocc 0.8
```

Checking occupancy:

```bash
$ evol occupancy RnaseA_full.slx -l RNAS1_BOVIN -o col -S
```

Conservation analysis:

```bash
$ evol conserv RnaseA_full_refined.slx
```

Coevolution analysis:

```bash
$ evol coevol RnaseA_full_refined.slx -S -c apc
```

Rank order analysis:

```bash
$ evol rankorder RnaseA_full_refined_mutinfo_corr_apc.txt -p 2w5i_1-121.pdb --seq-sep 3
```

**2.2.8 evol refine**

**Usage**

Running **evol refine -h** displays:

```
```

position arguments:

- msa: MSA filename to be refined

optional arguments:

- `-h`, `--help`: show this help message and exit
- `--quiet`: suppress info messages to stderr
- `--examples`: show usage examples and exit

refinement options:

- `-l STR`, `--label STR`: sequence label, UniProt ID code, or PDB and chain identifier

2.2. Evol Applications
-s FLOAT, --seqid FLOAT
  identity threshold for selecting unique sequences
-c FLOAT, --colocc FLOAT
  column (residue position) occupancy
-r FLOAT, --rowocc FLOAT
  row (sequence) occupancy
-k, --keep
  keep columns corresponding to residues not resolved in
  PDB structure, applies label argument is a PDB
  identifier

output options:
-o STR, --outname STR
  output filename, default is msa filename with _refined
  suffix
-f STR, --format STR
  output MSA file format, default is same as input
-z, --compressed
  gzip refined MSA output

Examples

Running evol refine --examples displays:

Sequence coevolution analysis involves several steps that including
retrieving data and refining it for calculations. These steps are
illustrated below for RnaseA protein family.

Search Pfam database:

$ evol search 2w5i

Download Pfam MSA file:

$ evol fetch RnaseA

Refine MSA file:

$ evol refine RnaseA_full.slx -l RNAS1_BOVIN --seqid 0.98 --rowocc 0.8

Checking occupancy:

$ evol occupancy RnaseA_full.slx -l RNAS1_BOVIN -o col -S

Conservation analysis:

$ evol conserv RnaseA_full_refined.slx

Coevolution analysis:

$ evol coevol RnaseA_full_refined.slx -S -c apc

Rank order analysis:

$ evol rankorder RnaseA_full_refined_mutinfo_corr_apc.txt -p 2w5i_1-121.pdb --seq-sep 3

2.2. Evol Applications
2.2.9 evol search

Usage

Running evol search -h displays:

[-t INT] [-o STR] [-d STR] 
query

positional arguments:
query   protein UniProt ID or sequence, a PDB identifier, or a 
        sequence file, where sequence have no gaps and 12 or 
        more characters

optional arguments:
-h, --help             show this help message and exit
--quiet               suppress info messages to stderr
--examples            show usage examples and exit

sequence search options:
-b, --searchBs         search Pfam-B families
-s, --skipAs           do not search Pfam-A families
-g, --ga               use gathering threshold
-e FLOAT, --evalue FLOAT 
        e-value cutoff, must be less than 10.0
-t INT, --timeout INT  
        timeout in seconds for blocking connection attempt
        (default: 60)

output options:
-o STR, --outname STR  
        name for output file, default is standard output
-d STR, --delimiter STR 
        delimiter for output data columns (default: )

Examples

Running evol search --examples displays:

Sequence coevolution analysis involves several steps that including 
retrieving data and refining it for calculations. These steps are 
illustrated below for RnaseA protein family.

Search Pfam database:

$ evol search 2w5i

Download Pfam MSA file:

$ evol fetch RnaseA

Refine MSA file:

$ evol refine RnaseA_full.slx -l RNAS1_BOVIN --seqid 0.98 --rowocc 0.8

Checking occupancy:
$ evol occupancy RnaseA_full.slx -l RNAS1_BOVIN -o col -S

Conservation analysis:
$ evol conserv RnaseA_full_refined.slx

Coevolution analysis:
$ evol coevol RnaseA_full_refined.slx -S -c apc

Rank order analysis:
$ evol rankorder RnaseA_full_refined_mutinfo_corr_apc.txt -p 2w5i_1-121.pdb --seq-sep 3

Running evol command will provide a description of applications:
$ evol

usage: evol [-h] [-c] [-v] [-e]
         {search,fetch,filter,refine,merge,occupancy,conserv,coevol,rankorder} ...

Evol: Sequence Evolution and Dynamics Analysis

optional arguments:
  -h, --help            show this help message and exit
  -c, --cite            print citation info and exit
  -v, --version         print ProDy version and exit
  -e, --examples        show usage examples and exit

subcommands:
  {search,fetch,filter,refine,merge,occupancy,conserv,coevol,rankorder}
  search     search Pfam with given query
  fetch      fetch MSA files from Pfam
  filter     filter an MSA using sequence labels
  refine     refine an MSA by removing gapped rows/columns
  merge      merge multiple MSAs based on common labels
  occupancy  calculate occupancy of rows and columns in MSA
  conserv    analyze conservation using Shannon entropy
  coevol     analyze co-evolution using mutual information
  rankorder  identify highly coevolving pairs of residues

See ‘evol <command> -h’ for more information on a specific command.

Detailed information on a specific application can be obtained by typing the command and application names as evol search -h.

Running prody search application as follows will search Pfam database for protein families that match the proteins in PDB structure 2w5i:
$ evol search 2w5i

On Linux, when installing ProDy from source, application scripts are placed into a default folder that is included in PATH environment variable, e.g. /usr/local/bin/.

On Windows, installer places the scripts into the Scripts folder under Python distribution folder, e.g.

---

2.2. Evol Applications 29

---

2http://matplotlib.sourceforge.net/faq/environment_variables_faq.html#envvar-PATH
You may need to add this path to PATH environment variable yourself.

---

3http://matplotlib.sourceforge.net/faq/environment_variables_faq.html#envvar-PATH
3.1 Atomic Data

This module defines classes for handling atomic data. Read this page using help(atomic).

3.1.1 Atomic classes

ProDy stores atomic data in instances of AtomGroup (page 37) class, which supports multiple coordinate sets, e.g. models from an NMR structure or snapshots from a molecular dynamics trajectory.

Instances of the class can be obtained by parsing a PDB file as follows:

In [1]: from prody import *

In [2]: ag = parsePDB('1aar')

In [3]: ag
Out[3]: <AtomGroup: laar (1218 atoms)>

In addition to AtomGroup (page 37) class, following classes that act as pointers provide convenient access subset of data:

- Selection (page 86) - Points to an arbitrary subset of atoms. See Atom Selections (page 78) and Operations on Selections\(^1\) for usage examples.
- Segment (page 74) - Points to atoms that have the same segment name.
- Chain (page 49) - Points to atoms in a segment that have the same chain identifier.
- Residue (page 69) - Points to atoms in a chain that have the same residue number and insertion code.
- AtomMap (page 45) - Points to arbitrary subsets of atoms while allowing for duplicates and missing atoms. Indices of atoms are stored in the order provided by the user.
- Atom (page 33) - Points to a single atom
- Bond (page 49) - Points to two connected atoms

\(^1\)http://prody.csb.pitt.edu/tutorials/prody_tutorial/selection.html#selection-operations
3.1.2 Atom data fields

*Atom Data Fields* (page 54) defines an interface for handling data parsed from molecular data files, in particular PDB files. Aforementioned classes offer `get` and `set` functions for manipulating this data. For example, the following prints residue names:

```python
In [4]: ag.getResnames()
Out[4]:
array(['MET', 'MET', 'MET', ..., 'HOH', 'HOH', 'HOH'],
      dtype='|S6')
```

3.1.3 Atom flags

*Atom Flags* (page 56) module defines a way to mark atoms with certain properties, such as atoms that are part of a protein. Following example checks whether all atoms of `ag` are protein atoms:

```python
In [5]: ag.isprotein
Out[5]: False
```

This indicates that there are some non-protein atoms, probably water atoms. We can easily make a count as follows:

```python
In [6]: ag.numAtoms('protein')
Out[6]: 1203
```

```python
In [7]: ag.numAtoms('hetero')
Out[7]: 15
```

```python
In [8]: ag.numAtoms('water')
Out[8]: 15
```

3.1.4 Atom selections

*Atom Selections* (page 78) offer a flexible and powerful way to access subsets of selections and is one of the most important features of ProDy. The details of the selection grammar is described in *Atom Selections* (page 78). Following examples show how to make quick selections using the overloaded . operator:

```python
In [9]: ag.chain_A  # selects chain A
Out[9]: <Selection: 'chain A' from 1aar (608 atoms)>
```

```python
In [10]: ag.calpha  # selects alpha carbons
Out[10]: <Selection: 'calpha' from 1aar (152 atoms)>
```

```python
In [11]: ag.resname_ALA  # selects alanine residues
Out[11]: <Selection: 'resname ALA' from 1aar (20 atoms)>
```

It is also possible to combine selections with and and or operators:

```python
In [12]: ag.chain_A_and_backbone
Out[12]: <Selection: 'chain A and backbone' from 1aar (304 atoms)>
```

```python
In [13]: ag.acidic_or_basic
Out[13]: <Selection: 'acidic or basic' from 1aar (422 atoms)>
```

Using dot operator will behave like the logical and operator:
In [14]: aq.chain_A.backbone
Out[14]: <Selection: '(backbone) and (chain A)' from laar (304 atoms)>

For this to work, the first word following the dot operator must be a flag label or a field name, e.g. resname, name, apolar, protein, etc. Underscores will be interpreted as white space, as obvious from the previous examples. The limitation of this is that parentheses, special characters cannot be used.

### 3.1.5 Functions

Following functions can be used for permanent data storage:

- `loadAtoms()` (page 66)
- `saveAtoms()` (page 66)

Following function can be used to identify fragments in a group (`AtomGroup` (page 37)) or subset (`Selection` (page 86)) of atoms:

- `findFragments()` (page 66)
- `iterFragments()` (page 66)

Following function can be used to get an `AtomMap` (page 45) that sorts atoms based on a given property:

- `sortAtoms()` (page 67)

Following function can be used check whether a word is reserved because it is used internally by `prody.atomic` (page 31) classes:

- `isReserved()` (page 66)
- `listReservedWords()` (page 66)

### 3.1.6 Atom

This module defines classes to handle individual atoms.

**class** `Atom` *(ag, index, acsi)*

A class for handling individual atoms in an `AtomGroup` (page 37).

- `copy()`
  - Return a copy of atoms (and atomic data) in an `AtomGroup` (page 37) instance.
- `getACSIndex()`
  - Return index of the coordinate set.
- `getACSLabel()`
  - Return active coordinate set label.
- `getAltloc()`
  - Return alternate location indicator of the atom. Alternate location indicator can be used in atom selections, e.g. `altloc A B`, `altloc _`.
- `getAnisou()`
  - Return anisotropic temperature factor of the atom.
- `getAnistd()`
  - Return standard deviations for anisotropic temperature factor of the atom.
- `getAtomGroup()`
  - Return associated atom group.
getBeta()  
Return β-value (temperature factor) of the atom. β-value can be used in atom selections, e.g. ‘beta 555.55’, ‘beta 0 to 500’, ‘beta 0:500’, ‘beta < 500’.

getCSLabels()  
Return coordinate set labels.

getcharge()  
Return partial charge of the atom. Partial charge can be used in atom selections, e.g. ‘charge 1’, ‘abs(charge) == 1’, ‘charge < 0’.

getchid()  
Return chain identifier of the atom. Chain identifier can be used in atom selections, e.g. ‘chain A’, ‘chid A B C’, ‘chain _’. Note that chid is a synonym for chain.

getchindex()  
Return chain index of the atom. Chain indices are assigned to subsets of atoms with distinct pairs of chain identifier and segment name. Chain indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Chain index can be used in atom selections, e.g. ‘chindex 0’.

getcode()  
Return a copy of coordinates of the atom from the active coordinate set.

getcode sets(indices=None)  
Return a copy of coordinate set(s) at given indices.

getdata (label)  
Return a copy of data associated with label, if it is present.

getdatalabels (which=None)  
Return data labels. For which='user', return only labels of user provided data.

getdatatypes (label)  
Return type of the data (i.e. data.dtype) associated with label, or None label is not used.

gettelement()  
Return element symbol of the atom. Element symbol can be used in atom selections, e.g. ‘element C O N’.

getflag (label)  
Return atom flag.

getflaglabels (which=None)  
Return flag labels. For which='user', return labels of user or parser (e.g. hetatm) provided flags, for which='all' return all possible Atom Flags (page 56) labels in addition to those present in the instance.

getfragindex()  
Return fragment index of the atom. Fragment indices are assigned to connected subsets of atoms. Bonds needs to be set using AtomGroup.setBonds() (page 42) method. Fragment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Fragment index can be used in atom selections, e.g. ‘fragindex 0’, ‘fragment 1’. Note that fragment is a synonym for fragindex.

getIcode()  
Return insertion code of the atom. Insertion code can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

getIndex()  
Return index of the atom.
getIndices()
Return index of the atom in an numpy.ndarray².

def getMass()
Return mass of the atom. Mass can be used in atom selections, e.g. '12 <= mass <= 13.5'.

def getName()
Return name of the atom. Name can be used in atom selections, e.g. 'name CA CB'.

def getOccupancy()
Return occupancy value of the atom. Occupancy value can be used in atom selections, e.g. 'occupancy 1','occupancy > 0'.

def getRadius()
Return radius of the atom. Radius can be used in atom selections, e.g. 'radii < 1.5','radii ** 2 < 2.3'.

def getResindex()
Return residue index of the atom. Residue indices are assigned to subsets of atoms with distinct sequences of residue number, insertion code, chain identifier, and segment name. Residue indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Residue index can be used in atom selections, e.g. 'resindex 0'.

def getResname()
Return residue name of the atom. Residue name can be used in atom selections, e.g. 'resname ALA GLY'.

def getResnum()
Return residue number of the atom. Residue number can be used in atom selections, e.g. 'resnum 1 2 3', 'resnum 120A 120B', 'resnum 10 to 20', 'resnum 10:20:2', 'resnum < 10'. Note that resid is a synonym for resnum.

def getSecstr()
Return secondary structure assignment of the atom. Secondary structure assignment can be used in atom selections, e.g. 'secondary H E', 'secstr H E'. Note that secstr is a synonym for secondary.

def getSegindex()
Return segment index of the atom. Segment indices are assigned to subsets of atoms with distinct segment names. Segment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Segment index can be used in atom selections, e.g. 'segindex 0'.

def getSegname()
Return segment name of the atom. Segment name can be used in atom selections, e.g. 'segment PROT', 'segname PROT'. Note that segname is a synonym for segment.

def getSelstr()
Return selection string that will select this atom.

def getSerial()
Return serial number (from file) of the atom. Serial number can be used in atom selections, e.g. 'serial 1 2 3','serial 1 to 10','serial 1:10:2','serial < 10'.

def getType()
Return type of the atom. Type can be used in atom selections, e.g. 'type CT1 CT2 CT3'.

²http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
**isDataLabel** *(label)*
Return `True` if data associated with *label* is present.

**isFlagLabel** *(label)*
Return `True` if flags associated with *label* are present.

**iterAtoms** *
Yield atoms.

**iterBonded** *
Yield bonded atoms. Use `setBonds()` for setting bonds.

**iterBonds** *
Yield bonds formed by the atom. Use `setBonds()` for setting bonds.

**iterCoordsets** *
Yield copies of coordinate sets.

**numAtoms** *(flag=None)*
Return number of atoms, or number of atoms with given *flag*.

**numBonds** *
Return number of bonds formed by this atom. Bonds must be set first using `AtomGroup.setBonds()` (page 42).

**numCoordsets** *
Return number of coordinate sets.

**select** *(selstr, **kwargs)*
Return atoms matching *selstr* criteria. See `select` (page 78) module documentation for details and usage examples.

**setACSIndex** *(index)*
Set coordinates at *index* active.

**setAltloc** *(data)*
Set alternate location indicator of the atom. Alternate location indicator can be used in atom selections, e.g. `'altloc A B', 'altloc _'`

**setAnisou** *(data)*
Set anisotropic temperature factor of the atom.

**setAnistd** *(data)*
Set standard deviations for anisotropic temperature factor of the atom.

**setBeta** *(data)*
Set $\beta$-value (temperature factor) of the atom. $\beta$-value can be used in atom selections, e.g. `'beta 555.55', 'beta 0 to 500', 'beta 0:500', 'beta < 500'`.

**setCharge** *(data)*
Set partial charge of the atom. Partial charge can be used in atom selections, e.g. `'charge 1', 'abs(charge) == 1', 'charge < 0'`.

**setChid** *(data)*
Set chain identifier of the atom. Chain identifier can be used in atom selections, e.g. `'chain A', 'chid A B C', 'chain _'`. Note that *chid* is a synonym for *chain*.

**setCoords** *(coords)*
Set coordinates of the atom in the active coordinate set.

**setData** *(label, data)*
Update data associated with *label*.
Raises AttributeError when label is not in use or read-only

**setElement**(data)
Set element symbol of the atom. Element symbol can be used in atom selections, e.g. ‘element C O N’.

**setFlag**(label, value)
Update flag associated with label.

Raises AttributeError when label is not in use or read-only

**setIcode**(data)
Set insertion code of the atom. Insertion code can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

**setMass**(data)
Set mass of the atom. Mass can be used in atom selections, e.g. ‘12 <= mass <= 13.5’.

**setName**(data)
Set name of the atom. Name can be used in atom selections, e.g. ‘name CA CB’.

**setOccupancy**(data)
Set occupancy value of the atom. Occupancy value can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.

**setRadius**(data)
Set radius of the atom. Radius can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

**setResname**(data)
Set residue name of the atom. Residue name can be used in atom selections, e.g. ‘resname ALA GLY’.

**setResnum**(data)
Set residue number of the atom. Residue number can be used in atom selections, e.g. ‘resnum 1 2 3’, ‘resnum 120A 120B’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’. Note that resid is a synonym for resnum.

**setSecstr**(data)
Set secondary structure assignment of the atom. Secondary structure assignment can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that secstr is a synonym for secondary.

**setSegname**(data)
Set segment name of the atom. Segment name can be used in atom selections, e.g. ‘segment PROT’, ‘segname PROT’. Note that segname is a synonym for segment.

**setSerial**(data)
Set serial number (from file) of the atom. Serial number can be used in atom selections, e.g. ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’.

**setType**(data)
Set type of the atom. Type can be used in atom selections, e.g. ‘type CT1 CT2 CT3’.

### 3.1.7 Atom Group

This module defines AtomGroup (page 37) class that stores atomic data and multiple coordinate sets in numpy.ndarray instances.

---

class **AtomGroup** *(title='Unnamed')*

A class for storing and accessing atomic data. The number of atoms of the atom group is inferred at the first set method call from the size of the data array.

**Atomic data**

All atomic data is stored in `numpy.ndarray` instances.

**Get and set methods**

*get* methods, e.g. `getResnames()` (page 41), return copies of the data arrays.

*set* methods, e.g. `setResnums()` (page 43), accept data in `list()` or `ndarray` instances. The length of the list or array must match the number of atoms in the atom group. These methods set attributes of all atoms at once.

**Coordinate sets**

Atom groups with multiple coordinate sets may have one of these sets as the *active coordinate set*. The active coordinate set may be changed using `setACSIndex()` (page 42) method. `getCoords()` (page 40) returns coordinates from the *active set*.

**Atom subsets**

To access and modify data associated with a subset of atoms in an atom group, `Selection` (page 86) instances may be used. A `Selection` (page 86) has initially the same coordinate set as the *active coordinate set*, but it may be changed using `Selection.setACSIndex()` (page 89) method.

**Customizations**

Following built-in functions are customized for this class:

- `len()` returns the number of atoms, i.e. `numAtoms()` (page 42)
- `iter()` yields `Atom` (page 33) instances

**Indexing AtomGroup** (page 37) instances by:

- `int (int())`, e.g. 10, returns an `Atom` (page 33)
- `slice (slice())`, e.g. 10:20:2, returns a `Selection` (page 86)
- `segment name (str())`, e.g. 'PROT', returns a a `Segment` (page 74)
- `chain identifier (str())`, e.g. 'A', returns a a `Chain` (page 49)
- `[segment name,] chain identifier, residue number[, insertion code] (tuple())`, e.g. 'A', 10 or 'A', 10, 'B' or 'PROT', 'A', 10, 'B', returns a `Residue` (page 69)

**Addition**

Addition of two `AtomGroup` (page 37) instances, let’s say A and B, results in a new `AtomGroup` (page 37) instance, say C. C stores an independent copy of the data of A and B. If A or B is missing a certain data type, zero values will be used for that part in C. If A and B has same number of coordinate

---

4 http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
5 http://docs.python.org/library/functions.html#list
6 http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
7 http://docs.python.org/library/functions.html#len
8 http://docs.python.org/library/functions.html#iter
9 http://docs.python.org/library/functions.html#int
10 http://docs.python.org/library/functions.html#slice
11 http://docs.python.org/library/functions.html#str
12 http://docs.python.org/library/functions.html#str
13 http://docs.python.org/library/functions.html#tuple
sets, C will have a copy of all coordinate sets, otherwise C will have a single coordinate set, which is a copy of active coordinate sets of A and B.

**addCoordset** *(coords, label=None)*

Add a coordinate set. `coords` argument may be an object with `getCoordsets()` (page 40) method.

**copy()**

Return a copy of atoms (and atomic data) in an `AtomGroup` (page 37) instance.

**delCoordset** *(index)*

Delete a coordinate set from the atom group.

**delData** *(label)*

Return data associated with `label` and remove from the instance. If data associated with `label` is not found, return `None`.

**delFlags** *(label)*

Return flags associated with `label` and remove from the instance. If flags associated with `label` is not found, return `None`.

**getACSIndex()**

Return index of the coordinate set.

**getACSLabel()**

Return active coordinate set label.

**getAltlocs()**

Return a copy of alternate location indicators. Alternate location indicators can be used in atom selections, e.g. ‘altloc A B’, ‘altloc _’.

**getAnisous()**

Return a copy of anisotropic temperature factors.

**getAnistds()**

Return a copy of standard deviations for anisotropic temperature factors.

**getBetas()**

Return a copy of $\beta$-values (or temperature factors). $\beta$-values can be used in atom selections, e.g. ‘$\beta$ 555.55’, ‘$\beta$ 0 to 500’, ‘$\beta$ 0:500’, ‘$\beta$ < 500’.

**getBySerial** *(serial, stop=None, step=None)*

Get an atom(s) by serial number (range). `serial` must be zero or a positive integer. `stop` may be `None`, or an integer greater than `serial`. `getBySerial(i, j)` will return atoms whose serial numbers are $i+1, i+2, ..., j-1$. Atom whose serial number is `stop` will be excluded as it would be in indexing a Python list. `step` (default is 1) specifies increment. If atoms with matching serial numbers are not found, `None` will be returned.

**getCSLabels()**

Return coordinate set labels.

**getCharges()**

Return a copy of partial charges. Partial charges can be used in atom selections, e.g. ‘charge 1’, ‘abs(charge) == 1’, ‘charge < 0’.

**getChids()**

Return a copy of chain identifiers. Chain identifiers can be used in atom selections, e.g. ‘chain A’, ‘chid A B C’, ‘chain _’. Note that `chid` is a synonym for `chain`.

**getChindices()**

Return a copy of chain indices. Chain indices are assigned to subsets of atoms with distinct pairs of chain identifier and segment name. Chain indices start from zero, are incremented by one,

### 3.1. Atomic Data
and are assigned in the order of appearance in AtomGroup (page 37) instance. Chain indices can be used in atom selections, e.g. ‘chindex 0’.

**getCoords()**
Return a copy of coordinates from active coordinate set.

**getCoordsets(indices=None)**
Return a copy of coordinate set(s) at given indices. indices may be an integer, a list of integers, or None meaning all coordinate sets.

**getData(label)**
Return a copy of the data array associated with label, or None if such data is not present.

**getDataLabels(which=None)**
Return data labels. For which=‘user’, return only labels of user provided data.

**getDataType(label)**
Return type of the data (i.e. data.dtype) associated with label, or None label is not used.

**getElements()**
Return a copy of element symbols. Element symbols can be used in atom selections, e.g. ‘element C O N’.

**getFlagLabels(which=None)**
Return flag labels. For which=‘user’, return labels of user or parser (e.g. hetatm) provided flags, for which=‘all’ return all possible Atom Flags (page 56) labels in addition to those present in the instance.

**getFlags(label)**
Return a copy of atom flags for given label, or None when flags for label is not set.

**getFragindices()**
Return a copy of fragment indices. Fragment indices are assigned to connected subsets of atoms. Bonds needs to be set using AtomGroup.setBonds() (page 42) method. Fragment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Fragment indices can be used in atom selections, e.g. ‘fragindex 0’, ‘fragment 1’. Note that fragment is a synonym for fragindex.

**getHierView(**kwargs**)**
Return a hierarchical view of the atom group.

**getIcodes()**
Return a copy of insertion codes. Insertion codes can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

**getMasses()**
Return a copy of masses. Masses can be used in atom selections, e.g. ‘12 <= mass <= 13.5’.

**getNames()**
Return a copy of names. Names can be used in atom selections, e.g. ‘name CA CB’.

**getOccupancies()**
Return a copy of occupancy values. Occupancy values can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.

**getRadii()**
Return a copy of radii. Radii can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

**getResindices()**
Return a copy of residue indices. Residue indices are assigned to subsets of atoms with distinct
sequences of residue number, insertion code, chain identifier, and segment name. Residue indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Residue indices can be used in atom selections, e.g. ‘resindex 0’.

getResnames()
Return a copy of residue names. Residue names can be used in atom selections, e.g. ‘resname ALA GLY’.

getResnums()
Return a copy of residue numbers. Residue numbers can be used in atom selections, e.g. ‘resnum 1 2 3’, ‘resnum 120A 120B’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’. Note that resid is a synonym for resnum.

getSecstrs()
Return a copy of secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that secstr is a synonym for secondary.

getSegindices()
Return a copy of segment indices. Segment indices are assigned to subsets of atoms with distinct segment names. Segment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Segment indices can be used in atom selections, e.g. ‘segindex 0’.

getSegnames()
Return a copy of segment names. Segment names can be used in atom selections, e.g. ‘segment PROT’, ‘segname PROT’. Note that segname is a synonym for segment.

getSerials()
Return a copy of serial numbers (from file). Serial numbers can be used in atom selections, e.g. ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’.

getTitle()
Return title of the instance.

getTypes()
Return a copy of types. Types can be used in atom selections, e.g. ‘type CT1 CT2 CT3’.

isDataLabel(label)
Return True if data associated with label is present.

isFlagLabel(label)
Return True if flags associated with label are present.

iterAtoms()
Yield atom instances.

iterBonds()
Yield bonds. Use setBonds() (page 42) for setting bonds.

iterChains()
Iterate over chains.

iterCoordsets()
Iterate over coordinate sets by returning a copy of each coordinate set.

iterFragments()
Yield connected atom subsets as Selection (page 86) instances.

iterResidues()
Iterate over residues.
**iterSegments**
Iterate over chains.

**numAtoms** *(flag=None)*  
Return number of atoms, or number of atoms with given flag.

**numBonds**  
Return number of bonds. Use `setBonds()` (page 42) for setting bonds.

**numBytes** *(all=False)*  
Return number of bytes used by atomic data arrays, such as coordinate, flag, and attribute arrays. If `all` is `True`, internal arrays for indexing hierarchical views, bonds, and fragments will also be included. Note that memory usage of Python objects is not taken into account and that this may change in the future.

**numChains**  
Return number of chains.

**numCoordsets**  
Return number of coordinate sets.

**numFragments**  
Return number of connected atom subsets.

**numResidues**  
Return number of residues.

**numSegments**  
Return number of segments.

**select** *(selstr, **kwargs)*  
Return atoms matching `selstr` criteria. See `select` (page 78) module documentation for details and usage examples.

**setACSIndex** *(index)*  
Set the coordinate set at `index` active.

**setACSLable** *(label)*  
Set active coordinate set label.

**setAltlocs** *(data)*  
Set alternate location indicators. Alternate location indicators can be used in atom selections, e.g. `altloc A B`, `altloc _`.

**setAnisous** *(data)*  
Set anisotropic temperature factors.

**setAnistds** *(data)*  
Set standard deviations for anisotropic temperature factors.

**setBetas** *(data)*  
Set $\beta$-values (or temperature factors). $\beta$-values can be used in atom selections, e.g. `$\beta$ 555.55$, `$\beta$ 0 to 500$, `$\beta$ 0:500$, `$\beta$ 0 to 500$.

**setBonds** *(bonds)*  
Set covalent bonds between atoms. `bonds` must be a list or an array of pairs of indices. All bonds must be set at once. Bonding information can be used to make atom selections, e.g. "$\text{bonded to index 1}$". See `select` (page 78) module documentation for details. Also, a data array with number of bonds will be generated and stored with label `numbonds`. This can be used in atom selections, e.g. `numbonds 0` can be used to select ions in a system.
**setCSLabels** *(labels)*  
Set coordinate set labels. *labels* must be a list of strings.

**setCharges** *(data)*  
Set partial charges. Partial charges can be used in atom selections, e.g. ‘charge 1’, ‘abs(charge) == 1’, ‘charge < 0’.

**setChids** *(data)*  
Set chain identifiers. Chain identifiers can be used in atom selections, e.g. ‘chain A’, ‘chid A B C’, ‘chain _’. Note that *chid* is a synonym for *chain*.

**setCoords** *(coords, label=“”)*  
Set coordinates of atoms. *coords* may be any array like object or an object instance with `getCoords()` (page 40) method. If the shape of coordinate array is *(n_csets > 1, n_atoms, 3)*, it will replace all coordinate sets and the active coordinate set index will reset to zero. This situation can be avoided using `addCoordset()` (page 39). If shape of *coords* is *(n_atoms, 3)* or *(1, n_atoms, 3)*, it will replace the active coordinate set. *label* argument may be used to label coordinate set(s). *label* may be a string or a list of strings length equal to the number of coordinate sets.

**setData** *(label, data)*  
Store atomic *data* under *label*, which must:

- start with a letter
- contain only alphanumeric characters and underscore
- not be a reserved word (see `listReservedWords()` (page 66))

*data* must be a `list()`\(^{14}\) or a `ndarray`\(^ {15}\) and its length must be equal to the number of atoms. If the dimension of the *data* array is 1, i.e. *data.ndim==1*, *label* may be used to make atom selections, e.g. "label 1 to 10" or "label C1 C2". Note that, if data with *label* is present, it will be overwritten.

**setElements** *(data)*  
Set element symbols. Element symbols can be used in atom selections, e.g. ‘element C O N’.

**setFlags** *(label, flags)*  
Set atom flags for *label*.

**setIcodes** *(data)*  
Set insertion codes. Insertion codes can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

**setMasses** *(data)*  
Set masses. Masses can be used in atom selections, e.g. “12 <= mass <= 13.5’.

**setNames** *(data)*  
Set names. Names can be used in atom selections, e.g. ‘name CA CB’.

**setOccupancies** *(data)*  
Set occupancy values. Occupancy values can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.

**setRadii** *(data)*  
Set radii. Radii can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

**setResnames** *(data)*  
Set residue names. Residue names can be used in atom selections, e.g. ‘resname ALA GLY’.

\(^{14}\)http://docs.python.org/library/functions.html#list  
\(^{15}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

---

**3.1. Atomic Data**  
43
**setResnums** *(data)*
Set residue numbers. Residue numbers can be used in atom selections, e.g. ‘resnum 1 2 3’, ‘resnum 120A 120B’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’. Note that resid is a synonym for resnum.

**setSecstrs** *(data)*
Set secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that secstr is a synonym for secondary.

**setSegnames** *(data)*
Set segment names. Segment names can be used in atom selections, e.g. ‘segment PROT’, ‘segname PROT’. Note that segname is a synonym for segment.

**setSerials** *(data)*
Set serial numbers (from file). Serial numbers can be used in atom selections, e.g. ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’.

**setTitle** *(title)*
Set title of the instance.

**setTypes** *(data)*
Set types. Types can be used in atom selections, e.g. ‘type CT1 CT2 CT3’.

### 3.1.8 Atomic Base

This module defines base class **Atomic** (page 44) that all other **atomic** (page 31) classes are derived from.

**class Atomic**
Base class for all atomic classes that can be used for type checking.

**copy** ()
Return a copy of atoms (and atomic data) in an **AtomGroup** (page 37) instance.

**select** *(selstr, **kwargs)*
Return atoms matching selstr criteria. See **select** (page 78) module documentation for details and usage examples.

### 3.1.9 Atom Map

This module defines **AtomMap** (page 45) class that allows for pointing atoms in arbitrary order.

**How AtomMap’s work**

**AtomMap** (page 45) class adds great flexibility to manipulating atomic data.

First let’s see how an instance of **Selection** (page 86) (**Chain** (page 49), or **Residue** (page 69)) works. Below table shows indices for a selection of atoms in an **AtomGroup** (page 37) and values returned when **getNames** () (page 87), **getResnames** () (page 88) and **getResnums** () (page 88) methods are called.
Table 3.1: Atom Subset

<table>
<thead>
<tr>
<th>Indices</th>
<th>Names</th>
<th>Resnames</th>
<th>Resnums</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N</td>
<td>PHE</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>CA</td>
<td>PHE</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>PHE</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>O</td>
<td>PHE</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>CB</td>
<td>PHE</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>CG</td>
<td>PHE</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>CD1</td>
<td>PHE</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>CD2</td>
<td>PHE</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>CE1</td>
<td>PHE</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>CE2</td>
<td>PHE</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>CZ</td>
<td>PHE</td>
<td>1</td>
</tr>
</tbody>
</table>

Selection (page 86) instances keep indices ordered and do not allow duplicate values, hence their use is limited. In an AtomMap (page 45), indices do not need to be sorted, duplicate indices may exist, even “DUMMY” atoms are allowed.

Let’s say we instantiate the following AtomMap:

```python
amap = AtomMap(atomgroup, indices=[0, 1, 3, 8, 8, 9, 10],
               mapping=[5, 6, 7, 0, 1, 2, 3])
```

The size of the AtomMap (page 45) based on this mapping is 8, since the larger mapping is 7.

Calling the same functions for this AtomMap instance would result in the following:

Table 3.2: Atom Map

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Indices</th>
<th>Names</th>
<th>Resnames</th>
<th>Resnums</th>
<th>MappedFlags</th>
<th>DummyFlags</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>CE1</td>
<td>PHE</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>CE1</td>
<td>PHE</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>CE2</td>
<td>PHE</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>CZ</td>
<td>PHE</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>N</td>
<td>PHE</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>N</td>
<td>PHE</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>CA</td>
<td>PHE</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>O</td>
<td>PHE</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

For unmapped atoms, numeric attributes are set to 0, others to empty string, i.e. "".

See Also:
AtomMap (page 45) are used by proteins (page 146) module functions that match or map protein chains. Heterogeneous X-ray Structures\(^{16}\) and Multimeric Structures\(^{17}\) examples that make use of these functions and AtomMap (page 45) class.

class AtomMap (ag, indices, acsi=None, **kwargs)
A class for mapping atomic data.

 Instantiate an atom map.

 Parameters

• `ag` – AtomGroup instance from which atoms are mapped
• `indices` – indices of mapped atoms

\(^{16}\)http://prody.csb.pitt.edu/tutorials/ensemble_analysis/xray.html#pca-xray
\(^{17}\)http://prody.csb.pitt.edu/tutorials/ensemble_analysis/dimer.html#pca-dimer
• **acsi** – active coordinate set index, defaults is that of `ag`
• **mapping** – mapping of atom *indices*
• **dummies** – dummy atom indices
• **title** – title of the instance, default is ‘Unknown’

`mapping` and `dummies` arrays must be provided together. Length of `mapping` must be equal to length of *indices*. Elements of `mapping` must be an ordered in ascending order. When dummy atoms are present, number of atoms is the sum of lengths of `mapping` and `dummies`.

Following built-in functions are customized for this class:

- **len()**\(^{18}\) returns the number of atoms in the instance.
- **iter()**\(^{19}\) yields `Atom` (page 33) instances.
- Indexing returns an `Atom` (page 33) or an `AtomMap` (page 45) instance depending on the type and value of the index.

**copy()**
Return a copy of atoms (and atomic data) in an `AtomGroup` (page 37) instance.

**getACSIndex()**
Return index of the coordinate set.

**getACSLabel()**
Return active coordinate set label.

**getAltlocs()**
Return a copy of alternate location indicators. Entries for dummy atoms will be ‘’.

**getAnisous()**
Return a copy of anisotropic temperature factors. Entries for dummy atoms will be 0.0.

**getAnistds()**
Return a copy of standard deviations for anisotropic temperature factors. Entries for dummy atoms will be 0.0.

**getAtomGroup()**
Return associated atom group.

**getBetas()**
Return a copy of β-values (or temperature factors). Entries for dummy atoms will be 0.0.

**getCSLabels()**
Return coordinate set labels.

**getCharges()**
Return a copy of partial charges. Entries for dummy atoms will be 0.0.

**getChids()**
Return a copy of chain identifiers. Entries for dummy atoms will be ‘’.

**getChindices()**
Return a copy of chain indices. Chain indices are assigned to subsets of atoms with distinct pairs of chain identifier and segment name. Chain indices start from zero, are incremented by one, and are assigned in the order of appearance in `AtomGroup` (page 37) instance. Entries for dummy atoms will be 0.

\(^{18}\)http://docs.python.org/library/functions.html#len
\(^{19}\)http://docs.python.org/library/functions.html#iter
getCoords()
Return a copy of coordinates from the active coordinate set.

getCoordsets (indices=None)
Return coordinate set(s) at given indices, which may be an integer or a list/array of integers.

getdata (label)
Return a copy of data associated with label, if it is present.

getDataLabels (which=None)
Return data labels. For which=’user’, return only labels of user provided data.

getDataType (label)
Return type of the data (i.e. data.dtype) associated with label, or None label is not used.

getElements()
Return a copy of element symbols. Entries for dummy atoms will be ".

getFlagLabels (which=None)
Return flag labels. For which=’user’, return labels of user or parser (e.g. hetatm) provided flags, for which=’all’ return all possible Atom Flags (page 56) labels in addition to those present in the instance.

getFlags (label)
Return a copy of atom flags for given label, or None when flags for label is not set.

getFragindices()
Return a copy of fragment indices. Fragment indices are assigned to connected subsets of atoms.
Bonds needs to be set using AtomGroup.setBonds() (page 42) method. Fragment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Entries for dummy atoms will be 0.

getIcodes()
Return a copy of insertion codes. Entries for dummy atoms will be ".

getIndices()
Return a copy of indices of atoms, with maximum integer value dummies.

getMapping()
Return a copy of mapping of indices.

getMasses()
Return a copy of masses. Entries for dummy atoms will be 0.0.

getNames()
Return a copy of names. Entries for dummy atoms will be ".

getOccuPancies()
Return a copy of occupancy values. Entries for dummy atoms will be 0.0.

getRadii()
Return a copy of radii. Entries for dummy atoms will be 0.0.

getResindices()
Return a copy of residue indices. Residue indices are assigned to subsets of atoms with distinct sequences of residue number, insertion code, chain identifier, and segment name. Residue indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Entries for dummy atoms will be 0.

getResnames()
Return a copy of residue names. Entries for dummy atoms will be ".

3.1. Atomic Data
getResnums()  
Return a copy of residue numbers. Entries for dummy atoms will be 0.

getSecstrs()  
Return a copy of secondary structure assignments. Entries for dummy atoms will be "."

getSegindices()  
Return a copy of segment indices. Segment indices are assigned to subsets of atoms with distinct segment names. Segment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Entries for dummy atoms will be 0.

getSegnames()  
Return a copy of segment names. Entries for dummy atoms will be "."

gSelstr()  
Return a copy of segment names. Entries for dummy atoms will be "."

gSelstrs()  
Return selection string that selects mapped atoms.

gGetserials()  
Return a copy of serial numbers (from file). Entries for dummy atoms will be 0.

ggetTitle()  
Return title of the instance.

gGetTypes()  
Return a copy of types. Entries for dummy atoms will be ".

isDataLabel(label)  
Return True if data associated with label is present.

isFlagLabel(label)  
Return True if flags associated with label are present.

iterAtoms()  
Yield atoms, and None for dummies.

iterCoordsets()  
Yield copies of coordinate sets.

numAtoms(flag=None)  
Return number of atoms.

numCoordsets()  
Return number of coordinate sets.

numDummies()  
Return number of dummy atoms.

numMapped()  
Return number of mapped atoms.

select (selstr, **kwargs)  
Return atoms matching selstr criteria. See select (page 78) module documentation for details and usage examples.

setACSIndex(index)  
Set coordinates at index active.

setCoords(coords)  
Set coordinates of atoms in the active coordinate set.

setTitle(title)  
Set title of the instance.
3.1.10 Bond

This module defines Bond (page 49) for dealing with bond information provided by using AtomGroup.setBonds() (page 42) method.

class Bond (ag, indices, acsi=None)
A pointer class for bonded atoms. Following built-in functions are customized for this class:

• len() \(^{20}\) returns bond length, i.e. getLength() (page 49)
• iter() \(^{21}\) yields Atom (page 33) instances

getACSIndex ()
Return index of the coordinate set.

getAtomGroup ()
Return atom group.

getAtoms ()
Return bonded atoms.

getIndices ()
Return indices of bonded atoms.

getLength ()
Return bond length.

getVector ()
Return bond vector that originates from the first atom.

setACSIndex (index)
Set the coordinate set at index active.

3.1.11 Chain

This module defines classes for handling polypeptide/nucleic acid chains.

class Chain (ag, indices, hv, acsi=None, **kwargs)
Instances of this class point to atoms with same chain identifiers and are generated by HierView (page 67) class. Following built-in functions are customized for this class:

• len() \(^{22}\) returns the number of residues in the chain
• iter() \(^{23}\) yields Residue (page 69) instances

Indexing Chain (page 49) instances by:

• residue number [, insertion code]() \(^{24}\), e.g. 10 or 10, "B", returns a Residue (page 69)
• slice (slice() \(^{25}\)), e.g. 10:20, returns a list of Residue (page 69) instances

copy ()
Return a copy of atoms (and atomic data) in an AtomGroup (page 37) instance.

getACSIndex ()
Return index of the coordinate set.

---

\(^{20}\) http://docs.python.org/library/functions.html#len
\(^{21}\) http://docs.python.org/library/functions.html#iter
\(^{22}\) http://docs.python.org/library/functions.html#len
\(^{23}\) http://docs.python.org/library/functions.html#iter
\(^{24}\) http://docs.python.org/library/functions.html#tuple
\(^{25}\) http://docs.python.org/library/functions.html#slice
getACSLabel()  
Return active coordinate set label.

getAltlocs()  
Return a copy of alternate location indicators. Alternate location indicators can be used in atom selections, e.g. 'altloc A B','altloc _'.

getAnisous()  
Return a copy of anisotropic temperature factors.

getAnistds()  
Return a copy of standard deviations for anisotropic temperature factors.

getAtomGroup()  
Return associated atom group.

getBetas()  
Return a copy of \(\beta\)-values (or temperature factors). \(\beta\)-values can be used in atom selections, e.g. 'beta 555.55','beta 0 to 500','beta 0:500','beta < 500'.

getCSLabels()  
Return coordinate set labels.

getCharges()  
Return a copy of partial charges. Partial charges can be used in atom selections, e.g. 'charge 1','abs(charge) == 1','charge < 0'.

getChid()  
Return chain identifier.

getchids()  
Return a copy of chain identifiers. Chain identifiers can be used in atom selections, e.g. 'chain A','chid A B C','chain _'. Note that chid is a synonym for chain.

getchindex()  
Return chain index.

getchindices()  
Return a copy of chain indices. Chain indices are assigned to subsets of atoms with distinct pairs of chain identifier and segment name. Chain indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Chain indices can be used in atom selections, e.g. 'chindex 0'.

getCoords()  
Return a copy of coordinates from the active coordinate set.

getCoordsets(indices=None)  
Return coordinate set(s) at given indices, which may be an integer or a list/array of integers.

gedata (label)  
Return a copy of data associated with label, if it is present.

gedataLabels(which=None)  
Return data labels. For which='user', return only labels of user provided data.

gedataType(label)  
Return type of the data (i.e. data.dtype) associated with label, or None label is not used.

gedataElements()  
Return a copy of element symbols. Element symbols can be used in atom selections, e.g. 'element C O N'.
**getFlagLabels** *(which=None)*

Return flag labels. For *which='user'* , return labels of user or parser (e.g. *hetatm*) provided flags, for *which='all'* return all possible *Atom Flags* (page 56) labels in addition to those present in the instance.

**getFlags** *(label)*

Return a copy of atom flags for given *label*, or *None* when flags for *label* is not set.

**getFragindices** *(*)

Return a copy of fragment indices. Fragment indices are assigned to connected subsets of atoms. Bonds needs to be set using *AtomGroup.setBonds()* (page 42) method. Fragment indices start from zero, are incremented by one, and are assigned in the order of appearance in *AtomGroup* (page 37) instance. Fragment indices can be used in atom selections, e.g. ‘fragindex 0’, ‘fragment 1’. Note that *fragment* is a synonym for *fragindex*.

**getIcodes** *(*)

Return a copy of insertion codes. Insertion codes can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

**getIndices** *(*)

Return a copy of the indices of atoms.

**getMasses** *(*)

Return a copy of masses. Masses can be used in atom selections, e.g. ‘12 <= mass <= 13.5’.

**getName** *(*)

Return a copy of names. Names can be used in atom selections, e.g. ‘name CA CB’.

**getOccupancies** *(*)

Return a copy of occupancy values. Occupancy values can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.

**getRadii** *(*)

Return a copy of radii. Radii can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

**getResidue** *(resnum, icode=None)*

Return residue with number *resnum* and insertion code *icode*.

**getResindices** *(*)

Return a copy of residue indices. Residue indices are assigned to subsets of atoms with distinct sequences of residue number, insertion code, chain identifier, and segment name. Residue indices start from zero, are incremented by one, and are assigned in the order of appearance in *AtomGroup* (page 37) instance. Residue indices can be used in atom selections, e.g. ‘resindex 0’.

**getResnames** *(*)

Return a copy of residue names. Residue names can be used in atom selections, e.g. ‘resname ALA GLY’.

**getResnums** *(*)

Return a copy of residue numbers. Residue numbers can be used in atom selections, e.g. ‘resnum 1 2 3’, ‘resnum 120A 120B’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’. Note that *resid* is a synonym for *resnum*.

**getSecstrs** *(*)

Return a copy of secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that *secstr* is a synonym for *secondary*.
getSegindices()
Return a copy of segment indices. Segment indices are assigned to subsets of atoms with distinct segment names. Segment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Segment indices can be used in atom selections, e.g. ‘segindex 0’.

def getSegment():
    """Return segment of the chain."

def getSegname():
    """Return segment name."

def getSegnames():
    """Return a copy of segment names. Segment names can be used in atom selections, e.g. ‘segment PROT’, ‘segname PROT’. Note that segname is a synonym for segment."

def getSelstr():
    """Return selection string that selects atoms in this chain."

def getSequence(**kwargs):
    """Return one-letter sequence string for amino acids in the chain. When allres keyword argument is True, sequence will include all residues (e.g. water molecules) in the chain and X will be used for non-standard residue names."

def getSerials():
    """Return a copy of serial numbers (from file). Serial numbers can be used in atom selections, e.g. ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’."

def getTypes():
    """Return a copy of types. Types can be used in atom selections, e.g. ‘type CT1 CT2 CT3’."

def isDataLabel(label):
    """Return True if data associated with label is present."

def isFlagLabel(label):
    """Return True if flags associated with label are present."

def iterAtoms():
    """Yield atoms."

def iterCoordsets():
    """Yield copies of coordinate sets."

def iterResidues():
    """Yield residues."

def numAtoms(flag=None):
    """Return number of atoms, or number of atoms with given flag."

def numCoordsets():
    """Return number of coordinate sets."

def numResidues():
    """Return number of residues."

def select(selstr, **kwargs):
    """Return atoms matching selstr criteria. See select (page 78) module documentation for details and usage examples."

def setACSIndex(index):
    """Set coordinates at index active."

3.1. Atomic Data
setAltlocs(data)
Set alternate location indicators. Alternate location indicators can be used in atom selections, e.g. ‘altloc A B’, ‘altloc _’.

setAnisous(data)
Set anisotropic temperature factors.

setAnistds(data)
Set standard deviations for anisotropic temperature factors.

setBetas(data)
Set $\beta$-values (or temperature factors). $\beta$-values can be used in atom selections, e.g. ‘beta 555.55’, ‘beta 0 to 500’, ‘beta 0:500’, ‘beta < 500’.

setCharges(data)
Set partial charges. Partial charges can be used in atom selections, e.g. ‘charge 1’, ‘abs(charge) == 1’, ‘charge < 0’.

setChid(chid)
Set chain identifier.

setChids(data)
Set chain identifiers. Chain identifiers can be used in atom selections, e.g. ‘chain A’, ‘chid A B C’, ‘chain _’. Note that chid is a synonym for chain.

setCoords(coords)
Set coordinates in the active coordinate set.

setData(label, data)
Update data associated with label.

   Raises AttributeError when label is not in use or read-only

setElements(data)
Set element symbols. Element symbols can be used in atom selections, e.g. ‘element C O N’.

setFlags(label, value)
Update flag associated with label.

   Raises AttributeError when label is not in use or read-only

setIcodes(data)
Set insertion codes. Insertion codes can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

setMasses(data)
Set masses. Masses can be used in atom selections, e.g. ‘12 <= mass <= 13.5’.

setName(data)
Set names. Names can be used in atom selections, e.g. ‘name CA CB’.

setOccupancies(data)
Set occupancy values. Occupancy values can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.

setRadii(data)
Set radii. Radii can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

setResnames(data)
Set residue names. Residue names can be used in atom selections, e.g. ‘resname ALA GLY’.

setResnums(data)
Set residue numbers. Residue numbers can be used in atom selections, e.g. ‘resnum 1 2’.
Note that resid is a synonym for resnum.

**setSecstrs** *(data)*

Set secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that secstr is a synonym for secondary.

**setSegnames** *(data)*

Set segment names. Segment names can be used in atom selections, e.g. ‘segment PROT’, ‘segname PROT’. Note that segname is a synonym for segment.

**setSerials** *(data)*

Set serial numbers (from file). Serial numbers can be used in atom selections, e.g. ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’.

**setTypes** *(data)*

Set types. Types can be used in atom selections, e.g. ‘type CT1 CT2 CT3’.

### 3.1.12 Atom Data Fields

This module defines atomic data fields. You can read this page in interactive sessions using `help(fields)`.

Data parsed from PDB and other supported files for these fields are stored in `AtomGroup` instances. Available data fields are listed in the table below. Atomic classes, such as `Selection` (page 86), offer `get` and `set` for handling parsed data:

Many of these data fields can be used to make Atom Selections (page 78). Following table lists definitions of fields and selection examples. Note that fields noted as *read only* do not have a `set` method.

- **altloc** alternate location indicator
  
  E.g.: ‘altloc A B’, ‘altloc _’

- **anisou** anisotropic temperature factor

- **beta** $\beta$-value (temperature factor)
  
  E.g.: ‘beta 555.55’, ‘beta 0 to 500’, ‘beta 0:500’, ‘beta < 500’

- **chain, chid** chain identifier
  
  E.g.: ‘chain A’, ‘chid A B C’, ‘chain _’

- **charge** partial charge
  
  E.g.: ‘charge 1’, ‘abs(charge) == 1’, ‘charge < 0’

- **chindex** chain index *(read only)*
  
  E.g.: ‘chindex 0’

- **element** element symbol
  
  E.g.: ‘element C O N’

- **fragindex, fragment** fragment index *(read only)*
  
  E.g.: ‘fragindex 0’, ‘fragment 1’

- **icode** insertion code
  
  E.g.: ‘icode A’, ‘icode _’
mass  mass
  E.g.: ‘12 <= mass <= 13.5’

name  name
  E.g.: ‘name CA CB’

numbonds  number of bonds (read only)
  E.g.: ‘numbonds 0’, ‘numbonds 1’

occupancy  occupancy value
  E.g.: ‘occupancy 1’, ‘occupancy > 0’

radius  radius
  E.g.: ‘radii < 1.5’, ‘radii ** 2 < 2.3’

resindex  residue index (read only)
  E.g.: ‘resindex 0’

resname  residue name
  E.g.: ‘resname ALA GLY’

resnum, resid  residue number
  E.g.: ‘resnum 1 2 3’, ‘resnum 120A 120B’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’

secondary, secstr  secondary structure assignment
  E.g.: ‘secondary H E’, ‘secstr H E’

segindex  segment index (read only)
  E.g.: ‘segindex 0’

segment, segname  segment name
  E.g.: ‘segment PROT’, ‘segname PROT’

serial  serial number (from file)
  E.g.: ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’

siguij  standard deviations for anisotropic temperature factor

type  type
  E.g.: ‘type CT1 CT2 CT3’

class Field(name, dtype, **kwargs)
  Atomic data field.

getDocstr(meth, plural=True, selex=True)
  Return documentation string for the field.

call
  list of AtomGroup (page 37) methods to call when getMethod is called
depr
  deprecated method name
depr_pl
  deprecated method name in plural form

3.1. Atomic Data  55
3.1.13 Atom Flags

This module defines atom flags that are used in Atom Selections (page 78). You can read this page in interactive sessions using help(flags).

Flag labels can be used in atom selections:

```
In [1]: from prody import *

In [2]: p = parsePDB('lubi')

In [3]: p.select('protein')
Out[3]: <Selection: 'protein' from lubi (602 atoms)>
```

Flag labels can be combined with dot operator as follows to make selections:

```
In [4]: p.protein
Out[4]: <Selection: 'protein' from lubi (602 atoms)>
```
In [5]: p.protein.acidic  # selects acidic residues
Out[5]: <Selection: '(acidic) and (protein)' from 1ubi (94 atoms)>

Flag labels can be prefixed with 'is' to check whether all atoms in an Atomic (page 44) instance are flagged the same way:

In [6]: p.protein.ishetero
Out[6]: False

In [7]: p.water.ishetero
Out[7]: True

Flag labels can also be used to make quick atom counts:

In [8]: p.numAtoms()
Out[8]: 683

In [9]: p.numAtoms('protein')
Out[9]: 602

In [10]: p.numAtoms('water')
Out[10]: 81

Protein

protein, aminoacid indicates the twenty standard amino acids (stdaa) and some non-standard amino acids (nonstdaa) described below. Residue must also have an atom named 'CA' in addition to having a qualifying residue name.

stdaa indicates the standard amino acid residues: ALA26, ARG27, ASN28, ASP29, CYS30, GLN31, GLU32, GLY33, HIS34, ILE35, LEU36, LYS37, MET38, PHE39, PRO40, SER41, THR42, TRP43, TYR44, and VAL45

nonstdaa indicates one of the following residues:

http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=ALA
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=ARG
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=ASN
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=ASP
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CYS
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=GLN
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=GLU
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=GLY
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HIS
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=ILE
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=LEU
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=LYS
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=MET
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=PHE
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=PRO
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=SER
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=THR
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=TRP
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=TYR
http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=VAL

3.1. Atomic Data 57
ASX$^{46}$ (B) asparagine or aspartic acid
GLX$^{47}$ (Z) glutamine or glutamic acid
CSO$^{48}$ (C) S-hydroxycysteine
HIP$^{49}$ (H) ND1-phosphohistidine
HSD (H) prototropic tautomer of histidine, H on ND1 (CHARMM)
HSE (H) prototropic tautomer of histidine, H on NE2 (CHARMM)
HSP (H) protonated histidine
MSE$^{50}$ selenomethionine
SEC$^{51}$ (U) selenocysteine
SEP$^{52}$ (S) phosphoserine
TPO$^{53}$ (T) phosphothreonine
PTR$^{54}$ (Y) O-phosphotyrosine
XLE (J) leucine or isoleucine
XAA (X) unspecified or unknown

You can modify the list of non-standard amino acids using addNonstdAminoacid() (page 65), delNonstdAminoacid() (page 65), and listNonstdAAProps() (page 65).

calpha, ca Ca atoms of protein residues, same as selection 'name CA and protein'
backbone, bb non-hydrogen backbone atoms of protein residues, same as selection 'name CA C O N and protein'
backbonefull, bbfull backbone atoms of protein residues, same as selection 'name CA C O N H1 H2 H3 OXT and protein'
sidechain, sc side-chain atoms of protein residues, same as selection 'protein and not backbonefull'
acidic residues ASP, GLU, HSP, PTR, SEP, TPO
acyclic residues ALA, ARG, ASN, ASP, ASX, CSO, CYS, GLN, GLU, GLX, GLY, ILE, EU, LYS, MET, MSE, SEC, SEP, SER, THR, TPO, VAL, XLE
aliphatic residues ALA, GLY, ILE, LEU, PRO, VAL, XLE
aromatic residues HIS, PHE, PTR, TRP, TYR
basic residues ARG, HIP, HIS, HSD, HSE, LYS
buried residues ALA, CYS, ILE, LEU, MET, MSE, PHE, SEC, TRP, VAL, XLE
charged residues ARG, ASP, GLU, HIS, LYS
cyclic residues HIP, HIS, HSD, HSE, HSP, PHE, PRO, PTR, TRP, TYR
hydrophobic residues ALA, ILE, LEU, MET, PHE, PRO, TRP, VAL, XLE
large residues ARG, GLN, GLU, GLX, HIP, HIS, HSD, HSE, HSP, ILE, LEU, LYS, ET, MSE, PHE, PTR, SEP, TPO, TRP, TYR, XLE

46http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=ASX
47http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=GLX
48http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CSO
49http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HIP
50http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=MSE
51http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=SEC
52http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=SEP
53http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=TPO
54http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=PTR
medium residues ASN, ASP, ASX, CSO, CYS, PRO, SEC, THR, VAL

neutral residues ALA, ASN, CSO, CYS, GLN, GLY, ILE, LEU, MET, MSE, PHE, PRO, EC, SER, THR, TRP, TYR, VAL

polar residues ARG, ASN, ASP, ASX, CSO, CYS, GLU, GLX, GLY, HIP, HIS, SD, HSE, HSP, LYS, PTR, SEC, SEP, SER, THR, TPO, TYR

small residues ALA, GLY, SER

surface residues ARG, ASN, ASP, ASX, CSO, GLN, GLU, GLX, GLY, HIP, HIS, HSD, SE, HSP, LYS, PRO, PTR, SEP, SER, THR, TPO, TYR

Nucleic

nucleic indicates nucleobase, nucleotide, and some nucleoside derivatives that are described below, so it is same as ‘nucleobase or nucleotide or nucleoside’.
nucleobase indicates ADE\(^{55}\) (adenine), GUN\(^{56}\) (guanine), CYT\(^{57}\) (cytosine), THY\(^{58}\) (thymine), and URA\(^{59}\) (uracil).
nucleotide indicates residues with the following names:

| DA\(^{60}\)  | 2’-deoxyadenosine-5’-monophosphate |
| DC\(^{61}\)  | 2’-deoxycytidine-5’-monophosphate  |
| DG\(^{62}\)  | 2’-deoxyguanosine-5’-monophosphate |
| DT\(^{63}\)  | 2’-deoxythymidine-5’-monophosphate |
| DU\(^{64}\)  | 2’-deoxyuridine-5’-monophosphate  |
| A\(^{65}\)   | adenosine-5’-monophosphate         |
| C\(^{66}\)   | cytidine-5’-monophosphate          |
| G\(^{67}\)   | guanosine-5’-monophosphate         |
| T\(^{68}\)   | 2’-deoxythymidine-5’-monophosphate |
| U\(^{69}\)   | uridine-5’-monophosphate           |

nucleoside indicates following nucleoside derivatives that are recognized by PDB:

\(^{55}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=ADE
\(^{56}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=GUN
\(^{57}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CYT
\(^{58}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=THY
\(^{59}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=URA
\(^{60}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=DA
\(^{61}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=DC
\(^{62}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=DG
\(^{63}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=DT
\(^{64}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=DU
\(^{65}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=A
\(^{66}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=C
\(^{67}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=G
\(^{68}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=T
\(^{69}\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=U
| AMP<sup>70</sup> | adenosine monophosphate |
| ADP<sup>71</sup> | adenosine-5'-diphosphate |
| ATP<sup>72</sup> | adenosine-5'-triphosphate |
| CDP<sup>73</sup> | cytidine-5'-diphosphate |
| CTP<sup>74</sup> | cytidine-5'-triphosphate |
| GMP<sup>75</sup> | guanosine |
| GDP<sup>76</sup> | guanosine-5'-diphosphate |
| GTP<sup>77</sup> | guanosine-5'-triphosphate |
| TMP<sup>78</sup> | thymidine-5'-phosphate |
| TTP<sup>79</sup> | thymidine-5'-triphosphate |
| UMP<sup>80</sup> | 2'-deoxyuridine 5'-monophosphate |
| UDP<sup>81</sup> | uridine 5'-diphosphate |
| UTP<sup>82</sup> | uridine 5'-triphosphate |

at  same as selection 'resname ADE A THY T'
cg  same as selection 'resname CYT C GUN G'
purine same as selection 'resname ADE A GUN G'
pyrimidine same as selection 'resname CYT C THY T URA U'

**Heteros**

hetero  indicates anything other than a protein or a nucleic residue, i.e. 'not (protein or nucleic)'.
hetatm  is available when atomic data is parsed from a PDB or similar format file and indicates atoms that are marked 'HETATM' in the file.
water  indices HOH<sup>83</sup> and DOD<sup>84</sup> recognized by PDB and also WAT, TIP3, H2O, OH2, TIP, TIP2, and TIP4 recognized by molecular dynamics (MD) force fields.

Previously used water types HH0, OHH, and SOL conflict with other compounds in the PDB, so are removed from the definition of this flag.
ion  indicates the following ions most of which are recognized by the PDB and others by MD force fields.

<table>
<thead>
<tr>
<th>ion&lt;sup&gt;85&lt;/sup&gt;</th>
<th>PDB</th>
<th>Source</th>
<th>Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL&lt;sup&gt;86&lt;/sup&gt;</td>
<td>aluminum</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>BA&lt;sup&gt;86&lt;/sup&gt;</td>
<td>barium</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

<sup>70</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=AMP
<sup>71</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=ADP
<sup>72</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=ATP
<sup>73</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CDP
<sup>74</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CTP
<sup>75</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=GMP
<sup>76</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=GDP
<sup>77</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=GTP
<sup>78</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=TMP
<sup>79</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=TTP
<sup>80</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=UMP
<sup>81</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=UDP
<sup>82</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=UTP
<sup>83</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HOH
<sup>84</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=DOD
<sup>85</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=AL
<sup>86</sup>http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=BA

3.1. Atomic Data

60
### Table 3.3 – continued from previous page

<table>
<thead>
<tr>
<th>Ion Identifier</th>
<th>Element</th>
<th>PDB</th>
<th>CHARMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>calcium</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td>cadmium</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CL</td>
<td>chloride</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CO</td>
<td>cobalt (ii)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td>cesium</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CU</td>
<td>copper (ii)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CU1</td>
<td>copper (i)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CUA</td>
<td>dinuclear copper</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>HG</td>
<td>mercury (ii)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>indium (iii)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>IOD</td>
<td>iodide</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>potassium</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>MG</td>
<td>magnesium</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>MN3</td>
<td>manganese (iii)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>NA</td>
<td>sodium</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>PB</td>
<td>lead (ii)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>PT</td>
<td>platinum (ii)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>RB</td>
<td>rubidium</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>TB</td>
<td>terbium (iii)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>TL</td>
<td>thallium (i)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>WO4</td>
<td>thungstate (vi)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>YB</td>
<td>ytterbium (ii)</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>ZN</td>
<td>zinc</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>CAL</td>
<td>calcium</td>
<td>No</td>
<td>CHARMM</td>
</tr>
<tr>
<td>CES</td>
<td>cesium</td>
<td>No</td>
<td>CHARMM</td>
</tr>
<tr>
<td>CLA</td>
<td>chloride</td>
<td>No</td>
<td>CHARMM</td>
</tr>
<tr>
<td>POT</td>
<td>potassium</td>
<td>No</td>
<td>CHARMM</td>
</tr>
<tr>
<td>SOD</td>
<td>sodium</td>
<td>No</td>
<td>CHARMM</td>
</tr>
<tr>
<td>ZN2</td>
<td>zinc</td>
<td>No</td>
<td>CHARMM</td>
</tr>
</tbody>
</table>

Ion identifiers that are obsoleted by PDB (MO3, MO4, MO5, MO6, NAW, OC7, and ZN1) are removed from this definition.

87 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CA
88 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CD
89 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CL
90 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CO
91 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CS
92 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CU
93 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=CUA
94 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HG
95 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=IN
96 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=IOD
97 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=K
98 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=MG
99 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=MN3
100 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=NA
101 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=PB
102 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=PT
103 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=RB
104 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=TB
105 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=TL
107 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=YB
108 http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=ZN

3.1. Atomic Data
lipid indicates GPE\textsuperscript{110}, LPP\textsuperscript{111}, OLA\textsuperscript{112}, SDS\textsuperscript{113}, and STE\textsuperscript{114} from PDB, and also POPC, LPPC, POPE, DLPE, PCGL, STEA, PALM, OLEO, DMPC from CHARMM force field.

sugar indicates BGC\textsuperscript{115}, GLC\textsuperscript{116}, and GLO\textsuperscript{117} from PDB, and also AGLC from CHARMM.

heme indicates 1FH\textsuperscript{118}, 2FH\textsuperscript{119}, DDH\textsuperscript{120}, DHE\textsuperscript{121}, HAS\textsuperscript{122}, HDD\textsuperscript{123}, HDE\textsuperscript{124}, HDM\textsuperscript{125}, HEA\textsuperscript{126}, HEB\textsuperscript{127}, HEC\textsuperscript{128}, HEM\textsuperscript{129}, HEO\textsuperscript{130}, HES\textsuperscript{131}, HEV\textsuperscript{132}, NTE\textsuperscript{133}, SRM\textsuperscript{134}, and VER\textsuperscript{135} from PDB, and also HEMO and HEMR from CHARMM.

\texttt{pdbter} is available when atomic data is parsed from a PDB format file and indicates atoms that were followed by ‘TER’ record.

### Elements

Following elements found in proteins are recognized by applying regular expressions to atom names:

- **carbon** carbon atoms, same as ‘name "C.*" and not ion’
- **nitrogen** nitrogen atoms, same as ‘name "N.*" and not ion’
- **oxygen** oxygen atoms, same as ‘name "O.*" and not ion’
- **sulfur** sulfur atoms, same as ‘name "S.*" and not ion’
- **hydrogen** hydrogen atoms, same as ‘name "[1-9]?H.*" and not ion’
- **noh, heavy** non hydrogen atoms, same as ‘not hydrogen

‘not ion’ is appended to above definitions to avoid conflicts with ion atoms.

### Structure

Following secondary structure flags are defined but before they can be used, secondary structure assignments must be made.

- **extended** extended conformation, same as ‘secondary E’

\begin{itemize}
  \item[\textsuperscript{110}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=GPE
  \item[\textsuperscript{111}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=LPP
  \item[\textsuperscript{112}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=OLA
  \item[\textsuperscript{113}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=SDS
  \item[\textsuperscript{114}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=STE
  \item[\textsuperscript{115}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=BGC
  \item[\textsuperscript{116}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=GLC
  \item[\textsuperscript{117}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=GLO
  \item[\textsuperscript{118}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=1FH
  \item[\textsuperscript{119}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=2FH
  \item[\textsuperscript{120}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=DDH
  \item[\textsuperscript{121}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=DHE
  \item[\textsuperscript{122}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HAS
  \item[\textsuperscript{123}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HDM
  \item[\textsuperscript{124}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HEA
  \item[\textsuperscript{125}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HEB
  \item[\textsuperscript{126}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HEC
  \item[\textsuperscript{127}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HEM
  \item[\textsuperscript{128}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HEO
  \item[\textsuperscript{129}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HES
  \item[\textsuperscript{130}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=HEV
  \item[\textsuperscript{131}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=NTE
  \item[\textsuperscript{132}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=SRM
  \item[\textsuperscript{133}]http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=VER
\end{itemize}
helix  α-helix conformation, same as ‘secondary H’

helix310  3_10-helix conformation, same as ‘secondary G’

helixpi  π-helix conformation, same as ‘secondary I’

turn  hydrogen bonded turn conformation, same as ‘secondary T’

bridge  isolated beta-bridge conformation, same as ‘secondary B’

bend  bend conformation, same as ‘secondary S’

coil  not in one of above conformations, same as ‘secondary C’

Others

all  indicates all atoms, returns a new view of the instance

none  indicates no atoms, returns None

dummy  indicates dummy atoms in an AtomMap (page 45)

mapped  indicates mapped atoms in an AtomMap (page 45)

Functions

Following functions can be used to customize flag definitions:

- `flagDefinition()` (page 63)
- `addNonstdAminoacid()` (page 65)
- `delNonstdAminoacid()` (page 65)
- `listNonstdAAProps()` (page 65)

`flagDefinition(*arg, **kwarg)`

Learn, change, or reset Atom Flags (page 56) definitions.

Learn a definition

Calling this function with no arguments will return list of flag names whose definitions you can learn:

```
In [1]: flagDefinition()
Out[1]:
['acidic',
 'acyclic',
 'aliphatic',
 'aminoacid',
 'aromatic',
 'at',
 'backbone',
 'backbonefull',
 'basic',
 'bb',
 'bbfull',
 'buried',
 'carbon',
 'cg',
 'charged',
 'cyclic',
 'heme',
```
Passing a flag name will return its definition:

In [2]: flagDefinition('backbone')
Out[2]: ['C', 'CA', 'N', 'O']

In [3]: flagDefinition('hydrogen')
Out[3]: '[0-9]?H.*'

Change a definition

Calling the function with editable=True argument will return flag names those definitions that can be edited:

In [4]: flagDefinition(editable=True)
Out[4]:
['at', 'backbone', 'backbonefull', 'bb', 'bbfull', 'carbon', 'cg', 'heme', 'hydrogen', 'ion', 'lipid', 'nitrogen', 'nucleobase', 'nucleoside', 'nucleotide', 'oxygen', 'purine', 'pyrimidine', 'sugar',...

3.1. Atomic Data
Pass an editable flag name with its new definition:

In [5]: flagDefinition(nitrogen='N.*')

In [6]: flagDefinition(backbone=['CA', 'C', 'O', 'N'])

In [7]: flagDefinition(nucleobase=['ADE', 'CYT', 'GUN', 'THY', 'URA'])

Note that the type of the new definition must be the same as the type of the old definition. Flags with editable definitions are: at, backbone, backbonefull, bb, bbfull, carbon, cg, heme, hydrogen, ion, lipid, nitrogen, nucleobase, nucleoside, nucleotide, oxygen, purine, pyrimidine, sugar, sulfur, and water

Reset definitions

Pass reset keyword as follows to restore all default definitions of editable flags and also non-standard amino acids.

In [8]: flagDefinition(reset='all')

Or, pass a specific editable flag label to restore its definition:

In [9]: flagDefinition(reset='nitrogen')

listNonstdAAProps (resname)
Return properties of non-standard amino acid resname.

In [1]: listNonstdAAProps('PTR')
Out[1]: ['acidic', 'aromatic', 'cyclic', 'large', 'polar', 'surface']

getNonstdProperties (resname)
Deprecated for removal in v1.4, use listNonstdAAProps() (page 65) instead.

addNonstdAminoacid (resname, *properties)
Add non-standard amino acid resname with properties selected from:

• cyclic: acyclic, or cyclic
• charge: acidic, basic, or neutral
• depth: buried, or surface
• hydrophobicity: hydrophobic, or polar
• aromaticity: aliphatic, or aromatic
• size: large, medium, or small

In [1]: addNonstdAminoacid('PTR', 'acidic', 'aromatic', 'cyclic', 'large', ...
   ...: 'polar', 'surface')

Default set of non-standard amino acids can be restored as follows:

In [2]: flagDefinition(reset='nonstdaa')

delNonstdAminoacid (resname)
Delete non-standard amino acid resname.
In [1]: delNonstdAminoacid('PTR')

In [2]: flagDefinition('nonstdaa')
Out[2]:
['ASX',
 'CSO',
 'GLX',
 'HIP',
 'HSD',
 'HSE',
 'HSP',
 'MSE',
 'SEC',
 'SEP',
 'TPO',
 'XAA',
 'XLE']

Default set of non-standard amino acids can be restored as follows:

In [3]: flagDefinition(reset='nonstdaa')

3.1.14 Supporting Functions

This module defines some functions for handling atomic classes and data.

iterFragments (atoms)
Yield fragments, connected subsets in atoms, as Selection (page 86) instances.

findFragments (atoms)
Return list of fragments, connected subsets in atoms. See also iterFragments() (page 66).

loadAtoms (filename)
Return AtomGroup (page 37) instance loaded from filename using numpy.load() function. See also saveAtoms() (page 66).

saveAtoms (atoms, filename=None, **kwargs)
Save atoms in ProDy internal format. All Atomic (page 44) classes are accepted as atoms argument. This function saves user set atomic data as well. Note that title of theAtomGroup (page 37) instance is used as the filename when atoms is not an AtomGroup (page 37). To avoid overwriting an existing file with the same name, specify a filename.

isReserved (word)
Return True if word is reserved for internal data labeling or atom selections. See listReservedWords() (page 66) for a list of reserved words.

listReservedWords ()
Return list of words that are reserved for atom selections and internal variables. These words are: abs, acidic, acos, acyclic, aliphatic, all, altloc, aminoacid, and, anisou, aromatic, as, asin, at, atan, backbone, backbonefull, basic, bb, bbfull, bend, beta, bmap, bonded, bonds, bridge, buried, ca, calpha, carbon, ceil, cg, chain, charge, charged, chid, chinindex, coil, coordinates, cos, cosh, cslabels, cyclic, dummy, element, exbonded, exp, extended, exwithin, floor, fragindex, fragment, heavy, helix, helix310, helixpi, heme, hetero, hydrogen, hydrophobic, icode, index, ion, large, lipid, log, log10, mapped, mass, medium, n_atoms, n_csets, name, neutral, nitrogen, noh, none, nonstdaa, not, nucleic, nucleobase, nucleotide, numbonds, occupancy, of, or, oxygen, polar, protein, purine, pyrimidine, radius, resid, resindex, resname, resnum, same, sc, secondary, secstr,
segindex, segment, segname, sequence, serial, sidechain, siguij, sin, sinh, small, sq, sqrt, stdaa, sugar, sulfur, surface, tahn, tan, title, to, turn, type, water, within, x, y, z.

**sortAtoms** *(atoms, label, reverse=False)*

Return an *AtomMap* (page 45) pointing to *atoms* sorted in ascending data *label* order, or optionally in *reverse* order.

### 3.1.15 Hierarchical Views

This module defines *HierView* (page 67) class that builds a hierarchical views of atom groups.

**class HierView**(atoms, **kwargs)

Hierarchical views can be generated for *AtomGroup* (page 37) and *Selection* (page 86) instances. Indexing a *HierView* (page 67) instance returns a *Chain* (page 49) instance.

Some object methods are customized as follows:

- **len()**
  - Returns the number of atoms, i.e. *numChains()* (page 68)

- **iter()**
  - Yields Chain (page 49) instances

- **indexing by:**
  - Segment name (str()), e.g. "PROT", returns a Segment (page 74)
  - Chain identifier (str()), e.g. "A", returns a Chain (page 49)
  - [segment name,] chain identifier, residue number[, insertion code] (tuple()), e.g. "A", 10 or "A", 10, "B" or "PROT", "A", 10, "B", returns a Residue (page 69)

Note that when an *AtomGroup* (page 37) instance have distinct segments, they will be considered when building the hierarchical view. A Segment (page 74) instance will be generated for each distinct segment name. Then, for each segment chains and residues will be evaluated. Having segments in the structure will not change most behaviors of this class, except indexing. For example, when indexing a hierarchical view for chain P in segment PROT needs to be indexed as *hv[’PROT’, ’P’]*.

**getAtoms()**

Return atoms for which the hierarchical view was built.

**getChain**(chid, segname=None)

Return chain with identifier *chid*, if it is present.

**getResidue**(chid, resnum, icode=None, segname=None)

Return residue with number *resnum* and insertion code *icode* from the chain with identifier *chid* in segment with name *segname*.

**getSegment**(segname)

Return segment with name *segname*, if it is present.

**iterChains()**

Yield chains.

**iterResidues()**

Yield residues.

**iterSegments()**

Yield segments.

---

[137]http://docs.python.org/library/functions.html#len


[139]http://docs.python.org/library/functions.html#str

[140]http://docs.python.org/library/functions.html#str

[141]http://docs.python.org/library/functions.html#tuple
numChains()
    Return number of chains.

numResidues()
    Return number of residues.

numSegments()
    Return number of chains.

update(**kwargs)
    Update (or build) hierarchical view of atoms. This method is called at instantiation, but can be
    used to rebuild the hierarchical view when attributes of atoms change.

3.1.16 Atom Pointer

This module defines atom pointer base class.

class AtomPointer (ag, acsi)
    A base for classes pointing to atoms in AtomGroup (page 37) instances. Derived classes are:
    • Atom (page 33)
    • AtomSubset (page 90)
    • AtomMap (page 45)

copy()
    Return a copy of atoms (and atomic data) in an AtomGroup (page 37) instance.

getACSIndex()
    Return index of the coordinate set.

getACSLabel()
    Return active coordinate set label.

getAtomGroup()
    Return associated atom group.

getCSLabels()
    Return coordinate set labels.

dataLabels *(which=None)*
    Return data labels. For which='user', return only labels of user provided data.

dataType *(label)*
    Return type of the data (i.e. data.dtype) associated with label, or None label is not used.

getFlagLabels *(which=None)*
    Return flag labels. For which='user', return labels of user or parser (e.g. hetatm) provided
    flags, for which='all' return all possible Atom Flags (page 56) labels in addition to those
    present in the instance.

dataLabel *(label)*
    Return True if data associated with label is present.

FlagLabel *(label)*
    Return True if flags associated with label are present.

numCoordsets()
    Return number of coordinate sets.
select (selstr, **kwargs)
    Return atoms matching selstr criteria. See select (page 78) module documentation for details and usage examples.

setACSIndex (index)
    Set coordinates at index active.

3.1.17 Residue

This module defines classes for handling residues.

class Residue (ag, indices, hv, acsi=None, **kwargs)
    Instances of this class point to atoms with same residue numbers (and insertion codes) and are generated by HierView (page 67) class. Following built-in functions are customized for this class:

    • len()\(^{142}\) returns the number of atoms in the instance.
    • iter()\(^{143}\) yields Atom (page 33) instances.

Indexing Residue (page 69) instances by atom name (str())\(^{144}\), e.g. "CA" returns an Atom (page 33) instance.

    copy ()
        Return a copy of atoms (and atomic data) in an AtomGroup (page 37) instance.

    getACSIndex ()
        Return index of the coordinate set.

    getACSLable ()
        Return active coordinate set label.

    getAltlocs ()
        Return a copy of alternate location indicators. Alternate location indicators can be used in atom selections, e.g. 'altloc A B', 'altloc _'.

    getAnisous ()
        Return a copy of anisotropic temperature factors.

    getAnistds ()
        Return a copy of standard deviations for anisotropic temperature factors.

    getAtom (name)
        Return atom with given name, None if not found. Assumes that atom names in the residue are unique. If more than one atoms with the given name exists, the one with the smaller index will be returned.

    getAtomGroup ()
        Return associated atom group.

    getBetas ()
        Return a copy of β-values (or temperature factors). β-values can be used in atom selections, e.g. 'beta 555.55', 'beta 0 to 500', 'beta 0:500', 'beta < 500'.

    getCSLabels ()
        Return coordinate set labels.

    getChain ()
        Return the chain that the residue belongs to.

---

\(^{142}\)http://docs.python.org/library/functions.html#len
\(^{143}\)http://docs.python.org/library/functions.html#iter
\(^{144}\)http://docs.python.org/library/functions.html#str
getCharges()  
Return a copy of partial charges. Partial charges can be used in atom selections, e.g. ‘charge 1’, ‘abs(charge) == 1’, ‘charge < 0’.

getChid()  
Return chain identifier.

getChids()  
Return a copy of chain identifiers. Chain identifiers can be used in atom selections, e.g. ‘chain A’, ‘chid A B C’, ‘chain _’. Note that chid is a synonym for chain.

getChindices()  
Return a copy of chain indices. Chain indices are assigned to subsets of atoms with distinct pairs of chain identifier and segment name. Chain indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Chain indices can be used in atom selections, e.g. ‘chindex 0’.

getCoords()  
Return a copy of coordinates from the active coordinate set.

getCoordsets(indices=None)  
Return coordinate set(s) at given indices, which may be an integer or a list/array of integers.

data (label)  
Return a copy of data associated with label, if it is present.

data (which=None)  
Return data labels. For which='user', return only labels of user provided data.

data (label)  
Return type of the data (i.e. data.dtype) associated with label, or None label is not used.

elements()  
Return a copy of element symbols. Element symbols can be used in atom selections, e.g. ‘element C O N’.

getFlagLabels(which=None)  
Return flag labels. For which='user', return labels of user or parser (e.g. hetatm) provided flags, for which='all' return all possible Atom Flags (page 56) labels in addition to those present in the instance.

getFlags(label)  
Return a copy of atom flags for given label, or None when flags for label is not set.

getFragindices()  
Return a copy of fragment indices. Fragment indices are assigned to connected subsets of atoms. Bonds needs to be set using AtomGroup.setBonds() (page 42) method. Fragment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Fragment indices can be used in atom selections, e.g. ‘fragindex 0’, ‘fragment 1’. Note that fragment is a synonym for fragindex.

getIcode()  
Return residue insertion code.

getIcodes()  
Return a copy of insertion codes. Insertion codes can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

getIndices()  
Return a copy of the indices of atoms.
getMasses()  
Return a copy of masses. Masses can be used in atom selections, e.g. ‘12 <= mass <= 13.5’.

getNames()  
Return a copy of names. Names can be used in atom selections, e.g. ’name CA CB’.

ggetNext()  
Return following residue in the atom group.

getOccupancies()  
Return a copy of occupancy values. Occupancy values can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.

getPrev()  
Return preceding residue in the atom group.

getRadii()  
Return a copy of radii. Radii can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

getResindex()  
Return residue index.

getResindices()  
Return a copy of residue indices. Residue indices are assigned to subsets of atoms with distinct sequences of residue number, insertion code, chain identifier, and segment name. Residue indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Residue indices can be used in atom selections, e.g. ’resindex 0’.

getResname()  
Return residue name.

getResnames()  
Return a copy of residue names. Residue names can be used in atom selections, e.g. ’resname ALA GLY’.

getResnum()  
Return residue number.

getResnums()  
Return a copy of residue numbers. Residue numbers can be used in atom selections, e.g. ‘resnum 1 2 3’, ‘resnum 120A 120B’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’. Note that resid is a synonym for resnum.

getSecstrs()  
Return a copy of secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that secstr is a synonym for secondary.

getSegindices()  
Return a copy of segment indices. Segment indices are assigned to subsets of atoms with distinct segment names. Segment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Segment indices can be used in atom selections, e.g. ’segindex 0’.

getSegment()  
Return segment of the residue.

getSegname()  
Return segment name.
getSegnames()
    Return a copy of segment names. Segment names can be used in atom selections, e.g. 'segment PROT', 'segname PROT'. Note that segname is a synonym for segment.

getSelstr()
    Return selection string that will select this residue.

getSerials()
    Return a copy of serial numbers (from file). Serial numbers can be used in atom selections, e.g. 'serial 1 2 3', 'serial 1 to 10', 'serial 1:10:2', 'serial < 10'.

getTypes()
    Return a copy of types. Types can be used in atom selections, e.g. 'type CT1 CT2 CT3'.

isDataLabel(label)
    Return True if data associated with label is present.

isFlagLabel(label)
    Return True if flags associated with label are present.

iterAtoms()
    Yield atoms.

iterCoordsets()
    Yield copies of coordinate sets.

numAtoms(flag=None)
    Return number of atoms, or number of atoms with given flag.

numCoordsets()
    Return number of coordinate sets.

select(selstr, **kwargs)
    Return atoms matching selstr criteria. See select (page 78) module documentation for details and usage examples.

setACSIndex(index)
    Set coordinates at index active.

setAltlocs(data)
    Set alternate location indicators. Alternate location indicators can be used in atom selections, e.g. 'altloc A B', 'altloc _'.

setAnisous(data)
    Set anisotropic temperature factors.

setAnistds(data)
    Set standard deviations for anisotropic temperature factors.

setBetas(data)
    Set β-values (or temperature factors). β-values can be used in atom selections, e.g. 'beta 555.55', 'beta 0 to 500', 'beta 0:500', 'beta < 500'.

setCharges(data)
    Set partial charges. Partial charges can be used in atom selections, e.g. 'charge 1', 'abs(charge) == 1', 'charge < 0'.

setChids(data)
    Set chain identifiers. Chain identifiers can be used in atom selections, e.g. 'chain A', 'chid A B C', 'chain _'. Note that chid is a synonym for chain.

setCoords(coords)
    Set coordinates in the active coordinate set.

3.1. Atomic Data 72
setData (label, data)
    Update data associated with label.

    Raises AttributeError when label is not in use or read-only

setElements (data)
    Set element symbols. Element symbols can be used in atom selections, e.g. 'element C O N'.

setFlags (label, value)
    Update flag associated with label.

    Raises AttributeError when label is not in use or read-only

setIcode (icode)
    Set residue insertion code.

setIcodes (data)
    Set insertion codes. Insertion codes can be used in atom selections, e.g. 'icode A', 'icode _'.

setMasses (data)
    Set masses. Masses can be used in atom selections, e.g. '12 <= mass <= 13.5'.

setNames (data)
    Set names. Names can be used in atom selections, e.g. 'name CA CB'.

setOccupancies (data)
    Set occupancy values. Occupancy values can be used in atom selections, e.g. 'occupancy 1', 'occupancy > 0'.

setRadii (data)
    Set radii. Radii can be used in atom selections, e.g. 'radii < 1.5', 'radii ** 2 < 2.3'.

setResname (name)
    Set residue name.

setResnames (data)
    Set residue names. Residue names can be used in atom selections, e.g. 'resname ALA GLY'.

setResnum (number)
    Set residue number.

setResnums (data)
    Set residue numbers. Residue numbers can be used in atom selections, e.g. 'resnum 1 2 3', 'resnum 120A 120B', 'resnum 10 to 20', 'resnum 10:20:2', 'resnum < 10'.
    Note that resid is a synonym for resnum.

setSecstrs (data)
    Set secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. 'secondary H E', 'secstr H E'. Note that secstr is a synonym for secondary.

setSegnames (data)
    Set segment names. Segment names can be used in atom selections, e.g. 'segment PROT', 'segname PROT'. Note that segname is a synonym for segment.

setSerials (data)
    Set serial numbers (from file). Serial numbers can be used in atom selections, e.g. 'serial 1 2 3', 'serial 1 to 10', 'serial 1:10:2', 'serial < 10'.

setTypes (data)
    Set types. Types can be used in atom selections, e.g. 'type CT1 CT2 CT3'.

3.1. Atomic Data
3.1.18 Segment

This module defines a class to handle segments of atoms in an atom group.

class Segment (ag, indices, hv, acsi=None, **kwargs)

Instances of this class point to atoms with same segment names and are generated by HierView (page 67) class. Following built-in functions are customized for this class:

- `len()` returns the number of chains in the segment.
- `iter()` yields Chain (page 49) instances.

Indexing Segment (page 74) instances by a chain identifier (str)\(^{147}\), e.g. A, returns a Chain (page 49).

copy()
Return a copy of atoms (and atomic data) in an AtomGroup (page 37) instance.

getACSIIndex()
Return index of the coordinate set.

getACSLabel()
Return active coordinate set label.

getAltlocs()
Return a copy of alternate location indicators. Alternate location indicators can be used in atom selections, e.g. ‘altloc A B’, ‘altloc _’.

getAnisous()
Return a copy of anisotropic temperature factors.

getAnistds()
Return a copy of standard deviations for anisotropic temperature factors.

getAtomGroup()
Return associated atom group.

getBetas()
Return a copy of β-values (or temperature factors). β-values can be used in atom selections, e.g. ‘beta 555.55’, ‘beta 0 to 500’, ‘beta 0:500’, ‘beta < 500’.

getCSLabels()
Return coordinate set labels.

getChain(chid)
Return chain with identifier chid.

getCharges()
Return a copy of partial charges. Partial charges can be used in atom selections, e.g. ‘charge 1’, ‘abs(charge) == 1’, ‘charge < 0’.

getChids()
Return a copy of chain identifiers. Chain identifiers can be used in atom selections, e.g. ‘chain A’, ‘chid A B C’, ‘chain _’. Note that chid is a synonym for chain.

getchindices()
Return a copy of chain indices. Chain indices are assigned to subsets of atoms with distinct pairs of chain identifier and segment name. Chain indices start from zero, are incremented by one,

\(^{145}\)http://docs.python.org/library/functions.html#len
\(^{146}\)http://docs.python.org/library/functions.html#iter
\(^{147}\)http://docs.python.org/library/functions.html#str
and are assigned in the order of appearance in AtomGroup (page 37) instance. Chain indices can be used in atom selections, e.g. ‘chindex 0’.

**getCoords()**
Return a copy of coordinates from the active coordinate set.

**getCoordsets(indices=None)**
Return coordinate set(s) at given indices, which may be an integer or a list/array of integers.

**getData(label)**
Return a copy of data associated with label, if it is present.

**getDatatLabels(which=None)**
Return data labels. For which='user’, return only labels of user provided data.

**getDataType(label)**
Return type of the data (i.e. data.dtype) associated with label, or None label is not used.

**getElements()**
Return a copy of element symbols. Element symbols can be used in atom selections, e.g. ‘element C O N’.

**getFlagLabels(which=None)**
Return flag labels. For which='user’, return labels of user or parser (e.g. hetatm) provided flags, for which='all’ return all possible Atom Flags (page 56) labels in addition to those present in the instance.

**getFlags(label)**
Return a copy of atom flags for given label, or None when flags for label is not set.

**getFragindices()**
Return a copy of fragment indices. Fragment indices are assigned to connected subsets of atoms. Bonds needs to be set using AtomGroup.setBonds() (page 42) method. Fragment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Fragment indices can be used in atom selections, e.g. ‘fragindex 0’, ‘fragment 1’. Note that fragment is a synonym for fragindex.

**getIcodes()**
Return a copy of insertion codes. Insertion codes can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

**getIndices()**
Return a copy of the indices of atoms.

**getMasses()**
Return a copy of masses. Masses can be used in atom selections, e.g. ‘12 <= mass <= 13.5’.

**getNames()**
Return a copy of names. Names can be used in atom selections, e.g. ‘name CA CB’.

**getOccupancies()**
Return a copy of occupancy values. Occupancy values can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.

**getRadii()**
Return a copy of radii. Radii can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

**getResindices()**
Return a copy of residue indices. Residue indices are assigned to subsets of atoms with distinct sequences of residue number, insertion code, chain identifier, and segment name. Residue indices start from zero, are incremented by one, and are assigned in the order of appearance in
**AtomGroup** (page 37) instance. Residue indices can be used in atom selections, e.g. ‘resindex 0’.

**getResnames**
Return a copy of residue names. Residue names can be used in atom selections, e.g. ‘resname ALA GLY’.

**getResnums**
Return a copy of residue numbers. Residue numbers can be used in atom selections, e.g. ‘resnum 1 2 3’, ‘resnum 120A 120B’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’. Note that resid is a synonym for resnum.

**getSecstrs**
Return a copy of secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that secstr is a synonym for secondary.

**getSegindices**
Return a copy of segment indices. Segment indices are assigned to subsets of atoms with distinct segment names. Segment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Segment indices can be used in atom selections, e.g. ‘segindex 0’.

**getSegname**
Return segment name.

**getSegnames**
Return a copy of segment names. Segment names can be used in atom selections, e.g. ‘segment PROT’, ‘segname PROT’. Note that segname is a synonym for segment.

**getSelstr**
Return selection string that selects atoms in this segment.

**getSerials**
Return a copy of serial numbers (from file). Serial numbers can be used in atom selections, e.g. ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’.

**getTypes**
Return a copy of types. Types can be used in atom selections, e.g. ‘type CT1 CT2 CT3’.

**isDataLabel** (label)
Return True if data associated with label is present.

**isFlagLabel** (label)
Return True if flags associated with label are present.

**iterAtoms**
Yield atoms.

**iterChains**
Yield chains.

**iterCoordsets**
Yield copies of coordinate sets.

**numAtoms** (flag=None)
Return number of atoms, or number of atoms with given flag.

**numChains**
Return number of chains.
3.1. Atomic Data

numCoordsets()
Return number of coordinate sets.

select(selstr,**kwargs)
Return atoms matching selstr criteria. See select (page 78) module documentation for details and usage examples.

setACSIndex(index)
Set coordinates at index active.

setAltlocs(data)
Set alternate location indicators. Alternate location indicators can be used in atom selections, e.g. ‘altloc A B’, ‘altloc _’.

setAnisous(data)
Set anisotropic temperature factors.

setAnistds(data)
Set standard deviations for anisotropic temperature factors.

setBetas(data)
Set β-values (or temperature factors). β-values can be used in atom selections, e.g. ‘beta 555.55’, ‘beta 0 to 500’, ‘beta 0:500’, ‘beta < 500’.

setCharges(data)
Set partial charges. Partial charges can be used in atom selections, e.g. ‘charge 1’, ‘abs(charge) == 1’, ‘charge < 0’.

setChids(data)
Set chain identifiers. Chain identifiers can be used in atom selections, e.g. ‘chain A’, ‘chid A B C’, ‘chain _’. Note that chid is a synonym for chain.

setCoords(coords)
Set coordinates in the active coordinate set.

setData(label, data)
Update data associated with label.

    Raises AttributeError when label is not in use or read-only

setElements(data)
Set element symbols. Element symbols can be used in atom selections, e.g. ‘element C O N’.

setFlags(label, value)
Update flag associated with label.

    Raises AttributeError when label is not in use or read-only

setIcodes(data)
Set insertion codes. Insertion codes can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

setMasses(data)
Set masses. Masses can be used in atom selections, e.g. ‘12 <= mass <= 13.5’.

setNames(data)
Set names. Names can be used in atom selections, e.g. ‘name CA CB’.

setOccupancies(data)
Set occupancy values. Occupancy values can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.
setRadii(data)
Set radii. Radii can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

setResnames(data)
Set residue names. Residue names can be used in atom selections, e.g. ‘resname ALA GLY’.

setResnums(data)
Set residue numbers. Residue numbers can be used in atom selections, e.g. ‘resnum 1 2 3’, ‘resnum 120A 120B’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’. Note that resid is a synonym for resnum.

setSecstrs(data)
Set secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that secstr is a synonym for secondary.

setSegname(segname)
Set segment name.

setSegnames(data)
Set segment names. Segment names can be used in atom selections, e.g. ‘segment PRO’, ‘segname PRO’. Note that segname is a synonym for segment.

setSerials(data)
Set serial numbers (from file). Serial numbers can be used in atom selections, e.g. ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’.

setTypes(data)
Set types. Types can be used in atom selections, e.g. ‘type CT1 CT2 CT3’.

### 3.1.19 Atom Selections

This module defines a class for selecting subsets of atoms. You can read this page in interactive sessions using help(select).

ProDy offers a fast and powerful atom selection class, Select (page 85). Selection features, grammar, and keywords are similar to those of VMD. Small differences, that is described below, should not affect most practical uses of atom selections. With added flexibility of Python, ProDy selection engine can also be used to identify intermolecular contacts. You may see this and other usage examples in Intermolecular Contacts\(^\text{148}\) and Operations on Selections\(^\text{149}\).

First, we import everything from ProDy and parse a protein-DNA-ligand complex structure:

```python
In [1]: from prody import *
In [2]: p = parsePDB('3mht')
```

parsePDB() (page 162) returns AtomGroup (page 37) instances, p in this case, that stores all atomic data in the file. We can count different types of atoms using Atom Flags (page 56) and numAtoms() (page 42) method as follows:

```python
In [3]: p.numAtoms('protein')
Out[3]: 2606
In [4]: p.numAtoms('nucleic')
Out[4]: 509
In [5]: p.numAtoms('hetero')
```

\(^\text{148}\)http://prody.csb.pitt.edu/tutorials/structure_analysis/contacts.html#contacts  
\(^\text{149}\)http://prody.csb.pitt.edu/tutorials/prody_tutorial/selection.html#selection-operations
Last two counts suggest that ligand has 26 atoms, i.e. number of hetero atoms less the number of water atoms.

**Atom flags**

We select subset of atoms by using `AtomGroup.select()` (page 42) method. All `Atom Flags` (page 56) can be input arguments to this methods as follows:

```plaintext
In [7]: p.select('protein')
Out[7]: <Selection: 'protein' from 3mht (2606 atoms)>
```

```plaintext
In [8]: p.select('water')
Out[8]: <Selection: 'water' from 3mht (70 atoms)>
```

This operation returns `Selection` (page 86) instances, which can be an input to functions that accepts an `atoms` argument.

**Logical operators**

Flags can be combined using `and` and `or` operators:

```plaintext
In [9]: p.select('protein and water')
```

'protein and water' did not result in selection of protein and water atoms. This is because, no atom is flagged as a protein and a water atom at the same time.

**Note:** Interpreting selection strings

You may think as if a selection string, such as 'protein and water', is evaluated on a per atom basis and an atom is selected if it satisfies the given criterion. To select both water and protein atoms, 'or' logical operator should be used instead. A protein or a water atom would satisfy 'protein or water' criterion.

```plaintext
In [10]: p.select('protein or water')
Out[10]: <Selection: 'protein or water' from 3mht (2676 atoms)>
```

We can also use 'not' operator to negate an atom flag. For example, the following selection will only select ligand atoms:

```plaintext
In [11]: p.select('not water and hetero')
Out[11]: <Selection: 'not water and hetero' from 3mht (26 atoms)>
```

If you omit the 'and' operator, you will get the same result:

```plaintext
In [12]: p.select('not water hetero')
Out[12]: <Selection: 'not water hetero' from 3mht (26 atoms)>
```

**Note:** Default operator between two flags, or other selection tokens that will be discussed later, is 'and'. For example, 'not water hetero' is equivalent to 'not water and hetero'.
We can select C$_\alpha$ atoms of acidic residues by omitting the default logical operator as follows:

In [13]: sel = p.select('acidic calpha')

In [14]: sel
Out[14]: <Selection: ‘acidic calpha’ from 3mht (39 atoms)>

In [15]: set(sel.getResnames())
Out[15]: {'ASP', 'GLU'}

Quick selections

For simple selections, such as shown above, following may be preferable over the select() (page 42) method:

In [16]: p.acidic_calpha
Out[16]: <Selection: ‘acidic calpha’ from 3mht (39 atoms)>

The result is the same as using p.select('acidic calpha'). Underscore, _, is considered as a white-space. The limitation of this approach is that special characters cannot be used.

Atom data fields

In addition to Atom Flags (page 56), Atom Data Fields (page 54) can be used in atom selections when combined with some values. For example, we can select C$_\alpha$ and C$_\beta$ atoms of alanine residues as follows:

In [17]: p.select('resname ALA name CA CB')
Out[17]: <Selection: ‘resname ALA name CA CB’ from 3mht (32 atoms)>

Note that we omitted the default ‘and’ operator.

Note: Whitespace or empty string can be specified using an ‘_’. Atoms with string data fields empty, such as those with no a chain identifiers or alternate location identifiers, can be selected using an underscore.

In [18]: p.select('chain _')  # chain identifiers of all atoms are specified in 3mht
In [19]: p.select('altloc _')  # altloc identifiers for all atoms are empty
Out[19]: <Selection: ‘altloc _’ from 3mht (3211 atoms)>

Numeric data fields can also be used to make selections:

In [20]: p.select('ca resnum 1 2 3 4')
Out[20]: <Selection: ‘ca resnum 1 2 3 4’ from 3mht (4 atoms)>

A special case for residues is having insertion codes. Residue numbers and insertion codes can be specified together as follows:

- ‘resnum 5’ selects residue 5 (all insertion codes)
- ‘resnum 5A’ selects residue 5 with insertion code A
- ‘resnum 5_’ selects residue 5 with no insertion code
Number ranges

A range of numbers using 'to' or Python style slicing with ':':

In [21]: p.select('ca resnum 1to4')
Out[21]: <Selection: 'ca resnum 1to4' from 3mht (4 atoms)>

In [22]: p.select('ca resnum 1:4')
Out[22]: <Selection: 'ca resnum 1:4' from 3mht (3 atoms)>

In [23]: p.select('ca resnum 1:4:2')
Out[23]: <Selection: 'ca resnum 1:4:2' from 3mht (2 atoms)>

Note: Number ranges specify continuous intervals:

- 'to' is all inclusive, e.g. 'resnum 1 to 4' means '1 <= resnum <= 4'
- ':' is left inclusive, e.g. 'resnum 1:4' means '1 <= resnum < 4'

Consecutive use of ':', however, specifies a discrete range of numbers, e.g. 'resnum 1:4:2' means 'resnum 1 3'

Special characters

Following characters can be specified when using Atom Data Fields (page 54) for atom selections:

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
~@#$$.;_;',

For example, "name C' N' O~ C$ C#" is a valid selection string.

Note: Special characters (~!@#$%^&*(-_=+[]\;,<>./?()'") must be escaped using grave accent characters (``).

Negative numbers

Negative numbers and number ranges must also be escaped using grave accent characters, since negative sign '-' is considered a special character unless it indicates subtraction operation (see below).

In [24]: p.select('x '-25 to 25''
Out[24]: <Selection: 'x '-25 to 25'' from 3mht (1941 atoms)>

In [25]: p.select('x '-22.542''
Out[25]: <Selection: 'x '-22.542'' from 3mht (1 atoms)>

Omitting the grave accent character will cause a SelectionError (page 85).

Regular expressions

Finally, you can specify regular expressions to select atoms based on data fields with type string. Following will select residues whose names start with capital letter A

3.1. Atomic Data
In [26]: sel = p.select('resname “A.*”')

In [27]: set(sel.getResnames())
Out[27]: {'ALA', 'ARG', 'ASN', 'ASP'}

Note: Regular expressions can be specified using double quotes, "...". For more information on regular expressions see re\textsuperscript{150}.

Numerical comparisons

Atom Data Fields (page 54) with numeric types can be used as operands in numerical comparisons:

In [28]: p.select(‘x < 0’)  
Out[28]: <Selection: ‘x < 0’ from 3mht (3095 atoms)>

In [29]: p.select(‘occupancy = 1’)  
Out[29]: <Selection: ‘occupancy = 1’ from 3mht (3211 atoms)>

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal</td>
</tr>
<tr>
<td>==</td>
<td>equal</td>
</tr>
<tr>
<td>!=</td>
<td>not equal</td>
</tr>
</tbody>
</table>

It is also possible to chain comparison statements as follows:

In [30]: p.select(‘-10 <= x < 0’)  
Out[30]: <Selection: ‘-10 <= x < 0’ from 3mht (557 atoms)>

This would be the same as the following selection:

In [31]: p.select(‘-10 <= x and x < 0’) == p.select(‘-10 <= x < 0’)  
Out[31]: True

Furthermore, numerical comparisons may involve the following operations:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x**y</td>
<td>x to the power y</td>
</tr>
<tr>
<td>x^y</td>
<td>x to the power y</td>
</tr>
<tr>
<td>x*y</td>
<td>x times y</td>
</tr>
<tr>
<td>x/y</td>
<td>x divided by y</td>
</tr>
<tr>
<td>x//y</td>
<td>x divided by y (floor division)</td>
</tr>
<tr>
<td>x%y</td>
<td>x modulo y</td>
</tr>
<tr>
<td>x+y</td>
<td>x plus y</td>
</tr>
<tr>
<td>x-y</td>
<td>x minus y</td>
</tr>
</tbody>
</table>

These operations must be used with a numerical comparison, e.g.

In [32]: p.select(‘x ** 2 < 10’)  
Out[32]: <Selection: ‘x ** 2 < 10’ from 3mht (238 atoms)>

\textsuperscript{150}\url{http://docs.python.org/library/re.html#re}

3.1. Atomic Data 82
In [33]: p.select('x ** 2 ** 2 < 10')
Out[33]: <Selection: 'x ** 2 ** 2 < 10' from 3mht (134 atoms)>

Finally, following functions can be used in numerical comparisons:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs(x)</td>
<td>absolute value of x</td>
</tr>
<tr>
<td>acos(x)</td>
<td>arccos of x</td>
</tr>
<tr>
<td>asin(x)</td>
<td>arcsin of x</td>
</tr>
<tr>
<td>atan(x)</td>
<td>arctan of x</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>smallest integer not less than x</td>
</tr>
<tr>
<td>cos(x)</td>
<td>cosine of x</td>
</tr>
<tr>
<td>cosh(x)</td>
<td>hyperbolic cosine of x</td>
</tr>
<tr>
<td>floor(x)</td>
<td>largest integer not greater than x</td>
</tr>
<tr>
<td>exp(x)</td>
<td>e to the power x</td>
</tr>
<tr>
<td>log(x)</td>
<td>natural logarithm of x</td>
</tr>
<tr>
<td>log10(x)</td>
<td>base 10 logarithm of x</td>
</tr>
<tr>
<td>sin(x)</td>
<td>sine of x</td>
</tr>
<tr>
<td>sinh(x)</td>
<td>hyperbolic sine of x</td>
</tr>
<tr>
<td>sq(x)</td>
<td>square of x</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>square-root of x</td>
</tr>
<tr>
<td>tan(x)</td>
<td>tangent of x</td>
</tr>
<tr>
<td>tanh(x)</td>
<td>hyperbolic tangent of x</td>
</tr>
</tbody>
</table>

In [34]: p.select('sqrt(sq(x) + sq(y) + sq(z)) < 100')  # within 100 Å of origin
Out[34]: <Selection: 'sqrt(sq(x) + sq(y) + sq(z)) < 100' from 3mht (1975 atoms)>

**Distance based selections**

Atoms within a user specified distance (Å) from a set of user specified atoms can be selected using 'within . of . ' keyword, e.g. 'within 5 of water' selects atoms that are within 5 Å of water molecules. This setting will results selecting water atoms as well.

User can avoid selecting specified atoms using exwithin . of .. setting, e.g. 'exwithin 5 of water' will not select water molecules and is equivalent to 'within 5 of water and not water'

In [35]: p.select('exwithin 5 of water') == p.select('not water within 5 of water')
Out[35]: True

**Sequence selections**

One-letter amino acid sequences can be used to make atom selections. 'sequence SAR' will select SER-ALA-ARG residues in a chain. Note that the selection does not consider connectivity within a chain. Regular expressions can also be used to make selections: 'sequence "MI.*KQ"' will select MET-ILE-(XXX)n-ASP-LYS-GLN pattern, if present.

In [36]: sel = p.select('ca sequence "MI.*DKQ"')

In [37]: sel
Out[37]: <Selection: 'ca sequence "MI.*DKQ"' from 3mht (8 atoms)>

In [38]: sel.getResnames()
Out[38]:

3.1. Atomic Data
Expanding selections

A selection can be expanded to include the atoms in the same residue, chain, or segment using same .. as .. setting, e.g. ‘same residue as exwithin 4 of water’ will select residues that have at least an atom within 4 Å of any water molecule.

In [39]: p.select('same residue as exwithin 4 of water')
Out[39]: <Selection: 'same residue as...thin 4 of water' from 3mht (1554 atoms)>

Additionally, a selection may be expanded to the immediately bonded atoms using bonded [n] to ... setting, e.g. bonded 1 to calpha will select atoms bonded to Ca atoms. For this setting to work, bonds must be set by the user using the AtomGroup.setBonds() (page 42) method. It is also possible to select bonded atoms by excluding the originating atoms using exbonded [n] to ... setting. Number ’[n]’ indicates number of bonds to consider from the originating selection and defaults to 1.

Selection macros

ProDy allows you to define a macro for any valid selection string. Below functions are for manipulating selection macros:

- defSelectionMacro() (page 85)
- delSelectionMacro() (page 86)
- getSelectionMacro() (page 86)
- isSelectionMacro() (page 86)

In [40]: defSelectionMacro('alanine', 'resname ALA')

In [41]: p.select('alanine') == p.select('resname ALA')
Out[41]: True

You can also use this macro as follows:

In [42]: p.alanine
Out[42]: <Selection: 'alanine' from 3mht (80 atoms)>

Macros are stored in ProDy configuration file permanently. You can delete them if you wish as follows:

In [43]: delSelectionMacro('alanine')

Keyword arguments

select() (page 85) method also accepts keyword arguments that can simplify some selections. Consider the following case where you want to select some protein atoms that are close to its center:

In [44]: protein = p.protein

In [45]: calcCenter(protein).round(2)
Out[45]: array([-21.17, 35.86, 79.97])

In [46]: sel1 = protein.select('sqrt(sq(x--21.17) + sq(y-35.86) + sq(z-79.97)) < 5')
Instead, you could pass a keyword argument and use the keyword in the selection string:

```
In [48]: sel2 = protein.select('within 5 of center', center=calcCenter(protein))
```

```
In [49]: sel2
Out[49]: <Selection: 'index 1452 to 1...33 2935 to 2944' from 3mht (20 atoms)>
```

```
In [50]: sel1 == sel2
Out[50]: True
```

Note that selection string for `sel2` lists indices of atoms. This substitution is performed automatically to ensure reproducibility of the selection without the keyword `center`.

Keywords cannot be reserved words (see `listReservedWords()` (page 66)) and must be all alphanumeric characters.

```python
exception SelectionError (sel, loc=0, msg='', tkns=None)
    Exception raised when there are errors in the selection string.
```

```python
exception SelectionWarning (sel='', loc=0, msg='', tkns=None)
    A class used for issuing warning messages when potential typos are detected in a selection string. Warnings are issued to `sys.stderr` via ProDy package logger. Use `confProDy()` (page 204) to selection warnings on or off, e.g. `confProDy(selection_warning=False)`.
```

```python
class Select
    Select subsets of atoms based on a selection string. See `select` (page 78) module documentation for selection grammar and examples. This class makes use of `pyparsing` module.
```

```python
def getBoolArray (atoms, selstr, **kwargs)
    Return a boolean array with `True` values for `atoms` matching `selstr`. The length of the boolean `numpy.ndarray` will be equal to the length of `atoms` argument.
```

```python
def getIndices (atoms, selstr, **kwargs)
    Return indices of atoms matching `selstr`. Indices correspond to the order in `atoms` argument. If `atoms` is a subset of atoms, they should not be used for indexing the corresponding `AtomGroup` instance.
```

```python
def select (atoms, selstr, **kwargs)
    Return a `Selection` (page 86) of atoms matching `selstr`, or `None`, if selection string does not match any atoms.
```

Parameters

- `atoms` (`Atomic` (page 44)) – atoms to be evaluated
- `selstr` (`str`) – selection string

Note that, if `atoms` is an `AtomMap` (page 45) instance, an `AtomMap` (page 45) is returned, instead of a a `Selection` (page 86).

```python
defSelectionMacro (name, selstr)
    Define selection macro `selstr` with name `name`. Both `name` and `selstr` must be string. An existing keyword cannot be used as a macro name. If a macro with given `name` exists, it will be overwritten.
```

---

151 http://pyparsing.wikispaces.com
152 http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
153 http://docs.python.org/library/functions.html#str

3.1. Atomic Data
In [1]: defSelectionMacro('cbeta', 'name CB and protein')

defSelectionMacro (name)
    Delete the macro name.

In [1]: delSelectionMacro('cbeta')

defSelectionMacro (name=None)
    Return the definition of the macro name. If name is not given, returns a copy of the selection macros dictionary.

delSelectionMacro (name)
    Delete the macro name.

giveSelectionMacro (name=None)
    Return the definition of the macro name. If name is not given, returns a copy of the selection macros dictionary.

isSelectionMacro (word)
    Return True if word is a user defined selection macro.

3.1.20 Selection

This module defines Selection (page 86) class for handling arbitrary subsets of atom.

class Selection (ag, indices, selstr, acsi=None, **kwargs)
    A class for accessing and manipulating attributes of selection of atoms in an AtomGroup (page 37) instance. Instances can be generated using select() (page 42) method. Following built-in functions are customized for this class:

    • `len()`\(^\textit{154}\) returns the number of selected atoms
    • `iter()`\(^\textit{155}\) yields Atom (page 33) instances

    copy ()
        Return a copy of atoms (and atomic data) in an AtomGroup (page 37) instance.

getACSIndex ()
    Return index of the coordinate set.

getACSLabel ()
    Return active coordinate set label.

getAltlocs ()
    Return a copy of alternate location indicators. Alternate location indicators can be used in atom selections, e.g. 'altloc A B', 'altloc _'.

getAnisous ()
    Return a copy of anisotropic temperature factors.

getAnistds ()
    Return a copy of standard deviations for anisotropic temperature factors.

getAtomGroup ()
    Return associated atom group.

getBetas ()
    Return a copy of β-values (or temperature factors). β-values can be used in atom selections, e.g. 'beta 555.55', 'beta 0 to 500', 'beta 0:500', 'beta < 500'.

getCSLabels ()
    Return coordinate set labels.

\(^{154}\)\url{http://docs.python.org/library/functions.html#len}
\(^{155}\)\url{http://docs.python.org/library/functions.html#iter}
getCharges()  
Return a copy of partial charges. Partial charges can be used in atom selections, e.g. ‘charge 1’, ‘abs(charge) == 1’, ‘charge < 0’.

def getChids()  
Return a copy of chain identifiers. Chain identifiers can be used in atom selections, e.g. ‘chain A’, ‘chid A B C’, ‘chain _’. Note that chid is a synonym for chain.

def getChindices()  
Return a copy of chain indices. Chain indices are assigned to subsets of atoms with distinct pairs of chain identifier and segment name. Chain indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Chain indices can be used in atom selections, e.g. ‘chindex 0’.

def getCoords()  
Return a copy of coordinates from the active coordinate set.

def getCoordsets(indices=None)  
Return coordinate set(s) at given indices, which may be an integer or a list/array of integers.

def getData(label)  
Return a copy of data associated with label, if it is present.

def getDataLabels(which=None)  
Return data labels. For which=’user’, return only labels of user provided data.

def getDataType(label)  
Return type of the data (i.e. data.dtype) associated with label, or None label is not used.

def getElements()  
Return a copy of element symbols. Element symbols can be used in atom selections, e.g. ‘element C O N’.

def getFlagLabels(which=None)  
Return flag labels. For which=’user’, return labels of user or parser (e.g. hetatm) provided flags, for which=’all’ return all possible Atom Flags (page 56) labels in addition to those present in the instance.

def getFlags(label)  
Return a copy of atom flags for given label, or None when flags for label is not set.

def getFragindices()  
Return a copy of fragment indices. Fragment indices are assigned to connected subsets of atoms. Bonds needs to be set using AtomGroup.setBonds() (page 42) method. Fragment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Fragment indices can be used in atom selections, e.g. ‘fragindex 0’, ‘fragment 1’. Note that fragment is a synonym for fragindex.

def getHierView(**kwargs)  
Return a hierarchical view of the atom selection.

def getIcodes()  
Return a copy of insertion codes. Insertion codes can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

def getIndices()  
Return a copy of the indices of atoms.

def getMasses()  
Return a copy of masses. Masses can be used in atom selections, e.g. ‘12 <= mass <= 13.5’.
getNames()  
Return a copy of names. Names can be used in atom selections, e.g. ‘name CA CB’.

getOccupancies()  
Return a copy of occupancy values. Occupancy values can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.

getRadii()  
Return a copy of radii. Radii can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

getResindices()  
Return a copy of residue indices. Residue indices are assigned to subsets of atoms with distinct sequences of residue number, insertion code, chain identifier, and segment name. Residue indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Residue indices can be used in atom selections, e.g. ‘resindex 0’.

getResnames()  
Return a copy of residue names. Residue names can be used in atom selections, e.g. ‘resname ALA GLY’.

getResnums()  
Return a copy of residue numbers. Residue numbers can be used in atom selections, e.g. ‘resnum 1 2 3’, ‘resnum 120A 120B’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’. Note that resid is a synonym for resnum.

getSecstrs()  
Return a copy of secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that secstr is a synonym for secondary.

getSegindices()  
Return a copy of segment indices. Segment indices are assigned to subsets of atoms with distinct segment names. Segment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Segment indices can be used in atom selections, e.g. ‘segindex 0’.

getSegnames()  
Return a copy of segment names. Segment names can be used in atom selections, e.g. ‘segment PROT’, ‘segname PROT’. Note that segname is a synonym for segment.

getSelstr()  
Return selection string that selects this atom subset.

getSerials()  
Return a copy of serial numbers (from file). Serial numbers can be used in atom selections, e.g. ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’.

getTypes()  
Return a copy of types. Types can be used in atom selections, e.g. ‘type CT1 CT2 CT3’.

isDataLabel(label)  
Return True if data associated with label is present.

isFlagLabel(label)  
Return True if flags associated with label are present.

iterAtoms()  
Yield atoms.
iterCoordsets()
Yield copies of coordinate sets.

numAtoms (flag=None)
Return number of atoms, or number of atoms with given flag.

numCoordsets()
Return number of coordinate sets.

select (selstr, **kwargs)
Return atoms matching selstr criteria. See select (page 78) module documentation for details and usage examples.

setACSIndex (index)
Set coordinates at index active.

setAltlocs (data)
Set alternate location indicators. Alternate location indicators can be used in atom selections, e.g. ‘altloc A B’,’altloc _’.

setAnisous (data)
Set anisotropic temperature factors.

setAnistds (data)
Set standard deviations for anisotropic temperature factors.

setBetas (data)
Set β-values (or temperature factors). β-values can be used in atom selections, e.g. ‘beta 555.55’,‘beta 0 to 500’,‘beta 0:500’,‘beta < 500’.

setCharges (data)
Set partial charges. Partial charges can be used in atom selections, e.g. ‘charge 1’,‘abs(charge) == 1’,‘charge < 0’.

setChids (data)
Set chain identifiers. Chain identifiers can be used in atom selections, e.g. ‘chain A’,’chid A B C’,’chain _’. Note that chid is a synonym for chain.

setCoordsets (coords)
Set coordinates in the active coordinate set.

setData (label, data)
Update data associated with label.

   Raises AttributeError when label is not in use or read-only

setElements (data)
Set element symbols. Element symbols can be used in atom selections, e.g. ‘element C O N’.

setFlags (label, value)
Update flag associated with label.

   Raises AttributeError when label is not in use or read-only

setIcodes (data)
Set insertion codes. Insertion codes can be used in atom selections, e.g. ‘icode A’,’icode _’.

setMasses (data)
Set masses. Masses can be used in atom selections, e.g. ‘12 <= mass <= 13.5’.

setName (data)
Set names. Names can be used in atom selections, e.g. ‘name CA CB’.
setOccupancies(data)
Set occupancy values. Occupancy values can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.

setRadii(data)
Set radii. Radii can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

setResnames(data)
Set residue names. Residue names can be used in atom selections, e.g. ‘resname ALA GLY’.

setResnums(data)
Set residue numbers. Residue numbers can be used in atom selections, e.g. ‘resnum 1 2 3’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’. Note that resid is a synonym for resnum.

setSecstrs(data)
Set secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that secstr is a synonym for secondary.

setSegnames(data)
Set segment names. Segment names can be used in atom selections, e.g. ‘segment PROT’, ‘segname PROT’. Note that segname is a synonym for segment.

setSerials(data)
Set serial numbers (from file). Serial numbers can be used in atom selections, e.g. ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’.

setTypes(data)
Set types. Types can be used in atom selections, e.g. ‘type CT1 CT2 CT3’.

update()
Update selection.

3.1.21 Atom Subsets

class AtomSubset(ag, indices, acsi, **kwargs)
A class for manipulating subset of atoms in an AtomGroup (page 37). Derived classes are:
• Selection (page 86)
• Segment (page 74)
• Chain (page 49)
• Residue (page 69)

This class stores a reference to an AtomGroup (page 37) instance, a set of atom indices, and active coordinate set index for the atom group.

copy()
Return a copy of atoms (and atomic data) in an AtomGroup (page 37) instance.

getACSIndex()
Return index of the coordinate set.

getACSLabel()
Return active coordinate set label.

getAltlocs()
Return a copy of alternate location indicators. Alternate location indicators can be used in atom selections, e.g. ‘altloc A B’, ‘altloc _’. 
getAnisous()  
Return a copy of anisotropic temperature factors.

getAnistds()  
Return a copy of standard deviations for anisotropic temperature factors.

getAtomGroup()  
Return associated atom group.

getBetas()  
Return a copy of $\beta$-values (or temperature factors). $\beta$-values can be used in atom selections, e.g. 'beta 555.55', 'beta 0 to 500', 'beta 0:500', 'beta < 500'.

getCSLabels()  
Return coordinate set labels.

getCharges()  
Return a copy of partial charges. Partial charges can be used in atom selections, e.g. 'charge 1', 'abs(charge) == 1', 'charge < 0'.

getChids()  
Return a copy of chain identifiers. Chain identifiers can be used in atom selections, e.g. 'chain A', 'chid A B C', 'chain _'. Note that chid is a synonym for chain.

getChindices()  
Return a copy of chain indices. Chain indices are assigned to subsets of atoms with distinct pairs of chain identifier and segment name. Chain indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup instance. Chain indices can be used in atom selections, e.g. 'chindex 0'.

getCoords()  
Return a copy of coordinates from the active coordinate set.

getCoordsets(indices=None)  
Return coordinate set(s) at given indices, which may be an integer or a list/array of integers.

getDatada(label)  
Return a copy of data associated with label, if it is present.

getDatataLabels(which=None)  
Return data labels. For which='user', return only labels of user provided data.

getDataType(label)  
Return type of the data (i.e. data.dtype) associated with label, or None label is not used.

getElements()  
Return a copy of element symbols. Element symbols can be used in atom selections, e.g. 'element C O N'.

getFlagLabels(which=None)  
Return flag labels. For which='user', return labels of user or parser (e.g. hetatm) provided flags, for which='all' return all possible Atom Flags (page 56) labels in addition to those present in the instance.

getFlags(label)  
Return a copy of atom flags for given label, or None when flags for label is not set.

getFragindices()  
Return a copy of fragment indices. Fragment indices are assigned to connected subsets of atoms. Bonds needs to be set using AtomGroup.setBonds() (page 42) method. Fragment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup.
instance. Fragment indices can be used in atom selections, e.g. ‘fragindex 0’, ‘fragment 1’. Note that fragment is a synonym for fragindex.

getIcodes()  
Return a copy of insertion codes. Insertion codes can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

getIndices()  
Return a copy of the indices of atoms.

getMasses()  
Return a copy of masses. Masses can be used in atom selections, e.g. ‘12 <= mass <= 13.5’.

getNames()  
Return a copy of names. Names can be used in atom selections, e.g. ‘name CA CB’.

getOccupancies()  
Return a copy of occupancy values. Occupancy values can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.

getRadii()  
Return a copy of radii. Radii can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

getResindices()  
Return a copy of residue indices. Residue indices are assigned to subsets of atoms with distinct sequences of residue number, insertion code, chain identifier, and segment name. Residue indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Residue indices can be used in atom selections, e.g. ‘resindex 0’.

getResnames()  
Return a copy of residue names. Residue names can be used in atom selections, e.g. ‘resname ALA GLY’.

getResnums()  
Return a copy of residue numbers. Residue numbers can be used in atom selections, e.g. ‘resnum 1 2 3’, ‘resnum 120A 120B’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’. Note that resid is a synonym for resnum.

getSecstrs()  
Return a copy of secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that secstr is a synonym for secondary.

getSegindices()  
Return a copy of segment indices. Segment indices are assigned to subsets of atoms with distinct segment names. Segment indices start from zero, are incremented by one, and are assigned in the order of appearance in AtomGroup (page 37) instance. Segment indices can be used in atom selections, e.g. ‘segindex 0’.

getSegnames()  
Return a copy of segment names. Segment names can be used in atom selections, e.g. ‘segment PROT’, ‘segname PROT’. Note that segname is a synonym for segment.

getSerials()  
Return a copy of serial numbers (from file). Serial numbers can be used in atom selections, e.g. ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’.

getTypes()  
Return a copy of types. Types can be used in atom selections, e.g. ‘type CT1 CT2 CT3’.

3.1. Atomic Data 92
**isDataLabel** *(label)*
Return `True` if data associated with `label` is present.

**isFlagLabel** *(label)*
Return `True` if flags associated with `label` are present.

**iterAtoms** *( )*
Yield atoms.

**iterCoordsets** *( )*
Yield copies of coordinate sets.

**numAtoms** *(flag=None)*
Return number of atoms, or number of atoms with given `flag`.

**numCoordsets** *( )*
Return number of coordinate sets.

**select** *(selstr, **kwargs)*
Return atoms matching `selstr` criteria. See `select` (page 78) module documentation for details and usage examples.

**setACSIndex** *(index)*
Set coordinates at `index` active.

**setAltlocs** *(data)*
Set alternate location indicators. Alternate location indicators can be used in atom selections, e.g. ‘altloc A B’, ‘altloc _’.

**setAnisous** *(data)*
Set anisotropic temperature factors.

**setAnistds** *(data)*
Set standard deviations for anisotropic temperature factors.

**setBetas** *(data)*
Set β-values (or temperature factors). β-values can be used in atom selections, e.g. ‘beta 555.55’, ‘beta 0 to 500’, ‘beta 0:500’, ‘beta < 500’.

**setCharges** *(data)*
Set partial charges. Partial charges can be used in atom selections, e.g. ‘charge 1’, ‘abs(charge) == 1’, ‘charge < 0’.

**setChids** *(data)*
Set chain identifiers. Chain identifiers can be used in atom selections, e.g. ‘chain A’, ‘chid A B C’, ‘chain _’. Note that `chid` is a synonym for `chain`.

**setCoords** *(coords)*
Set coordinates in the active coordinate set.

**setData** *(label, data)*
Update data associated with `label`.

*Raise AttributeError* when `label` is not in use or read-only

**setElements** *(data)*
Set element symbols. Element symbols can be used in atom selections, e.g. ‘element C O N’.

**setFlags** *(label, value)*
Update flag associated with `label`.

*Raise AttributeError* when `label` is not in use or read-only
setIcodes(data)
Set insertion codes. Insertion codes can be used in atom selections, e.g. ‘icode A’, ‘icode _’.

setMasses(data)
Set masses. Masses can be used in atom selections, e.g. ‘12 <= mass <= 13.5’.

setNames(data)
Set names. Names can be used in atom selections, e.g. ‘name CA CB’.

setOccupancies(data)
Set occupancy values. Occupancy values can be used in atom selections, e.g. ‘occupancy 1’, ‘occupancy > 0’.

setRadii(data)
Set radii. Radii can be used in atom selections, e.g. ‘radii < 1.5’, ‘radii ** 2 < 2.3’.

setResnames(data)
Set residue names. Residue names can be used in atom selections, e.g. ‘resname ALA GLY’.

setResnums(data)
Set residue numbers. Residue numbers can be used in atom selections, e.g. ‘resnum 1 2 3’, ‘resnum 120A 120B’, ‘resnum 10 to 20’, ‘resnum 10:20:2’, ‘resnum < 10’. Note that resid is a synonym for resnum.

setSecstrs(data)
Set secondary structure assignments. Secondary structure assignments can be used in atom selections, e.g. ‘secondary H E’, ‘secstr H E’. Note that secstr is a synonym for secondary.

setSegnames(data)
Set segment names. Segment names can be used in atom selections, e.g. ‘segment PROT’, ‘segname PROT’. Note that segname is a synonym for segment.

setSerials(data)
Set serial numbers (from file). Serial numbers can be used in atom selections, e.g. ‘serial 1 2 3’, ‘serial 1 to 10’, ‘serial 1:10:2’, ‘serial < 10’.

setTypes(data)
Set types. Types can be used in atom selections, e.g. ‘type CT1 CT2 CT3’.

3.2 Database Support

This module contains features for accessing databases containing protein related data.

3.2.1 Pfam

Following functions can be used to search and retrieve Pfam data:

- `fetchPfamMSA()` (page 95) - download MSA files
- `searchPfam()` (page 95) - search families of a protein

http://pfam.sanger.ac.uk/
3.2.2 Pfam Access Functions

This module defines functions for interfacing Pfam database.

`searchPfam(query, search_b=False, skip_a=False, **kwargs)`

Return Pfam search results in a dictionary. Matching Pfam accession as keys will map to evalue, alignment start and end residue positions.

**Parameters**

- **query** *(str)* – UniProt ID, PDB identifier, protein sequence, or a sequence file, sequence queries must not contain without gaps and must be at least 16 characters long
- **search_b** *(bool)* – search Pfam-B families when True
- **skip_a** *(bool)* – do not search Pfam-A families when True
- **ga** *(bool)* – use gathering threshold when True
- **evalue** *(float)* – user specified e-value cutoff, must be smaller than 10.0
- **timeout** *(int)* – timeout for blocking connection attempt in seconds, default is 60

`query` can also be a PDB identifier, e.g. ‘1mkp’ or ‘1mkpA’ with chain identifier. UniProt ID of the specified chain, or the first protein chain will be used for searching the Pfam database.

`fetchPfamMSA(acc, alignment='full', compressed=False, **kwargs)`

Return a path to the downloaded Pfam MSA file.

**Parameters**

- **acc** *(str)* – Pfam ID or Accession Code
- **alignment** – alignment type, one of ‘full’ (default), ‘seed’, ‘ncbi’, ‘metagenomics’, ‘rp15’, ‘rp35’, ‘rp55’, or ‘rp75’ where rp stands for representative proteomes
- **compressed** – gzip the downloaded MSA file, default is False

**Alignment Options**

**Parameters**

- **format** – a Pfam supported MSA file format, one of ‘selex’, (default), ‘stockholm’ or ‘fasta’
- **order** – ordering of sequences, ‘tree’ (default) or ‘alphabetical’
- **inserts** – letter case for inserts, ‘upper’ (default) or ‘lower’
- **gaps** – gap character, one of ‘dashes’ (default), ‘dots’, ‘mixed’ or None for unaligned

**Other Options**

**Parameters**

- **timeout** – timeout for blocking connection attempt in seconds, default is 60
• **outname** – out filename, default is input ‘acc_alignment.format’

• **folder** – output folder, default is ‘.’

## 3.3 Dynamics Analysis

This module defines classes and functions for protein dynamics analysis.

### 3.3.1 Dynamics Models

Following classes are designed for modeling and analysis of protein dynamics:

- **ANM** (page 101) - Anisotropic network model, for coarse-grained NMA
- **GNM** (page 112) - Gaussian network model, for coarse-grained dynamics analysis
- **PCA** (page 120) - Principal component analysis of conformation ensembles
- **EDA** (page 122) - Essential dynamics analysis of dynamics trajectories
- **NMA** (page 117) - Normal mode analysis, for analyzing data from external programs
- **RTB** (page 127) - Rotations and Translation of Blocks method

Usage of these classes are shown in Anisotropic Network Model (ANM)\(^\text{164}\), Gaussian Network Model (GNM)\(^\text{165}\), Ensemble Analysis\(^\text{166}\), and Essential Dynamics Analysis\(^\text{167}\) examples.

Following classes are for analysis of individual modes or subsets of modes:

- **Mode** (page 115) - analyze individual normal/principal/essential modes
- **ModeSet** (page 116) - analyze subset of modes from a dynamics model
- **Vector** (page 115) - analyze modified modes or deformation vectors

### 3.3.2 Customize ENMs

Following classes allow for using structure or distance based, or other custom force constants and cutoff distances in **ANM** (page 101) and **GNM** (page 112) calculations:

- **Gamma** (page 108) - base class for developing property custom force constant calculation methods
- **GammaStructureBased** (page 108) - secondary structure based force constants
- **GammaVariableCutoff** (page 109) - atom type based variable cutoff function

### 3.3.3 Function library

Dynamics of the functions described below accept a *modes* argument (may also appear in different names), which may refer to one or more of the following:

- a dynamics model, **ANM** (page 101), **GNM** (page 112), **NMA** (page 117), **PCA** (page 120), or **EDA** (page 122)
- a **Mode** (page 115) obtained by indexing an NMA model, e.g. `anm[0]`

\(^\text{164}\)http://prody.csb.pitt.edu/tutorials/enm_analysis/anm.html#anm

\(^\text{165}\)http://prody.csb.pitt.edu/tutorials/enm_analysis/gnm.html#gnm

\(^\text{166}\)http://prody.csb.pitt.edu/tutorials/ensemble_analysis/index.html#pca

\(^\text{167}\)http://prody.csb.pitt.edu/tutorials/trajectory_analysis/eda.html#eda
• a `ModeSet` (page 116) obtained by slicing an NMA model, e.g. `anm[0:10]`

Some of these functions may also accept `Vector` instances as `mode` argument. These are noted in function documentations.

### 3.3.4 Analyze models

Following functions are for calculating atomic properties from normal modes:

- `calcCollectivity()` (page 99) - degree of collectivity of a mode
- `calcCovariance()` (page 99) - covariance matrix for given modes
- `calcCrossCorr()` (page 99) - cross-correlations of fluctuations
- `calcFractVariance()` (page 100) - fraction of variance explained by a mode
- `calcPerturbResponse()` (page 100) - response to perturbations in positions
- `calcProjection()` (page 100) - projection of conformations onto modes
- `calcSqFlucts()` (page 100) - square-fluctuations
- `calcTempFactors()` (page 100) - temperature factors fitted to exp. data

### 3.3.5 Compare models

Following functions are for comparing normal modes or dynamics models:

- `calcOverlap()` (page 103) - overlap (correlation) between modes
- `calcCumulOverlap()` (page 103) - cumulative overlap between modes
- `calcSubspaceOverlap()` (page 103) - overlap between normal mode subspaces
- `calcCovOverlap()` (page 103) - covariance overlap between models
- `printOverlapTable()` (page 103) - formatted overlap table printed on screen

### 3.3.6 Generate conformers

Following functions can be used to generate conformers along normal modes:

- `deformAtoms()` (page 128) - deform atoms along a mode
- `sampleModes()` (page 128) - deform along random combination of a set of modes
- `traverseMode()` (page 129) - traverse a mode along both directions

### 3.3.7 Editing models

Following functions can be used to reduce, slice, or extrapolate models:

- `sliceMode()` (page 104) - take a slice of the normal mode
- `extendMode()` (page 104) - extend a coarse-grained mode to all-atoms
- `sliceModel()` (page 104) - take a slice of a model
- `extendModel()` (page 104) - extend a coarse-grained model to all-atoms


- `reduceModel()` (page 105) - reduce a model to a subset of atoms
- `sliceVector()` (page 104) - take a slice of a vector
- `extendVector()` (page 104) - extend a coarse-grained vector to all-atoms

### 3.3.8 Parse/write data

Following functions are parsing or writing normal mode data:

- `parseArray()` (page 105) - numeric arrays, e.g. coordinates, eigenvectors
- `parseModes()` (page 105) - normal modes
- `parseNMD()` (page 119) - normal mode, coordinate, and atomic data for NMWiz
- `parseSparseMatrix()` (page 106) - matrix data in sparse coordinate list format
- `writeArray()` (page 107) - numeric arrays, e.g. coordinates, eigenvectors
- `writeModes()` (page 107) - normal modes
- `writeNMD()` (page 119) - normal mode, coordinate, and atomic data
- `writeOverlapTable()` (page 103) - overlap between modes in a formatted table

### 3.3.9 Save/load models

Dynamics objects can be efficiently saved and loaded in later Python sessions using the following functions:

- `loadModel()` (page 107), `saveModel()` (page 107) - load/save dynamics models
- `loadVector()` (page 107), `saveVector()` (page 107) - load/save modes or vectors

### 3.3.10 Short-hand functions

Following allow for performing some dynamics calculations in one function call:

- `calcANM()` (page 103) - perform ANM calculations
- `calcGNM()` (page 114) - perform GNM calculations

### 3.3.11 Plotting functions

Plotting functions are called by the name of the plotted data/property and are prefixed with “show”. Function documentations refers to the `matplotlib.pyplot` function utilized for actual plotting. Arguments and keyword arguments are passed to the Matplotlib functions.

- `showMode()` (page 124) - mode shape
- `showOverlap()` (page 124) - overlap between modes
- `showSqFlucts()` (page 126) - square-fluctuations
- `showEllipsoid()` (page 126) - depict projection of a normal mode space on another
- `showContactMap()` (page 124) - contact map based on a Kirchhoff matrix
- `showProjection()` (page 124) - projection of conformations onto normal modes

[^168]: http://matplotlib.sourceforge.net/api/pyplot_api.html#module-matplotlib.pyplot
• showOverlapTable() (page 124) - overlaps between two models
• showScaledSqFlucts() (page 126) - square-fluctuations fitted to experimental data
• showNormedSqFlucts() (page 126) - normalized square-fluctuations
• showCrossProjection() (page 125) - project conformations onto modes from different models
• showCrossCorr() (page 124) - cross-correlations between fluctuations in atomic positions
• showCumulOverlap() (page 124) - cumulative overlap of a mode with multiple modes from another model
• showFractVars() (page 124) - fraction of variances
• showCumulFractVars() (page 124) - cumulative fraction of variances
• resetTicks() (page 126) - change ticks in a plot

3.3.12 Heat Mapper support

Following functions can be used to read, write, and plot VMD plugin Heat Mapper\(^\text{169}\) files.

• showHeatmap() (page 114)
• parseHeatmap() (page 114)
• writeHeatmap() (page 114)

3.3.13 Visualize modes

Finally, normal modes can be visualized and animated using VMD plugin Normal Mode Wizard\(^\text{170}\). Following functions allow for running NMWiz from within Python:

• viewNMDinVMD() (page 120) - run VMD and load normal mode data
• pathVMD() (page 119) - get/set path to VMD executable

3.3.14 Analysis Functions

This module defines functions for calculating atomic properties from normal modes.

\textbf{calcCollectivity} (mode, masses=None)

Return collectivity of the mode. This function implements collectivity as defined in equation 5 of [BR95] (page 260). If \texttt{masses} are provided, they will be incorporated in the calculation. Otherwise, atoms are assumed to have uniform masses.

\textbf{Parameters}

• \texttt{mode} (\texttt{Mode} (page 115) or \texttt{Vector} (page 115)) – mode or vector
• \texttt{masses} (\texttt{numpy.ndarray}\(^\text{171}\)) – atomic masses

\textbf{calcCovariance} (modes)

Return covariance matrix calculated for given \texttt{modes}.

\(^{169}\)http://www.ks.uiuc.edu/Research/vmd/plugins/heatmapper/
\(^{170}\)http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz
\(^{171}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

3.3. Dynamics Analysis 99
calcCrossCorr(modes, n_cpu=1)
Return cross-correlations matrix. For a 3-d model, cross-correlations matrix is an NxN matrix, where
N is the number of atoms. Each element of this matrix is the trace of the submatrix corresponding
to a pair of atoms. Covariance matrix may be calculated using all modes or a subset of modes of an
NMA instance. For large systems, calculation of cross-correlations matrix may be time consuming.
Optionally, multiple processors may be employed to perform calculations by passing n_cpu=2 or
more.

calcFractVariance(mode)
Return fraction of variance explained by the mode. Fraction of variance is the ratio of the variance
along a mode to the trace of the covariance matrix of the model.

calcSqFlucts(modes)
Return sum of square-fluctuations for given set of normal modes. Square fluctuations for a single mode
is obtained by multiplying the square of the mode array with the variance (Mode.getVariance() (page
115)) along the mode. For PCA (page 120) and EDA (page 122) models built using coordinate
data in Å, unit of square-fluctuations is Å², for ANM (page 101) and GNM (page 112), on the other hand,
it is arbitrary or relative units.

calcTempFactors(modes, atoms)
Return temperature (β) factors calculated using modes from a ANM (page 101) or GNM (page 112) in-
stance scaled according to the experimental β-factors from atoms.

calcProjection(ensemble, modes, rmsd=True)
Return projection of conformational deviations onto given modes. ensemble coordinates are used to
calculate the deviations that are projected onto modes. For K conformations and M modes, a (K,M)
matrix is returned.

Parameters
• ensemble (Ensemble (page 132), Conformation (page 131), Vector (page 115),
Trajectory (page 181)) – an ensemble, trajectory or a conformation for which
deviation(s) will be projected, or a deformation vector
• modes (Mode (page 115), ModeSet (page 116), NMA (page 117)) – up to three normal
modes
By default root-mean-square deviation (RMSD) along the normal mode is calculated. To calculate the
projection pass rmsd=True. Vector (page 115) instances are accepted as ensemble argument to allow
for projecting a deformation vector onto normal modes.

calcCrossProjection(ensemble, mode1, mode2, scale=None, **kwargs)
Return projection of conformational deviations onto modes from different models.

Parameters
• ensemble (Ensemble (page 132)) – ensemble for which deviations will be projected
• mode1 (Mode (page 115), Vector (page 115)) – normal mode to project conforma-
tions onto
• mode2 (Mode (page 115), Vector (page 115)) – normal mode to project conforma-
tions onto
• scale – scale width of the projection onto mode x or y, best scaling factor will be
calculated and printed on the console, absolute value of scalar makes the with of
two projection same, sign of scalar makes the projections yield a positive correlation

calcPerturbResponse(model, atoms=None, repeats=100)
Return a matrix of profiles from scanning of the response of the structure to random perturbations at
specific atom (or node) positions. The function implements the perturbation response scanning (PRS)
method described in [CA09] (page 260). Rows of the matrix are the average magnitude of the responses obtained by perturbing the atom/node position at that row index, i.e. \( \text{prs\_profile}[i,j] \) will give the response of residue/node \( j \) to perturbations in residue/node \( i \). PRS is performed using the covariance matrix from \( \text{model} \), e.t. ANM (page 101) instance. Each residue/node is perturbed \( \text{repeats} \) times with a random unit force vector. When \( \text{atoms} \) instance is given, PRS profile for residues will be added as an attribute which then can be retrieved as \( \text{atoms}.\text{getData}(\text{\textquoteleft\textquoteleftprs\_profile\textquoteright\textquoteright}) \). \( \text{model} \) and \( \text{atoms} \) must have the same number of atoms. \( \text{atoms} \) must be an \( \text{AtomGroup} \) (page 37) instance.

The RPS matrix can be save as follows:

```python
prs_matrix = calcPerturbationResponse(p38_anm)
writeArray(\textquotesingle\textquotesingleprs\_matrix.txt\textquoteright\textquotesingle, prs_matrix, format='\%8.6f', delimiter=' ')
```

### 3.3.15 Anisotropic Network Model

This module defines a class and a function for anisotropic network model (ANM) calculations.

```python
class ANM(name='Unknown'):
    Class for Anisotropic Network Model (ANM) analysis of proteins ([PD00] (page 260), [ARA01] (page 260)).

    See a usage example in Anisotropic Network Model (ANM)\textsuperscript{172}.

    \text{addEigenpair}(\text{vector, value=None})
    Add eigen vector and eigen value pair(s) to the instance. If eigen value is omitted, it will be set to 1. Inverse eigenvalues are set as variances.

    \text{buildHessian}(\text{coords, cutoff=15.0, gamma=1.0, **kwargs})
    Build Hessian matrix for given coordinate set.

    \text{Parameters}
    \begin{itemize}
    \item \text{coords} (numpy.ndarray\textsuperscript{173}) – a coordinate set or an object with \text{getCoords} method
    \item \text{cutoff} (float\textsuperscript{174}) – cutoff distance (Å) for pairwise interactions, default is 15.0 Å, minimum is 4.0 Å
    \item \text{gamma} (float, Gamma) – spring constant, default is 1.0
    \item \text{sparse} (bool\textsuperscript{175}) – elect to use sparse matrices, default is \text{False}. If Scipy is not found, \text{ImportError} is raised.
    \item \text{kdtree} (bool\textsuperscript{176}) – elect to use KDTree for building Hessian matrix, default is \text{False} since KDTree method is slower
    \end{itemize}

    Instances of Gamma classes and custom functions are accepted as \text{gamma} argument.

    When Scipy is available, user can select to use sparse matrices for efficient usage of memory at the cost of computation speed.

    \text{calcModes}(\text{n\_modes=20, zeros=False, turbo=True})
    Calculate normal modes. This method uses \text{scipy.linalg.eigh()}\textsuperscript{177} function to diagonalize the Hessian matrix. When Scipy is not found, \text{numpy.linalg.eigh()}\textsuperscript{178} is used.
```

\textsuperscript{172}http://prody.csb.pitt.edu/tutorials/enm\_analysis/anm.html#anm
\textsuperscript{173}http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
\textsuperscript{174}http://docs.python.org/library/functions.html#float
\textsuperscript{175}http://docs.python.org/library/functions.html#bool
\textsuperscript{176}http://docs.python.org/library/functions.html#bool
\textsuperscript{177}http://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.eigh.html#scipy.linalg.eigh
\textsuperscript{178}http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.eigh.html#numpy.linalg.eigh

3.3. Dynamics Analysis 101
Parameters

- **n_modes** *(int or None, default is 20)* – number of non-zero eigenvalues/vectors to calculate. If `None` is given, all modes will be calculated.
- **zeros** *(bool, default is False)* – If `True`, modes with zero eigenvalues will be kept.
- **turbo** *(bool, default is True)* – Use a memory intensive, but faster way to calculate modes.

**getArray()**
Return a copy of eigenvectors array.

**getCovariance()**
Return covariance matrix. If covariance matrix is not set or yet calculated, it will be calculated using available modes.

**getCutoff()**
Return cutoff distance.

**getEigvals()**
Return eigenvalues. For **PCA** (page 120) and **EDA** (page 122) models built using coordinate data in Å, unit of eigenvalues is Å². For **ANM** (page 101), **GNM** (page 112), and **RTB** (page 127), on the other hand, eigenvalues are in arbitrary or relative units but they correlate with stiffness of the motion along associated eigenvector.

**getEigvecs()**
Return a copy of eigenvectors array.

**getGamma()**
Return spring constant (or the gamma function or Gamma instance).

**getHessian()**
Return a copy of the Hessian matrix.

**getKirchhoff()**
Return a copy of the Kirchhoff matrix.

**getModel()**
Return self.

**getTitle()**
Return title of the model.

**getVariances()**
Return variances. For **PCA** (page 120) and **EDA** (page 122) models built using coordinate data in Å, unit of variance is Å². For **ANM** (page 101), **GNM** (page 112), and **RTB** (page 127), on the other hand, variance is the inverse of the eigenvalue, so it has arbitrary or relative units.

**is3d()**
Return `True` if model is 3-dimensional.

**numAtoms()**
Return number of atoms.

**numDOF()**
Return number of degrees of freedom.

**numModes()**
Return number of modes in the instance (not necessarily maximum number of possible modes).
**setEigens** (*vectors, values=None*)
Set eigen vectors and eigen values. If eigen values are omitted, they will be set to 1. Inverse eigenvalues are set as variances.

**setHessian** (*hessian*)
Set Hessian matrix. A symmetric matrix is expected, i.e. not a lower- or upper-triangular matrix.

**setTitle** (*title*)
Set title of the model.

**calcANM** (*pdb, selstr='calpha', cutoff=15.0, gamma=1.0, n_modes=20, zeros=False*)
Return an ANM (page 101) instance and atoms used for the calculations. By default only alpha carbons are considered, but selection string helps selecting a subset of it. *pdb* can be *Atomic* (page 44) instance.

### 3.3.16 Comparison Functions

This module defines functions for comparing normal modes from different models.

**calcOverlap** (*rows, cols*)
Return overlap (or correlation) between two sets of modes (*rows* and *cols*). Returns a matrix whose rows correspond to modes passed as *rows* argument, and columns correspond to those passed as *cols* argument. Both rows and columns are normalized prior to calculating overlap.

**calcCumulOverlap** (*modes1, modes2, array=False*)
Return cumulative overlap of modes in *modes2* with those in *modes1*. Returns a number of *modes1* contains a single *Mode* (page 115) or a *Vector* (page 115) instance. If *modes1* contains multiple modes, returns an array. Elements of the array correspond to cumulative overlaps for modes in *modes1* with those in *modes2*. If *array* is True, Return array of cumulative overlaps. Returned array has the shape (*len(modes1)*, *len(modes2)*). Each row corresponds to cumulative overlaps calculated for modes in *modes1* with those in *modes2*. Each value in a row corresponds to cumulative overlap calculated using upto that many number of modes from *modes2*.

**calcSubspaceOverlap** (*modes1, modes2*)
Return subspace overlap between two sets of modes (*modes1* and *modes2*). Also known as the root mean square inner product (RMSIP) of essential subspaces [AA99] (page 260). This function returns a single number.

**calcCovOverlap** (*modelA, modelB*)
Return overlap between covariances of *modelA* and *modelB*. Overlap between covariances are calculated using normal modes (eigenvectors), hence modes in both models must have been calculated. This function implements equation 11 in [BH02] (page 260).

**printOverlapTable** (*rows, cols*)
Print table of overlaps (correlations) between two sets of modes. *rows* and *cols* are sets of normal modes, and correspond to rows and columns of the printed table. This function may be used to take a quick look into mode correspondences between two models.

```python
>>> # Compare top 3 PCs and slowest 3 ANM modes
>>> printOverlapTable(p38_pca[:3], p38_anm[:3])
Overlap Table

<table>
<thead>
<tr>
<th></th>
<th>ANM 1p38</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#1</td>
</tr>
<tr>
<td>PCA</td>
<td>p38 xray</td>
</tr>
<tr>
<td>PCA</td>
<td>p38 xray</td>
</tr>
<tr>
<td>PCA</td>
<td>p38 xray</td>
</tr>
</tbody>
</table>
```

**writeOverlapTable** (*filename, rows, cols*)
Write table of overlaps (correlations) between two sets of modes to a file. *rows* and *cols* are...
sets of normal modes, and correspond to rows and columns of the overlap table. See also `printOverlapTable()` (page 103).

### 3.3.17 NMA Model Editing

This module defines functions for editing normal mode data.

- **extendModel** *(model, nodes, atoms, norm=False)*
  Extend a coarse grained *model* built for *nodes* to *atoms*. *model* may be ANM (page 101), GNM (page 112), PCA (page 120), or NMA (page 117) instance. This function will take part of the normal modes for each node (i.e. Cα atoms) and extend it to all other atoms in the same residue. For each atom in *nodes* argument *atoms* argument must contain a corresponding residue. If *norm* is True, extended modes are normalized.

- **extendMode** *(mode, nodes, atoms, norm=False)*
  Extend a coarse grained normal *mode* built for *nodes* to *atoms*. This function will take part of the normal modes for each node (i.e. Cα atoms) and extend it to all other atoms in the same residue. For each atom in *nodes* argument *atoms* argument must contain a corresponding residue. Extended mode is multiplied by the square root of variance of the mode. If *norm* is True, extended mode is normalized.

- **extendVector** *(vector, nodes, atoms)*
  Extend a coarse grained *vector* for *nodes* to *atoms*. This function will take part of the normal modes for each node (i.e. Cα atoms) and extend it to all other atoms in the same residue. For each atom in *nodes* argument *atoms* argument must contain a corresponding residue.

- **sliceMode** *(mode, atoms, select)*
  Return part of the *mode* for *atoms* matching *select*. This works slightly different from `sliceVector()` (page 104). Mode array (eigenvector) is multiplied by square-root of the variance along the mode. If mode is from an elastic network model, variance is defined as the inverse of the eigenvalue. Note that returned Vector (page 115) instance is not normalized.

  **Parameters**
  - `mode (Mode (page 115))` – mode instance to be sliced
  - `atoms (Atomic (page 44))` – atoms for which *mode* describes a deformation, motion, etc.
  - `select (Selection (page 86), str)` – an atom selection or a selection string

  **Returns** *(Vector (page 115), Selection (page 86))*

- **sliceModel** *(model, atoms, select)*
  Return a part of the *model* for *atoms* matching *select*. Note that normal modes (eigenvectors) are not normalized.

  **Parameters**
  - `mode (NMA (page 117))` – NMA model instance to be sliced
  - `atoms (Atomic (page 44))` – atoms for which the *model* was built
  - `select (Selection (page 86), str)` – an atom selection or a selection string

  **Returns** *(NMA (page 117), Selection (page 86))*

- **sliceVector** *(vector, atoms, select)*
  Return part of the *vector* for *atoms* matching *select*. Note that returned Vector (page 115) instance is not normalized.

  **Parameters**

---

**3.3. Dynamics Analysis**

104
• **vector** (*VectorBase*) – vector instance to be sliced
• **atoms** (*Atomic* (page 44)) – atoms for which *vector* describes a deformation, motion, etc.
• **select** (*Selection* (page 86), str) – an atom selection or a selection string

**Returns** (*Vector* (page 115), *Selection* (page 86))

**reduceModel** (*model, atoms, select*)

Return reduced NMA model. Reduces a *NMA* (page 117) model to a subset of *atoms* matching *select*. This function behaves differently depending on the type of the *model* argument. For *ANM* (page 101) and *GNM* (page 112) or other *NMA* (page 117) models, force constant matrix for system of interest (specified by the *select*) is derived from the force constant matrix for the *model* by assuming that for any given displacement of the system of interest, other atoms move along in such a way as to minimize the potential energy. This is based on the formulation in [KH00] (page 260). For *PCA* (page 120) models, this function simply takes the sub-covariance matrix for selection.

**Parameters**

• **model** (*ANM* (page 101), *GNM* (page 112), or *PCA* (page 120)) – dynamics model
• **atoms** (*Atomic* (page 44)) – atoms that were used to build the model
• **select** (*Selection* (page 86), str) – an atom selection or a selection string

**Returns** (*NMA* (page 117), *Selection* (page 86))

### 3.3.18 Supporting Functions

This module defines input and output functions.

**parseArray** (*filename, delimiter=None, skiprows=0, usecols=None, dtype=<type 'float'>*)

Parse array data from a file.

This function is using `numpy.loadtxt()`\(^{179}\) to parse the file. Each row in the text file must have the same number of values.

**Parameters**

• **filename** (*str or file*) – File or filename to read. If the filename extension is `.gz` or `.bz2`, the file is first decompressed.
• **delimiter** (*str*)\(^{180}\) – The string used to separate values. By default, this is any whitespace.
• **skiprows** (*int*)\(^{181}\) – Skip the first *skiprows* lines, default is 0.
• **usecols** (*list*)\(^{182}\) – Which columns to read, with 0 being the first. For example, `usecols = (1, 4, 5)` will extract the 2nd, 5th and 6th columns. The default, `None`, results in all columns being read.
• **dtype** (*numpy.dtype*)\(^{183}\) – Data-type of the resulting array, default is `float()`\(^{184}\).

\(^{179}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html#numpy.loadtxt

\(^{180}\)http://docs.python.org/library/functions.html#str

\(^{181}\)http://docs.python.org/library/functions.html#int

\(^{182}\)http://docs.python.org/library/functions.html#list

\(^{183}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.dtype.html#numpy.dtype

\(^{184}\)http://docs.python.org/library/functions.html#float
parseModes (normalmodes, eigenvalues=None, nm_delimiter=None, nm_skiprows=0, nm_usecols=None,  
    ev_delimiter=None, ev_skiprows=0, ev_usecols=None, ev_usevalues=None)

Return NMA (page 117) instance with normal modes parsed from normalmodes.

In normal mode file normalmodes, columns must correspond to modes (eigenvectors). Optionally, eigenvalues can be parsed from a separate file. If eigenvalues are not provided, they will all be set to 1.

Parameters

- **normalmodes** *(str or file)* – File or filename that contains normal modes. If the filename extension is .gz or .bz2, the file is first decompressed.

- **eigenvalues** *(str or file)* – Optional, file or filename that contains eigenvalues. If the filename extension is .gz or .bz2, the file is first decompressed.

- **nm_delimiter** *(str)* – The string used to separate values in normalmodes. By default, this is any whitespace.

- **nm_skiprows** *(int)* – Skip the first skiprows lines in normalmodes. Default is 0.

- **nm_usecols** *(list)* – Which columns to read from normalmodes, with 0 being the first. For example, usecols = (1, 4, 5) will extract the 2nd, 5th and 6th columns. The default, None, results in all columns being read.

- **ev_delimiter** *(str)* – The string used to separate values in eigenvalues. By default, this is any whitespace.

- **ev_skiprows** *(int)* – Skip the first skiprows lines in eigenvalues. Default is 0.

- **ev_usecols** *(list)* – Which columns to read from eigenvalues, with 0 being the first. For example, usecols = (1, 4, 5) will extract the 2nd, 5th and 6th columns. The default, None, results in all columns being read.

- **ev_usevalues** *(list)* – Which columns to use after the eigenvalue column is parsed from eigenvalues, with 0 being the first. This can be used if eigenvalues contains more values than the number of modes in normalmodes.

See parseArray() (page 105) for details of parsing arrays from files.

parseSparseMatrix (filename, symmetric=False, delimiter=None, skiprows=0, irow=0, icol=1, first=1)

Parse sparse matrix data from a file.

This function is using parseArray() (page 105) to parse the file. Input must have the following format:

```
1 1 9.958948135375977e+00
1 2 -3.788214445114136e+00
1 3 6.236155629158020e-01
1 4 -7.820609807968140e-01
```

Each row in the text file must have the same number of values.

Parameters

- **filename** *(str or file)* – File or filename to read. If the filename extension is .gz or .bz2, the file is first decompressed.

185\[http://docs.python.org/library/functions.html#str
186\[http://docs.python.org/library/functions.html#list
187\[http://docs.python.org/library/functions.html#str
188\[http://docs.python.org/library/functions.html#list
189\[http://docs.python.org/library/functions.html#list

3.3. Dynamics Analysis 106
• **symmetric** \((\text{bool}^{190})\) – Set True if the file contains triangular part of a symmetric matrix, default is False.
• **delimiter** \((\text{str}^{191})\) – The string used to separate values. By default, this is any whitespace.
• **skiprows** \((\text{int}^{192})\) – Skip the first skiprows lines, default is 0.
• **irow** \((\text{int}^{193})\) – Index of the column in data file corresponding to row indices, default is 0.
• **icol** \((\text{int}^{194})\) – Index of the column in data file corresponding to row indices, default is 0.
• **first** \((\text{int}^{195})\) – First index in the data file (0 or 1), default is 1.

Data-type of the resulting array, default is float().

**writeArray** \((\text{filename, array, format}='\%d', \text{delimiter}=' ')\)
Write 1-d or 2-d array data into a delimited text file.

This function is using numpy.savetxt() to write the file, after making some type and value checks. Default format argument is "\%d". Default delimiter argument is white space, " ".

filename will be returned upon successful writing.

**writeModes** \((\text{filename, modes, format}='\%.18e', \text{delimiter}=' ')\)
Write modes (eigenvectors) into a plain text file with name filename. See also writeArray() (page 107).

**saveModel** \((\text{nma, filename=None, matrices=False, **kwargs})\)
Save nma model data as filename.nma.npz. By default, eigenvalues, eigenvectors, variances, trace of covariance matrix, and name of the model will be saved. If matrices is True, covariance, Hessian or Kirchhoff matrices are saved too, whichever are available. If filename is None, name of the NMA instance will be used as the filename, after " " (white spaces) in the name are replaced with "_" (underscores). Extension may differ based on the type of the NMA model. For ANM models, it is .anm.npz. Upon successful completion of saving, filename is returned. This function makes use of numpy.savez() function.

**loadModel** \((\text{filename})\)
Return NMA instance after loading it from file (filename). This function makes use of numpy.load() function. See also saveModel() (page 107).

**saveVector** \((\text{vector, filename, **kwargs})\)
Save vector data as filename.vec.npz. Upon successful completion of saving, filename is returned. This function makes use of numpy.savez() function.

**loadVector** \((\text{filename})\)
Return Vector (page 115) instance after loading it from filename using numpy.load() function. See also saveVector() (page 107).
3.3.19 Custom Gamma Functions

This module defines customized gamma functions for elastic network model analysis.

class Gamma

Base class for facilitating use of atom type, residue type, or residue property dependent force constants ($\gamma$).

Derived classes:

• GammaStructureBased (page 108)
• GammaVariableCutoff (page 109)

\( \text{gamma} (\text{dist2}, i, j) \)

Return force constant.

For efficiency purposes square of the distance between interacting atom/residue (node) pairs is passed to this function. In addition, node indices are passed.

class GammaStructureBased (atoms, gamma=1.0, helix=6.0, sheet=6.0, connected=10.0)

Facilitate setting the spring constant based on the secondary structure and connectivity of the residues.

A recent systematic study [LT10] (page 260) of a large set of NMR-structures analyzed using a method based on entropy maximization showed that taking into consideration properties such as sequential separation between contacting residues and the secondary structure types of the interacting residues provides refinement in the ENM description of proteins.

This class determines pairs of connected residues or pairs of proximal residues in a helix or a sheet, and assigns them a larger user defined spring constant value.

DSSP single letter abbreviations are recognized:

• H: $\alpha$-helix
• G: 3-10-helix
• I: $\pi$-helix
• E: extended part of a sheet

\textit{helix:} Applies to residue (or C$\alpha$ atom) pairs that are in the same helical segment, at most 7 Å apart, and separated by at most 3 (3-10-helix), 4 ($\alpha$-helix), or 5 ($\pi$-helix) residues.

\textit{sheet:} Applies to C$\alpha$ atom pairs that are in different $\beta$-strands and at most 6 Å apart.

\textit{connected:} Applies to C$\alpha$ atoms that are at most 4 Å apart.

Note that this class does not take into account insertion codes.

Example:

Let’s parse coordinates and header data from a PDB file, and then assign secondary structure to the atoms.

\texttt{In [1]: from prody import *}

\texttt{In [2]: ubi, header = parsePDB('laar', chain='A', subset='calpha', header=True)}

\texttt{In [3]: assignSecstr(header, ubi)}

\texttt{Out[3]: <AtomGroup: laar_A_ca (76 atoms)>>}
In the above we parsed only the atoms needed for this calculation, i.e. Cα atoms from chain A.
We build the Hessian matrix using structure based force constants as follows;

```
In [4]: gamma = GammaStructureBased(ubi)
In [5]: anm = ANM('')
In [6]: anm.buildHessian(ubi, gamma=gamma)
```

We can obtain the force constants assigned to residue pairs from the Kirchhoff matrix as follows:

```
In [7]: k = anm.getKirchhoff()
In [8]: k[0,1]  # a pair of connected residues
Out[8]: -10.0
In [9]: k[0,16]  # a pair of residues from a sheet
Out[9]: -6.0
```

Setup the parameters.

**Parameters**

- **atoms** *(Atomic (page 44)) – A set of atoms with chain identifiers, residue numbers, and secondary structure assignments are set.*
- **gamma** *(float)* – Force constant in arbitrary units. Default is 1.0.
- **helix** *(float)* – Force constant factor for residues hydrogen bonded in α-helices, 3,10-helices, and π-helices. Default is 6.0, i.e. 6.0 *gamma.
- **sheet** *(float)* – Force constant factor for residue pairs forming a hydrogen bond in a β-sheet. Default is 6.0, i.e. 6.0 *gamma.
- **connected** *(float)* – Force constant factor for residue pairs that are connected. Default is 10.0, i.e. 10.0 *gamma.

```
gamma (dist2, i, j)
Return force constant.
```

```
getChids ()
Return a copy of chain identifiers.
```

```
getResnums ()
Return a copy of residue numbers.
```

```
getSecstrs ()
Return a copy of secondary structure assignments.
```

```
class GammaVariableCutoff (identifiers, gamma=1.0, default_radius=7.5, **kwargs)
Facilitate setting the cutoff distance based on user defined atom/residue (node) radii.
Half of the cutoff distance can be thought of as the radius of a node. This class enables setting different radii for different node types.
```

**Example:**

---

202 http://docs.python.org/library/functions.html#float
203 http://docs.python.org/library/functions.html#float
204 http://docs.python.org/library/functions.html#float
205 http://docs.python.org/library/functions.html#float
ProDy Documentation, Release 1.5.1

Let’s think of a protein-DNA complex for which we want to use different radius for different residue
types. Let’s say, for protein Cα atoms we want to set the radius to 7.5 Å, and for nucleic acid phosphate
atoms to 10 Å. We use the HhaI-DNA complex structure 1mht.
In [1]: hhai = parsePDB(’1mht’)
In [2]: ca_p = hhai.select(’(protein and name CA) or (nucleic and name P)’)
In [3]: ca_p.getNames()
Out[3]:
array([’P’, ’P’, ’P’, ’P’, ’P’, ’P’,
’P’, ’P’, ’P’, ’P’, ’P’, ’P’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’],
dtype=’|S6’)

’P’, ’P’, ’P’, ’P’, ’P’, ’P’, ’P’,
’P’, ’P’, ’P’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,
’CA’, ’CA’, ’CA’, ’CA’, ’CA’, ’CA’,

We set the radii of atoms:
In [4]: varcutoff = GammaVariableCutoff(ca_p.getNames(), gamma=1,
...:
default_radius=7.5, debug=False, P=10)
...:
In [5]: varcutoff.getRadii()
Out[5]:
array([ 10. , 10. , 10. , 10. ,
10. , 10. , 10. , 10. ,
10. , 10. , 10. , 10. ,
7.5,
7.5,
7.5,
7.5,
7.5,
7.5,
7.5,
7.5,
7.5,
7.5,
7.5,
7.5,
7.5,
7.5,
7.5,
7.5,

3.3. Dynamics Analysis

10. ,
10. ,
7.5,
7.5,
7.5,
7.5,
7.5,

10. ,
10. ,
7.5,
7.5,
7.5,
7.5,
7.5,

10. ,
10. ,
7.5,
7.5,
7.5,
7.5,
7.5,

10. ,
10. ,
7.5,
7.5,
7.5,
7.5,
7.5,

10. ,
10. ,
7.5,
7.5,
7.5,
7.5,
7.5,

110


The above shows that for phosphate atoms radii is set to 10 Å, because we passed the `P=10` argument. As for Cα atoms, the default 7.5 Å is set as the radius (`default_radius=7.5`). You can also try this with `debug=True` argument to print debugging information on the screen.

We build ANM (page 101) Hessian matrix as follows:

```
In [6]: annm = ANM('HhaI-DNA')
```

```
In [7]: annm.buildHessian(ca_p, gamma=varcutoff, cutoff=20)
```

Note that we passed `cutoff=20.0` to the `ANM.buildHessian()` (page 101) method. This is equal to the largest possible cutoff distance (between two phosphate atoms) for this system, and ensures that all of the potential interactions are evaluated.

For pairs of atoms for which the actual distance is larger than the effective cutoff, the `GammaVariableCutoff.gamma()` (page 112) method returns 0. This annuls the interaction between those atom pairs.

Set the radii of atoms.

**Parameters**

- `identifiers` (list or `numpy.ndarray`) – List of atom names or types, or residue names.

---

• **gamma** *(float)* – Uniform force constant value. Default is 1.0.

• **default_radius** *(float)* – Default radius for atoms whose radii is not set as a keyword argument. Default is 7.5

Keywords in keyword arguments must match those in `atom_identifiers`. Values of keyword arguments must be float.

    gamma(dist2, i, j)
    Return force constant.

    getGamma()
    Return the uniform force constant value.

    getRadii()
    Return a copy of radii array.

### 3.3.20 Gaussian Network Model

This module defines a class and a function for Gaussian network model (GNM) calculations.

```python
class GNM(name='Unknown')
A class for Gaussian Network Model (GNM) analysis of proteins ([IB97] (page 260), [TH97] (page 260)).

See example Gaussian Network Model (GNM).
```

    addEigenpair(vector, value=None)
    Add eigen vector and eigen value pair(s) to the instance. If eigen value is omitted, it will be set to 1. Inverse eigenvalues are set as variances.

    buildKirchhoff(coords, cutoff=10.0, gamma=1.0, **kwargs)
    Build Kirchhoff matrix for given coordinate set.

**Parameters**

• **coords** *(numpy.ndarray)* or *Atomic* – a coordinate set or an object with `getCoords` method

• **cutoff** *(float)* – cutoff distance (Å) for pairwise interactions default is 10.0 Å, minimum is 4.0 Å

• **gamma** *(float)* – spring constant, default is 1.0

• **sparse** *(bool)* – elect to use sparse matrices, default is False. If Scipy is not found, `ImportError` is raised.

• **kdtree** *(bool)* – elect to use KDTree for building Kirchhoff matrix faster, default is True

Instances of `Gamma` classes and custom functions are accepted as `gamma` argument.

When Scipy is available, user can select to use sparse matrices for efficient usage of memory at the cost of computation speed.

---

207 http://docs.python.org/library/functions.html#float
208 http://docs.python.org/library/functions.html#float
209 http://prody.csb.pitt.edu/tutorials/enm_analysis/gnm.html#gnm
210 http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
211 http://docs.python.org/library/functions.html#float
212 http://docs.python.org/library/functions.html#float
213 http://docs.python.org/library/functions.html#bool
214 http://docs.python.org/library/functions.html#bool
**calcModes** *(n_modes=20, zeros=False, turbo=True)*
Calculate normal modes. This method uses `scipy.linalg.eigh()`\(^{215}\) function to diagonalize the Kirchhoff matrix. When Scipy is not found, `numpy.linalg.eigh()`\(^{216}\) is used.

### Parameters

- **n_modes** *(int or None, default is 20)* – number of non-zero eigenvalues/vectors to calculate. If None is given, all modes will be calculated.
- **zeros** *(bool, default is False)* – If True, modes with zero eigenvalues will be kept.
- **turbo** *(bool, default is True)* – Use a memory intensive, but faster way to calculate modes.

**getArray()**
Return a copy of eigenvectors array.

**getCovariance()**
Return covariance matrix. If covariance matrix is not set or yet calculated, it will be calculated using available modes.

**getCutoff()**
Return cutoff distance.

**getEigvals()**
Return eigenvalues. For PCA (page 120) and EDA (page 122) models built using coordinate data in Å, unit of eigenvalues is Å\(^2\). For ANM (page 101), GNM (page 112), and RTB (page 127), on the other hand, eigenvalues are in arbitrary or relative units but they correlate with stiffness of the motion along associated eigenvector.

**getEigvecs()**
Return a copy of eigenvectors array.

**getGamma()**
Return spring constant (or the gamma function or Gamma instance).

**getKirchhoff()**
Return a copy of the Kirchhoff matrix.

**getModel()**
Return self.

**getTitle()**
Return title of the model.

**getVariances()**
Return variances. For PCA (page 120) and EDA (page 122) models built using coordinate data in Å, unit of variance is Å\(^2\). For ANM (page 101), GNM (page 112), and RTB (page 127), on the other hand, variance is the inverse of the eigenvalue, so it has arbitrary or relative units.

**is3d()**
Return True if model is 3-dimensional.

**numAtoms()**
Return number of atoms.

**numDOF()**
Return number of degrees of freedom.

**numModes()**
Return number of modes in the instance (not necessarily maximum number of possible modes).

\(^{215}\)http://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.eigh.html#scipy.linalg.eigh

\(^{216}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.eigh.html#numpy.linalg.eigh
setEigens \( (vectors, values=None) \)
Set eigen vectors and eigen values. If eigen values are omitted, they will be set to 1. Inverse eigenvalues are set as variances.

setKirchhoff \((kirchhoff)\)
Set Kirchhoff matrix.

setTitle \((title)\)
Set title of the model.

calcGNM \((pdb, selstr='alpha', cutoff=15.0, gamma=1.0, n_modes=20, zeros=False)\)
Return a GNM (page 112) instance and atoms used for the calculations. By default only alpha carbons are considered, but selection string helps selecting a subset of it. pdb can be Atomic (page 44) instance.

### 3.3.21 Heatmapper Functions

This module defines functions for supporting VMD plugin Heat Mapper\(^{217}\) format files.

parseHeatmap \((heatmap, **kwargs)\)
Return a two dimensional array and a dictionary with information parsed from heatmap, which may be an input stream or an .hm file in VMD plugin Heat Mapper format.

writeHeatmap \((filename, heatmap, **kwargs)\)
Return filename that contains heatmap in Heat Mapper .hm file (extension is automatically added when not found). filename may also be an output stream.

Parameters

- title \((str)\)^{218} – title of the heatmap
- xlabel \((str)\)^{219} – x-axis lab, default is ‘unknown’
- ylabel \((str)\)^{220} – y-axis lab, default is ‘unknown’
- xorigin \((float)\)^{221} – x-axis origin, default is 0
- xstep \((float)\)^{222} – x-axis step, default is 1
- min \((float)\)^{223} – minimum value, default is minimum in heatmap
- max \((float)\)^{224} – maximum value, default is maximum in heatmap
- format \((str)\)^{225} – number format, default is ‘%f’

Other keyword arguments that are arrays with length equal to the y-axis (second dimension of heatmap) will be considered as numbering.

showHeatmap \((heatmap, *args, **kwargs)\)
Show heatmap, which can be an two dimensional array or a Heat Mapper .hm file.

Heatmap is plotted using imshow() \(^{226}\) function. Default values passed to this function are interpolation=’nearest’, aspect=’auto’, and origin=’lower’.

---

\(^{217}\)http://www.ks.uiuc.edu/Research/vmd/plugins/heatmapper/

\(^{218}\)http://docs.python.org/library/functions.html#str

\(^{219}\)http://docs.python.org/library/functions.html#str

\(^{220}\)http://docs.python.org/library/functions.html#str

\(^{221}\)http://docs.python.org/library/functions.html#float

\(^{222}\)http://docs.python.org/library/functions.html#float

\(^{223}\)http://docs.python.org/library/functions.html#float

\(^{224}\)http://docs.python.org/library/functions.html#float

\(^{225}\)http://docs.python.org/library/functions.html#str

\(^{226}\)http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.imshow
### 3.3.22 Normal Mode

This module defines classes for handling mode data.

**class Mode**(model, index)

A class to provide access to and operations on mode data.

Initialize mode object as part of an NMA model.

**Parameters**

- **model** *(NMA (page 117), GNM (page 112), or PCA (page 120)) – a normal mode analysis instance*
- **index** *(int)* – index of the mode

**getArray()**

Return a copy of the normal mode array (eigenvector).

**getArrayNx3()**

Return a copy of array with shape (N, 3).

**getEigval()**

Return normal mode eigenvalue. For **PCA** (page 120) and **EDA** (page 122) models built using coordinate data in Å, unit of eigenvalues is Å². For **ANM** (page 101) and **GNM** (page 112), on the other hand, eigenvalues are in arbitrary or relative units but they correlate with stiffness of the motion along associated eigenvector.

**getEigvec()**

Return a copy of the normal mode array (eigenvector).

**getIndex()**

Return the index of the mode. Note that mode indices are zero-based.

**getModel()**

Return the model that the mode instance belongs to.

**getTitle()**

A descriptive title for the mode instance.

**getVariance()**

Return variance along the mode. For **PCA** (page 120) and **EDA** (page 122) models built using coordinate data in Å, unit of variance is Å². For **ANM** (page 101) and **GNM** (page 112), on the other hand, variance is the inverse of the eigenvalue, so it has arbitrary or relative units.

**is3d()**

Return True if mode instance is from a 3-dimensional model.

**numAtoms()**

Return number of atoms.

**numDOF()**

Return number of degrees of freedom (three times the number of atoms).

**numModes()**

Return 1.

**class Vector**(array, title='Unknown', is3d=True)

A class to provide operations on a modified mode array. This class holds only mode array (i.e. eigenvector) data, and has no associations with an NMA instance. Scalar multiplication of **Mode** (page 115) instance or addition of two **Mode** (page 115) instances results in a Vector (page 115) instance.

---

227[http://docs.python.org/library/functions.html#int](http://docs.python.org/library/functions.html#int)
Instantiate with a name, an array, and a 3d flag.

getArray()
   Return a copy of array.

getArrayNx3()
   Return a copy of array with shape (N, 3).

getNormed()
   Return mode after normalizing it.

ggetTitle()
   Get the descriptive title for the vector instance.

is3d()
   Return True if vector instance describes a 3-dimensional property, such as a deformation for a set of atoms.

numAtoms()
   Return number of atoms. For a 3-dimensional vector, returns length of the vector divided by 3.

numDOF()
   Return number of degrees of freedom.

numModes()
   Return 1.

setTitle(title)
   Set the descriptive title for the vector instance.

### 3.3.23 Mode Set

This module defines a pointer class for handling subsets of normal modes.

class ModeSet (model, indices)
   A class for providing access to subset of mode data. Instances are obtained by slicing an NMA model (ANM (page 101), GNM (page 112), or PCA (page 120)). ModeSet’s contain a reference to the model and a list of mode indices. Methods common to NMA models are also defined for mode sets.

getArray()
   Return a copy of eigenvectors array.

getEigvals()
   Return eigenvalues. For PCA (page 120) and EDA (page 122) models built using coordinate data in Å, unit of eigenvalues is Å^2. For ANM (page 101) and GNM (page 112), on the other hand, eigenvalues are in arbitrary or relative units but they correlate with stiffness of the motion along associated eigenvector.

getEigvecs()
   Return a copy of eigenvectors array.

getIndices()
   Return indices of modes in the mode set.

getModel()
   Return the model that the modes belongs to.

ggetTitle()
   Return title of the mode set.
getVariances()
Return variances. For PCA (page 120) and EDA (page 122) models built using coordinate data in Å, unit of variance is Å². For ANM (page 101) and GNM (page 112), on the other hand, variance is the inverse of the eigenvalue, so it has arbitrary or relative units.

is3d()
Return True if model is 3-dimensional.

numAtoms()
Return number of atoms.

numDOF()
Return number of degrees of freedom.

numModes()
Return number of modes in the instance (not necessarily maximum number of possible modes).

3.3.24 Normal Mode Analysis

This module defines a class handling normal mode analysis data.

class NMA (title='Unknown')
A class for handling Normal Mode Analysis (NMA) data.

addEigenpair(vector, value=None)
Add eigen vector and eigen value pair(s) to the instance. If eigen value is omitted, it will be set to 1. Inverse eigenvalues are set as variances.

getArray()
Return a copy of eigenvectors array.

g getCovariance()
Return covariance matrix. If covariance matrix is not set or yet calculated, it will be calculated using available modes.

g getEigvals()
Return eigenvalues. For PCA (page 120) and EDA (page 122) models built using coordinate data in Å, unit of eigenvalues is Å². For ANM (page 101), GNM (page 112), and RTB (page 127), on the other hand, eigenvalues are in arbitrary or relative units but they correlate with stiffness of the motion along associated eigenvector.

g getEigvecs()
Return a copy of eigenvectors array.

g getModel()
Return self.

g getTitle()
Return title of the model.

g getVariances()
Return variances. For PCA (page 120) and EDA (page 122) models built using coordinate data in Å, unit of variance is Å². For ANM (page 101), GNM (page 112), and RTB (page 127), on the other hand, variance is the inverse of the eigenvalue, so it has arbitrary or relative units.

is3d()
Return True if model is 3-dimensional.

numAtoms()
Return number of atoms.
numDOF()
Return number of degrees of freedom.

numModes()
Return number of modes in the instance (not necessarily maximum number of possible modes).

setEigens(vectors, values=None)
Set eigen vectors and eigen values. If eigen values are omitted, they will be set to 1. Inverse eigenvalues are set as variances.

setTitle(title)
Set title of the model.

3.3.25 NMD File

This module defines input and output functions for NMD format.

NMD Format

Description

NMD files (extension .nmd) are plain text files that contain at least normal mode and system coordinate data.

NMD files can be visualized using Normal Mode Wizard\(^\text{228}\). ProDy functions writeNMD() (page 119) and parseNMD() (page 119) can be used to read and write NMD files.

Data fields

Data fields in bold face are required. All data arrays and lists must be in a single line and items must be separated by one or more space characters.

coordinates: system coordinates as a list of decimal numbers Coordinate array is the most important line in an NMD file. All mode array lengths must match the length of the coordinate array. Also, number of atoms in the system is deduced from the length of the coordinate array.

coordinates 27.552 4.354 23.629 24.179 4.807 21.907 ...

mode: normal mode array as a list of decimal numbers Optionally, mode index and a scaling factor may be provided in the same line as a mode array. Both of these must precede the mode array. Providing a scaling factor enables relative scaling of the mode arrows and the amplitude of the fluctuations in animations. For NMA, scaling factors may be chosen to be the square-root of the inverse-eigenvalue associated with the mode. Analogously, for PCA data, scaling factor would be the square-root of the eigenvalue.

If a mode line contains numbers preceding the mode array, they are evaluated based on their type. If an integer is encountered, it is considered the mode index. If a decimal number is encountered, it is considered the scaling factor. Scaling factor may be the square-root of the inverse eigenvalue if data is from an elastic network model, or the square-root of the eigenvalue if data is from an essential dynamics (or principal component) analysis.

For example, all of the following lines are valid. The first line contains mode index and scaling factor. Second and third lines contain mode index or scaling factor. Last line contains only the mode array.

\(^{228}\)http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz
mode 1 2.37 0.039 0.009 0.058 0.038 -0.011 0.052 ...
mode 1 0.039 0.009 0.058 0.038 -0.011 0.052 ...
mode 2.37 0.039 0.009 0.058 0.038 -0.011 0.052 ...
mode 0.039 0.009 0.058 0.038 -0.011 0.052 0.043 ...

name: name of the model

The length of all following data fields must be equal to the number of atoms in the system. NMWiz uses such data when writing a temporary PDB files for loading coordinate data into VMD.

atomnames: list of atom names  If not provided, all atom names are set to “CA”.
resnames: list of residue names  If not provided, all residue names are set to “GLY”.
chainids: list of chain identifiers  If not provided, all chain identifiers are set to “A”.
resids: list of residue numbers  If not provided, residue numbers are started from 1 and incremented by one for each atom.
bfactors: list of experimental beta-factors  If not provided, all beta-factors are set to zero. Beta-factors can be used to color the protein representation.

NMD files may contain additional lines. Only lines that start with one of the above field names are evaluated by NMWiz.

Autoload Trick

By adding a special line in an NMD file, file content can be automatically loaded into VMD at startup. The first line calls a NMWiz function to load the file itself (xyzeros.nmd).

nmwiz_load xyzeros.nmd
coordinates 0 0 0 0 0 0 ...
mode 0.039 0.009 0.058 0.038 -0.011 0.052 ...
mode -0.045 -0.096 -0.009 -0.040 -0.076 -0.010 ...
mode 0.007 -0.044 0.080 0.015 -0.037 0.062 ...

In this case, VMD must be started from the command line by typing vmd -e xyzeros.nmd.

parseNMD (filename, type=None)
Return NMA (page 117) and AtomGroup (page 37) instances storing data parsed from filename in .nmd format. Type of NMA (page 117) instance, e.g. PCA (page 120), ANM (page 101), or GNM (page 112) will be determined based on mode data.

writeNMD (filename, modes, atoms)
Return filename that contains modes and atoms data in NMD format described in NMD Format (page 118). .nmd extension is appended to filename, if it does not have an extension.

Note:
1. This function skips modes with zero eigenvalues.
2. If a Vector (page 115) instance is given, it will be normalized before it is written. It’s length before normalization will be written as the scaling factor of the vector.

pathVMD (*path)
Return VMD path, or set it to be a user specified path.

getVMDpath ()
Deprecated for removal in v1.5, use pathVMD () (page 119) instead.
setVMDpath(path)
Deprecated for removal in v1.5, use pathVMD() (page 119) instead.

viewNMDinVMD(filename)
Start VMD in the current Python session and load NMD data.

3.3.26 Principal Component Analysis

This module defines classes for principal component analysis (PCA) and essential dynamics analysis (EDA) calculations.

class PCA(name='Unknown')
A class for Principal Component Analysis (PCA) of conformational ensembles. See examples in Ensemble Analysis.229

addEigenpair(eigenvector, eigenvalue=None)
Add eigenvector and eigenvalue pair(s) to the instance. If eigenvalue is omitted, it will be set to 1. Eigenvalues are set as variances.

buildCovariance(coordsets, **kwargs)
Build a covariance matrix for coordsets using mean coordinates as the reference. coordsets argument may be one of the following:

• Atomic (page 44)
• Ensemble (page 132)
• TrajBase (page 179)
• numpy.ndarray with shape (n_csets, n_atoms, 3)

For ensemble and trajectory objects, update_coords=True argument can be used to set the mean coordinates as the coordinates of the object.

When coordsets is a trajectory object, such as DCDFile (page 175), covariance will be built by superposing frames onto the reference coordinate set (see Frame.superpose() (page 178)). If frames are already aligned, use aligned=True argument to skip this step.

Note: If coordsets is a PDBEnsemble (page 135) instance, coordinates are treated specially. Let's say C_ij is the element of the covariance matrix that corresponds to atoms i and j. This super element is divided by number of coordinate sets (PDB models or structures) in which both of these atoms are observed together.

calcModes(n_modes=20, turbo=True)
Calculate principal (or essential) modes. This method uses scipy.linalg.eigh()231, or numpy.linalg.eigh()232, function to diagonalize the covariance matrix.

Parameters

• n_modes (int233) – number of non-zero eigenvalues/vectors to calculate, default is 20, for None all modes will be calculated

229http://prody.csb.pitt.edu/tutorials/ensemble_analysis/index.html#pca
230http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
231http://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.eigh.html#scipy.linalg.eigh
233http://docs.python.org/library/functions.html#int
- `turbo` *(bool)* – when available, use a memory intensive but faster way to calculate modes, default is True

`getArray()`

Return a copy of eigenvectors array.

`getCovariance()`

Return covariance matrix. If covariance matrix is not set or yet calculated, it will be calculated using available modes.

`getEigvals()`

Return eigenvalues. For **PCA** (page 120) and **EDA** (page 122) models built using coordinate data in Å, unit of eigenvalues is Å². For **ANM** (page 101), **GNM** (page 112), and **RTB** (page 127), on the other hand, eigenvalues are in arbitrary or relative units but they correlate with stiffness of the motion along associated eigenvector.

`getEigvecs()`

Return a copy of eigenvectors array.

`getModel()`

Return self.

`getTitle()`

Return title of the model.

`getVariances()`

Return variances. For **PCA** (page 120) and **EDA** (page 122) models built using coordinate data in Å, unit of variance is Å². For **ANM** (page 101), **GNM** (page 112), and **RTB** (page 127), on the other hand, variance is the inverse of the eigenvalue, so it has arbitrary or relative units.

`is3d()`

Return True if model is 3-dimensional.

`numAtoms()`

Return number of atoms.

`numDOF()`

Return number of degrees of freedom.

`numModes()`

Return number of modes in the instance (not necessarily maximum number of possible modes).

`performSVD(coordsets)`

Calculate principal modes using singular value decomposition (SVD). `coordsets` argument may be a **Atomic** (page 44), **Ensemble** (page 132), or **numpy.ndarray** instance. If `coordsets` is a numpy array, its shape must be `(n_csets, n_atoms, 3)`. Note that coordinate sets must be aligned prior to SVD calculations.

This is a considerably faster way of performing PCA calculations compared to eigenvalue decomposition of covariance matrix, but is an approximate method when heterogeneous datasets are analyzed. Covariance method should be preferred over this one for analysis of ensembles with missing atomic data. See *Calculations* example for comparison of results from SVD and covariance methods.

`setCovariance(covariance)`

Set covariance matrix.

234http://docs.python.org/library/functions.html#bool

235http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

236http://prody.csb.pitt.edu/tutorials/ensemble_analysis/xray_calculations.html#pca-xray-calculations
**setEigens** *(vectors, values=None)*
Set eigen vectors and eigen values. If eigen values are omitted, they will be set to 1. Eigenvalues are set as variances.

**setTitle** *(title)*
Set title of the model.

**class EDA** *(name=’Unknown’)*
A class for Essential Dynamics Analysis (EDA) [AA93] (page 260). See examples in Essential Dynamics Analysis.<sup>237</sup>

**addEigenpair** *(eigenvector, eigenvalue=None)*
Add eigen vector and eigen value pair(s) to the instance. If eigen value is omitted, it will be set to 1. Eigenvalues are set as variances.

**buildCovariance** *(coordsets, **kwargs)*
Build a covariance matrix for coordsets using mean coordinates as the reference. coordsets argument may be one of the following:

- Atomic (page 44)
- Ensemble (page 132)
- TrajBase (page 179)
- numpy.ndarray<sup>238</sup> with shape (n_csets, n_atoms, 3)

For ensemble and trajectory objects, update_coords=True argument can be used to set the mean coordinates as the coordinates of the object.

When coordsets is a trajectory object, such as DCDFile (page 175), covariance will be built by superposing frames onto the reference coordinate set (see Frame.superpose() (page 178)). If frames are already aligned, use aligned=True argument to skip this step.

**Note:** If coordsets is a PDBEnsemble (page 135) instance, coordinates are treated specially. Let’s say C<sub>ij</sub> is the element of the covariance matrix that corresponds to atoms i and j. This super element is divided by number of coordinate sets (PDB models or structures) in which both of these atoms are observed together.

**calcModes** *(n_modes=20, turbo=True)*
Calculate principal (or essential) modes. This method uses scipy.linalg.eigh()<sup>239</sup>, or numpy.linalg.eigh()<sup>240</sup>, function to diagonalize the covariance matrix.

**Parameters**
- **n_modes** *(int<sup>241</sup>) – number of non-zero eigenvalues/vectors to calculate, default is 20, for None all modes will be calculated
- **turbo** *(bool<sup>242</sup>) – when available, use a memory intensive but faster way to calculate modes, default is True

**getArray** *
Return a copy of eigenvectors array.

---
<sup>237</sup>http://prody.csb.pitt.edu/tutorials/trajectory_analysis/eda.html#eda
<sup>238</sup>http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
<sup>239</sup>http://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.eigh.html#scipy.linalg.eigh
<sup>240</sup>http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.eigh.html#numpy.linalg.eigh
<sup>241</sup>http://docs.python.org/library/functions.html#int
<sup>242</sup>http://docs.python.org/library/functions.html#bool

---

3.3. Dynamics Analysis 122
getCovariance()
Return covariance matrix. If covariance matrix is not set or yet calculated, it will be calculated using available modes.

gEigvals()
Return eigenvalues. For PCA (page 120) and EDA (page 122) models built using coordinate data in Å, unit of eigenvalues is Å². For ANM (page 101), GNM (page 112), and RTB (page 127), on the other hand, eigenvalues are in arbitrary or relative units but they correlate with stiffness of the motion along associated eigenvector.

gEigvecs()
Return a copy of eigenvectors array.

getModel()
Return self.

getTitle()
Return title of the model.

getVariances()
Return variances. For PCA (page 120) and EDA (page 122) models built using coordinate data in Å, unit of variance is Å². For ANM (page 101), GNM (page 112), and RTB (page 127), on the other hand, variance is the inverse of the eigenvalue, so it has arbitrary or relative units.

is3d()
Return True if model is 3-dimensional.

numAtoms()
Return number of atoms.

numDOF()
Return number of degrees of freedom.

numModes()
Return number of modes in the instance (not necessarily maximum number of possible modes).

performSVD(coordsets)
Calculate principal modes using singular value decomposition (SVD). coordsets argument may be a Atomic (page 44), Ensemble (page 132), or numpy.ndarray instance. If coordsets is a numpy array, its shape must be (n_csets, n_atoms, 3). Note that coordinate sets must be aligned prior to SVD calculations.

This is a considerably faster way of performing PCA calculations compared to eigenvalue decomposition of covariance matrix, but is an approximate method when heterogeneous datasets are analyzed. Covariance method should be preferred over this one for analysis of ensembles with missing atomic data. See Calculations244 example for comparison of results from SVD and covariance methods.

setCovariance(covariance)
Set covariance matrix.

setEigens(vectors, values=None)
Set eigen vectors and eigen values. If eigen values are omitted, they will be set to 1. Eigenvalues are set as variances.

setTitle(title)
Set title of the model.

243 http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
244 http://prody.csb.pitt.edu/tutorials/ensemble_analysis/xray_calculations.html#pca-xray-calculations
3.3.27 Plotting Functions

This module defines plotting functions for protein dynamics analysis.

Plotting functions are called by the name of the plotted data/property and are prefixed with show. Function
documentations refers to the matplotlib.pyplot function utilized for actual plotting. Arguments and
keyword arguments are passed to the Matplotlib functions.

\textbf{showContactMap} \texttt{(enm, *args, **kwargs)}

Show Kirchhoff matrix using \texttt{spy()}. 245

\textbf{showCrossCorr} \texttt{(modes, *args, **kwargs)}

Show cross-correlations using \texttt{imshow()}. By default, \texttt{origin=lower} and \texttt{interpolation=bilinear}
keyword arguments are passed to this function, but user can overwrite these parameters. See also \texttt{calcCrossCorr()} (page 99).

\textbf{showCumulOverlap} \texttt{(mode, modes, *args, **kwargs)}

Show cumulative overlap using \texttt{plot()}. 248

\textbf{showFractVars} \texttt{(modes, *args, **kwargs)}

Show fraction of variances using \texttt{bar()}. Note that mode indices are incremented by 1. 249

\textbf{showCumulFractVars} \texttt{(modes, *args, **kwargs)}

Show fraction of variances of \texttt{modes} using \texttt{plot()}. Note that mode indices are incremented by 1.
See also \texttt{showFractVars()} (page 124) function.

\textbf{showMode} \texttt{(mode, *args, **kwargs)}

Show mode array using \texttt{plot()}. 250

\textbf{showOverlap} \texttt{(mode, modes, *args, **kwargs)}

Show overlap \texttt{bar()}. 251

\textbf{showOverlapTable} \texttt{(modes_x, modes_y, **kwargs)}

Show overlap table using \texttt{pcolor()}. \texttt{modes_x} and \texttt{modes_y} are sets of normal modes, and corre-
spond to x and y axes of the plot. Note that mode indices are incremented by 1. List of modes is
assumed to contain a set of contiguous modes from the same model.

Default arguments for \texttt{pcolor()} 253:

\begin{itemize}
  \item \texttt{cmap=pl.t.cm.jet}
  \item \texttt{norm=pl.t.normalize(0, 1)}
\end{itemize}

245\url{http://matplotlib.sourceforge.net/api/pyplot_api.html#module-matplotlib.pyplot}
246\url{http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.spy}
247\url{http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.imshow}
248\url{http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot}
249\url{http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.pcolor}
250\url{http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.pcolor}
251\url{http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.pcolor}
252\url{http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.pcolor}
253\url{http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.pcolor}
**showProjection** *(ensemble, modes, *args, **kwargs)*

Show a projection of conformational deviations onto up to three normal modes from the same model.

**Parameters**

- **ensemble** *(Ensemble (page 132), Conformation (page 131), Vector (page 115), Trajectory (page 181)) – an ensemble, trajectory or a conformation for which deviation(s) will be projected, or a deformation vector*

- **modes** *(Mode (page 115), ModeSet (page 116), NMA (page 117)) – up to three normal modes*

- **color** *(str, list) – a color name or a list of color name, default is ’blue’*

- **label** *(str, list) – label or a list of labels*

- **marker** *(str, list) – a marker or a list of markers, default is ‘o’*

- **linestyle** *(str, list) – line style, default is ‘None’*

- **text** *(list) – list of text labels, one for each conformation*

- **fontsize** *(int) – font size for text labels*

  The projected values are by default converted to RMSD. Pass rmsd=False to use projection itself.

  Matplotlib function used for plotting depends on the number of modes:

  - 1 mode: `hist()`
  - 2 modes: `plot()`
  - 3 modes: `plot()`

**showCrossProjection** *(ensemble, mode_x, mode_y, scale=None, *args, **kwargs)*

Show a projection of conformational deviations onto modes from different models using `plot()`.

This function differs from `showProjection()` (page 124) by accepting modes from two different models.

**Parameters**

- **ensemble** *(Ensemble (page 132), Conformation (page 131), Vector (page 115), Trajectory (page 181)) – an ensemble or a conformation for which deviation(s) will be projected, or a deformation vector*

- **mode_x** *(Mode (page 115), Vector (page 115)) – projection onto this mode will be shown along x-axis*

- **mode_y** *(Mode (page 115), Vector (page 115)) – projection onto this mode will be shown along y-axis*

- **scale** *(str, list) – scale width of the projection onto mode x or y, best scaling factor will be calculated and printed on the console, absolute value of scalar makes the width of two projection same, sign of scalar makes the projections yield a positive correlation*

---

254 [http://docs.python.org/library/functions.html#str](http://docs.python.org/library/functions.html#str)
255 [http://docs.python.org/library/functions.html#list](http://docs.python.org/library/functions.html#list)
256 [http://docs.python.org/library/functions.html#int](http://docs.python.org/library/functions.html#int)
257 [http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.hist](http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.hist)
258 [http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot](http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot)
260 [http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot](http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot)
261 [http://docs.python.org/library/functions.html#str](http://docs.python.org/library/functions.html#str)
• **scalar** *(float)* – scalar factor for projection onto selected mode
• **color** *(str, list)* – a color name or a list of color name, default is *’blue’*
• **label** *(str, list)* – label or a list of labels
• **marker** *(str, list)* – a marker or a list of markers, default is *’o’*
• **linestyle** *(str)* – line style, default is *’None’*
• **text** *(list)* – list of text labels, one for each conformation
• **fontsize** *(int)* – font size for text labels

The projected values are by default converted to RMSD. Pass *rmsd=False* to calculate raw projection values. See *Plotting* for a more elaborate example.

**showEllipsoid** *(modes, onto=None, n_std=2, scale=1.0, *args, **kwargs)*
Show an ellipsoid using *plot_wireframe()*.

Ellipsoid volume gives an analytical view of the conformational space that given modes describe.

**Parameters**

• **modes** *(ModeSet (page 116), PCA (page 120), ANM (page 101), NMA (page 117)) – 3 modes for which ellipsoid will be drawn.*
• **onto** – 3 modes onto which ellipsoid will be projected.
• **n_std** *(float)* – Number of standard deviations to scale the ellipsoid.
• **scale** *(float)* – Used for scaling the volume of ellipsoid. This can be obtained from *sampleModes()* (page 128).

**showSqFlucts** *(modes, *args, **kwargs)*
Show square fluctuations using *plot()*.

**showScaledSqFlucts** *(modes, *args, **kwargs)*
Show scaled square fluctuations using *plot()*.
Modes or mode sets given as additional arguments will be scaled to have the same mean squared fluctuations as *modes*.

**showNormedSqFlucts** *(modes, *args, **kwargs)*
Show normalized square fluctuations via *plot()*.

**resetTicks** *(x, y=None)*
Reset X (and Y) axis ticks using values in given *array*. Ticks in the current figure should not be fractional values for this function to work as expected.

**showDiffMatrix** *(matrix1, matrix2, *args, **kwargs)*
Show the difference between two cross-correlation matrices from different models. For given *matrix1* and *matrix2* show the difference between them in the form of (matrix2 - matrix1) and plot the difference matrix using *imshow()*.
When *NMA* (page 117) models are passed instead of matrices, the functions could call *calcCrossCorr()* (page 99) function to calculate the matrices for given modes.
To display the absolute values in the difference matrix, user could set `abs` keyword argument `True`. By default, `origin=lower` and `interpolation=bilinear` keyword arguments are passed to this function, but user can overwrite these parameters.

### 3.3.28 Rotation Translation Blocks

This module defines a class and a function for rotating translating blocks (RTB) calculations.

**class RTB (name='Unknown')**

Class for Rotations and Translations of Blocks (RTB) method ([FT00] (page 260)). Optional arguments permit imposing constrains along Z-direction as in imANM method described in [TL12] (page 260).

**addEigenpair (vector, value=None)**

Add eigen vector and eigen value pair(s) to the instance. If eigen value is omitted, it will be set to 1. Inverse eigenvalues are set as variances.

**buildHessian (coords, blocks, cutoff=15.0, gamma=1.0, **kwargs)**

Build Hessian matrix for given coordinate set.

**Parameters**

- `coords` *(numpy.ndarray)* – a coordinate set or an object with `getCoords` method
- `blocks` *(list, numpy.ndarray)* – a list or array of block identifiers
- `cutoff` *(float)* – cutoff distance (Å) for pairwise interactions, default is 15.0 Å
- `gamma` *(float)* – spring constant, default is 1.0
- `scale` *(float)* – scaling factor for force constant along Z-direction, default is 1.0

**calcModes (n_modes=20, zeros=False, turbo=True)**

Calculate normal modes. This method uses `scipy.linalg.eigh()` function to diagonalize the Hessian matrix. When Scipy is not found, `numpy.linalg.eigh()` is used.

**Parameters**

- `n_modes` *(int or None, default is 20)* – number of non-zero eigenvalues/vectors to calculate. If None is given, all modes will be calculated.
- `zeros` *(bool, default is False)* – If True, modes with zero eigenvalues will be kept.
- `turbo` *(bool, default is True)* – Use a memory intensive, but faster way to calculate modes.

**getArray ()**

Return a copy of eigenvectors array.

**getCovariance ()**

Return covariance matrix. If covariance matrix is not set or yet calculated, it will be calculated using available modes.

---

275 [http://docs.python.org/library/functions.html#float](http://docs.python.org/library/functions.html#float)
276 [http://docs.python.org/library/functions.html#float](http://docs.python.org/library/functions.html#float)
277 [http://docs.python.org/library/functions.html#float](http://docs.python.org/library/functions.html#float)
getEigvals()
Return eigenvalues. For PCA (page 120) and EDA (page 122) models built using coordinate data in Å, unit of eigenvalues is Å². For ANM (page 101), GNM (page 112), and RTB (page 127), on the other hand, eigenvalues are in arbitrary or relative units but they correlate with stiffness of the motion along associated eigenvector.

getEigvecs()
Return a copy of eigenvectors array.

getHessian()
Return a copy of the Hessian matrix.

getModel()
Return self.

getProjection()
Return a copy of the projection matrix.

getTitle()
Return title of the model.

getVariances()
Return variances. For PCA (page 120) and EDA (page 122) models built using coordinate data in Å, unit of variance is Å². For ANM (page 101), GNM (page 112), and RTB (page 127), on the other hand, variance is the inverse of the eigenvalue, so it has arbitrary or relative units.

is3d()
Return True if model is 3-dimensional.

numAtoms()
Return number of atoms.

numDOF()
Return number of degrees of freedom.

numModes()
Return number of modes in the instance (not necessarily maximum number of possible modes).

setEigens(vectors, values=None)
Set eigen vectors and eigen values. If eigen values are omitted, they will be set to 1. Inverse eigenvalues are set as variances.

setHessian(hessian)
Set Hessian matrix. A symmetric matrix is expected, i.e. not a lower- or upper-triangular matrix.

setTitle(title)
Set title of the model.

3.3.29 Sampling Functions

This module defines functions for generating alternate conformations along normal modes.

defformAtoms(atoms, mode, rmsd=None)
Generate a new coordinate set for atoms along the mode. atoms must be a AtomGroup (page 37) instance. New coordinate set will be appended to atoms. If rmsd is provided, mode will be scaled to generate a coordinate set with given RMSD distance to the active coordinate set.

sampleModes(modes, atoms=None, n_confs=1000, rmsd=1.0)
Return an ensemble of randomly sampled conformations along given modes. If atoms are provided, sampling will be around its active coordinate set. Otherwise, sampling is around the 0 coordinate set.
Parameters

- **modes** *(Mode (page 115), ModeSet (page 116), PCA (page 120), ANM (page 101) or NMA (page 117)) – modes along which sampling will be performed*
- **atoms** *(Atomic (page 44)) – atoms whose active coordinate set will be used as the initial conformation*
- **n_confs** – number of conformations to generate, default is 1000
- **rmsd** *(float) – average RMSD that the conformations will have with respect to the initial conformation, default is 1.0 Å*

Returns **Ensemble** *(page 132)*

For given normal modes $[u_1 u_2 ... u_m]$ and their eigenvalues $[\lambda_1 \lambda_2 ... \lambda_m]$, a new conformation is sampled using the relation:

$$ R_k = R_0 + s \sum_{i=1}^{m} r_k^i \lambda_i^{-0.5} u_i $$

(3.1)

$R_0$ is the active coordinate set of $atoms$. $[r_1^k \ r_2^k ... r_m^k]$ are normally distributed random numbers generated for conformation $k$ using `numpy.random.randn()`.

RMSD of the new conformation from $R_0$ can be calculated as

$$ RMSD^k = \sqrt{\left( s \sum_{i=1}^{m} r_i^k \lambda_i^{-0.5} u_i \right)^2 / N} = \frac{s}{\sqrt{N}} \sqrt{\sum_{i=1}^{m} (r_i^k)^2 \lambda_i^{-1}} $$

(3.2)

Average RMSD of the generated conformations from the initial conformation is:

$$ \langle RMSD^k \rangle = \frac{s}{\sqrt{N}} \sqrt{\sum_{i=1}^{m} (r_i^k)^2 \lambda_i^{-1}} $$

(3.3)

From this relation $s$ scaling factor obtained using the relation

$$ s = \langle RMSD^k \rangle \sqrt{N} \sqrt{\sum_{i=1}^{m} (r_i^k)^2 \lambda_i^{-1}}^{-1} $$

(3.4)

Note that random numbers are generated before conformations are sampled, hence exact value of $s$ is known from this relation to ensure that the generated ensemble will have user given average rmsd value.

Note that if modes are from a PCA (page 120), variances are used instead of inverse eigenvalues, i.e. $\sigma_i \sim \lambda_i^{-1}$.

See also `showEllipsoid()` *(page 126).*

**traverseMode** *(mode, atoms, n_steps=10, rmsd=1.5)*

Generates a trajectory along a given mode, which can be used to animate fluctuations in an external program.

Parameters

- **mode** *(Mode (page 115)) – mode along which a trajectory will be generated*
- **atoms** *(Atomic (page 44)) – atoms whose active coordinate set will be used as the initial conformation*
• `n_steps` (int\(^{282}\)) – number of steps to take along each direction, for example, for `n_steps=10`, 20 conformations will be generated along the first mode, default is 10.

• `rmsd` (float\(^{283}\)) – maximum RMSD that the conformations will have with respect to the initial conformation, default is 1.5 Å

Returns `Ensemble` (page 132)

For given normal mode \(u_i\), its eigenvalue \(\lambda_i\), number of steps \(n\), and maximum RMSD conformations \([R_{-n}R_{-n+1}...R_{-1}R_0R_1...R_n]\) are generated.

\(R_0\) is the active coordinate set of atoms. \(R_k = R_0 + sk\lambda_i u_i\), where \(s\) is found using \(s = ((N(\frac{\text{RMSD}}{n})^2)/\lambda_i^{-1})^{0.5}\), where \(N\) is the number of atoms.

### 3.4 Ensemble Analysis

This module defines classes for handling conformational ensembles.

#### 3.4.1 Conformational ensembles

The following two classes are implemented for handling arbitrary but uniform conformational ensembles, e.g. NMR models, MD snapshots:

• `Ensemble` (page 132)

• `Conformation` (page 131)

See usage examples in `NMR Models\(^{284}\)` and `Essential Dynamics Analysis\(^{285}\)`.

#### 3.4.2 PDB ensembles

PDB ensembles, such as multiple structures of the same protein, are in general heterogeneous. This just means that different residues in different structures are missing. The following classes extend above to support this heterogeneity:

• `PDBEnsemble` (page 135)

• `PDBConformation` (page 131)

Following functions are for editing PDB ensembles, e.g. finding and removing residues that are missing in too many structures:

• `alignPDBEnsemble()` (page 134)

• `calcOccupancies()` (page 134)

• `showOccupancies()` (page 134)

• `trimPDBEnsemble()` (page 134)

See usage examples in `Heterogeneous X-ray Structures\(^{286}\)`, `Multimeric Structures\(^{287}\)`, `Homologous Proteins\(^{288}\)`.

---

\(^{282}\)http://docs.python.org/library/functions.html#int

\(^{283}\)http://docs.python.org/library/functions.html#float

\(^{284}\)http://prody.csb.pitt.edu/tutorials/ensemble_analysis/nmr.html#pca-nmr

\(^{285}\)http://prody.csb.pitt.edu/tutorials/trajectory_analysis/eda.html#eda

\(^{286}\)http://prody.csb.pitt.edu/tutorials/ensemble_analysis/xray.html#pca-xray

\(^{287}\)http://prody.csb.pitt.edu/tutorials/ensemble_analysis/dimer.html#pca-dimer

\(^{288}\)http://prody.csb.pitt.edu/tutorials/ensemble_analysis/blast.html#pca-blast
3.4.3 Save/load ensembles

- `saveEnsemble()` (page 134)
- `loadEnsemble()` (page 134)

3.4.4 Conformation

This module defines classes handling individual conformations.

```python
class Conformation(ensemble, index):
    A class to provide methods on a conformation in an ensemble. Instances of this class do not keep coordinate and weights data.
    getAtoms()
        Return associated atom group.
    getCoords()
        Return a copy of the coordinates of the conformation. If a subset of atoms are selected in the ensemble, coordinates for selected atoms will be returned.
    getDeviations()
        Return deviations from the ensemble reference coordinates. Deviations are calculated for (selected) atoms.
    getEnsemble()
        Return the ensemble that this conformation belongs to.
    getIndex()
        Return conformation index.
    getRMSD()
        Return RMSD from the ensemble reference coordinates. RMSD is calculated for (selected) atoms.
    getWeights()
        Return coordinate weights for (selected) atoms.
    numAtoms()
        Return number of atoms.
    numSelected()
        Return number of selected atoms.
```

```python
class PDBConformation(ensemble, index):
    This class is the same as Conformation (page 131), except that the conformation has a name (or identifier), e.g. PDB identifier.
    getAtoms()
        Return associated atom group.
    getCoords()
        Return a copy of the coordinates of the conformation. If a subset of atoms are selected in the ensemble, coordinates for selected atoms will be returned.
        Warning: When there are atoms with weights equal to zero (0), their coordinates will be replaced with the coordinates of the ensemble reference coordinate set.
    getDeviations()
        Return deviations from the ensemble reference coordinates. Deviations are calculated for (selected) atoms.
```

3.4. Ensemble Analysis
getEnsemble()  
Return the ensemble that this conformation belongs to.

getIndex()  
Return conformation index.

getLabel()  
Return the label of the conformation.

getRMSD()  
Return RMSD from the ensemble reference coordinates. RMSD is calculated for (selected) atoms.

getTransformation()  
Return the Transformation (page 143) used to superpose this conformation onto reference coordinates. The transformation can be used to superpose original PDB file onto the reference PDB file.

getWeights()  
Return coordinate weights for (selected) atoms.

numAtoms()  
Return number of atoms.

numSelected()  
Return number of selected atoms.

setLabel(label)  
Set the label of the conformation.

3.4.5 Conformational Ensemble

This module defines a class for handling ensembles of conformations.

class Ensemble (title='Unknown')  
A class for analysis of arbitrary conformational ensembles.

Indexing (e.g. ens[0]) returns a Conformation (page 131) instance that points to a coordinate set in the ensemble. Slicing (e.g. ens[0:10]) returns an Ensemble (page 132) instance that contains a copy of the subset of conformations (coordinate sets).

Instantiate with a title or a Atomic (page 44) instance. All coordinate sets from atomic instances will be added to the ensemble.

addCoordset(coords)  
Add coordinate set(s) to the ensemble. coords must be a Numpy array with suitable data type, shape and dimensionality, or an object with getCoordsets() (page 132) method.

delCoordset(index)  
Delete a coordinate set from the ensemble.

getAtoms()  
Return associated/selected atoms.

getConformation(index)  
Return conformation at given index.

getCoords(index)  
Return a copy of reference coordinates for selected atoms.

getCoordsets(indices=None)  
Return a copy of coordinate set(s) at given indices, which may be an integer, a list of integers
or None. None returns all coordinate sets. For reference coordinates, use getCoordinates() method.

getDeviations()
Return deviations from reference coordinates for selected atoms. Conformations can be aligned using one of superpose() (page 134) or iterpose() (page 133) methods prior to calculating deviations.

getMSFs()
Return mean square fluctuations (MSFs) for selected atoms. Conformations can be aligned using one of superpose() (page 134) or iterpose() (page 133) methods prior to MSF calculation.

getRMSDs()
Return root mean square deviations (RMSDs) for selected atoms. Conformations can be aligned using one of superpose() (page 134) or iterpose() (page 133) methods prior to RMSD calculation.

getRMSFs()
Return root mean square fluctuations (RMSFs) for selected atoms. Conformations can be aligned using one of superpose() (page 134) or iterpose() (page 133) methods prior to RMSF calculation.

ggetTitle()
Return title of the ensemble.

getWeights()
Return a copy of weights of selected atoms.

iterCoordsets()
Iterate over coordinate sets. A copy of each coordinate set for selected atoms is returned. Reference coordinates are not included.

iterpose(rmsd=0.0001)
Iteratively superpose the ensemble until convergence. Initially, all conformations are aligned with the reference coordinates. Then mean coordinates are calculated, and are set as the new reference coordinates. This is repeated until reference coordinates do not change. This is determined by the value of RMSD between the new and old reference coordinates. Note that at the end of the iterative procedure the reference coordinate set will be average of conformations in the ensemble.

Parameters rmsd (float) – change in reference coordinates to determine convergence, default is 0.0001 Å RMSD

numAtoms()
Return number of atoms.

numConfs()
Return number of conformations.

numCoordsets()
Return number of conformations.

numSelected()
Return number of selected atoms. Number of all atoms will be returned if a selection is not made. A subset of atoms can be selected by passing a selection to setAtoms() (page 133).

setAtoms(atomic)
Set atoms or specify a selection of atoms to be considered in calculations and coordinate requests. When a selection is set, corresponding subset of coordinates will be considered in, for example,
alignments and RMSD calculations. Setting atoms also allows some functions to access atomic data when needed. For example, Ensemble (page 132) and Conformation (page 131) instances become suitable arguments for writePDB() (page 163). Passing None as atoms argument will deselect atoms.

 setCoords (coords)
 Set coords as the ensemble reference coordinate set. coords may be an array with suitable data type, shape, and dimensionality, or an object with getCoords() (page 132) method.

 setTitle (title)
 Set title of the ensemble.

 setWeights (weights)
 Set atomic weights.

 superpose ()
 Superpose the ensemble onto the reference coordinates.

### 3.4.6 Supporting Functions

This module defines a functions for handling conformational ensembles.

 saveEnsemble (ensemble, filename=None, **kwargs)
 Save ensemble model data as filename.ens.npz. If filename is None, title of the ensemble will be used as the filename, after white spaces in the title are replaced with underscores. Extension is .ens.npz. Upon successful completion of saving, filename is returned. This function makes use of numpy.savez() function.

 loadEnsemble (filename)
 Return ensemble instance loaded from filename. This function makes use of numpy.load() function. See also saveEnsemble() (page 134)

 trimPDBEnsemble (pdb_ensemble, **kwargs)
 Return a new PDB ensemble obtained by trimming given pdb_ensemble. This function helps selecting atoms in a pdb ensemble based on one of the following criteria, and returns them in a new PDBEnsemble (page 135) instance.

#### Occupancy

Resulting PDB ensemble will contain atoms whose occupancies are greater or equal to occupancy keyword argument. Occupancies for atoms will be calculated using calcOccupancies(pdb_ensemble, normed=True).

**Parameters** occupancy (float) – occupancy for selecting atoms, must satisfy 0 < occupancy <= 1

 calcOccupancies (pdb_ensemble, normed=False)
 Return occupancy calculated from weights of a PDBEnsemble (page 135). Any non-zero weight will be considered equal to one. Occupancies are calculated by binary weights for each atom over the conformations in the ensemble. When normed is True, total weights will be divided by the number of atoms. This function can be used to see how many times a residue is resolved when analyzing an ensemble of X-ray structures.

 showOccupancies (pdbensemble, *args, **kwargs)
 Show occupancies for the PDB ensemble using plot(). Occupancies are calculated using calcOccupancies() (page 134).
alignPDBEnsemble(ensemble, suffix='_aligned', outdir='.', gzip=False)

Align PDB files using transformations from ensemble, which may be a PDBEnsemble (page 135) or a PDBConformation (page 131) instance. Label of the conformation (see getLabel() (page 132)) will be used to determine the PDB structure and model number. First four characters of the label is expected to be the PDB identifier and ending numbers to be the model number. For example, the Transformation (page 143) from conformation with label 2k39_ca_selection_resnum_<71’m116 will be applied to 116th model of structure 2k39. After applicable transformations are made, structure will be written into outputdir as 2k39_aligned.pdb. If gzip is True, output files will be compressed. Return value is the output filename or list of filenames, in the order files are processed. Note that if multiple models from a file are aligned, that filename will appear in the list multiple times.

3.4.7 PDB Structure Ensemble

This module defines a class for handling ensembles of PDB conformations.

class PDBEnsemble(title='Unknown')

This class enables handling coordinates for heterogeneous structural datasets and stores identifiers for individual conformations.

See usage in Heterogeneous X-ray Structures\textsuperscript{293}, Multimeric Structures\textsuperscript{294}, and Homologous Proteins\textsuperscript{295}.

Note: This class is designed to handle conformations with missing coordinates, e.g. atoms that are not resolved in an X-ray structure. For unresolved atoms, the coordinates of the reference structure is assumed in RMSD calculations and superpositions.

addCoordset(coords, weights=None, label=None)

Add coordinate set(s) to the ensemble. coords must be a Numpy array with suitable shape and dimensionality, or an object with getCoordsets() (page 135) method. weights is an optional argument. If provided, its length must match number of atoms. Weights of missing (not resolved) atoms must be 0 and weights of those that are resolved can be anything greater than 0. If not provided, weights of all atoms for this coordinate set will be set equal to 1. label, which may be a PDB identifier or a list of identifiers, is used to label conformations.

delCoordset(index)

Delete a coordinate set from the ensemble.

getAtoms()

Return associated/selected atoms.

getConformation(index)

Return conformation at given index.

getCoords()

Return a copy of reference coordinates for selected atoms.

getCoordsets(indices=None)

Return a copy of coordinate set(s) at given indices for selected atoms. indices may be an integer, a list of integers or None. None returns all coordinate sets.

Warning: When there are atoms with weights equal to zero (0), their coordinates will be replaced with the coordinates of the ensemble reference coordinate set.

\textsuperscript{293}http://prody.csb.pitt.edu/tutorials/ensemble_analysis/xray.html#pca-xray
\textsuperscript{294}http://prody.csb.pitt.edu/tutorials/ensemble_analysis/dimer.html#pca-dimer
\textsuperscript{295}http://prody.csb.pitt.edu/tutorials/ensemble_analysis/blast.html#pca-blast
getDeviations()
Return deviations from reference coordinates for selected atoms. Conformations can be aligned using one of superpose() (page 137) or iterpose() (page 136) methods prior to calculating deviations.

getLabels()
Return identifiers of the conformations in the ensemble.

getMSFs()
Calculate and return mean square fluctuations (MSFs). Note that you might need to align the conformations using superpose() (page 137) or iterpose() (page 136) before calculating MSFs.

getRMSDs()
Calculate and return root mean square deviations (RMSDs). Note that you might need to align the conformations using superpose() (page 137) or iterpose() (page 136) before calculating RMSDs.

getRMSFs()
Return root mean square fluctuations (RMSFs) for selected atoms. Conformations can be aligned using one of superpose() (page 137) or iterpose() (page 136) methods prior to RMSF calculation.

ggetTitle()
Return title of the ensemble.

getWeights()
Return a copy of weights of selected atoms.

iterCoordsets()
Iterate over coordinate sets. A copy of each coordinate set for selected atoms is returned. Reference coordinates are not included.

iterpose(rmsd=0.0001)
Iteratively superpose the ensemble until convergence. Initially, all conformations are aligned with the reference coordinates. Then mean coordinates are calculated, and are set as the new reference coordinates. This is repeated until reference coordinates do not change. This is determined by the value of RMSD between the new and old reference coordinates. Note that at the end of the iterative procedure the reference coordinate set will be average of conformations in the ensemble.

Parameters rmsd (float296) – change in reference coordinates to determine convergence, default is 0.0001 Å RMSD

numAtoms()
Return number of atoms.

numConfs()
Return number of conformations.

numCoordsets()
Return number of conformations.

numSelected()
Return number of selected atoms. Number of all atoms will be returned if a selection is not made. A subset of atoms can be selected by passing a selection to setAtoms() (page 136).

setAtoms(atomic)
Set atoms or specify a selection of atoms to be considered in calculations and coordinate requests.

296http://docs.python.org/library/functions.html#float
When a selection is set, corresponding subset of coordinates will be considered in, for example, alignments and RMSD calculations. Setting atoms also allows some functions to access atomic data when needed. For example, Ensemble (page 132) and Conformation (page 131) instances become suitable arguments for writePDB() (page 163). Passing None as atoms argument will deselect atoms.

`setCoords(coords)`
Set `coords` as the ensemble reference coordinate set. `coords` may be an array with suitable data type, shape, and dimensionality, or an object with `getCoords()` (page 135) method.

`setTitle(title)`
Set title of the ensemble.

`setWeights(weights)`
Set atomic weights.

`superpose()`
Superpose the ensemble onto the reference coordinates.

### 3.5 KDTree

This module provides KDTree (page 137) class as an interface to Thomas Hamelryck’s KDTree C module distributed with Biopython.

#### 3.5.1 KD Tree

This module defines KDTree (page 137) class for dealing with atomic coordinate sets and handling periodic boundary conditions.

`class KDTree(coords, **kwargs)`
An interface to Thomas Hamelryck’s C KDTree module that can handle periodic boundary conditions. Both point and pair search are performed using the single `search()` (page 138) method and results are retrieved using `getIndices()` (page 138) and `getDistances()` (page 138).

**Periodic Boundary Conditions**

*Point search*

A point search around a center, indicated with a question mark (?) below, involves making images of the point in cells sharing a wall or an edge with the unitcell that contains the system. The search is performed for all images of the center (27 in 3-dimensional space) and unique indices with the minimum distance from them to the center are returned.

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>o</td>
<td>h</td>
</tr>
<tr>
<td>5</td>
<td>h</td>
<td>o</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>h</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

? and H interact in periodic image 4

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

but not in the original unitcell (5)

There are two requirements for this approach to work: (i) the center must be in the original unitcell, and (ii) the system must be in the original unitcell with parts in its immediate periodic images.

*Pair search*
A pair search involves making 26 (or 8 in 2-d) replicas of the system coordinates. A KDTree is built for the system (O and H) and all its replicas (o and h). After pair search is performed, unique pairs of indices and minimum distance between them are returned.

```
| o  h | h  h 1 | o  h 2 | o  h 3 |
| h  o | h  o  h | h  o  |
|_______|_________|_________|
| o  h 4 | O  H 5 | o  h 6 |
| h  o  H | H  O  h | h  o  |
|_______|_________|_________|
| o  h 7 | o  h 8 | o  h 9 |
| h  o  h | h  o  h | h  o  |
|_______|_________|_________|
```

Only requirement for this approach to work is that the system must be in the original unitcell with parts in its immediate periodic images.

See Also:

wrapAtoms() (page 145) can be used for wrapping atoms into the single periodic image of the system.

Parameters

- `coords` (numpy.ndarray\(^{297}\), Atomic (page 44), Frame (page 178)) – coordinate array with shape \((N, 3)\), where \(N\) is number of atoms
- `unitcell` (numpy.ndarray\(^{298}\)) – orthorhombic unitcell dimension array with shape \((3,)\)
- `bucketsize` (int\(^{299}\)) – number of points per tree node, default is 10

getCount()
Return number of points or pairs.

getDistances()
Return array of distances.

getIndices()
Return array of indices for points or pairs, depending on the type of the most recent search.

search(radius, center=None)
Search pairs within \(radius\) of each other or points within \(radius\) of \(center\).

Parameters

- `radius` (float\(^{300}\)) – distance (Å)
- `center` (numpy.ndarray\(^{301}\)) – a point in Cartesian coordinate system

3.6 Measurement Tools

This module defines classes measuring quantities, transforming coordinates, and identifying contacts.

\(^{297}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

\(^{298}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

\(^{299}\)http://docs.python.org/library/functions.html#int

\(^{300}\)http://docs.python.org/library/functions.html#float

\(^{301}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray
3.6.1 Identify contacts

Following class and functions are for contact identifications:

- \texttt{Contacts} (page 140) - identify intermolecular contacts
- \texttt{findNeighbors()} (page 140) - identify interacting atom pairs
- \texttt{iterNeighbors()} (page 140) - identify interacting atom pairs

3.6.2 Measure quantities

Following functions are for measuring simple quantities:

- \texttt{calcDistance()} (page 140) - calculate distance(s)
- \texttt{calcAngle()} (page 141) - calculate bond angle
- \texttt{calcDihedral()} (page 141) - calculate dihedral angle
- \texttt{calcOmega()} (page 141) - calculate omega (\( \omega \)) angle
- \texttt{calcPhi()} (page 141) - calculate phi (\( \phi \)) angle
- \texttt{calcPsi()} (page 141) - calculate psi (\( \psi \)) angle
- \texttt{calcGyradius()} (page 141) - calculate radius of gyration
- \texttt{calcCenter()} (page 141) - calculate geometric (or mass) center
- \texttt{calcDeformVector()} (page 142) - calculate deformation vector

3.6.3 Anisotropic factors

Following functions handle anisotropic displacement parameter (ADP) present in some X-ray structures.

- \texttt{buildADPMatrix()} (page 142) - build ADP matrix
- \texttt{calcADPAxes()} (page 142) - calculate ADP axes
- \texttt{calcADPs()} (page 143) - calculate ADPs

3.6.4 Transformations

Following class and functions are for handling coordinate transformations:

- \texttt{Transformation} (page 143) - store transformation matrix
- \texttt{alignCoordsets()} (page 144) - align multiple coordinate sets
- \texttt{applyTransformation()} (page 144) - apply a transformation
- \texttt{calcTransformation()} (page 144) - calculate a transformation
- \texttt{calcRMSD()} (page 144) - calculate root-mean-square distance
- \texttt{superpose()} (page 144) - superpose atoms or coordinate sets
- \texttt{moveAtoms()} (page 144) - move atoms by given offset
3.6.5 Contact Identification

This module defines a class and function for identifying contacts.

class Contacts (atoms, unitcell=None)

A class for contact identification. Contacts are identified using the coordinates of atoms at the time of instantiation.

atoms must be an Atomic (page 44) instance. When an orthorhombic unitcell array is given

getAtoms ()

Return atoms, or coordinate array, provided at instantiation..

getUnitcell ()

Return unitcell array, or None if one was not provided.

select (radius, center)

Select atoms radius radius (Å) of center, which can be point(s) in 3-d space (numpy.ndarray\(^{302}\) with shape (n_atoms, 3)) or a set of atoms, e.g. Selection (page 86).

iterNeighbors (atoms, radius, atoms2=None, unitcell=None)

Yield pairs of atoms that are within radius of each other and the distance between them. If atoms2 is also provided, one atom from atoms and another from atoms2 will be yielded. If one of atoms or atoms2 is a coordinate array, pairs of indices and distances will be yielded. When orthorhombic unitcell dimensions are provided, periodic boundary conditions will be taken into account (see KDTree (page 137) and also wrapAtoms() for details). If atoms is a Frame (page 178) instance and unitcell is not provided, unitcell information from frame will be if available.

findNeighbors (atoms, radius, atoms2=None, unitcell=None)

Return list of neighbors that are within radius of each other and the distance between them. See iterNeighbors() (page 140) for more details.

3.6.6 Measurement Tools

This module defines a class and methods and for comparing coordinate data and measuring quantities.

buildDistMatrix (atoms1, atoms2=None, unitcell=None, format='mat')

Return distance matrix. When atoms2 is given, a distance matrix with shape (len(atoms1), len(atoms2)) is built. When atoms2 is None, a symmetric matrix with shape (len(atoms1), len(atoms1)) is built. If unitcell array is provided, periodic boundary conditions will be taken into account.

Parameters

- atoms1 (Atomic (page 44), numpy.ndarray\(^{303}\)) – atom or coordinate data
- atoms2 (Atomic (page 44), numpy.ndarray\(^{304}\)) – atom or coordinate data
- unitcell (numpy.ndarray\(^{305}\)) – orthorhombic unitcell dimension array with shape (3,)
- format (bool\(^{306}\)) – format of the resulting array, one of ’mat’ (matrix, default), ’rcd’ (arrays of row indices, column indices, and distances), or ’arr’ (only array of distances)

---

\(^{302}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

\(^{303}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

\(^{304}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

\(^{305}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

\(^{306}\)http://docs.python.org/library/functions.html#bool
calcDistance \((\text{atoms1}, \text{atoms2}, \text{unitcell}=\text{None})\)

Return the Euclidean distance between \text{atoms1} and \text{atoms2}. Arguments may be \text{Atomic} (page 44) instances or NumPy arrays. Shape of numpy arrays must be \((M, N, 3)\), where \(M\) is number of coordinate sets and \(N\) is the number of atoms. If \text{unitcell} array is provided, periodic boundary conditions will be taken into account.

**Parameters**

- \text{atoms1} (\text{Atomic} (page 44), \text{numpy.ndarray}^{307}) – atom or coordinate data
- \text{atoms2} (\text{Atomic} (page 44), \text{numpy.ndarray}^{308}) – atom or coordinate data
- \text{unitcell} (\text{numpy.ndarray}^{309}) – orthorhombic unitcell dimension array with shape \((3,)\)

calcCenter \((\text{atoms}, \text{weights}=\text{None})\)

Return geometric center of \text{atoms}. If \text{weights} is given it must be a flat array with length equal to number of atoms. Mass center of atoms can be calculated by setting weights equal to atom masses, i.e. \text{weights} = \text{atoms}.getMasses().

calcGyradius \((\text{atoms}, \text{weights}=\text{None})\)

Calculate radius of gyration of \text{atoms}.

calcAngle \((\text{atoms1}, \text{atoms2}, \text{atoms3}, \text{radian}=\text{False})\)

Return the angle between atoms in degrees.

calcDihedral \((\text{atoms1}, \text{atoms2}, \text{atoms3}, \text{atoms4}, \text{radian}=\text{False})\)

Return the dihedral angle between atoms in degrees.

calcOmega \((\text{residue}, \text{radian}=\text{False}, \text{dist}=4.1)\)

Return \(\omega\) (omega) angle of \text{residue} in degrees. This function checks the distance between \(\text{Ca}\) atoms of two residues and raises an exception if the residues are disconnected. Set \text{dist} to \text{None}, to avoid this.

calcPhi \((\text{residue}, \text{radian}=\text{False}, \text{dist}=4.1)\)

Return \(\phi\) (phi) angle of \text{residue} in degrees. This function checks the distance between \(\text{Ca}\) atoms of two residues and raises an exception if the residues are disconnected. Set \text{dist} to \text{None}, to avoid this.

calcPsi \((\text{residue}, \text{radian}=\text{False}, \text{dist}=4.1)\)

Return \(\psi\) (psi) angle of \text{residue} in degrees. This function checks the distance between \(\text{Ca}\) atoms of two residues and raises an exception if the residues are disconnected. Set \text{dist} to \text{None}, to avoid this.

calcMSF \((\text{coordsets})\)

Calculate mean square fluctuation(s) (MSF). \text{coordsets} may be an instance of \text{Ensemble} (page 132), \text{TrajBase} (page 179), or \text{Atomic} (page 44). For trajectory objects, e.g. \text{DCDFile} (page 175), frames will be considered after they are superposed. For other ProDy objects, coordinate sets should be aligned prior to MSF calculation.

Note that using trajectory files that store 32-bit coordinate will result in lower precision in calculations. Over 10,000 frames this may result in up to 5% difference from the values calculated using 64-bit arrays. To ensure higher-precision calculations for \text{DCDFile} (page 175) instances, you may use \text{astype} argument, i.e. \text{astype} = \text{float}, to auto recast coordinate data to double-precision (64-bit) floating-point format.

calcRMSF \((\text{coordsets})\)

Return root mean square fluctuation(s) (RMSF). \text{coordsets} may be an instance of \text{Ensemble} (page 132), \text{TrajBase} (page 179), or \text{Atomic} (page 44). For trajectory objects, e.g. \text{DCDFile} (page 175), frames will be considered after they are superposed. For other ProDy objects, coordinate sets should be aligned prior to MSF calculation.

3.6. Measurement Tools
Note that using trajectory files that store 32-bit coordinate will result in lower precision in calculations. Over 10,000 frames this may result in up to 5% difference from the values calculated using 64-bit arrays. To ensure higher-precision calculations for DCDFile (page 175) instances, you may use astype argument, i.e. astype=float, to auto recast coordinate data to double-precision (64-bit) floating-point format.

**calcDeformVector** *(from_atoms, to_atoms)*
Return deformation from *from_atoms* to *atoms_to* as a Vector (page 115) instance.

**buildADPMatrix** *(atoms)*
Return a 3Nx3N symmetric matrix containing anisotropic displacement parameters (ADPs) along the diagonal as 3x3 super elements.

```
In [1]: from prody import *

In [2]: protein = parsePDB('1ejg')

In [3]: calphas = protein.select('calpha')

In [4]: adp_matrix = buildADPMatrix(calphas)
```

**calcADPAxes** *(atoms, **kwargs)*
Return a 3Nx3 array containing principal axes defining anisotropic displacement parameter (ADP, or anisotropic temperature factor) ellipsoids.

Parameters

- **atoms** *(Atomic (page 44)) – a ProDy object for handling atomic data*
- **frac** *(float)* – For an atom, if the fraction of anisotropic displacement explained by its largest axis/eigenvector is less than given value, all axes for that atom will be set to zero. Values larger than 0.33 and smaller than 1.0 are accepted.
- **ratio2** *(float)* – For an atom, if the ratio of the second-largest eigenvalue to the largest eigenvalue axis less than or equal to the given value, all principal axes for that atom will be returned. Values less than 1 and greater than 0 are accepted.
- **ratio3** *(float)* – For an atom, if the ratio of the smallest eigenvalue to the largest eigenvalue is less than or equal to the given value, all principal axes for that atom will be returned. Values less than 1 and greater than 0 are accepted.
- **ratio** *(float)* – Same as ratio3.

Keyword arguments frac, ratio3, or ratio3 can be used to set principal axes to 0 for atoms showing relatively lower degree of anisotropy.

3Nx3 axis contains N times 3x3 matrices, one for each given atom. Columns of these 3x3 matrices are the principal axes which are weighted by square root of their eigenvalues. The first columns correspond to largest principal axes.

The direction of the principal axes for an atom is determined based on the correlation of the axes vector with the principal axes vector of the previous atom.

```
In [1]: from prody import *

In [2]: protein = parsePDB('1ejg')
```

---

3.6. Measurement Tools
In [3]: calphas = protein.select('calpha')
In [4]: adp_axes = calcADPAxes( calphas )
In [5]: adp_axes.shape
Out[5]: (138, 3)

These can be written in NMD format as follows:
In [6]: nma = NMA('ADPs')
In [7]: nma.setEigens(adp_axes)
In [8]: nma
Out[8]: <NMA: ADPs (3 modes; 46 atoms)>
In [9]: writeNMD('adp_axes.nmd', nma, calphas)
Out[9]: 'adp_axes.nmd'

calcADPs (atom)
Calculate anisotropic displacement parameters (ADPs) from anisotropic temperature factors (ATFs).
atom must have ATF values set for ADP calculation. ADPs are returned as a tuple, i.e. (eigenvalues, eigenvectors).

pickCentral (obj, weights=None)
Return Atom (page 33) or Conformation (page 131) that is closest to the center of obj, which may be an Atomic (page 44) or Ensemble (page 132) instance. See also pickCentralAtom() (page 143), and pickCentralConf() (page 143) functions.

pickCentralAtom (atoms, weights=None)
Return Atom (page 33) that is closest to the center, which is calculated using calcCenter() (page 141).

pickCentralConf (ens, weights=None)
Return Conformation (page 131) that is closest to the center of ens. In addition to Ensemble (page 132) instances, Atomic (page 44) instances are accepted as ens argument. In this case a Selection (page 86) with central coordinate set as active will be returned.

3.6.7 Transformations
This module defines a class for identifying contacts.

class Transformation (*args)
A class for storing a transformation matrix.

   Either 4x4 transformation matrix, or rotation matrix and translation vector must be provided at instantiation.

   apply (atoms)
   Apply transformation to atoms, see applyTransformation() (page 144) for details.

   getMatrix()
   Returns a copy of the 4x4 transformation matrix whose top left is rotation matrix and last column is translation vector.

   getRotation()
   Return rotation matrix.

3.6. Measurement Tools
getTranslation()
    Return translation vector.

setRotation(rotation)
    Set rotation matrix.

setTranslation(translation)
    Set translation vector.

applyTransformation(transformation, atoms)
    Return atoms after applying transformation. If atoms is a Atomic (page 44) instance, it will be returned after transformation is applied to its active coordinate set. If atoms is an AtomPointer (page 68) instance, transformation will be applied to the corresponding coordinate set in the associated AtomGroup (page 37).

alignCoordsets(atoms, weights=None)
    Return atoms after superposing coordinate sets onto its active coordinate set. Transformations will be calculated for atoms and applied to its AtomGroup (page 37), when applicable. Optionally, atomic weights can be passed for weighted superposition.

calcRMSD(reference, target=None, weights=None)
    Return root-mean-square deviation(s) (RMSD) between reference and target coordinates.

calcTransformation(mobile, target, weights=None)
    Returns a Transformation (page 143) instance which, when applied to the atoms in mobile, minimizes the weighted RMSD between mobile and target. mobile and target may be NumPy coordinate arrays, or Atomic (page 44) instances, e.g. AtomGroup (page 37), Chain (page 49), or Selection (page 86).

superpose(mobile, target, weights=None)
    Return mobile, after its RMSD minimizing superposition onto target, and the transformation that minimizes the RMSD.

moveAtoms(atoms, **kwargs)
    Move atoms to a new location or by an offset. This method will change the active coordinate set of the atoms. Note that only one of to or by keyword arguments is expected.

Move protein so that its centroid is at the origin, [0., 0., 0.]:

In [1]: from prody import *

In [2]: from numpy import ones, zeros

In [3]: protein = parsePDB('1ubi')

In [4]: calcCenter(protein).round(3)
Out[4]: array([ 30.173,  28.658,  15.262])

In [5]: moveAtoms(protein, to=zeros(3))

In [6]: calcCenter(protein).round(3)
Out[6]: array([ 0.,  0., -0.])

Move protein so that its mass center is at the origin:

In [7]: protein.setMasses(ones(len(protein)))

In [8]: protein.carbon.setMasses(12)

In [9]: protein.nitrogen.setMasses(14)
In [10]: protein.oxygen.setMasses(16)

In [11]: moveAtoms(protein, to=zeros(3), weights=protein.getMasses())

In [12]: calcCenter(protein, weights=protein.getMasses()).round(3)
   Out[12]: array([-0., -0., 0.])

Move protein so that centroid of Cα atoms is at the origin:

In [13]: moveAtoms(protein.ca, to=zeros(3), ag=True)

In [14]: calcCenter(protein).round(3)
   Out[14]: array([-0.268, -0.343, -0.259])

In [15]: calcCenter(protein.ca).round(3)
   Out[15]: array([ 0., -0., -0.])

Move protein by 10 Å along each direction:

In [16]: moveAtoms(protein, by=ones(3) * 10)

In [17]: calcCenter(protein).round(3)

In [18]: calcCenter(protein.ca).round(3)
   Out[18]: array([ 10., 10., 10.])

Parameters

- **by** (numpy.ndarray\(^{314}\)) – an offset array with shape ([1,] 3) or (n_atoms, 3) or a transformation matrix with shape (4, 4)
- **to** (numpy.ndarray\(^{315}\)) – a point array with shape ([1,] 3)
- **ag** (bool\(^{316}\)) – when atoms is a AtomSubset (page 90), apply translation vector (to) or transformation matrix to the AtomGroup (page 37), default is False
- **weights** (numpy.ndarray\(^{317}\)) – array of atomic weights with shape (n_atoms[, 1])

When to argument is passed, calcCenter() (page 141) function is used to calculate centroid or mass center.

wrapAtoms(frame, unitcell=None, center=array([ 0., 0., 0.]))

Wrap atoms into an image of the system simulated under periodic boundary conditions. When frame is a Frame (page 178), unitcell information will be retrieved automatically.

**Note:** This function will wrap all atoms into the specified periodic image, so covalent bonds will be broken.

Parameters

---

\(^{314}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

\(^{315}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

\(^{316}\)http://docs.python.org/library/functions.html#bool

\(^{317}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

3.6. Measurement Tools
• frame \(\text{Frame}\) (page 178), \text{AtomGroup}\ (page 37), \text{numpy.ndarray}\(^{318}\) – a frame instance or a coordinate set

• unitcell \(\text{numpy.ndarray}\(^{319}\) – orthorhombic unitcell array with shape (3,)

• center \(\text{numpy.ndarray}\(^{320}\) – coordinates of the center of the wrapping cell, default is the origin of the Cartesian coordinate system

\text{printRMSD}\ (\text{reference}, \text{target}=\text{None}, \text{weights}=\text{None}, \text{log}=\text{True}, \text{msg}=\text{None})

Print RMSD to the screen. If \text{target} has multiple coordinate sets, minimum, maximum and mean RMSD values are printed. If \text{log} is \text{True} (default), RMSD is written to the standard error using package logger, otherwise standard output is used. When \text{msg} string is given, it is printed before the RMSD value. See also \text{calcRMSD()}\ (page 144) function.

3.7 Protein Structure

This module defines classes and functions to fetch, parse, and write structural data files, execute structural analysis programs, and to access and search structural databases, e.g. \text{ProteinDataBank}\(^{321}\).

3.7.1 PDB resources

• \text{fetchPDB()}\ (page 160) - retrieve PDB files

• \text{fetchPDBviaFTP()}\ (page 165) - download PDB/PDBML/mmCIF files

• \text{fetchPDBviaHTTP()}\ (page 165) - download PDB files

You can use following functions to manage PDB file resources:

• \text{pathPDBFolder()}\ (page 160) - local folder for storing PDB files

• \text{pathPDBMirror()}\ (page 160) - local PDB mirror path

• \text{wwPDBServer()}\ (page 165) - set \text{wwPDB} FTP/HTTP server for downloads

Following functions can be used to handle local PDB files:

• \text{findPDBFiles()}\ (page 160) - return a dictionary containing files in a path

• \text{iterPDBFilenames()}\ (page 160) - yield file names in a path or local PDB mirror

3.7.2 Blast search PDB

The following are for blast searching PDB content.

• \text{blastPDB()}\ (page 149) - blast search NCBI PDB database

• \text{PDBBlastRecord}\ (page 148) - store/evaluate NCBI PDB blast search results

PDB clusters biopolymer chains using blast weekly. These clusters can be retrieved using the following functions. Using cluster data is as good as blast searching PDB most of the time and incredibly faster always.

• \text{listPDBCluster()}\ (page 161) - get list of identifiers in a PDB sequence cluster

---

\(^{318}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

\(^{319}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

\(^{320}\)http://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html#numpy.ndarray

\(^{321}\)http://wwpdb.org
• loadPDBClusters() (page 161) - load PDB clusters into memory
• fetchPDBClusters() (page 161) - retrieve PDB sequence cluster data from wwPDB

3.7.3 Parse/write PDB files

Following ProDy functions are for parsing and writing .pdb files:

• parsePDB() (page 162) - parse .pdb formatted file
• parsePDBStream() (page 161) - parse .pdb formatted stream
• writePDB() (page 163) - write .pdb formatted file
• writePDBStream() (page 163) write .pdb formatted stream

Since .pqr format is similar to .pdb format, following functions come as bonus features:

• writePQR() (page 153) - write atomic data to a file in .pqr format
• parsePQR() (page 163) - parse atomic data from files in .pqr format

See Also:
Atom data (coordinates, atom names, residue names, etc.) parsed from PDB/PSF/PQR files are stored in AtomGroup (page 37) instances. See atomic (page 31) module documentation for more details.

3.7.4 Quick visualization

showProtein() (page 153) function can be used to take a quick look at protein structures.

3.7.5 Edit structures

Following functions allow editing structures using structural data from PDB header records:

• assignSecstr() (page 158) - add secondary structure data from header to atoms
• buildBiomolecules() (page 159) - build biomolecule from header records

3.7.6 PDB header data

Use the following to parse and access header data in PDB files:

• parsePDBHeader() (page 157) - parse header data from .pdb files
• Chemical (page 153) - store PDB chemical (heterogen) component data
• Polymer (page 154) - store PDB polymer (macromolecule) component data
• DBRef (page 156) - store polymer sequence database reference records

3.7.7 Ligand data

Following function can be used to fetch meta data on PDB ligands:

• fetchPDBLigand() (page 164) - retrieve ligand from Ligand-Expo
3.7.8 Compare/align chains

Following functions can be used to match, align, and map polypeptide chains:

- **matchChains()** (page 149) - finds matching chains in two protein structures
- **matchAlign()** (page 150) - finds best matching chains and aligns structures
- **mapOntoChain()** (page 151) - maps chains in a structure onto a reference chain

Following functions can be used to adjust alignment parameters:

- **get.AlignmentMethod()** (page 152), **set.AlignmentMethod()** (page 152)
- **get.MatchScore()** (page 151), **set.MatchScore()** (page 151)
- **get.MismatchScore()** (page 151), **set.MismatchScore()** (page 151)
- **get.GapPenalty()** (page 151), **set.GapPenalty()** (page 151)
- **get.GapExtPenalty()** (page 152), **set.GapExtPenalty()** (page 152)

3.7.9 Execute DSSP

Following functions can be used to execute DSSP structural analysis program and/or parse results:

- **execDSSP()** (page 152) - execute dssp
- **performDSSP()** (page 153) - execute dssp and parse results
- **parseDSSP()** (page 152) - parse structural data from dssp output

3.7.10 Execute STRIDE

Following functions can be used to execute STRIDE structural analysis program and/or parse results:

- **execSTRIDE()** (page 164) - execute stride
- **performSTRIDE()** (page 165) - execute stride and parse results
- **parseSTRIDE()** (page 165) - parse structural data from stride output

3.7.11 PDB Blast Search

This module defines functions for blast searching Protein Data Bank.

**class PDBBlastRecord(xml, sequence=None)**

A class to store results from ProteinDataBank blast search.

Instantiate a PDBlast object instance.

Parameters

- **xml** (*str*) – blast search results in XML format or an XML file that contains the results
- **sequence** (*str*) – query sequence

---

322 [http://docs.python.org/library/functions.html#str](http://docs.python.org/library/functions.html#str)
323 [http://docs.python.org/library/functions.html#str](http://docs.python.org/library/functions.html#str)
getBest()  
Return a dictionary containing structure and alignment information for the hit with highest sequence identity.

getHits(percent_identity=90.0, percent_overlap=70.0, chain=False)  
Return a dictionary in which PDB identifiers are mapped to structure and alignment information.

Parameters

- percent_identity (float) – PDB hits with percent sequence identity equal to or higher than this value will be returned, default is 90.0
- percent_overlap (float) – PDB hits with percent coverage of the query sequence equivalent or better will be returned, default is 70.0
- chain (bool) – if chain is True, individual chains in a PDB file will be considered as separate hits, default is False

getParameters()  
Return parameters used in blast search.

getSequence()  
Return the query sequence that was used in the search.

blastPDB(sequence, filename=None, **kwargs)  
Return a PDBBlastRecord (page 148) instance that contains results from blast searching of Protein-DataBank database sequence using NCBI blastp.

Parameters

- sequence (str) – single-letter code amino acid sequence of the protein without any gap characters, all white spaces will be removed
- filename (str) – a filename to save the results in XML format

hitlist_size (default is 250) and expect (default is 1e-10) search parameters can be adjusted by the user. sleep keyword argument (default is 2 seconds) determines how long to wait to reconnect for results. Sleep time is doubled when results are not ready. timeout (default is 120s) determines when to give up waiting for the results.

3.7.12 Structure Comparison

This module defines functions for comparing and mapping polypeptide chains.

matchChains(atoms1, atoms2, **kwargs)  
Return pairs of chains matched based on sequence similarity. Makes an all-to-all comparison of chains in atoms1 and atoms2. Chains are obtained from hierarchical views (HierView (page 67)) of atom groups. This function returns a list of matching chains in a tuples that contain 4 items:

- matching chain from atoms1 as a AtomMap (page 45) instance,
- matching chain from atoms2 as a AtomMap (page 45) instance,
- percent sequence identity of the match,
- percent sequence overlap of the match.

324 http://docs.python.org/library/functions.html#float  
325 http://docs.python.org/library/functions.html#float  
326 http://docs.python.org/library/functions.html#bool  
327 http://docs.python.org/library/functions.html#str  
328 http://docs.python.org/library/functions.html#str
List of matches are sorted in decreasing percent sequence identity order. `AtomMap` (page 45) instances can be used to calculate RMSD values and superpose atom groups.

**Parameters**

- `atoms1` ([Chain](page 49), [AtomGroup](page 37), [Selection](page 86)) – atoms that contain a chain
- `atoms2` ([Chain](page 49), [AtomGroup](page 37), [Selection](page 86)) – atoms that contain a chain
- `subset` ([string](329)) – one of the following well-defined subsets of atoms: "calpha" (or "ca"), "backbone" (or "bb"), "heavy" (or "noh"), or "all", default is "calpha"
- `seqid` ([float](330)) – percent sequence identity, default is 90
- `overlap` ([float](331)) – percent overlap, default is 90
- `pwalign` ([bool](332)) – perform pairwise sequence alignment

If `subset` is set to `calpha` or `backbone`, only alpha carbon atoms or backbone atoms will be paired. If set to `all`, all atoms common to matched residues will be returned.

This function tries to match chains based on residue numbers and names. All chains in `atoms1` is compared to all chains in `atoms2`. This works well for different structures of the same protein. When it fails, `Bio.pairwise2` is used for pairwise sequence alignment, and matching is performed based on the sequence alignment. User can control, whether sequence alignment is performed or not with `pwalign` keyword. If `pwalign=True` is passed, pairwise alignment is enforced.

`matchAlign` (mobile, target, **kwargs)

Superpose `mobile` onto `target` based on best matching pair of chains. This function uses `matchChains()` (page 149) for matching chains and returns a tuple that contains the following items:

- `mobile` after it is superposed,
- matching chain from `mobile` as a `AtomMap` (page 45) instance,
- matching chain from `target` as a `AtomMap` (page 45) instance,
- percent sequence identity of the match,
- percent sequence overlap of the match.

**Parameters**

- `mobile` ([Chain](page 49), [AtomGroup](page 37), [Selection](page 86)) – atoms that contain a protein chain
- `target` ([Chain](page 49), [AtomGroup](page 37), [Selection](page 86)) – atoms that contain a protein chain
- `tarsel` ([str](333)) – `target` atoms that will be used for alignment, default is ‘calpha’
- `allcsets` ([bool](334)) – align all coordinate sets of `mobile`, default is `True`
- `seqid` ([float](335)) – percent sequence identity, default is 90

---

[329](http://docs.python.org/library/string.html#string)
[330](http://docs.python.org/library/functions.html#float)
[331](http://docs.python.org/library/functions.html#float)
[332](http://docs.python.org/library/functions.html#bool)
[333](http://docs.python.org/library/functions.html#str)
[334](http://docs.python.org/library/functions.html#bool)
[335](http://docs.python.org/library/functions.html#float)
- **overlap** *(float)* – percent overlap, default is 90
- **pwalign** *(bool)* – perform pairwise sequence alignment

**mapOntoChain** *(atoms, chain, **kwargs)*

Map *atoms* onto *chain*. This function returns a list of mappings. Each mapping is a tuple that contains 4 items:

- Mapped chain as an *AtomMap* (page 45) instance,
- *chain* as an *AtomMap* (page 45) instance,
- Percent sequence identity,
- Percent sequence overlap

Mappings are returned in decreasing percent sequence identity order. *AtomMap* (page 45) that keeps mapped atom indices contains dummy atoms in place of unmapped atoms.

**Parameters**

- **atoms** *(Chain (page 49), AtomGroup (page 37), Selection (page 86))* – atoms that will be mapped to the target *chain*
- **chain** *(Chain (page 49))* – chain to which atoms will be mapped
- **subset** *(string)* – one of the following well-defined subsets of atoms: "ca", "calpha", "backbone", "heavy", "noh", or "all", default is "calpha"
- **seqid** *(float)* – percent sequence identity, default is 90
- **overlap** *(float)* – percent overlap, default is 90
- **pwalign** *(bool)* – perform pairwise sequence alignment

This function tries to map *atoms* to *chain* based on residue numbers and types. Each individual chain in *atoms* is compared to target *chain*. This works well for different structures of the same protein. When it fails, *Bio.pairwise2* is used for sequence alignment, and mapping is performed based on the sequence alignment. User can control, whether sequence alignment is performed or not with *pwalign* keyword. If *pwalign=True* is passed, pairwise alignment is enforced.

**getMatchScore**()

Return match score used to align sequences.

**setMatchScore** *(match_score)*

Set match score used to align sequences.

**getMismatchScore**()

Return mismatch score used to align sequences.

**setMismatchScore** *(mismatch_score)*

Set mismatch score used to align sequences.

**getGapPenalty**()

Return gap opening penalty used for pairwise alignment.

---

336. [http://docs.python.org/library/functions.html#float](http://docs.python.org/library/functions.html#float)
337. [http://docs.python.org/library/functions.html#bool](http://docs.python.org/library/functions.html#bool)
338. [http://docs.python.org/library/string.html#string](http://docs.python.org/library/string.html#string)
339. [http://docs.python.org/library/functions.html#float](http://docs.python.org/library/functions.html#float)
340. [http://docs.python.org/library/functions.html#float](http://docs.python.org/library/functions.html#float)
341. [http://docs.python.org/library/functions.html#bool](http://docs.python.org/library/functions.html#bool)
setGapPenalty\( (\text{gap\_penalty}) \)
Set gap opening penalty used for pairwise alignment.

gapOpeningPenalty()
Return gap opening penalty used for pairwise alignment.

setGapExtPenalty\( (\text{gap\_ext\_penalty}) \)
Set gap extension penalty used for pairwise alignment.

gapExtPenalty()
Return gap extension penalty used for pairwise alignment.

setAlignmentMethod\( (\text{method}) \)
Set pairwise alignment method (global or local).

getAlignmentMethod()
Return pairwise alignment method.

#### 3.7.13 DSSP Tools

This module defines functions for executing DSSP program and parsing its output.

execDSSP\( (\text{pdb}, \text{outputname=\None}, \text{outputdir=\None}, \text{stderr=\True}) \)
Execute DSSP for given \text{pdb}. \text{pdb} can be a PDB identifier or a PDB file path. If \text{pdb} is a compressed file, it will be decompressed using Python \text{gzip} library. When no \text{outputname} is given, output name will be \text{pdb.dssp}. \text{dssp} extension will be appended automatically to \text{outputname}. If \text{outputdir} is given, DSSP output and uncompressed PDB file will be written into this folder. Upon successful execution of \text{dssp pdb > outputname} command, output filename is returned. On Linux platforms, when \text{stderr} is false, standard error messages are suppressed, i.e. \text{dssp pdb > outputname 2> /dev/null}.

For more information on DSSP see \url{http://swift.cmbi.ru.nl/gv/dssp/}. If you benefited from DSSP, please consider citing [WK83] (page 260).

parseDSSP\( (\text{dssp}, \text{ag}, \text{parseall=\False}) \)
Parse DSSP data from file \text{dssp} into \text{AtomGroup} instance \text{ag}. DSSP output file must be in the new format used from July 1995 and onwards. When \text{dssp} file is parsed, following attributes are added to \text{ag}:

- \text{dssp_resnum}: DSSP's sequential residue number, starting at the first residue actually in the data set and including chain breaks; this number is used to refer to residues throughout.
- \text{dssp_acc}: number of water molecules in contact with this residue *10. or residue water exposed surface in Angstrom^2.
- \text{dssp_kappa}: virtual bond angle (bend angle) defined by the three C\text{\alpha} atoms of residues I-2,I,I+2. Used to define bend (structure code \text{‘S’}).
- \text{dssp_alpha}: virtual torsion angle (dihedral angle) defined by the four C\text{\alpha} atoms of residues I-1,I+1,I+2. Used to define chirality (structure code \text{‘+’} or \text{‘-’}).
- \text{dssp_phi} and \text{dssp_psi}: IUPAC peptide backbone torsion angles

The following attributes are parsed when \text{parseall=True} is passed:

- \text{dssp_bp1}, \text{dssp_bp2}, and \text{dssp_sheet_label}: residue number of first and second bridge partner followed by one letter sheet label
- \text{dssp_tco}: cosine of angle between C=O of residue I and C=O of residue I-1. For \alpha\text{-}helices, TCO is near +1, for \beta\text{-}sheets TCO is near -1. Not used for structure definition.

[^342]: \url{http://docs.python.org/library/gzip.html#gzip}
• dssp\_NH\_O\_1\_index, dssp\_NH\_O\_1\_energy, etc.: hydrogen bonds; e.g. -3, -1.4 means: if this residue is residue i then N-H of I is h-bonded to C=O of I-3 with an electrostatic H-bond energy of -1.4 kcal/mol. There are two columns for each type of H-bond, to allow for bifurcated H-bonds.


performDSSP (pdb, parseall=False, stderr=True)
Perform DSSP calculations and parse results. DSSP data is returned in an AtomGroup (page 37) instance. See also execDSSP() (page 152) and parseDSSP() (page 152).

3.7.14 Miscellaneous Tools
This module defines miscellaneous functions dealing with protein data.

showProtein (*atoms, **kwargs)
Show protein representation using Axes3D(). This function is designed for generating a quick view of the contents of a AtomGroup (page 37) or Selection (page 86).

Protein atoms matching "calpha" selection are displayed using solid lines by picking a random and unique color per chain. Line width can be adjusted using lw argument, e.g. lw=12. Default width is 4. Chain colors can be overwritten using chain identifier as in A='green'.

Water molecule oxygen atoms are represented by red colored circles. Color can be changed using water keyword argument, e.g. water='aqua'. Water marker and size can be changed using wmarker and wsize keywords, default values are wmarker='.', wsize=6.

Hetero atoms matching "hetero and noh" selection are represented by circles and unique colors are picked at random on a per residue basis. Colors can be customized using residue name as in NAH='purple'. Note that this will color all distinct residues with the same name in the same color. Hetero atom marker and size can be changed using hmarker and hsize keywords, default values are hmarker='o', hsize=6.

ProDy will set the size of axis so the representation is not distorted when the shape of figure window is close to a square. Colors are picked at random, except for water oxygens which will always be colored red.

writePQR (filename, atoms)
Write atoms in PQR format to a file with name filename. Only current coordinate set is written. Returns filename upon success. If filename ends with .gz, a compressed file will be written.

3.7.15 PDB File Header
This module defines functions for parsing header data from PDB files.

class Chemical (resname)
A data structure for storing information on chemical components (or heterogens) in PDB structures.

A Chemical (page 153) instance has the following attributes:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description (RECORD TYPE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>resname</td>
<td>str</td>
<td>residue name (or chemical component identifier) (HET)</td>
</tr>
<tr>
<td>name</td>
<td>str</td>
<td>chemical name (HETNAM)</td>
</tr>
<tr>
<td>chain</td>
<td>str</td>
<td>chain identifier (HET)</td>
</tr>
<tr>
<td>resnum</td>
<td>int</td>
<td>residue (or sequence) number (HET)</td>
</tr>
<tr>
<td>icode</td>
<td>str</td>
<td>insertion code (HET)</td>
</tr>
<tr>
<td>natoms</td>
<td>int</td>
<td>number of atoms present in the structure (HET)</td>
</tr>
<tr>
<td>description</td>
<td>str</td>
<td>description of the chemical component (HET)</td>
</tr>
<tr>
<td>synonyms</td>
<td>list</td>
<td>synonyms (HETSYN)</td>
</tr>
<tr>
<td>formula</td>
<td>str</td>
<td>chemical formula (FORMUL)</td>
</tr>
<tr>
<td>pdbentry</td>
<td>str</td>
<td>PDB entry that chemical data is extracted from</td>
</tr>
</tbody>
</table>

Chemical class instances can be obtained as follows:

```python
In [1]: from prody import *

In [2]: chemical = parsePDBHeader('1zz2', 'chemicals')[0]

In [3]: chemical
Out[3]: <Chemical: B11 (1ZZ2_A_362)>

In [4]: chemical.name
Out[4]: 'N-[3-(4-FLUOROPHENOXY)PHENYL]-4-[(2-HYDROXYBENZYL)AMINO]PIPERIDINE-1-SULFONAMIDE'

In [5]: chemical.natoms
Out[5]: 33

In [6]: len(chemical)
Out[6]: 33
```

- `chain`: chain identifier
- `description`: description of the chemical component
- `formula`: chemical formula
- `icode`: insertion code
- `name`: chemical name
- `natoms`: number of atoms present in the structure
- `pdbentry`: PDB entry that chemical data is extracted from
- `resname`: residue name (or chemical component identifier)
- `resnum`: residue (or sequence) number
- `synonyms`: list of synonyms

3.7. Protein Structure
class **Polymer** *(chid)*

A data structure for storing information on polymer components (protein or nucleic) of PDB structures.

A **Polymer** (page 154) instance has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description (RECORD TYPE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>chid</td>
<td>str</td>
<td>chain identifier</td>
</tr>
<tr>
<td>name</td>
<td>str</td>
<td>name of the polymer (macromolecule) (COMPND)</td>
</tr>
<tr>
<td>fragment</td>
<td>str</td>
<td>specifies a domain or region of the molecule (COMPND)</td>
</tr>
<tr>
<td>synonyms</td>
<td>list</td>
<td>synonyms for the polymer (COMPND)</td>
</tr>
<tr>
<td>ec</td>
<td>list</td>
<td>associated Enzyme Commission numbers (COMPND)</td>
</tr>
<tr>
<td>engineered</td>
<td>bool</td>
<td>indicates that the polymer was produced using recombinant technology or by purely chemical synthesis (COMPND)</td>
</tr>
<tr>
<td>mutation</td>
<td>bool</td>
<td>indicates presence of a mutation (COMPND)</td>
</tr>
<tr>
<td>comments</td>
<td>str</td>
<td>additional comments</td>
</tr>
<tr>
<td>sequence</td>
<td>str</td>
<td>polymer chain sequence (SEQRES)</td>
</tr>
<tr>
<td>dbrefs</td>
<td>list</td>
<td>sequence database records (DBREF[1</td>
</tr>
<tr>
<td>modified</td>
<td>list</td>
<td>modified residues (SEQMOD) when modified residues are present, each will be represented as: (resname, resnum, icode, stdname, comment)</td>
</tr>
<tr>
<td>pdbentry</td>
<td>str</td>
<td>PDB entry that polymer data is extracted from</td>
</tr>
</tbody>
</table>

Polymer class instances can be obtained as follows:

```python
In [7]: polymer = parsePDBHeader('2k39', 'polymers')[0]

In [8]: polymer
Out[8]: <Polymer: UBIQUITIN (2K39_A)>

In [9]: polymer.pdbentry
Out[9]: '2K39'

In [10]: polymer.chid
Out[10]: 'A'

In [11]: polymer.name
Out[11]: 'UBIQUITIN'

In [12]: polymer.sequence
```

3.7. Protein Structure
Out[12]: ‘MQIFVKTGTITLEVEPSDTIENVKAKIQDEGIPPDQQLFAGKQLEDGRTLSHYENIQKESTLHLVLRLRGG’

In [13]: len(polymer.sequence)
Out[13]: 76

In [14]: len(polymer)
Out[14]: 76

In [15]: dbref = polymer.dbrefs[0]

In [16]: dbref.database
Out[16]: ‘UniProt’

In [17]: dbref.accesion
Out[17]: ‘P62972’

In [18]: dbref.idcode
Out[18]: ‘UBIQ_XENLA’

<table>
<thead>
<tr>
<th>chid</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain identifier</td>
</tr>
</tbody>
</table>

| comments |
| additional comments |

| dbrefs |
| sequence database reference records |

| ec |
| list of associated Enzyme Commission numbers |

| engineered |
| indicates that the molecule was produced using recombinant technology or by purely chemical synthesis |

| fragment |
| specifies a domain or region of the molecule |

| modified |
| modified residues |

| mutation |
| indicates presence of a mutation |

| name |
| name of the polymer (macromolecule) |

| pdbentry |
| PDB entry that polymer data is extracted from |

| sequence |
| polymer chain sequence |

| synonyms |
| list of synonyms for the molecule |

**class DBRef**

A data structure for storing reference to sequence databases for polymer components in PDB structures. Information if parsed from DBREF[1|2] and SEQADV records in PDB header.

| accession |
| database accession code |
database
    sequence database, one of UniProt, GenBank, Norine, UNIMES, or PDB

dbabbr
    database abbreviation, one of UNP, GB, NORINE, UNIMES, or PDB

diff
    list of differences between PDB and database sequences, (resname, resnum, icode,
    dbResname, dbResnum, comment)

first
    initial residue numbers, (resnum, icode, dbnum)

idcode
    database identification code, i.e. entry name in UniProt

last
    ending residue numbers, (resnum, icode, dbnum)

parsePDBHeader (pdb, *keys)
    Return header data dictionary for pdb. This function is equivalent to
    parsePDB(pdb, header=True, model=0, meta=False), likewise pdb may
    be an identifier or a filename.

    List of header records that are parsed.
<table>
<thead>
<tr>
<th>Record type</th>
<th>Dictionary key(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEADER</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>classification</td>
<td>molecule classification</td>
</tr>
<tr>
<td></td>
<td>deposition_date</td>
<td>deposition date</td>
</tr>
<tr>
<td></td>
<td>identifier</td>
<td>PDB identifier</td>
</tr>
<tr>
<td>TITLE</td>
<td>title</td>
<td>title for the experiment or analysis</td>
</tr>
<tr>
<td>SPLIT</td>
<td>split</td>
<td>list of PDB entries that make up the whole structure when combined with this one</td>
</tr>
<tr>
<td>COMPND</td>
<td>polymers</td>
<td>see Polymer (page 154)</td>
</tr>
<tr>
<td>EXPDTA</td>
<td>experiment</td>
<td>information about the experiment</td>
</tr>
<tr>
<td>NUMMDL</td>
<td>n_models</td>
<td>number of models</td>
</tr>
<tr>
<td>MDLTYP</td>
<td>model_type</td>
<td>additional structural annotation</td>
</tr>
<tr>
<td>AUTHOR</td>
<td>authors</td>
<td>list of contributors</td>
</tr>
<tr>
<td>JRNL</td>
<td>reference</td>
<td>reference information dictionary:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authors: list of authors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• title: title of the article</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• editors: list of editors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• issn:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• reference: journal, vol, issue, etc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• publisher: publisher information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• pmid: pubmed identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• doi: digital object identifier</td>
</tr>
<tr>
<td>DBREF[1</td>
<td>2]</td>
<td>polymers</td>
</tr>
<tr>
<td>SEQADV</td>
<td>polymers</td>
<td>see Polymer (page 154)</td>
</tr>
<tr>
<td>SEQRES</td>
<td>polymers</td>
<td>see Polymer (page 154)</td>
</tr>
<tr>
<td>MODRES</td>
<td>polymers</td>
<td>see Polymer (page 154)</td>
</tr>
<tr>
<td>HELIX</td>
<td>polymers</td>
<td>see Polymer (page 154)</td>
</tr>
<tr>
<td>SHEET</td>
<td>polymers</td>
<td>see Polymer (page 154)</td>
</tr>
<tr>
<td>HET</td>
<td>chemicals</td>
<td>see Chemical (page 153)</td>
</tr>
<tr>
<td>HETNAM</td>
<td>chemicals</td>
<td>see Chemical (page 153)</td>
</tr>
<tr>
<td>HETSYN</td>
<td>chemicals</td>
<td>see Chemical (page 153)</td>
</tr>
<tr>
<td>FORMUL</td>
<td>chemicals</td>
<td>see Chemical (page 153)</td>
</tr>
<tr>
<td>REMARK 2</td>
<td>resolution</td>
<td>resolution of structures, when applicable</td>
</tr>
<tr>
<td>REMARK 4</td>
<td>version</td>
<td>PDB file version</td>
</tr>
<tr>
<td>REMARK 350</td>
<td>biomoltrans</td>
<td>biomolecular transformation lines (unprocessed)</td>
</tr>
</tbody>
</table>

Header records that are not parsed are: OBSLTE, CAVEAT, SOURCE, KEYWDS, REVDAT, SPRSDE, SSBOND, LINK, CISPEP, CRYS1, ORIGX1, ORIGX2, ORIGX3, MTRIX1, MTRIX2, MTRIX3, and REMARK X not mentioned above.
**assignSecstr** *(header, atoms, coil=False)*

Assign secondary structure from header dictionary to atoms. header must be a dictionary parsed using the `parsePDB()` (page 162). atoms may be an instance of `AtomGroup` (page 37), `Selection` (page 86), `Chain` (page 49) or `Residue` (page 69). ProDy can be configured to automatically parse and assign secondary structure information using `confProDy(auto_secondary=True)` command. See also `confProDy()` (page 204) function.

The Dictionary of Protein Secondary Structure, in short DSSP, type single letter code assignments are used:

- **G** = 3-turn helix (310 helix). Min length 3 residues.
- **H** = 4-turn helix (alpha helix). Min length 4 residues.
- **I** = 5-turn helix (pi helix). Min length 5 residues.
- **T** = hydrogen bonded turn (3, 4 or 5 turn)
- **E** = extended strand in parallel and/or anti-parallel beta-sheet conformation. Min length 2 residues.
- **B** = residue in isolated beta-bridge (single pair beta-sheet hydrogen bond formation)
- **S** = bend (the only non-hydrogen-bond based assignment).
- **C** = residues not in one of above conformations.

See [http://en.wikipedia.org/wiki/Protein_secondary_structure#The_DSSP_code](http://en.wikipedia.org/wiki/Protein_secondary_structure#The_DSSP_code) for more details.

Following PDB helix classes are omitted:

- Right-handed omega (2, class number)
- Right-handed gamma (4)
- Left-handed alpha (6)
- Left-handed omega (7)
- Left-handed gamma (8)
- 2-7 ribbon/helix (9)
- Polyproline (10)

Secondary structures are assigned to all atoms in a residue. Amino acid residues without any secondary structure assignments in the header section will be assigned coil (C) conformation. This can be prevented by passing `coil=False` argument.

**buildBiomolecules** *(header, atoms, biomol=None)*

Return atoms after applying biomolecular transformations from header dictionary. Biomolecular transformations are applied to all coordinate sets in the molecule.

Some PDB files contain transformations for more than 1 biomolecules. A specific set of transformations can be chosen using `biomol` argument. Transformation sets are identified by numbers, e.g. "1", "2", ...

If multiple biomolecular transformations are provided in the header dictionary, biomolecules will be returned as `AtomGroup` (page 37) instances in a `list()`.

If the resulting biomolecule has more than 26 chains, the molecular assembly will be split into multiple `AtomGroup` (page 37) instances each containing at most 26 chains. These `AtomGroup` (page 37) instances will be returned in a tuple.

---

343 [http://docs.python.org/library/functions.html#list](http://docs.python.org/library/functions.html#list)
Note that atoms in biomolecules are ordered according to chain identifiers.

### 3.7.16 Local PDB Handlers

This module defines functions for handling local PDB folders.

**pathPDBFolder (folder=None, divided=False)**

Return or specify local PDB folder for storing PDB files downloaded from wwPDB servers. Files stored in this folder can be accessed via `fetchPDB()` (page 160) from any working directory. To release the current folder, pass an invalid path, e.g. `folder=""`.

If `divided` is `True`, the divided folder structure of wwPDB servers will be assumed when reading from and writing to the local folder. For example, a structure with identifier `1XYZ` will be present as `pdblocalfolder/yz/pdb1xyz.pdb.gz`.

If `divided` is `False`, a plain folder structure will be expected and adopted when saving files. For example, the same structure will be present as `pdblocalfolder/1xyz.pdb.gz`.

Finally, in either case, lower case letters will be used and compressed files will be stored.

**pathPDBMirror (path=None, format=None)**

Return or specify PDB mirror path to be used by `fetchPDB()` (page 160). To release the current mirror, pass an invalid path, e.g. `path=""`. If you are keeping a partial mirror, such as PDB files in `/data/structures/divided/pdb/` folder, specify `format`, which is `’pdb’` in this case.

**fetchPDB (*pdb, **kwargs)**

Return path(s) to PDB file(s) for specified `pdb` identifier(s). Files will be sought in user specified `folder` or current working director, and then in local PDB folder and mirror, if they are available. If `copy` is set `True`, files will be copied into `folder`. If `compressed` is `False`, all files will be decompressed. See `pathPDBFolder()` (page 160) and `pathPDBMirror()` (page 160) for managing local resources, `fetchPDBviaFTP()` (page 165) and `fetchPDBviaFTP()` (page 165) for downloading files from PDB servers.

**fetchPDBfromMirror (*pdb, **kwargs)**

Return path(s) to PDB (default), PDBML, or mmCIF file(s) for specified `pdb` identifier(s). If a `folder` is specified, files will be copied into this folder. If `compressed` is `False`, files will decompressed. `format` argument can be used to get PDBML and mmCIF files: `format=’cif’` will fetch an mmCIF file, and `format=’xml’` will fetch a PDBML file. If PDBML header file is desired, `noatom=True` argument will do the job.

**iterPDBFilenames (path=None, sort=False, unique=True, **kwargs)**

Yield PDB filenames in `path` specified by the user or in local PDB mirror (see `pathPDBMirror()` (page 160)). When `unique` is `True`, files one of potentially identical files will be yielded (e.g. `/pdb1mkp.ent.gz1` and `/pdb1mkp.pdb.gz`). .pdb and .ent extensions, and compressed files are considered.

**findPDBFiles (path, case=None, **kwargs)**

Return a dictionary that maps PDB filenames to file paths. If `case` is specified (‘u[pper]’ or ‘l[ower]’), dictionary keys (filenames) will be modified accordingly. If a PDB filename has pdb prefix, it will be trimmed, for example ‘1lmkp’ will be mapped to file path `.pdb1lkmkp.ent.gz1`). .pdb and .ent extensions, and compressed files are considered.

---

344 http://www.wwpdb.org/
345 http://pdbml.pdb.org/
346 http://mmcif.pdb.org/
3.7.17 PDB Sequence Clusters

This module defines functions for handling PDB sequence clusters.

**fetchPDBClusters**(sqid=None)

Retrieve PDB sequence clusters. PDB sequence clusters are results of the weekly clustering of protein chains in the PDB generated by blastclust. They are available at FTP site: ftp://resources.rcsb.org/sequence/clusters/

This function will download about 10 Mb of data and save it after compressing in your home directory in .prody/pdbclusters. Compressed files will be less than 4 Mb in size. Cluster data can be loaded using **loadPDBClusters()**(page 161) function and be accessed using **listPDBCluster()**(page 161).

**loadPDBClusters**(sqid=None)

Load previously fetched PDB sequence clusters from disk to memory.

**listPDBCluster**(pdb, ch, sqid=95)

Return the PDB sequence cluster that contains chain ch in structure pdb for sequence identity level sqid. PDB sequence cluster will be returned in as a list of tuples, e.g. [('1XXX', 'A'), ]. Note that PDB clusters individual chains, so the same PDB identifier may appear twice in the same cluster if the corresponding chain is present in the structure twice.

Before this function is used, **fetchPDBClusters()**(page 161) needs to be called. This function will load the PDB sequence clusters for sqid automatically using **loadPDBClusters()**(page 161).

3.7.18 PDB File

This module defines functions for parsing and writing PDB files.

**parsePDBStream**(stream,**kwargs)

Return an AtomGroup (page 37) and/or dictionary containing header data parsed from a stream of PDB lines.

Parameters

- **stream** – Anything that implements the method readlines (e.g. file, buffer, stdin)
- **title**(str) – title of the AtomGroup (page 37) instance, default is the PDB filename or PDB identifier
- **ag**(AtomGroup (page 37)) – AtomGroup (page 37) instance for storing data parsed from PDB file, number of atoms in ag and number of atoms parsed from the PDB file must be the same and atoms in ag and those in PDB file must be in the same order. Non-coordinate data stored in ag will be overwritten with those parsed from the file.
- **chain**(str) – chain identifiers for parsing specific chains, e.g. chain='A', chain='B', chain='DE', by default all chains are parsed
- **subset**(str) – a predefined keyword to parse subset of atoms, valid keywords are 'calpha' ('ca'), 'backbone' ('bb'), or None (read all atoms), e.g. subset='bb'

3.7. Protein Structure 161
• **model** (*int, list*) – model index or None (read all models), e.g. `model=10`

• **header** (*bool*[^351]) – if True PDB header content will be parsed and returned

• **altloc** (*str*[^352]) – if a location indicator is passed, such as ‘A’ or ‘B’, only indicated alternate locations will be parsed as the single coordinate set of the AtomGroup, if `altloc` is set True all alternate locations will be parsed and each will be appended as a distinct coordinate set, default is "A"

• **biomol** (*False*[^353]) – if True, biomolecule obtained by transforming the coordinates using information from header section will be returned

• **secondary** (*False*[^354]) – if True, secondary structure information from header section will be assigned atoms

If `model=0` and `header=True`, return header dictionary only.

Note that this function does not evaluate CONECT records.

**parsePDB** (*pdb*, **kwargs)

Return an AtomGroup (page 37) and/or dictionary containing header data parsed from a PDB file.

This function extends parsePDBStream() (page 161).

See Parse PDB files[^355] for a detailed usage example.

**Parameters**

• **pdb** – a PDB identifier or a filename If needed, PDB files are downloaded using `fetchPDB()` (page 160) function.

• **title** (*str*[^356]) – title of the AtomGroup (page 37) instance, default is the PDB filename or PDB identifier

• **ag** (*AtomGroup* (page 37)) – AtomGroup (page 37) instance for storing data parsed from PDB file, number of atoms in `ag` and number of atoms parsed from the PDB file must be the same and atoms in `ag` and those in PDB file must be in the same order. Non-coordinate data stored in `ag` will be overwritten with those parsed from the file.

• **chain** (*str*[^357]) – chain identifiers for parsing specific chains, e.g. `chain='A'`, `chain='B'`, `chain='DE'`, by default all chains are parsed

• **subset** (*str*[^358]) – a predefined keyword to parse subset of atoms, valid keywords are ‘calpha’ (‘ca’), ‘backbone’ (‘bb’), or None (read all atoms), e.g. `subset='bb'`

• **model** (*int, list*) – model index or None (read all models), e.g. `model=10`

• **header** (*bool*[^359]) – if True PDB header content will be parsed and returned

• **altloc** (*str*[^360]) – if a location indicator is passed, such as ‘A’ or ‘B’, only indicated alternate locations will be parsed as the single coordinate set of the AtomGroup, if

[^351]: http://docs.python.org/library/functions.html#bool
[^352]: http://docs.python.org/library/functions.html#str
[^353]: http://docs.python.org/library/constants.html#False
[^354]: http://docs.python.org/library/constants.html#False
[^355]: http://prody.csb.pitt.edu/tutorials/structure_analysis/pdbfiles.html#parsepdb
[^356]: http://docs.python.org/library/functions.html#str
[^357]: http://docs.python.org/library/functions.html#str
[^358]: http://docs.python.org/library/functions.html#str
[^359]: http://docs.python.org/library/functions.html#bool
[^360]: http://docs.python.org/library/functions.html#str
allocation is set `True` all alternate locations will be parsed and each will be appended as a distinct coordinate set, default is "A"

- **biomol** (`False`\(^{361}\)) – if `True`, biomolecule obtained by transforming the coordinates using information from header section will be returned
- **secondary** (`False`\(^{362}\)) – if `True`, secondary structure information from header section will be assigned atoms

If `model=0` and `header=True`, return header dictionary only.

Note that this function does not evaluate CONECT records.

### parsePQR(filename, **kwargs)

Return an `AtomGroup` (page 37) containing data parsed from PDB lines.

#### Parameters

- **filename** (`str`\(^{363}\)) – a PQR filename
- **title** (`str`\(^{364}\)) – title of the `AtomGroup` (page 37) instance, default is the PDB filename or PDB identifier
- **ag** (`AtomGroup` (page 37)) – `AtomGroup` (page 37) instance for storing data parsed from PDB file, number of atoms in `ag` and number of atoms parsed from the PDB file must be the same and atoms in `ag` and those in PDB file must be in the same order. Non-coordinate data stored in `ag` will be overwritten with those parsed from the file.
- **chain** (`str`\(^{365}\)) – chain identifiers for parsing specific chains, e.g. `chain='A'`, `chain='B'`, `chain='DE'`, by default all chains are parsed
- **subset** (`str`\(^{366}\)) – a predefined keyword to parse subset of atoms, valid keywords are `calpha` (‘ca’), `backbone` (‘bb’), or `None` (read all atoms), e.g. `subset='bb'`

### writePDBStream(stream, atoms, csets=None, **kwargs)

Write `atoms` in PDB format to a stream.

#### Parameters

- **stream** – anything that implements a `write()` method (e.g. file, buffer, stdout)
- **atoms** – an object with atom and coordinate data
- **csets** – coordinate set indices, default is all coordinate sets
- **beta** – a list or array of number to be outputted in beta column
- **occupancy** – a list or array of number to be outputted in occupancy column

### writePDB(filename, atoms, csets=None, autoext=True, **kwargs)

Write `atoms` in PDB format to a file with name `filename` and return `filename`. If `filename` ends with `.gz`, a compressed file will be written.

#### Parameters

- **atoms** – an object with atom and coordinate data

---

\(^{360}\)http://docs.python.org/library/constants.html#False
\(^{361}\)http://docs.python.org/library/constants.html#False
\(^{362}\)http://docs.python.org/library/constants.html#False
\(^{363}\)http://docs.python.org/library/functions.html#str
\(^{364}\)http://docs.python.org/library/functions.html#str
\(^{365}\)http://docs.python.org/library/functions.html#str
\(^{366}\)http://docs.python.org/library/functions.html#str

3.7. Protein Structure 163
• csets – coordinate set indices, default is all coordinate sets
• beta – a list or array of number to be outputted in beta column
• occupancy – a list or array of number to be outputted in occupancy column
• autoext – when not present, append extension .pdb to filename

3.7.19 PDB Ligands

This module defines functions for fetching PDB ligand data.

```python
fetchPDBLigand(cci, filename=None)
```

Fetch PDB ligand data from PDB for chemical component cci. cci may be 3-letter chemical component identifier or a valid XML filename. If filename is given, XML file will be saved with that name.

If you query ligand data frequently, you may configure ProDy to save XML files in your computer. Set ligand_xml_save option True, i.e. confProDy(ligand_xml_save=True). Compressed XML files will be save to ProDy package folder, e.g. /home/user/.prody/pdbligands. Each file is around 5Kb when compressed.

This function is compatible with PDBx/PDBML v 4.0.

Ligand data is returned in a dictionary. Ligand coordinate atom data with model and ideal coordinate sets are also stored in this dictionary. Note that this dictionary will contain data that is present in the XML file and all Ligand Expo XML files do not contain every possible data field. So, it may be better if you use dict.get() instead of indexing the dictionary, e.g. to retrieve formula weight (or relative molar mass) of the chemical component use data.get('formula_weight') instead of data['formula_weight'] to avoid exceptions when this data field is not found in the XML file. URL and/or path of the XML file are returned in the dictionary with keys url and path, respectively.

Following example downloads data for ligand STI (a.k.a. Gleevec and Imatinib) and calculates RMSD between model (X-ray structure 1IEP) and ideal (energy minimized) coordinate sets:

```python
In [1]: from prody import *

In [2]: ligand_data = fetchPDBLigand('STI')

In [3]: ligand_data['model_coordinates_db_code']
Out[3]: '1IEP'

In [4]: ligand_model = ligand_data['model']

In [5]: ligand_ideal = ligand_data['ideal']

In [6]: transformation = superpose(ligand_ideal.noh, ligand_model.noh)

In [7]: calcRMSD(ligand_ideal.noh, ligand_model.noh)
Out[7]: 2.2678638214526528
```

3.7.20 Stride Tools

This module defines functions for executing STRIDE program and parsing its output.

---

3.7. Protein Structure

---
execSTRIDE (pdb, outputname=None, outputdir=None)

Execute STRIDE program for given pdb. pdb can be an identifier or a PDB file path. If pdb is a compressed file, it will be decompressed using Python gzip library. When no outputname is given, output name will be pdb.stride. stride extension will be appended automatically to outputname. If outputdir is given, STRIDE output and uncompressed PDB file will be written into this folder. Upon successful execution of stride pdb > out command, output filename is returned.

For more information on STRIDE see http://webclu.bio.wzw.tum.de/stride/. If you benefited from STRIDE, please consider citing [DF95] (page 260).

parseSTRIDE (stride, ag)

Parse STRIDE output from file stride into AtomGroup (page 37) instance ag. STRIDE output file must be in the new format used from July 1995 and onwards. When stride file is parsed, following attributes are added to ag:

• **stride_resnum**: STRIDE’s sequential residue number, starting at the first residue actually in the data set.
• **stride_phi, stride_psi**: peptide backbone torsion angles phi and psi
• **stride_area**: residue solvent accessible area

performSTRIDE (pdb)

Perform STRIDE calculations and parse results. STRIDE data is returned in an AtomGroup (page 37) instance. See also execSTRIDE() (page 164) and parseSTRIDE() (page 165).

### 3.7.21 wwPDB Tools

This module defines functions for accessing wwPDB servers.

wwPDBServer (*key*)

Set/get wwPDB FTP/HTTP server location used for downloading PDB structures. Use one of the following keywords for setting a server:

<table>
<thead>
<tr>
<th>wwPDB FTP server</th>
<th>Key (case insensitive)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCSB PDB (USA) (default)</td>
<td>RCSB, USA, US</td>
</tr>
<tr>
<td>PDBe (Europe)</td>
<td>PDBe, Europe, Euro, EU</td>
</tr>
<tr>
<td>PDBj (Japan)</td>
<td>PDBj, Japan, Jp</td>
</tr>
</tbody>
</table>

fetchPDBviaFTP (*pdb, **kwargs*)

Retrieve PDB (default), PDBML, or mmCIF file(s) for specified pdb identifier(s) and return path(s). Downloaded files will be stored in local PDB folder, if one is set using pathPDBFolder(), and copied into folder, if specified by the user. If no destination folder is specified, files will be saved in the current working directory. If compressed is False, decompressed files will be copied into folder. format keyword argument can be used to retrieve PDBML and mmCIF files: format='cif' will fetch an mmCIF file, and format='xml' will fetch a PDBML file. If PDBML header file is desired, noatom=True argument will do the job.

fetchPDBviaHTTP (*pdb, **kwargs*)

Retrieve PDB file(s) for specified pdb identifier(s) and return path(s). Downloaded files will be stored in local PDB folder, if one is set using pathPDBFolder(), and copied into folder, if specified by the user. If no destination folder is specified, files will be saved in the current working directory. If compressed is False, decompressed files will be copied into folder.

---

369http://docs.python.org/library/gzip.html#gzip
370http://www.wwpdb.org/
371http://pdbml.pdb.org/
372http://mmcif.pdb.org/
3.8 Sequence Analysis

This module contains features for analyzing protein sequences.

3.8.1 Classes

- **MSA** (page 169) - store MSA data indexed by label
- **Sequence** (page 173) - store sequence data

3.8.2 MSA IO

- **MSAFile** (page 171) - read/write MSA files in FASTA/SELEX/Stockholm formats
- **parseMSA()** (page 172) - parse MSA files
- **writeMSA()** (page 172) - write MSA files

3.8.3 Editing

- **mergeMSA()** (page 171) - merge MSA data for multi-domain proteins
- **refineMSA()** (page 170) - refine MSA by removing gapped columns and/or sequences

3.8.4 Analysis

- **calcMSAOccupancy()** (page 167) - calculate row (sequence) or column occupancy
- **calcShannonEntropy()** (page 167) - calculate Shannon entropy
- **buildMutinfoMatrix()** (page 167) - build mutual information matrix
- **buildOMESMatrix()** (page 168) - build mutual observed minus expected squared covariance matrix
- **buildSCAMatrix()** (page 169) - build statistical coupling analysis matrix
- **buildSeqidMatrix()** (page 168) - build sequence identity matrix
- **buildDirectInfoMatrix()** (page 169) - build direct information matrix
- **uniqueSequences()** (page 168) - select unique sequences
- **applyMutinfoCorr()** (page 167) - apply correction to mutual information matrix
- **applyMutinfoNorm()** (page 167) - apply normalization to mutual information matrix
- **calcMeff()** (page 169) - calculate sequence weights
- **calcRankorder()** (page 168) - rank order scores

3.8.5 Plotting

- **showShannonEntropy()** (page 172) - plot Shannon entropy
- **showMSAOccupancy()** (page 172) - plot row (sequence) or column occupancy
- **showMutinfoMatrix()** (page 173) - show mutual information matrix
3.8.6 Analysis Functions

This module defines MSA analysis functions.

*calcShannonEntropy* (*msa, ambiguity=True, omitgaps=True, **kwargs*)

Return Shannon entropy array calculated for *msa*, which may be an MSA (page 169) instance or a 2D Numpy character array. Implementation is case insensitive and handles ambiguous amino acids as follows:

- **B** (Asx) count is allocated to **D** (Asp) and **N** (Asn)
- **Z** (Glx) count is allocated to **E** (Glu) and **Q** (Gln)
- **J** (Xle) count is allocated to **I** (Ile) and **L** (Leu)
- **X** (Xaa) count is allocated to the twenty standard amino acids

Selenocysteine (**U**, Sec) and pyrrolysine (**O**, Pyl) are considered as distinct amino acids. When *ambiguity* is set *False*, all alphabet characters are considered as distinct types.

All non-alphabet characters are considered as gaps, and they are handled in two ways:

- non-existent, the probability of observing amino acids in a given column is adjusted, by default
- as a distinct character with its own probability, when *omitgaps* is *False*

*buildMutinfoMatrix* (*msa, ambiguity=True, turbo=True, **kwargs*)

Return mutual information matrix calculated for *msa*, which may be an MSA (page 169) instance or a 2D Numpy character array. Implementation is case insensitive and handles ambiguous amino acids as follows:

- **B** (Asx) count is allocated to **D** (Asp) and **N** (Asn)
- **Z** (Glx) count is allocated to **E** (Glu) and **Q** (Gln)
- **J** (Xle) count is allocated to **I** (Ile) and **L** (Leu)
- **X** (Xaa) count is allocated to the twenty standard amino acids

- Joint probability of observing a pair of ambiguous amino acids is allocated to all potential combinations, e.g. probability of **XX** is allocated to 400 combinations of standard amino acids, similarly probability of **XB** is allocated to 40 combinations of **D** and **N** with the standard amino acids.

Selenocysteine (**U**, Sec) and pyrrolysine (**O**, Pyl) are considered as distinct amino acids. When *ambiguity* is set *False*, all alphabet characters are considered as distinct types. All non-alphabet characters are considered as gaps.

Mutual information matrix can be normalized or corrected using *applyMINormalization()* and *applyMICorrection()* methods, respectively. Normalization by joint entropy can performed using this function with *norm* option set *True*.

By default, *turbo* mode, which uses memory as large as the MSA array itself but runs four to five times faster, will be used. If memory allocation fails, the implementation will fall back to slower and memory efficient mode.

*calcMSAOccupancy* (*msa, occ='res', count=False*)

Return occupancy array calculated for residue positions (default, *'res'* or *'col'* for *occ*) or sequences (*'seq'* or *'row'* for *occ*) of *msa*, which may be an MSA (page 169) instance or a 2D NumPy character array. By default, occupancy [0-1] will be calculated. If *count* is *True*, count of non-gap characters will be returned. Implementation is case insensitive.

*applyMutinfoCorr* (*mutinfo, corr='prod'*)

Return a copy of *mutinfo* array after average product correction (default) or average sum correction is applied. See [DSD08] (page 260) for details.
applyMutinfoNorm\((\text{mutinfo}, \text{entropy}, \text{norm}='\text{sument}')\)
Apply one of the normalizations discussed in [MLC05] (page 261) to mutinfo matrix. norm can be one of the following:

- **'sument':** \(H(X) + H(Y)\), sum of entropy of columns
- **'minent':** \(\min\{H(X), H(Y)\}\), minimum entropy
- **'maxent':** \(\max\{H(X), H(Y)\}\), maximum entropy
- **'mincon':** \(\min\{H(X|Y), H(Y|X)\}\), minimum conditional entropy
- **'maxcon':** \(\max\{H(X|Y), H(Y|X)\}\), maximum conditional entropy

where \(H(X)\) is the entropy of a column, and \(H(X|Y) = H(X) - MI(X,Y)\). Normalization with joint entropy, i.e. \(H(X,Y)\), can be done using buildMutinfoMatrix() (page 167) norm argument.

calcRankorder\((\text{matrix}, \text{zscore=False, **kwargs})\)
Returns indices of elements and corresponding values sorted in descending order, if descend is True (default). Can apply a zscore normalization; by default along axis - 0 such that each column has mean=0 and std=1. If zcore analysis is used, return value contains the zscores. If matrix is symmetric only lower triangle indices will be returned, with diagonal elements if diag is True (default).

buildSeqidMatrix\((\text{msa}, \text{turbo=True})\)
Return sequence identity matrix for msa.

By default, turbo mode, which uses memory as large as the MSA array itself but runs four to five times faster, will be used. If memory allocation fails, the implementation will fall back to slower and memory efficient mode.

uniqueSequences\((\text{msa}, \text{seqid=0.98, turbo=True})\)
Return a boolean array marking unique sequences in msa. A sequence sharing sequence identity of sqid or more with another sequence coming before itself in msa will have a False value in the array.

By default, turbo mode, which uses memory as large as the MSA array itself but runs four to five times faster, will be used. If memory allocation fails, the implementation will fall back to slower and memory efficient mode.

buildOMESMatrix\((\text{msa}, \text{ambiguity=True, turbo=True, **kwargs})\)
Return OMES (Observed Minus Expected Squared) covariance matrix calculated for msa, which may be an MSA (page 169) instance or a 2D NumPy character array. OMES is defined as:

\[
\text{OMES}_{ij} = \frac{(N_{\text{OBS}} - N_{\text{EX}})^2}{N_{\text{EX}}} \cdot \left(\frac{f_{ij} - f_{i} \cdot f_{j}}{f_{i} \cdot f_{j}}\right)^2
\]

Implementation is case insensitive and handles ambiguous amino acids as follows:

- **B** (Asx) count is allocated to **D** (Asp) and **N** (Asn)
- **Z** (Glx) count is allocated to **E** (Glu) and **Q** (Gln)
- **J** (Xle) count is allocated to **I** (Ile) and **L** (Leu)
- **X** (Xaa) count is allocated to the twenty standard amino acids
- Joint probability of observing a pair of ambiguous amino acids is allocated to all potential combinations, e.g. probability of **XX** is allocated to 400 combinations of standard amino acids, similarly probability of **XB** is allocated to 40 combinations of **D** and **N** with the standard amino acids.

Selenocysteine (**U**, Sec) and pyrolysine (**O**, Pyl) are considered as distinct amino acids. When ambiguity is set False, all alphabet characters as considered as distinct types. All non-alphabet characters are considered as gaps.

3.8. Sequence Analysis 168
By default, *turbo* mode, which uses memory as large as the MSA array itself but runs four to five times faster, will be used. If memory allocation fails, the implementation will fall back to slower and memory efficient mode.

**buildSCAMatrix** *(msa, turbo=True, **kwargs)*

Return SCA matrix calculated for *msa*, which may be an MSA (page 169) instance or a 2D Numpy character array.

Implementation is case insensitive and handles ambiguous amino acids as follows:

- **B** (Asx) count is allocated to **D** (Asp) and **N** (Asn)
- **Z** (Glx) count is allocated to **E** (Glu) and **Q** (Gln)
- **J** (Xle) count is allocated to **I** (Ile) and **L** (Leu)
- **X** (Xaa) count is allocated to the twenty standard amino acids

- Joint probability of observing a pair of ambiguous amino acids is allocated to all potential combinations, e.g. probability of **XX** is allocated to 400 combinations of standard amino acids, similarly probability of **XB** is allocated to 40 combinations of **D** and **N** with the standard amino acids.

Selenocysteine (**U**, Sec) and pyrrolysine (**O**, Pyl) are considered as distinct amino acids. When *ambiguity* is set **False**, all alphabet characters as considered as distinct types. All non-alphabet characters are considered as gaps.

By default, *turbo* mode, which uses memory as large as the MSA array itself but runs four to five times faster, will be used. If memory allocation fails, the implementation will fall back to slower and memory efficient mode.

**buildDirectInfoMatrix** *(msa, seqid=0.8, pseudo_weight=0.5, refine=False, **kwargs)*

Return direct information matrix calculated for *msa*, which may be an MSA (page 169) instance or a 2D Numpy character array.

Sequences sharing sequence identity of *seqid* or more with another sequence are regarded as similar sequences for calculating their weights using **calcMeff** (page 169).

*pseudo_weight* are the weight for pseudo count probability.

Sequences are not refined by default. When *refine* is set **True**, the MSA will be refined by the first sequence and the shape of direct information matrix will be smaller.

**calcMeff** *(msa, seqid=0.8, refine=False, weight=False, **kwargs)*

Return the Meff for *msa*, which may be an MSA (page 169) instance or a 2D Numpy character array.

Since similar sequences in an *msa* decreases the diversity of *msa*, **Meff** gives a weight for sequences in the *msa*.

For example: One sequence in MSA has 5 other similar sequences in this MSA(itself included). The weight of this sequence is defined as 1/5=0.2. **Meff** is the sum of all sequence weights. In another word, **Meff** can be understood as the effective number of independent sequences.

Sequences sharing sequence identity of *seqid* or more with another sequence are regarded as similar sequences to calculate **Meff**.

Sequences are not refined by default. When *refine* is set **True**, the MSA will be refined by the first sequence.

The weight for each sequence are returned when *weight* is **True**.

### 3.8.7 Multiple Sequence Alignment

This module defines MSA analysis functions.
class MSA (msa, title='Unknown', labels=None, **kwargs)

Store and manipulate multiple sequence alignments.

msa must be a 2D Numpy character array. labels is a list of sequence labels (or titles). mapping should map label or part of label to sequence index in msa array. If mapping is not given, one will be build from labels.

countLabel (label)

Return the number of sequences that label maps onto.

getArray ()

Return a copy of the MSA character array.

ggetIndex (label)

Return index of the sequence that label maps onto. If label maps onto multiple sequences or label is a list of labels, a list of indices is returned. If an index for a label is not found, return None.

getLabel (index, full=False)

Return label of the sequence at given index. Residue numbers will be removed from the sequence label, unless full is True.

getResnums (index)

Return starting and ending residue numbers (resnum) for the sequence at given index.

ggetTitle ()

Return title of the instance.

isAligned ()

Return True if MSA is aligned.

iterLabels (full=False)

Yield sequence labels. By default the part of the label used for indexing sequences is yielded.

numIndexed ()

Return number of sequences that are indexed using the identifier part or all of their labels. The return value should be equal to number of sequences.

numResidues ()

Return number of residues (or columns in the MSA), if MSA is aligned.

numSequences ()

Return number of sequences.

setTitle (title)

Set title of the instance.

split

Return split label when iterating or indexing.

refineMSA (msa, label=None, rowocc=None, seqid=None, colocc=None, **kwargs)

Refine msa by removing sequences (rows) and residues (columns) that contain gaps.

Parameters

- msa (\texttt{MSA} (page 169)) – multiple sequence alignment
- label (\texttt{str})\(^{373}\) – remove columns that are gaps in the sequence matching label, msa.getIndex(label) must return a sequence index, a PDB identifier is also acceptable

\(^{373}\)http://docs.python.org/library/functions.html\#str

3.8. Sequence Analysis 170
- **rowocc** (*float*) – row occupancy, sequences with less occupancy will be removed after label refinement is applied

- **seqid** (*float*) – keep unique sequences at specified sequence identity level, unique sequences are identified using `uniqueSequences()` (page 168)

- **colocc** (*float*) – column occupancy, residue positions with less occupancy will be removed after other refinements are applied

- **keep** – keep columns corresponding to residues not resolved in the PDB structure, default is `False`, applies when `label` is a PDB identifier

- **type** – bool

For Pfam MSA data, `label` is Uniprot entry name for the protein. You may also use PDB structure and chain identifiers, e.g. ‘1p38’ or ‘1p38A’, for `label` argument and Uniprot entry names will be parsed using `parsePDBHeader()` (page 157) function (see also `Polymer` (page 154) and `DBRef` (page 156)).

The order of refinements are applied in the order of arguments. If `label` and `unique` is specified is specified, sequence matching `label` will be kept in the refined MSA (page 169) although it may be similar to some other sequence.

```
mergeMSA(*msa, **kwargs)
```

Return an MSA (page 169) obtained from merging parts of the sequences of proteins present in multiple `msa` instances. Sequences are matched based on protein identifiers found in the sequence labels. Order of sequences in the merged MSA will follow the order of sequences in the first `msa` instance. Note that protein identifiers that map to multiple sequences will be excluded.

### 3.8.8 MSA File

This module defines functions and classes for parsing, manipulating, and analyzing multiple sequence alignments.

```
class MSAFile(msa, mode='r', format=None, aligned=True, **kwargs)
```

Handle MSA files in FASTA, SELEX and Stockholm formats.

- `msa` may be a filename or a stream. Multiple sequence alignments can be read from or written in FASTA (.fasta), Stockholm (.sth), or SELEX (.slx) format. For specified extensions, `format` argument is not needed. If `aligned` is True, unaligned sequences in the file or stream will cause an IOError exception. `filter`, a function that returns a boolean, can be used for filtering sequences, see `setFilter()` (page 172) for details. `slice` can be used to slice sequences, and is applied after filtering, see `setSlice()` (page 172) for details.

```
close()
```

Close the file. This method will not affect a stream.

```
getFilename()
```

Return filename, or `None` if instance is handling a stream.

```
getFilter()
```

Return function used for filtering sequences.

```
getFormat()
```

Return file format.

```
getSlice()
```

Return object used to slice sequences.

---

[374]http://docs.python.org/library/functions.html#float
[375]http://docs.python.org/library/functions.html#float
[376]http://docs.python.org/library/functions.html#float

### 3.8. Sequence Analysis
getTitle() 
Return title of the instance.

isAligned() 
Return True if MSA is aligned.

reset() 
Return to the beginning of the file.

setFilter (filter, filter_full=False) 
Set function used for filtering sequences. filter will be applied to split sequence label, by default. 
If filter_full is True, filter will be applied to the full label.

setSlice (slice) 
Set object used to slice sequences, which may be a slice() or a list() of numbers.

setTitle (title) 
Set title of the instance.

write (seq) 
Write seq, an Sequence (page 173) instance, into the MSA file.

closed 
True for closed file.

format 
Format of the MSA file.

splitSeqLabel (label) 
Return label, starting residue number, and ending residue number parsed from sequence label.

parseMSA (filename, **kwargs) 
Return an MSA (page 169) instance that stores multiple sequence alignment and sequence labels parsed 
from Stockholm, SELEX, or FASTA format filename file, which may be a compressed file. Uncom-
pressed MSA files are parsed using C code at a fraction of the time it would take to parse compressed 
files in Python.

writeMSA (filename, msa, **kwargs) 
Return filename containing msa, a MSA (page 169) or MSAFile (page 171) instance, in the specified 
format, which can be SELEX, Stockholm, or FASTA. If compressed is True or filename ends with .gz, 
a compressed file will be written. MSA (page 169) instances will be written using C function into 
uncompressed files.

3.8.9 Plotting Functions

This module defines MSA analysis functions.

showMSAOccupancy (msa, occ='res', indices=None, count=False, **kwargs) 
Show a bar plot of occupancy calculated for MSA (page 169) instance msa using 
calcMSAOccupancy() (page 167). occ may be 'res' or 'col', or a a pre-calculated occup-
cypancy array. If x-axis indices are not specified, they will be inferred from msa or given label that may 
correspond to a sequence in the msa.

Occupancy is plotted using bar() function.

377http://docs.python.org/library/functions.html#slice
378http://docs.python.org/library/functions.html#list
379http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.bar

3.8. Sequence Analysis 172
showShannonEntropy(entropy, indices=None, **kwargs)

Show a bar plot of Shannon entropy array. MSA (page 169) instances or Numpy character arrays storing sequence alignments are also accepted as entropy argument, in which case calcShannonEntropy() (page 167) will be used for calculations. indices may be residue numbers, when not specified they will be inferred from msa or indices starting from 1 will be used.

Entropy is plotted using bar() function.

showMutinfoMatrix(mutinfo, *args, **kwargs)

Show a heatmap of mutual information array. MSA (page 169) instances or Numpy character arrays storing sequence alignment are also accepted as mutinfo argument, in which case buildMutinfoMatrix() (page 167) will be used for calculations. Note that x, y axes contain indices of the matrix starting from 1.

Mutual information is plotted using imshow() function. vmin and vmax values can be set by user to achieve better signals using this function.

showDirectInfoMatrix(dirinfo, *args, **kwargs)

Show a heatmap of direct information array. MSA (page 169) instances or Numpy character arrays storing sequence alignment are also accepted as dirinfo argument, in which case buildDirectInfoMatrix() (page 169) will be used for calculations. Note that x, y axes contain indices of the matrix starting from 1.

Direct information is plotted using imshow() function. vmin and vmax values can be set by user to achieve better signals using this function.

showSCAMatrix(scainfo, *args, **kwargs)

Show a heatmap of SCA (statistical coupling analysis) array. MSA (page 169) instances. blah or Numpy character arrays storing sequence alignment are also accepted as scainfo argument, in which case buildSCAMatrix() (page 169) will be used for calculations. Note that x, y axes contain indices of the matrix starting from 1.

SCA information is plotted using imshow() function. vmin and vmax values can be set by user to achieve better signals using this function.

3.8.10 Sequence

This module handles individual sequences.

class Sequence(*args)

Handle individual sequences of an MSA (page 169) object

Depending on input arguments, instances may point to an MSA (page 169) object or store its own data.

copy()

Return a copy of the instance that owns its sequence data.

getIndex()

Return sequence index or None.

getLabel(full=False)

Return label of the sequence.
getMSA()
Return MSA (page 169) instance or None.

getResnums(gaps=False)
Return list of residue numbers associated with non-gapped seq. When gaps is True, return a list containing the residue numbers with gaps appearing as None. Residue numbers are inferred from the full label. When label does not contain residue number information, indices a range of numbers starting from 1 is returned.

numGaps()
Return number of gap characters.

numResidues()
Return the number of alphabet characters.

3.9 Trajectory I/O

This module defines classes for handling trajectory files in DCD format.

3.9.1 Parse/write DCD files

- DCDFile (page 175)
- parseDCD() (page 177)
- writeDCD() (page 177)

3.9.2 Parse structure files

- parsePSF() (page 178)

3.9.3 Handle multiple files

- Trajectory (page 181)

3.9.4 Handle frame data

- Frame (page 178)

3.9.5 Examples

Following examples show how to use trajectory classes and functions:

- Trajectory Analysis
- Trajectory Analysis II
- Essential Dynamics Analysis

---

384 http://prody.csb.pitt.edu/tutorials/trajectory_analysis/trajectory.html#trajectory
385 http://prody.csb.pitt.edu/tutorials/trajectory_analysis/trajectory2.html#trajectory2
386 http://prody.csb.pitt.edu/tutorials/trajectory_analysis/eda.html#eda
3.9.6 DCD File

This module defines classes for handling trajectory files in DCD format. A class for reading and writing DCD files. DCD header and first frame is parsed at instantiation. Coordinates from the first frame is set as the reference coordinate set. This class has been tested for 32-bit DCD files. 32-bit floating-point coordinate array can be casted automatically to a specified type, such as 64-bit float, using `astype` keyword argument, i.e. `astype=float`, using `ndarray.astype()` method.

Open `filename` for reading (default, `mode="r"`), writing (`mode="w"`), or appending (`mode="r+"` or `mode="a"`). Binary mode option will be appended automatically.

```
close()
   Close trajectory file.

flush()
   Flush the internal output buffer.

getAtoms()
   Return associated/selected atoms.

getCoords()
   Return a copy of reference coordinates for (selected) atoms.

getCoordsets(indices=None)
   Returns coordinate sets at given `indices`. `indices` may be an integer, a list of ordered integers or `None`. `None` returns all coordinate sets. If a list of indices is given, unique numbers will be selected and sorted. That is, this method will always return unique coordinate sets in the order they appear in the trajectory file. Shape of the coordinate set array is $(n_{sets}, n_{atoms}, 3)$.

getFilename(absolute=False)
   Return relative path to the current file. For absolute path, pass `absolute=True` argument.

getFirstTimestep()
   Return first timestep value.

gETFramex(index)
   Return frame at given `index`.

gETFramexFreq()
   Return timesteps between frames.

getLinked()
   Return linked `AtomGroup` (page 37) instance, or `None` if a link is not established.

getRemarks()
   Return remarks parsed from DCD file.

g ETFramex()
   Return timestep size.

gETFramex()
   Return title of the ensemble.

gETFramex()
   Return a copy of weights of (selected) atoms.
```

387http://www.ks.uiuc.edu/Research/namd/2.6/ug/node13.html
**goto**(*n*)

Go to the frame at index *n*. *n*=0 will rewind the trajectory to the beginning, same as calling reset() (page 176) method. *n*=−1 will go to the last frame. Frame *n* will not be parsed until one of next() (page 176) or nextCoordset() (page 176) methods is called.

**hasUnitcell()**

Return True if trajectory has unitcell data.

**isLinked()**

Return True if trajectory is linked to an AtomGroup (page 37) instance.

**iterCoordsets()**

Yield coordinate sets for (selected) atoms. Reference coordinates are not included. Iteration starts from the next frame in line.

**link(*ag*)**

Link, return, or unlink an AtomGroup (page 37) instance. When a link to *ag* is established, coordinates of new frames parsed from the trajectory file will be set as the coordinates of *ag* and this will update coordinates of all selections and atom subsets pointing to it. At link time, if *ag* does not have any coordinate sets and reference coordinates of the trajectory is set, reference coordinates of the trajectory will be passed to *ag*. To break an established link, pass None argument, or to return the linked atom group instance, call with no arguments.

**Warning:** Every time a frame is parsed from the trajectory, all coordinate sets present in the linked AtomGroup (page 37) will be overwritten.

**next()**

Return next coordinate set in a Frame (page 178) instance. Note that when atoms are set for the trajectory, this method will return the same frame instance after updating its coordinates.

**nextCoordset()**

Return next coordinate set.

**nextIndex()**

Return the index of the next frame.

**numAtoms()**

Return number of atoms.

**numCoordsets()**

Return number of frames.

**numFixed()**

Return number of fixed atoms.

**numFrames()**

Return number of frames.

**numSelected()**

Return number of selected atoms. A subset of atoms can be selected by passing a selection to setAtoms() (page 176).

**reset()**

Go to first frame at index 0. First frame will not be parsed until one of next() (page 176) or nextCoordset() (page 176) methods is called.

**setAtoms(atoms)**

Set atoms or specify a selection of atoms to be considered in calculations and coordinate requests. When a selection is set, corresponding subset of coordinates will be considered in, for example, alignments and RMSD calculations. Setting atoms also allows some functions to access atomic
data when needed. For example, `Trajectory` (page 181) and `Frame` (page 178) instances become suitable arguments for `writePDB()` (page 163). Passing `None` as `atoms` argument will deselect atoms. Note that setting atoms does not change the reference coordinates of the trajectory. To change the reference, use `setCoords()` (page 177) method.

```python
setCoords(coords)
```

Set `coords` as the trajectory reference coordinate set. `coords` must be an object with `getCoords()` (page 175) method, or a Numpy array with suitable data type, shape, and dimensionality.

```python
setTitle(title)
```

Set title of the ensemble.

```python
setWeights(weights)
```

Set atomic weights.

```python
skip(n)
```

Skip `n` frames. `n` must be a positive integer. Skipping some frames will only change the next frame index (`nextIndex()` (page 176)). Next frame will not be parsed until one of `next()` (page 176) or `nextCoordset()` (page 176) methods is called.

```python
write(coords, unitcell=None, **kwargs)
```

Write `coords` to a file open in ‘a’ or ‘w’ mode. `coords` may be a Numpy array or a ProDy object that stores or points to coordinate data. Note that all coordinate sets of ProDy object will be written. Number of atoms will be determined from the file or based on the size of the first coordinate set written. If `unitcell` is provided for the first coordinate set, it will be expected for the following coordinate sets as well. If `coords` is an `Atomic` (page 44) or `Ensemble` (page 132) all coordinate sets will be written.

Following keywords are used when writing the first coordinate set:

- **timestep** – timestep used for integration, default is 1
- **firsttimestep** – number of the first timestep, default is 0
- **framefreq** – number of timesteps between frames, default is 1

```python
parseDCD(filename, start=None, stop=None, step=None,.astype=None)
```

Parse CHARMM format DCD files (also NAMD 2.1 and later). Returns an `Ensemble` instance. Conformations in the ensemble will be ordered as they appear in the trajectory file. Use `DCDFile` (page 175) class for parsing coordinates of a subset of atoms.

```python
writeDCD(filename, trajectory, start=None, stop=None, step=None, align=False)
```

Write 32-bit CHARMM format DCD file (also NAMD 2.1 and later). `trajectory` can be an...
:class:`Trajectory`, :class:`DCDFile`, or :class:`Ensemble` instance. *filename* is returned upon successful output of file.

### 3.9.7 Frame

This module defines a class for handling trajectory frames.

```python
class Frame(traj, index, coords, unitcell=None, velocs=None)
```

A class for storing trajectory frame coordinates and provide methods acting on them.

- **getAtoms()**
  Return associated/selected atoms.

- **getCoords()**
  Return a copy of coordinates of (selected) atoms.

- **getDeviations()**
  Return deviations from the trajectory reference coordinates.

- **getIndex()**
  Return index.

- **getRMSD()**
  Return RMSD from the trajectory reference coordinates. If weights for the trajectory are set, weighted RMSD will be returned.

- **getTrajectory()**
  Return the trajectory.

- **getUnitcell()**
  Return a copy of unitcell array.

- **getVelocities()**
  Return a copy of velocities of (selected) atoms.

- **getWeights()**
  Return coordinate weights for selected atoms.

- **numAtoms()**
  Return number of atoms.

- **numSelected()**
  Return number of selected atoms.

- **superpose()**
  Superpose frame onto the trajectory reference coordinates. Note that transformation matrix is calculated based on selected atoms and applied to all atoms. If atom weights for the trajectory are set, they will be used to calculate the transformation.

### 3.9.8 PSF File

This module defines a function for parsing protein structure files in PSF format.³⁹³

```python
parsePSF(filename, title=None, ag=None)
```

Return an :class:`AtomGroup` instance storing data parsed from X-PLOR format PSF file *filename*. Atom and bond information is parsed from the file. If *title* is not given, *filename* will be set as the title of the :class:`AtomGroup` instance. An :class:`AtomGroup` instance may be provided as

When provided, \texttt{ag} must have the same number of atoms in the same order as the file. Data from PSF file will be added to the \texttt{ag}. This may overwrite present data if it overlaps with PSF file content. Note that this function does not evaluate angles, dihedrals, and impropers sections.

\texttt{writePSF (filename, atoms)}

Write atoms in X-PLOR format PSF file with name \texttt{filename} and return \texttt{filename}. This function will write available atom and bond information only.

### 3.9.9 Trajectory Base

This module defines base class for trajectory handling.

\texttt{class TrajBase (title='Unknown')} Base class for \texttt{Trajectory} (page 181) and \texttt{TrajFile} (page 183). Derived classes must implement functions described in this class.

\texttt{close ()} Close trajectory file.

\texttt{getAtoms ()} Return associated/selected atoms.

\texttt{getCoords ()} Return a copy of reference coordinates for (selected) atoms.

\texttt{getCoordsets (indices=None)} Returns coordinate sets at given \texttt{indices}. \texttt{indices} may be an integer, a list of ordered integers or \texttt{None}. \texttt{None} returns all coordinate sets. If a list of indices is given, unique numbers will be selected and sorted. That is, this method will always return unique coordinate sets in the order they appear in the trajectory file. Shape of the coordinate set array is \texttt{(n_sets, n_atoms, 3)}.

\texttt{getFrame (index)} Return frame at given \texttt{index}.

\texttt{getLinked ()} Return \texttt{True} if trajectory is linked to an \texttt{AtomGroup} (page 37) instance, or \texttt{None} if a link is not established.

\texttt{getTitle ()} Return title of the ensemble.

\texttt{getWeights ()} Return a copy of weights of (selected) atoms.

\texttt{goto (n)} Go to the frame at index \texttt{n}. \texttt{n=0} will rewind the trajectory to the beginning, same as calling \texttt{reset ()} (page 180) method. \texttt{n=-1} will go to the last frame. Frame \texttt{n} will not be parsed until one of \texttt{next ()} (page 180) or \texttt{nextCoordset ()} (page 180) methods is called.

\texttt{hasUnitcell ()} Return \texttt{True} if trajectory has unitcell data.

\texttt{isLinked ()} Return \texttt{True} if trajectory is linked to an \texttt{AtomGroup} (page 37) instance.

\texttt{iterCoordsets ()} Yield coordinate sets for (selected) atoms. Reference coordinates are not included. Iteration starts from the next frame in line.

\texttt{link ("ag")} Link, return, or unlink an \texttt{AtomGroup} (page 37) instance. When a link to \texttt{ag} is established, coordinates of new frames parsed from the trajectory file will be set as the coordinates of \texttt{ag} and this
will update coordinates of all selections and atom subsets pointing to it. At link time, if \texttt{ag} does not have any coordinate sets and reference coordinates of the trajectory is set, reference coordinates of the trajectory will be passed to \texttt{ag}. To break an established link, pass \texttt{None} argument, or to return the linked atom group instance, call with no arguments.

**Warning:** Every time a frame is parsed from the trajectory, all coordinate sets present in the linked \texttt{AtomGroup} (page 37) will be overwritten.

\texttt{next} ()
Return next coordinate set in a \texttt{Frame} (page 178) instance. Note that when atoms are set for the trajectory, this method will return the same frame instance after updating its coordinates.

\texttt{nextCoordset} ()
Return next coordinate set.

\texttt{nextIndex} ()
Return the index of the next frame.

\texttt{numAtoms} ()
Return number of atoms.

\texttt{numCoordsets} ()
Return number of frames.

\texttt{numFrames} ()
Return number of frames.

\texttt{numSelected} ()
Return number of selected atoms. A subset of atoms can be selected by passing a selection to \texttt{setAtoms()} (page 180).

\texttt{reset} ()
Go to first frame at index 0. First frame will not be parsed until one of \texttt{next()} (page 180) or \texttt{nextCoordset()} (page 180) methods is called.

\texttt{setAtoms} (\texttt{atoms})
Set \texttt{atoms} or specify a selection of atoms to be considered in calculations and coordinate requests. When a selection is set, corresponding subset of coordinates will be considered in, for example, alignments and RMSD calculations. Setting atoms also allows some functions to access atomic data when needed. For example, \texttt{Trajectory} (page 181) and \texttt{Frame} (page 178) instances become suitable arguments for \texttt{writePDB()} (page 163). Passing \texttt{None} as \texttt{atoms} argument will deselect atoms. Note that setting atoms does not change the reference coordinates of the trajectory. To change the reference, use \texttt{setCoords()} (page 180) method.

\texttt{setCoords} (\texttt{coords})
Set \texttt{coords} as the trajectory reference coordinate set. \texttt{coords} must be an object with \texttt{getCoords()} (page 179) method, or a Numpy array with suitable data type, shape, and dimensionality.

\texttt{setTitle} (\texttt{title})
Set title of the ensemble.

\texttt{setWeights} (\texttt{weights})
Set atomic weights.

\texttt{skip} (\texttt{n})
Skip \texttt{n} frames. \texttt{n} must be a positive integer. Skipping some frames will only change the next frame index (\texttt{nextIndex()} (page 180)) Next frame will not be parsed until one of \texttt{next()} (page 180) or \texttt{nextCoordset()} (page 180) methods is called.
3.9.10 Trajectory

This module defines a class for handling multiple trajectories.

```python
class Trajectory(name, **kwargs)
    A class for handling trajectories in multiple files.
    Trajectory can be instantiated with a name or a filename. When name is a valid path to a trajectory file
    it will be opened for reading.

def addFile(filename, **kwargs)
    Add a file to the trajectory instance. Currently only DCD files are supported.

def close()
    Close trajectory file.

def getAtoms()
    Return associated/selected atoms.

def getCoords()
    Return a copy of reference coordinates for (selected) atoms.

def getCoordsets(indices=None)
    Returns coordinate sets at given indices. indices may be an integer, a list of ordered integers or
    None. None returns all coordinate sets. If a list of indices is given, unique numbers will be
    selected and sorted. That is, this method will always return unique coordinate sets in the order
    they appear in the trajectory file. Shape of the coordinate set array is (n_sets, n_atoms, 3).

def getFilenames(absolute=False)
    Return list of filenames opened for reading.

def getFirstTimestep()
    Return list of first timestep values, one number from each file.

def getFrameFreq()
    Return list of timesteps between frames, one number from each file.

def getLinked()
    Return linked AtomGroup (page 37) instance, or None if a link is not established.

def getTimestep()
    Return list of timestep sizes, one number from each file.

def getTitle()
    Return title of the ensemble.

def getWeights()
    Return a copy of weights of (selected) atoms.

def goto(n)
    Go to the frame at index n. n=0 will rewind the trajectory to the beginning, same as calling
    reset() (page 182) method. n=-1 will go to the last frame. Frame n will not be parsed until
    one of next() (page 182) or nextCoordset() (page 182) methods is called.

def hasUnitcell()
    Return True if trajectory has unitcell data.

def isLinked()
    Return True if trajectory is linked to an AtomGroup (page 37) instance.

def iterCoordsets()
    Yield coordinate sets for (selected) atoms. Reference coordinates are not included. Iteration starts
    from the next frame in line.
```
**link**(*ag*)

Link, return, or unlink an AtomGroup (page 37) instance. When a link to *ag* is established, coordinates of new frames parsed from the trajectory file will be set as the coordinates of *ag* and this will update coordinates of all selections and atom subsets pointing to it. At link time, if *ag* does not have any coordinate sets and reference coordinates of the trajectory is set, reference coordinates of the trajectory will be passed to *ag*. To break an established link, pass `None` argument, or to return the linked atom group instance, call with no arguments.

**Warning:** Every time a frame is parsed from the trajectory, all coordinate sets present in the linked AtomGroup (page 37) will be overwritten.

**next**()

Return next coordinate set in a Frame (page 178) instance. Note that when atoms are set for the trajectory, this method will return the same frame instance after updating its coordinates.

**nextCoordset**()

Return next coordinate set.

**nextIndex**()

Return the index of the next frame.

**numAtoms**()

Return number of atoms.

**numCoordsets**()

Return number of frames.

**numFiles**()

Return number of open trajectory files.

**numFixed**()

Return a list of fixed atom numbers, one from each file.

**numFrames**()

Return number of frames.

**numSelected**()

Return number of selected atoms. A subset of atoms can be selected by passing a selection to `setAtoms()` (page 182).

**reset**()

Go to first frame at index 0. First frame will not be parsed until one of `next()` (page 182) or `nextCoordset()` (page 182) methods is called.

**setAtoms**(*atoms*)

Set *atoms* or specify a selection of atoms to be considered in calculations and coordinate requests. When a selection is set, corresponding subset of coordinates will be considered in, for example, alignments and RMSD calculations. Setting atoms also allows some functions to access atomic data when needed. For example, Trajectory (page 181) and Frame (page 178) instances become suitable arguments for `writePDB()` (page 163). Passing `None` as *atoms* argument will deselect atoms. Note that setting atoms does not change the reference coordinates of the trajectory. To change the reference, use `setCoords()` (page 182) method.

**setCoords**(*coords*)

Set *coords* as the trajectory reference coordinate set. *coords* must be an object with `getCoords()` (page 181) method, or a Numpy array with suitable data type, shape, and dimensionality.

**setTitle**(*title*)

Set title of the ensemble.
setWeights(weights)
Set atomic weights.

skip(n)
Skip n frames. n must be a positive integer. Skipping some frames will only change the next
frame index (nextIndex() (page 182)) Next frame will not be parsed until one of next ()
(page 182) or nextCoordset() (page 182) methods is called.

3.9.11 Trajectory File

This module defines a base class for format specific trajectory classes.

class TrajFile (filename, mode='r')
A base class for trajectory file classes:

• DCDFile (page 175)
Open filename for reading (default, mode="r"), writing (mode="w"), or appending (mode="r+" or
mode="a"). Binary mode option will be appended automatically.

close()
Close trajectory file.

getAtoms()
Return associated/selected atoms.

getCoords()
Return a copy of reference coordinates for (selected) atoms.

getCoordsets(indices=None)
Returns coordinate sets at given indices. indices may be an integer, a list of ordered integers or
None. None returns all coordinate sets. If a list of indices is given, unique numbers will be
selected and sorted. That is, this method will always return unique coordinate sets in the order
they appear in the trajectory file. Shape of the coordinate set array is (n_sets, n_atoms, 3).

getFilename(absolute=False)
Return relative path to the current file. For absolute path, pass absolute=True argument.

getFirstTimestep()
Return first timestep value.

getFrame(index)
Return frame at given index.

getFrameFreq()
Return timesteps between frames.

getLinked()
Return linked AtomGroup (page 37) instance, or None if a link is not established.

getTimestep()
Return timestep size.

ggetTitle()
Return title of the ensemble.

ggetWeights()
Return a copy of weights of (selected) atoms.

goto(n)
Go to the frame at index n. n=0 will rewind the trajectory to the beginning, same as calling
reset() (page 184) method. \( n = -1 \) will go to the last frame. Frame \( n \) will not be parsed until one of next() (page 184) or nextCoordset() (page 184) methods is called.

hasUnitcell()
Return True if trajectory has unitcell data.

isLinked()
Return True if trajectory is linked to an AtomGroup (page 37) instance.

iterCoordsets()
Yield coordinate sets for (selected) atoms. Reference coordinates are not included. Iteration starts from the next frame in line.

link("ag")
Link, return, or unlink an AtomGroup (page 37) instance. When a link to \( ag \) is established, coordinates of new frames parsed from the trajectory file will be set as the coordinates of \( ag \) and this will update coordinates of all selections and atom subsets pointing to it. At link time, if \( ag \) does not have any coordinate sets and reference coordinates of the trajectory is set, reference coordinates of the trajectory will be passed to \( ag \). To break an established link, pass None argument, or to return the linked atom group instance, call with no arguments.

**Warning:** Every time a frame is parsed from the trajectory, all coordinate sets present in the linked AtomGroup (page 37) will be overwritten.

next()
Return next coordinate set in a Frame (page 178) instance. Note that when atoms are set for the trajectory, this method will return the same frame instance after updating its coordinates.

nextCoordset()
Return next coordinate set.

nextIndex()
Return the index of the next frame.

numAtoms()
Return number of atoms.

numCoordsets()
Return number of frames.

numFixed()
Return number of fixed atoms.

numFrames()
Return number of frames.

numSelected()
Return number of selected atoms. A subset of atoms can be selected by passing a selection to setAtoms() (page 184).

reset()
Go to first frame at index 0. First frame will not be parsed until one of next() (page 184) or nextCoordset() (page 184) methods is called.

setAtoms(atoms)
Set atoms or specify a selection of atoms to be considered in calculations and coordinate requests. When a selection is set, corresponding subset of coordinates will be considered in, for example, alignments and RMSD calculations. Setting atoms also allows some functions to access atomic data when needed. For example, Trajectory (page 181) and Frame (page 178) instances become suitable arguments for writePDB() (page 163). Passing None as atoms argument will
deselect atoms. Note that setting atoms does not change the reference coordinates of the trajectory. To change the reference, use `setCoords()` (page 185) method.

```python
setCoords(coords)
```
Set `coords` as the trajectory reference coordinate set. `coords` must be an object with `getCoords()` (page 183) method, or a Numpy array with suitable data type, shape, and dimensionality.

```python
setTitle(title)
```
Set title of the ensemble.

```python
setWeights(weights)
```
Set atomic weights.

```python
skip(n)
```
Skip `n` frames. `n` must be a positive integer. Skipping some frames will only change the next frame index (`nextIndex()` (page 184)) Next frame will not be parsed until one of `next()` (page 184) or `nextCoordset()` (page 184) methods is called.

### 3.10 ProDy Utilities

This module provides utility functions and classes for handling files, logging, type checking, etc. Contents of this module are not included in ProDy namespace, as it is not safe to import them all due to name conflicts. Required or classes should be imported explicitly, e.g. `from prody.utilities import PackageLogger, openFile`.

#### 3.10.1 Package utilities

- `PackageLogger` (page 187)
- `PackageSettings` (page 191)
- `getPackagePath()` (page 192)
- `setPackagePath()` (page 192)

#### 3.10.2 Type/Value checkers

- `checkCoords()` (page 186)
- `checkWeights()` (page 187)
- `checkTypes()` (page 187)

#### 3.10.3 Path/file handling

- `gunzip()` (page 190)
- `openFile()` (page 190)
- `openDB()` (page 191)
- `openSQLite()` (page 191)
- `openURL()` (page 191)
- `copyFile()` (page 191)
• isExecutable() (page 191)
• isReadable() (page 191)
• isWritable() (page 191)
• makePath() (page 191)
• relpath() (page 191)
• which() (page 191)
• pickle() (page 191)
• unpickle() (page 191)
• glob() (page 191)

3.10.4 Documentation tools

• joinRepr() (page 187)
• joinRepr() (page 187)
• joinTerms() (page 187)
• tabulate() (page 187)
• wrapText() (page 187)

3.10.5 Miscellaneous tools

• rangeString() (page 189)
• alnum() (page 190)
• importLA() (page 190)
• dictElement() (page 190)

3.10.6 Type Checkers

This module defines functions for type, value, and/or attribute checking.

checkCoords(coords, csets=False, natoms=None, dtype=(<type 'float'>, <type 'numpy.float32'>), name='coords')

Return True if shape, dimensionality, and data type of coords array are as expected.

Parameters

• coords – coordinate array
• csets – whether multiple coordinate sets (i.e. .ndim in (2, 3)) are allowed, default is False
• natoms – number of atoms, if None number of atoms is not checked
• dtype – allowed data type(s), default is (float, numpy.float32), if None data type is not checked
• name – name of the coordinate argument
Raises TypeError when `coords` is not an instance of `numpy.ndarray`.

Raises ValueError when wrong shape, dimensionality, or data type is encountered.

**`checkWeights`** *(weights, natoms, nsets=None, dtype=<type 'float'>)*

Return `weights` if it has correct shape ([nsets, natoms, 1]). after its shape and data type is corrected. otherwise raise an exception. All items of `weights` must be greater than zero.

**`checkTypes`** *(args, **types)*

Return True if types of all `args` match those given in `types`.

Raises TypeError when type of an argument is not one of allowed types.

```python
def incr(n, i):
    '''Return sum of *n* and *i*.'''
    checkTypes(locals(), n=(float, int), i=(float, int))
    return n + i
```

### 3.10.7 Documentation Tools

This module defines miscellaneous utility functions.

**`joinLinks`** *(links, sep=' ', last=None, sort=False)*

Return a string joining `links` as reStructuredText.

**`joinRepr`** *(items, sep=' ', last=None, sort=False)*

Return a string joining representations of `items`.

**`joinTerms`** *(terms, sep=' ', last=None, sort=False)*

Return a string joining `terms` as reStructuredText.

**`tabulate`** *(*cols, **kwargs)*

Return a table for columns of data.

**Parameters**

- header *(bool)* – make first row a header, default is True
- width *(int)* – 79

**Kwargs**

- space number of white space characters between columns, default is 2

**`wrapText`** *(text, width=70, join='\n', **kwargs)*

Return wrapped lines from `textwrap.wrap()` after joining them.

### 3.10.8 Package Logger

This module defines class that can be used a package wide logger.

**`PackageLogger`** *(name, **kwargs)*

A class for package wide logging functionality.

Start logger for the package. Returns a logger instance.

**Parameters**

---


395 [http://docs.python.org/library/functions.html#bool](http://docs.python.org/library/functions.html#bool)

396 [http://docs.python.org/library/functions.html#int](http://docs.python.org/library/functions.html#int)

397 [http://docs.python.org/library/textwrap.html#textwrap.wrap](http://docs.python.org/library/textwrap.html#textwrap.wrap)
• **prefix** – prefix to console log messages, default is ‘@>’
• **console** – log level for console (*sys.stderr*) messages, default is ‘debug’
• **info** – prefix to log messages at info level
• **warning** – prefix to log messages at warning level, default is ‘WARNING’
• **error** – prefix to log messages at error level, default is ‘ERROR’

`addHandler (hdlr)`  
Add the specified handler to this logger.

`clear ()`  
Clear current specified line in *sys.stderr*.

`close (filename)`  
Close logfile *filename*.

`critical (msg)`  
Log *msg* with severity ‘CRITICAL’.

`debug (msg)`  
Log *msg* with severity ‘DEBUG’.

`delHandler (index)`  
Remove handler at given *index* from the logger instance.

`error (msg)`  
Log *msg* with severity ‘ERROR’ and terminate with status 2.

`exit (status=0)`  
Exit the interpreter.

`getHandlers ()`  
Return handlers.

`info (msg)`  
Log *msg* with severity ‘INFO’.

`progress (msg, steps, label=None, **kwargs)`  
Instantiate a labeled process with message and number of steps.

`report (msg='Completed in %.2fs.', label=None)`  
Write *msg* with timing information for a labeled or default process at debug logging level.

`sleep (seconds, msg='')`  
Sleep for seconds while updating screen message every second. Message will start with ‘Waiting for Xs’ followed by *msg*.

`start (filename, **kwargs)`  
Start a logfile. If *filename* does not have an extension, .log will be appended to it.

**Parameters**

• **filename** – name of the logfile
• **mode** – mode in which logfile will be opened, default is “w”
• **backupcount** – number of existing *filename.log* files to backup, default is 1

`timeit (label=None)`  
Start timing a process. Use `timing()` (page 188) and `report()` (page 188) to learn and report timing, respectively.
`timing` *(label= None)*
Return timing for a labeled or default *(None)* process.

`update` *(step, label= None)*
Update progress status to current line in the console.

`warn` *(msg)*
Log *msg* with severity ‘WARNING’.

`warning` *(msg)*
Log *msg* with severity ‘WARNING’.

`write` *(line)*
Write *line* into *sys.stderr*.

`prefix` 
String prepended to console log messages.

`verbosity` 
Verbosity *level* of the logger, default level is *debug*. Log messages are written to *sys.stderr*. Following logging levels are recognized:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug</td>
<td>Everything will be printed to the <em>sys.stderr</em>.</td>
</tr>
<tr>
<td>info</td>
<td>Only brief information will be printed.</td>
</tr>
<tr>
<td>warning</td>
<td>Only warning messages will be printed.</td>
</tr>
<tr>
<td>none</td>
<td>Nothing will be printed.</td>
</tr>
</tbody>
</table>

### 3.10.9 Miscellaneous Tools

This module defines miscellaneous utility functions.

`class Everything`
A place for everything.

`rangeString` *(lint, sep=’,’, rng=’ to ’, exc=False, pos=True)*
Return a structured string for a given list of integers.

**Parameters**

- *lint* – integer list or array
- *sep* – range or number separator
- *rng* – range symbol
- *exc* – set *True* if range symbol is exclusive
- *pos* – only consider zero and positive integers

```python
In [1]: from prody.utilities import rangeString
In [2]: lint = [1, 2, 3, 4, 10, 15, 16, 17]
In [3]: rangeString(lint)
Out[3]: '1 to 4 10 15 to 17'
In [4]: rangeString(lint, sep=’,’, rng=’-’)
Out[4]: '1-4,10,15-17'
```
**In [5]:** rangeString(lint, ',', ':', exc=True)
**Out[5]:** '1:5,10,15:18'

**alnum**(string, alt='_', trim=False, single=False)
Replace non alpha numeric characters with alt. If trim is True remove preceding and trailing arg characters. If single is True, contain only a single joining alt character.

**importLA()**
Return one of scipy.linalg or numpy.linalg.

**dictElement**(element, prefix=None)
Returns a dictionary built from the children of element, which must be a xml.etree.ElementTree.Element instance. Keys of the dictionary are tag of children without the prefix, or namespace. Values depend on the content of the child. If a child does not have any children, its text attribute is the value. If a child has children, then the child is the value.

**intorfloat**(x)
Return int(x), or float(x) upon ValueError.

**startswith**(this, that)
Return True if this or that starts with the other.

**showFigure**( )
Call show() function with block=False argument to avoid blocking behavior in non-interactive sessions. If block keyword argument is not recognized, try again without it.

**countBytes**(arrays, base=False)
Return total number of bytes consumed by elements of arrays. If base is True, use number of bytes from the base array.

### 3.10.10 Path Tools

This module defines functions for handling files and paths.

**gunzip**(filename, outname=None)
Return output name that contains decompressed contents of filename. When no outname is given, filename is used as the output name as it is or after .gz extension is removed. filename may also be a string buffer, in which case decompressed string buffer or outname that contains buffer will be returned.

**backupFile**(filename, backup=None, backup_ext='.BAK', **kwargs)
Rename filename with backup_ext appended to its name for backup purposes, if backup is True or if automatic backups is turned on using confProDy() (page 204). Default extension .BAK is used when one is not set using confProDy() (page 204). If filename does not exist, no action will be taken and filename will be returned. If file is successfully renamed, new filename will be returned.

**openFile**(filename, *args, **kwargs)
Open filename for reading, writing, or appending. First argument in args is treated as the mode. Opening .gz and .zip files for reading and writing is handled automatically.

**Parameters**

- **backup**(bool[401]) – backup existing file using backupFile() (page 190) when opening in append or write modes, default is obtained from package settings

---

398http://docs.scipy.org/doc/scipy/reference/linalg.html#scipy.linalg
399http://docs.python.org/library/xml.etree.elementtree.html#xml.etree.ElementTree.Element
400http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.show
401http://docs.python.org/library/functions.html#bool
• backup_ext (str) – extension for backup file, default is .BAK

openDB (filename, *args)
Open a database with given filename.

openSQLite (filename, *args)
Return a connection to SQLite database filename. If 'n' argument is passed, remove any existing
databases with the same name and return connection to a new empty database.

openURL (url, timeout=5, **kwargs)
Open url for reading. Raise an IOError if url cannot be reached. Small timeout values are suitable if
url is an ip address. kwargs will be used to make urllib.request.Request instance for opening
the url.

copyFile (src, dst)
Return dst, a copy of src.

isExecutable (path)
Return true if path is an executable.

isReadable (path)
Return true if path is readable by the user.

isWritable (path)
Return true if path is writable by the user.

makePath (path)
Make all directories that does not exist in a given path.

relpath (path)
Return path on Windows, and relative path elsewhere.

sympath (path, beg=2, end=1, ellipsis='...')
Return a symbolic path for a long path, by replacing folder names in the middle with ellipsis. beg and
end specified how many folder (or file) names to include from the beginning and end of the path.

which (program)
This function is based on the example in: http://stackoverflow.com/questions/377017/

pickle (obj, filename, protocol=2, **kwargs)
Pickle obj using pickle.dump() in filename. protocol is set to 2 for compatibility between Python 2
and 3.

unpickle (filename, **kwargs)
Unpickle object in filename using pickle.load().

glob (*pathnames)
Return concatenation of ordered lists of paths matching patterns in pathnames.

adext (filename, extension)
Return filename, with extension if it does not have one.

### 3.10.11 Package Settings

This module defines class for handling and storing package settings.
class **PackageSettings** *(pkg, rcfie=None, logger=None)*

A class for managing package settings. Settings are saved in user’s home director. When settings are changed by the users, the changes are automatically saved. Settings are stored in a :obj:`dict` instance. The dictionary is pickled in user’s home directory for permanent storage.

*rcfile* is the filename for pickled settings dictionary, and by default is set to `.pkgrc`.

**get** *(key, default=None)*

Return value corresponding to specified *key*, or *default* if *key* is not found.

**load()**

Load settings by unpickling the settings dictionary.

**pop** *(key, default=None)*

Remove specified *key* and return corresponding value. If *key* is not found, *default* is returned.

**save** *(backup=False)*

Save settings by pickling the settings dictionary.

**update** *(*args, **kwargs)*

Update settings dictionary.

**getPackagePath()**

Return package path.

**setPackagePath** *(path)*

Set package path.

## 3.11 Applications API

This module contains ProDy applications.

### 3.11.1 Dynamics analysis

- prody_anm() *(page 199)*
- prody_gnm() *(page 202)*
- prody_pca() *(page 203)*

### 3.11.2 Structure analysis

- prody_align() *(page 198)*
- prody_biomol() *(page 200)*
- prody_blast() *(page 200)*
- prody_catdcd() *(page 201)*
- prody_contacts() *(page 201)*
- prody_fetch() *(page 202)*
- prody_select() *(page 204)*

---

[^405]: http://docs.python.org/library/stdtypes.html#dict
3.11.3 Sequence analysis

- `evol_search()` (page 198)
- `evol_fetch()` (page 194)
- `evol_filter()` (page 195)
- `evol_refine()` (page 197)
- `evol_merge()` (page 195)
- `evol_conserv()` (page 193)
- `evol_coevol()` (page 193)
- `evol_occupancy()` (page 196)
- `evol_rankorder()` (page 196)

3.11.4 Coevolution Application

MSA residue coevolution calculation application.

```python
def evol_coevol(msa, **kwargs):
    Analyze co-evolution using mutual information.
```

**Parameters**

- `msa` – refined MSA file

**Calculation Options**

- `ambiguity` (`bool`): treat amino acids characters B, Z, J, and X as non-ambiguous, default is `True`
- `correction` (`str`): also save corrected mutual information matrix data and plot, one of `'apc'`, `'asc'`
- `normalization` (`str`): also save normalized mutual information matrix data and plot, one of `'sument'`, `'minent'`, `'maxent'`, `'mincon'`, `'maxcon'`, `'joint'`

**Output Options**

- `heatmap` (`bool`): save heatmap files for all mutual information matrices
- `prefix` (`str`): output filename prefix, default is msa filename with _coevol suffix
- `numformat` (`str`): number output format, default is `%12g`

3.11.5 Conservation Application

Calculate conservation in an MSA using Shannon entropy.

406 http://docs.python.org/library/functions.html#bool
407 http://docs.python.org/library/functions.html#str
408 http://docs.python.org/library/functions.html#str
409 http://docs.python.org/library/functions.html#bool
410 http://docs.python.org/library/functions.html#str
411 http://docs.python.org/library/functions.html#str
evol_conserv (msa, **kwargs)
Analyze conservation using Shannon entropy.

Parameters msa – refined MSA file

Calculation Options

Parameters

- ambiguity (bool) – treat amino acids characters B, Z, J, and X as non-ambiguous, default is True
- omitgaps (bool) – do not omit gap characters, default is True

Output Options

Parameters

- prefix (str) – output filename prefix, default is msa filename with _conserv suffix
- numformat (str) – number output format, default is ‘%12g’

3.11.6 Pfam MSA Fetcher

Pfam MSA download application.

evol_fetch (acc, **kwargs)
Fetch MSA files from Pfam.

Parameters acc (str) – Pfam accession or ID

Download Options

Parameters

- alignment (str) – alignment type, one of ‘full’, ‘seed’, ‘ncbi’, ‘metagenomics’, default is ‘full’
- format (str) – Pfam supported MSA format, one of ‘selex’, ‘fasta’, ‘stockholm’, default is ‘selex’
- order (str) – ordering of sequences, one of ‘tree’, ‘alphabetical’, default is ‘tree’
- inserts (str) – letter case for inserts, one of ‘upper’, ‘lower’, default is ‘upper’
- gaps (str) – gap character, one of ‘dashes’, ‘dots’, ‘mixed’, default is ‘dashes’
- timeout (int) – timeout for blocking connection attempts, default is 60

Output Options
Parameters
• folder (str) – output directory, default is ‘.’
• outname (str) – output filename, default is accession and alignment type
• compressed (bool) – gzip downloaded MSA file

3.11.7 MSA File Filter
Refine MSA application.

`evol_filter (msa, *word, **kwargs)`
Filter an MSA using sequence labels.

Parameters
• msa – MSA filename to be filtered
• word – word to be compared to sequence label

Filtering Method (Required)
Parameters
• `startswith` (bool) – sequence label starts with given words
• `endswith` (bool) – sequence label ends with given words
• `contains` (bool) – sequence label contains with given words

Filter Option
Parameters `filter_full` (bool) – compare full label with word(s)

Output Options
Parameters
• `outname` (str) – output filename, default is msa filename with _refined suffix
• `format` (str) – output MSA file format, default is same as input
• `compressed` (bool) – gzip refined MSA output

3.11.8 MSA File Merger
Merge multiple MSAs based on common labels.

`evol_merge (*msa, **kwargs)`
Merge multiple MSAs based on common labels.

Parameters
• msa – MSA filenames to be merged

---

423 http://docs.python.org/library/functions.html#str
424 http://docs.python.org/library/functions.html#str
425 http://docs.python.org/library/functions.html#bool
426 http://docs.python.org/library/functions.html#bool
427 http://docs.python.org/library/functions.html#bool
428 http://docs.python.org/library/functions.html#bool
429 http://docs.python.org/library/functions.html#str
430 http://docs.python.org/library/functions.html#str
431 http://docs.python.org/library/functions.html#str
432 http://docs.python.org/library/functions.html#bool
Output Options

Parameters

- **outname** *(str)* – output filename, default is first input filename with _merged suffix
- **format** *(str)* – output MSA file format, default is same as first input MSA
- **compressed** *(bool)* – gzip merged MSA output

3.11.9 MSA Occupancy Calculation

MSA residue coevolution calculation application.

**evol_occupancy** *(msa, **kwargs)*

Calculate occupancy of rows and columns in MSA.

Parameters

- **msa** – MSA file

Calculation Options

Parameters

- **occaxis** *(str)* – calculate row or column occupancy or both, one of ‘row’, ‘col’, ‘both’, default is ‘row’

Output Options

Parameters

- **prefix** *(str)* – output filename prefix, default is msa filename with _occupancy suffix
- **label** *(str)* – index for column based on msa label
- **numformat** *(str)* – number output format, default is ‘%12g’

3.11.10 Identify Coevolving Pairs

Refine MSA application.

**evol_rankorder** *(mutinfo, **kwargs)*

Identify highly coevolving pairs of residues.

Parameters

- **mutinfo** – mutual information matrix

Input Options

Parameters

- **zscore** *(bool)* – apply zscore for identifying top ranked coevolving pairs
- **delimiter** *(str)* – delimiter used in mutual information matrix file
• **pdb** (*str*[^442]) – PDB file that contains same number of residues as the mutual information matrix, output residue numbers will be based on PDB file

• **msa** (*str*[^443]) – MSA file used for building the mutual info matrix, output residue numbers will be based on the most complete sequence in MSA if a PDB file or sequence label is not specified

• **label** (*str*[^444]) – label in MSA file for output residue numbers

**Output Options**

**Parameters**

• **numpairs** (*int*[^445]) – number of top ranking residue pairs to list, default is 100

• **seqsep** (*int*[^446]) – report coevolution for residue pairs that are sequentially separated by input value, default is 3

• **dist** (*float*[^447]) – report coevolution for residue pairs whose CA atoms are spatially separated by at least the input value, used when a PDB file is given and –use-dist is true, default is 10.0

• **usedist** (*bool*[^448]) – use structural separation to report coevolving pairs

• **outname** (*str*[^449]) – output filename, default is mutinfo_rankorder.txt

### 3.11.11 MSA Refinement

Refine MSA application.

```python
def evol_refine(msa, **kwargs):
    # Refine an MSA by removing gapped rows/colums.
```

**Parameters**

- **msa** – MSA filename to be refined

**Refinement Options**

**Parameters**

• **label** (*str*[^450]) – sequence label, UniProt ID code, or PDB and chain identifier

• **seqid** (*float*[^451]) – identity threshold for selecting unique sequences

• **colocc** (*float*[^452]) – column (residue position) occupancy

• **rowocc** (*float*[^453]) – row (sequence) occupancy

• **pdbres** (*bool*[^454]) – keep columns corresponding to residues not resolved in PDB structure, applies label argument is a PDB identifier

**Output Options**

[^442]: http://docs.python.org/library/functions.html#str
[^443]: http://docs.python.org/library/functions.html#str
[^444]: http://docs.python.org/library/functions.html#str
[^445]: http://docs.python.org/library/functions.html#int
[^446]: http://docs.python.org/library/functions.html#int
[^447]: http://docs.python.org/library/functions.html#float
[^448]: http://docs.python.org/library/functions.html#bool
[^449]: http://docs.python.org/library/functions.html#str
[^450]: http://docs.python.org/library/functions.html#str
[^451]: http://docs.python.org/library/functions.html#float
[^452]: http://docs.python.org/library/functions.html#float
[^453]: http://docs.python.org/library/functions.html#float
[^454]: http://docs.python.org/library/functions.html#bool
Parameters

- `outname (str)` – output filename, default is msa filename with _refined suffix
- `format (str)` – output MSA file format, default is same as input
- `compressed (bool)` – gzip refined MSA output

### 3.11.12 Pfam Search

Pfam search application.

**evol_search (query, **kwargs)**

Search Pfam with given `query`.

**Parameters**

- `query` – protein UniProt ID or sequence, a PDB identifier, or a sequence file, where sequence have no gaps and 12 or more characters

**Sequence Search Options**

**Parameters**

- `search_b (bool)` – search Pfam-B families
- `skip_a (bool)` – do not search Pfam-A families
- `ga (bool)` – use gathering threshold
- `evalue (float)` – e-value cutoff, must be less than 10.0
- `timeout (int)` – timeout in seconds for blocking connection attempt, default is 60

**Output Options**

**Parameters**

- `outname (str)` – name for output file, default is standard output
- `delimiter (str)` – delimiter for output data columns, default is ‘	’

### 3.11.13 PDB Model/Structure Alignment

Align models in a PDB file or multiple structures in separate PDB files.

**prody_align (*pdbs, **kwargs)**

Align models in a PDB file or multiple structures in separate PDB files. By default, protein chains will be matched based on selected atoms and alignment will be performed based on matching residues. If non-protein atoms are selected and selected atoms match in multiple structures, they will be used for alignment.

**Parameters**

- `pdbs` – PDB identifier(s) or filename(s)
3.11.14 ANM Application

Perform ANM calculations and output the results in plain text, NMD, and graphical formats.

`prody_anm(pdb, **kwargs)`
Perform ANM calculations for `pdb`.

Parameters

- `cutoff` – cutoff distance (Å), default is 15.0
- `extend` – write NMD file for the model extended to “backbone” (“bb”) or “all” atoms of the residue, model must have one node per residue, default is ”
- `figall` – save all figures, default is `False`
- `figbeta` – save beta-factors figure, default is `False`
- `figcc` – save cross-correlations figure, default is `False`
- `figcmap` – save contact map (Kirchhoff matrix) figure, default is `False`
- `figdpi` – figure resolution (dpi), default is 300
- `figformat` – figure file format, default is `'pdf'`
- `figheight` – figure height (inch), default is 6.0
- `figmode` – save mode shape figures for specified modes, e.g. “1-3 5” for modes 1, 2, 3 and 5, default is ”
- `figsf` – save square-fluctuations figure, default is `False`
- `figwidth` – figure width (inch), default is 8.0
- `gamma` – spring constant, default is 1.0
- `hessian` – write Hessian matrix, default is `False`
- `kirchhoff` – write Kirchhoff matrix, default is `False`
- `model` – index of model that will be used in the calculations, default is 1
- `nmodes` – number of non-zero eigenvectors (modes) to calculate, default is 10
- `numdelim` – number delimiter, default is `' '`
- `numext` – numeric file extension, default is `'txt'`
- `numformat` – number output format, default is `'%12g'`
- `outall` – write all outputs, default is `False`
- `outbeta` – write beta-factors calculated from GNM modes, default is `False`
- `outcc` – write cross-correlations, default is `False`
• **outcov** – write covariance matrix, default is False
• **outdir** – output directory, default is ‘.’
• **outeig** – write eigenvalues/vectors, default is False
• **outhm** – write cross-correlations heatmap file, default is False
• **outnpz** – write compressed ProDy data file, default is False
• **outsf** – write square-fluctuations, default is False
• **prefix** – output file prefix, default is ‘_anm’
• **select** – atom selection, default is "protein and name CA or nucleic and name P C4’ C2”

### 3.11.15 Biomolecule Builder

Generate biomolecule structure using the transformation from the header section of the PDB file.

```python
prody_biomol (pdbname, **kwargs)
```

Generate biomolecule coordinates.

**Parameters**

- **pdb** – PDB identifier or filename
- **prefix** – prefix for output files, default is _biomol
- **biomol** – index of the biomolecule, by default all are generated

### 3.11.16 Blast Search PDB

Blast Protein Data Bank for structures matching a user given sequence.

```python
prody_blast (sequence, **kwargs)
```

Blast search PDB and download hits.

**Parameters**

- **sequence** – sequence or file in fasta format
- **identity** (float) – percent sequence identity for blast search, default is 90.0
- **overlap** (float) – percent sequence overlap between sequences, default is 90.0
- **outdir** (str) – download uncompressed PDB files to given directory
- **gzip** – write compressed PDB file

**Blast Parameters**

**Parameters**

- **filename** (str) – a filename to save the results in XML format
- **hitlist_size** (int) – search parameters, default is 250

---

465 [http://docs.python.org/library/functions.html#float](http://docs.python.org/library/functions.html#float)
466 [http://docs.python.org/library/functions.html#float](http://docs.python.org/library/functions.html#float)
467 [http://docs.python.org/library/functions.html#str](http://docs.python.org/library/functions.html#str)
468 [http://docs.python.org/library/functions.html#str](http://docs.python.org/library/functions.html#str)
469 [http://docs.python.org/library/functions.html#int](http://docs.python.org/library/functions.html#int)
• **expect** (*float*[^70]) – search parameters, default is 1e-10
• **sleep** (*int*[^71]) – how long to wait to reconnect for results, default is 2 sleep time is doubled when results are not ready.
• **timeout** (*int*[^72]) – when to give up waiting for results. default is 30

### 3.11.17 DCD Files Concatenation

Concatenate, slice, and/or select DCD files.

```python
prody_catdcd(*dcd, **kwargs)
```

Concatenate dcd files.

**Parameters**

- **select** – atom selection
- **align** – atom selection for aligning frames
- **pdb** – PDB file used in atom selections and as reference for alignment
- **psf** – PSF file used in atom selections
- **output** – output filename
- **first** – index of the first output frame
- **last** – index of the last output frame
- **stride** – number of steps between output frames

### 3.11.18 Contact Identification

This module defines a routine for contact identification.

```python
prody_contacts(**kwargs)
```

Identify contacts of a target structure with one or more ligands. Contacting atoms (or extended subset of atoms, such as residues) are outputted in PDB file format.

**Parameters**

- **target** – target PDB identifier or filename
- **ligand** – ligand PDB identifier(s) or filename(s)
- **select** – atom selection string for target structure
- **radius** – contact radius (Å), default is 4.0
- **extend** – output same ‘residue’, ‘chain’, or ‘segment’ along with contacting atoms
- **prefix** – prefix for output file, default is target filename
- **suffix** – output filename suffix, default is ligand filename

[^70]: http://docs.python.org/library/functions.html#float
[^71]: http://docs.python.org/library/functions.html#int
[^72]: http://docs.python.org/library/functions.html#int
3.11.19 PDB File Fetcher

Download PDB files for given identifiers.

\[ \text{prody_fetch}(*\text{pdb}, **\text{kwargs}) \]

Fetch PDB files from PDB FTP server.

**Parameters**

- \text{pdb} – PDB identifier(s) or filename(s)
- \text{dir} – target directory for saving PDB file(s), default is ‘.’
- \text{gzip} – gzip fetched files or not, default is \text{False}

3.11.20 GNM Application

Perform GNM calculations and output the results in plain text NMD, and graphical formats.

\[ \text{prody_gnm}(\text{pdb}, **\text{kwargs}) \]

Perform GNM calculations for \text{pdb}.

**Parameters**

- \text{cutoff} – cutoff distance (Å), default is 10.0
- \text{extend} – write NMD file for the model extended to “backbone” (“bb”) or “all” atoms of the residue, model must have one node per residue, default is ”
- \text{figall} – save all figures, default is \text{False}
- \text{figbeta} – save beta-factors figure, default is \text{False}
- \text{figcc} – save cross-correlations figure, default is \text{False}
- \text{figcmap} – save contact map (Kirchhoff matrix) figure, default is \text{False}
- \text{figdpi} – figure resolution (dpi), default is 300
- \text{figformat} – figure file format, default is ‘pdf’
- \text{figheight} – figure height (inch), default is 6.0
- \text{figmode} – save mode shape figures for specified modes, e.g. “1-3 5” for modes 1, 2, 3 and 5, default is ”
- \text{figsf} – save square-fluctuations figure, default is \text{False}
- \text{figwidth} – figure width (inch), default is 8.0
- \text{gamma} – spring constant, default is 1.0
- \text{kirchhoff} – write Kirchhoff matrix, default is \text{False}
- \text{model} – index of model that will be used in the calculations, default is 1
- \text{nmodes} – number of non-zero eigenvectors (modes) to calculate, default is 10
- \text{numdelim} – number delimiter, default is ’ ’
- \text{numext} – numeric file extension, default is ’.txt’
- \text{numformat} – number output format, default is ’%12g’
- \text{outall} – write all outputs, default is \text{False}
- \text{outbeta} – write beta-factors calculated from GNM modes, default is \text{False}
• **outcc** – write cross-correlations, default is False
• **outcov** – write covariance matrix, default is False
• **outdir** – output directory, default is ‘.’
• **outeig** – write eigenvalues/vectors, default is False
• **outhm** – write cross-correlations heatmap file, default is False
• **outnpz** – write compressed ProDy data file, default is False
• **outsf** – write square-fluctuations, default is False
• **prefix** – output file prefix, default is ‘_gnm’
• **select** – atom selection, default is "protein and name CA or nucleic and name P C4’ C2"

### 3.11.21 PCA Application

Perform PCA/EDA calculations and output the results in plain text, NMD, and graphical formats.

```python
prody_pca(coords, **kwargs)
```

Perform PCA calculations for PDB or DCD format `coords` file.

**Parameters**

- **aligned** – trajectory is already aligned, default is False
- **extend** – write NMD file for the model extended to “backbone” (“bb”) or “all” atoms of the residue, model must have one node per residue, default is ”
- **figall** – save all figures, default is False
- **figcc** – save cross-correlations figure, default is False
- **figdpi** – figure resolution (dpi), default is 300
- **figformat** – figure file format, default is ’pdf’
- **figheight** – figure height (inch), default is 6.0
- **figproj** – save projections onto specified subspaces, e.g. “1,2” for projections onto PCs 1 and 2; “1,2 1,3” for projections onto PCs 1,2 and 1, 3; “1 1,2,3” for projections onto PCs 1 and 1, 2, 3, default is ”
- **figsf** – save square-fluctuations figure, default is False
- **figwidth** – figure width (inch), default is 8.0
- **nmodes** – number of non-zero eigenvectors (modes) to calculate, default is 10
- **numdelim** – number delimiter, default is ’ ‘
- **numext** – numeric file extension, default is ’.txt’
- **numformat** – number output format, default is ’%12g’
- **outall** – write all outputs, default is False
- **outcc** – write cross-correlations, default is False
- **outcov** – write covariance matrix, default is False
- **outdir** – output directory, default is ’.’
- **outeig** – write eigenvalues/vectors, default is False
• **outhm** – write cross-correlations heatmap file, default is False
• **outnpz** – write compressed ProDy data file, default is False
• **outproj** – write projections onto PCs, default is False
• **outsf** – write square-fluctuations, default is False
• **prefix** – output file prefix, default is '_pca'
• **select** – atom selection, default is "protein and name CA or nucleic and name P C4’ C2"

### 3.11.22 Atom Selection

Extract a selection of atoms from a PDB file.

**prody_select** *(selstr, *pdbs, **kwargs)*

Write selected atoms from a PDB file in PDB format.

**Parameters**

- **selstr** – atom selection string, see *Atom Selections* (page 78)
- **pdbs** – PDB identifier(s) or filename(s)
- **output** – output filename, default is pdb_selected.pdb
- **prefix** – prefix for output file, default is PDB filename
- **suffix** – output filename suffix, default is _selected

### 3.12 Configuration & Logging

This module defines functions for logging in files, configuring ProDy, and running tests.

- **confProDy()** (page 204)
- **checkUpdates()** (page 205)
- **startLogfile()** (page 205)
- **closeLogfile()** (page 205)
- **plog()** (page 205)

**confProDy** *(*args, **kwargs)*

Configure ProDy.

<table>
<thead>
<tr>
<th>Option</th>
<th>Default (acceptable values)</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_secondary</td>
<td>False</td>
</tr>
<tr>
<td>auto_show</td>
<td>True</td>
</tr>
<tr>
<td>backup</td>
<td>False</td>
</tr>
<tr>
<td>backup_ext</td>
<td><code>.BAK</code></td>
</tr>
<tr>
<td>check_updates</td>
<td>0</td>
</tr>
<tr>
<td>ligand_xml_save</td>
<td>False</td>
</tr>
<tr>
<td>local_pdb_folder</td>
<td>&quot;See also pathPDBFolder() (page 160).&quot;</td>
</tr>
<tr>
<td>pdb_mirror_path</td>
<td>&quot;See also pathPDBMirror() (page 160).&quot;</td>
</tr>
<tr>
<td>selection_warning</td>
<td>True</td>
</tr>
<tr>
<td>typo_warnings</td>
<td>True</td>
</tr>
</tbody>
</table>
Usage example:

```python
confProDy('backup')
confProDy('backup', 'backup_ext')
confProDy(backup=True, backup_ext='.bak')
confProDy(backup_ext='.BAK')
```

**checkUpdates()**
Check PyPI to see if there is a newer ProDy version available. Setting ProDy configuration parameter `check_updates` to a positive integer will make ProDy automatically check updates, e.g.:

```python
confProDy(check_updates=7) # check at most once a week
confProDy(check_updates=0) # do not auto check updates
confProDy(check_updates=-1) # check at the start of every session
```

**startLogfile (filename, **kwargs)**
Start a logfile. If `filename` does not have an extension. `.log` will be appended to it.

**Parameters**

- `filename` – name of the logfile
- `mode` – mode in which logfile will be opened, default is “w”
- `backupcount` – number of existing `filename.log` files to backup, default is 1

**closeLogfile (filename)**
Close logfile with `filename`.

**plog(*text)**
Log `text` using ProDy logger at log level info. Multiple arguments are accepted. Each argument will be converted to string and joined using a white space as delimiter.
CHAPTER
FOUR

DEVELOPER’S GUIDE

4.1 Contributing to ProDy

- Install Git and a GUI (page 206)
- Fork and Clone ProDy (page 206)
- Setup Working Environment (page 207)
- Modify, Test, and Commit (page 207)
- Push and Pull Request (page 208)
- Update Local Copy (page 208)

4.1.1 Install Git and a GUI

ProDy source code is managed using Git\(^1\) distributed revision controlling system. You need to install git, and if you prefer a GUI for it, on your computer to be able to contribute to development of ProDy.

On Debian/Ubuntu Linux, for example, you can run the following to install git and gitk:

```
$ sudo apt-get install git gitk
```

For other operating systems, you can obtain installation instructions and files from Git\(^2\).

You will only need to use a few basic git commands. These commands are provided below, but usually without an adequate description. Please refer to Git book\(^3\) and Git docs\(^4\) for usage details and examples.

4.1.2 Fork and Clone ProDy

ProDy source code an issue tracker are hosted on Github\(^5\). You need to create an account on this service, if you do not have one already.

If you work on Mac OS or Windows, you may consider getting GitHub Mac\(^6\) or GitHub Windows\(^7\) to help you manage a copy of the repository.

---

\(^1\)http://git-scm.com/downloads
\(^2\)http://git-scm.com/downloads
\(^3\)http://git-scm.com/book
\(^4\)http://git-scm.com/docs
\(^5\)http://github.com/prody/ProDy
\(^6\)http://mac.github.com
\(^7\)http://windows.github.com
Once you have an account, you need to make a fork of ProDy, which is creating a copy of the repository in your account. You will see a link for this on ProDy\(^8\) source code page. You will have write access to this fork and later will use it share your changes with others.

The next step is cloning the fork from your online account to your local system. If you are not using the GitHub software, you can do it as follows:

```
$ git clone https://github.com/prody/ProDy.git
```

This will create ProDy folder with a copy of the project files in it:

```
$ cd ProDy
$ ls
bdist_wininst.bat  docs  INSTALL.rst  LICENSE.rst  Makefile
MANIFEST.in  prody  README.rst  scripts  setup.py
```

### 4.1.3 Setup Working Environment

You can use ProDy directly from this clone by adding ProDy folder to your PYTHONPATH\(^9\) environment variable, e.g.:

```
export PYTHONPATH=$PYTHONPATH:$/home/USERNAME/path/to/ProDy
```

This will not be enough though, since you also need to compile C extensions. You can run the following series of commands to build and copy C modules to where they need to be:

```
$ cd ProDy
$ python setup.py build_ext --inplace --force
```

or, on Linux you can:

```
$ make build
```

You may also want to make sure that you can run ProDy Applications (page 3) from anywhere on your system. One way to do this by adding ProDy/scripts folder to your PATH\(^10\) environment variable, e.g.:

```
export PATH=$PATH:$/home/USERNAME/path/to/ProDy/scripts
```

### 4.1.4 Modify, Test, and Commit

When modifying ProDy files you may want to follow the Style Guide for ProDy (page 211). Closely following the guidelines therein will allow for incorporation of your changes to ProDy quickly.

If you changed .py files, you should ensure to check the integrity of the package. To do this, you should at least run fast ProDy tests as follows:

```
$ cd ProDy
$ nosetests
```

See Testing ProDy (page 214) for alternate and more comprehensive ways of testing. ProDy unittest suit may not include a test for the function or the class that you just changed, but running the tests will ensure that the ProDy package can be imported and run without problems.

---

\(^8\)http://prody.csb.pitt.edu  
\(^9\)http://matplotlib.sourceforge.net/faq/environment_variables_faq.html#envvar-PYTHONPATH  
\(^10\)http://matplotlib.sourceforge.net/faq/environment_variables_faq.html#envvar-PATH
After ensuring that the package runs, you can commit your changes as follows:

```bash
$ git commit modified_file_1.py modified_file_2.py
```

or:

```bash
$ git commit -a
```

This command will open a text editor for you to describe the changes that you just committed.

### 4.1.5 Push and Pull Request

After you have committed your changes, you will need to push them to your Bitbucket account:

```bash
git push origin master
```

This step will ask for your account user name. If you are going to push to your GitHub/Bitbucket account frequently, you may add an SSH key for automatic authentication. To add an SSH key for your system, go to `Edit Your Profile → SSH keys` page on GitHub or `Manage Account → SSH keys` page on Bitbucket.

After pushing your changes, you will need to make a pull request from your to notify ProDy developers of the changes you made and facilitate their incorporation to ProDy.

### 4.1.6 Update Local Copy

You can also keep an up-to-date copy of ProDy by pulling changes from the master ProDy repository on a regular basis. You need add to the master repository as a remote to your local copy. You can do this running the following command from the ProDy project folder:

```bash
$ cd prody
$ git remote add prodymaster git@github.com:abakan/ProDy.git
```

or:

```bash
$ cd prody
$ git remote add prodymaster git@bitbucket.org:abakan/prody.git
```

You may use any name other than `prodymaster`, but `origin`, which points to the ProDy fork in your account.

After setting up this remote, calling `git pull` command will fetch latest changes from ProDy master repository and merge them to your local copy:

```bash
$ git pull prodymaster master
```

Note that when there are changes in C modules, you need to run the following commands again to update the binary module files:

```bash
$ python setup.py build_ext --inplace --force
```

### 4.2 Documenting ProDy

ProDy documentation is written using reStructuredText\textsuperscript{13} markup and prepared using Sphinx\textsuperscript{14}. You may install Sphinx using \texttt{easy_install}, i.e. \texttt{easy_install -U Sphinx}, or using package manager on your Linux machine.

### 4.2.1 Building Manual

ProDy Manual in HTML and PDF formats can be build as follows:

```bash
$ cd docs
$ make html
$ make pdf
```

If all documentation strings and pages are properly formatted according to reStructuredText\textsuperscript{15} markup, documentation pages should compile without any warnings. Note that to build PDF files, you need to install \texttt{latex} and \texttt{pdflatex} programs.

**Read the Docs**

A copy of ProDy manual is hosted on Read the Docs\textsuperscript{16} and can be viewed at http://prody.readthedocs.org/. Read the Docs is configured to build manual pages for the \texttt{devel} branch (latest) and the recent stable versions. The user name for Read the Docs is \texttt{prody}.

### 4.2.2 Building Website

ProDy-website source is hosted at https://github.com/prody/ProDy-website This project contains tutorial files and the home pages for ProDy and other related software.

**Latest version**

To build website on ProDy server, start with pulling changes:

```bash
$ cd ProDy-website
$ git pull
```

Running the following command will build HTML pages for the latest stable release of ProDy:

```bash
$ make html
```

HTML pages for manual and all tutorials are build as a single project, which allows for referencing from manual to tutorials.

PDF files for the manual and tutorials, and also download files are build as follows:

```bash
$ make pdf
```

PDF and TGZ/ZIP files are copied to appropriate places after they are built.

**Development version**

Finally, HTML and PDF pages for the development version can be built as follows:

\textsuperscript{13}http://docutils.sf.net/rst.html
\textsuperscript{14}http://sphinx.pocoo.org/
\textsuperscript{15}http://docutils.sf.net/rst.html
\textsuperscript{16}https://readthedocs.org/
$ make devel

Again, this will copy HTML and PDF files to appropriate places, and a link to these files will be provided from the homepage.

4.3 How to Make a Release

1. Make sure ProDy imports and passes all unit tests both Python 2 and Python 3, and using nose nosetests command:

   $ cd ProDy
   $ nosetests
   $ nosetests3

   See Testing ProDy (page 214) for more on testing.

2. Update the version number in:
   - prody/__init__.py
   Also, commend + '-dev' out, so that documentation will build for a stable release.

3. Update the most recent changes and the latest release date in:
   - docs/release/vX.Y_series.rst.

   If there is a new incremental release, start a new file.

4. Make sure the following files are up-to-date.
   - README.txt
   - MANIFEST.in
   - setup.py

   If there is a new file format, that is a new extensions not captured in MANIFEST.in, it should be included.

   If there is a new C extension, it should be listed in setup.py.

   After checking these files, commit change and push them to GitHub.

5. Generate the source distributions:

   $ cd ..
   $ python setup.py sdist --formats=gztar,zip

6. Prepare and test Windows installers (see Making Windows Installers (page 218)):

   $ C:\Python26\python setup.py bdist_wininst
   $ C:\Python27\python setup.py bdist_wininst
   $ C:\Python32\python setup.py bdist_wininst
   $ C:\Python33\python setup.py bdist_wininst

   Alternatively, use bdist_wininst.bat to run these commands. When there is a newer Python major release, it should be added to this list. Don’t forget to pull most recent changes to your Windows machine.

17http://github.com/prody/ProDy
A good practice is installing ProDy using all newly created installers and checking that it works. ProDy script can be used to check that, e.g.:

$ C:\Python33\Scripts\prody.bat anm lubi

If this command runs for all supported Python versions, release is good to go.

7. Register new release to PyPI:

   $ python setup.py register

   This will offer a number of options. ProDy on PyPI is owned by user prody.devel.

8. Upload the new release files to the PyPI\(^{18}\).

9. Commit final changes, if there are any:

   $ cd ..
   $ git commit -a

10. Tag the repository with the current version number:

    $ git tag vX.Y

11. Rebase \texttt{devel} branch to \texttt{master}:

    $ git checkout master
    $ git rebase devel

12. Push the changes with the new tag:

    $ git checkout master
    $ git push --tags
    $ git checkout devel
    $ git push --tags

13. Update the documentation on ProDy\(^{19}\) website. See Documenting ProDy (page 208).

14. Now that you made a release, you can go back to development. You may start with appending '-dev' to \texttt{__release__} in \texttt{prody/__init__.py}.

### 4.4 Style Guide for ProDy

- Introduction (page 211)
- Code Layout (page 212)
- Whitespaces (page 212)
- Naming Conventions (page 213)
- Variable Names (page 213)

#### 4.4.1 Introduction

PEP 8\(^{20}\), the Style Guide for Python Code, is adopted in the development of ProDy package. Contributions to

\(^{18}\)http://pypi.python.org/pypi/ProDy

\(^{19}\)http://prody.csb.pitt.edu

\(^{20}\)http://www.python.org/dev/peps/pep-0008
ProDy shall follow PEP 8\(^{21}\) and the specifications and additions provided in this addendum.

### 4.4.2 Code Layout

**Indentation**

Use 4 spaces per indentation level in source code (.py) and never use tabs as a substitute.

In documentation files (.rst), use 2 spaces per indentation level.

**Maximum line length**

Limit all lines to a maximum of 79 characters in both source code and documentation files. Exceptions may be made when tabulating data in documentation files and strings. The length of lines in a paragraph may be much less than 79 characters if the line ends align better with the first line, as in this paragraph.

**Encodings**

In cases where an encoding for a .py file needs to be specified, such as when characters like α, β, or Å are used in docstrings, use UTF-8 encoding, i.e. start the file with the following line:

```python
# -*- coding: utf-8 -*-
```

**Imports**

In addition to PEP 8\(^{22}\) recommendations regarding imports, the following should be applied:

- relative intra-ProDy imports are discouraged, use `from prody.atomic import AtomGroup` not `from atomic import AtomGroup`
- always import from second top level module, use `from prody.atomic import AtomGroup` and not `from prody.atomic.atompgroup import AtomGroup`, because file names may change or files that grow too big may be split into smaller modules, etc.

Here is a series of properly formatted imports following a module documentation string:

```python
"""This module defines a function to calculate something interesting."""

import os.path
from collections import defaultdict
from time import time
import numpy as np

from prody.atomic import AtomGroup
from prody.measure import calcRMSD
from prody.tools import openFile
from prody import LOGGER, SETTINGS

__all__ = ['calcSomething']
```

### 4.4.3 Whitespaces

In addition to recommendations regarding whitespace use in Python code (PEP 8\(^{23}\)), two whitespace characters should follow a period in documentation files and strings to help reading documentation in terminal windows and text editors.

---

\(^{21}\)http://www.python.org/dev/peps/pep-0008
\(^{22}\)http://www.python.org/dev/peps/pep-0008#imports
\(^{23}\)http://www.python.org/dev/peps/pep-0008#whitespace-in-expressions-and-statements

---

4.4. Style Guide for ProDy
4.4.4 Naming Conventions

ProDy naming conventions aim at making the library suitable for interactive sessions, i.e. easy to remember and type.

Class names

Naming style for classes is **CapitalizedWords** (or **CapWords**, or **CamelCase**). Abbreviations and/or truncated names should be used to keep class names short. Some class name examples are:

- ANM (page 101) for Anisotropic Network Model
- HierView (page 67) for Hierarchical View

Exception names

Prefer using a suitable standard-library exception over defining a new one. If you absolutely need to define one, use the class naming convention. Use the suffix “Error” for exception names, when exception is an error:

- **SelectionError** (page 85), the only exception defined in ProDy package

Method and function names

Naming style for methods and functions is **mixedCase**, that differs from **CapWords** by initial lowercase character. Starting with a lowercase (no shift key) and using no underscore characters decreases the number of key strokes by half in many cases in interactive sessions.

Method and function names should start with a verb, suggestive on the action, and followed by one or two names, where the second name may start with a lower case letter. Some examples are **moveAtoms()** (page 144), **wrapAtoms()** (page 145), **assignSecstr()** (page 158), and **calcSubspaceOverlap()** (page 103).

Abbreviations and/or truncated names should be used and obvious words should be omitted to limit number of names to 20 characters. For example, **buildHessian()** (page 101) is preferred over **buildHessianMatrix()**. Another example is the change from using **getResidueNames()** to using **AtomGroup.getResnames()** (page 41). In fact, this was part of a series of major **Release Notes** (page 219) aimed at refining the library for interactive usage.

In addition, the following should be applied to enable grouping of methods and functions based on their action and/or return value:

- **buildSomething()**: methods and functions that calculate a matrix should start with build, e.g. **GNM.buildKirchhoff()** (page 112) and **buildDistMatrix()** (page 140)
- **calcSomething()**: methods that calculate new data but does not necessarily return anything and especially those that take timely actions, should start with calc, e.g. **PCA.calcModes()** (page 120)
- **getSomething()**: methods, and sometimes functions, that return a copy of data should start with get, such as **listReservedWords()** (page 66)
- **setSomething()**: methods, and sometimes functions, that alter internal data should start with set

4.4.5 Variable Names

Variable names in functions and methods should contain only lower case letters, and may contain underscore characters to increase readability.
4.5 Testing ProDy

4.5.1 Running Unittests

The easiest way to run ProDy unit tests is using nose\(^{24}\). The following will run all tests:

\$ nosetests prody

To skip tests that are slow, use the following:

\$ nosetests prody -a '=slow'

To run tests for a specific module do as follows:

\$ nosetests prody.tests.atomic prody.tests.sequence

4.5.2 Unittest Development

Unit test development should follow these guidelines:

1. For comparing Python numerical types and objects, e.g. int, list, tuple, use methods of unittest.TestCase\(^{25}\).
2. For comparing Numpy arrays, use assertions available in numpy.testing module.
3. All test files should be stored in tests folder in the ProDy package directory, i.e. prody/tests/
4. All tests for functions and classes in a ProDy module should be in a single test file named after the module, e.g. test_atomic/test_select.py.
5. Data files for testing should be located in tests/test_datafiles.

4.6 Writing Tutorials

This is a short guide for writing ProDy tutorials that are published as part of online documentation pages, and also as individual downloadable PDF files.

\(^{24}\)http://nose.readthedocs.org

\(^{25}\)http://docs.python.org/library/unittest.html#unittest.TestCase
4.6.1 Tutorial Setup

First go to doc folder in ProDy package and generate necessary files for your tutorial using start-tutorial.sh script:

```
$ cd doc
$ ./start-tutorial.sh
```

Enter tutorial title: ENM Analysis using ProDy
Enter a short title: ENM Analysis
Enter author name: First Last

This will generate following folder and files:

```
$ cd tutorials/enm_analysis/
$ ls -lgo
-rw-r--r-- 1 328 Apr 30 16:48 conf.py
-rw-r--r-- 1 395 Apr 30 16:48 index.rst
-rw-r--r-- 1 882 Apr 30 16:48 intro.rst
-rw-r--r-- 1 1466 Apr 30 16:48 Makefile
lrwxrwxrwx 1 13 Apr 30 16:48 _static -> ../../_static
```

Note that short title will be used as filename and part of the URL of the online documentation pages.

If tutorial logo/image that you want to use is different from ProDy logo, update the following line in conf.py:

```python
tutorial_logo = u'enm.png'  # default is ProDy logo
```

Also, note ProDy version if the tutorial is developed for a specific release.

4.6.2 Style and Organization

ProDy documentation and tutorials are written using reStructuredText\footnote{http://docutils.sourceforge.net/rst.html}, an easy-to-read/write file format. See reStructuredText Primer\footnote{http://sphinx-doc.org/rest.html} for a quick introduction.

reStructuredText is stored in plain-text files with .rst extension, and converted to HTML and PDF pages using Sphinx\footnote{http://sphinx-doc.org/}.

index.rst and intro.rst files are automatically generated. index.rst file should include title and table of contents of the tutorial. Table of contents is just a list of .rst files that are part of the tutorial. They be listed in the order that they should appear in the final PDF file:

```rst
.. _enm-analysis:

ENM Analysis using ProDy

.. add .rst files to 'toctree' in the order that you want them
```

\footnote{http://docutils.sourceforge.net/rst.html}
\footnote{http://sphinx-doc.org/rest.html}
\footnote{http://sphin.html}
Add more .rst files as needed. See other tutorials in doc/tutorials folder as examples.

### 4.6.3 Input/Output Files

All files needed to follow the tutorial should be stored in tutorial_name_files folder. There is usually no need to provide PDB files, as ProDy automatically downloads them when needed. Optionally, output files can also be provided.

**Note:** Small input and output files that contain textual information may be included in the git repository, but please avoid including large files in particular those that contain binary data.

### 4.6.4 Including Code

Python code in tutorials should be included using IPython Sphinx directive\(^\text{29}\). In the beginning of each .rst file, you should make necessary imports as follows:

```ipython:: python
from prody import *
from matplotlib.pylab import *
ion()
```

This will convert to the following:

**In [1]:** from prody import *

**In [2]:** from matplotlib.pylab import *

**In [3]:** ion()

Then you can add the code for the tutorial:

```ipython:: python
pdb = parsePDB('1p38')
```

**In [4]:** pdb = parsePDB('1p38')

### 4.6.5 Including Figures

IPython directive should also be used for including figures:

```ipython:: python
@savefig tutorial_name_figure_name.png width=4in
```

plot(range(10))

@savefig tutorial_name_figure_two.png width=4in
plot(range(100)); # used ; to suppress output

@savefig decorator was used to save the figure.

**Note:** Figure names needs to be unique within the tutorial and should be prefixed with the tutorial name.

Note that in the second `plot()` call, we used a semicolon to suppress the output of the function.

If you want to make modifications to the figure, save it after the last modification:

```
.. ipython:: python
   :width: 80%

   plot(range(10));
   grid();
   xlabel('X-axis')
   @savefig tutorial_name_figure_three.png width=4in
   ylabel('Y-axis')
```

### 4.6.6 Testing Code

If there is any particular code output that you want to test, you can use `@doctest` decorator as follows:

```
.. ipython::
   :width: 80%

   @doctest
   In [1]: 2 + 2
   Out[1]: 4

   In [5]: 2 + 2
   Out[5]: 4
```

Failing to produce the correct output will prevent building the documentation.

### 4.6.7 Publishing Tutorial

To see how your `.rst` files convert to HTML format, use the following command:

```
$ make html
```

You will find HTML files in `_build/html` folder.

Once your tutorial is complete and looks good in HTML (no code execution problems), following commands can be used to generate a PDF file and tutorial file achieves:

```
$ make pdf
$ make files
```

ProDy online documentation will contain these files as well as tutorial pages in HTML format.

---

Note: Plotting functions in Matplotlib are defined in


---
4.7 Making Windows Installers

MinGW\(^\text{31}\) can be used for compiling C modules when making Windows installers. Install MinGW and make `distutils.cfg` file in `PythonXY\Lib\distutils` folder that contains:

```
[build]
compiler = mingw32
```

4.8 Cross-platform Issues

- **Numpy integer type** (page 218)
- **Relative paths** (page 218)

This section describes cross-platform issues that may emerge and provides possible solutions for them.

4.8.1 Numpy integer type

Issues may arise when comparing Numpy integer types with Python `int()`\(^\text{32}\). Python `int()`\(^\text{33}\) equivalent Numpy integer type on Windows (Win7 64bit, Python 32bit) is `int32`, while on Linux (Ubuntu 64bit) it is `int64`. For example, the statement `isinstance(np.array([1], np.int64), int)` may return `False` resulting in unexpected behavior in ProDy functions or methods. If Numpy integer type needs to be specified, using `int` seems a safe option.

4.8.2 Relative paths

`os.path.relpath()`\(^\text{34}\) function raises exceptions when the working directory and the path of interest are on separate drives, e.g. trying to write a `C:\temp` while running tests on `D:\ProDy`. Instead of this `os.path.relpath()`\(^\text{35}\), ProDy function `relpath()` (page 191) should be used to avoid problems.

\(^{31}\)http://www.mingw.org/
\(^{32}\)http://docs.python.org/library/functions.html#int
\(^{33}\)http://docs.python.org/library/functions.html#int
\(^{34}\)http://docs.python.org/library/os.path.html#os.path.relpath
\(^{35}\)http://docs.python.org/library/os.path.html#os.path.relpath
RELEASE NOTES

5.1 ProDy 1.5 Series

5.1.1 1.5.1 (Dec 24, 2013)

Changes:

• PDBBlastRecord become picklable.

5.1.2 1.5 (Dec 23, 2013)

New Features:

• buildDirectInfoMatrix() (page 169) and calcMeff() (page 169) are implemented for calculation of direct information from multiple sequence alignments.

• showDirectInfoMatrix() (page 173) and showSCAMatrix() (page 173) functions are implemented for displaying coevolutionary data.

• RTB (page 127) is implemented for Rotations-Translations of Blocks calculations. Optional arguments also permit imANM calculations.

Availability:

• Source is moved from lib/prody to prody.

• Source code will be hosted only at GitHub¹.

Improvements:

• DCDFile (page 175) and parseDCD() (page 177) support DCD files written by cpptraj.

Testing:

• ProDy test command (prody test) and function prody.test() has been removed for easier maintenance of testing functions. See Testing ProDy (page 214) for more information on how to test ProDy.

¹http://github.com/prody/ProDy
5.2 ProDy 1.4 Series

- 1.4.9 (Nov 14, 2013) (page 220)
- 1.4.8 (Nov 4, 2013) (page 221)
- 1.4.7 (Oct 29, 2013) (page 221)
- 1.4.6 (Oct 16, 2013) (page 221)
- 1.4.5 (Sep 6, 2013) (page 222)
- 1.4.4 (July 22, 2013) (page 222)
- 1.4.3 (June 14, 2013) (page 222)
- 1.4.2 (April 19, 2013) (page 223)
- 1.4.1 (Dec 16, 2012) (page 223)
  - Normal Mode Wizard (page 224)
- 1.4 (Dec 2, 2012) (page 224)

5.2.1 1.4.9 (Nov 14, 2013)

Upcoming changes:
- Support for Python 3.1 and NumPy 1.5 will be dropped, meaning no Windows installers will be built for these versions of them.

Improvements:

- HierView (page 67) can handle Residue (page 69) instances that have same segment name, chain identifier, and resnum, if PDB file contains TER lines to terminate these residues. If these three identifiers are shared by multiple residues, indexing AtomGroup (page 37) instances will return a list of residues. This behavior can be used as follows. Note that in v1.5, this will be the default behavior.

```python
>>> from StringIO import StringIO

>>> pdb_lines = '''
... ATOM 1 O WAT A 1 4.694 -3.891 -0.592 1.00 1.00
... ATOM 2 H1 WAT A 1 5.096 -3.068 -0.190 1.00 1.00
... ATOM 3 H2 WAT A 1 5.420 -4.544 -0.808 1.00 1.00
... TER
... ATOM 4 O WAT A 1 -30.035 19.116 -2.193 1.00 1.00
... ATOM 5 H1 WAT A 1 -30.959 18.736 -2.244 1.00 1.00
... ATOM 6 H2 WAT A 1 -29.993 19.960 -2.728 1.00 1.00
... TER
... ATOM 7 O WAT A 1 -77.584 -21.524 -37.894 1.00 1.00
... ATOM 8 H1 WAT A 1 -77.226 -21.966 -38.717 1.00 1.00
... ATOM 9 H2 WAT A 1 -77.023 -20.726 -37.674 1.00 1.00
... TER"

>>> from StringIO import StringIO

>>> atoms = parsePDBStream(StringIO(pdb_lines))

Current behavior:

```python
>>> print(atoms.numResidues())
1
``` python

```python
>>> atoms[‘A’, 1]
<Residue: WAT 1 from Chain A from Unknown (9 atoms)>
```

To activate the new behavior (which will be the default behavior in v1.5):
>>> hv = atoms.getHierView(ter=True)
>>> print(hv.numResidues())
>>> hv['A', 1]

- `parsePDB()` (page 162) reads TER records in PDB files. Atoms and hetero atoms (*hetatm*) that are followed by a TER record are now flagged as *pdbter*.

**Bugfixes:**

- Fixed memory leaks in `uniqueSequences()` (page 168) and `buildSeqidMatrix()` (page 168).

### 5.2.2 1.4.8 (Nov 4, 2013)

**New Features:**

- New analysis functions `buildOMESMatrix()` (page 168) and `buildSCAMatrix()` (page 169) are implemented.
  - `AtomGroup.numBytes()` (page 42) method returns an estimate of memory usage.
  - `countBytes()` (page 190) utility function is added for counting bytes used by NumPy arrays.

**Improvements:**

- `parsePDB()` (page 162) resizes data arrays to decrease memory usage.

**Bugfixes:**

- Fixed memory leaks in MSA analysis (page 167) functions.
- Fixed potential problems with importing contributed libraries.

### 5.2.3 1.4.7 (Oct 29, 2013)

**Improvements:**

- `AtomGroup`, `Selection` (page 86), and other `Atomic` (page 44) classes are picklable.
  - Improved equality tests for `AtomGroup` (page 37). Two different instances are considered equal if they contain identical data and coordinate sets.

### 5.2.4 1.4.6 (Oct 16, 2013)

**Bugfixes:**

- Selection problem with using `resid` is fixed (issue 160)
  - Fixed a memory leak in MSA parsers written in C. When dealing with large files, leak would cause a segmentation fault.
  - Fixed a memory leak in MSA parsers written in C. When dealing with large files, leak would cause a segmentation fault.
  - Fixed a reference counting problem in MSA parsers in C that would cause segmentation fault when reading files that uses the same label for multiple sequences.
  - Updated `fetchPDBLigand()` (page 164) to use PDB for fetching XML files.
  - Revised handling of MSA file formats to avoid exceptions for unknown extensions.

---

2https://github.com/prody/ProDy/issues/160

5.2. ProDy 1.4 Series 221
5.2.5 1.4.5 (Sep 6, 2013)

New Features:

- `parsePDBHeader()` (page 157) function can parse space group information from header section specified as REMARK 290, e.g. `parsePDBHeader('1mkp', 'space_group')` or `parsePDBHeader('1mkp')['space_group']`
- `heavy` selection flag is defined as an alias for `noh`.
- `matchChains()` (page 149) function can match non-hydrogen atoms using `subset='heavy'` keyword argument.
- Added `update_coords` keyword argument to `PCA.buildCovariance()`, so that average coordinates calculated internally can be stored in ensemble or trajectory objects used as input.

Improvements:

- Unit tests can be run with Python 2.6 when `unittest2` module is installed.

Bugfixes:

- Fixed problems with reading compressed PDB files using Python 3.3.
- Fixed a bug in `parseSTRIDE()` (page 165) function that prevented reading files.
- Improved parsing of biomolecular transformations.
- Fixed memory allocation in C code used by `parseMSA()` (page 172) (Python 2.6).
- Fixed a potential name error in trajectory classes.
- Fixed problems in handling compressed files when using Python 2.6 and 3.3.
- Fixed a problem with indexing `NMA` (page 117) instances in Python 3 series.

5.2.6 1.4.4 (July 22, 2013)

Improvements:

- `writeNMD()` (page 119) and `parseNMD()` (page 119) write and read segment names. NMWiz is also improved to handle segment names. Improvements will be available in VMD v1.9.2.

Bugfixes:

- A bug in `saveAtoms()` (page 66) that would cause `KeyError` when bonds are set but fragments are not determined is fixed.
- Import ProDy would fail when `HOME` is not set. Changed `PackageSettings` (page 191) to handle this case graciously.

5.2.7 1.4.3 (June 14, 2013)

Changes:

- `getVMDpath()` (page 119) and `setVMDpath()` (page 119) functions are deprecated for removal, use `pathVMD()` (page 119) instead.
- Increased `blastPDB()` (page 149) `timeout` to 60 seconds.

---

3http://matplotlib.sourceforge.net/faq/environment_variables_faq.html#envvar-HOME
• `extendModel()` (page 104) and `extendMode()` (page 104) functions have a new option for normalizing extended mode(s).

• `sampleModes()` (page 128) and `traverseMode()` (page 129) automatically normalizes input modes.

**Bugfixes:**

• A bug in `applyTransformation()` (page 144) is fixed. The function would interpret some external transformation matrices incorrectly.

• A bug in `fetchPDBLigand()` (page 164) function is fixed.

5.2.8 1.4.2 (April 19, 2013)

**Improvements:**

• `fetchPDB()` (page 160) and `fetchPDBfromMirror()` (page 160) functions can handle partial PDB mirrors. See `pathPDBMirror()` (page 160) for setting a mirror path.

**Changes:**

• `MSE`\(^4\) is included in the definition of non-standard amino acids, i.e. `nonstdaa`.

**Bugfixes:**

• Atom selection problems related to using `all` and `none` in composite selections, e.g. `’calpha and all’`, is fixed by defining these keywords as `Atom Flags` (page 56).

• Fasta files with sequence labels using multiple pipe characters would cause C parser (and so `parseMSA()` (page 172)) to fail. This issue is fixed by completely disregarding pipe characters.

• Empty chain identifiers for PDB hits would cause a problem in parsing XML results file and `blastPDB()` (page 149) would throw an exception. This case is handled by slicing the chain identifier string.

• A problem in `viewNMDinVMD()` (page 120) related to module imports is fixed.

• A problem with handling weights in `loadEnsemble()` (page 134) is fixed.

5.2.9 1.4.1 (Dec 16, 2012)

**New Features:**

• `buildSeqidMatrix()` (page 168) and `uniqueSequences()` (page 168) functions are implemented for comparing sequences in an `MSA` (page 169) object.

• `showHeatmap()` (page 114), `parseHeatmap()` (page 114), and `writeHeatmap()` (page 114) functions are implemented to support VMD plugin `Heat Mapper`\(^5\) file format.

• `Sequence` (page 173) is implemented to handle individual sequence records and point to sequences in `MSA` (page 169) instances.

• `evol occupancy` (page 23) application is implemented for refined MSA quality checking purposes.

• `mergeMSA()` (page 171) function and `evol merge` (page 22) application are implemented for merging Pfam MSA to study multi-domain proteins.

**Improvements:**

\(^4\)http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=MSE
\(^5\)http://www.ks.uiuc.edu/Research/vmd/plugins/heatmapper/
• **refineMSA()** (page 170) function and **evol refine** (page 26) application can perform MSA refinements by removing similar sequences.

• **writePDB()** (page 163) function takes beta and occupancy arguments to be outputted in corresponding columns.

• **MSA** (page 169) indexing and slicing are revised and improved.

• **parseMSA()** (page 172) is improved to handle indexing of sequences that have the same label in an MSA file, e.g. domains repeated in a protein.

• **prody anm** (page 4), **prody gnm** (page 11), and **prody pca** (page 13) applications can write heatmap files for visualization using NMWiz and Heatmapper plugins.

• Several improvements made to handling sequence labels in Pfam MSA files. Files that contain sequence parts with same protein UniProt ID are handled delicately.

**Changes:**

• ProDy will not emit a warning message when a wwPDB server is not set using **wwPDBServer()** (page 165), and use the default US server.

• Indexing **MSA** (page 169) returns **Sequence** (page 173) instances.

• Iterating over **MSA** (page 169) and **MSAFile** (page 171) yields **Sequence** (page 173) instances.

**Bugfixes:**

• Fixed a syntax problem that prevented running ProDy using Python 2.6.

• Fixed **NMA** (page 117) indexing problem that was introduced in v1.4.

**Normal Mode Wizard**

• NMWiz can visualize heatmaps linked to structural view via Heatmapper. Clicking on the heatmap will highlight atom or residue pairs.

• ProDy interface has the option to write and load cross-correlations.

• NMWiz can determined whether a model is an extended model. For extended models plotting mobility has been improved. Only a single value per residue will be plotted, and clicking on the plot will highlight all of the residue atoms.

**5.2.10 1.4 (Dec 2, 2012)**

**New Features:**

**Python 3 Support**

• ProDy has been refactored to support Python 3. Windows installers for Python 2.6, 2.7, 3.1, and 3.2 are available in **Installation** (page 1).

• Unit tests are compatible with Python 2.7 and 3.2, and running them with other versions gives errors due to unavailability of some unittest features.

**Sequence Analysis**

• New applications **Evol Applications** (page 16) are available.

---

6http://docs.python.org/library/unittest.html#unittest
• `searchPfam()` (page 95) and `fetchPfamMSA()` (page 95) functions are implemented for searching and retrieving Pfam data. See MSA Files\textsuperscript{7} for usage examples.

• `MSAFile` (page 171) class, `parseMSA()` (page 172) and `writeMSA()` (page 172) functions are implemented for reading and writing multiple sequence alignments. See MSA Files\textsuperscript{8} for usage examples.

• `MSA` (page 169) class has been implemented for storing and manipulating MSAs in memory.

• `calcShannonEntropy()` (page 167), `buildMutinfoMatrix()` (page 167), and `calcMSAOccupancy()` (page 167) functions are implemented for MSA analysis. See Evolution Analysis\textsuperscript{9} for usage examples.

• `showShannonEntropy()` (page 172), `showMutinfoMatrix()` (page 173), and `showMSAOccupancy()` (page 172) functions are implemented for MSA analysis. See Evolution Analysis\textsuperscript{10} for usage examples.

• `applyMutinfoCorr()` (page 167) and `applyMutinfoNorm()` (page 167) functions are implemented for applying normalization and corrections to mutual information matrices.

• `calcRankorder()` (page 168) function is implemented for identifying highly correlated/co-evolving pairs of residues.

Bugfix:
• Fixed selection issues involving use of $x$ or negative numbers.

5.3 ProDy 1.3 Series

• 1.3.1 (Nov 6, 2012) (page 225)
• 1.3 (Sep 30, 2012) (page 226)

5.3.1 1.3.1 (Nov 6, 2012)

New Features:
• Added `fetchPDBviaHTTP()` (page 165) and `fetchPDBviaFTP()` (page 165) functions.

• Added `copyFile()` (page 191) function to utilities (page 185).

• Added `prody test` command for convenient testing of ProDy package.

Improvements:
• Improved `gunzip()` (page 190) function to handle .gz extensions and string buffers.

Changes:
• `getWWPDBFTPServer()` and `setWWPDBFTPServer()` are deprecated for removal in v1.4, use `wwPDBServer()` (page 165) instead.

• `getPDBLocalFolder()` and `setPDBLocalFolder()` are deprecated for removal in v1.4, use `pathPDBFolder()` (page 160) instead.

\textsuperscript{7}http://prody.csb.pitt.edu/tutorials/evol_tutorial/msafiles.html#msafiles
\textsuperscript{8}http://prody.csb.pitt.edu/tutorials/evol_tutorial/msafiles.html#msafiles
\textsuperscript{9}http://prody.csb.pitt.edu/tutorials/evol_tutorial/msaanalysis.html#msa-analysis
\textsuperscript{10}http://prody.csb.pitt.edu/tutorials/evol_tutorial/msaanalysis.html#msa-analysis
• `getPDBMirrorPath()` and `setPDBMirrorPath()` are deprecated for removal in v1.4, use `pathPDBMirror()` (page 160) instead.
• `getPDBCluster()` is deprecated for removal in v1.4, use `listPDBCluster()` (page 161) instead.
• `getReservedWords()` is deprecated for removal in v1.4, use `listReservedWords()` (page 66) instead.
• `getNonstdProperties()` (page 65) is deprecated for removal in v1.4, use `listNonstdAAProps()` (page 65) instead.

Bugfix:
• Fixed a bug in `HierView` (page 67) that would cause wrong assignment of residue/chain indices to atoms when residue or chain atoms are separated by atoms of other entities. This would also caused problems when making keyword selections, such as `protein`.
• Added dummy atom check in `Ensemble.setAtoms()` (page 133) and `Trajectory.setAtoms()` (page 182) methods to avoid indexing problems.

5.3.2 1.3 (Sep 30, 2012)

Improvements:
• `select` (page 78) module and its documentation are completely rewritten. `Select` (page 85) class uses simplest possible parser to evaluate selection strings and achieves more than 25% speed-up on average.
• `Atom Selections` (page 78) become more forgiving of small typos, but will issue warning messages when they are detected via `SelectionWarning` (page 85). These messages can be turned off using `confProDy()` (page 204)
• Functions used in `ProDy Applications` (page 3) have been refactored to allow for using them directly. See `apps` (page 192) for their documentation.

Bugfix:
• A problem in `prody catcde` (page 7) command that was introduced when refactoring `trajectory` (page 174) classes is fixed.

5.4 ProDy 1.2 Series

5.4.1 1.2.1 (Sep 6, 2012)

If you are upgrading from ProDy v1.1, see also the below changes introduced in v1.2.

Bugfix:
• A problem in `select` module regarding Numpy numeric types is fixed. Problem would emerge on platforms which do not offer some numeric types, e.f. `np.float16`.

11http://docs.python.org/library/select.html#select
• Fixed problems in *prody anm* (page 4), *prody gnm* (page 11), and *prody fetch* (page 11) related to writing output files.

**Changes:**

• The way that *prody fetch* (page 11) command handles files containing PDB identifiers has changed.

### 5.4.2 1.2 (Aug 30, 2012)

**Important Changes:**

Package folder *prody* is moved into *lib* folder to prevent exceptions related to importing compiled packages from the installation folder.

Some changes in *Trajectory* (page 181) and *Ensemble* (page 132) methods related to linking, setting, and selecting atoms were made to make the interface more intuitive. These changes, which may break your code, are as follows:

• *AtomGroup* (page 37) instances can be linked to a *Trajectory* (page 181) using *Trajectory.link()* (page 181) method and linking status of an instance can be checked using *Trajectory.isLinked()* (page 181) method.

• *Trajectory.setAtoms()* (page 182) method accepts *AtomGroup* (page 37) and *Selection* (page 86) instances and should be used to select a subset of atoms. This method will not link *AtomGroup* (page 37) instance to the trajectory and also will not update the reference coordinates of the instance.

• *Trajectory.select()* and *Ensemble.select()* methods are removed and their functions are overloaded to *Trajectory.setAtoms()* (page 182) and *Ensemble.setAtoms()* (page 133) methods, respectively.

• *Trajectory.getSelection()* and *Ensemble.getSelection()* methods are removed, use *Trajectory.getAtoms()* (page 181) and *Ensemble.getAtoms()* (page 132) instead.

• *Trajectory* (page 181) reference coordinates must be changed using *Trajectory.setCoords()* (page 182) method.

For usage examples see *Trajectory Analysis*[^12], *Trajectory Analysis II*[^13], *Frames and Atom Groups*[^14], and *Trajectory Output*[^15].

**New Features:**

• *Atom Flags* (page 56), that are used in *Atom Selections* (page 78), is implemented. See its documentation for handy usage examples.

• *sortAtoms()* (page 67) function is implemented.

• *pickCentralConf()* (page 143) function is implemented to pick the conformation or the active coordinate set that is closest to the average of coordinate sets.

• *writePSF()* (page 179), a simple PSF file writer, is implemented.

• *glob()* (page 191) utility function is implemented.

• *iterPDBFilenames()* (page 160) function is implemented, which can be used to iterate over all PDB files stored in a local mirror of Protein Data Bank.

[^12]: http://prody.csb.pitt.edu/tutorials/trajectory_analysis/trajectory.html#trajectory
[^13]: http://prody.csb.pitt.edu/tutorials/trajectory_analysis/trajectory2.html#trajectory2
[^14]: http://prody.csb.pitt.edu/tutorials/trajectory_analysis/frame.html#frame
[^15]: http://prody.csb.pitt.edu/tutorials/trajectory_analysis/outputtraj.html#outputtraj
• **findPDBFiles()** (page 160) function is implemented, which can be used to access PDB files in a path.

**Improvements:**

• **HierView** (page 67) instances are built more efficiently. Two times speed-up is achieved by delaying instantiation of Chain (page 49) and Residue (page 69) instances until they are needed.

• Multiple **Atom Flags** (page 56) can be used in **Atom Selections** (page 78) without using ‘and’ operator, e.g. ‘sidechain carbon’ is the same as ‘sidechain and carbon’.

• **writePDB()** (page 163) accepts Ensemble (page 132), Conformation (page 131), and Frame (page 178) instances as atoms argument.

• **writePDB()** (page 163) function is around 25% faster.

• **pickCentral()** (page 143) is extended to accept Atomic (page 44) and Ensemble (page 132) instances. Old function is now **pickCentralAtom()** (page 143).

• **prody_align** (page 3) command and **prody_align()** (page 198) function can handle non-protein atom selections (see examples for **prody_align** (page 3)).

• **parsePDB()** (page 162) and **writePDB()** (page 163) supports 100K and more atoms.

**Changes:**

• **showOverlapTable()** (page 124) displays first set of modes along x axis of the plot.

• **AtomGroup.setData()** (page 43) does not accept arrays with boolean data type, use **AtomGroup.setFlags()** (page 43) instead.

• **writePDB()** (page 163) function argument **model** is changed to **csets** that indicates the coordinate set index of **atoms** argument.

• **PackageLogger.timing()** (page 188) does not return elapsed time, only logs this information.

• **PackageLogger.startLogfile()** is deprecated for removal in v1.3, use **PackageLogger.start()** (page 188) instead.

• **PackageLogger.closeLogfile()** is deprecated for removal in v1.3, use **PackageLogger.close()** (page 188) instead.

• from prody.utilities import * will not work anymore due to potential name conflicts with Python standard library functions. Import required functions explicitly.

• **writePDB()** (page 163) appends .pdb extension to filename when it is not present

• **prody select** (page 15) command positional argument order is changed to allow for handling multiple PDBs at a time. Old older will be supported until v1.4, but a warning message will be issued.

• **select** argument in **alignCoordsets()** (page 144) is removed, make selection outside of the function instead.

**Deprecations:**

• **AtomGroup.getHeteros()** method has been deprecated for removal in v1.3, use **getFlags('hetatm')** instead.

• **AtomMap.getMappedFlags()** and **AtomMap.getDummyFlags()** methods have been deprecated for removal in v1.3, use **getFlags('mapped')** and **getFlags('dummy')** instead.

• **getVerbosity()** and **setVerbosity()** are deprecated for removal in v1.3, use **confProDy()** (page 204) instead which save changes permanently.
• NMA.getModes() and ModeSet.getModes() methods are deprecated for removal in v1.3, use list()\textsuperscript{16}, e.g. list(model), instead.

Bugfixes:
• Fixed a bug in \textit{prody contacts} (page 8) command that arose problems when when selecting a subset of the target atoms.

Normal Mode Wizard

Improvements:
• \textit{ProDy Interface} shows the size of the trajectory output file for PCA calculations.
• \textit{Mode Graphics Options} allows for copying arrows settings from one mode to another.
• Color scale method and midpoint for protein coloring based on mobility and bfactors can be adjusted from \textit{Protein Graphics Options} panel.

5.5 ProDy 1.1 Series

• 1.1 (June 1, 2012) (page 229)
  – Normal Mode Wizard (page 230)

5.5.1 1.1 (June 1, 2012)

New Features:
• \texttt{iterFragments()} (page 66) function is added.
• \texttt{findNeighbors()} (page 140) function is added.
• \texttt{calcMSF()} (page 141) and \texttt{calcRMSF()} (page 141) functions are added.
• \texttt{wrapAtoms()} (page 145) functions is added.
• \texttt{extendMode()} (page 104) and \texttt{extendVector()} (page 104) functions are added.
• \texttt{prody contacts} (page 8) command is added.

Improvements:
• \texttt{moveAtoms()} (page 144) function is improved to move atoms to a specified location.
• \texttt{DCDFile} (page 175) and \texttt{parseDCD()} (page 177) take \texttt{astype} keyword argument for automatic type recasting for coordinate arrays. This option can be used to convert 32-bit coordinate arrays to 64-bit automatically for higher precision calculations.
• Commands \texttt{prody anm} (page 4), \texttt{prody gnm} (page 11), and \texttt{prody pca} (page 13) can extend a coarse grained model to backbone or all atoms of the residues. See their documentation pages.

Changes:
• Color scale used by \texttt{showOverlapTable()} (page 124) is normalized by default.
• \texttt{tools} module is deprecation for removal, use \texttt{utilities} (page 185) instead.

\textsuperscript{16}http://docs.python.org/library/functions.html#list
• *array* argument in `moveAtoms()` (page 144) is replaced with *by* keyword argument.

• *which* argument in `AtomGroup.copy()` (page 39) method is deprecated for removal in version 1.2.

• *DCDFile* (page 175) does not log information for most common type of DCD file, i.e. 32-bit CHARMM format.

• `Trajectory.getNextIndex()` method is deprecated for removal in v1.2, use `nextIndex()` (page 182) instead.

**Bugfixes:**

• Fixed several problems in `iterNeighbors()` (page 140) function and `Contacts` (page 140) class that were introduced after transition to new `KDTree` (page 137) interface.

• Fixed a problem in setting selection strings of fragments identified using `findFragments()` (page 66).

• Fixed a problem in `calcCenter()` (page 141) related to weighted center calculation.

• Fixed a problem of in copying `AtomMap` (page 45) instances, which would emerge when bond information was present in unusual mappings, such as when atom orders are changed or an atom is present multiple times in the mapping.

**Normal Mode Wizard**

**Improvements:**

• Mode scaling options are improved.

• Options added for extending coarse grained NMA models to residue backbone or all atoms.

### 5.6 ProDy 1.0 Series

<table>
<thead>
<tr>
<th>1.0.4 (May 2, 2012) (page 230)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.3 (May 1, 2012) (page 231)</td>
</tr>
<tr>
<td>1.0.2 (May 1, 2012) (page 231)</td>
</tr>
<tr>
<td>1.0.1 (Apr 6, 2012) (page 232)</td>
</tr>
<tr>
<td>1.0 (Mar 7, 2012) (page 233)</td>
</tr>
</tbody>
</table>

### 5.6.1 1.0.4 (May 2, 2012)

**Bugfixes:**

• Fixed a problem in `calcPhi()` (page 141) function that raised a name error.

• Fixed a problem in `KDTree.getDistances()` (page 138) method that raised a name error when unitcell is provided.

• Fixed a problem in `buildDistMatrix()` (page 140) and `calcDistance()` (page 140) functions causing miscalculations when unitcell is given.

• Revised `KDTree` (page 137) methods dealing with to handle special cases where unitcell might have some dimensions zero.

**Changes:**
• **buildKDTree()** method is removed, earlier than planned due to unexpected bugfix releases.

### 5.6.2 1.0.3 (May 1, 2012)

**Bugfixes:**

• Fixed *kdtree* (page 137) import problem.

**New Features:**

• **buildDistMatrix** (page 140) function that can take periodic boundary conditions is implemented.

**Improvements:**

• **calcDistance** (page 140) function is improved to take periodic boundary conditions into account when provided by the users.

### 5.6.3 1.0.2 (May 1, 2012)

**New Features:**

• Methods to deal with connected subsets of atoms are implemented, see `AtomGroup.iterFragments()` (page 41) and `AtomGroup.numFragments()` (page 42).

• **pickCentral** (page 143) method is implemented for picking the atom that is closest to the centroid of a group or subset of atoms.

• ProDy configuration option `auto_secondary` is implemented to allow for parsing and assigning secondary structure information from PDB file header data automatically. See `assignSecstr()` (page 158) and `confProDy()` (page 204) for usage details.

• **prody align** makes use of --select when aligning multiple structures. See usage examples: `prody align` (page 3)

• **printRMSD** (page 146) function that prints minimum, maximum, and mean RMSD values when comparing multiple coordinate sets is implemented.

• **findFragments** (page 66) function that identifies fragments in atom subsets, e.g. `Selection` (page 86), is implemented.

• A new *KDTree* (page 137) interface with coherent method names and capability to handle periodic boundary conditions is implemented.

**Improvements:**

• Performance improvements made in `saveAtoms()` (page 66) and `loadAtoms()` (page 66).

• **sliceMode** (page 104), **sliceModel** (page 104), **sliceVector** (page 104), and **reduceModel** (page 105) functions accept `Selection` (page 86) instances as well as selection strings. In repeated use of this function, if selections are already made out of the function, considerable speed-ups are achieved when selection is passed instead of selection string.

• Fragment iteration (`AtomGroup.iterFragments()` (page 41)) is improved to yield items faster.

**Changes:**

• There is a change in the behavior of addition operation on instances of `AtomGroup` (page 37). When operands do not have same number of coordinate sets, the result will have one coordinate set that is concatenation of the *active coordinate sets* of operands.

• **buildKDTree()** function is deprecated for removal, use the new *KDTree* (page 137) class instead.
Bugfixes:

- A problem in building hierarchical views when making selections using resindex, chindex, and segindex keywords is fixed.
- A problem in Chain (page 49) and Residue (page 69) selection strings that would emerge when a HierView (page 67) is build using a selection is fixed.
- A problem with copying AtomGroup (page 37) instances whose coordinates are not set is fixed.
- AtomGroup (page 37) fragment detection algorithm is rewritten to avoid the problem of reaching maximum recursion depth for large molecules with the old recursive algorithm.
- A problem with picking central atom of AtomGroup (page 37) instances in pickCentral() (page 143) function is fixed.
- A problem in Select (page 85) class that caused exceptions when evaluating complex macro definitions is fixed.
- Fixed a problem in handling multiple trajectory files. The problem would emerge when a file was added (addFile() (page 181)) to a Trajectory (page 181) after atoms were set (setAtoms() (page 182)). Newly added file would not be associated with the atoms and coordinates parsed from this file would not be set for the AtomGroup (page 37) instance.

5.6.4 1.0.1 (Apr 6, 2012)

New Features:

- ProDy can be configured to automatically check for updates on a regular basis, see checkUpdates() (page 205) and confProDy() (page 204) functions for details.
- alignPDBEnsemble() (page 134) function is implemented to align PDB files using transformations calculated in ensemble analysis. See usage example in Homologous Proteins\(^\text{17}\) example.
- PDBConformation.getTransformation() (page 132) is implemented to return the transformation that was used to superpose conformation onto reference coordinates. This transformation can be used to superpose the original PDB file onto the reference PDB file.
- Amino acid sequences with regular expressions can be used to make atom selections, e.g. sequence "C..C". See Atom Selections (page 78) for usage details.
- calcCrossProjection() (page 100) function is implemented.

Improvements:

- Select (page 85) class raises a SelectionError when potential typos are detected in a selection string, e.g. 'chain AB' is a grammatically correct selection string that will return None since no atoms have chain identifier 'AB'. In such cases, an exception noting that values exceed maximum number of characters is raised.
- prody align command accepts percent sequence identity and overlap parameters used when matching chains from given multiple structures.
- When using prody align command to align multiple structure, all models in NMR structures are aligned onto the reference structure.
- prody catdcd command accepts --align SELSTR argument that can be used to align frames when concatenating files.

\(^\text{17}\)http://prody.csb.pitt.edu/tutorials/ensemble_analysis/blast.html#pca-blast
• `showProjection()` (page 124) and `showCrossProjection()` (page 125) functions are improved to evaluate list of markers, color, labels, and texts. See usage example in Plotting\textsuperscript{18}.

• `Trajectory` (page 181) instances can be used for calculating and plotting projections using `calcProjection()` (page 100), `showProjection()` (page 124), `calcCrossProjection()` (page 100), and `showCrossProjection()` (page 125) functions.

Changes:

• Phosphorylated amino acids, phosphothreonine (TPO), O-phosphotyrosine (PTR), and phosphoserine (SEP), are recognized as acidic protein residues. This prevents having breaks in protein chains which contains phosphorylated residues. See Atom Selections (page 78) for definitions of protein and acidic keywords.

• Hit dictionaries from `PDBBlastRecord` (page 148) will use `percent_overlap` instead of `percent_coverage`. Older key will be removed in v1.1.

• Transformation.get4x4Matrix() method is deprecated for removal in v1.1, use Transformation.getMatrix() (page 143) method instead.

Bugfixes:

• A bug in some ProDy Applications (page 3) is fixed. The bug would emerge when invalid arguments were passed to effected commands and throw an unrelated exception hiding the error message related to the arguments.

• A bug in ‘bonded to . . .’ is fixed that emerged when ‘. . .’ selected nothing.

• A bug in ‘not’ selections using . operator is fixed.

5.6.5 1.0 (Mar 7, 2012)

Improvements:

• `ANM.buildHessian()` (page 101) method is not using a KDTree by default, since with some code optimization the version not using KDTree is running faster. Same optimization has gone into `GNM.buildKirchhoff()` (page 112) too, but for Kirchhoff matrix, version using KDTree is faster and is the default. Both methods have `ktree` argument to choose whether to use it or not.

• `prody` script is updated. Importing Prody and Numpy libraries are avoided. Script responses to help queries faster. See ProDy Applications (page 3) for script usage details.

• Added bonded to . . . selection method that expands a selection to immediately bound atoms. See Atom Selections (page 78) for its description.

• `fetchPDBLigand()` (page 164) parses bond data from the XML file.

• `fetchPDBLigand()` (page 164) can optionally save compressed XML files into ProDy package folder so that frequent access to same files will be more rapid. See `confProDy()` (page 204) function for setting this option.

• `Select` (page 85) class is revised. All exceptions are handled delicately to increase the stability of the class.

• Distance based atom selection is 10 to 15% faster for atom groups with more than 5K atoms.

• Added uncompressed file saving option to `prody blast` (page 6) command.

Changes:

• All deprecated method and functions scheduled for removal are removed.

\textsuperscript{18}http://prody.csb.pitt.edu/tutorials/ensemble_analysis/xray_plotting.html#pca-xray-plotting
getEigenvector() and getEigenvalue() methods are deprecated for removal in v1.1, use Mode.getEigvec() (page 115) and Mode.getEigval() (page 115) instead.

getEigenvectors() and getEigenvalues() methods are deprecated for removal in v1.1, use NMA.getEigvecs() (page 117) and NMA.getEigvals() (page 117) instead.

Mode.getCovariance() and ModeSet.getCovariance() methods are deprecated for removal in v1.1, use calcCovariance() (page 99) method instead.

Mode.getCollectivity() method is removed, use calcCollectivity() (page 99) function instead.

Mode.getFractOfVariance() method is removed, use the new calcFractVariance() (page 100) function instead.

Mode.getSqFlucts() method is removed, use calcSqFlucts() (page 100) function instead.

Renamed showFractOfVar() function as showFractVars() (page 124) function instead.

Removed calcCumOverlapArray(), use calcCumulOverlap() (page 103) with array=True argument instead.

Renamed extrapolateModel() as extendModel() (page 104).

The relation between AtomGroup (page 37), Trajectory (page 181), and Frame (page 178) instances have changed. See Trajectory Analysis II\(^\text{19}\) and Trajectory Output\(^\text{20}\), and Frames and Atom Groups\(^\text{21}\) usage examples.

AtomGroup (page 37) cannot be deformed by direct addition with a vector instance.

Unmapped atoms in AtomMap (page 45) instances are called dummies. AtomMap.numUnmapped() method, for example, is renamed as AtomMap.numDummies() (page 48).

fetchPDBLigand() (page 164) accepts only filename (instead of save and folder) argument to save an XML file.

Bugfixes:

- A problem in distance based atom selection which would could cause problems when a distance based selection is made from a selection is fixed.
- Changed prody blast (page 6) so that when a path for downloading files are given files are not save to local PDB folder.

### 5.7 ProDy 0.9 Series

- 0.9.4 (Feb 4, 2012) (page 235)
- 0.9.3 (Feb 1, 2012) (page 235)
- 0.9.2 (Jan 11, 2012) (page 236)
- 0.9.1 (Nov 9, 2011) (page 237)
- 0.9 (Nov 8, 2011) (page 237)
  - Normal Mode Wizard (page 241)

\(^{19}\)http://prody.csb.pitt.edu/tutorials/trajectory_analysis/trajectory2.html#trajectory2
\(^{20}\)http://prody.csb.pitt.edu/tutorials/trajectory_analysis/outputtraj.html#outputtraj
\(^{21}\)http://prody.csb.pitt.edu/tutorials/trajectory_analysis/frame.html#frame
5.7.1 0.9.4 (Feb 4, 2012)

Changes:

• `setAtomGroup()` and `getAtomGroup()` methods are renamed as `Ensemble.setAtoms()` (page 133) and `Ensemble.getAtoms()` (page 132).

• `AtomGroup` (page 37) class trajectory methods, i.e. `AtomGroup.setTrajectory()`, `AtomGroup.getTrajectory()`, `AtomGroup.nextFrame()`, `AtomGroup.nextFrame()`, and `AtomGroup.gotoFrame()` methods are deprecated. Version 1.0 will feature a better integration of `AtomGroup` (page 37) and `Trajectory` (page 181) classes.

Bugfixes:

• Bugfixes in `Bond.setACSIndex()` (page 49), `saveAtoms()` (page 66), and `HierView.getSegment()` (page 67).

• Bugfixes in `GammaVariableCutoff` (page 109) and `GammaStructureBased` (page 108) classes.

• Bugfix in `calcCrossCorr()` (page 99) function.

• Bugfixes in `Ensemble.getWeights()` (page 133), `showOccupancies()` (page 134), `DCDFile.flush()` (page 175).

• Bugfixes in ProDy commands `prody blast` (page 6), `prody fetch` (page 11), and `prody pca` (page 13).

• Bugfix in `calcCenter()` (page 141) function.

5.7.2 0.9.3 (Feb 1, 2012)

New Features:

• `DBRef` (page 156) class is implemented for storing references to sequence databases parsed from PDB header records.

• Methods for storing coordinate set labels in `AtomGroup` (page 37) instances are implemented: `getACSLabel()` (page 39), and `getACSLabel()` (page 39).

• `calcCenter()` (page 141) and `moveAtoms()` (page 144) functions are implemented for dealing with coordinate translation.

• Hierarchical view, `HierView` (page 67), is completely redesigned. PDB files that contain non-empty segment name column (or when such information is parsed from a PSF file), new design delicately handles this information to identify distinct chains and residues. This prevents merging distinct chains in different segments but with same identifiers and residues in those with same numbers. New design is also using ordered dictionaries `collections.OrderedDict` and lists so that chain and residue iterations yield them in the order they are parsed from file. These improvements also bring modest improvements in speed.

• `Segment` (page 74) class is implemented for handling segments of atoms defined in molecular dynamics simulations setup, using `psfgen` for example.

• Context manager methods are added to trajectory classes. A trajectory file can be opened as follows:

```python
with Trajectory('mdm2.dcd') as traj:
    for frame in traj:
        calcGyradius(frame)
```

• `Chain` (page 49) slicing is implemented:

---

22http://docs.python.org/library/collections.html#collections.OrderedDict
p38 = parsePDB('1p38')
chA = p38['A']
res_4to10 = chA[4:11]
res_100toLAST = chA[100:]

• Some support for bonds is implemented to AtomGroup (page 37) class. Bonds can be set using setBonds() (page 42) method. All bonds must be set at once. iterBonds() (page 41) or iterBonds() (page 36) methods can be used to iterate over bonds in an AtomGroup or an Atom.

• parsePSF() (page 178) parses bond information and sets to the atom group.

• Selection.update() (page 90) method is implemented, which may be useful to update a distance based selection after coordinate changes.

• buildKDTTree() and iterNeighbors() (page 140) methods are implemented for facilitating identification of pairs of atoms that are proximal.

• iterAtoms() (page 41) method is implemented to all atomic (page 31) classes to provide uniformity for atom iterations.

• calcAngle() (page 141), calcDihedral() (page 141), calcPhi() (page 141), calcPsi() (page 141), and calcOmega() (page 141) methods are implemented.

Improvements:

• Chain.getSelstr() (page 52) and Residue.getSelstr() (page 72) methods are improved to include the selection string of a Selection (page 86) when they are built using one.

Changes:

• Residue (page 69) methods getNumber(), setNumber(), getName(), setName() methods are deprecated and will be removed in v1.0.

• Chain (page 49) methods getIdentifier() and setIdentifier() methods are deprecated and will be removed in v1.0.

• Polymer (page 154) attribute identifier is renamed as chid (page 156).

• Chemical (page 153) attribute identifier is renamed as resname (page 154).

• getACSI() and setACSI() are renamed as getACSIndex() (page 39) and setACSIndex() (page 42), respectively.

• calcRadiusOfGyration() is deprecated and will be removed in v1.0. Use calcGyradius() (page 141) instead.

Bugfixes:

• Fixed a problem in parsePDB() (page 162) that caused loosing existing coordinate sets in an AtomGroup (page 37) when passed as ag argument.

• Fixed a problem with "same ... as ..." argument of Select (page 85) that selected atoms when followed by an incorrect atom selection.

• Fixed another problem with "same ... as ..." which result in selecting multiple chains when same chain identifier is found in multiple segments or multiple residues when same residue number is found in multiple segments.

• Improved handling of negative integers in indexing AtomGroup (page 37) instances.

5.7.3 0.9.2 (Jan 11, 2012)

New Features:
• `prody catdcd` command is implemented for concatenating and/or slicing `.dcd` files. See `prody catdcd` (page 7) for usage examples.
• `DCDFile` (page 175) can be opened in write or append mode, and coordinate sets can be added using `write()` (page 177) method.
• `getReservedWords()` can be used to get a list of words that cannot be used to label user data.
• `confProDy()` (page 204) function is added for configuring ProDy.
• ProDy can optionally backup existing files with `.BAK` (or another) extension instead of overwriting them. This behavior can be activated using `confProDy()` (page 204) function.

**Improvements:**
• `writeDCD()` (page 177) file accepts `AtomGroup` (page 37) or other `Atomic` (page 44) instances as `trajectory` argument.
• `prody align` command can be used to align multiple PDB structures.
• `prody pca` command allows atom selections for DCD files that are accompanied with a PDB or PSF file.

**Changes:**
• `DCDFile` (page 175) instances, when closed, raise exception, similar to behavior of `file` objects in Python.
• Title of `AtomGroup` (page 37) instances resulting from copying an `Atomic` (page 44) instances does not start with ‘Copy of’.
• `changeVerbosity()` and `getVerbosityLevel()` are renamed as `setVerbosity()` and `getVerbosity()`, respectively. Old names will be removed in v1.0.
• ProDy applications (commands) module is rewritten to use new `argparse` module. See `ProDy Applications` (page 3) for details of changes.
• `argparse` module is added to the package for Python versions 2.6 and older.

**Bugfixes:**
• Fixed problems in `loadAtoms()` (page 66) and `saveAtoms()` (page 66) functions.
• Bugfixes in `parseDCD()` (page 177) and `writeDCD()` (page 177) functions for Windows compatibility.

### 5.7.4 0.9.1 (Nov 9, 2011)

**Bug Fixes:**
• Fixed problems with reading and writing configuration files.
• Fixed problem with importing nose for testing.

### 5.7.5 0.9 (Nov 8, 2011)

**New Features:**

---

23 [http://docs.python.org/library/argparse.html#argparse](http://docs.python.org/library/argparse.html#argparse)

24 [http://docs.python.org/library/argparse.html#argparse](http://docs.python.org/library/argparse.html#argparse)
• PDBML\textsuperscript{25} and mmCIF\textsuperscript{26} files can be retrieved using \texttt{fetchPDB()} (page 160) function.

• \texttt{getPDBLocalFolder()} and \texttt{setPDBLocalFolder()} functions are implemented for local PDB folder management.

• \texttt{parsePDBHeader()} (page 157) is implemented for convenient parsing of header data from .pdb files.

• \texttt{showProtein()} (page 153) is implemented to allow taking a quick look at protein structure.

• Chemical (page 153) and Polymer (page 154) classes are implemented for storing chemical and polymer component data parsed from PDB header records.

Changes:

| Warning: | This release introduces numerous changes in method and function names all aiming to improve the interactive usage experience. All changes are listed below. Currently these functions and methods are present in both old and new names, so code using ProDy must not be affected. Old function names will be removed from version 1.0, which is expected to happen late in the first quarter of 2012. Old function names are marked as deprecated, but ProDy will not issue any warnings until the end of 2011. In 2012, ProDy will automatically start issuing \texttt{DeprecationWarning} upon calls using old names to remind the user of the name change. For deprecated methods that are present in multiple classes, only the affected modules are listed for brevity. |

| Note: | When modifying code using ProDy to adjust the name changes, turning on deprecation warnings may help locating all use cases of the deprecated names. See \texttt{turnonDepracationWarnings()} for this purpose. |

Functions:

The following function name changes are mainly to reduce the length of the name in order to make them more suitable for interactive sessions:

\textsuperscript{25}http://pdbml.pdb.org/
\textsuperscript{26}http://mmcif.pdb.org/
<table>
<thead>
<tr>
<th>Old name</th>
<th>New name</th>
<th>Affected modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>applyBiomolecularTransformations()</td>
<td>buildBiomolecules()</td>
<td>atomic (page 31), ensemble (page 130), dynamics (page 96)</td>
</tr>
<tr>
<td>assignSecondaryStructure()</td>
<td>assignSecstr()</td>
<td>atomic (page 31)</td>
</tr>
<tr>
<td>scanPerturbationResponse()</td>
<td>calcPerturbResponse()</td>
<td>atomic (page 31)</td>
</tr>
<tr>
<td>calcCrossCorrelations()</td>
<td>calcCrossCorr()</td>
<td>atomic (page 31)</td>
</tr>
<tr>
<td>calcCumulativeOverlap()</td>
<td>calcCumulOverlap()</td>
<td>atomic (page 31)</td>
</tr>
<tr>
<td>showFractOfVariances()</td>
<td>showFractVars()</td>
<td>atomic (page 31)</td>
</tr>
<tr>
<td>showCumFractOfVariances()</td>
<td>showCumulFractVars()</td>
<td>atomic (page 31)</td>
</tr>
<tr>
<td>showCrossCorrelations()</td>
<td>showCrossCorr()</td>
<td>atomic (page 31)</td>
</tr>
<tr>
<td>showCumulativeOverlap()</td>
<td>showCumulOverlap()</td>
<td>ensemble (page 130)</td>
</tr>
<tr>
<td>deform()</td>
<td>deformAtoms()</td>
<td>ensemble (page 130)</td>
</tr>
<tr>
<td>calcSumOfWeights()</td>
<td>calcOccupancies()</td>
<td>dynamics (page 96)</td>
</tr>
<tr>
<td>trimEnsemble()</td>
<td>trimPDBEnsemble()</td>
<td>ensemble (page 130)</td>
</tr>
<tr>
<td>getKeywordResidueNames()</td>
<td>getKeywordResnames()</td>
<td></td>
</tr>
<tr>
<td>setKeywordResidueNames()</td>
<td>setKeywordResnames()</td>
<td></td>
</tr>
<tr>
<td>getPairwiseAlignmentMethod()</td>
<td>getAlignmentMethod()</td>
<td></td>
</tr>
<tr>
<td>getPairwiseMatchScore()</td>
<td>getMatchScore()</td>
<td></td>
</tr>
<tr>
<td>getPairwiseMismatchScore()</td>
<td>getMismatchScore()</td>
<td></td>
</tr>
<tr>
<td>getPairwiseGapOpeningPenalty()</td>
<td>getGapPenalty()</td>
<td></td>
</tr>
<tr>
<td>getPairwiseGapExtensionPenalty()</td>
<td>getGapExtPenalty()</td>
<td></td>
</tr>
<tr>
<td>getName method:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Coordinate methods:

All `getCoordinates()` and `setCoordinates()` methods in atomic (page 31) and ensemble (page 130) classes are renamed as `getCoords()` and `setCoords()`, respectively.

### getNumOf methods:

All method names starting with `getNumOf` now start with `num`. This change brings two advantages: method names (i) are considerably shorter, and (ii) do not suggest that there might also be corresponding `set` methods.

<table>
<thead>
<tr>
<th>Old name</th>
<th>New name</th>
<th>Affected modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>getNumOfAtoms()</td>
<td>numAtoms()</td>
<td>atomic (page 31), ensemble (page 130), dynamics (page 96)</td>
</tr>
<tr>
<td>getNumOfChains()</td>
<td>numChains()</td>
<td>atomic (page 31)</td>
</tr>
<tr>
<td>getNumOfConfs()</td>
<td>numConfs()</td>
<td>ensemble (page 130)</td>
</tr>
<tr>
<td>getNumOfCoordsets()</td>
<td>numCoordsets()</td>
<td>atomic (page 31), ensemble (page 130)</td>
</tr>
<tr>
<td>getNumOfDegOfFreedom()</td>
<td>numDOF()</td>
<td>dynamics (page 96)</td>
</tr>
<tr>
<td>getNumOfFixed()</td>
<td>numFixed()</td>
<td>ensemble (page 130)</td>
</tr>
<tr>
<td>getNumOfFrames()</td>
<td>numFrames()</td>
<td>ensemble (page 130)</td>
</tr>
<tr>
<td>getNumOfResidues()</td>
<td>numResidues()</td>
<td>atomic (page 31)</td>
</tr>
<tr>
<td>getNumOfMapped()</td>
<td>numMapped()</td>
<td>atomic (page 31)</td>
</tr>
<tr>
<td>getNumOfModes()</td>
<td>numModes()</td>
<td>dynamics (page 96)</td>
</tr>
<tr>
<td>getNumOfSelected()</td>
<td>numSelected()</td>
<td>ensemble (page 130)</td>
</tr>
<tr>
<td>getNumOfUnmapped()</td>
<td>numUnmapped()</td>
<td>atomic (page 31)</td>
</tr>
</tbody>
</table>
getName() methods are renamed as getTitle() to avoid confusions that might arise from changes in atomic (page 31) method names listed below. All classes in atomic (page 31), ensemble (page 130), and dynamics (page 96) are affected from this change.

In line with this change, parsePDB() (page 162) and parsePQR() (page 163) name arguments are changed to title, but name argument will also work until release 1.0.

This name change conflicted with DCDFile.getTitle() (page 175) method. The conflict is resolved in favor of the general getTitle() method. An alternative method will be implemented to handle title strings in DCD files.

get/set methods of atomic classes:

Names of get and set methods allowing access to atomic data are all shortened as follows:

<table>
<thead>
<tr>
<th>Old name</th>
<th>New name</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAtomNames()</td>
<td>getNames()</td>
</tr>
<tr>
<td>getAtomTypes()</td>
<td>getTypes()</td>
</tr>
<tr>
<td>getAltLocIndicators()</td>
<td>getAltlocs()</td>
</tr>
<tr>
<td>getAnisoTempFactors()</td>
<td>getAnisos()</td>
</tr>
<tr>
<td>getAnisoStdDevs()</td>
<td>getAnistds()</td>
</tr>
<tr>
<td>getChainIdentifiers()</td>
<td>getChains()</td>
</tr>
<tr>
<td>getElementSymbols()</td>
<td>getElements()</td>
</tr>
<tr>
<td>getHeteroFlags()</td>
<td>getHeteros()</td>
</tr>
<tr>
<td>getInsertionCodes()</td>
<td>getIcodes()</td>
</tr>
<tr>
<td>getResidueNames()</td>
<td>getResnames()</td>
</tr>
<tr>
<td>getResidueNumbers()</td>
<td>getResnums()</td>
</tr>
<tr>
<td>get SecondaryStrs()</td>
<td>getSecstrs()</td>
</tr>
<tr>
<td>getSegmentNames()</td>
<td>getSegnames()</td>
</tr>
<tr>
<td>getSerialNumbers()</td>
<td>getSerials()</td>
</tr>
<tr>
<td>getTempFactors()</td>
<td>getBetas()</td>
</tr>
</tbody>
</table>

This change affects all atomic (page 31) classes, AtomGroup (page 37), Atom (page 33), Chain (page 49), Residue (page 69), Selection (page 86) and AtomMap (page 45).

Other changes in atomic methods:

- getSelectionString() renamed as getSelstr()

Methods handling user data (which was previously called attribute) are renamed as follows:

<table>
<thead>
<tr>
<th>Old name</th>
<th>New name</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAttribute()</td>
<td>getData()</td>
</tr>
<tr>
<td>getAttrNames()</td>
<td>getDataLabels()</td>
</tr>
<tr>
<td>getAttrType()</td>
<td>getDataType()</td>
</tr>
<tr>
<td>delAttribute()</td>
<td>delData()</td>
</tr>
<tr>
<td>isAttribute()</td>
<td>isData()</td>
</tr>
<tr>
<td>setAttribute()</td>
<td>setData()</td>
</tr>
</tbody>
</table>

To be removed:

Finally, the following methods will be removed, but other suitable methods are overloaded to perform their action:

- removed AtomGroup.getBySerialRange(), overloaded AtomGroup.getBySerial() (page 39)
- removed getProteinResidueNames(), overloaded getKeywordResnames()
- removed setProteinResidueNames(), overloaded setKeywordResnames()
Scripts:
The way ProDy scripts work has changed. See ProDy Applications (page 3) for details. Using older scripts will start issuing deprecation warnings in 2012.

Bug Fixes:
- Bugs in execDSSP () (page 152) and execSTRIDE() (page 164) functions that caused exceptions when compressed files were passed is fixed.
- A problem in scripts for PCA of DCD files is fixed.

Normal Mode Wizard
Development of NMWiz is finalized and it will not be distributed in the ProDy installation package anymore. See Normal Mode Wizard\textsuperscript{27} pages for instructions on installing it.

5.8 ProDy 0.8 Series

- 0.8.3 (Oct 16, 2011) (page 241)
- 0.8.2 (Oct 14, 2011) (page 242)
- 0.8.1 (Sep 16, 2011) (page 242)
  - Normal Mode Wizard (page 243)
- 0.8 (Aug 24, 2011) (page 243)
  - Normal Mode Wizard\textsuperscript{a} (page 245)

\textsuperscript{a}http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz

5.8.1 0.8.3 (Oct 16, 2011)

New Features:
- Functions to read and write PQR files: \texttt{parsePQR()} (page 163) and \texttt{writePQR()} (page 153).
- Added \texttt{PDBEnsemble.getIdentifiers()} method that returns identifiers of all conformations in the ensemble.
- ProDy tests are incorporated to the package installer. If you are using Python version 2.7, you can run the tests by calling \texttt{prody.test()}.

Improvements:
- \texttt{blastPDB()} (page 149) function and \texttt{PDBBlastRecord} (page 148) class are rewritten to use faster and more compact code.
- New \texttt{PackageLogger} (page 187) function is implemented to unify logging and reporting task progression.
- Improvements in PDB ensemble support functions, e.g. \texttt{trimPDBEnsemble()} (page 134), are made.
- Improvements in ensemble concatenations are made.

Bug Fixes:

\textsuperscript{27}http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz
• Bugfixes in `PDBEnsemble()` slicing operation. This may have affected users when slicing a PDB ensemble for plotting projections in color for different forms of the protein.

5.8.2 0.8.2 (Oct 14, 2011)

New Features:

• `fetchPDBClusters()` (page 161), `loadPDBClusters()` (page 161), and `getPDBCluster()` functions are implemented for handling PDB sequence cluster data. These functions can be used instead of `blastPDB()` (page 149) function for fast access to structures of the same protein (at 95% sequence identity level) or similar proteins.

• Perturbation response scanning method described in [CA09] (page 260) is implemented as `scanPerturbationResponse()` based on the code provided by Ying Liu.

Changes:

• `fetchPDBLigand()` (page 164) returns the URL of the XML file in the ligand data dictionary.

• Name of the ProDy configuration file in user home directory is renamed as `.prodyrc` (used to be `.prody`).

• `applyBiomolecularTransformations()` and `assignSecondaryStructure()` functions raise `ValueError` when the function fails to perform its action due to missing data in header dictionary.

• `fetchPDB()` (page 160) decompresses PDB files found in the working directory when user asks for decompressed files.

• `parsePDB()` (page 162) appends `chain` and `subset` arguments to `AtomGroup()` name.

• `chain` argument is added to `PDBBlastRecord.getHits()` (page 149).

Improvements:

• Atom selection class `Select` (page 85) is completely redesigned to prevent breaking of the parser when evaluating invalid selection strings.

• Improved type checking in `parsePDB()` (page 162) function.

Bug Fixes:

• Bugfixes in `parseDSSP()` (page 152): one emerged problems in lines indicating chain breaks, another did not parse bridge-partners correctly. Both fixes are contributed by Kian Ho.

• Bugfix in `parsePDB()` (page 162) function. When only header is desired (`header=True, model=0`), would return a tuple containing an empty atom group and the header.

Developmental:

• Unit tests for `proteins` (page 146) and `select` modules are developed.

5.8.3 0.8.1 (Sep 16, 2011)

New Features:

• `fetchLigandData()` is implemented for fetching ligand data from Ligand Expo.

• `parsePSF()` (page 178) function is implemented for parsing X-PLOR format PSF files.

Changes:
• __slots__ is used in AtomGroup (page 37) and Atomic (page 44) classes. This change prevents user from assigning new variables to instances of all classes derived from the base Atomic (page 44).

• pyparsing is updated to version 1.5.6.

Bug Fixes:

• A bug in AtomGroup.copy() (page 39) method is fixed. When AtomGroup instance itself is copied, deep copies of data arrays were not made.

• A bug in Select (page 85) class raising exceptions when negative residue number values are present is fixed.

• Another bug in Select (page 85) class misinterpreting same residue as ... statement when specific chains are involved is fixed.

• A bug in AtomGroup.addCoordset() (page 39) method duplicating coordinates when no coordinate sets are present in the instance is fixed.

Normal Mode Wizard

Changes:

• Version number in main window is iterated.

• Mode graphics material is stored for individual modes.

• Mode scaling factor is printed when active mode or RMSD is changed.

• All selections are deleted to avoid memory leaks.

5.8.4 0.8 (Aug 24, 2011)

Note: After installing v0.8, you may need to make a small change in your existing scripts. If you are using Ensemble (page 132) class for analyzing PDB structures, rename it as PDBEnsemble (page 135). See the other changes that may affect your work below and the class documentation for more information.

New Features:

• DCDFile (page 175) is implemented for handling DCD files.

• Trajectory (page 181) is implemented for handling multiple trajectory files.

• writeDCD() (page 177) is implemented for writing DCD files.

• Trajectory Analysis example to illustrate usage of new classes for handling DCD files. Essential Dynamics Analysis example is updated to use new ProDy classes.

• PCA (page 120) supports Trajectory (page 181) and DCDFile (page 175) instances.

• Ensemble (page 132) and PDBEnsemble (page 135) classes can be associated with AtomGroup (page 37) instances. This allows selecting and evaluating coordinates of subset of atoms. See setAtomGroup(), select(), getAtomGroup(), and getSelection() methods.

• execDSSP() (page 152), parseDSSP() (page 152), and performDSSP() (page 153) functions are implemented for executing and parsing DSSP calculations.

28http://prody.csb.pitt.edu/tutorials/trajectory_analysis/trajectory.html#trajectory
29http://prody.csb.pitt.edu/tutorials/trajectory_analysis/eda.html#eda
• execSTRIDE() (page 164), parseSTRIDE() (page 165), and performSTRIDE() (page 165) functions are implemented for executing and parsing DSSP calculations.

• parsePDB() (page 162) function parses atom serial numbers. Atoms can be retrieved from an AtomGroup (page 37) instance by their serial numbers using getBySerial() (page 39) and getBySerialRange() methods.

• calcADPs() (page 143) function can be used to calculate anisotropic displacement parameters for atoms with anisotropic temperature factor data.

• getRMSFs() (page 133) is implemented for calculating root mean square fluctuations.

• AtomGroup (page 37) and Mode (page 115) or Vector (page 115) additions are supported. This adds a new coordinate set to the AtomGroup (page 37) instance.

• getAttrNames() is implemented for listing user set attribute names.

Improvements:

• calcProjection() (page 100), showProjection() (page 124), and showCrossProjection() (page 125) functions can optionally calculate/display RMSD along the normal mode.

• ANM, GNM, and PCA applications can optionally write compressed ProDy data files.

• fetchPDB() (page 160) function can optionally write decompressed files and force copying a file from local mirror to target folder.

• PCA.buildCovariance() (page 120) and PCA.performSVD() (page 121) methods accept Numpy arrays as coordinate sets.

• Performance of PCA.buildCovariance() (page 120) method is optimized for evaluation of PDB ensembles.

• calcRMSD() (page 144) and superpose() (page 144) functions are optimized for speed and memory usage.

• Ensemble.getMSFs() (page 133) is optimized for speed and memory usage.

• Improvements in memory operations in atomic (page 31), ensemble (page 130), and dynamics (page 96) modules for faster data (PDB/NMD) output.

• Optimizations in Select (page 85) and Contacts (page 140) classes.

Changes:

• Ensemble (page 132) does not store conformation names. Instead, newly implemented PDBEnsemble (page 135) class stores identifiers for individual conformations (PDB IDs). This class should be used in cases where source of individual conformations is important.

• calcProjection() (page 100), showProjection() (page 124), and showCrossProjection() (page 125) function calculate/display root mean square deviations, by default.

• Oxidized cysteine residue abbreviation CSO is added to the definition of protein keyword.

• getMSF() method is renamed as getMSFs() (page 133).

• parseDCD() (page 177) function returns Ensemble (page 132) instances.

Bug Fixes:

• A bug in select module causing exceptions when regular expressions are used is fixed.

• Another bug in select module raising exception when“(not ..)” is passed is fixed.

• Various bugfixes in ensemble (page 130) module.

• Problem in prody fetch that occurred when a file is found in a local mirror is fixed.
• Bugfix in `AtomPointer.copy()` (page 68) method.

**Normal Mode Wizard**

**New Features:**
- NMWiz can be used to compare two structures by calculating and depicting structural changes.
- Arrow graphics is scaled based on a user specified RMSD value.

**Improvements:**
- NMWiz writes DCD format trajectories for PCA using ProDy. This provides significant speed up in cases where IO rate is the bottleneck.

**Changes:**
- Help is provided in a text window to provide a cleaner GUI.

## 5.9 ProDy 0.7 Series

- 0.7.2 (Jun 21, 2011) (page 245)
- 0.7.1 (Apr 28, 2011) (page 245)
- 0.7 (Apr 4, 2011) (page 246)
  - Normal Mode Wizard (page 247)

### 5.9.1 0.7.2 (Jun 21, 2011)

**New Features:**
- `parseDCD()` (page 177) is implemented for parsing coordinate sets from DCD files.

**Improvements:**
- `parsePDB()` (page 162) parses SEQRES records in header sections.

**Changes:**
- Major classes can be instantiated without passing a name argument.
- Default selection in NMWiz ProDy interface is changed to ensure selection only protein Cα atoms.

**Bug Fixes:**
- A bug in `writeNMD()` (page 119) function causing problems when writing a single mode is fixed.
- Other bugfixes in `dynamics` (page 96) module functions.

### 5.9.2 0.7.1 (Apr 28, 2011)

**Highlights:**
- Atomic (page 44) `__getattribute__()` is overloaded to interpret atomic selections following the dot operator. For example, `atoms.calpha` is interpreted as `atoms.select('calpha')`. See :ref:`` for more details.
• **AtomGroup** (page 37) class is integrated with **HierView** (page 67) class. Atom group instances now can be indexed to get chains or residues and number of chains/residues can be retrieved. A hierarchical view is generated and updated when needed. See :ref:`` for more details.

**New Features:**

• **matchAlign()** (page 150) is implemented for quick alignment of protein structures. See Ligand Extraction\[30\] usage example.

• **setAttribute(), getAttribute(), delAttribute(), and isAttribute() functions are implemented for AtomGroup** (page 37) class to facilitate storing user provided atomic data. See Storing data in AtomGroup\[31\] example.

• **saveAtoms()** (page 66) and **loadAtoms()** (page 66) functions are implemented to allow for saving atomic data and loading it. This saves custom atomic attributes and much faster than parsing data from PDB files.

• **calcCollectivity()** (page 99) function is implemented to allow for calculating collectivity of deformation vectors.

**Improvements:**

• **parsePDB()** (page 162) can optionally return biomolecule when biomol=True keyword argument is passed.

• **parsePDB()** (page 162) can optionally make secondary structure assignments when secondary=True keyword argument is passed.

• **calcSqFlucts()** (page 100) function is changed to accept Vector (page 115) instances, e.g. deformation vectors.

**Changes:**

• Changes were made in **calcADPAxes()** (page 142) function to follow the conventions in analysis ADPs. See its documentation.

**Bug Fixes:**

• A in **Ensemble** (page 132) slicing operations is fixed. Weights are now copied to the new instances obtained by slicing.

• Bug fixes in **dynamics** (page 96) plotting functions **showScaledSqFlucts()** (page 126), **showNormedSqFlucts()** (page 126),

5.9.3 0.7 (Apr 4, 2011)

**New Features:**

• Regular expressions can be used in atom selections. See select module for details.

• User can define selection macros using **defSelectionMacro()** function. Macros are saved in ProDy configuration and loaded in later sessions. See select module for other related functions.

• **parseSparseMatrix()** (page 106) function is implemented for parsing matrices in sparse format. See the usage example in Using an External Matrix\[32\].

• **deform()** function is implemented for deforming coordinate sets along a normal mode or linear combination of multiple modes.

---

\[30\]http://prody.csb.pitt.edu/tutorials/structure_analysis/ligands.html#extract-ligands
\[31\]http://prody.csb.pitt.edu/tutorials/prody_tutorial/atomgroup.html#id1
\[32\]http://prody.csb.pitt.edu/tutorials/enm_analysis/external.html#external-matrix
• **sliceModel()** (page 104) function is implemented for slicing normal mode data to be used with functions calculating atomic properties using normal modes.

**Improvements:**

• Atom selections using bare keyword arguments is optimized. New keyword definitions are added. See `select` module for the complete list.

• A new keyword argument for `calcADFaxes()` (page 142) allows for comparing largest axis to the second largest one.

**Changes:**

• There are changes in function used to alter definitions of selection keywords. See `select` for details.

• `assignSecondaryStructure()` function assigns SS identifiers to all atoms in a residue. Residues with no SS information specified is assigned coil conformation.

• When `Ensemble` (page 132) and `NMA` (page 117) classes are instantiated with an empty string, instances are called “Unnamed”.

• `sliceMode()` (page 104), `sliceVector()` (page 104) and `reduceModel()` (page 105) functions return the atom selection in addition to the sliced vector/mode/model instance.

**Bug Fixes:**

• Default selection for `calcGNM()` (page 114) function is set to “calpha”.

**Normal Mode Wizard**

**New Features:**

• NMWiz supports GNM data and can use ProDy for GNM calculations.

• NMWiz can gather normal mode data from molecules loaded into VMD. This allows NMWiz to support all formats supported by VMD.

• User can write data loaded into NMWiz in NMD format.

• An Arrow Graphics option allows the user to draw arrows in both directions.

• User can select Licorice representation for the protein if model is an all atom mode.

• User can select Custom as the representation of the protein to prevent NMWiz from chancing a user set representation.

• Trace is added as a protein backbone representation option.

**Improvements:**

• NMWiz remembers all adjustments on arrow graphics for all modes.

• Plotting Clear button clears only atom labels that are associated with the dataset.

• Removing a dataset removes all associated molecule objects.

• Selected atom representations are turned on based on atom index.

• Padding around interface button has been standardized to provide a uniform experience between different platforms.
5.10 ProDy 0.6 Series

- 0.6.2 (Mar 16, 2011) (page 248)
- 0.6.1 (Mar 2, 2011) (page 248)
- 0.6 (Feb 22, 2011) (page 249)
  - Normal Mode Wizard (page 250)

5.10.1 0.6.2 (Mar 16, 2011)

New Features:

- `performSVD()` (page 121) function is implemented for faster and more memory efficient principal component analysis.
- `extrapolateModel()` function is implemented for extrapolating a coarse-grained model to an all atom model. See the usage example [Extend a coarse-grained model](http://prody.csb.pitt.edu/tutorials/enm_analysis/extend.html#extendmodel).
- `plog()` is implemented for enabling users to make log entries.

Improvements:

- `compare` functions are improved to handle insertion codes.
- `HierView` (page 67) allows for indexing using chain identifier and residue numbers. See usage example [Hierarchical Views](http://prody.csb.pitt.edu/tutorials/prody_tutorial/hierview.html#hierview).
- `Chain` (page 49) allows for indexing using residue number and insertion code. See usage example [Hierarchical Views](http://prody.csb.pitt.edu/tutorials/prody_tutorial/hierview.html#hierview).
- `addCoordset()` (page 39) function accepts `Atomic` (page 44) and `Ensemble` (page 132) instances as `coords` argument.
- New method `HierView.getAtoms()` (page 67) is implemented.
- `AtomGroup` (page 37) set functions check the correctness of dimension of data arrays to prevent run-time problems.
- `prody pca` script is updated to use the faster PCA method that uses SVD.

Changes:

- “backbone” definition now includes the backbone hydrogen atom (Thanks to Nahren Mascarenhas for pointing to this discrepancy in the keyword definition).

Bug Fixes:

- A bug in `PCA` (page 120) allowed calculating covariance matrix for less than 3 coordinate sets is fixed.
- A bug in `mapOntoChain()` (page 151) function that caused problems when mapping all atoms is fixed.

5.10.2 0.6.1 (Mar 2, 2011)

New Features:

- `performSVD()` (page 121) function is implemented for faster and more memory efficient principal component analysis.
- `extrapolateModel()` function is implemented for extrapolating a coarse-grained model to an all atom model. See the usage example [Extend a coarse-grained model](http://prody.csb.pitt.edu/tutorials/enm_analysis/extend.html#extendmodel).
- `plog()` is implemented for enabling users to make log entries.

Improvements:

- `compare` functions are improved to handle insertion codes.
- `HierView` (page 67) allows for indexing using chain identifier and residue numbers. See usage example [Hierarchical Views](http://prody.csb.pitt.edu/tutorials/prody_tutorial/hierview.html#hierview).
- `Chain` (page 49) allows for indexing using residue number and insertion code. See usage example [Hierarchical Views](http://prody.csb.pitt.edu/tutorials/prody_tutorial/hierview.html#hierview).
- `addCoordset()` (page 39) function accepts `Atomic` (page 44) and `Ensemble` (page 132) instances as `coords` argument.
- New method `HierView.getAtoms()` (page 67) is implemented.
- `AtomGroup` (page 37) set functions check the correctness of dimension of data arrays to prevent run-time problems.
- `prody pca` script is updated to use the faster PCA method that uses SVD.

Changes:

- “backbone” definition now includes the backbone hydrogen atom (Thanks to Nahren Mascarenhas for pointing to this discrepancy in the keyword definition).

Bug Fixes:

- A bug in `PCA` (page 120) allowed calculating covariance matrix for less than 3 coordinate sets is fixed.
- A bug in `mapOntoChain()` (page 151) function that caused problems when mapping all atoms is fixed.
• **setWWPDBFTPServer()** and **getWWPDBFTPServer()** functions allow user to change or learn the WWPDB FTP server that ProDy uses to download PDB files. Default server is RCSB PDB in USA. User can change the default server to one in Europe or Japan.

• **setPDBMirrorPath()** and **getPDBMirrorPath()** functions allow user to specify or learn the path to a local PDB mirror. When specified, a local PDB mirror is preferred for accessing PDB files, over downloading them from FTP servers.

• **mapOntoChain()** (page 151) function is improved to map backbone or all atoms.

**Improvements:**

• WWPDB_PDBFetcher can download PDB files from different WWPDB FTP servers.

• WWPDB_PDBFetcher can also use local PDB mirrors for accessing PDB files.

**Changes:**

• RCSB_PDBFetcher is renamed as WWPDB_PDBFetcher.

• **mapOntoChain()** (page 151) and **matchChains()** (page 149) functions accept "ca" and "bb" as subset arguments.

• Definition of selection keyword “protein” is updated to include some non-standard amino acid abbreviations.

**Bug Fixes:**

• A bug in WWPDB_PDBFetcher causing exceptions when non-string items passed in a list is fixed.

• An important bug in parsePDB() (page 162) is fixed. When parsing backbone or C\(\alpha\) atoms, residue names were not checked and this caused parsing water atoms with name "\(\text{O}\)" or calcium ions with name "\(\text{CA}\)".

### 5.10.3 0.6 (Feb 22, 2011)

**New Features:**

• Biopython module pairwise2 and packages KDTree and Blast are incorporated in ProDy package to make installation easier. Only NumPy needs to be installed before ProDy can be used. For plotting, Matplotlib is still required.

• **Normal Mode Wizard**\(^{36}\) is distributed with ProDy source. On Linux, if VMD is installed, ProDy installer locates VMD plugins folder and installs NMWiz. On Windows, user needs to follow a separate set of instructions (see Normal Mode Wizard\(^{37}\)).

• **Gamma** (page 108) class is implemented for facilitating use of force constants based on atom type, residue type, or property. An example derived classes are GammaStructureBased (page 108) and GammaVariableCutoff (page 109).

• **calcTempFactors()** (page 100) function is implemented to calculate theoretical temperature factors.

• 5 new **ProDy Applications** (page 3) are implemented, and existing scripts are improved to output figures.

• **getModel()** (page 117) method is implemented to make function development easier.

• **resetTicks()** (page 126) function is implemented to change X and/or Y axis ticks in plots when there are discontinuities in the plotted data.

\(^{36}\)http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz

\(^{37}\)http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz
Improvements:

- **ANM.buildHessian()** (page 101) and **GNM.buildKirchhoff()** (page 112) classes are improved to accept **Gamma** (page 108) instances or other custom function as `gamma` argument. See also Custom Gamma Functions\(^{38}\).

- **Select** (page 85) class is changed to treat single word keywords differently, e.g. “backbone” or “protein”. They are interpreted 10 times faster and in use achieve much higher speed-ups when compared to composite selections. For example, using the keyword “calpha” instead of the name CA and protein, which returns the same selection, works >20 times faster.

- Optimizations in Select class to increase performance (Thanks to Paul McGuire for providing several Pythonic tips and Pyparsing specific advice).

- **applyBiomolecularTransformations()** function is improved to handle large biomolecular assemblies.

- Performance optimizations in **parsePDB()** (page 162) and other functions.

- **Ensemble** (page 132) class accepts **Atomic** (page 44) instances and automatically adds coordinate sets to the ensemble.

Changes:

- **PDBlastRecord** is renamed as **PDBBlastRecord** (page 148).

- **NMA** (page 117) instances can be index using a list or tuple of integers, e.g. `anm[1, 3, 5]`.

- “ca”, “bb”, and “sc” keywords are defined as short-hands for “calpha”, “backbone”, and “sidechain”, respectively.

- Behavior of **calcANM()** (page 103) and **calcGNM()** (page 114) functions have changed. They return the atoms used for calculation as well.

Bug Fixes:

- A bug in **assignSecondaryStructure()** function is fixed.

- Bug fixes in **prody anm** (page 4) and **prody gnm** (page 11).

- Bug fixes in **showSqFlucts()** (page 126) and **showProjection()** (page 124) functions.

**Normal Mode Wizard**

- NMWiz can be used as a graphical interface to ProDy. ANM or PCA calculations can be performed for molecules that are loaded in VMD.

- User can set default color for arrow graphics and paths to ANM and PCA scripts.

- Optionally, NMWiz can preserve the current view in VMD display window when loading a new dataset. Check the box in the NMWiz GUI main window.

- A bug that prevented selecting residues from plot window is fixed.

5.11 ProDy 0.5 Series

\(^{38}\)http://prody.csb.pitt.edu/tutorials/enm_analysis/gamma.html#gamma
5.11.1 0.5.3 (Feb 11, 2011)

New Features:

- Membership, equality, and non-equality test operation are defined for all atomic classes. See Operations on Selections\(^\text{39}\).
- Two functions are implemented for dealing with anisotropic temperature factors: calcADPAxes() (page 142) and buildADPMatrix() (page 142).
- NMA.setEigens() (page 118) and NMA.addEigenpair() (page 117) methods are implemented to assist analysis of normal modes calculated using external software.
- parseNMD() (page 119) is implemented for parsing NMD files.
- parseModes() (page 105) is implemented for parsing normal mode data.
- parseArray() (page 105) is implementing for reading numeric data, particularly normal mode data calculated using other software for analysis using ProDy.
- The method in [BH02] (page 260) to calculate overlap between covariance matrices is implemented as calcCovOverlap() (page 103) function.
- trimEnsemble() to trim Ensemble (page 132) instances is implemented.
- checkUpdates() (page 205) to check for ProDy updates is implemented.

Changes:

- Change in default behavior of parsePDB() (page 162) function. When alternate locations exist, those indicated by A are parsed. For parsing all alternate locations user needs to pass altloc=True argument.
- getSumOfWeights() is renamed as calcSumOfWeights().
- mapAtomsToChain() is renamed as mapOntoChain() (page 151).
- ProDyStartLogFile() is renamed as startLogfile() (page 205).
- ProDyCloseLogFile() is renamed as closeLogFile() (page 205).
- ProDySetVerbosity() is renamed as changeVerbosity().

Improvements:

- A few bugs in ensemble and dynamics classes are fixed.
- Improvements in RCSB_PDBFetcher allow it not to miss a PDB file if it exists in the target folder.
- writeNMD() (page 119) is fixed to output B-factors (Thanks to Dan Holloway for pointing it out).

\(^{39}\)http://prody.csb.pitt.edu/tutorials/prody_tutorial/selection.html#selection-operations
5.11.2 0.5.2 (Jan 12, 2011)

Bug Fixes:

• An important fix in `sampleModes()` (page 128) function was made (Thanks to Alberto Perez for finding the bug and suggesting a solution).

Improvements:

• Improvements in `ANM.calcModes()` (page 101), `GNM.calcModes()` (page 112), and `PCA.calcModes()` (page 120) methods prevent Numpy/Scipy throwing an exception when more than available modes are requested by the user.

• Improvements in `blastPDB()` (page 149) enable ProDy to throw an exception when no internet connection is found, and warn user when downloads fail due to restriction in network regulations (Thanks to Serkan Apaydin for helping identify these improvements).

• New example `Write PDB file`.

5.11.3 0.5.1 (Dec 31, 2010)

Changes in dependencies:

• Scipy (linear algebra module) is not required package anymore. When available it replaces Numpy (linear algebra module) for greater flexibility and efficiency. A warning message is printed when Scipy is not found.

• Biopython KDTree module is not required for ENM calculations (specifically for building Hessian (ANM) or Kirchoff (GNM) matrices). When available it is used to increase the performance. A warning message is printed when KDTree is not found.

5.11.4 0.5 (Dec 21, 2010)

New Features:

• `AtomPointer` (page 68) base class for classes pointing to atoms in an `AtomGroup` (page 37).

• `AtomPointer` (page 68) instances (Selection, Residue, etc.) can be added. See `Operations on Selections` for examples.

• `Select.getIndices()` (page 85) and `Select.getBoolArray()` (page 85) methods to expand the usage of `Select` (page 85).

• `sliceVector()` (page 104) and `sliceMode()` (page 104) functions.

• `saveModel()` (page 107) and `loadModel()` (page 107) functions for saving and loading NMA data.

• `parsePDBStream()` (page 161) can now parse specific chains or alternate locations from a PDB file.

• `alignCoordsets()` (page 144) is implemented to superimpose coordinate sets of an `AtomGroup` (page 37) instance.

Bug Fixes:

• A bug in `parsePDBStream()` (page 161) that caused unidentified errors when a model in a multiple model file did not have the same number of atoms is fixed.

Changes:

40http://prody.csb.pitt.edu/tutorials/structure_analysis/pdbfiles.html#writepdb
41http://prody.csb.pitt.edu/tutorials/prody_tutorial/selection.html#selection-operations
• Iterating over a `Chain` (page 49) instance yields `Residue` (page 69) instances.

• `Vector` (page 115) instantiation requires an `array` only. `name` is an optional argument.

• Functions starting with `get` and performing a calculations are renamed to start with `calc`, e.g. `getRMSD()` is now `calcRMSD()` (page 144).

## 5.12 ProDy 0.2 Series

### 5.12.1 0.2 (Nov 16, 2010)

**Important Changes:**

• Single word keywords *not* followed by “and” logical operator are not accepted, e.g. “protein within 5 of water” will raise a `SelectionError`, use “protein and within 5 of water” instead.

• `findMatchingChains()` is renamed to `matchChains()` (page 149).

• `showOverlapMatrix()` is renamed to `showOverlapTable()` (page 124).

• Modules are reorganized.

**New Features:**

• `Atomic` (page 44) for easy type checking.

• `Contacts` (page 140) for faster intermolecular contact identification.

• `Select` (page 85) can identify intermolecular contacts. See `Intermolecular Contacts`[^42] for an examples and details.

• `sampleModes()` (page 128) implemented for sampling conformations along normal modes.

**Improvements:**

• `proteins.compare` (page 149) functions are improved. Now they perform sequence alignment if simple residue number/identity based matchin does not work, or if user passes `palign=True` argument. This impacts the speed of X-ray ensemble analysis.

• `Select` (page 85) can cache data optionally. This results in speeds up from 2 to 50 folds depending on number of atoms and selection operations.

• Implementation of `showProjection()` (page 124) is completed.

### Normal Mode Wizard

**Release 0.2.3**

• For each mode a molecule for drawing arrows and a molecule for showing animation is formed in VMD on demand. NMWiz remembers a color associated with a mode.

• Deselecting a residue by clicking on a plot is possible.

[^42]: [http://prody.csb.pitt.edu/tutorials/structure_analysis/contacts.html#contacts](http://prody.csb.pitt.edu/tutorials/structure_analysis/contacts.html#contacts)
• A bug causing incorrect parsing of NMD files from ANM server is fixed.

**Release 0.2.2**

• Selection string option allows user to show a subset of arrows matching a VMD selection string. Optionally, this selection string may affect protein and animation representations.
• A bug that caused problems when over plotting modes is removed.
• A bug affecting line width changes in plots is removed.
• Selected residue representations are colored according to the color of the plot.

**Release 0.2.1**

• Usability improvements.
• Loading the same data file more than once is prevented.
• If a GUI window for a dataset is closed, it can be reloaded from the main window.
• A dataset and GUI can be deleted from the VMD session via the main window.

**Release 0.2**

• Instant documentation is improved.
• Problem with clearing selections is fixed.
• Plotting options frame is populated.
• Multiple modes can be plotted on the same canvas.

---

### 5.13 ProDy 0.1 Series

- 0.1.2 (Nov 9, 2010) (page 254)
- 0.1.1 (Nov 8, 2010) (page 254)
- 0.1 (Nov 7, 2010) (page 255)

---

#### 5.13.1 0.1.2 (Nov 9, 2010)

• Important bug fixes and improvements in NMA helper and plotting functions.
• Documentation updates and improvements.

#### 5.13.2 0.1.1 (Nov 8, 2010)

• Important bug fixes and improvements in chain comparison functions.
• Bug fixes.
• Source clean up.
• Documentation improvements.
5.13.3 0.1 (Nov 7, 2010)

- First release.
ProDy is a free and open-source Python package for protein structural dynamics and sequence evolution analysis. It is designed as a flexible and responsive API suitable for interactive usage and application development.

6.1 People

ProDy is being developed in Bahar Lab¹ at the University of Pittsburgh² with support from NIH R01 GM099738 award.

6.1.1 Development Team

Ahmet Bakan³ initiated the ProDy project, designed and developed ProDy, NMWiz, Evol, and DruGUI.

Chakra Chennubhotla⁴ is currently overseeing the overall development of ProDy.

Anindita Dutta⁵ contributed to the development of Evol, database (page 94) and sequence (page 166) modules.

Tim Lezon contributed to development of Rotations and Translation of Blocks and Membrane ENM models.

Wenzhi Mao⁶ contributed to development of MSA analysis functions.

Lidio Meireles⁷ provided insightful comments on the design of ProDy, and contributed to the development of ProDy Applications (page 3).

6.1.2 Contributors

In addition to the development team members, we acknowledge contributions and feedback from the following individuals:

Ying Liu⁸ provided the code for Perturbation Response Scanning method.

¹http://www.ccb.pitt.edu/faculty/bahar/
²http://www.pitt.edu/
³http://ahmetbakan.com
⁴http://www.csb.pitt.edu/Faculty/Chakra/
⁵http://www.linkedin.com/pub/anindita-dutta/5a/568/a90
⁶http://www.linkedin.com/pub/wenzhi-mao/2a/29a/29
⁷http://www.linkedin.com/in/lidio
⁸http://www.linkedin.com/pub/ying-liu/15/48b/5a9
Kian Ho\textsuperscript{9} contributed with bug fixes and unit tests for DSSP functions.
Gökçen Eraslan\textsuperscript{10} contributed with bug fixes and development and maintenance insights.

6.2 Citing

When using \textit{ProDy} or \textit{NMWiz} in published work, please cite:

Bakan A, Meireles LM, Bahar I.
ProDy: Protein Dynamics Inferred from Theory and Experiments.

When using \textit{pairwise2} or \textit{KDTree} modules in published work, please cite:

Biopython: freely available Python tools for computational molecular biology and bioinformatics.

6.3 Credits

ProDy makes use of the following great software:

\texttt{pyparsing}\textsuperscript{11} is used to define the sophisticated atom selection grammar. This makes every user a power user by enabling fast access to and easy handling of atomic data via simple selection statements.

\texttt{Biopython}\textsuperscript{12} KDTree package and pairwise2 module, which are distributed ProDy, significantly enrich and improve the ProDy user experience. KDtree package allows for fast distance based selections making atom selections suitable for contact identification. pairwise2 module enables performing sequence alignment for protein structure comparison and ensemble analysis.

ProDy requires \texttt{NumPy}\textsuperscript{13} for almost all major functionality including, but not limited to, storing atomic data and performing normal mode calculations. The power and speed of NumPy makes ProDy suitable for interactive and high-throughput structural analysis.

Finally, ProDy can benefit from \texttt{SciPy}\textsuperscript{14} and \texttt{Matplotlib}\textsuperscript{15} packages. SciPy makes ProDy normal calculations more flexible and on low memory machines possible. Matplotlib allows greatly enriches user experience by allowing plotting protein dynamics data calculated using ProDy.

6.4 Funding

Continued development of protein dynamics software \textit{ProDy} is supported by NIH through R01 GM099738 award.

\textsuperscript{9}https://github.com/kianho
\textsuperscript{10}http://blog.yeredusuncedernegi.com/
\textsuperscript{11}http://pyparsing.wikispaces.com
\textsuperscript{12}http://biopython.org
\textsuperscript{13}http://www.numpy.org
\textsuperscript{14}http://www.scipy.org
\textsuperscript{15}http://matplotlib.org
6.5 License

6.5.1 ProDy

ProDy is available under the MIT License\(^\text{16}\):

ProDy: A Python Package for Protein Dynamics Analysis

Copyright (C) 2010-2014 University of Pittsburgh

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

6.5.2 Biopython

Biopython\(^\text{17}\) KDTree package and pairwise2 module are distributed with the ProDy package. Biopython is developed by The Biopython Consortium and is available under the Biopython license\(^\text{18}\):

Biopython License Agreement

Permission to use, copy, modify, and distribute this software and its documentation with or without modifications and for any purpose and without fee is hereby granted, provided that any copyright notices appear in all copies and that both those copyright notices and this permission notice appear in supporting documentation, and that the names of the contributors or copyright holders not be used in advertising or publicity pertaining to distribution of the software without specific prior permission.

THE CONTRIBUTORS AND COPYRIGHT HOLDERS OF THIS SOFTWARE DISCLAIM ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

\(^{16}\)http://opensource.org/licenses/MIT

\(^{17}\)http://biopython.org

\(^{18}\)http://www.biopython.org/DIST/LICENSE
6.5.3 Pyparsing

The pyparsing\textsuperscript{19} module is distributed with the ProDy package. Pyparsing is developed by Paul T. McGuire and is available under the MIT License\textsuperscript{20}:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

6.5.4 Argparse

The argparse module\textsuperscript{21} is distributed with the ProDy package. Argparse is developed by Steven J. Bethard and is available under the Python Software Foundation License\textsuperscript{22}.

\textsuperscript{19}http://pyparsing.wikispaces.com
\textsuperscript{20}http://opensource.org/licenses/MIT
\textsuperscript{21}http://code.google.com/p/argparse/
\textsuperscript{22}http://docs.python.org/license.html
BIBLIOGRAPHY


[TL12] Lezon TR, Bahar I, Constraints Imposed by the Membrane Selectively Guide the Alternating Access Dynamics of the Glutamate Transporter GltPh


<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>prody.apps</td>
<td>192</td>
</tr>
<tr>
<td>prody.apps.evol_apps.evol_coevol</td>
<td>193</td>
</tr>
<tr>
<td>prody.apps.evol_apps.evol_conserv</td>
<td>193</td>
</tr>
<tr>
<td>prody.apps.evol_apps.evol_fetch</td>
<td>194</td>
</tr>
<tr>
<td>prody.apps.evol_apps.evol_filter</td>
<td>195</td>
</tr>
<tr>
<td>prody.apps.evol_apps.evol_merge</td>
<td>195</td>
</tr>
<tr>
<td>prody.apps.evol_apps.evol_occupancy</td>
<td>196</td>
</tr>
<tr>
<td>prody.apps.evol_apps.evol_rankorder</td>
<td>196</td>
</tr>
<tr>
<td>prody.apps.evol_apps.evol_refine</td>
<td>197</td>
</tr>
<tr>
<td>prody.apps.evol_apps.evol_search</td>
<td>198</td>
</tr>
<tr>
<td>prody.apps.prody_apps.prody_align</td>
<td>198</td>
</tr>
<tr>
<td>prody.apps.prody_apps.prody_anm</td>
<td>199</td>
</tr>
<tr>
<td>prody.apps.prody_apps.prody_biomol</td>
<td>200</td>
</tr>
<tr>
<td>prody.apps.prody_apps.prody_blast</td>
<td>200</td>
</tr>
<tr>
<td>prody.apps.prody_apps.prody_catdcd</td>
<td>201</td>
</tr>
<tr>
<td>prody.apps.prody_apps.prody_contacts</td>
<td>201</td>
</tr>
<tr>
<td>prody.apps.prody_apps.prody_fetch</td>
<td>202</td>
</tr>
<tr>
<td>prody.apps.prody_apps.prody_gnm</td>
<td>202</td>
</tr>
<tr>
<td>prody.apps.prody_apps.prody_pca</td>
<td>203</td>
</tr>
<tr>
<td>prody.apps.prody_apps.prody_pcapr</td>
<td>204</td>
</tr>
<tr>
<td>prody.atomic</td>
<td>31</td>
</tr>
<tr>
<td>prody.atomic.atom</td>
<td>33</td>
</tr>
<tr>
<td>prody.atomic.atomgroup</td>
<td>37</td>
</tr>
<tr>
<td>prody.atomic.atomic</td>
<td>44</td>
</tr>
<tr>
<td>prody.atomic.atommap</td>
<td>44</td>
</tr>
<tr>
<td>prody.atomic.bond</td>
<td>49</td>
</tr>
<tr>
<td>prody.atomic.chain</td>
<td>49</td>
</tr>
<tr>
<td>prody.atomic.fields</td>
<td>54</td>
</tr>
<tr>
<td>prody.atomic.flags</td>
<td>56</td>
</tr>
<tr>
<td>prody.atomic.functions</td>
<td>66</td>
</tr>
<tr>
<td>prody.atomic.hierview</td>
<td>67</td>
</tr>
<tr>
<td>prody.atomic.pointer</td>
<td>68</td>
</tr>
<tr>
<td>prody.atomic.residue</td>
<td>69</td>
</tr>
<tr>
<td>prody.atomic.segment</td>
<td>74</td>
</tr>
<tr>
<td>prody.atomic.select</td>
<td>78</td>
</tr>
<tr>
<td>prody.atomic.select.function</td>
<td>86</td>
</tr>
<tr>
<td>prody.atomic.subset</td>
<td>90</td>
</tr>
<tr>
<td>prody.database</td>
<td>94</td>
</tr>
<tr>
<td>prody.database.pfam</td>
<td>95</td>
</tr>
<tr>
<td>prody.dynamics</td>
<td>96</td>
</tr>
<tr>
<td>prody.dynamics.analysis</td>
<td>99</td>
</tr>
<tr>
<td>prody.dynamics.anm</td>
<td>101</td>
</tr>
<tr>
<td>prody.dynamics.compare</td>
<td>103</td>
</tr>
<tr>
<td>prody.dynamics.editing</td>
<td>104</td>
</tr>
<tr>
<td>prody.dynamics.functions</td>
<td>105</td>
</tr>
<tr>
<td>prody.dynamics.gamma</td>
<td>108</td>
</tr>
<tr>
<td>prody.dynamics.gnm</td>
<td>112</td>
</tr>
<tr>
<td>prody.dynamics.heatmap</td>
<td>114</td>
</tr>
<tr>
<td>prody.dynamics.mode</td>
<td>115</td>
</tr>
<tr>
<td>prody.dynamics.mode_set</td>
<td>116</td>
</tr>
<tr>
<td>prody.dynamics.nma</td>
<td>117</td>
</tr>
<tr>
<td>prody.dynamics.nmdfile</td>
<td>118</td>
</tr>
<tr>
<td>prody.dynamics.pca</td>
<td>120</td>
</tr>
<tr>
<td>prody.dynamics.plotting</td>
<td>124</td>
</tr>
<tr>
<td>prody.dynamics.rtb</td>
<td>127</td>
</tr>
<tr>
<td>prody.dynamics.sampling</td>
<td>128</td>
</tr>
<tr>
<td>prody.ensemble</td>
<td>130</td>
</tr>
<tr>
<td>prody.ensemble.conformation</td>
<td>131</td>
</tr>
<tr>
<td>prody.ensemble.ensemble</td>
<td>132</td>
</tr>
<tr>
<td>prody.ensemble.functions</td>
<td>134</td>
</tr>
<tr>
<td>prody.ensemble.pdbensemble</td>
<td>135</td>
</tr>
<tr>
<td>prody.kdtree</td>
<td>137</td>
</tr>
<tr>
<td>prody.kdtree.kdtree</td>
<td>137</td>
</tr>
<tr>
<td>prody.measure</td>
<td>138</td>
</tr>
<tr>
<td>prody.measure.contacts</td>
<td>140</td>
</tr>
<tr>
<td>prody.measure.measure</td>
<td>140</td>
</tr>
<tr>
<td>prody.measure.transform</td>
<td>143</td>
</tr>
<tr>
<td>prody.proteins</td>
<td>146</td>
</tr>
<tr>
<td>prody.proteins.proteins</td>
<td>146</td>
</tr>
</tbody>
</table>

262
A
accession (DBRef attribute), 156
acidic, 58
acyclic, 58
addCoordset() (AtomGroup method), 39
addCoordset() (Ensemble method), 132
addCoordset() (PDBEnsemble method), 135
addEigenpair() (ANM method), 101
addEigenpair() (EDA method), 122
addEigenpair() (GNM method), 112
addEigenpair() (NMA method), 117
addEigenpair() (PCA method), 120
addEigenpair() (RTB method), 127
addext() (in module prody.utilities.pathtools), 191
addFile() (Trajectory method), 181
addHandler() (PackageLogger method), 188
addNonstdAminoacid() (in module prody.atomic.flags), 65
alignCoordsets() (in module prody.measure.transform), 144
alignPDBEnsemble() (in module prody.ensemble.functions), 134
aliphatic, 58
all, 63
alnum() (in module prody.utilities.misctools), 190
altloc, 54
aminoacid, 57
anisou, 54
ANM (class in prody.dynamics.anm), 101
apply() (Transformation method), 143
applyMutinfoCorr() (in module prody.sequence.analysis), 167
applyMutinfoNorm() (in module prody.sequence.analysis), 167
applyTransformation() (in module prody.measure.transform), 144
aromatic, 58
assignSecstr() (in module prody.proteins.header), 158
at, 60
Atom (class in prody.atomic.atom), 33
AtomGroup (class in prody.atomic.atomgroup), 37
Atomic (class in prody.atomic.atomic), 44
AtomMap (class in prody.atomic.atommmap), 45
AtomPointer (class in prody.atomic.pointer), 68
AtomSubset (class in prody.atomic.subset), 90
B
backbone, 58
backbonefull, 58
backupFile() (in module prody.utilities.pathtools), 190
basic, 58
bb, 58
bbfull, 58
bend, 63
beta, 54
blastPDB() (in module prody.proteins.blastpdb), 149
Bond (class in prody.atomic.bond), 49
bridge, 63
buildADPMatrix() (in module prody.measure.measure), 142
buildBiomolecules() (in module prody.proteins.header), 159
buildCovariance() (EDA method), 122
buildCovariance() (PCA method), 120
buildDirectInfoMatrix() (in module prody.sequence.analysis), 169
buildDistMatrix() (in module prody.sequence.analysis), 169
buildHessian() (ANM method), 101
buildHessian() (RTB method), 127
buildKirchhoff() (GNM method), 112
buildMutinfoMatrix() (in module prody.sequence.analysis), 167
buildOMESMatrix() (in module prody.sequence.analysis), 168
buildSCAMatrix() (in module prody.sequence.analysis), 169
buildSeqidMatrix() (in module prody.sequence.analysis), 168
buried, 58
calcADPAxes() (in module prody.measure.measure), 142
calcADPs() (in module prody.measure.measure), 143
calcAngle() (in module prody.measure.measure), 141
calcANM() (in module prody.dynamics.anm), 103
calcCenter() (in module prody.measure.measure), 141
calcCollectivity() (in module prody.dynamics.analysis), 99
calcCovariance() (in module prody.dynamics.analysis), 99
calcCovOverlap() (in module prody.dynamics.compare), 103
calcCrossCorr() (in module prody.dynamics.analysis), 99
calcCrossProjection() (in module prody.dynamics.analysis), 99
calcCumulOverlap() (in module prody.dynamics.compare), 103
calcDeformVector() (in module prody.measure.measure), 142
calcDihedral() (in module prody.measure.measure), 141
calcDistance() (in module prody.measure.measure), 140
calcFractVariance() (in module prody.dynamics.analysis), 100
calcGNM() (in module prody.dynamics.gnm), 114
calcGyradius() (in module prody.measure.measure), 141
calcMeff() (in module prody.sequence.analysis), 169
calcModes() (ANM method), 101
calcModes() (EDA method), 122
calcModes() (GNM method), 112
calcModes() (PCA method), 120
calcModes() (RTB method), 127
calcMSAOccupancy() (in module prody.sequence.analysis), 167
calcMSF() (in module prody.dynamics.measure), 141
calcOccupancies() (in module prody.ensemble.functions), 134
calcOmega() (in module prody.measure.measure), 141
calcOverlap() (in module prody.dynamics.compare), 103
calcPerturbResponse() (in module prody.dynamics.analysis), 100
calcPhi() (in module prody.measure.measure), 141
calcProjection() (in module prody.dynamics.analysis), 100
calcPsi() (in module prody.measure.measure), 141
calcRankorder() (in module prody.sequence.analysis), 168
calcRMSD() (in module prody.measure.transform), 144
calcRMSF() (in module prody.measure.measure), 141
calcShannonEntropy() (in module prody.sequence.analysis), 167
calcSqFlucts() (in module prody.dynamics.analysis), 100
calcSubspaceOverlap() (in module prody.dynamics.compare), 103
calcTempFactors() (in module prody.dynamics.analysis), 100
calcTransformation() (in module prody.measure.transform), 144
call (Field attribute), 55
calpha, 58
carbon, 62
cg, 60
cg (Polymer attribute), 156
chain, 54
chain (Chemical attribute), 154
Chain (class in prody.atomic.chain), 49
charge, 54
charged, 58
checkCoords() (in module prody.utilities.checkers), 186
checkTypes() (in module prody.utilities.checkers), 187
checkUpdates() (in module prody), 205
checkWeights() (in module prody.utilities.checkers), 187
Chemical (class in prody.proteins.header), 153
chid, 54
chid (Polymer attribute), 156
chid (Polymer attribute), 156
chid (Polymer attribute), 156
chindex, 54
clear() (PackageLogger method), 188
close() (DCDFile method), 175
close() (MSAFile method), 171
close() (PackageLogger method), 188
close() (TrajBase method), 179
close() (Trajectory method), 181
close() (TrajFile method), 183
closed (MSAFile attribute), 172
closeLogfile() (in module prody), 205
coil, 63
comments (Polymer attribute), 156
Conformation (class in prody.ensemble.conformation), 131
confProDy() (in module prody), 204
Contacts (class in prody.measure.contacts), 140
copy() (Atom method), 33
copy() (AtomGroup method), 39
copy() (Atomic method), 44
copy() (AtomMap method), 46
copy() (AtomPointer method), 68
copy() (AtomSubset method), 90
copy() (Chain method), 49
copy() (Residue method), 69
copy() (Segment method), 74
copy() (Selection method), 86
copy() (Sequence method), 173
copyFile() (in module prody.utilities.pathtools), 191
countBytes() (in module prody.utilities.misctools), 190
countLabel() (MSA method), 170
critical() (PackageLogger method), 188
cyclic, 58

D
database (DBRef attribute), 156
dbabbr (DBRef attribute), 157
DBRef (class in prody.proteins.header), 156
dbrefs (Polymer attribute), 156
DCDFile (class in prody.trajectory.dcdfile), 175
debug() (PackageLogger method), 188
deformAtoms() (in module prody.dynamics.sampling), 128
defSelectionMacro() (in module prody.atomic.select), 85
delCoordset() (AtomGroup method), 39
delCoordset() (Ensemble method), 132
delCoordset() (PDBEnsemble method), 135
delData() (AtomGroup method), 39
delFlags() (AtomGroup method), 39
delHandler() (PackageLogger method), 188
delNonstdAminoacid() (in module prody.atomic.flags), 65
delSelectionMacro() (in module prody.atomic.select), 86
dep (Field attribute), 55
dep_pl (Field attribute), 55
desc (Field attribute), 55
description (Chemical attribute), 154
dictElement() (in module prody.utilities.misctools), 190
diff (DBRef attribute), 157
doc (Field attribute), 56
doc_pl (Field attribute), 56
dtype (Field attribute), 56
dummy, 63

e
ec (Polymer attribute), 156
EDA (class in prody.dynamics.pca), 122
element, 54
engineered (Polymer attribute), 156
Ensemble (class in prody.ensemble.ensemble), 132
environment variable
HOME, 222
PATH, 2, 29, 30, 207
PYTHONPATH, 207
error() (PackageLogger method), 188
Everything (class in prody.utilities.misctools), 189
evol_coevol() (in module prody.apps.evol_apps.evol_coevol), 193
evol_conserv() (in module prody.apps.evol_apps.evol_conserv), 193
evol_fetch() (in module prody.apps.evol_apps.evol_fetch), 194
evol_filter() (in module prody.apps.evol_apps.evol_filter), 195
evol_merge() (in module prody.apps.evol_apps.evol_merge), 195
evol_occupancy() (in module prody.apps.evol_apps.evol_occupancy), 196
evol_rankorder() (in module prody.apps.evol_apps.evol_rankorder), 196
evol_refine() (in module prody.apps.evol_apps.evol_refine), 197
evol_search() (in module prody.apps.evol_apps.evol_search), 198
evecDSSP() (in module prody.proteins.dssp), 152
evecSTRIDE() (in module prody.proteins.stride), 164
exit() (PackageLogger method), 188
extended, 62
extendModel() (in module prody.dynamics.editing), 104
extendModel() (in module prody.dynamics.editing), 104
extendVector() (in module prody.dynamics.editing), 104

F
fetchPDB() (in module prody.proteins.localpdb), 160
fetchPDBClusters() (in module prody.proteins.pdbclusters), 161
fetchPDBfromMirror() (in module prody.proteins.localpdb), 160
fetchPDBligand() (in module prody.proteins.pdbligands), 164
fetchPDBviaFTP() (in module prody.proteins.wwpdb), 165
fetchPDBviaHTTP() (in module prody.proteins.wwpdb), 165
fetchPfamMSA() (in module prody.database.pfam), 95
Field (class in prody.atomic.fields), 55
<table>
<thead>
<tr>
<th>Function</th>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>getBetas()</td>
<td>AtomGroup</td>
<td>39</td>
</tr>
<tr>
<td>getBetas()</td>
<td>AtomMap</td>
<td>46</td>
</tr>
<tr>
<td>getBetas()</td>
<td>AtomSubset</td>
<td>91</td>
</tr>
<tr>
<td>getBetas()</td>
<td>Chain</td>
<td>50</td>
</tr>
<tr>
<td>getBetas()</td>
<td>Residue</td>
<td>69</td>
</tr>
<tr>
<td>getBetas()</td>
<td>Segment</td>
<td>74</td>
</tr>
<tr>
<td>getBetas()</td>
<td>Selection</td>
<td>86</td>
</tr>
<tr>
<td>getBoolArray()</td>
<td>Select</td>
<td>85</td>
</tr>
<tr>
<td>getBySerial()</td>
<td>AtomGroup</td>
<td>39</td>
</tr>
<tr>
<td>getChain()</td>
<td>HierView</td>
<td>67</td>
</tr>
<tr>
<td>getChain()</td>
<td>Residue</td>
<td>69</td>
</tr>
<tr>
<td>getChain()</td>
<td>Segment</td>
<td>74</td>
</tr>
<tr>
<td>getCharge()</td>
<td>Atom</td>
<td>34</td>
</tr>
<tr>
<td>getCharge()</td>
<td>Chain</td>
<td>50</td>
</tr>
<tr>
<td>getCharge()</td>
<td>Residue</td>
<td>69</td>
</tr>
<tr>
<td>getCharge()</td>
<td>Segment</td>
<td>74</td>
</tr>
<tr>
<td>getCharge()</td>
<td>Selection</td>
<td>86</td>
</tr>
<tr>
<td>getChid()</td>
<td>Atom</td>
<td>34</td>
</tr>
<tr>
<td>getChid()</td>
<td>Chain</td>
<td>50</td>
</tr>
<tr>
<td>getChid()</td>
<td>Residue</td>
<td>70</td>
</tr>
<tr>
<td>getChid()</td>
<td>Segment</td>
<td>74</td>
</tr>
<tr>
<td>getChid()</td>
<td>Selection</td>
<td>87</td>
</tr>
<tr>
<td>getChindex()</td>
<td>Atom</td>
<td>34</td>
</tr>
<tr>
<td>getChindex()</td>
<td>Chain</td>
<td>50</td>
</tr>
<tr>
<td>getChindex()</td>
<td>Residue</td>
<td>70</td>
</tr>
<tr>
<td>getChindex()</td>
<td>Segment</td>
<td>74</td>
</tr>
<tr>
<td>getChindex()</td>
<td>Selection</td>
<td>87</td>
</tr>
<tr>
<td>getChindices()</td>
<td>Atom</td>
<td>34</td>
</tr>
<tr>
<td>getChindices()</td>
<td>Chain</td>
<td>50</td>
</tr>
<tr>
<td>getChindices()</td>
<td>Residue</td>
<td>70</td>
</tr>
<tr>
<td>getChindices()</td>
<td>Segment</td>
<td>74</td>
</tr>
<tr>
<td>getChindices()</td>
<td>Selection</td>
<td>87</td>
</tr>
<tr>
<td>getConformation()</td>
<td>Ensemble</td>
<td>132</td>
</tr>
<tr>
<td>getConformation()</td>
<td>PDBEnsemble</td>
<td>135</td>
</tr>
<tr>
<td>getCoords()</td>
<td>Atom</td>
<td>34</td>
</tr>
<tr>
<td>getCoords()</td>
<td>Chain</td>
<td>50</td>
</tr>
<tr>
<td>getCoords()</td>
<td>Residue</td>
<td>70</td>
</tr>
<tr>
<td>getCoords()</td>
<td>Segment</td>
<td>75</td>
</tr>
<tr>
<td>getCoords()</td>
<td>Selection</td>
<td>87</td>
</tr>
<tr>
<td>getCoords()</td>
<td>TrajBase</td>
<td>179</td>
</tr>
<tr>
<td>getCoords()</td>
<td>Trajectory</td>
<td>181</td>
</tr>
<tr>
<td>getCoords()</td>
<td>TrajFile</td>
<td>183</td>
</tr>
<tr>
<td>getCoordsets()</td>
<td>Atom</td>
<td>34</td>
</tr>
<tr>
<td>getCoordsets()</td>
<td>AtomGroup</td>
<td>40</td>
</tr>
<tr>
<td>getCoordsets()</td>
<td>AtomMap</td>
<td>47</td>
</tr>
<tr>
<td>getCoordsets()</td>
<td>AtomSubset</td>
<td>91</td>
</tr>
<tr>
<td>getCoordsets()</td>
<td>Chain</td>
<td>50</td>
</tr>
<tr>
<td>getCoordsets()</td>
<td>DCDFile</td>
<td>175</td>
</tr>
<tr>
<td>getCoordsets()</td>
<td>Ensemble</td>
<td>132</td>
</tr>
<tr>
<td>getCoordsets()</td>
<td>PDBEnsemble</td>
<td>135</td>
</tr>
<tr>
<td>getCoordsets()</td>
<td>Residue</td>
<td>70</td>
</tr>
<tr>
<td>getCoordsets()</td>
<td>Segment</td>
<td>75</td>
</tr>
<tr>
<td>getCoordsets()</td>
<td>Selection</td>
<td>87</td>
</tr>
<tr>
<td>getCutoff()</td>
<td>ANM</td>
<td>102</td>
</tr>
<tr>
<td>getCutoff()</td>
<td>GNM</td>
<td>113</td>
</tr>
<tr>
<td>getCutoff()</td>
<td>NMA</td>
<td>117</td>
</tr>
<tr>
<td>getCutoff()</td>
<td>PCA</td>
<td>121</td>
</tr>
<tr>
<td>getCutoff()</td>
<td>RTB</td>
<td>127</td>
</tr>
<tr>
<td>getCSLabels()</td>
<td>Atom</td>
<td>34</td>
</tr>
<tr>
<td>getCSLabels()</td>
<td>AtomGroup</td>
<td>39</td>
</tr>
<tr>
<td>getCSLabels()</td>
<td>AtomMap</td>
<td>46</td>
</tr>
<tr>
<td>getCSLabels()</td>
<td>AtomSubset</td>
<td>91</td>
</tr>
<tr>
<td>getCSLabels()</td>
<td>Chain</td>
<td>50</td>
</tr>
<tr>
<td>getCSLabels()</td>
<td>Residue</td>
<td>70</td>
</tr>
<tr>
<td>getCSLabels()</td>
<td>Segment</td>
<td>75</td>
</tr>
<tr>
<td>getCSLabels()</td>
<td>Selection</td>
<td>87</td>
</tr>
<tr>
<td>getData()</td>
<td>Atom</td>
<td>34</td>
</tr>
<tr>
<td>getData()</td>
<td>AtomGroup</td>
<td>40</td>
</tr>
<tr>
<td>getData()</td>
<td>AtomMap</td>
<td>47</td>
</tr>
<tr>
<td>getData()</td>
<td>AtomSubset</td>
<td>91</td>
</tr>
<tr>
<td>getData()</td>
<td>Chain</td>
<td>50</td>
</tr>
<tr>
<td>getData()</td>
<td>Residue</td>
<td>70</td>
</tr>
<tr>
<td>getData()</td>
<td>Segment</td>
<td>75</td>
</tr>
<tr>
<td>getData()</td>
<td>Selection</td>
<td>87</td>
</tr>
<tr>
<td>getDataLabels()</td>
<td>Atom</td>
<td>34</td>
</tr>
<tr>
<td>getDataLabels()</td>
<td>AtomGroup</td>
<td>40</td>
</tr>
<tr>
<td>getDataLabels()</td>
<td>AtomMap</td>
<td>47</td>
</tr>
<tr>
<td>getDataLabels()</td>
<td>AtomPointer</td>
<td>68</td>
</tr>
<tr>
<td>getDataLabels()</td>
<td>AtomSubset</td>
<td>91</td>
</tr>
<tr>
<td>getDataLabels()</td>
<td>Chain</td>
<td>50</td>
</tr>
<tr>
<td>getDataLabels()</td>
<td>Residue</td>
<td>70</td>
</tr>
<tr>
<td>getDataLabels()</td>
<td>Segment</td>
<td>75</td>
</tr>
<tr>
<td>getDataLabels()</td>
<td>Selection</td>
<td>87</td>
</tr>
</tbody>
</table>
getDataType() (Atom method), 34
getDataType() (AtomGroup method), 40
getDataType() (AtomMap method), 47
getDataType() (AtomPointer method), 68
getDataType() (AtomSubset method), 91
getDataType() (Chain method), 50
getDataType() (Residue method), 70
getDataType() (Segment method), 75
getDataType() (Selection method), 87
getDeviations() (Conformation method), 131
getDeviations() (Ensemble method), 133
getDeviations() (Frame method), 178
getDeviations() (PDBConformation method), 131
getDeviations() (PDBEnsemble method), 135
getDistances() (KDTree method), 138
getDocstr() (Field method), 55
getEigval() (Mode method), 115
getEigvals() (ANM method), 102
getEigvals() (EDA method), 123
getEigvals() (GNM method), 113
getEigvals() (ModeSet method), 116
getEigvals() (NMA method), 117
getEigvals() (PCA method), 121
getEigvals() (RTB method), 127
getEigvec() (Mode method), 115
getEigvecs() (ANM method), 102
getEigvecs() (EDA method), 123
getEigvecs() (GNM method), 113
getEigvecs() (ModeSet method), 116
getEigvecs() (NMA method), 117
getEigvecs() (PCA method), 121
getEigvecs() (RTB method), 128
getElement() (Atom method), 34
ggetElements() (AtomGroup method), 40
ggetElements() (AtomMap method), 47
ggetElements() (AtomSubset method), 91
ggetElements() (Chain method), 50
ggetElements() (Residue method), 70
ggetElements() (Segment method), 75
ggetElements() (Selection method), 87
ggetEnsemble() (Conformation method), 131
getEnsemble() (PDBConformation method), 132
getFilename() (DCDFile method), 175
getFilename() (MSAFile method), 171
getFilename() (TrajFile method), 183
getFilenames() (Trajectory method), 181
getFilter() (MSAFile method), 171
getFirstTimestep() (DCDFile method), 175
getFirstTimestep() (Trajectory method), 181
getFirstTimestep() (TrajFile method), 183
ggetFlag() (Atom method), 34
ggetFlagLabels() (Atom method), 34
ggetFlagLabels() (AtomGroup method), 40
ggetFlagLabels() (AtomMap method), 47
ggetFlagLabels() (AtomPointer method), 68
ggetFlagLabels() (AtomSubset method), 91
ggetFlagLabels() (Chain method), 50
ggetFlagLabels() (Residue method), 70
ggetFlagLabels() (Segment method), 75
ggetFlagLabels() (Selection method), 87
ggetFlags() (AtomGroup method), 40
ggetFlags() (AtomMap method), 47
ggetFlags() (AtomSubset method), 91
ggetFlags() (Chain method), 51
ggetFlags() (Residue method), 70
ggetFlags() (Segment method), 75
ggetFlags() (Selection method), 87
ggetFormat() (MSAFile method), 171
ggetFragindex() (Atom method), 34
ggetFragindices() (AtomGroup method), 40
ggetFragindices() (AtomMap method), 47
ggetFragindices() (AtomSubset method), 91
ggetFragindices() (Chain method), 51
ggetFragindices() (Residue method), 70
ggetFragindices() (Segment method), 75
ggetFragindices() (Selection method), 87
ggetFrame() (DCDFile method), 175
ggetFrame() (TrajBase method), 179
ggetFrame() (TrajFile method), 183
ggetFrameFreq() (DCDFile method), 175
ggetFrameFreq() (Trajectory method), 181
ggetFrameFreq() (TrajFile method), 183
ggetGamma() (ANM method), 102
ggetGamma() (GammaVariableCutoff method), 112
ggetGamma() (GNM method), 113
ggetGapExtPenalty() (in module prody.proteins.compare), 152
ggetGapPenalty() (in module prody.proteins.compare), 151
ggetHessian() (PackageLogger method), 188
ggetHessian() (ANM method), 102
ggetHessian() (RTB method), 128
ggetHierView() (AtomGroup method), 40
ggetHierView() (Selection method), 87
ggetHits() (PDBBlastRecord method), 149
ggetIcode() (Atom method), 34
ggetIcode() (Residue method), 70
ggetIcode() (AtomGroup method), 40
ggetIcode() (AtomMap method), 47
ggetIcode() (AtomSubset method), 92
ggetIcode() (Chain method), 51
ggetIcode() (Residue method), 70
ggetIcode() (Segment method), 75
ggetIcode() (Selection method), 87
ggetIndex() (Atom method), 34
ggetIndex() (Conformation method), 131
ggetIndex() (Frame method), 178
ggetIndex() (Mode method), 115
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Module/Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>getIndex() (MSA method)</td>
<td>prody.proteins.compare</td>
</tr>
<tr>
<td>getIndex() (PDBConformation method)</td>
<td></td>
</tr>
<tr>
<td>getIndex() (Sequence method)</td>
<td></td>
</tr>
<tr>
<td>getIndices() (Atom method)</td>
<td></td>
</tr>
<tr>
<td>getIndices() (AtomMap method)</td>
<td></td>
</tr>
<tr>
<td>getIndices() (AtomSubset method)</td>
<td></td>
</tr>
<tr>
<td>getIndices() (Bond method)</td>
<td></td>
</tr>
<tr>
<td>getIndices() (Chain method)</td>
<td></td>
</tr>
<tr>
<td>getIndices() (KDTree method)</td>
<td></td>
</tr>
<tr>
<td>getIndices() (ModeSet method)</td>
<td></td>
</tr>
<tr>
<td>getIndices() (Residue method)</td>
<td></td>
</tr>
<tr>
<td>getIndices() (Segment method)</td>
<td></td>
</tr>
<tr>
<td>getIndices() (Select method)</td>
<td></td>
</tr>
<tr>
<td>getIndices() (Selection method)</td>
<td></td>
</tr>
<tr>
<td>getKirchhoff() (ANM method)</td>
<td></td>
</tr>
<tr>
<td>getKirchhoff() (GNM method)</td>
<td></td>
</tr>
<tr>
<td>getLabel() (MSA method)</td>
<td></td>
</tr>
<tr>
<td>getLabel() (PDBConformation method)</td>
<td></td>
</tr>
<tr>
<td>getLabel() (Sequence method)</td>
<td></td>
</tr>
<tr>
<td>getLabels() (PDBEnsemble method)</td>
<td></td>
</tr>
<tr>
<td>getLength() (Bond method)</td>
<td></td>
</tr>
<tr>
<td>getLinked() (DCDFile method)</td>
<td></td>
</tr>
<tr>
<td>getLinked() (TrajBase method)</td>
<td></td>
</tr>
<tr>
<td>getLinked() (TrajFile method)</td>
<td></td>
</tr>
<tr>
<td>getMapping() (AtomMap method)</td>
<td></td>
</tr>
<tr>
<td>getMass() (Atom method)</td>
<td></td>
</tr>
<tr>
<td>getMasses() (AtomGroup method)</td>
<td></td>
</tr>
<tr>
<td>getMasses() (AtomMap method)</td>
<td></td>
</tr>
<tr>
<td>getMasses() (AtomSubset method)</td>
<td></td>
</tr>
<tr>
<td>getMasses() (Chain method)</td>
<td></td>
</tr>
<tr>
<td>getMasses() (Residue method)</td>
<td></td>
</tr>
<tr>
<td>getMasses() (Segment method)</td>
<td></td>
</tr>
<tr>
<td>getMasses() (Selection method)</td>
<td></td>
</tr>
<tr>
<td>getMatchScore()</td>
<td></td>
</tr>
<tr>
<td>getMatrix() (Transformation method)</td>
<td></td>
</tr>
<tr>
<td>getMismatchScore()</td>
<td></td>
</tr>
<tr>
<td>getMSA() (Sequence method)</td>
<td></td>
</tr>
<tr>
<td>getMSFs() (Ensemble method)</td>
<td></td>
</tr>
<tr>
<td>getMSFs() (PDBEnsemble method)</td>
<td></td>
</tr>
<tr>
<td>getNonstdProperties()</td>
<td></td>
</tr>
<tr>
<td>getOccupancies() (AtomGroup method)</td>
<td></td>
</tr>
<tr>
<td>getOccupancies() (AtomMap method)</td>
<td></td>
</tr>
<tr>
<td>getOccupancies() (AtomSubset method)</td>
<td></td>
</tr>
<tr>
<td>getOccupancies() (Chain method)</td>
<td></td>
</tr>
<tr>
<td>getOccupancies() (Residue method)</td>
<td></td>
</tr>
<tr>
<td>getOccupancies() (Segment method)</td>
<td></td>
</tr>
<tr>
<td>getOccupancies() (Selection method)</td>
<td></td>
</tr>
<tr>
<td>getPackagePath()</td>
<td></td>
</tr>
<tr>
<td>getPrev() (Residue method)</td>
<td></td>
</tr>
<tr>
<td>getProjection() (RTB method)</td>
<td></td>
</tr>
<tr>
<td>getRadii() (AtomGroup method)</td>
<td></td>
</tr>
<tr>
<td>getRadii() (AtomMap method)</td>
<td></td>
</tr>
<tr>
<td>getRadii() (AtomSubset method)</td>
<td></td>
</tr>
<tr>
<td>getRadii() (Chain method)</td>
<td></td>
</tr>
<tr>
<td>getRadii() (GammaVariableCutoff method)</td>
<td></td>
</tr>
<tr>
<td>getRadii() (Residue method)</td>
<td></td>
</tr>
<tr>
<td>getRadii() (Segment method)</td>
<td></td>
</tr>
<tr>
<td>getRadii() (Selection method)</td>
<td></td>
</tr>
<tr>
<td>getResidue() (Chain method)</td>
<td></td>
</tr>
<tr>
<td>getResidue() (HierView method)</td>
<td></td>
</tr>
<tr>
<td>getResindex() (Atom method)</td>
<td></td>
</tr>
<tr>
<td>getResindex() (Residue method)</td>
<td></td>
</tr>
<tr>
<td>getResindices() (AtomGroup method)</td>
<td></td>
</tr>
<tr>
<td>getResindices() (AtomMap method)</td>
<td></td>
</tr>
<tr>
<td>getResindices() (AtomSubset method)</td>
<td></td>
</tr>
<tr>
<td>getResindices() (Chain method)</td>
<td></td>
</tr>
<tr>
<td>getResindices() (Residue method)</td>
<td></td>
</tr>
<tr>
<td>getResindices() (Segment method)</td>
<td></td>
</tr>
<tr>
<td>getResindices() (Selection method)</td>
<td></td>
</tr>
<tr>
<td>getResnames() (AtomGroup method)</td>
<td></td>
</tr>
<tr>
<td>getResnames() (AtomMap method)</td>
<td></td>
</tr>
<tr>
<td>getResnames() (AtomSubset method)</td>
<td></td>
</tr>
<tr>
<td>getResnames() (Chain method)</td>
<td></td>
</tr>
<tr>
<td>getResnames() (Residue method)</td>
<td></td>
</tr>
<tr>
<td>getResnames() (Segment method)</td>
<td></td>
</tr>
<tr>
<td>getResnames() (Selection method)</td>
<td></td>
</tr>
<tr>
<td>getResnum() (Atom method)</td>
<td></td>
</tr>
<tr>
<td>getResnum() (Residue method)</td>
<td></td>
</tr>
<tr>
<td>getResnums() (AtomGroup method)</td>
<td></td>
</tr>
<tr>
<td>getResnums() (AtomMap method)</td>
<td></td>
</tr>
<tr>
<td>getResnums() (AtomSubset method)</td>
<td></td>
</tr>
<tr>
<td>getResnums() (Chain method)</td>
<td></td>
</tr>
<tr>
<td>getResnums() (Residue method)</td>
<td></td>
</tr>
<tr>
<td>getResnums() (Segment method)</td>
<td></td>
</tr>
<tr>
<td>getResnums() (Selection method)</td>
<td></td>
</tr>
<tr>
<td>getResnames() (Atom method)</td>
<td></td>
</tr>
<tr>
<td>getResnames() (Residue method)</td>
<td></td>
</tr>
<tr>
<td>getResnames() (Selection method)</td>
<td></td>
</tr>
</tbody>
</table>

Index: 270
getVariances() (NMA method), 117
getVariances() (PCA method), 121
getVariances() (RTB method), 128
getVector() (Bond method), 49
getVelocities() (Frame method), 178
getVMDpath() (in module prody.dynamics.nmdfile), 119
gerWeights() (Conformation method), 131
gerWeights() (DCDFile method), 175
gerWeights() (Ensemble method), 133
gerWeights() (Frame method), 178
gerWeights() (PDBConformation method), 132
gerWeights() (PDBEnsemble method), 136
gerWeights() (TrajBase method), 179
gerWeights() (Trajectory method), 181
gerWeights() (TrajFile method), 183
glob() (in module prody.utilities.pathtools), 191
GNM (class in prody.dynamics.gnm), 112
goto() (DCDFile method), 175
goto() (TrajBase method), 179
goto() (Trajectory method), 181
goto() (TrajFile method), 183
gunzip() (in module prody.utilities.pathtools), 190
H
hasUnitcell() (DCDFile method), 176
hasUnitcell() (TrajBase method), 179
hasUnitcell() (Trajectory method), 181
hasUnitcell() (TrajFile method), 184
heavy, 62
helix, 63
helix310, 63
helixpi, 63
heme, 62
hetatm, 60
hetero, 60
HierView (class in prody.atomic.hierview), 67
HOME, 222
hydrogen, 62
hydrophobic, 58
I
icode, 54
icode (Chemical attribute), 154
idcode (DBRef attribute), 157
importLA() (in module prody.utilities.misctools), 190
info() (PackageLogger method), 188
intorfloat() (in module prody.utilities.misctools), 190
ion, 60
is3d() (ANM method), 102
is3d() (EDA method), 123
is3d() (GNM method), 113
is3d() (Mode method), 115
is3d() (ModeSet method), 117
is3d() (NMA method), 117
is3d() (PCA method), 121
is3d() (RTB method), 128
is3d() (Vector method), 116
isAligned() (MSA method), 170
isAligned() (MSAFile method), 172
isDataLabel() (Atom method), 35
isDataLabel() (AtomGroup method), 41
isDataLabel() (AtomMap method), 48
isDataLabel() (AtomPointer method), 68
isDataLabel() (AtomSubset method), 92
isDataLabel() (Chain method), 52
isDataLabel() (Residue method), 72
isDataLabel() (Segment method), 76
isDataLabel() (Selection method), 88
isExecutable() (in module prody.utilities.pathtools), 191
isFlagLabel() (Atom method), 36
isFlagLabel() (AtomGroup method), 41
isFlagLabel() (AtomMap method), 48
isFlagLabel() (AtomPointer method), 68
isFlagLabel() (AtomSubset method), 93
isFlagLabel() (Chain method), 52
isFlagLabel() (Residue method), 72
isFlagLabel() (Segment method), 76
isFlagLabel() (Selection method), 88
isLinked() (DCDFile method), 176
isLinked() (TrajBase method), 179
isLinked() (Trajectory method), 181
isLinked() (TrajFile method), 184
isReadable() (in module prody.utilities.pathtools), 191
isReserved() (in module prody.atomic.functions), 66
isSelectionMacro() (in module prody.atomic.select), 86
isWritable() (in module prody.utilities.pathtools), 191
iterAtoms() (Atom method), 36
iterAtoms() (AtomGroup method), 41
iterAtoms() (AtomMap method), 48
iterAtoms() (AtomSubset method), 93
iterAtoms() (Chain method), 52
iterAtoms() (Residue method), 72
iterAtoms() (Segment method), 76
iterAtoms() (Selection method), 88
iterBonded() (Atom method), 36
iterBonds() (Atom method), 36
iterBonds() (AtomGroup method), 41
iterChains() (AtomGroup method), 41
iterChains() (HierView method), 67
iterChains() (Segment method), 76
iterCoordsets() (Atom method), 36
iterCoordsets() (AtomGroup method), 41

Index
Index

iterCoordsets() (AtomMap method), 48
iterCoordsets() (AtomSubset method), 93
iterCoordsets() (Chain method), 52
iterCoordsets() (DCDFile method), 176
iterCoordsets() (Ensemble method), 133
iterCoordsets() (PDBEnsemble method), 136
iterCoordsets() (Residue method), 72
iterCoordsets() (Selection method), 88
iterCoordsets() (TrajBase method), 179
iterCoordsets() (Trajectory method), 181
iterCoordsets() (TrajFile method), 184
iterFragments() (AtomGroup method), 41
iterFragments() (in module prody.atomic.functions), 66
iterLabels() (MSA method), 170
iterNeighbors() (in module prody.measure.contacts), 140
iterPDBFilenames() (in module prody.proteins.localpdb), 160
iterPose() (Ensemble method), 133
iterPose() (PDBEnsemble method), 136
iterResidues() (AtomGroup method), 41
iterResidues() (Chain method), 52
iterResidues() (HierView method), 67
iterSegments() (AtomGroup method), 41
iterSegments() (HierView method), 67
joinLinks() (in module prody.utilities.doctools), 187
joinRepr() (in module prody.utilities.doctools), 187
joinTerms() (in module prody.utilities.doctools), 187
K
KDTree (class in prody.kdtree.kdtree), 137
L
large, 58
last (DBRef attribute), 157
link() (DCDFile method), 176
link() (TrajBase method), 179
link() (Trajectory method), 181
link() (TrajFile method), 184
lipid, 62
listNonstdAAProps() (in module prody.atomic.flags), 65
listPDBClusters() (in module prody.proteins.pdbclusters), 161
listReservedWords() (in module prody.atomic.functions), 66
load() (PackageSettings method), 192
loadAtoms() (in module prody.atomic.functions), 66
loadModel() (in module prody.dynamics.functions), 107
loadPDBClusters() (in module prody.proteins.pdbclusters), 161
loadVector() (in module prody.dynamics.functions), 107
makePath() (in module prody.utilities.pathtools), 191
mapOntoChain() (in module prody.proteins.compare), 151
mapped, 63
matchAlign() (in module prody.proteins.compare), 150
matchChains() (in module prody.proteins.compare), 149
medium, 59
mergeMSA() (in module prody.sequence.msa), 171
meth (Field attribute), 56
meth_pl (Field attribute), 56
Mode (class in prody.dynamics.mode), 115
ModeSet (class in prody.dynamics.modeset), 116
modified (Polymer attribute), 156
moveAtoms() (in module prody.measure.transform), 144
MSA (class in prody.sequence.msa), 169
MSAFile (class in prody.sequence.msafile), 171
mutation (Polymer attribute), 156
N
ame, 55
name (Chemical attribute), 154
name (Field attribute), 56
name (Polymer attribute), 156
natoms (Chemical attribute), 154
ndim (Field attribute), 56
neutral, 59
next() (DCDFile method), 176
next() (TrajBase method), 180
next() (Trajectory method), 182
next() (TrajFile method), 184
nextCoordset() (DCDFile method), 176
nextCoordset() (TrajBase method), 180
nextCoordset() (Trajectory method), 182
nextCoordset() (TrajFile method), 184
nextIndex() (DCDFile method), 176
nextIndex() (TrajBase method), 180
nextIndex() (Trajectory method), 182
nextIndex() (TrajFile method), 184
nitrogen, 62
NMA (class in prody.dynamics.nma), 117
noh, 62
none, 63
none (Field attribute), 56
none (Field attribute), 56
nonstdaa, 57
nucleic, 59
nucleobase, 59
nucleoside, 59
nucleotide, 59
numAtoms() (ANM method), 102
numAtoms() (Atom method), 36
numAtoms() (AtomGroup method), 42
numAtoms() (AtomMap method), 48
numAtoms() (AtomSubset method), 93
numAtoms() (Chain method), 52
numAtoms() (Conformation method), 131
numAtoms() (DCDFile method), 176
numAtoms() (EDA method), 123
numAtoms() (Ensemble method), 133
numAtoms() (Frame method), 178
numAtoms() (GNM method), 113
numAtoms() (Mode method), 115
numAtoms() (ModeSet method), 117
numAtoms() (NMA method), 117
numAtoms() (PCA method), 121
numAtoms() (PDBConformation method), 132
numAtoms() (PDBEnsemble method), 136
numAtoms() (Residue method), 72
numAtoms() (RTB method), 128
numAtoms() (Segment method), 76
numAtoms() (Selection method), 89
numAtoms() (TrajBase method), 180
numAtoms() (Trajectory method), 182
numAtoms() (TrajFile method), 184
numDOF() (ANM method), 102
numDOF() (EDA method), 123
numDOF() (GNM method), 113
numDOF() (Mode method), 115
numDOF() (ModeSet method), 117
numDOF() (PCA method), 121
numDOF() (RTB method), 128
numDOF() (Vector method), 116
numDummies() (AtomMap method), 48
numFiles() (Trajectory method), 182
numFixed() (DCDFile method), 176
numFixed() (Trajectory method), 182
numFixed() (TrajFile method), 184
numFrames() (AtomGroup method), 42
numFrames() (DCDFile method), 176
numFrames() (TrajBase method), 180
numFrames() (Trajectory method), 182
numFrames() (TrajFile method), 184
numGaps() (Sequence method), 174
numIndexed() (MSA method), 170
numMapped() (AtomMap method), 48
numModes() (ANM method), 102
numModes() (EDA method), 123
numModes() (GNM method), 113
numModes() (Mode method), 115
numModes() (ModeSet method), 117
numModes() (NMA method), 118
numModes() (PCA method), 121
numModes() (RTB method), 128
numModes() (Vector method), 116
numResidues() (AtomGroup method), 42
numResidues() (Chain method), 52
numResidues() (HierView method), 68
numResidues() (MSA method), 170
numResidues() (Sequence method), 174
numSegments() (AtomGroup method), 42
numSegments() (HierView method), 68
numSelected() (Conformation method), 131
numSelected() (DCDFile method), 176
numSelected() (Ensemble method), 133
numSelected() (Frame method), 178
numSelected() (PDBConformation method), 132
numSelected() (PDBEnsemble method), 136
numSelected() (TrajBase method), 180
numSelected() (Trajectory method), 182
numSelected() (TrajFile method), 184
numSequences() (MSA method), 170
O
occupancy, 55
openDB() (in module prody.utilities.pathtools), 191
openFile() (in module prody.utilities.pathtools), 190
openSQLite() (in module prody.utilities.pathtools), 191
openURL() (in module prody.utilities.pathtools), 191
oxygen, 62
PackageLogger (class in prody.utilities.logger), 187
PackageSettings (class in prody.utilities.settings), 191
parseArray() (in module prody.dynamics.functions), 105
parseDCD() (in module prody.trajectory.dcdfile), 177
parseDSSP() (in module prody.proteins.dssp), 152
parseHeatmap() (in module prody.dynamics.heatmapper), 114
parseModes() (in module prody.dynamics.functions), 105
parseMSA() (in module prody.sequence.msafile), 172
parseNMD() (in module prody.dynamics.nmdfile), 119
parsePDB() (in module prody.proteins.pdbfile), 162
parsePDBHeader() (in module prody.proteins.header), 157
parsePDBStream() (in module prody.proteins.pdbfile), 161
parsePQR() (in module prody.proteins.pdbfile), 163
parsePSF() (in module prody.trajectory.psffile), 178
parseSparseMatrix() (in module prody.dynamics.functions), 106
parseSTRIDE() (in module prody.proteins.stride), 165
PATH, 2, 29, 30, 207
pathPDBFolder() (in module prody.proteins.localpdb), 160
pathPDBMirror() (in module prody.proteins.localpdb), 160
pathVMD() (in module prody.dynamics.nmdfile), 119
PCA (class in prody.dynamics.pca), 120
PDBBlastRecord (class in prody.proteins.blastpdb), 148
PDBConformation (class in prody.ensemble.conformation), 131
PDBEnsemble (class in prody.ensemble.pdbensemble), 135
pdbentry (Chemical attribute), 154
pdbentry (Polymer attribute), 156
pdbeter, 62
performDSSP() (in module prody.proteins.dssp), 153
performSTRIDE() (in module prody.proteins.stride), 165
performSVD() (EDA method), 123
performSVD() (PCA method), 121
pickCentral() (in module prody.measure.measure), 143
pickCentralAtom() (in module prody.measure.measure), 143
pickCentralConf() (in module prody.measure.measure), 143
plog() (in module prody), 205
polar, 59
Polymer (class in prody.proteins.header), 154
pop() (PackageSettings method), 192
prefix (PackageLogger attribute), 189
printOverlapTable() (in module prody.dynamics.compare), 103
printRMSD() (in module prody.measure.transform), 146
private (Field attribute), 56
prody (module), 204
prody.apps (module), 192
prody.apps.evol_apps.evol_coevol (module), 193
prody.apps.evol_apps.evol_conserv (module), 193
prody.apps.evol_apps.evol_fetch (module), 194
prody.apps.evol_apps.evol_filter (module), 195
prody.apps.evol_apps.evol_merge (module), 195
prody.apps.evol_apps.evol_occupancy (module), 196
prody.apps.evol_apps.evol_rankorder (module), 196
prody.apps.evol_apps.evol_refine (module), 197
prody.apps.evol_apps.evol_search (module), 198
prody.apps.prody_apps.prody_align (module), 198
prody.apps.prody_apps.prody_anm (module), 199
prody.apps.prody_apps.prody_biomol (module), 200
prody.apps.prody_apps.prody_blast (module), 200
prody.apps.prody_apps.prody_catdcd (module), 201
prody.apps.prody_apps.prody_contacts (module), 201
prody.apps.prody_apps.prody_fetch (module), 202
prody.apps.prody_apps.prody_gnm (module), 202
prody.apps.prody_apps.prody_pca (module), 203
prody.apps.prody_apps.prody_select (module), 204
prody.atomic (module), 31
prody.atomic.atom (module), 33
prody.atomic.atomgroup (module), 37
prody.atomic.atomic (module), 44
prody.atomic.atommap (module), 44
prody.atomic.bond (module), 49
prody.atomic.chain (module), 49

PEP 8#imports, 212
PEP 8#whitespace-in-expressions-and-statements, 212
PYTHONPATH, 207
R
radius, 55
rangeString() (in module prody.utilities.miscools), 189
readonly (Field attribute), 56
reduceModel() (in module prody.dynamics.editing), 105
refineMSA() (in module prody.sequence.msa), 170
relpath() (in module prody.utilities.pathtools), 191
report() (PackageLogger method), 188
reset() (DCDFile method), 176
reset() (MSAFile method), 172
reset() (TrajBase method), 180
reset() (TrajFile method), 184
resetTicks() (in module prody.dynamics.plotting), 126
resid, 55
Residue (class in prody.atomic.residue), 69
resindex, 55
resname, 55
resname (Chemical attribute), 154
resnum, 55
resnum (Chemical attribute), 154
RTB (class in prody.dynamics.rtb), 127
S
sampleModes() (in module prody.dynamics.sampling), 128
save() (PackageSettings method), 192
saveAtoms() (in module prody.atomic.functions), 66
saveEnsemble() (in module prody.ensemble.functions), 134
saveModel() (in module prody.dynamics.functions), 107
saveVector() (in module prody.dynamics.functions), 107
sc, 58
search() (KDTree method), 138
searchPfam() (in module prody.database.pfam), 95
secondary, 55
secstr, 55
segindex, 55
segment, 55
Segment (class in prody.atomic.segment), 74
segname, 55
Select (class in prody.atomic.select), 85
select() (Atom method), 85
select() (AtomGroup method), 42
select() (AtomMap method), 48
select() (AtomPointer method), 68
select() (AtomSubset method), 93
select() (Chain method), 52
select() (Contacts method), 140
select() (Residue method), 72
select() (Segment method), 77
select() (Select method), 85
select() (Selection method), 89
Selection (class in prody.atomic.selection), 86
SelectionError, 85
SelectionWarning, 85
selstr (Field attribute), 56
Sequence (class in prody.sequence.sequence), 173
sequence (Polymer attribute), 156
serial, 55
setACSIndex() (Atom method), 36
setACSIndex() (AtomGroup method), 42
setACSIndex() (AtomMap method), 48
setACSIndex() (AtomPointer method), 69
setACSIndex() (AtomSubset method), 93
setACSIndex() (Bond method), 49
setACSIndex() (Chain method), 52
setACSIndex() (Residue method), 72
setACSIndex() (Segment method), 77
setACSIndex() (Selection method), 89
setACSLabel() (AtomGroup method), 42
setAlignmentMethod() (in module prody.proteins.compare), 152
setAltloc() (Atom method), 36
setAltlocs() (AtomGroup method), 42
setAltlocs() (AtomSubset method), 93
setAltlocs() (Chain method), 52
setAltlocs() (Residue method), 72
setAltlocs() (Segment method), 77
setAltlocs() (Selection method), 89
setAnisou() (Atom method), 36
setAnisous() (AtomGroup method), 42
setAnisous() (AtomSubset method), 93
setAnisous() (Chain method), 53
setAnisous() (Residue method), 72
setAnisous() (Segment method), 77
setAnisous() (Selection method), 89
setAnistd() (Atom method), 36
setAnistds() (AtomGroup method), 42
setAnistds() (AtomSubset method), 93
setAnistds() (Chain method), 53
setAnistds() (Residue method), 72
setAnistds() (Segment method), 77
setAnistds() (Selection method), 89
setAtoms() (DCDFile method), 176
setAtoms() (Ensemble method), 133
setAtoms() (PDBEnsemble method), 136

Index 277
setAtoms() (TrajBase method), 180
setAtoms() (Trajectory method), 182
setAtoms() (TrajFile method), 184
setBeta() (Atom method), 36
setBetas() (AtomGroup method), 42
setBetas() (AtomSubset method), 93
setBetas() (Chain method), 53
setBetas() (Residue method), 72
setBetas() (Segment method), 77
setBetas() (Selection method), 89
setBonds() (AtomGroup method), 42
setCharge() (Atom method), 36
setCharges() (AtomGroup method), 43
setCharges() (AtomSubset method), 93
setCharges() (Chain method), 53
setCharges() (Residue method), 72
setCharges() (Segment method), 77
setCharges() (Selection method), 89
setChid() (Atom method), 36
setChid() (Chain method), 53
setChids() (AtomGroup method), 43
setChids() (AtomSubset method), 93
setChids() (Chain method), 53
setChids() (Residue method), 72
setChids() (Segment method), 77
setChids() (Selection method), 89
setCoords() (Atom method), 36
setCoords() (AtomGroup method), 43
setCoords() (AtomMap method), 48
setCoords() (AtomSubset method), 93
setCoords() (Chain method), 53
setCoords() (DCDFile method), 177
setCoords() (Ensemble method), 134
setCoords() (PDBEnsemble method), 137
setCoords() (Residue method), 72
setCoords() (Segment method), 77
setCoords() (Selection method), 89
setCoords() (TrajBase method), 180
setCoords() (Trajectory method), 182
setCovariance() (EDA method), 123
setCovariance() (PCA method), 121
setCSLabels() (AtomGroup method), 42
setData() (Atom method), 36
setData() (AtomGroup method), 43
setData() (AtomSubset method), 93
setData() (Chain method), 53
setData() (DCDFile method), 177
setData() (Ensemble method), 134
setData() (PDBEnsemble method), 137
setData() (Residue method), 72
setData() (Segment method), 77
setData() (Selection method), 89
setData() (TrajBase method), 180
setData() (Trajectory method), 182
setEigens() (PCA method), 121
setEigens() (RTB method), 128
setElement() (Atom method), 37
setElements() (AtomGroup method), 43
setElements() (AtomSubset method), 93
setElements() (Chain method), 53
setElements() (Residue method), 73
setElements() (Segment method), 77
setElements() (Selection method), 89
setFlag() (Atom method), 37
setFlags() (AtomGroup method), 43
setFlags() (AtomSubset method), 93
setFlags() (Chain method), 53
setFlags() (Residue method), 73
setFlags() (Segment method), 77
setFlags() (Selection method), 89
setGapExtPenalty() (in module prody.proteins.compare), 152
setGapPenalty() (in module prody.proteins.compare), 151
setHessian() (ANM method), 103
setHessian() (RTB method), 128
setIcode() (Atom method), 37
setIcode() (Residue method), 73
setIcodes() (AtomGroup method), 43
setIcodes() (AtomSubset method), 93
setIcodes() (Chain method), 53
setIcodes() (Residue method), 73
setIcodes() (Segment method), 77
setIcodes() (Selection method), 89
setKirchhoff() (GNM method), 114
setLabel() (PDBConformation method), 132
setMass() (Atom method), 37
setMasses() (AtomGroup method), 43
setMasses() (AtomSubset method), 94
setMasses() (Chain method), 53
setMasses() (Residue method), 73
setMasses() (Segment method), 77
setMasses() (Selection method), 89
setMatchScore() (in module prody.proteins.compare), 151
setMismatchScore() (in module prody.proteins.compare), 151
setName() (Atom method), 37
setName() (AtomGroup method), 43
setName() (AtomSubset method), 94
setName() (Chain method), 53
setName() (Residue method), 73
setName() (Segment method), 77
setName() (Selection method), 89
setOccupancies() (AtomGroup method), 43
setOccupancies() (AtomSubset method), 94
setOccupancies() (Chain method), 53
setOccupancies() (Residue method), 73
setOccupancies() (Segment method), 77
setOccupancies() (Selection method), 89
setOccupancy() (Atom method), 37
setPackagePath() (in module prody.utilities.settings), 192
setRadii() (AtomGroup method), 43
setRadii() (AtomSubset method), 94
setRadii() (Chain method), 53
setRadii() (Residue method), 73
setRadii() (Segment method), 77
setRadii() (Selection method), 90
setRadius() (Atom method), 37
setResname() (Atom method), 37
setResname() (Residue method), 73
setResnames() (AtomGroup method), 43
setResnames() (AtomSubset method), 94
setResnames() (Chain method), 53
setResnames() (Residue method), 73
setResnames() (Segment method), 78
setResnames() (Selection method), 90
setResnum() (Atom method), 37
setResnum() (Residue method), 73
setResnums() (AtomGroup method), 43
setResnums() (AtomSubset method), 94
setResnums() (Chain method), 53
setResnums() (Residue method), 73
setResnums() (Segment method), 78
setResnums() (Selection method), 90
setRotation() (Transformation method), 144
setSecstrs() (AtomGroup method), 37
setSecstrs() (AtomSubset method), 94
setSecstrs() (Chain method), 54
setSecstrs() (Residue method), 73
setSecstrs() (Segment method), 78
setSecstrs() (Selection method), 90
setSegname() (Atom method), 37
setSegname() (Residue method), 73
setSegnames() (AtomGroup method), 44
setSegnames() (AtomSubset method), 94
setSegnames() (Chain method), 54
setSegnames() (Residue method), 73
setSegnames() (Segment method), 78
setSegnames() (Selection method), 90
setSerial() (Atom method), 37
setSerials() (AtomGroup method), 44
setSerials() (AtomSubset method), 94
setSerials() (Chain method), 54
setSerials() (Residue method), 73
setSerials() (Segment method), 78
setSerials() (Selection method), 90
setSlice() (MSAFile method), 172
setTitle() (ANM method), 48
setTitle() (AtomGroup method), 44
setTitle() (AtomMap method), 48
setTitle() (DCDFile method), 177
setTitle() (EDA method), 123
setTitle() (Ensemble method), 134
setTitle() (GNM method), 114
setTitle() (MSA method), 170
setTitle() (MSAFile method), 172
setTitle() (NMA method), 118
setTitle() (PCA method), 122
setTitle() (PDBEnsemble method), 137
setTitle() (RTB method), 128
setTitle() (TrajBase method), 180
setTitle() (Trajectory method), 182
setTitle() (TrajFile method), 185
setTitle() (Vector method), 116
setTranslation() (Transformation method), 144
setType() (Atom method), 37
setTypes() (AtomGroup method), 44
setTypes() (AtomSubset method), 94
setTypes() (Chain method), 54
setTypes() (Residue method), 73
setTypes() (Segment method), 54
setTypes() (Selection method), 90
setVMDpath() (in module prody.dynamics.nmdfile), 119
setWeights() (DCDFile method), 177
setWeights() (Ensemble method), 134
setWeights() (PDBEnsemble method), 137
setWeights() (TrajBase method), 180
setWeights() (Trajectory method), 182
setWeights() (Triangular method), 185
showContactMap() (in module prody.dynamics.plotting), 124
showCrossCorr() (in module prody.dynamics.plotting), 124
showCrossProjection() (in module prody.dynamics.plotting), 125
showCumulFractVars() (in module prody.dynamics.plotting), 124
showCumulOverlap() (in module prody.dynamics.plotting), 124
showDiffMatrix() (in module prody.dynamics.plotting), 126
showDirectInfoMatrix() (in module prody.sequence.plotting), 173
showEllipsoid() (in module prody.dynamics.plotting), 126
showFigure() (in module prody.utilities.misctools), 190
showFractVars() (in module prody.dynamics.plotting), 124
showHeatmap() (in module prody.dynamics.heatmapper), 114
showMode() (in module prody.dynamics.plotting), 124
showMSAOccupancy() (in module prody.sequence.plotting), 172
showMutinfoMatrix() (in module prody.sequence.plotting), 173
showNormedSqFlucts() (in module prody.dynamics.plotting), 126
showOccupancies() (in module prody.ensemble.functions), 134
showOverlap() (in module prody.dynamics.plotting), 124
showOverlapTable() (in module prody.dynamics.plotting), 124
showProjection() (in module prody.dynamics.plotting), 124
showProtein() (in module prody.proteins.functions), 153
showScaledSqFlucts() (in module prody.dynamics.plotting), 126
showSCAMatrix() (in module prody.sequence.plotting), 173
showShannonEntropy() (in module prody.sequence.plotting), 172
showSqFlucts() (in module prody.dynamics.plotting), 126
sliceMode() (in module prody.dynamics.editing), 104
sliceModel() (in module prody.dynamics.editing), 104
sliceVector() (in module prody.dynamics.editing), 104
small, 59
sortAtoms() (in module prody.atomic.functions), 67
split (MSA attribute), 170
splitSeqLabel() (in module prody.sequence.msafile), 172
start() (PackageLogger method), 188
startLogfile() (in module prody), 205
startwith() (in module prody.proteins.functions), 190
stdout, 58
stdaa, 57
sugar, 62
sulfur, 62
superpose() (Ensemble method), 134
superpose() (Frame method), 178
superpose() (in module prody.measure.transform), 144
superpose() (PDB Ensemble method), 137
surface, 59
sympath() (in module prody.utilities.pathtools), 191
synonym (Field attribute), 56
synonyms (Chemical attribute), 154
synonyms (Polymer attribute), 156
T
tabulate() (in module prody.utilities.doctools), 187
timeit() (PackageLogger method), 188
timing() (PackageLogger method), 188
TrajBase (class in prody.trajectory.trajbase), 179
Trajectory (class in prody.trajectory.trajectory), 181
TrajFile (class in prody.trajectory.trajfile), 183
Transformation (class in prody.measure.transform), 143
turn, 63
type, 55
U
uniqueSequences() (in module prody.sequence.analysis), 168
unpickle() (in module prody.utilities.pathtools), 191
update() (HierView method), 68
update() (PackageLogger method), 189
update() (PackageSettings method), 192
update() (Selection method), 90
V
Vector (class in prody.dynamics.mode), 115
verbosity (PackageLogger attribute), 189
viewNMDinVMD() (in module prody.dynamics.nmdfile), 120
W
warn() (PackageLogger method), 189
warning() (PackageLogger method), 189
water, 60
which() (in module prody.utilities.pathtools), 191
wrapAtoms() (in module prody.measure.transform), 145
wrapText() (in module prody.utilities.doctools), 187
write() (DCDFile method), 177
write() (MSAFile method), 172
write() (PackageLogger method), 189
writeArray() (in module prody.dynamics.functions), 107
writeDCD() (in module prody.trajecory.dcdfile), 177
writeHeatmap() (in module prody.dynamics.heatmap), 114
writeModes() (in module prody.dynamics.functions), 107
writeMSA() (in module prody.sequence.msafile), 172
writeNMD() (in module prody.dynamics.nmdfile), 119
writeOverlapTable() (in module prody.dynamics.compare), 103
writePDB() (in module prody.proteins.pdbfile), 163
writePDBStream() (in module prody.proteins.pdbfile), 163
writePQR() (in module prody.proteins.functions), 153
writePSF() (in module prody.trajectory.psffile), 179
wwPDBServer() (in module prody.proteins.wwpdb), 165