ProDy

Protein Dynamics & Sequence Analysis

ProDy Documentation

Release 1.10.0

Ahmet Bakan

Apr 30, 2018

Contents

CHAPTER 1

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 4), and *Evol Applications* (page 18) 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] http://www.python.org
[2] http://www.numpy.org
[3] http://matplotlib.org
[4] https://pypi.python.org/pypi/pip
[5] https://pypi.python.org/pypi/pip

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. For changes and list of new features see *Release Notes* (page 49).

Linux

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

```
$ 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 Modules[6].

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 4) and *Evol Applications* (page 18) in command prompt (**`cmd.exe`**), append Python and scripts folders (e.g. `C:\Python27` and `C:\Python27\Scripts`) to `PATH`[7] environment variable.

1.4 Recommended Software

- Scipy[8], when installed, replaces linear algebra module of Numpy. Scipy linear algebra module is more flexible and can be faster.
- IPython[9] is a must have for interactive ProDy sessions.
- PyReadline[10] for colorful IPython sessions on Windows.
- MDAnalysis[11] for reading molecular dynamics trajectories.

1.5 Included in ProDy

Following software is included in the ProDy installation packages:

- pyparsing[12] is used to define the atom selection grammar.
- Biopython[13] KDTree package and pairwise2 module are used for distance based atom selections and pairwise sequence alignment, respectively.

[6] http://docs.python.org/install/index.html
[7] https://matplotlib.org/faq/environment_variables_faq.html#envvar-PATH
[8] http://www.scipy.org
[9] http://ipython.org
[10] http://ipython.org/pyreadline.html
[11] http://code.google.com/p/mdanalysis
[12] http://pyparsing.wikispaces.com
[13] http://biopython.org

- argparse[14] 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.

[14] http://code.google.com/p/argparse/

CHAPTER 2

Applications

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:

```
usage: prody align [-h] [--quiet] [--examples] [-s SEL] [-m INT] [-i INT]
                   [-o INT] [-p STR] [-x STR]
                   pdb [pdb ...]

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):

  $ prody align 2k39

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

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

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

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

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

  $ prody align --select "resnum 1 to 70" 1aar 2k39

Align 1fi7 onto 1hrc using heme atoms:

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

2.1.2 prody anm

Usage

Running **`prody anm -h`** displays:

```
usage: prody anm [-h] [--quiet] [--examples] [-n INT] [-s SEL] [-c FLOAT]
                 [-g FLOAT] [-m INT] [-a] [-o PATH] [-e] [-r] [-u] [-q] [-v]
                 [-z] [-t STR] [-b] [-l] [-k] [-p STR] [-f STR] [-d STR]
                 [-x STR] [-A] [-R] [-Q] [-B] [-K] [-F STR] [-D INT]
                 [-W FLOAT] [-H FLOAT]
                 pdb

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 (A) (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:

  $ 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:

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

2.1.3 prody biomol

Usage

Running **`prody biomol -h`** displays:

```
usage: prody biomol [-h] [--quiet] [--examples] [-p STR] [-b INT] pdb

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:

  $ prody biomol 2bfu
```

2.1.4 prody blast

Usage

Running **prody blast -h** displays:

```
usage: prody blast [-h] [--quiet] [--examples] [-i FLOAT] [-o FLOAT] [-d PATH]
                   [-z] [-f STR] [-e FLOAT] [-l INT] [-s INT] [-t INT]
                   sequence

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␣
↪MQIFVKTLTGKTITLEVEPSDTIENVKAKIQDKEGIPPDQQRLIFAGKQLEDGRTLSDYNIQKESTLHLVLRLRGG

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

  $ prody blast -d .␣
↪ARKCSLTGKWTNDLGSNMTIGAVNSRGEFTGTYITAVTATSNEIKESPLHGTQNTINKRTQPTFGFTVNWKFSESTTVFT
```

```
  $ prody align 2avi.pdb *pdb
```

2.1.5 prody catdcd

Usage

Running **prody catdcd -h** displays:

```
usage: prody catdcd [-h] [--quiet] [--examples] [-s SEL] [-o FILE] [-n]
                    [--psf PSF] [--pdb PDB] [--first INT] [--last INT]
                    [--stride INT] [--align SEL]
                    dcd [dcd ...]

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 atmos:

  $ 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
  -r FLOAT, --radius FLOAT
                        contact radius (default: 4.0)
  -t STR, --extend STR  output same residue, chain, or segment as contacting
                        atoms
  -p STR, --prefix STR  output filename prefix (default: target filename)
  -x STR, --suffix STR  output filename suffix (default: _contacts)
```

Examples

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:

  $ 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:

```
usage: prody eda [-h] [--quiet] [--examples] [-n INT] [-s SEL] [-a] [-o PATH]
                 [-e] [-r] [-u] [-q] [-v] [-z] [-t STR] [-j] [-p STR] [-f STR]
                 [-d STR] [-x STR] [-A] [-R] [-Q] [-J STR] [-F STR] [-D INT]
                 [-W FLOAT] [-H FLOAT] [--psf PSF | --pdb PDB] [--aligned]
                 dcd

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:

```
usage: prody fetch [-h] [--quiet] [--examples] [-d PATH] [-z] pdb [pdb ...]

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:

```
usage: prody gnm [-h] [--quiet] [--examples] [-n INT] [-s SEL] [-c FLOAT]
                 [-g FLOAT] [-m INT] [-a] [-o PATH] [-e] [-r] [-u] [-q] [-v]
                 [-z] [-t STR] [-b] [-k] [-p STR] [-f STR] [-d STR] [-x STR]
                 [-A] [-R] [-Q] [-B] [-K] [-M STR] [-F STR] [-D INT]
                 [-W FLOAT] [-H FLOAT]
                 pdb

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 (A) (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:

```
usage: prody pca [-h] [--quiet] [--examples] [-n INT] [-s SEL] [-a] [-o PATH]
                 [-e] [-r] [-u] [-q] [-v] [-z] [-t STR] [-j] [-p STR] [-f STR]
                 [-d STR] [-x STR] [-A] [-R] [-Q] [-J STR] [-F STR] [-D INT]
                 [-W FLOAT] [-H FLOAT] [--psf PSF | --pdb PDB] [--aligned]
                 dcd

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 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:

  $ 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
```

2.1.11 prody select

Usage

Running **`prody select -h`** displays:

```
usage: prody select [-h] [--quiet] [--examples] [-o STR] [-p STR] [-x STR]
                    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:

  $ 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}
             ...

ProDy: A Python Package for Protein 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

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 struc-

ture, and will write eigenvalues/vectors in plain text and NMD Format[15]:

```
$ 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[16].

2.2 Evol Applications

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

2.2.1 evol coevol

Usage

Running **`evol coevol -h`** displays:

```
usage: evol coevol [-h] [--quiet] [--examples] [-n] [-c STR] [-m STR] [-t]
                   [-p STR] [-f STR] [-S] [-L FLOAT] [-U FLOAT] [-X STR]
                   [-T STR] [-D INT] [-H FLOAT] [-W FLOAT] [-F STR]
                   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
  -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
```

[15] http://prody.csb.pitt.edu/manual/reference/dynamics/nmdfile.html#nmd-format
[16] http://csb.pitt.edu/NMWiz

```
                        _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:

```
usage: evol conserv [-h] [--quiet] [--examples] [-n] [-g] [-p STR] [-f STR]
                    [-S] [-H FLOAT] [-W FLOAT] [-F STR] [-D INT]
                    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:
  -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
```

2.2.3 evol fetch

Usage

Running **`evol fetch -h`** displays:

```
usage: evol fetch [-h] [--quiet] [--examples] [-a STR] [-f STR] [-o STR]
                  [-i STR] [-g STR] [-t INT] [-d PATH] [-p STR] [-z]
                  acc

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:

```
usage: evol filter [-h] [--quiet] [--examples] (-s | -e | -c) [-F] [-o STR]
                   [-f STR] [-z]
                   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
```

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:

  $  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
```

2.2.5 evol merge

Usage

Running **`evol merge -h`** displays:

```
usage: evol merge [-h] [--quiet] [--examples] [-o STR] [-f STR] [-z]
                  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
  -f STR, --format STR  output MSA file format, default is same as first input
                        MSA
  -z, --compressed      gzip merged MSA output
```

Examples

Running **`evol merge --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
```

2.2.6 evol occupancy

Usage

Running **`evol occupancy -h`** displays:

```
usage: evol occupancy [-h] [--quiet] [--examples] [-o STR] [-p STR] [-l STR]
                      [-f STR] [-S] [-X STR] [-Y STR] [-T STR] [-D INT]
                      [-W FLOAT] [-F STR] [-H FLOAT]
                      msa

positional arguments:
  msa                   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:
  -o STR, --occ-axis STR
                        calculate row or column occupancy or both., one of
                        row, col, both (default: row)

output options:
  -p STR, --prefix STR  output filename prefix, default is msa filename with
                        _occupancy suffix
  -l STR, --label STR   index for column based on msa label
  -f STR, --number-format STR
                        number output format (default: %12g)

figure options:
  -S, --save-plot       save occupancy plot/s
  -X STR, --xlabel STR  specify xlabel
  -Y STR, --ylabel STR  specify ylabel
  -T STR, --title STR   figure title
  -D INT, --dpi INT     figure resolution (dpi) (default: 300)
  -W FLOAT, --width FLOAT
                        figure width (inch) (default: 8)
  -F STR, --figure-format STR
```

```
                        figure file format, one of png, pdf, raw, svg, eps,
                        ps, svgz, rgba (default: pdf)
  -H FLOAT, --height FLOAT
                        figure height (inch) (default: 6)
```

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.slx -S -c apc

Rank order analysis:

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

2.2.7 evol rankorder

Usage

Running **`evol rankorder -h`** displays:

```
usage: evol rankorder [-h] [--quiet] [--examples] [-z] [-d STR] [-p STR]
                      [-m STR] [-l STR] [-n INT] [-q INT] [-t FLOAT] [-u]
                      [-o STR]
                      mutinfo

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
```

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:

  $  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
```

2.2.8 evol refine

Usage

Running **`evol refine -h`** displays:

```
usage: evol refine [-h] [--quiet] [--examples] [-l STR] [-s FLOAT] [-c FLOAT]
                   [-r FLOAT] [-k] [-o STR] [-f STR] [-z]
                   msa

positional 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
  -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
```

2.2.9 evol search

Usage

Running **evol search -h** displays:

```
usage: evol search [-h] [--quiet] [--examples] [-b] [-s] [-g] [-e FLOAT]
                   [-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/colums
    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`[17] environment variable, e.g. `/usr/local/bin/`.

On Windows, installer places the scripts into the `Scripts` folder under Python distribution folder, e.g. `C:\Python27\Scripts`. You may need to add this path to `PATH`[18] environment variable yourself.

[17] https://matplotlib.org/faq/environment_variables_faq.html#envvar-PATH
[18] https://matplotlib.org/faq/environment_variables_faq.html#envvar-PATH

CHAPTER 3

Reference Manual

3.1 Atomic Data

3.1.1 Atom

3.1.2 Atom Group

3.1.3 Atomic Base

3.1.4 Atom Map

3.1.5 Bond

3.1.6 Chain

3.1.7 Atom Data Fields

3.1.8 Atom Flags

3.1.9 Supporting Functions

3.1.10 Hierarchical Views

3.1.11 Atom Pointer

3.1.12 Residue

3.1.13 Segment

3.1.14 Atom Selections

3.1.15 Selection

3.1.16 Atom Subsets

- confProDy()[19]
- checkUpdates()[20]
- startLogfile()[21]
- closeLogfile()[22]
- plog()[23]

[19] http://prody.csb.pitt.edu/manual/reference/prody.html#prody.confProDy
[20] http://prody.csb.pitt.edu/manual/reference/prody.html#prody.checkUpdates
[21] http://prody.csb.pitt.edu/manual/reference/prody.html#prody.startLogfile
[22] http://prody.csb.pitt.edu/manual/reference/prody.html#prody.closeLogfile
[23] http://prody.csb.pitt.edu/manual/reference/prody.html#prody.plog

CHAPTER 4

Developer's Guide

4.1 Contributing to ProDy

4.1.1 Install Git and a GUI

ProDy source code is managed using Git[24] 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[25].

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[26] and Git docs[27] for usage details and examples.

[24] http://git-scm.com/downloads
[25] http://git-scm.com/downloads
[26] http://git-scm.com/book
[27] http://git-scm.com/docs

4.1.2 Fork and Clone ProDy

ProDy source code an issue tracker are hosted on Github[28]. 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[29] or GitHub Windows[30] to help you manage a copy of the repository.

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[31] 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
```

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`[32] 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 4) from anywhere on your system. One way to do this by adding `ProDy/scripts` folder to your `PATH`[33] environment variable, e.g.:

```
export PATH=$PATH:$/home/USERNAME/path/to/ProDy/scripts
```

[28] http://github.com/prody/ProDy
[29] http://mac.github.com
[30] http://windows.github.com
[31] http://prody.csb.pitt.edu
[32] https://docs.python.org/3/using/cmdline.html#envvar-PYTHONPATH
[33] https://matplotlib.org/faq/environment_variables_faq.html#envvar-PATH

4.1.4 Modify, Test, and Commit

When modifying ProDy files you may want to follow the *Style Guide for ProDy* (page 40). 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 43) 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.

After ensuring that the package runs, you can commit your changes as follows:

```
$ git commit modified_file_1.py modified_file_2.py
```

or:

```
$ 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:

```
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[34] 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:

```
$ cd prody
$ git remote add prodymaster git@github.com:abakan/ProDy.git
```

or:

```
$ cd prody
$ git remote add prodymaster git@bitbucket.org:abakan/prody.git
```

[34] http://prody.csb.pitt.edu

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[35] master repository and merge them to your local copy:

```
$ 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:

```
$ python setup.py build_ext --inplace --force
```

4.2 Documenting ProDy

ProDy documentation is written using reStructuredText[36] markup and prepared using Sphinx[37]. You may install Sphinx using **`easy_install`**, i.e. `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:

```
$ cd docs
$ make html
$ make pdf
```

If all documentation strings and pages are properly formatted according to reStructuredText[38] markup, documentation pages should compile without any warnings. Note that to build PDF files, you need to install **`latex`** and **`pdflatex`** programs.

Read the Docs

A copy of ProDy manual is hosted on Read the Docs[39] and can be viewed at http://prody.readthedocs.org/. Read the Docs is configured to build manual pages for the `devel` branch (latest) and the recent stable versions. The user name for Read the Docs is `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

[35] http://prody.csb.pitt.edu
[36] http://docutils.sf.net/rst.html
[37] http://sphinx.pocoo.org/
[38] http://docutils.sf.net/rst.html
[39] https://readthedocs.org/

To build website on ProDy server, start with pulling changes:

```
$ cd ProDy-website
$ git pull
```

Running the following command will build HTML pages for the latest stable release of ProDy:

```
$ 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:

```
$ make pdf
```

PDF and TGZ/ZIP files are copied to appropriate places after they are built.

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 43) 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[40].

5. Generate the source distributions:

   ```
   $ cd ..
   $ python setup.py sdist --formats=gztar,zip
   ```

[40] http://github.com/prody/ProDy

6. Prepare and test Windows installers (see *Making Windows Installers* (page 48)).

   Installers should be prepared for the following versions of Python:

   ```
   $ C:\Python27\python setup.py bdist_wininst
   $ C:\Python35\python setup.py bdist_wininst
   $ C:\Python36\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.

   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 1ubi
   ```

   If this command runs for all supported Python versions, release is good to go.

7. Put all installation source and executable in dist directory.
8. Upload the new release files to the PyPI[41] using twine:

   ```
   $ twine upload dist/*
   ```

   This will offer a number of options. ProDy on PyPI is owned by user `prody.devel`.

9. Commit final changes, if there are any:

   ```
   $ cd ..
   $ git commit -a
   ```

10. Tag the repository with the current version number and push new tag:

    ```
    $ git tag vX.Y
    $ git push --tags
    ```

11. Rebase `devel` branch to `master`:

    ```
    $ git checkout master
    $ git rebase devel
    $ git push
    ```

12. Update the documentation on ProDy[42] website. See *Documenting ProDy* (page 38).
13. Now that you made a release, you can go back to development. You may start with appending `'-dev'` to `__release__` in `prody/__init__.py`.

4.4 Style Guide for ProDy

[41] http://pypi.python.org/pypi/ProDy
[42] http://prody.csb.pitt.edu

4.4.1 Introduction

PEP 8[43], the *Style Guide for Python Code*, is adopted in the development of ProDy package. Contributions to ProDy shall follow **PEP 8**[44] 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:

```
# -*- coding: utf-8 -*-
```

Imports

In addition to **PEP 8#imports**[45] 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.atomgroup 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:

```
"""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
```

[43] https://www.python.org/dev/peps/pep-0008
[44] https://www.python.org/dev/peps/pep-0008
[45] https://www.python.org/dev/peps/pep-0008#imports

```
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#whitespace-in-expressions-and-statements**[46]), two whitespace characters should follow a period in documentation files and strings to help reading documentation in terminal windows and text editors.

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` for Anisotropic Network Model
- `HierView` 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`, 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()`, `wrapAtoms()`, `assignSecstr()`, and `calcSubspaceOverlap()`.

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()` is preferred over `buildHessianMatrix()`. Another example is the change from using `getResidueNames()` to using `AtomGroup.getResnames()`. In fact, this was part of a series of major *Release Notes* (page 49) 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()` and `buildDistMatrix()`
- `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()`

[46] https://www.python.org/dev/peps/pep-0008#whitespace-in-expressions-and-statements

- `getSomething()`: methods, and sometimes functions, that return a copy of data should start with `get`, such as `listReservedWords()`
- `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[47]. 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`[48].
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`.

[47] http://nose.readthedocs.org
[48] https://docs.python.org/3/library/unittest.html#unittest.TestCase

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.

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

Tutorial folders and files are prepared, see tutorials/enm_analysis
```

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`:

```
tutorial_logo = u'enm.png'      # default is ProDy logo
tutorial_prody_version = u''        # default is latest ProDy version
```

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[49], an easy-to-read/write file format. See reStructuredText Primer[50] for a quick introduction.

reStructuredText is stored in plain-text files with `.rst` extension, and converted to HTML and PDF pages using Sphinx[51].

`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:

```
.. _enm-analysis:

.. use "enm-analysis" to refer to this file, i.e. :ref:`enm-analysis`

*****************************************************************************
ENM Analysis using ProDy
*****************************************************************************


.. add .rst files to `toctree` in the order that you want them

.. toctree::
   :glob:
   :maxdepth: 2

   intro
```

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[52]. In the beginning of each `.rst` file, you should make necessary imports as follows:

```
.. ipython:: python

   from prody import *
   from matplotlib.pylab import *
   ion()
```

[49] http://docutils.sourceforge.net/rst.html
[50] http://sphinx-doc.org/rest.html
[51] http://sphinx-doc.org/
[52] http://ipython.org/ipython-doc/dev/development/ipython_directive.html

This will convert to the following:

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

ImportErrorTraceback (most recent call last)
<ipython-input-1-5d14cc12dc44> in <module>()
----> 1 from prody import *

/home/docs/checkouts/readthedocs.org/user_builds/prody/envs/v1.10/local/lib/python2.7/
↪site-packages/prody/__init__.pyc in <module>()
     59 __all__ = ['checkUpdates', 'confProDy', 'startLogfile', 'closeLogfile', 'plog
↪']
     60
---> 61 from . import utilities
     62 from .utilities import *
     63 from .utilities import PackageLogger, PackageSettings

/home/docs/checkouts/readthedocs.org/user_builds/prody/envs/v1.10/local/lib/python2.7/
↪site-packages/prody/utilities/__init__.py in <module>()
     71 from .doctools import *
     72
---> 73 from . import catchall
     74 from .catchall import *
     75 __all__.extend(catchall.__all__)

/home/docs/checkouts/readthedocs.org/user_builds/prody/envs/v1.10/local/lib/python2.7/
↪site-packages/prody/utilities/catchall.py in <module>()
      7 from scipy import spatial
      8 from .misctools import addBreaks, interpY
----> 9 from Bio import Phylo
     10
     11 __all__ = ['calcTree', 'clusterMatrix', 'showLines', 'showMatrix',
↪'reorderMatrix', 'findSubgroups']

ImportError: No module named Bio

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')

NameErrorTraceback (most recent call last)
<ipython-input-4-08265ebed54c> in <module>()
----> 1 pdb = parsePDB('1p38')

NameError: name 'parsePDB' is not defined
```

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()`[53] 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

   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::

   @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:

[53] https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot

```
$ make pdf
$ make files
```

ProDy online documentation will contain these files as well as tutorial pages in HTML format.

4.7 Making Windows Installers

MinGW[54] 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

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()`. Python `int()` 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 **True** 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()`[55] 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()`[56], ProDy function `relpath()` should be used to avoid problems.

[54] http://www.mingw.org/
[55] https://docs.python.org/3/library/os.path.html#os.path.relpath
[56] https://docs.python.org/3/library/os.path.html#os.path.relpath

CHAPTER 5

Release Notes

5.1 ProDy 1.10 Series

5.1.1 1.10 (Apr 30, 2019)

Signature Dynamics

- Added `calcEnsembleENMs()` to compute ENMs on each conformation of a given ensemble to obtain an ensemble of modes.
- Added `ModeEnsemble` and `sdarray` classes as the basic data types for signature dynamics.
- Added functions such as `calcSignatureSqFlucts()`, `calcSignatureCrossCorr()`,`calcSignatureFractVariance()` to extract signature dynamics.
- Added `calcEnsembleSpectralOverlaps()` to obtain dynamical overlaps/distances among the conformations in a given ensemble.

New Features:

Visualization

- Added `showAtomicLines()` and `showAtomicMatrix()` functions to improve visualization.

- Added an *networkx* option to `showTree()` so that the user can choose to use **:module:'~networkx'** to visualize a given tree.

Ensemble and PDBEnsemble

- Associated an `MSA` object to the `PDBEnsemble` class.
- Added an *pairwise* option to **:method:'.Ensemble.getRMSDs'** to obtain an RMSD table of every pair of conformations in the ensemble.
- Improved **:method:'.Ensemble.setAtoms'** for selecting a subset of residues/atoms of the ensemble.

Databases and Web Services

- Added methods and classes for obtaining data from *CATH* and *Dali*.
- Added additional functions for *Uniprot* and *Pfam* such as `queryUniprot()` and `parsePfamPDBs()`.

Bug Fixes and Improvement

- Fixed compatibility problems for Python 2 and 3.
- Improved the `saveModel()` function to include class-specific features.
- Fixed a bug related to the `Atomgroup` addition method.
- Bug fixes to `NMA` classes.
- Fixed a problem with `MSA` indexing.
- Reorganized file structures and functions for consistency.
- Other bug fixes.

5.2 ProDy 1.9 Series

5.2.1 1.9.4 (Feb 02, 2018)

- Undocumented release and fixes.

5.2.2 1.9.3 (Oct 09, 2017)

Bugfixes

- Bug fix about http and ftp based pdb downloads.
- Bug fixes in PRS calculations.

5.2.3 1.9.2 (Aug 29, 2017)

** New Features**:

Migration to pypi.org

- All repositories are moved to pypi.org

5.2.4 1.9.1 (Aug 18, 2017)

** New Features**:

PDB Secondary Structures

- It is possible to write secondary structure infrmation to PDBs.

Bugfixes

- Fixed the problem about clang compiler for saxs tools.
- If FTP client is not working, HTTP client will be used when downloading PDBs.

5.2.5 1.9 (May 23, 2017)

New Features:

Perturbation Response Scanning

- Perturbation Response Scanning method is fully implemented with new plotting tools.
- Effectors and sensors are calculated from PRS tool.

Visualization with py3Dmol

- In jupyter notebook, if you have installed py3Dmol you can use

  py3Dmol visualization directly instead of simple matplotlib visualization.

mmcif parser

- Another structural format cif is also a part of ProDy parser now.

Bugfixes

- Various indexing issues are fixed.
- Some of the obsolete pdbs will not be downloaded anymore, instead

  replaced pdbs will be downloaded. This will change the priority between ftp and http servers.

5.3 ProDy 1.8 Series

5.3.1 1.8.2 (Jun 5, 2016)

- `addCoordset()` in `PDBEnsemble` class, has an additional argument for NMR models.

5.3.2 1.8.1 (May 28, 2016)

Bugfixes

- `getHits()` in `PDBBlastRecord` class, default overlap threshold changed to 0.7 to match with `mapOntoChain()`.
- `calcModes()` in `RTB` have a bug on number of modes and fixed.
- Tab and indentation errors with Python 3.4 are fixed.

5.3.3 1.8 (May 13, 2016)

MechStiff

- Identification of the weakest/strongest elements of the structure architecture provided together with 3D vizualization and statistics analysis.
- Determination of the effective spring constant for selected pair of residues - useful for Single Molecule Force Spectroscopy (SMFS, AFM) and Steered Molecular Dynamics simulations.
- Evaluating the contributions of each mode to selected deformations

New Features:

Python 2 and 3 Support

- ProDy has been refactored to support Python 2.7 and 3.4. Windows installers for Python 2.7 and 3.4 are available in *Installation* (page 1).
- Unit tests are compatible with Python 2.7 and 3.4, and running them with other versions gives errors due to unavailability of some `unittest`[57] features.

Bugfixes

- Various indexing issues are fixed.
- Compatibility issue of `searchPfam()` with Python 2.7.11 is fixed.

[57] https://docs.python.org/3/library/unittest.html#module-unittest

5.4 ProDy 1.7 Series

5.4.1 1.7.1 (May 31, 2015)

Changes:

- `searchPfam()` uses hmmer for given sequence inputs instead of pfam search.

5.4.2 1.7 (Dec 23, 2013)

New Features:

- `buildPCMatrix()` is implemented for calculation of coevolution with PSICOV method from multiple sequence alignments.
- `specMergeMSA()` is implemented for merging multiple sequence alignment files based on the species identifiers of sequences.
- `exANM` is implemented for explicit membrane ANM calculations.
- `writeMembranePDB()` is implemented for writing PDB structures of created membranes for exANM class.

5.5 ProDy 1.6 Series

5.5.1 1.6.1 (May 31, 2015)

Changes:

- `searchPfam()` uses hmmer for given sequence inputs instead of pfam search.

5.5.2 1.5 (Dec 23, 2013)

New Features:

- `buildPCMatrix()` is implemented for calculation of coevolution with PSICOV method from multiple sequence alignments.
- `specMergeMSA()` is implemented for merging multiple sequence alignment files based on the species identifiers of sequences.

- `exANM` is implemented for explicit membrane ANM calculations.
- `writeMembranePDB()` is implemented for writing PDB structures of created membranes for exANM class.

5.6 ProDy 1.5 Series

5.6.1 1.5.1 (Dec 24, 2013)

Changes:

- `PDBBlastRecord` become picklable.

5.6.2 1.5 (Dec 23, 2013)

New Features:

- `buildDirectInfoMatrix()` and `calcMeff()` are implemented for calculation of direct information from multiple sequence alignments.
- `showDirectInfoMatrix()` and `showSCAMatrix()` functions are implemented for displaying coevolutionary data.
- `RTB` 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 GitHub[58].

Improvements:

- `DCDFile` and `parseDCD()` 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 43) for more information on how to test ProDy.

5.7 ProDy 1.4 Series

[58] http://github.com/prody/ProDy

5.7.1 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` can handle `Residue` instances that have same segment[59] name, chain[60] identifier, and resnum[61], if PDB file contains `TER` lines to terminate these residues. If these three identifiers are shared by multiple residues, indexing `AtomGroup` 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.

```
>>> 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:

```
>>> print(atoms.numResidues())
1
>>> atoms['A', 1]
<Residue: WAT 1 from Chain A from Unknown (9 atoms)>
```

[59] http://prody.csb.pitt.edu/manual/reference/atomic/fields.html#term-segment
[60] http://prody.csb.pitt.edu/manual/reference/atomic/fields.html#term-chain
[61] http://prody.csb.pitt.edu/manual/reference/atomic/fields.html#term-resnum

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()` reads `TER` records in PDB files. Atoms and hetero atoms (hetatm[62]) that are followed by a `TER` record are now flagged as pdbter[63].

Bugfixes:

- Fixed memory leaks in `uniqueSequences()` and `buildSeqidMatrix()`.

5.7.2 1.4.8 (Nov 4, 2013)

New Features:

- New analysis functions `buildOMESMatrix()` and `buildSCAMatrix()` are implemented.
- New `AtomGroup.numBytes()` method returns an estimate of memory usage.
- New `countBytes()` utility function is added for counting bytes used by NumPy arrays.

Improvements:

- `parsePDB()` resizes data arrays to decrease memory usage.

Bugfixes:

- Fixed memory leaks in MSA `analysis` functions.
- Fixed potential problems with importing contributed libraries.

5.7.3 1.4.7 (Oct 29, 2013)

Improvements:

- `AtomGroup`, `Selection`, and other `Atomic` classes are picklable.
- Improved equality tests for `AtomGroup`. Two different instances are considered equal if they contain identical data and coordinate sets.

5.7.4 1.4.6 (Oct 16, 2013)

Bugfixes:

- Selection problem with using resid[64] is fixed (issue 160[65])
- 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.

[62] http://prody.csb.pitt.edu/manual/reference/atomic/flags.html#term-hetatm
[63] http://prody.csb.pitt.edu/manual/reference/atomic/flags.html#term-pdbter
[64] http://prody.csb.pitt.edu/manual/reference/atomic/fields.html#term-resid
[65] https://github.com/prody/ProDy/issues/160

- Updated `fetchPDBLigand()` to use PDB for fetching XML files.
- Revised handling of MSA file formats to avoid exceptions for unknown extensions.

5.7.5 1.4.5 (Sep 6, 2013)

New Features:

- `parsePDBHeader()` 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[66] selection flag is defined as an alias for noh[67].
- `matchChains()` function can match non-hydrogen atoms using `subset='heavy'` keyword argument.
- Added `update_coords` keyword argument to `PCA.builCovariance()`, 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()` function that prevented reading files.
- Improved parsing of biomolecular transformations.
- Fixed memory allocation in C code used by `parseMSA()` (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` instances in Python 3 series.

5.7.6 1.4.4 (July 22, 2013)

Improvements:

- `writeNMD()` and `parseNMD()` 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()` that would cause `KeyError`[68] when bonds are set but fragments are not determined is fixed.
- Import ProDy would fail when `HOME`[69] is not set. Changed `PackageSettings` to handle this case graciously.

[66] http://prody.csb.pitt.edu/manual/reference/atomic/flags.html#term-heavy
[67] http://prody.csb.pitt.edu/manual/reference/atomic/flags.html#term-noh
[68] https://docs.python.org/3/library/exceptions.html#KeyError
[69] https://matplotlib.org/faq/environment_variables_faq.html#envvar-HOME

5.7.7 1.4.3 (June 14, 2013)

Changes:

- `getVMDpath()` and `setVMDpath()` functions are deprecated for removal, use `pathVMD()` instead.
- Increased `blastPDB()` *timeout* to 60 seconds.
- `extendModel()` and `extendMode()` functions have a new option for normalizing extended mode(s).
- `sampleModes()` and `traverseMode()` automatically normalizes input modes.

Bugfixes:

- A bug in `applyTransformation()` is fixed. The function would interpret some external transformation matrices incorrectly.
- A bug in `fetchPDBLigand()` function is fixed.

5.7.8 1.4.2 (April 19, 2013)

Improvements:

- `fetchPDB()` and `fetchPDBfromMirror()` functions can handle partial PDB mirrors. See `pathPDBMirror()` for setting a mirror path.

Changes:

- MSE[70] is included in the definition of non-standard amino acids, i.e. nonstdaa[71].

Bugfixes:

- Atom selection problems related to using all[72] and none[73] in composite selections, e.g. `'calpha and all'`, is fixed by defining these keywords as *Atom Flags* (page 33).
- Fasta files with sequence labels using multiple pipe characters would cause C parser (and so `parseMSA()`) 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()` would throw an exception. This case is handled by slicing the chain identifier string.
- A problem in `viewNMDinVMD()` related to module imports is fixed.
- A problem with handling weights in `loadEnsemble()` is fixed.

5.7.9 1.4.1 (Dec 16, 2012)

New Features:

- `buildSeqidMatrix()` and `uniqueSequences()` functions are implemented for comparing sequences in an `MSA` object.
- `showHeatmap()`, `parseHeatmap()`, and `writeHeatmap()` functions are implemented to support VMD plugin Heat Mapper[74] file format.

[70] http://www.pdb.org/pdb/ligand/ligandsummary.do?hetId=MSE
[71] http://prody.csb.pitt.edu/manual/reference/atomic/flags.html#term-nonstdaa
[72] http://prody.csb.pitt.edu/manual/reference/atomic/flags.html#term-all
[73] http://prody.csb.pitt.edu/manual/reference/atomic/flags.html#term-none
[74] http://www.ks.uiuc.edu/Research/vmd/plugins/heatmapper/

- `Sequence` is implemented to handle individual sequence records and point to sequences in `MSA` instances.
- *evol occupancy* (page 25) application is implemented for refined MSA quality checking purposes.
- `mergeMSA()` function and *evol merge* (page 24) application are implemented for merging Pfam MSA to study multi-domain proteins.

Improvements:

- `refineMSA()` function and *evol refine* (page 28) application can perform MSA refinements by removing similar sequences.
- `writePDB()` function takes *beta* and *occupancy* arguments to be outputted in corresponding columns.
- `MSA` indexing and slicing are revised and improved.
- `parseMSA()` 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 5), *prody gnm* (page 13), and *prody pca* (page 14) 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()`, and use the default US server.
- Indexing `MSA` returns `Sequence` instances.
- Iterating over `MSA` and `MSAFile` yields `Sequence` instances.

Bugfixes:

- Fixed a syntax problem that prevented running ProDy using Python 2.6.
- Fixed `NMA` 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.

5.7.10 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`[75] features.

Sequence Analysis

- New applications *Evol Applications* (page 18) are available.
- `searchPfam()` and `fetchPfamMSA()` functions are implemented for searching and retrieving Pfam data. See MSA Files[76] for usage examples.
- `MSAFile` class, `parseMSA()` and `writeMSA()` functions are implemented for reading and writing multiple sequence alignments. See MSA Files[77] for usage examples.
- `MSA` class has been implemented for storing and manipulating MSAs in memory.
- `calcShannonEntropy()`, `buildMutinfoMatrix()`, and `calcMSAOccupancy()` functions are implemented implemented for MSA analysis. See Evolution Analysis[78] for usage examples.
- `showShannonEntropy()`, `showMutinfoMatrix()`, and `showMSAOccupancy()` functions are implemented implemented for MSA analysis. See Evolution Analysis[79] for usage examples.
- `applyMutinfoCorr()` and `applyMutinfoNorm()` functions are implemented for applying normalization and corrections to mutual information matrices.
- `calcRankorder()` function is implemented for identifying highly correlated/co-evolving pairs of residues.

Bugfix:

- Fixed selection issues involving use of `x` or negative numbers.

5.8 ProDy 1.3 Series

5.8.1 1.3.1 (Nov 6, 2012)

New Features:

- Added `fetchPDBviaHTTP()` and `fetchPDBviaFTP()` functions.
- Added `copyFile()` function to `utilities`.
- Added `prody test` command for convenient testing of ProDy package.

Improvements:

- Improved `gunzip()` function to handle `.gz` extensions and string buffers.

Changes:

[75] https://docs.python.org/3/library/unittest.html#module-unittest
[76] http://prody.csb.pitt.edu/tutorials/evol_tutorial/msafiles.html#msafiles
[77] http://prody.csb.pitt.edu/tutorials/evol_tutorial/msafiles.html#msafiles
[78] http://prody.csb.pitt.edu/tutorials/evol_tutorial/msaanalysis.html#msa-analysis
[79] http://prody.csb.pitt.edu/tutorials/evol_tutorial/msaanalysis.html#msa-analysis

- `getWWPDBFTPServer()` and `setWWPDBFTPServer()` are deprecated for removal in v1.4, use `wwPDBServer()` instead.
- `getPDBLocalFolder()` and `setPDBLocalFolder()` are deprecated for removal in v1.4, use `pathPDBFolder()` instead.
- `getPDBMirrorPath()` and `setPDBMirrorPath()` are deprecated for removal in v1.4, use `pathPDBMirror()` instead.
- `getPDBCluster()` is deprecated for removal in v1.4, use `listPDBCluster()` instead.
- `getReservedWords()` is deprecated for removal in v1.4, use `listReservedWords()` instead.
- `getNonstdProperties()` is deprecated for removal in v1.4, use `listNonstdAAProps()` instead.

Bugfix:

- Fixed a bug in `HierView` 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[80].
- Added dummy atom check in `Ensemble.setAtoms()` and `Trajectory.setAtoms()` methods to avoid indexing problems.

5.8.2 1.3 (Sep 30, 2012)

Improvements:

- `select` module and its documentation are completely rewritten. `Select` class uses simplest possible parser to evaluate selection strings and achieves more than 25% speed-up on average.
- *Atom Selections* (page 33) become more forgiving of small typos, but will issue warning messages when they are detected via `SelectionWarning`. These messages can be turned of using `confProDy()`
- Functions used in *ProDy Applications* (page 4) have been refactored to allow for using them directly. See `apps` for their documentation.

Bugfix:

- A problem in *prody catdcd* (page 9) command that was introduced when refactoring `trajectory`[81] classes is fixed.

5.9 ProDy 1.2 Series

[80] http://prody.csb.pitt.edu/manual/reference/atomic/flags.html#term-protein
[81] http://prody.csb.pitt.edu/manual/reference/trajectory/index.html#module-prody.trajectory

5.9.1 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`[82] module regarding Numpy numeric types is fixed. Problem would emerge on platforms which do not offer some numeric types, e.f. `np.float16`.
- Fixed problems in *prody anm* (page 5), *prody gnm* (page 13), and *prody fetch* (page 12) related to writing output files.

Changes:

- The way that *prody fetch* (page 12) command handles files containing PDB identifiers has changed.

5.9.2 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` and `Ensemble` 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` instances can be linked to a `Trajectory` using `Trajectory.link()` method and linking status of an instance can be checked using `Trajectory.isLinked()` medhod.
- `Trajectory.setAtoms()` method accepts `AtomGroup` and `Selection` instances and should be used to select a subset of atoms. This method will not link `AtomGroup` 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()` and `Ensemble.setAtoms()` methods, respectively.
- `Trajectory.getSelection()` and `Ensemble.getSelection()` methods are removed, use `Trajectory.getAtoms()` and `Ensemble.getAtoms()` instead.
- `Trajectory` reference coordinates must be changed using `Trajectory.setCoords()` method.

For usage examples see Trajectory Analysis[83], Trajectory Analysis II[84], Frames and Atom Groups[85], and Trajectory Output[86].

New Features:

- *Atom Flags* (page 33), that are used in *Atom Selections* (page 33), is implemented. See its documentation for handy usage examples.
- `sortAtoms()` function is implemented.
- `pickCentralConf()` function is implemented to pick the conformation or the active coordinate set that is closest to the average of coordinate sets.
- `writePSF()`, a simple PSF file writer, is implemented.
- `glob()` utility function is implemented.

[82] https://docs.python.org/3/library/select.html#module-select
[83] http://prody.csb.pitt.edu/tutorials/trajectory_analysis/trajectory.html#trajectory
[84] http://prody.csb.pitt.edu/tutorials/trajectory_analysis/trajectory2.html#trajectory2
[85] http://prody.csb.pitt.edu/tutorials/trajectory_analysis/frame.html#frame
[86] http://prody.csb.pitt.edu/tutorials/trajectory_analysis/outputtraj.html#outputtraj

- `iterPDBFilenames()` function is implemented, which can be used to iterate over all PDB files stored in a local mirror of Protein Data Bank.
- `findPDBFiles()` function is implemented, which can be used to access PDB files in a path.

Improvements:

- `HierView` instances are built more efficiently. Two times speed-up is achieved by delaying instantiation of `Chain` and `Residue` instances until they are needed.
- Multiple *Atom Flags* (page 33) can be used in *Atom Selections* (page 33) without using `'and'` operator, e.g. `'sidechain carbon'` is the same as `'sidechain and carbon'`.
- `writePDB()` accepts `Ensemble`, `Conformation`, and `Frame` instances as atoms argument.
- `writePDB()` function is around 25% faster.
- `pickCentral()` is extended to accept `Atomic` and `Ensemble` instances. Old function is now `pickCentralAtom()`.
- *prody align* (page 4) command and `prody_align()` function can handle non-protein atom selections (see examples for *prody align* (page 4)).
- `parsePDB()` and `writePDB()` supports 100K and more atoms.

Changes:

- `showOverlapTable()` displays first set of modes along x axis of the plot.
- `AtomGroup.setData()` does not accept arrays with boolean data type, use `AtomGroup.setFlags()` instead.
- `writePDB()` function argument *model* is changed to *csets* that indicates the coordinate set index of *atoms* argument.
- `PackageLogger.timing()` does not return elapsed time, only logs this information.
- `PackageLogger.startLogfile()` is deprecated for removal in v1.3, use `PackageLogger.start()` instead.
- `PackageLogger.closeLogfile()` is deprecated for removal in v1.3, use `PackageLogger.close()` instead.
- `from prody.utilities import *` will not work anymore due to potential name conflicts with Python standard library functions. Import required functions explicitly.
- `writePDB()` appends `.pdb` extension to filename when it is not present
- *prody select* (page 16) 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()` 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()` instead which save changes permanently.
- `NMA.getModes()` and `ModeSet.getModes()` methods are deprecated for removal in v1.3, use `list()`, e.g. `list(model)`, instead.

Bugfixes:

- Fixed a bug in *prody contacts* (page 9) command that arose problems when when selecting a subset of the target atoms.

Normal Mode Wizard

Improvements:

- *ProDy Interface* shows the size of the trajectory output file for PCA calculations.
- *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 *Protein Graphics Options* panel.

5.10 ProDy 1.1 Series

5.10.1 1.1 (June 1, 2012)

New Features:

- `iterFragments()` function is added.
- `findNeighbors()` function is added.
- `calcMSF()` and `calcRMSF()` functions are added.
- `wrapAtoms()` functions is added.
- `extendMode()` and `extendVector()` functions are added.
- *prody contacts* (page 9) command is added.

Improvements:

- `moveAtoms()` function is improved to move atoms to a specified location.
- `DCDFile` and `parseDCD()` take *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 *prody anm* (page 5), *prody gnm* (page 13), and *prody pca* (page 14) can extend a coarse grained model to backbone or all atoms of the residues. See their documentation pages.

Changes:

- Color scale used by `showOverlapTable()` is normalized by default.
- `tools` module is depracated for removal, use `utilities` instead.
- *array* argument in `moveAtoms()` is replaced with *by* keyword argument.
- *which* argument in `AtomGroup.copy()` method is deprecated for removal in version 1.2.

- `DCDFile` 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()` instead.

Bugfixes:

- Fixed several problems in `iterNeighbors()` function and `Contacts` class that were introduced after transition to new `KDTree` interface.
- Fixed a problem in setting selection strings of fragments identified using `findFragments()`.
- Fixed a problem in `calcCenter()` related to weighted center calculation.
- Fixed a problem of in copying `AtomMap` 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.

5.11 ProDy 1.0 Series

5.11.1 1.0.4 (May 2, 2012)

Bugfixes:

- Fixed a problem in `calcPhi()` function that raised a name error.
- Fixed a problem in `KDTree.getDistances()` method that raised a name error when unitcell is provided.
- Fixed a problem in `buildDistMatrix()` and `calcDistance()` functions causing miscalculations when unitcell is given.
- Revised `KDTree` 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.

5.11.2 1.0.3 (May 1, 2012)

Bugfixes:

- Fixed `kdtree`[87] import problem.

New Features:

- `buildDistMatrix()` function that can take periodic boundary conditions is implemented.

Improvements:

- `calcDistance()` function is improved to take periodic boundary conditions into account when provided by the users.

5.11.3 1.0.2 (May 1, 2012)

New Features:

- Methods to deal with connected subsets of atoms are implemented, see `AtomGroup.iterFragments()` and `AtomGroup.numFragments()`.
- `pickCentral()` 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()` and `confProDy()` for usage details.
- **`prody align`** makes use of `--select` when aligning multiple structures. See usage examples: *prody align* (page 4)
- `printRMSD()` function that prints minimum, maximum, and mean RMSD values when comparing multiple coordinate sets is implemented.
- `findFragments()` function that identifies fragments in atom subsets, e.g. `Selection`, is implemented.
- A new `KDTree` interface with coherent method names and capability to handle periodic boundary conditions is implemented.

Improvements:

- Performance improvements made in `saveAtoms()` and `loadAtoms()`.
- `sliceMode()`, `sliceModel()`, `sliceVector()`, and `reduceModel()` functions accept `Selection` 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()`) is improved to yield items faster.

Changes:

- There is a change in the behavior of addition operation on instances of `AtomGroup`. 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` class instead.

Bugfixes:

[87] http://prody.csb.pitt.edu/manual/reference/kdtree/index.html#module-prody.kdtree

- A problem in building hierarchical views when making selections using *resindex*, *chindex*, and *segindex* keywords is fixed.
- A problem in `Chain` and `Residue` selection strings that would emerge when a `HierView` is build using a selection is fixed.
- A problem with copying `AtomGroup` instances whose coordinates are not set is fixed.
- `AtomGroup` 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` instances in `pickCentral()` function is fixed.
- A problem in `Select` 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()`) to a `Trajectory` after atoms were set (`setAtoms()`). Newly added file would not be associated with the atoms and coordinates parsed from this file would not be set for the `AtomGroup` instance.

5.11.4 1.0.1 (Apr 6, 2012)

New Features:

- ProDy can be configured to automatically check for updates on a regular basis, see `checkUpdates()` and `confProDy()` functions for details.
- `alignPDBEnsemble()` function is implemented to align PDB files using transformations calculated in ensemble analysis. See usage example in Homologous Proteins[88] example.
- `PDBConformation.getTransformation()` 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 33) for usage details.
- `calcCrossProjection()` function is implemented.

Improvements:

- `Select` 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.
- `showProjection()` and `showCrossProjection()` functions are improved to evaluate list of markers, color, labels, and texts. See usage example in Plotting[89].

[88] http://prody.csb.pitt.edu/tutorials/ensemble_analysis/blast.html#pca-blast
[89] http://prody.csb.pitt.edu/tutorials/ensemble_analysis/xray_plotting.html#pca-xray-plotting

- `Trajectory` instances can be used for calculating and plotting projections using `calcProjection()`, `showProjection()`, `calcCrossProjection()`, and `showCrossProjection()` 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 33) for definitions of *protein* and *acidic* keywords.
- Hit dictionaries from `PDBBlastRecord` 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()` method instead.

Bugfixes:

- A bug in some *ProDy Applications* (page 4) 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.

5.11.5 1.0 (Mar 7, 2012)

Improvements:

- `ANM.buildHessian()` 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()` too, but for Kirchoff matrix, version using KDTree is faster and is the default. Both methods have *kdtree* 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 4) for script usage details.
- Added `bonded to ...` selection method that expands a selection to immediately bound atoms. See *Atom Selections* (page 33) for its description.
- `fetchPDBLigand()` parses bond data from the XML file.
- `fetchPDBLigand()` can optionally save compressed XML files into ProDy package folder so that frequent access to same files will be more rapid. See `confProDy()` function for setting this option.
- `Select` 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 8) command.

Changes:

- All deprecated method and functions scheduled for removal are removed.
- `getEigenvector()` and `getEigenvalue()` methods are deprecated for removal in v1.1, use `Mode.getEigvec()` and `Mode.getEigval()` instead.
- `getEigenvectors()` and `getEigenvalues()` methods are deprecated for removal in v1.1, use `NMA.getEigvecs()` and `NMA.getEigvals()` instead.

- `Mode.getCovariance()` and `ModeSet.getCovariance()` methods are deprecated for removal in v1.1, use `calcCovariance()` method instead.
- `Mode.getCollectivity()` method is removed, use `calcCollectivity()` function instead.
- `Mode.getFractOfVariance()` method is removed, use the new `calcFractVariance()` function instead.
- `Mode.getSqFlucts()` method is removed, use `calcSqFlucts()` function instead.
- Renamed `showFractOfVar()` function as `showFractVars()` function instead.
- Removed `calcCumOverlapArray()`, use `calcCumulOverlap()` with `array=True` argument instead.
- Renamed `extrapolateModel()` as `extendModel()`.
- The relation between `AtomGroup`, `Trajectory`, and `Frame` instances have changed. See Trajectory Analysis II[90] and Trajectory Output[91], and Frames and Atom Groups[92] usage examples.
- `AtomGroup` cannot be deformed by direct addition with a vector instance.
- Unmapped atoms in `AtomMap` instances are called dummies. `AtomMap.numUnmapped()` method, for example, is renamed as `AtomMap.numDummies()`.
- `fetchPDBLigand()` 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 8) so that when a path for downloading files are given files are not save to local PDB folder.

5.12 ProDy 0.9 Series

5.12.1 0.9.4 (Feb 4, 2012)

Changes:

- `setAtomGroup()` and `getAtomGroup()` methods are renamed as `Ensemble.setAtoms()` and `Ensemble.getAtoms()`.

[90] http://prody.csb.pitt.edu/tutorials/trajectory_analysis/trajectory2.html#trajectory2
[91] http://prody.csb.pitt.edu/tutorials/trajectory_analysis/outputtraj.html#outputtraj
[92] http://prody.csb.pitt.edu/tutorials/trajectory_analysis/frame.html#frame

- `AtomGroup` 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` and `Trajectory` classes.

Bugfixes:

- Bugfixes in `Bond.setACSIndex()`, `saveAtoms()`, and `HierView.getSegment()`.
- Bugfixes in `GammaVariableCutoff` and `GammaStructureBased` classes.
- Bugfix in `calcCrossCorr()` function.
- Bugfixes in `Ensemble.getWeights()`, `showOccupancies()`, `DCDFile.flush()`.
- Bugfixes in ProDy commands *prody blast* (page 8), *prody fetch* (page 12), and *prody pca* (page 14).
- Bugfix in `calcCenter()` function.

5.12.2 0.9.3 (Feb 1, 2012)

New Features:

- `DBRef` class is implemented for storing references to sequence databases parsed from PDB header records.
- Methods for storing coordinate set labels in `AtomGroup` instances are implemented: `getACSLabel()`, and `getACSLabel()`.
- `calcCenter()` and `moveAtoms()` functions are implemented for dealing with coordinate translation.
- Hierarchical view, `HierView`, 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`[93] 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` 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:

```
with Trajectory('mdm2.dcd') as traj:
    for frame in traj:
        calcGyradius(frame)
```

- `Chain` slicing is implemented:

```
p38 = parsePDB('1p38')
chA = p38['A']
res_4to10 = chA[4:11]
res_100toLAST = chA[100:]
```

- Some support for bonds is implemented to `AtomGroup` class. Bonds can be set using `setBonds()` method. All bonds must be set at once. `iterBonds()` or `iterBonds()` methods can be used to iterate over bonds in an AtomGroup or an Atom.

[93] https://docs.python.org/3/library/collections.html#collections.OrderedDict

- `parsePSF()` parses bond information and sets to the atom group.
- `Selection.update()` method is implemented, which may be useful to update a distance based selection after coordinate changes.
- `buildKDTree()` and `iterNeighbors()` methods are implemented for facilitating identification of pairs of atoms that are proximal.
- `iterAtoms()` method is implemented to all `atomic`[94] classes to provide uniformity for atom iterations.
- `calcAngle()`, `calcDihedral()`, `calcPhi()`, `calcPsi()`, and `calcOmega()` methods are implemented.

Improvements:

- `Chain.getSelstr()` and `Residue.getSelstr()` methods are improved to include the selection string of a `Selection` when they are built using one.

Changes:

- `Residue` methods `getNumber()`, `setNumber()`, `getName()`, `setName()` methods are deprecated and will be removed in v1.0.
- `Chain` methods `getIdentifier()` and `setIdentifier()` methods are deprecated and will be removed in v1.0.
- `Polymer` attribute `identifier` is renamed as `chid`.
- `Chemical` attribute `identifier` is renamed as `resname`.
- `getACSI()` and `setACSI()` are renamed as `getACSIndex()` and `setACSIndex()`, respectively.
- `calcRadiusOfGyration()` is deprecated and will be removed in v1.0. Use `calcGyradius()` instead.

Bugfixes:

- Fixed a problem in `parsePDB()` that caused loosing existing coordinate sets in an `AtomGroup` when passed as *ag* argument.
- Fixed a problem with `"same ... as ..."` argument of `Select` 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` instances.

5.12.3 0.9.2 (Jan 11, 2012)

New Features:

- **`prody catdcd`** command is implemented for concatenating and/or slicing `.dcd` files. See *prody catdcd* (page 9) for usage examples.
- `DCDFile` can be opened in write or append mode, and coordinate sets can be added using `write()` method.
- `getReservedWords()` can be used to get a list of words that cannot be used to label user data.
- `confProDy()` function is added for configuring ProDy.

[94] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic

- ProDy can optionally backup existing files with `.BAK` (or another) extension instead of overwriting them. This behavior can be activated using `confProDy()` function.

Improvements:

- `writeDCD()` file accepts `AtomGroup` or other `Atomic` 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` instances, when closed, raise exception, similar to behavior of `file` objects in Python.
- Title of `AtomGroup` instances resulting from copying an `Atomic` 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`[95] module. See *ProDy Applications* (page 4) for details of changes.
- `argparse`[96] module is added to the package for Python versions 2.6 and older.

Bugfixes:

- Fixed problems in `loadAtoms()` and `saveAtoms()` functions.
- Bugfixes in `parseDCD()` and `writeDCD()` functions for Windows compatability.

5.12.4 0.9.1 (Nov 9, 2011)

Bug Fixes:

- Fixed problems with reading and writing configuration files.
- Fixed problem with importing nose for testing.

5.12.5 0.9 (Nov 8, 2011)

New Features:

- PDBML[97] and mmCIF[98] files can be retrieved using `fetchPDB()` function.
- `getPDBLocalFolder()` and `setPDBLocalFolder()` functions are implemented for local PDB folder management.
- `parsePDBHeader()` is implemented for convenient parsing of header data from `.pdb` files.
- `showProtein()` is implemented to allow taking a quick look at protein structure.
- `Chemical` and `Polymer` classes are implemented for storing chemical and polymer component data parsed from PDB header records.

Changes:

[95] https://docs.python.org/3/library/argparse.html#module-argparse
[96] https://docs.python.org/3/library/argparse.html#module-argparse
[97] http://pdbml.pdb.org/
[98] http://mmcif.pdb.org/

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 `DeprecationWarning`[99] 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 `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:

Old name	New name
`applyBiomolecularTransformations()`	`buildBiomolecules()`
`assignSecondaryStructure()`	`assignSecstr()`
`scanPerturbationResponse()`	`calcPerturbResponse()`
`calcCrossCorrelations()`	`calcCrossCorr()`
`calcCumulativeOverlap()`	`calcCumulOverlap()`
`calcCovarianceOverlap()`	`calcCovOverlap()`
`showFractOfVariances()`	`showFractVars()`
`showCumFractOfVariances()`	`showCumulFractVars()`
`showCrossCorrelations()`	`showCrossCorr()`
`showCumulativeOverlap()`	`showCumulOverlap()`
`deform()`	`deformAtoms()`
`calcSumOfWeights()`	`calcOccupancies()`
`showSumOfWeights()`	`showOccupancies()`
`trimEnsemble()`	`trimPDBEnsemble()`
`getKeywordResidueNames()`	`getKeywordResnames()`
`setKeywordResidueNames()`	`setKeywordResnames()`
`getPairwiseAlignmentMethod()`	`getAlignmentMethod()`
`setPairwiseAlignmentMethod()`	`setAlignmentMethod()`
`getPairwiseMatchScore()`	`getMatchScore()`
`setPairwiseMatchScore()`	`setMatchScore()`
`getPairwiseMismatchScore()`	`getMismatchScore()`
`setPairwiseMismatchScore()`	`setMismatchScore()`
`getPairwiseGapOpeningPenalty()`	`getGapPenalty()`
`setPairwiseGapOpeningPenalty()`	`setGapPenalty()`
`getPairwiseGapExtensionPenalty()`	`getGapExtPenalty()`
`setPairwiseGapExtensionPenalty()`	`setGapExtPenalty()`

[99] https://docs.python.org/3/library/exceptions.html#DeprecationWarning

Coordinate methods:

All `getCoordinates()` and `setCoordinates()` methods in `atomic`[100] and `ensemble`[101] 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.

Old name	New name	Affected modules
`getNumOfAtoms()`	`numAtoms()`	`atomic`[102], `ensemble`[103], `dynamics`
`getNumOfChains()`	`numChains()`	`atomic`[104]
`getNumOfConfs()`	`numConfs()`	`ensemble`[105]
`getNumOfCoordsets()`	`numCoordsets()`	`atomic`[106], `ensemble`[107]
`getNumOfDegOfFreedom()`	`numDOF()`	`dynamics`
`getNumOfFixed()`	`numFixed()`	`ensemble`[108]
`getNumOfFrames()`	`numFrames()`	`ensemble`[109]
`getNumOfResidues()`	`numResidues()`	`atomic`[110]
`getNumOfMapped()`	`numMapped()`	`atomic`[111]
`getNumOfModes()`	`numModes()`	`dynamics`
`getNumOfSelected()`	`numSelected()`	`ensemble`[112]
`getNumOfUnmapped()`	`numUnmapped()`	`atomic`[113]

`getName` **method**:

`getName()` methods are renamed as `getTitle()` to avoid confusions that might arise from changes in `atomic`[114] method names listed below. All classes in `atomic`[115], `ensemble`[116], and `dynamics`[117] are affected from this change.

In line with this change, `parsePDB()` and `parsePQR()` *name* arguments are changed to *title*, but *name* argument will also work until release 1.0.

This name change conflicted with `DCDFile.getTitle()` 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:

[100] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic
[101] http://prody.csb.pitt.edu/manual/reference/ensemble/index.html#module-prody.ensemble
[102] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic
[103] http://prody.csb.pitt.edu/manual/reference/ensemble/index.html#module-prody.ensemble
[104] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic
[105] http://prody.csb.pitt.edu/manual/reference/ensemble/index.html#module-prody.ensemble
[106] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic
[107] http://prody.csb.pitt.edu/manual/reference/ensemble/index.html#module-prody.ensemble
[108] http://prody.csb.pitt.edu/manual/reference/ensemble/index.html#module-prody.ensemble
[109] http://prody.csb.pitt.edu/manual/reference/ensemble/index.html#module-prody.ensemble
[110] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic
[111] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic
[112] http://prody.csb.pitt.edu/manual/reference/ensemble/index.html#module-prody.ensemble
[113] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic
[114] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic
[115] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic
[116] http://prody.csb.pitt.edu/manual/reference/ensemble/index.html#module-prody.ensemble
[117] http://prody.csb.pitt.edu/manual/reference/dynamics/index.html#module-prody.dynamics

Old name	New name
getAtomNames()	getNames()
getAtomTypes()	getTypes()
getAltLocIndicators()	getAltlocs()
getAnisoTempFactors()	getAnisos()
getAnisoStdDevs()	getAnistds()
getChainIdentifiers()	getChains()
getElementSymbols()	getElements()
getHeteroFlags()	getHeteros()
getInsertionCodes()	getIcodes()
getResidueNames()	getResnames()
getResidueNumbers()	getResnums()
getSecondaryStrs()	getSecstrs()
getSegmentNames()	getSegnames()
getSerialNumbers()	getSerials()
getTempFactors()	getBetas()

This change affects all atomic[118] classes, AtomGroup, Atom, Chain, Residue, Selection and AtomMap.

Other changes in atomic methods:

- getSelectionString() renamed as getSelstr()

Methods handling user data (which was previously called attribute) are renamed as follows:

Old name	New name
getAttribute()	getData()
getAttrNames()	getDataLabels()
getAttrType()	getDataType()
delAttribute()	delData()
isAttribute()	isData()
setAttribute()	setData()

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()
- removed getProteinResidueNames(), overloaded getKeywordResnames()
- removed setProteinResidueNames(), overloaded setKeywordResnames()

Scripts:

The way ProDy scripts work has changed. See *ProDy Applications* (page 4) for details. Using older scripts will start issuing deprecation warnings in 2012.

Bug Fixes:

- Bugs in execDSSP() and execSTRIDE() functions that caused exceptions when compressed files were passed is fixed.

[118] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic

- 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[119] pages for instructions on installing it.

5.13 ProDy 0.8 Series

5.13.1 0.8.3 (Oct 16, 2011)

New Features:

- Functions to read and write PQR files: `parsePQR()` and `writePQR()`.
- Added `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 `prody.test()`.

Improvements:

- `blastPDB()` function and `PDBBlastRecord` class are rewritten to use faster and more compact code.
- New `PackageLogger` function is implemented to unify logging and reporting task progression.
- Improvements in PDB ensemble support functions, e.g. `trimPDBEnsemble()`, are made.
- Improvements in ensemble concatenations are made.

Bug Fixes:

- 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.

[119] http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz
[120] http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz

5.13.2 0.8.2 (Oct 14, 2011)

New Features:

- `fetchPDBClusters()`, `loadPDBClusters()`, and `getPDBCluster()` functions are implemented for handling PDB sequence cluster data. These functions can be used instead of `blastPDB()` 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] is implemented as `scanPerturbationResponse()` based on the code provided by Ying Liu.

Changes:

- `fetchPDBLigand()` 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`[121] when the function fails to perform its action due to missing data in header dictionary.
- `fetchPDB()` decompresses PDB files found in the working directory when user asks for decompressed files.
- `parsePDB()` appends *chain* and *subset* arguments to `AtomGroup()` name.
- *chain* argument is added to `PDBBlastRecord.getHits()`.

Improvements:

- Atom selection class `Select` is completely redesigned to prevent breaking of the parser when evaluating invalid selection strings.
- Improved type checking in `parsePDB()` function.

Bug Fixes:

- Bugfixes in `parseDSSP()`: 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()` 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` and `select` modules are developed.

5.13.3 0.8.1 (Sep 16, 2011)

New Features:

- `fetchLigandData()` is implemented for fetching ligand data from Ligand Expo.
- `parsePSF()` function is implemented for parsing X-PLOR format PSF files.

Changes:

- __slots__ is used in `AtomGroup` and `Atomic` classes. This change prevents user from assigning new variables to instances of all classes derived from the base `Atomic`.

[121] https://docs.python.org/3/library/exceptions.html#ValueError

- `pyparsing` is updated to version 1.5.6.

Bug Fixes:

- A bug in `AtomGroup.copy()` method is fixed. When AtomGroup instance itself is copied, deep copies of data arrays were not made.
- A bug in `Select` class raising exceptions when negative residue number values are present is fixed.
- Another bug in `Select` class misinterpreting `same residue as ...` statement when specific chains are involved is fixed.
- A bug in `AtomGroup.addCoordset()` 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.

5.13.4 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` class for analyzing PDB structures, rename it as `PDBEnsemble`. See the other changes that may affect your work below and the class documentation for more information.

New Features:

- `DCDFile` is implemented for handling DCD files.
- `Trajectory` is implemented for handling multiple trajectory files.
- `writeDCD()` is implemented for writing DCD files.
- Trajectory Analysis[122] example to illustrate usage of new classes for handling DCD files. Essential Dynamics Analysis[123] example is updated to use new ProDy classes.
- `PCA` supports `Trajectory` and `DCDFile` instances.
- `Ensemble` and `PDBEnsemble` classes can be associated with `AtomGroup` instances. This allows selecting and evaluating coordinates of subset of atoms. See `setAtomGroup()`, `select()`, `getAtomGroup()`, and `getSelection()` methods.
- `execDSSP()`, `parseDSSP()`, and `performDSSP()` functions are implemented for executing and parsing DSSP calculations.
- `execSTRIDE()`, `parseSTRIDE()`, and `performSTRIDE()` functions are implemented for executing and parsing DSSP calculations.
- `parsePDB()` function parses atom serial numbers. Atoms can be retrieved from an `AtomGroup` instance by their serial numbers using `getBySerial()` and `getBySerialRange()` methods.

[122] http://prody.csb.pitt.edu/tutorials/trajectory_analysis/trajectory.html#trajectory
[123] http://prody.csb.pitt.edu/tutorials/trajectory_analysis/eda.html#eda

- `calcADPs()` function can be used to calculate anisotropic displacement parameters for atoms with anisotropic temperature factor data.
- `getRMSFs()` is implemented for calculating root mean square fluctuations.
- `AtomGroup` and `Mode` or `Vector` additions are supported. This adds a new coordinate set to the `AtomGroup` instance.
- `getAttrNames()` is implemented for listing user set attribute names.

Improvements:

- `calcProjection()`, `showProjection()`, and `showCrossProjection()` functions can optionally calculate/display RMSD along the normal mode.
- ANM, GNM, and PCA applications can optionally write compressed ProDy data files.
- `fetchPDB()` function can optionally write decompressed files and force copying a file from local mirror to target folder.
- `PCA.buildCovariance()` and `PCA.performSVD()` methods accept Numpy arrays as coordinate sets.
- Performance of `PCA.buildCovariance()` method is optimized for evaluation of PDB ensembles.
- `calcRMSD()` and `superpose()` functions are optimized for speed and memory usage.
- `Ensemble.getMSFs()` is optimized for speed and memory usage.
- Improvements in memory operations in `atomic`[124], `ensemble`[125], and `dynamics`[126] modules for faster data (PDB/NMD) output.
- Optimizations in `Select` and `Contacts` classes.

Changes:

- `Ensemble` does not store conformation names. Instead, newly implemented `PDBEnsemble` class stores identifiers for individual conformations (PDB IDs). This class should be used in cases where source of individual conformations is important.
- `calcProjection()`, `showProjection()`, and `showCrossProjection()` 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()`.
- `parseDCD()` function returns `Ensemble` 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`[127] module.
- Problem in **`prody fetch`** that occurred when a file is found in a local mirror is fixed.
- Bugfix in `AtomPointer.copy()` method.

[124] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic
[125] http://prody.csb.pitt.edu/manual/reference/ensemble/index.html#module-prody.ensemble
[126] http://prody.csb.pitt.edu/manual/reference/dynamics/index.html#module-prody.dynamics
[127] http://prody.csb.pitt.edu/manual/reference/ensemble/index.html#module-prody.ensemble

Normal Mode Wizard[128]

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.

5.14 ProDy 0.7 Series

5.14.1 0.7.2 (Jun 21, 2011)

New Features:

- `parseDCD()` is implemented for parsing coordinate sets from DCD files.

Improvements:

- `parsePDB()` 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()` function causing problems when writing a single mode is fixeed.
- Other bugfixes in `dynamics`[129] module functions.

5.14.2 0.7.1 (Apr 28, 2011)

Highlights:

[128] http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz
[129] http://prody.csb.pitt.edu/manual/reference/dynamics/index.html#module-prody.dynamics

- `Atomic __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` class is integrated with `HierView` 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()` is implemented for quick alignment of protein structures. See Ligand Extraction[130] usage example.
- `setAttribute()`, `getAttribute()`, `delAttribute()`, and `isAttribute()` functions are implemented for `AtomGroup` class to facilitate storing user provided atomic data. See Storing data in AtomGroup[131] example.
- `saveAtoms()` and `loadAtoms()` 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()` function is implemented to allow for calculating collectivity of deformation vectors.

Improvements:

- `parsePDB()` can optionally return biomolecule when `biomol=True` keyword argument is passed.
- `parsePDB()` can optionally make secondary structure assignments when `secondary=True` keyword argument is passed.
- `calcSqFlucts()` function is changed to accept `Vector` instances, e.g. deformation vectors.

Changes:

- Changes were made in `calcADPAxes()` function to follow the conventions in analysis ADPs. See its documentation.

Bug Fixes:

- A in `Ensemble` slicing operations is fixed. Weights are now copied to the new instances obtained by slicing.
- Bug fixes in `dynamics`[132] plotting functions `showScaledSqFlucts()`, `showNormedSqFlucts()`,

5.14.3 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()` function is implemented for parsing matrices in sparse format. See the usage example in Using an External Matrix[133].
- `deform()` function is implemented for deforming coordinate sets along a normal mode or linear combination of multiple modes.

[130] http://prody.csb.pitt.edu/tutorials/structure_analysis/ligands.html#extract-ligands
[131] http://prody.csb.pitt.edu/tutorials/prody_tutorial/atomgroup.html#id1
[132] http://prody.csb.pitt.edu/manual/reference/dynamics/index.html#module-prody.dynamics
[133] http://prody.csb.pitt.edu/tutorials/enm_analysis/external.html#external-matrix

- `sliceModel()` 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 `calcADPAxes()` 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` and `NMA` classes are instantiated with an empty string, instances are called "Unnamed".
- `sliceMode()`, `sliceVector()` and `reduceModel()` functions return the atom selection in addition to the sliced vector/mode/model instance.

Bug Fixes:

- Default selection for `calcGNM()` 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.

5.15 ProDy 0.6 Series

5.15.1 0.6.2 (Mar 16, 2011)

New Features:

- `performSVD()` function is implemented for faster and more memory efficient principal compoment 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[134].
- `plog()` is implemented for enabling users to make log entries.

Improvements:

- `compare` functions are improved to handle insertion codes.
- `HierView` allows for indexing using chain identifier and residue numbers. See usage example Hierarchical Views[135].
- `Chain` allows for indexing using residue number and insertion code. See usage example Hierarchical Views[136].
- `addCoordset()` function accepts `Atomic` and `Ensemble` instances as *coords* argument.
- New method `HierView.getAtoms()` is implemented.
- `AtomGroup` set functions check the correctness of dimension of data arrays to prevent runtime 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` allowed calculating covariance matrix for less than 3 coordinate sets is fixed.
- A bug in `mapOntoChain()` function that caused problems when mapping all atoms is fixed.

5.15.2 0.6.1 (Mar 2, 2011)

New Features:

[134] http://prody.csb.pitt.edu/tutorials/enm_analysis/extend.html#extendmodel
[135] http://prody.csb.pitt.edu/tutorials/prody_tutorial/hierview.html#hierview
[136] http://prody.csb.pitt.edu/tutorials/prody_tutorial/hierview.html#hierview

- `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()` 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()` and `matchChains()` 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()` is fixed. When parsing backbone or Cα atoms, residue names were not checked and this caused parsing water atoms with name `"O"` or calcium ions with name `"CA"`.

5.15.3 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[137] 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[138]).
- `Gamma` class is implemented for facilitating use of force constants based on atom type, residue type, or property. An example derived classes are `GammaStructureBased` and `GammaVariableCutoff`.
- `calcTempFactors()` function is implemented to calculate theoretical temperature factors.
- 5 new *ProDy Applications* (page 4) are implemented, and existing scripts are improved to output figures.
- `getModel()` method is implemented to make function development easier.
- `resetTicks()` function is implemented to change X and/or Y axis ticks in plots when there are discontinuities in the plotted data.

Improvements:

- `ANM.buildHessian()` and `GNM.buildKirchhoff()` classes are improved to accept `Gamma` instances or other custom function as *gamma* argument. See also Custom Gamma Functions[139].

[137] http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz
[138] http://prody.csb.pitt.edu/tutorials/nmwiz_tutorial/intro.html#nmwiz
[139] http://prody.csb.pitt.edu/tutorials/enm_analysis/gamma.html#gamma

- `Select` 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()` and other functions.
- `Ensemble` class accepts `Atomic` instances and automatically adds coordinate sets to the ensemble.

Changes:

- `PDBlastRecord` is renamed as `PDBBlastRecord`.
- `NMA` 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()` and `calcGNM()` 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 5) and *prody gnm* (page 13).
- Bug fixes in `showSqFlucts()` and `showProjection()` 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.

5.16 ProDy 0.5 Series

5.16.1 0.5.3 (Feb 11, 2011)

New Features:

- Membership, equality, and non-equality test operation are defined for all `atomic`[140] classes. See Operations on Selections[141].
- Two functions are implemented for dealing with anisotropic temperature factors: `calcADPAxes()` and `buildADPMatrix()`.
- `NMA.setEigens()` and `NMA.addEigenpair()` methods are implemented to assist analysis of normal modes calculated using external software.
- `parseNMD()` is implemented for parsing NMD files.
- `parseModes()` is implemented for parsing normal mode data.
- `parseArray()` is implementing for reading numeric data, particularly normal mode data calculated using other software for analysis using ProDy.
- The method in [BH02] to calculate overlap between covariance matrices is implemented as `calcCovOverlap()` function.
- `trimEnsemble()` to trim `Ensemble` instances is implemented.
- `checkUpdates()` to check for ProDy updates is implemented.

Changes:

- Change in default behavior of `parsePDB()` 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()`.
- `ProDyStartLogFile()` is renamed as `startLogfile()`.
- `ProDyCloseLogFile()` is renamed as `closeLogfile()`.
- `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()` is fixed to output B-factors (Thanks to Dan Holloway for pointing it out).

5.16.2 0.5.2 (Jan 12, 2011)

Bug Fixes:

- An important fix in `sampleModes()` function was made (Thanks to Alberto Perez for finding the bug and suggesting a solution).

Improvements:

- Improvements in `ANM.calcModes()`, `GNM.calcModes()`, and `PCA.calcModes()` methods prevent Numpy/Scipy throwing an exception when more than available modes are requested by the user.

[140] http://prody.csb.pitt.edu/manual/reference/atomic/index.html#module-prody.atomic
[141] http://prody.csb.pitt.edu/tutorials/prody_tutorial/selection.html#selection-operations

- Improvements in `blastPDB()` enable ProDy 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[142].

5.16.3 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.

5.16.4 0.5 (Dec 21, 2010)

New Features:

- `AtomPointer` base class for classes pointing to atoms in an `AtomGroup`.
- `AtomPointer` instances (Selection, Residue, etc.) can be added. See Operations on Selections[143] for examples.
- `Select.getIndices()` and `Select.getBoolArray()` methods to expand the usage of `Select`.
- `sliceVector()` and `sliceMode()` functions.
- `saveModel()` and `loadModel()` functions for saving and loading NMA data.
- `parsePDBStream()` can now parse specific chains or alternate locations from a PDB file.
- `alignCoordsets()` is implemented to superimpose coordinate sets of an `AtomGroup` instance.

Bug Fixes:

- A bug in `parsePDBStream()` that caused unidentified errors when a model in a multiple model file did not have the same number of atoms is fixed.

Changes:

- Iterating over a `Chain` instance yields `Residue` instances.
- `Vector` 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()`.

5.17 ProDy 0.2 Series

[142] http://prody.csb.pitt.edu/tutorials/structure_analysis/pdbfiles.html#writepdb
[143] http://prody.csb.pitt.edu/tutorials/prody_tutorial/selection.html#selection-operations

5.17.1 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()`.
- `showOverlapMatrix()` is renamed to `showOverlapTable()`.
- Modules are reorganized.

New Features:

- `Atomic` for easy type checking.
- `Contacts` for faster intermolecular contact identification.
- `Select` can identify intermolecular contacts. See Intermolecular Contacts[144] for an examples and details.
- `sampleModes()` implemented for sampling conformations along normal modes.

Improvements:

- `proteins.compare` functions are improved. Now they perform sequence alignment if simple residue number/identity based matchin does not work, or if user passes `pwalign=True` argument. This impacts the speed of X-ray ensemble analysis.
- `Select` 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()` 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.
- 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.

[144] http://prody.csb.pitt.edu/tutorials/structure_analysis/contacts.html#contacts

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.

5.18 ProDy 0.1 Series

5.18.1 0.1.2 (Nov 9, 2010)

- Important bug fixes and improvements in NMA helper and plotting functions.
- Documentation updates and improvements.

5.18.2 0.1.1 (Nov 8, 2010)

- Important bug fixes and improvements in chain comparison functions.
- Bug fixes.
- Source clean up.
- Documentation improvements.

5.18.3 0.1 (Nov 7, 2010)

- First release.

CHAPTER 6

About ProDy

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 the Bahar Lab[145] at the University of Pittsburgh[146] with support from NIH R01 GM099738 award.

6.1.1 Development Team

Ahmet Bakan[147] initiated the *ProDy* project, designed and developed *ProDy*, *NMWiz*, *Evol*, and *DruGUI*.

Cihan Kaya[148] is currently overseeing the overall development of *ProDy*.

She (John) Zhang[149] is currently helping on maintaining and developing *ProDy*.

Hongchun Li[150] is currently maintaining and developing ANM and GNM servers.

Anindita Dutta[151] contributed to the development of *Evol*, `database` and `sequence`[152] modules.

Tim Lezon contributed to development of Rotations and Translation of Blocks and Membrane ENM.

Wenzhi Mao[153] contributed to development of MSA analysis functions.

[145] http://www.ccbb.pitt.edu/faculty/bahar/
[146] http://www.pitt.edu/
[147] http://ahmetbakan.com
[148] http://pitt.edu/~cihank
[149] http://www.csb.pitt.edu/Faculty/bahar/lab.html
[150] http://www.csb.pitt.edu/Faculty/bahar/lab.html
[151] http://www.linkedin.com/pub/anindita-dutta/5a/568/a90
[152] http://prody.csb.pitt.edu/manual/reference/sequence/index.html#module-prody.sequence
[153] http://www.linkedin.com/pub/wenzhi-mao/2a/29a/29

Lidio Meireles[154] provided insightful comments on the design of *ProDy*, and contributed to the development of *ProDy Applications* (page 4).

6.1.2 Contributors

In addition to the development team members, we acknowledge contributions and feedback from the following individuals:

Ying Liu[155] provided the code for Perturbation Response Scanning method.

Kian Ho[156] contributed with bug fixes and unit tests for DSSP functions.

Gökçen Eraslan[157] contributed with bug fixes and development and maintenance insights.

6.2 Citing

When using *ProDy* or *NMWiz* in published work, please cite:

> Bakan A, Meireles LM, Bahar I.
> ProDy: Protein Dynamics Inferred from Theory and Experiments.
> *Bioinformatics* **2011** 27(11):1575-1577.

When using *pairwise2* or *KDTree* modules in published work, please cite:

> Cock PJ, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, Friedberg I, Hamelryck T, Kauff F, Wilczynski B, de Hoon MJ.
> Biopython: freely available Python tools for computational molecular biology and bioinformatics.
> *Bioinformatics* **2009** 25(11):1422-3.

6.3 Credits

ProDy makes use of the following great software:

pyparsing[158] 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.

Biopython[159] 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 NumPy[160] 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.

[154] http://www.linkedin.com/in/lidio
[155] http://www.linkedin.com/pub/ying-liu/15/48b/5a9
[156] https://github.com/kianho
[157] http://blog.yeredusuncedernegi.com/
[158] http://pyparsing.wikispaces.com
[159] http://biopython.org
[160] http://www.numpy.org

Finally, ProDy can benefit from SciPy[161] and Matplotlib[162] 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 *ProDy* is supported by NIH through R01 GM099738 award.

6.5 License

6.5.1 ProDy

ProDy is available under the MIT License[163]:

```
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[164] 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[165]:

```
                Biopython License Agreement

Permission to use, copy, modify, and distribute this software and its
documentation with or without modifications and for any purpose and
```

[161] http://www.scipy.org
[162] http://matplotlib.org
[163] http://opensource.org/licenses/MIT
[164] http://biopython.org
[165] http://www.biopython.org/DIST/LICENSE

```
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.
```

6.5.3 Pyparsing

The pyparsing[166] module is distributed with the ProDy package. Pyparsing is developed by Paul T. McGuire and is available under the MIT License[167]:

```
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[168] is distributed with the ProDy package. Argparse is developed by Steven J. Bethard and is available under the Python Software Foundation License[169].

6.5.5 CEalign

CEalign module is distributed with ProDy. The original CE method was developed by Ilya Shindyalov and Philip Bourne. The Python version which is used by ProDy is developed by Jason Vertrees and available

[166] http://pyparsing.wikispaces.com
[167] http://opensource.org/licenses/MIT
[168] http://code.google.com/p/argparse/
[169] http://docs.python.org/license.html

under the New BSD license:

```
Copyright (c) 2007, Jason Vertrees.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

    * Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

    * Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in
    the documentation and/or other materials provided with the
    distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Python Module Index

p

Index