
BioShell 3.0

Release 3.0

Jul 19, 2023

Overview

1	What is BioShell	1
1.1	BioShell functionality	1
1.2	BioShell command-line utilities	1
1.3	BioShell tests & examples	2
1.4	BioShell library for Python (aka PyBioShell)	2
1.5	Previous versions	2
1.6	Citations	3
2	Installation	5
3	PyBioShell Installation	9
3.1	0. Prequisites	10
3.2	1. Clone and compile binder	10
3.3	2. Build PyBioShell	10
4	Welcome to BioShell documentation!	11
5	Indices and tables	13
6	BioShell programs	15
6.1	clust tutorial : clustering sequences and structures	15
6.2	BioShell cookbook	17
7	BioShell examples	23
7.1	BioShell examples list	23
7.2	Examples by functionality	736
7.3	Examples by keywords	737
8	BioShell C++ library	741
8.1	Reading and processing PDB files	741
9	BioShell Python library	743
9.1	Reading and writing PDB files	744
9.2	Help! My script crashes!	745
9.3	Using PyBioShell in PyMOL	745
10	SURPASS model	747
10.1	SURPASS force field	747

10.2	Input files	748
10.3	Output files	749
10.4	SURPASS representation	749
10.5	<i>surpass_annealing</i> program	749
11	Notes for BioShell developers	751
12	Indices and tables	753
Index		755

CHAPTER 1

What is BioShell

BioShell is a general bioinformatics toolkit, focused on biomolecular structures. It provides:

Command line applications that have been distributed since the original 1.0 version of the package. Some of them have changed their names (e.g. **HCPM** has been renamed to **clust**)

Many (currently over a hundred) small applications that also serve as integration tests. They come with example input data and expected output

Python library majority of BioShell classes may be directly used in Python

C++ library which offers highly optimized implementations of oftenly used BioInformatics algorithms and protocols.

1.1 BioShell functionality

BioShell functionality covers file processing such as data filtering and file formats conversion. It handle protein sequences, sequence profiles and alignments. Structures calculations capabilities include superimpositions, crmsd calculations, alignments, Phi/Psi angles and many more.

Since its first publication, BioShell has been providing a small **set of command-line programs** for easy data manipulation from a UNIX-like terminal or a shell script. The newest release extends this set by over a hundred simple command-line utilities. See [examples page](#) to see which program can help you in solving a particular problem.

1.2 BioShell command-line utilities

The original BioShell command line utilities are still maintained, although their functionality is a bit redundant with applications released with BioShell 3.0 version. See [Programs page](#) for details.

1.3 BioShell tests & examples

Since the most recently published version 3.0, BioShell package comes with **extensive set of example applications**, which have been created to simultaneously reach tree goals:

- **to extend the set of BioShell command line tools.** Programs with names starting with `ap_` are in fact yet another applications. The difference between these test and *standard* apps is that the latter perform only a single action and their command line is simplified. These programs are integration tests at the same time.
- **to provide high quality code snippets that help BioShell users write their own programs.** Small programs, that show how to use a particular class or a function, are named `ex_*`. At the same time they serve as *unit tests*
- **to test the code.** Both `ex_*` and `ap_*` tests are automatically executed by a test server to ensure the quality and integrity of the package. Input data as well as curated output of these tests is versioned in git repository along the source code.

All the examples are included in respective API documentation pages. Since the test are continuously tested, the serve as a source of validated snippets for creating future programs.

1.4 BioShell library for Python (aka PyBioShell)

BioShell distribution provides also bindings to Python scripting language; that is, BioShell is also a **versatile library for python scripting**. BioShell objects can be imported as any other python modules. Example scripts are also included in the repository.

Precompiled library (a single `.so` file) for Unix distributions can be downloaded for the following Python versions. Click on an appropriate link below:

- [python 3.7](#)
- [python 3.6](#)
- [python 3.5](#)
- [python 2.7](#)

or type this command in your terminal:

```
curl -O http://bioshell.pl/downloads/bioshell/Python37/pybioshell.so
```

Remember to add path with `pybioshell.so` to your PyBioShell script eg.

```
sys.path.append('/home/username/src.git/bioshell/bin/')
```

If you really need to compile your own version follow the instructions [here](#)

1.5 Previous versions

1.5.1 BioShell versions 1.x

The original BioShell package was designed as a suite of programs designed for pre- and post-processing in protein structure modeling protocols. The package has been providing a convenient set of tools for in conversion between

various sequence and structure formats. It has been also possible to calculate simple properties of protein conformations. The very first commands (e.g. HCPM for clustering protein structures) were implemented in C, later on the development switched to C++.

1.5.2 BioShell versions 2.x

Around 2006/07 BioShell has been reimplemented in JAVA, designed as a library for scripting languages running on Java Virtual Machine, most notably Python, but also Scala, Ruby, Groovy and many others. Currently the most recent stable release is 2.2. API docs as well as example scripts may be found in documentation. All program from 1.x versions were also ported to JAVA.

1.6 Citations

- **BioShell - the third version:** Joanna M. Macnar, Natalia A. Szulc, Justyna D. Kryś, Aleksandra E. Badaczewska-Dawid and Dominik Gront “*BioShell 3.0: Library for processing structural biology data.*” *Biomolecules* **2020**, 10, 461; <https://doi.org/10.3390/biom10030461>
- **Three-dimensional protein threading:**
 - D. Gront, M. Blaszczyk, P. Wojciechowski, A. Kolinski “*Bioshell Threader: protein homology detection based on sequence profiles and secondary structure profiles.*” *Nucleic Acids Research* **2012** doi:10.1093/nar/gks555
- **One-dimensional protein threading:**
 - P. Gniewek, A. Kolinski, D. Gront “*Optimization of profile-to-profile alignment parameters for one-dimensional threading.*” *J. Computational Biology* **2012** Jul;19(7):879-86
- **BioShell - the second version:**
 - D. Gront and A. Kolinski “*Utility library for structural bioinformatics*” *Bioinformatics* **2008** 24(4):584-585
- **BBQ - program for backbone reconstruction:**
 - D. Gront, S. Kmiecik, A. Kolinski “*Backbone Building from Quadrilaterals. A fast and accurate algorithm for protein backbone reconstruction from alpha carbon coordinates.*” *J. Comput. Chemistry* **2007** 28(9):1593-1597
- **BioShell - the first version:**
 - D. Gront and A. Kolinski “*BioShell - a package of tools for structural biology computations*” *Bioinformatics* **2006** 22(5):621-622
- **Program for clustering protein structures (currently named clust):**
 - D. Gront and A. Kolinski “*HCPM - program for hierarchical clustering of protein models*” *Bioinformatics* **2005** 21(14):3179-3180

CHAPTER 2

Installation

This document describes, how to install binary programs of BioShell toolkit. See [PyBioShell Installation page](#) for instruction regarding Python bindings.

BioShell package has been written in C++11 and must be built before use. This is a quite easy process, which requires CMake (<https://cmake.org>) and a relatively modern C++ compiler such as gcc 5.0 or clang 10.0

Just follow the steps below to compile the package:

1. *Install zlib*
2. *Clone bioshell*
3. *Run CMake*
4. *Run Make*
5. *Set BIOSHELL_DATA_DIR path*

The two additional sections below provide more information on customization of the building process:

6. *Additional parameters for compilation*
7. *Using IDE*

1. Install zlib

BioShell requires zlib library so it can handle compressed files. You must install developer version of the library to be able to compile BioShell. On Ubuntu linux it can be installed by the command:

```
sudo apt-get install zlib1g-dev
```

2. Clone BioShell

If you haven't done it yet, **clone bioshell repository** (<https://bitbucket.org/dgront/bioshell/src/master/>) from Bitbucket:

```
git clone https://bitbucket.org/dgront/bioshell.git  
cd bioshell
```

This should create `bioshell` directory in your current location. The second line steps into this new directory

2.1 Clone submodules for Bioshell

Now Bioshell package contains submodules to use machine learning. Update neccecary submodules with this command:

```
git submodule update --init
```

Submodules will be downloaded to `external/` directory in bioshell repository.

3. Run CMake:

```
cd build  
cmake ..
```

The `build` directory will contain compilation intermediate files and may be deleted once BioShell is compiled. The first line enters that direcotry, the second command calls `cmake` to set up the compilation process. CMake attempts to set up everything automatically, sometimes however it would require some guidance, e.g. to find the right compiler (see below)

4. Run Make:

```
make -j 4
```

where `-j 4` allows make use 4 cores to run parallel compilations. This command will attempt to compile **all targets**; the list of all targets can be printed by `make help`. As one can see, each executable is a separate target. There are also predefined group targets:

bioshell compiles only `bioshell` library

bioshell-apps compiles `bioshell` library and bioshell toolkit applications, such as **seqc** and **strc**

examples compiles all examples, i.e. all `ap_` and `ex_` application

5. Set BIOSHELL_DATA_DIR path

Last step is to add path to `data/` directory to your shell variables e.g.

```
export BIOSHELL_DATA_DIR="/Users/username/bioshell/data"
```

or add this variable to your `~/.bashrc`:

```
echo 'export BIOSHELL_DATA_DIR="/Users/username/bioshell/data"' >> ~/.bashrc
```

6. Additional parameters for compilation

The procedure described above compiles the package with the default settings: *Release* build with no profiling. To change it, you should **remove everything from** `./build` **directory** and generate new makefiles with new settings:

- in order to use a compiler other than the default one (e.g. gcc version 4.9), say:

```
cmake -DCMAKE_CXX_COMPILER=g++-4.9 -DCMAKE_C_COMPILER=gcc-4.9 -DCMAKE_<br>BUILD_TYPE=Release ..
```

or to use `icc` for instance:

```
cmake -DCMAKE_CXX_COMPILER=icc -DCMAKE_C_COMPILER=icc -DCMAKE_BUILD_TYPE=Release ..
```

- to selecting a different compiler and making a profile build

```
-DCMAKE_CXX_COMPILER=icc -DCMAKE_C_COMPILER=icc -D PROFILE=ON -DCMAKE_BUILD_TYPE=Release ..
```

- to brew a **debug** build, turn `-DCMAKE_BUILD_TYPE=Release` into `-DCMAKE_BUILD_TYPE=Debug`. So to make a **debug** build **without changing the compiler**, say just:

```
cmake -DCMAKE_BUILD_TYPE=Debug ..
```

- to make a profiling build (-pg option) for gcc or *Xcode Instruments* add `-D PROFILE=ON` to the `cmake` command (the custom `PROFILE` variable test is implemented in the main `CMakeLists.txt`).

```
cmake -D PROFILE=ON ..
```

7. Using IDE

In the above examples, `cmake` was used to produce makefiles for to compile BioShell. `cmake` command may be also used to generate project files for other environments, in particular:

- to produce *.xcodeproj file for xcode:

```
cmake -DCMAKE_BUILD_TYPE=Release -G Xcode
```

- or to prepare *solution* files for Microsoft Visual Studio (must be run on a Windows machine):

```
cmake -DCMAKE_BUILD_TYPE=Release -G "Visual Studio 2013"
```


CHAPTER 3

PyBioShell Installation

PyBioShell is a set of Python bindings to **BioShell** library. It allows use of BioShell classes like any other Python modules. The closest tool similar by functionality is Biopython, which however is partially written in Python.

The easiest option to get PyBioShell on your machine is to download precompiled library, available for the following Python versions. Click on an appropriate link below:

- [python 3.7](#)
- [python 3.6](#)
- [python 3.5](#)
- [python 2.7](#)

or type this command in your terminal:

```
curl -O http://bioshell.pl/~jkrys/pybioshell/pybioshell37/pybioshell.so
```

You also need `data/` directory, which contains files necessary to run BioShell. Download `data.tar.gz`, uncompress it and put it somewhere BioShell will be able to find it, see [here](#) for details.

Remember to add path with `pybioshell.so` to your shell variables e.g.

```
export PYTHONPATH="$PYTHONPATH:$HOME/bioshell/bin"
```

or add this variable to your `~/.bashrc`:

```
echo 'export PYTHONPATH="$PYTHONPATH:$HOME/bioshell/bin"' >> ~/.bashrc
```

Remember also to add `data/` directory to your shell variables. Look [here](#) for details.

Another way is to compile it from sources, following the steps given below. The procedure assumes your `bioshell` repository is located in `src.git/bioshell/` and `binder` in `src.git/binder/`; these paths are arbitrary but the commands must be adjusted accordingly.

3.1 0. Prerequisites

In order to compile `binder`, you need to have Ninja building tool ([website](#)) and cmake. You will also need python headers, available from `python-dev` package or similar (e.g. `python3.5-dev`). On Ubuntu Linux you can install them with `apt-get`:

```
sudo apt-get install ninja-build cmake python-dev
```

The use of `clang` compiler is advised. Try to get `clang-6.0` or newer (see [this link](#))

3.2 1. Clone and compile binder

To clone `binder` from its *github* repository:

```
git clone https://github.com/RosettaCommons/binder
cd binder
python3 ./build.py -j 4
```

where the last command actually builds `binder` using four CPU cores for that. Note, that `binder` uses more than 1GB of disc space and its compilation may take a few hours.

3.3 2. Build PyBioShell

Open `scripts/build_pybioshell.py` file and edit variables, adapting it to your system. In particular, you most likely have to fix `clang++` version (`LINKER_CMD` variable) as well as the path where the `binder` executable is located (`BINDER_PATH` variable). Make a directory `build_bindings` in the main BioShell directory, i.e in the directory where `pybioshell.config` is located. Choose your Python version and run the compilation as follow:

```
python3 ./scripts/build_pybioshell.py -v 3.5
```

You should find your compiled version in `bin/pybioshell.so`. If you have any problems with compilation, please do not hesitate to contact us.

CHAPTER 4

Welcome to BioShell documentation!

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

CHAPTER 6

BioShell programs

Currently, BioShell distribution provides the following programs:

seqc (*cookbook recipes*): sequence converter : a utility to convert between sequence data formats

strc (*cookbook recipes*): structure converter : a utility to work with PDB files

str_calc (*cookbook recipes*): structure calculator; perform various calculations on a PDB file

clust (*cookbook recipes*): calculates hierarchical clustering of arbitrary objects based on a map of pairwise distances between them

hist (*cookbook recipes*): simple utility to make 1D and 2D histograms

Now you can browse [BioShell cookbook](#), or read tutorials, listed below

6.1 clust tutorial : clustering sequences and structures

Clustering procedure allows one to divide arbitrary number of objects into groups according to their mutual (dis)similarity. This method is widely used in bioinformatics and molecular modeling to deal with data sets that are too large to be inspected manually. Here we give two examples of Hierarchical Agglomerative Clustering with BioShell package:

- 1) to cluster a pool of protein sequences
- 2) to cluster results of protein-peptide docking

The BioShell procedure for clustering divides the task into three steps:

- 1) **calculate a matrix of distances between elements subjected to a clustering analysis.**

As a result, a flat text file should be produced. The three columns of that file must provide i-th element ID, j-th element ID and the respective distance value

- 2) **run the actual clustering procedure.**

Although the procedure can be stopped at a particular cutoff distance, we advise to conduct the calculations i.e. until all the objects are merged into a single cluster. Clustering tree will be stored in an output file

3) analyse the clustering tree to retrieve clusters at a desired cutoff level

Below we show how to perform these three steps for two different clustering applications

6.1.1 Example 1. Clustering protein sequences by their mutual sequence identity

Step 1: Calculating the distances

Clustering procedure should merge close sequences (i.e. small mutual distance) into a single cluster, while dissimilar sequences should be placed in different clusters. Unfortunately, sequence identity value (seq_id) cannot be used here because its largest value (1.0) denotes identical sequences. Here we propose to use $1.0 - \text{seq_id}$ as a distance function.

First we use `ap_PairwiseSequenceIdentityProtocol` program to evaluate all pairwise distances:

```
ap_PairwiseSequenceIdentityProtocol inp.fasta 8 0.4 > seq_id.out 2>LOG
```

where `inp.fasta` is the input file (FASTA format), 8 is the number of cores (threads run in parallel) and `0.4` is the smallest `seq_id` value to be written to a file.

Then the `seq_id` values are converted into distances with `awk` command line tool:

```
awk '{print $1,$2,1.0-$3}' seq_id.out > distances.out
```

Step 2: Clustering the data

Then we run the `clust` tool:

```
clust -in::file=distances.out \
-n=46621 \
-complete \
-clustering:missing_distance=1.1 \
-clustering:out:tree=tree-complete >clust_out 2>clust_log
```

The `-n` option is necessary to provide the number of objects subjected to clustering (not the number of distance values!). `-clustering:missing_distance` Provides the default distance value for the cases it's undefined. The clustering tree will be stored in a file specified by `-clustering:out:tree` option

Step 3: Analysis

```
clust -in::file=distances.out \
-n=46621 \
-clustering:in:tree=tree-complete \
-clustering:out:clusters \
-clustering:out:distance=0.4 \
-clustering:out:min_size=1
```

6.1.2 Example 2. Clustering results of protein-peptide docking

The input data set contains 12500 conformations of a protein receptor (`1jd4`) with a short peptide bound to its surface. The conformations were calculated with `FlexPepDocking` program from Rosetta modelling suite.

Step 1: Calculating the distances

Step 2: Clustering the data

We run *clust* program as above, just should remember to put the correct input file name and to change the number of data elements (i.e. protein conformations)

```
clust -in::file=1jd4-pep-crsmd \
-n=12500 \
-complete \
-clustering:missing_distance=15.1 \
-clustering:out:tree=tree-complete >clust_out 2>clust_log
```

Step 3: Analysis

```
clust -in::file=all_vs_all_crmsd_15 \
-n=12500 -clustering:out:clusters \
-clustering:out:distance=2.5 \
-clustering:out:min_size=10 \
-clustering:in:tree=tree-complete
```

6.2 BioShell cookbook

This cookbook provides a bunch of handy one-liners that simplify daily tasks in structural bioinformatics.

6.2.1 bash-only recipes

Combine a bunch of .pdb files into a single multimodel-pdb:

```
k=0;
for i in *.pdb; do
    k=$((k+1));
    echo "MODEL      "$k;
    cat $i;
    echo "ENDMDL";
done > all-pdb
mv all-pdb all.pdb
```

6.2.2 1. seqc recipes

1.1 Create FASTA from PDB (prints FASTA on a screen):

```
seqc -in:pdb=2gb1.pdb -out:fasta
```

1.2 Create FASTA from PDB, including secondary structure:

```
seqc -in:pdb=2gb1.pdb -out:fasta -in::pdb::header -out:fasta:secondary
```

Secondary structure annotation is extracted from the PDB file header (-in::pdb::header option is necessary to parse it)

1.3 Create SS2 file from PDB:

```
seqc -in:pdb=2gb1.pdb -out:ss2 -in::pdb::header
```

As above, the secondary structure is extracted from the PDB file header; all the probability values (last three columns in a SS2 file) are set either to 1.0 or 0.0

1.4 Count secondary structure elements in a bunch of PDB files, create a nice table:

```
for i in 2gb1.pdb 2fdo.pdb 1rrx.pdb
do
    ss=`seqc -in:pdb=$i -out:ss2 -in::pdb::header -of -out::sequence::width=0 \
        | tail -1 | fold -w1 | uniq | sort | uniq -c | tr '\n' ' '
    echo $i $ss
done 2>/dev/null
```

As in **recipe 1.2**, but this time a combination of a few bash commands is used to parse the output and count the number of secondary structure elements: coil (C), strands (E) and helices (H). Example output looks as below:

```
2gb1.pdb 6 C 4 E 1 H
2fdo.pdb 7 C 6 E 3 H
1rrx.pdb 16 C 11 E 5 H
```

1.5 Write FASTA file with only one line per sequence (un-wrap sequences)

```
seqc -in:fasta=in.fasta -out:sequence:width=0 -out:fasta
```

1.6 Convert ASN.1 sequence profile (*psiblast* output) into a text format

```
seqc -in:profile:asn1=d1or4A_.asn1 -out:profile:txt
```

1.7 As in **recipe 1.5** (i.e. **.asn1 -> .txt**), but this time reorder profile columns

```
seqc -in:profile:asn1=d1or4A_.asn1 -out:profile:txt \
    -out:profile:columns=GAPVILMCHWFYKRQDNQST
```

1.8 Sort sequences from the longest to the shortest

```
seqc -in:fasta=in.fasta -seqc:sort -out:fasta
```

This recipe can obviously be combined with the one above (every FASTA sequence in a single line)

1.9 Basic sequence filtering

```
seqc -in:fasta=in.fasta -seqc:sort -select::sequence::protein -out:fasta \
    -select::sequence::long_at_least=30
```

Print only amino acid sequences (due to **-select::sequence::protein** filter) that are at least 30 residues long

1.10 Basic sequence filtering: keep nucleotide sequences

```
seqc -in:fasta=in.fasta -seqc:sort -select::sequence::nucleic -out:fasta \
    -select::sequence::long_at_least=30
```

Print only nucleic acid sequences (due to **-select::sequence::nucleic** filter) that are at least 30 residues long

6.2.3 2. strc recipes

2.1 Write only chain A of the given input PDB file

```
strc -in:pdb=5edw.pdb -select::chains=A -out:pdb=5edwA.pdb
```

2.2 Write only aminoacids of chain A (ligands, water etc will be removed)

```
strc -in:pdb=5edw.pdb -select::chains=A -out:pdb=5edwA.pdb -select::aa
```

2.3 Write only selected fragment of a given protein (residues from 1 to 83 of chain A)

```
strc -in:pdb=1PQX.pdb -select::substructure=A:1-83 -op=out.pdb
```

6.2.4 3. str_calc recipes

3.1 Find all pairwise all-atom crmsd distances between all the models in a given PDB

```
str_calc -in:pdb=2kmk-1.pdb -calc::crmsd -in:pdb::all_models -  
-in:pdb:native=2KMK.pdb.gz
```

3.2 Read in only CA atoms; find all pairwise crmsd distances between all the models in a given PDB

```
str_calc -select::ca -in:pdb=2kmk-1.pdb -calc::crmsd -in:pdb::all_models \  
-in:pdb:native=2KMK.pdb.gz
```

3.3 Generate *theoretical* NOE restraints on for a protein backbone

```
str_calc -in:pdb=2kwi.pdb -in:pdb:with_hydrogens \  
-calc::distmap::describe -calc::distmap::allatom
```

This command lists all distances between any two backbone atoms; `-in:pdb:with_hydrogens` option forces BioShell to read hydrogen atoms, which is false by default, `-calc::distmap::describe` turns on longer atom descriptions. The output may look as below:

A GLN	9	N	10	A GLY	8	N	1	3.602
A GLN	9	N	10	A GLY	8	CA	2	2.418
A GLN	9	N	10	A GLY	8	C	3	1.326
A GLN	9	N	10	A GLY	8	O	4	2.245
A GLN	9	N	10	A GLY	8	HA2	8	2.506
A GLN	9	N	10	A GLY	8	HA3	9	2.959
A GLN	9	CA	11	A GLY	8	N	1	4.834
A GLN	9	CA	11	A GLY	8	CA	2	3.788
A GLN	9	CA	11	A GLY	8	C	3	2.425
A GLN	9	CA	11	A GLY	8	O	4	2.756

```
str_calc -in:pdb=2kwi.pdb -in:pdb:with_hydrogens -calc::distmap::describe \  
-calc::distmap::allatom | awk '{if(($11<2.5) && ($3-$8>4)) print $0}'
```

This output is the filtered with awk. The ouput lines must satisfy the criteria: distance below 2.5 Angstroms, sequence separation at least 4 residues.

3.3 Find all-atom crmsd distances between all models in a single PDB and the reference native structure

```
str_calc -in:pdb=2kmk-1.pdb -calc::crmsd -in:pdb::all_models -  
-in:pdb:native=2KMK.pdb.gz
```

3.4 As in the above example, but after superimposing alpha-carbons, calculate crmsd on all the atoms:

```
str_calc -in:pdb=2kmk-1.pdb -calc::crmsd -in:pdb::all_models -  
-in:pdb:native=2KMK.pdb.gz \  
-calc::crmsd::matching_atoms=A:*:_CA_ -calc::crmsd::rotated_  
atoms=A:*:*
```

Check peptide docking results: superimpose two structures using alpha carbons of chain A (i.e. the receptor) and calculate crmsd of CA atoms of chain B (i.e. the ligand)

```
str_calc -in:pdb=model-1.pdb -calc::crmsd -in:pdb::all_models -  
-in:pdb:native=native.pdb \  
-calc::crmsd::matching_atoms=A:*:_CA_ -calc::crmsd::rotated_  
atoms=B:*:_CA_
```

3.5 Check peptide docking results: superimpose two structures using alpha carbons of chain A (i.e. the receptor) and calculate crmsd of CA atoms of chain B (i.e. the ligand)

```
str_calc -in:pdb=models-1.pdb -calc::crmsd -in:pdb::all_models -  
-in:pdb:native=native.pdb \  
-calc::crmsd::matching_atoms=A:*:_CA_ -calc::crmsd::rotated_  
atoms=B:*:_CA_
```

Note, that this recipe loads **all models** from the models-1.pdb file. For instance, if that file contains 10 structures, one can expect the following output:

#	name	crmsd	len	crmsd	len
	models-1-1.pdb	0.000	96	0.000	4
	models-1-2.pdb	0.262	96	22.598	4
	models-1-3.pdb	0.274	96	16.670	4
	models-1-4.pdb	0.260	96	16.123	4
	models-1-5.pdb	0.292	96	24.524	4
	models-1-6.pdb	0.320	96	27.575	4
	models-1-7.pdb	0.351	96	24.200	4
	models-1-8.pdb	0.385	96	24.613	4
	models-1-9.pdb	0.297	96	22.778	4
	models-1-10.pdb	0.325	96	25.136	4

The first column identifies a model structure (name-of-input-file + dash + model number), the second and third provide crmsd on the atoms used for superposition (CA atoms of chains A in this case) and the number of these atoms (here 96), respectively. Finally the last two columns provide crmds and atom count for the rotated atom set. The results come for tetrapeptide docking experiment, hence only 4 CA atoms were rotated.

6.2.5 4. clust recipes

4.1 Calculate hierarchical clustering of 140 elements; distances are stored in tm_dist file.

```
clust -i=tm_dist -n=140 -clustering:out:distance=0.4
```

Prints clusters for critical distance 0.4. By default *single link* clustering strategy is used

6.2.6 5. hist recipes

5.1 Calculate a histogram from the 14th column of a given input file:

```
hist -in:file=default.fsc -in:column=14 -hist:x_max=10 -hist:x_min=0
```

The command reads a score file produced by Rosetta and makes a histogram of crmsd, assuming it's in the 14th column

CHAPTER 7

BioShell examples

There are three groups of examples for BioShell library: *ap_** which are functional applications and are helpful for every user, *ex_** show how to use a particular BioShell class or function in your own C++ program. Finally Python scripts (*.py files) show how to solve bioinformatics problems using PyBioShell. You can automatically run these test on your local machine. Use

```
python3 call_all_tests.py
```

in your bioshell/doc_examples/cc-examples directory to run *ap_** and *ex_** or in bioshell/doc_examples/py-examples directory to run *.py scripts. You will find test_results.html in either cc-examples or py-examples directory, which you can open with your web browser to see the test results summary.

Overall there is more than 200 examples than can be accessed by the index pages below:

7.1 BioShell examples list

The latest BioShell 3.0 distribution provides an extensive set of examples. The purpose to create them is three-fold:

- to facilitate continuous testing of the package (unit and integration tests)
- to provide additional functionality to the package, and
- to serve as coding examples and provide ready-to-use snippets

All the tests, which in practice are small C++ applications, were divided into two broad groups; the tests are named staring from *ap_*, *ex_*. In additiion we provide also example Python scripts which use PyBioShell package.

7.1.1 *ap_** programs

These are integration tests, that besides testing whether the package is bug-free, should also do something usefull.

ap_BackboneHBondMap

Reads a PDB file and calculates a map of backbone hydrogen bonds, providing also the geometry of each bond. The resulting table, printed on the screen provides: - H donor residue name and id (columns 1 and 2) - H acceptor residue name and id (columns 4 and 5) - two distances: r(O..H) and r(N..O) (columns 7 and 8) - planar (C-O..H) and dihedral (C-O..H-N) (columns 9 and 10) - DSSP energy for this bond (column 11) - X,Y, Z coordinates of H atom in the local coordinates system (columns 12, 13 and 14) - theta, phi spherical coordinates of H atom (columns 15 and 16)

EXAMPLE OUTPUT:

```
# 42 hydrogen bonds found in backbone:
TYR      3 -> THR    18 : 2.620 3.346 165.58   94.25  -1.170   -0.702   0.211   2.515 ↴
↳ 16.25 163.29
LYS      4 -> LYS    50 : 2.259 3.156 120.71   159.88  -1.310   1.445   1.249   1.205 ↴
↳ 57.75 40.83
LEU      5 -> THR    16 : 1.838 2.802 143.84  -115.27  -2.834   0.102  -1.075   1.488 ↴
↳ 35.98 -84.57
```

USAGE:

```
./ap_BackboneHBondMap input.pdb
```

EXAMPLE:

```
./ap_BackboneHBondMap 5edw.pdb
```

Keywords:

- *PDB input*
- *Hydrogen bonds*
- *data_structures*
- *Protein structure features*

Categories:

- core/calc/structural/BackboneHBondMap

Input files:

- 2gb1.pdb
- 5edw.pdb
- 3dcg.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <core/data/io/Pdb.hh>
3 #include <core/data/structural/selectors/structure_selectors.hh>
4 #include <core/protocols/selection_protocols.hh>
5 #include <core/calc/structural/interactions/BackboneHBondMap.hh>
6 #include <utils/exit.hh>
7
8 using namespace core::data::structural;
9 using namespace core::data::io;
10 using namespace core::data::basic;
11
12 std::string program_info = R"(
13
14 Reads a PDB file and calculates a map of backbone hydrogen bonds, providing also the ↵
15 → geometry of each bond.
16 The resulting table, printed on the screen provides:
17 - H donor residue name and id (columns 1 and 2)
18 - H acceptor residue name and id (columns 4 and 5)
19 - two distances: r(O..H) and r(N..O) (columns 7 and 8)
20 - planar (C-O..H) and dihedral (C-O..H-N) (columns 9 and 10)
21 - DSSP energy for this bond (column 11)
22 - X,Y, Z coordinates of H atom in the local coordinates system (columns 12, 13 and 14)
23 - theta, phi spherical coordinates of H atom (columns 15 and 16)
24
25 EXAMPLE OUTPUT:
26 # 42 hydrogen bonds found in backbone:
27 TYR      3 -> THR     18 : 2.620 3.346 165.58    94.25   -1.170    -0.702    0.211    2.515 ↵
28 →      16.25 163.29
29 LYS      4 -> LYS     50 : 2.259 3.156 120.71   159.88   -1.310    1.445    1.249    1.205 ↵
30 →      57.75 40.83
31 LEU      5 -> THR     16 : 1.838 2.802 143.84  -115.27   -2.834    0.102   -1.075    1.488 ↵
32 →      35.98 -84.57
33
34 USAGE:
35 ./ap_BackboneHBondMap input.pdb
36
37 )";
38
39 /** @brief Calculates a map of backbone hydrogen bonds.
40 *
41 * BackboneHBondMap is derived from PairwiseResidueMap class
42 *
43 * CATEGORIES: core/calc/structural/BackboneHBondMap;
44 * KEYWORDS: PDB input; Hydrogen bonds; data_structures; Protein structure features
45 * GROUP: Structure calculations;
46 * IMG: ap_BackboneHBondMap-2gb1.png
47 * IMG_ALT: Map of backbone hydrogen bonds for 2GB1 protein
48 */
49 int main(const int argc, const char* argv[]) {
50
51     if(argc < 2) utils::exit_OK_with_message(program_info);
52     →      // --- complain about missing program parameter

```

(continues on next page)

(continued from previous page)

```

52 core::data::io::Pdb reader(argv[1], is_not_alternative, only_ss_from_header, true);  

53 // --- Read in a PDB file  

54 core::data::structural::Structure_SP strctr = reader.create_structure(0); // ---  

55 // create a Structure object from the first model  

56  

57 // ----- Remove everything but amino acids from an input structure;  

58 // BackboneHBondMap currently can process only AA  

59 core::protocols::keep_selected_atoms(core::data::structural::selectors::IsAA{},  

60 *strctr);  

61  

62 // --- The line below creates a map of backbone hydrogen bonds; -0.2 is the energy  

63 // cutoff (in kcal/mol) to recognize the bond  

64 core::calc::structural::interactions::BackboneHBondMap hb_map(*strctr, -0.2);  

65 std::cout << "#" << hb_map.count_bonds() << " hydrogen bonds found in backbone:\n";  

66 for (auto h_it = hb_map.cbegin(); h_it != hb_map.cend(); ++h_it) // --- iterate  

67 over the bonds and print each of them  

68 std::cout << (*h_it).second->donor_residue() << " -> " << (*h_it).second->  

69 acceptor_residue() << " : " << (*h_it).second << "\n";  

70  

71 // --- Here we test some other BackboneHBondMap methods  

72 const auto hbond = (*hb_map.cbegin()).second; // --- here we make a local copy of  

73 // the first h-bond to be used in tests  

74 std::cerr << "# Is residue 0 h-bonded to residue 3? " << (hb_map.are_H_bonded(0,  

75 3)) ? "yes\n" : "no\n";  

76 std::cerr << "# Is residue 0 h-bonded to residue 5? " << (hb_map.at(0, 5) !=  

77 nullptr) ? "yes\n" : "no\n";  

78 std::cerr << "# Are " << hbond->donor_residue() << " and " << hbond->acceptor_<br/>  

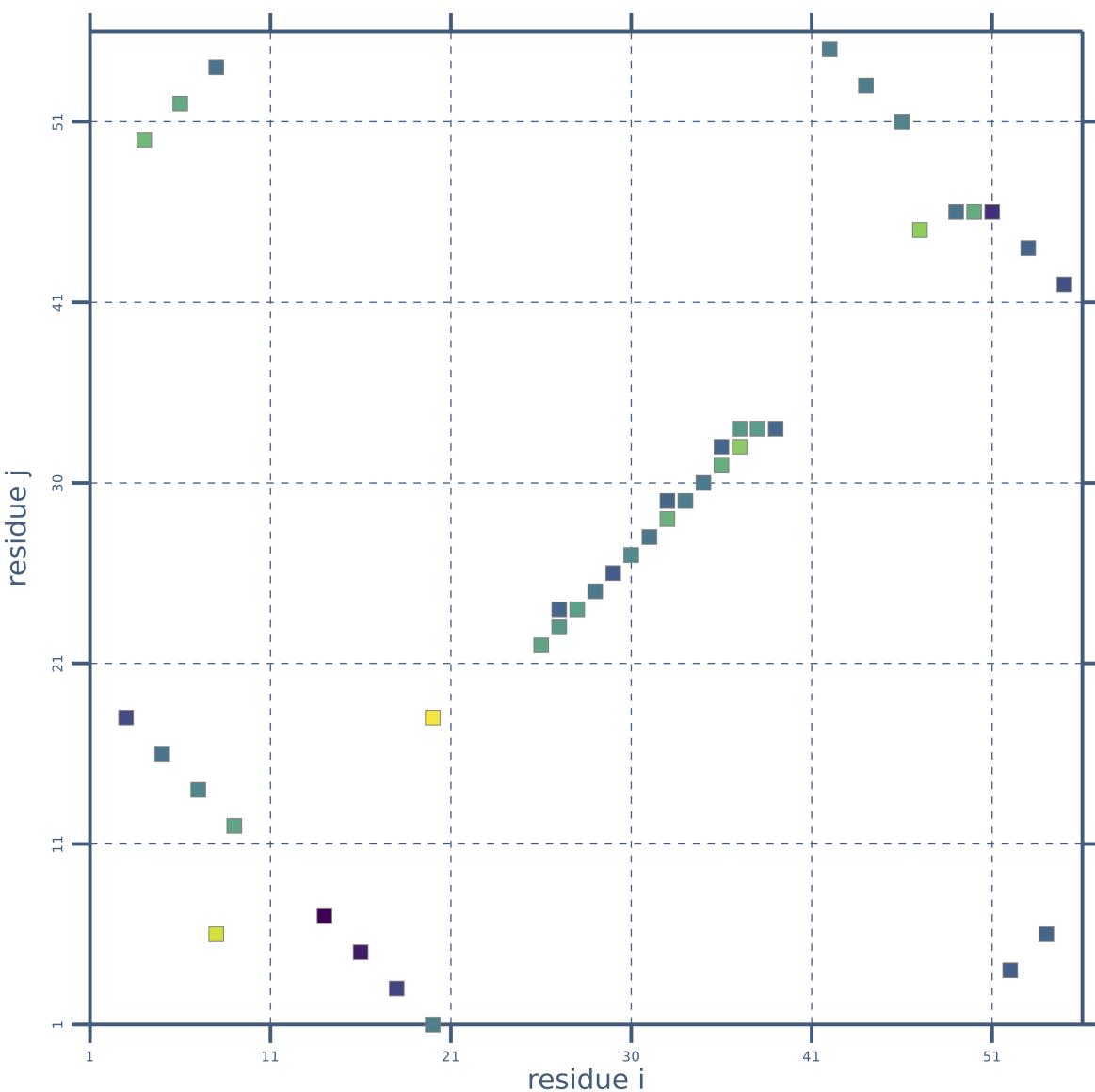
79 residue() << " really bonded? " <<  

80 (hb_map.at(*hbond->donor_residue(), *hbond->acceptor_residue()) != nullptr) ?  

81 "yes\n" : "no\n";  

82

```



ap_Crmsd

Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on C-alpha coordinates and prints it. If only one input PDB file is given, cRMSD is computed for every pair of models found in the input file (each-vs-each). If exactly two structures are provided, the program calculates cRMSD between the first model of structure A and the first model of structure B. Finally, when more than two input files are specified, each-vs-each calculations are performed for every pair of given structures. Note, that all the structures must contain the same number of C-alpha atoms.

USAGE:

```
./ap_Crmsd file1.pdb [file2.pdb ...]
```

EXAMPLEs:

```
./ap_Crmsd 1cey.pdb  
./ap_Crmsd 2gb1.pdb 2gb1-model1.pdb  
./ap_Crmsd 2gb1-model1.pdb 2gb1-model2.pdb 2gb1-model3.pdb 2gb1-model4.pdb
```

REFERENCE: Kabsch, W. "A Solution for the Best Rotation to Relate Two Sets of Vectors." Acta Cryst (1976) 32 922-923

Keywords:

- *PDB input*
- *crmsd*

Categories:

- core/calc/structural/transformations/Crmsd

Input files:

- 2gb1-model3.pdb
- 2gb1-model1.pdb
- 2gb1-model4.pdb
- 2gb1.pdb
- 1cey.pdb
- 2gb1-model2.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>  
2  
3 #include <core/data/io/Pdb.hh>  
4 #include <core/calc/structural/transformations/Crmsd.hh>  
5  
6 #include <utils/exit.hh>  
7  
8 std::string program_info = R"(  
9  
10 Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on C-alpha coordinates  
11 and prints it.  
12 If only one input PDB file is given, cRMSD is computed for every pair of models found  
13 in the input  
file (each-vs-each). If exactly two structures are provided, the program calculates  
cRMSD between  
the first model of structure A and the first model of structure B. Finally, when more  
than two input
```

(continues on next page)

(continued from previous page)

```

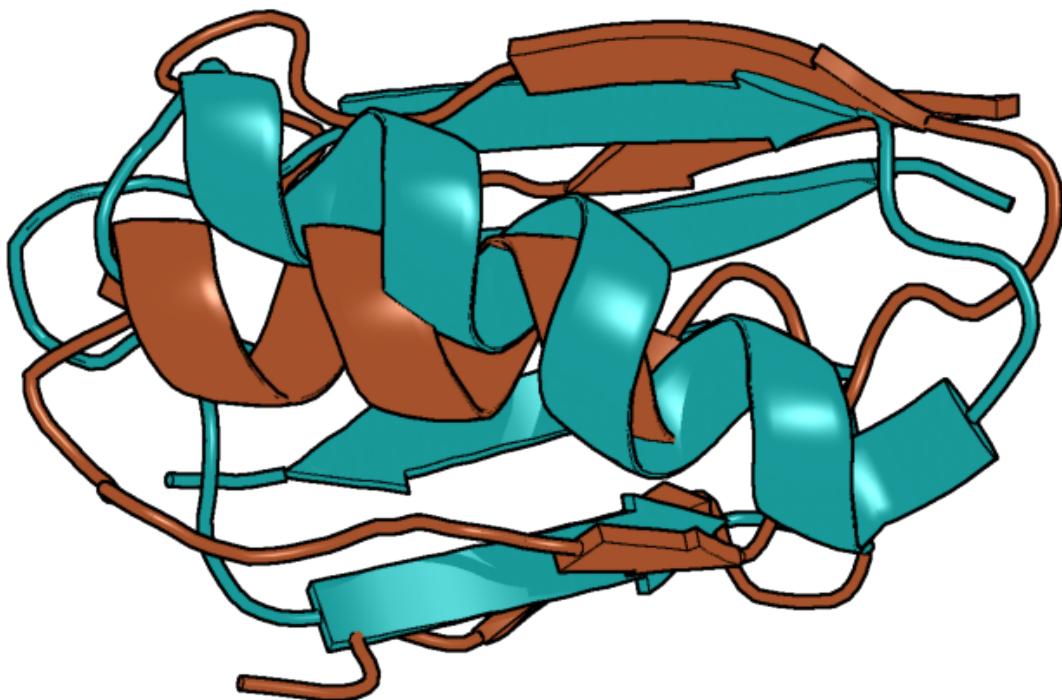
14 files are specified, each-vs-each calculations are performed for every pair of given_
15 ↪structures.
16 Note, that all the structures must contain the same number of C-alpha atoms.
17
18 USAGE:
19 ./ap_Crmsd file1.pdb [file2.pdb ...]
20
21 EXAMPLES:
22 ./ap_Crmsd 1cey.pdb
23 ./ap_Crmsd 2gb1.pdb 2gb1-model1.pdb
24 ./ap_Crmsd 2gb1-model1.pdb 2gb1-model2.pdb 2gb1-model3.pdb 2gb1-model4.pdb
25
26 REFERENCE:
27 Kabsch, W. "A Solution for the Best Rotation to Relate Two Sets of Vectors."
28 Acta Cryst (1976) 32 922-923
29
30 ) ";
31
32 /** @brief Calculates crmsd value on C-alpha coordinates. The program prints just the_
33 ↪crmsd value.
34 *
35 * CATEGORIES: core/calc/structural/transformations/Crmsd
36 * KEYWORDS: PDB input; crmsd
37 * GROUP: Structure calculations;
38 * IMG: ap_Crmsd_deepteal_brown_1.png
39 * IMG_ALT: 2GB1 model structure superimposed on the native, crmsd = 4.93952
40 */
41 int main(const int argc, const char *argv[]) {
42
43     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
44     ↪missing program parameter
45
46     using core::data::basic::Vec3;
47     using namespace core::calc::structural::transformations;
48
49     Crmsd<std::vector<Vec3>, std::vector<Vec3>> rms;
50
51     if(argc==2) { // --- The case of each-vs-each calculations between models of a_
52     ↪single PDB file
53         core::data::io::Pdb q_reader(argv[1], core::data::io::is_ca, core::data::io::keep_
54         ↪all, false);
55         core::data::structural::Structure_SP q_strctr = q_reader.create_structure(0); // -
56         ↪-- create a structure object
57
58         std::vector<std::vector<core::data::basic::Vec3>> models(q_reader.count_models());
59         for(int i=0; i<q_reader.count_models(); ++i) {
60             models[i].resize(q_strctr->count_atoms());
61             q_reader.fill_structure(i, models[i]);
62             for (int j = 0; j < i; ++j)
63                 std::cout << i << " " << j << " " << rms.crmsd(models[i], models[j],
64             ↪size()) << "\n";
65         }
66     } else { // --- The case when two PDB files are given
67         core::data::io::Pdb q_reader(argv[1], core::data::io::is_ca, core::data::io::keep_
68         ↪all, false);
69         core::data::structural::Structure_SP q_strctr = q_reader.create_structure(0); // -
70         ↪-- create a structure object

```

(continues on next page)

(continued from previous page)

```
63
64     core::data::io::Pdb t_reader(argv[2], core::data::io::is_ca, core::data::io::keep_
65     ↪all, false);
66     core::data::structural::Structure_SP t_strctr = t_reader.create_structure(0); // -
67     ↪-- create a structure object
68
69     if (q_strctr->count_atoms() != t_strctr->count_atoms())
70         utils::exit_OK_with_message("The two structures have different number of CA_
71     ↪atoms!\n");
72
73     std::vector<Vec3> q, t;
74     for (auto atom_it = q_strctr->first_atom(); atom_it != q_strctr->last_atom(); ↪
75     ↪++atom_it) q.push_back(**atom_it);
76     for (auto atom_it = t_strctr->first_atom(); atom_it != t_strctr->last_atom(); ↪
77     ↪++atom_it) t.push_back(**atom_it);
78     std::cout << "crmsd: " << rms.crmsd(q, t, q_strctr->count_atoms()) << "\n";
79 }
80 }
```



ap_Hexbins

Reads a file with 2D observations (two columns with real values) and makes hexbin histogram.

USAGE:

```
ap_Hexbins input.dat [bin_side_width]
```

Keywords:

- histogram
- *statistics*

Categories:

- core::calc::statistics::Hexbins

Input files:

- LEU_chi1_chi2_rad.dat

Output files:

- LEU_rotamers.dat
- stdout.out
- normal_2d.dat

Program source:

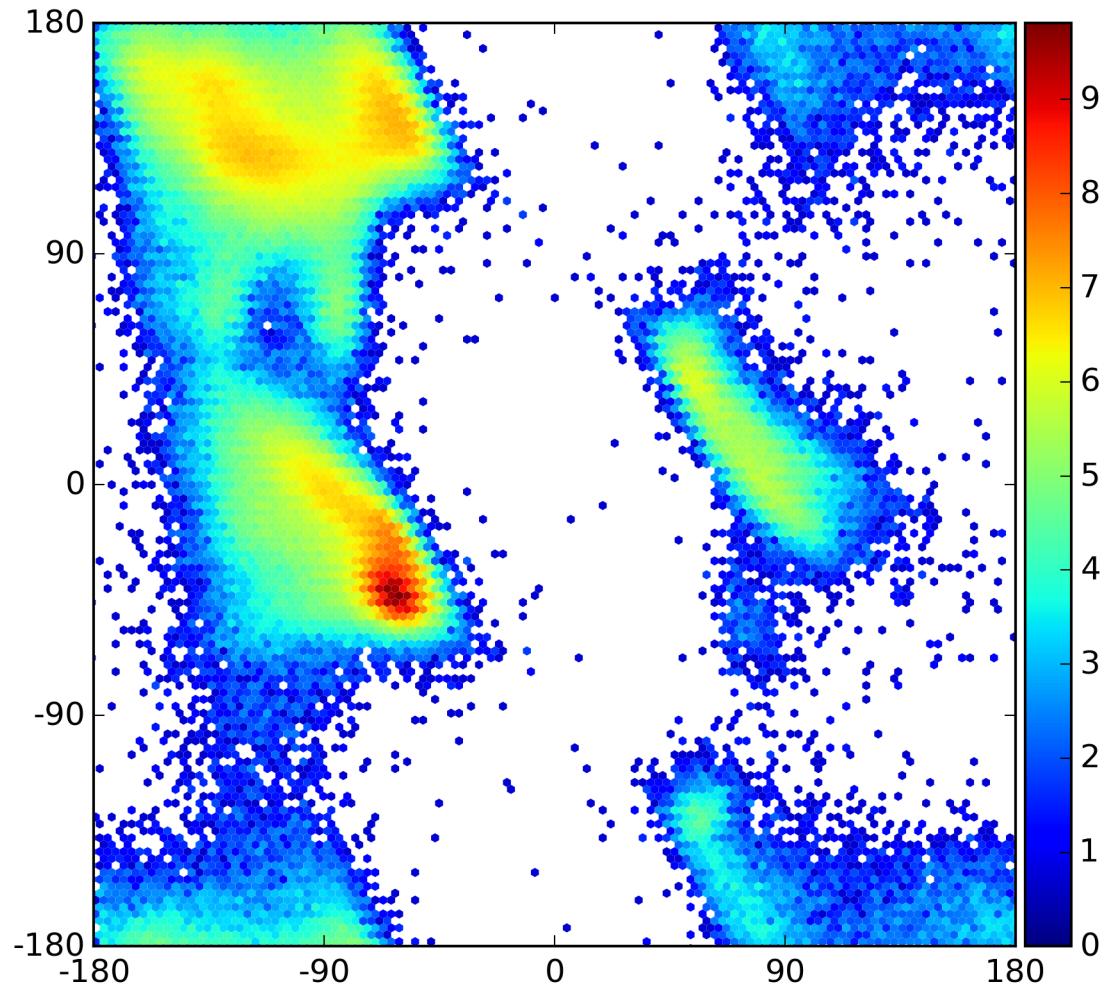
```
1 #include <iostream>
2 #include <random>
3 #include <vector>
4
5 #include <core/index.hh>
6
7 #include <core/calc/statistics/Hexbins.hh>
8 #include <core/calc/statistics/Random.hh>
9 #include <core/data/io/DataTable.hh>
10
11 std::string program_info = R"(
12
13 Reads a file with 2D observations (two columns with real values) and makes hexbin_
14 ↴ histogram.
15 USAGE:
16     ap_Hexbins input.dat [bin_side_width]
17 )";
18
19 /** @brief Reads a file with 2D observations (two columns) and makes hexbin histogram.
20 *
21 * CATEGORIES: core::calc::statistics::Hexbins
22 * KEYWORDS: histogram; statistics
23 * GROUP: Statistics;
24 * IMG: ramachandran_map_all.png
25 * IMG_ALT: hexabin representation of Ramachandran map (histogram made from non-
26 ↴redundant subset of PDB)
27 */
28 int main(const int argc, const char *argv[]) {
29
30     using namespace core::calc::statistics;
31
32     Hexbins<double, core::index4> hist(0.05);
33     if (argc > 1) { // --- If an input file was given, make histogram using this data
34         if (argc > 2) hist.bin_side(atof(argv[2]));
35         float x,y;
36         std::ifstream in(argv[1]);
37 }
```

(continues on next page)

(continued from previous page)

```

36     std::string line;
37     // --- here we read the input file using pure C API since it's faster than C++_
38     //→fancy streams
39     while (std::getline(in, line)) {
40         sscanf(line.c_str(), "%f %f", &x, &y);
41         hist.insert(x,y);
42     }
43 } else { // --- otherwise generate some random data
44     std::cerr << program_info << "\n";
45     Random r = Random::get();
46     r.seed(9876543);
47     NormalRandomDistribution<double> dist_x(1.0, 0.25);
48     NormalRandomDistribution<double> dist_y(3.0, 0.5);
49     for (size_t i = 0; i < 100000; ++i) {
50         hist.insert(dist_x(r), dist_y(r));
51     }
52     std::cout << "# Created histogram of " << hist.count_entries() << " observations, "
53     //→<< hist.count_outside()
54     //→      << " were outside\n";
55
56     std::vector<std::pair<double,double>> coordinates; // --- a vector used to retrieve_
57     //→coordinates of each hexagon
58
59     for (auto it = hist.cbegin(); it != hist.cend(); ++it) {
60         coordinates.clear();
61         auto bin = (*it).first;
62         hist.bin_vertices(bin,coordinates);
63         std::cout << utils::string_format("%4d %4d %4d ",bin.first, bin.second,(*it).
64         //→second);
65 // --- uncomment the lines below to print coordinates of hexbin vertexes in every line
66 // --- Note: this is a lot of (redundant) output; make_plots.py script may generate_
67 //→these coordinates for you based on bin indexes
68 //     std::cout << " : ";
69 //     for(const auto & xy : coordinates) std::cout << utils::string_format("%8.3f %8.
70 //→3f ",xy.first,xy.second);
71     std::cout << "\n";
72 }
73 }
```



ap_LigandTossingMover

The program creates a multimodel PDB with random orientations of a ligand in respect to the protein.

USAGE:

```
ap_LigandTossingMover 2kwi.pdb GNP [30]
```

where 2kwi.pdb is the name of the input PDB file, GNP is the code of the ligand (must be in the same PDB file) and 30 is the number of random conformations generated (optional argument)

Keywords:

- *docking*
- *Mover*

Categories:

- simulations::movers::LigandTossingMover

Input files:

- 2kwi.pdb

Output files:

- stdout.out
- out.pdb

Program source:

```

1 #include <core/data/basic/Vec3.hh>
2 #include <core/data/io/Pdb.hh>
3 #include <simulations/movers/LigandTossingMover.hh>
4 #include <simulations/forcefields/ConstEnergy.hh>
5 #include <simulations/systems/PdbAtomTyping.hh>
6 #include <simulations/observers/cartesian/PdbObserver_OBSOLETE.hh>
7 #include <simulations/observers/cartesian/ExplicitPdbFormatter_OBSOLETE.hh>
8 #include <simulations/sampling/AlwaysAccept.hh>
9
10 using namespace core::data::basic;
11 using namespace simulations;
12
13 #include <utils/exit.hh>
14
15 std::string program_info = R"(
16
17 The program creates a multimodel PDB with random orientations of a ligand in respect_
18 ↪ to the protein.
19
20 USAGE:
21     ap_LigandTossingMover 2kwi.pdb GNP [30]
22 where 2kwi.pdb is the name of the input PDB file, GNP is the code of the ligand (must_
23 ↪ be in the same PDB file)
24 and 30 is the number of random conformations generated (optional argument)
25 )";
26
27 /**
28 * @brief To test LigandTossingMover mover, tosses a ligand on a proteins surface
29 *
30 * The program creates a multimodel PDB with random orientations of a ligand in_
31 ↪ respect to the protein
32 *
33 * CATEGORIES: simulations::movers::LigandTossingMover
34 * KEYWORDS: docking; Mover
35 * IMG: ap_LigandTossingMover.png
36 * IMG_ALT: 25 conformations of the same ligand randomly placed on the surface of a_
37 ↪ protein
38 */
39 int main(int argc, char *argv[]) {
40
41     using namespace simulations::systems; // for AtomTypingInterface and ResidueChain
42     using namespace simulations::observers::cartesian;
43     using namespace core::data::io; // for Pdb reader

```

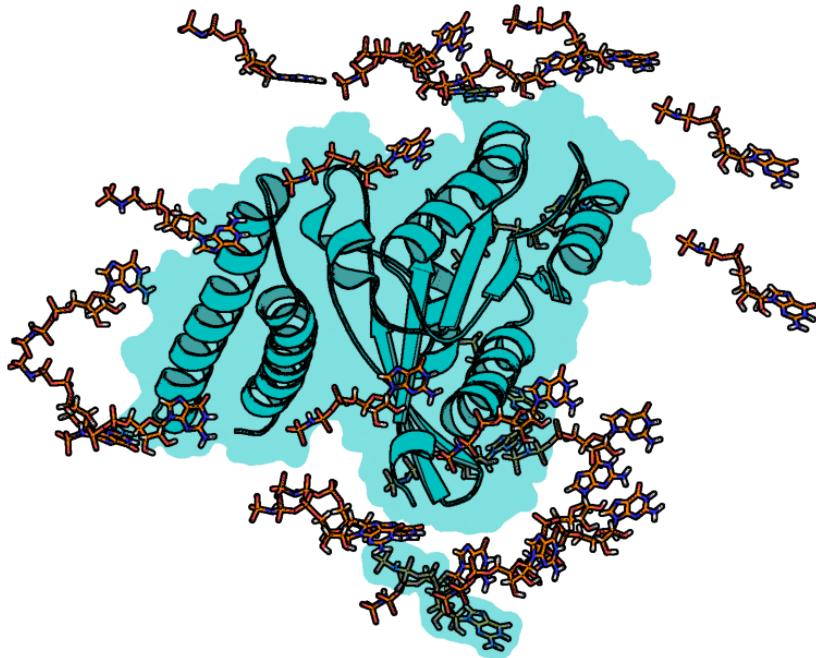
(continues on next page)

(continued from previous page)

```

41  using core::data::basic::Vec3;
42
43  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about
44  // missing program parameter
45
46  // --- Read the input PDB and create a structure object
47  // Pdb reader(argv[1], all_true(is_not_hydrogen, is_not_water, is_not_alternative),_
48  // true);
49  Pdb reader(argv[1]);
50  core::data::structural::Structure_SP strctr = reader.create_structure(0);
51
52  // --- Prepare a modeled system from a given PDB file
53  ResidueChain_OBSOLETE<Vec3> system(*strctr);
54
55  std::string ligand_code(argv[2]);
56  int from = -1, to = -1, i = 0;
57  for (auto atom_it = strctr->first_atom(); atom_it != strctr->last_atom(); ++atom_
58  it) {
59    if (ligand_code.size() == 3) { // truly it's a ligand code
60      if (((*atom_it)->owner()->residue_type().code3 == ligand_code) && (from == -1))_
61      from = i;
62      if (((*atom_it)->owner()->residue_type().code3 != ligand_code) && (to == -1) &&_
63      (from != -1)) to = i - 1;
64    } else { // it should be a chain code then
65      if (((*atom_it)->owner()->owner()->id() == ligand_code) && (from == -1)) from_=
66      i;
67      if (((*atom_it)->owner()->owner()->id() != ligand_code) && (to == -1) &&_
68      (from != -1)) to = i - 1;
69    }
70    ++i;
71  }
72  if (to == -1) to = i - 1; // --- assign the last atom if nothing has been assigned_
73  yet
74  AtomRange moving(from,to);
75  std::cout << "Moving atoms: " << moving << "\n";
76
77  core::index4 n_moves = (argc==4) ? atoi(argv[3]) : 10;
78  simulations::movers::LigandTossingMover<Vec3> mover(system, moving, 5.0);
79
80  simulations::sampling::AlwaysAccept alwaysMove;
81  std::shared_ptr<AbstractPdbFormatter_OBSOLETE<Vec3>> fmt = std::make_shared_
82  <ExplicitPdbFormatter_OBSOLETE<Vec3>>(*strctr);
83
84  simulations::observers::cartesian::PdbObserver_OBSOLETE<Vec3> trajectory(system,_
85  fmt, "out.pdb");
86  for (size_t i = 0; i < n_moves; i++) {
87    mover.move(alwaysMove);
88    trajectory.observe();
89  }
90
91

```



ap_aligned_pdb

Reads an alignment between two proteins (PIR format) and the two respective protein structures (PDB format) and writes the aligned parts of the two structures. The program concerns only the first two sequences found in the PIR file; they must be given in the same order as the input PDB files. Only the first chain will be used from either structure; if you want to use chain 'B', from a structure, use strc command to extract it prior using ap_aligned_pdb. The program writes 'query' and 'tmplt' files which contain the respective structure fragments, already superimposed (the template on the query). One of the two structures (either the query or the template) may be missing, e.g. in a case of gene duplication dash '-' should be used instead of the respective file name, as in the examples below.

USAGE:

```
ap_aligned_pdb alignment.pir prot1.pdb prot2.pdb
```

EXAMPLE:

```
ap_aligned_pdb luox_luox_1.pir luox.pdb -
ap_aligned_pdb luox_luox_1.pir - luox.pdb
```

Keywords:

- *PDB input*
- *PIR*
- *PDB output*

Categories:

- core/data/io/pir_io

Input files:

- 1uox.pdb
- 1uox_1uox_1.pir

Output files:

- query.pdb
- stdout.out
- tmplt.pdb

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/pir_io.hh>
4 #include <core/data/io/Pdb.hh>
5 #include <core/data/io/alignment_io.hh>
6 #include <core/data/io/faasta_io.hh>
7
8 #include <utils/exit.hh>
9 #include <core/alignment/PairwiseSequenceAlignment.hh>
10 #include <core/data/structural/selectors/structure_selectors.hh>
11 #include <core/calc/structural/transformations/Crmsd.hh>
12 #include <utils/Logger.hh>
13
14 std::string program_info = R"(
15
16 Reads an alignment between two proteins (PIR format) and the two respective protein_
17 ↪structures (PDB format)
18 and writes the aligned parts of the two structures.
19
20 The program concerns only the first two sequences found in the PIR file; they must be_
21 ↪given in the same order
22 as the input PDB files. Only the first chain will be used from either structure; if_
23 ↪you want to use chain 'B',
24 from a structure, use strc command to extract it prior using ap_aligned_pdb.
25
26 The program writes 'query' and 'tmplt' files which contain the respective structure_
27 ↪fragments, already
28 superimposed (the template on the query). One of the two structures (either the query_
29 ↪or the template)
may be missing, e.g. in a case of gene duplication dash '-' should be used instead of_
↪the respective
file name, as in the examples below.
30
31 USAGE:
32     ap_aligned_pdb alignment.pir prot1.pdb prot2.pdb
```

(continues on next page)

(continued from previous page)

```

30
31 EXAMPLE:
32     ap_aligned_pdb luox_luox_1.pir luox.pdb -
33     ap_aligned_pdb luox_luox_1.pir - luox.pdb
34
35 );
36
37 using namespace core::data::structural;
38 utils::Logger logs("ap_aligned_pdb");
39
40 Structure_SP process_input_pdb(const std::string &pdb_fname, std::vector<Residue_SP> &
41     residues,
42             const selectors::ResidueSelector & which_part) {
43
44     selectors::IsAA aa_only;
45     // --- Read the first structure and repack its amino acid residues (the other
46     // cannot be aligned)
47     core::data::io::Pdb reader(pdb_fname, core::data::io::is_not_alternative,_
48     core::data::io::only_ss_from_header, true);
49     Structure_SP strctr = reader.create_structure(0);
50
51     logs << utils::LogLevel::INFO << "Selecting " << utils::to_string(which_part) << "
52     //from "<<strctr->code()<<"\n";
53     for(auto i_chain : *strctr) {
54         for(auto i_resid : *i_chain)
55             if (which_part(*i_resid) && aa_only(*i_resid)) {
56                 residues.push_back(i_resid);
57             }
58     }
59
60     return strctr;
61 }
62
63 /**
64  * @brief Reads an alignment between two proteins (PIR format) and the two
65  * structures and writes PDB for the aligned parts
66  */
67 int main(const int argc, const char *argv[]) {
68
69     if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about
70     //missing program parameters
71
72     using core::data::sequence::Sequence_SP; // --- Sequence_SP is just a std::shared_
73     //ptr to core::data::sequence::Sequence type
74     using namespace core::alignment;
75     using namespace core::data::structural;
76
77     // --- Create a container where the sequences will be stored
78     std::vector<Sequence_SP> sequences;
79
78     // --- Read a file with PIR sequences and create an alignment object
79     core::data::io::read_pir_file(argv[1], sequences);

```

(continues on next page)

(continued from previous page)

```

80     PairwiseSequenceAlignment alignment("query", sequences[0]->sequence, 0, "tmplt",
81     ↪sequences[1]->sequence, 0, 0.0);
82
83     const auto select_query = core::data::structural::selectors::select_by_pir_
84     ↪header(*std::dynamic_pointer_cast<PirEntry>(sequences[0]));
85     const auto select_tmplt = core::data::structural::selectors::select_by_pir_
86     ↪header(*std::dynamic_pointer_cast<PirEntry>(sequences[1]));
87
88     // --- Read the first structure and repack its amino acid residues (the other_
89     ↪cannot be aligned)
90     Structure_SP query_structure, tmplt_structure;
91     std::vector<Residue_SP> query_residues, tmplt_residues;
92     if (strcmp(argv[2], "-", 1) != 0) query_structure = process_input_pdb(argv[2],
93     ↪query_residues,*select_query);
94     if (strcmp(argv[3], "-", 1) != 0) tmplt_structure = process_input_pdb(argv[3],
95     ↪tmplt_residues,*select_tmplt);
96
97     std::stringstream ss;
98     core::data::io::write_edinburgh(alignment,ss,65535);
99     logs << utils::LogLevel::INFO << "Input alignment\n" << ss.str() << "\n";
100
101    // --- Retrieve aligned residues from the two structures according to the alignment_
102    ↪object
103    std::vector<Residue_SP> tmplt_residues_aligned, query_residues_aligned; // ---
104    ↪container for the residues
105
106    if (query_residues.size() == 0) alignment.alignment->get_aligned_template(tmplt_
107    ↪residues, tmplt_residues_aligned);
108    if (tmplt_residues.size() == 0) alignment.alignment->get_aligned_query(query_
109    ↪residues, query_residues_aligned);
110
111    // --- If both sets of coordinates are present - retrieve both and superimpose
112    // --- Also, when both structures are given - calculate crmsd and roto-translation_
113    ↪transformation
114    if ((query_residues.size() > 0) && (tmplt_residues.size() > 0)) {
115        alignment.alignment->get_aligned_query_template(query_residues, tmplt_residues,
116        ↪query_residues_aligned, tmplt_residues_aligned);
117        std::vector<Vec3> query_xyz, tmplt_xyz;
118        for (auto res:query_residues_aligned) query_xyz.push_back(*res->find_atom(" CA
119        ↪"));
120        for (auto res:tmplt_residues_aligned) {
121            tmplt_xyz.push_back(*res->find_atom(" CA "));
122        }
123        core::calc::structural::transformations::Crmsd<std::vector<Vec3>, std::vector
124        ↪<Vec3>> rms;
125        std::cout << "crmsd between coordinates of " << query_xyz.size() << " CA atoms: "
126        <<
127            rms.crmsd(tmplt_xyz, query_xyz, query_xyz.size(), true) << "\n";
128        for (auto res:tmplt_residues_aligned) for (auto atom:*res) rms.apply(*atom);
129    }
130
131    // --- Rotate the query coordinates and superimpose them on the template; print_
132    ↪them in PDB format
133    if (query_residues_aligned.size() > 0) {
134        std::ofstream query_file("query.pdb");
135        for (auto res:query_residues_aligned)
136            for (auto atom:*res) query_file << atom->to_pdb_line() << "\n";

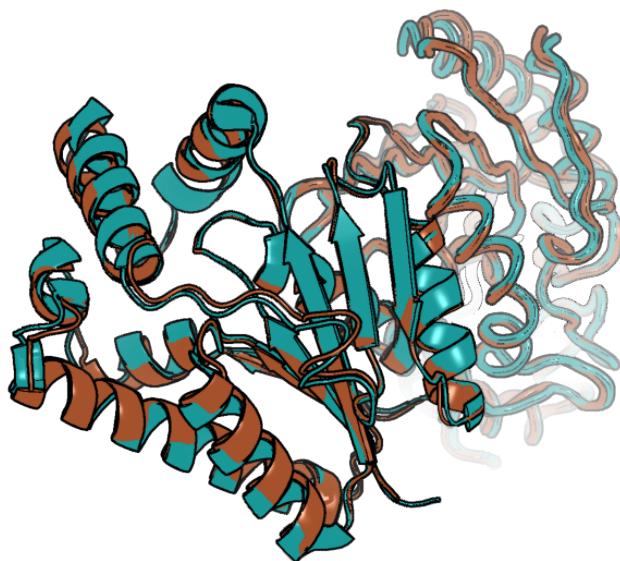
```

(continues on next page)

(continued from previous page)

```

121     query_file.close();
122 }
123 // --- Print template coordinates in PDB format
124 if (tmplt_residues_aligned.size() > 0) {
125     std::ofstream tmplt_file("tmplt.pdb");
126     for (auto res:tmplt_residues_aligned)
127         for (auto atom:*res) tmplt_file << atom->to_pdb_line() << "\n";
128     tmplt_file.close();
129 }
130 }
```



ap_chi1_rotamers_estimation

ap_chi1_rotamers_estimation reads a text file with Chi_1 angles (single column of real values) and fits a mixture of VonMisses distributions to the data. The program may be thus used for deriving rotamer library for VAL, THR, SER and CYS

USAGE:

```
ap_chi1_rotamers_estimation input-data
```

EXAMPLE:

```
ap_chi1_rotamers_estimation THR_chi1.dat
```

REFERENCE: Mardia, Kanti V., and Peter E. Jupp. Directional statistics. Vol. 494. John Wiley & Sons, 2009

Keywords:

- von Misses distribution

- *estimation*
- *expectation-maximization*
- *statistics*

Categories:

- core::calc::statistics::VonMissesDistribution

Input files:

- THR_chi1.dat

Output files:

- stdout.out

Program source:

```
1 #include <math.h>
2
3 #include <iostream>
4 #include <random>
5
6 #include <core/calc/statistics/VonMissesDistribution.hh>
7 #include <core/calc/statistics/expectation_maximization.hh>
8 #include <core/data/io/DataTable.hh>
9 #include <core/calc/numeric/numerical_integration.hh>
10 #include <utils/exit.hh>
11
12 std::string program_info = R"(
13
14 ap_chi1_rotamers_estimation reads a text file with Chi_1 angles (single column of_
15 ↪real values)
16 and fits a mixture of VonMisses distributions to the data. The program may be thus_
17 ↪used for deriving
18 rotamer library for VAL, THR, SER and CYS
19
20 USAGE:
21     ap_chi1_rotamers_estimation input-data
22
23 EXAMPLE:
24     ap_chi1_rotamers_estimation THR_chi1.dat
25
26 REFERENCE:
27 Mardia, Kanti V., and Peter E. Jupp. Directional statistics. Vol. 494. John Wiley &_
28 ↪Sons, 2009
29
30 )";
31
32 /**
33  * @brief Reads a file with 1D data and estimates a mixture of VonMissesDistribution_
34  * based on these observations.
35 *
```

(continues on next page)

(continued from previous page)

```

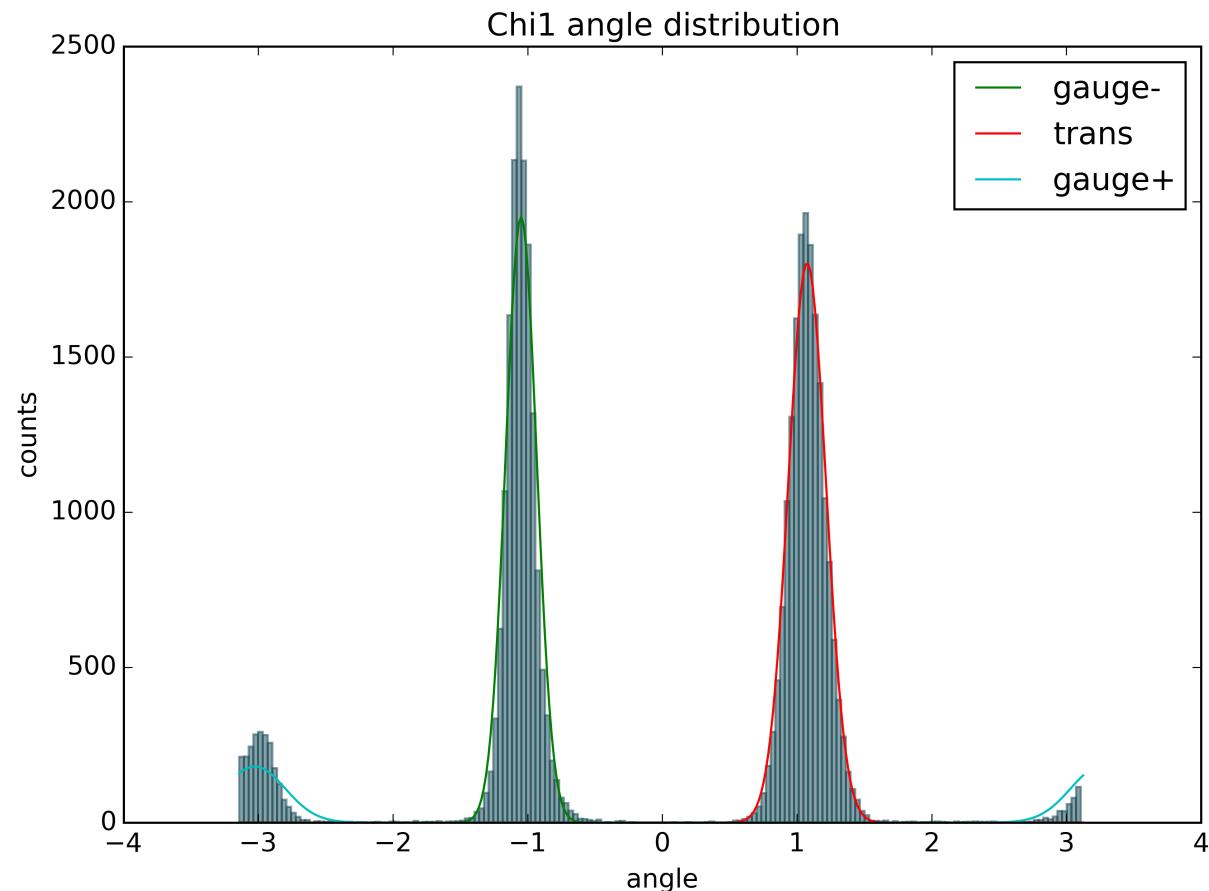
31 * This example may be used to approximate a $Chi_1$ rotamer (such as VAL, THR or
32 ←SER) with a mixture of
33 * VonMisses distributions.
34 *
35 * CATEGORIES: core:::calc:::statistics:::VonMissesDistribution
36 * KEYWORDS: von Misses distribution; estimation; expectation-maximization;
37 ←statistics
38 * GROUP: Statistics;
39 * IMG: ap_chil_rotamers_estimation.png
40 * IMG_ALT: Distribution of Chil angles of THR side chains approximated with a
41 ←mixture of three von Mises distribution
42 */
43 int main(const int argc, const char *argv[]) {
44
45     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
46 ←missing program parameter
47     const double deg_to_rad = M_PI/180.0;
48     using namespace core:::calc:::statistics;
49
50     std::vector<std::vector<double>> chi_angle;
51     if (argc == 2) {
52         core:::data::io:::DataTable in(argv[1]);
53         for(const auto & row : in) {
54             std::vector<double> chi;
55             chi.push_back( row.get<double>(0) );
56             chi_angle.push_back(chi);
57         }
58     }
59     // ----- Three distributions - one for each rotamer
60     VonMisesDistribution m(-60 * deg_to_rad, 10.0), p(60 * deg_to_rad, 10.0),
61     t(-180 * deg_to_rad, 10.0); // medium, gauge-plus and gauge-minus
62
63     std::vector<VonMisesDistribution> distributions_1D({{-60 * deg_to_rad, 10.0}, // ---
64 ← gauge minus
65                                     {60 * deg_to_rad, 10.0}, // ---
66 ← gauge plus
67                                     {-180 * deg_to_rad, 10.0}}); // ---
68
69     --- trans
70     std::vector<core:::index1> index_1D;
71     double score = expectation_maximization(chi_angle, distributions_1D, index_1D, 0.
72 ←000001);
73     core:::index4 cnt0 = std::count(index_1D.cbegin(), index_1D.cend(), 0);
74     core:::index4 cnt1 = std::count(index_1D.cbegin(), index_1D.cend(), 1);
75     core:::index4 cnt2 = std::count(index_1D.cbegin(), index_1D.cend(), 2);
76     std::cout << "# log-likelihood: " << score << "\n";
77     std::cout << "# " << cnt0 << " " << distributions_1D[0]
78     << " " << cnt1 << " " << distributions_1D[1]
79     << " " << cnt2 << " " << distributions_1D[2] << "\n";
80
81     for (double x = -M_PI; x < M_PI; x += 0.01)
82         std::cout << utils::string_format("%6.3f %8.5f %8.5f %8.5f\n", x,
83                                         cnt0 * distributions_1D[0].evaluate(x) / chi_
84 ←angle.size(),
85                                         cnt1 * distributions_1D[1].evaluate(x) / chi_
86 ←angle.size(),
87                                         cnt2 * distributions_1D[2].evaluate(x) / chi_
88 ←angle.size());

```

(continues on next page)

(continued from previous page)

77



ap_contact_map

ap_contact_map calculates a contact map for a given protein structure. If a multi-model PDB file was given, the program prints for every contact in how many models the contact was observed. The program can calculate the contacts either on side chains, on alpha carbon or on beta carbon atoms.

USAGE:

```
ap_contact_map atom-filter input.pdb cutoff
```

EXAMPLE:

```
ap_contact_map CA 2kwi.pdb 4.5
```

where 2kwi.pdb is the input file and 4.5 the contact distance in Angstroms. CA defines the contact map type; allowed options are: CA CB and SC for C-alpha, C-beta and all atom side chain, respectively

Keywords:

- *PDB input*

- *contact map*

Categories:

- core::calc::structural::ContactMap

Input files:

- 2kwi.pdb
- 1cey.pdb

Output files:

- stdout.out
- 1cey.sc.cmap
- 2kwi.sc.cmap
- 2kwi.ca.cmap
- 2kwi.cb.cmap

Program source:

```

1 #include <iostream>
2
3 #include <core/index.hh>
4 #include <core/data/io/Pdb.hh>
5 #include <core/calc/structural/interactions/ContactMap.hh>
6 #include <core/data/structural/selectors/structure_selectors.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(

10 ap_contact_map calculates a contact map for a given protein structure
11
12 If a multi-model PDB file was given, the program prints for every contact in how many ↴
13 ↴models
14 the contact was observed. The program can calculate the contacts either on side ↴
15 ↴chains,
16 on alpha carbon or on beta carbon atoms.

17 USAGE:
18     ap_contact_map atom-filter input.pdb cutoff
19
20 EXAMPLE:
21     ap_contact_map CA 2kwi.pdb 4.5
22
23 where 2kwi.pdb is the input file and 4.5 the contact distance in Angstroms. CA ↴
24 ↴defines the contact map type;
25 allowed options are: CA CB and SC for C-alpha, C-beta and all atom side chain, ↴
    ↴respectively

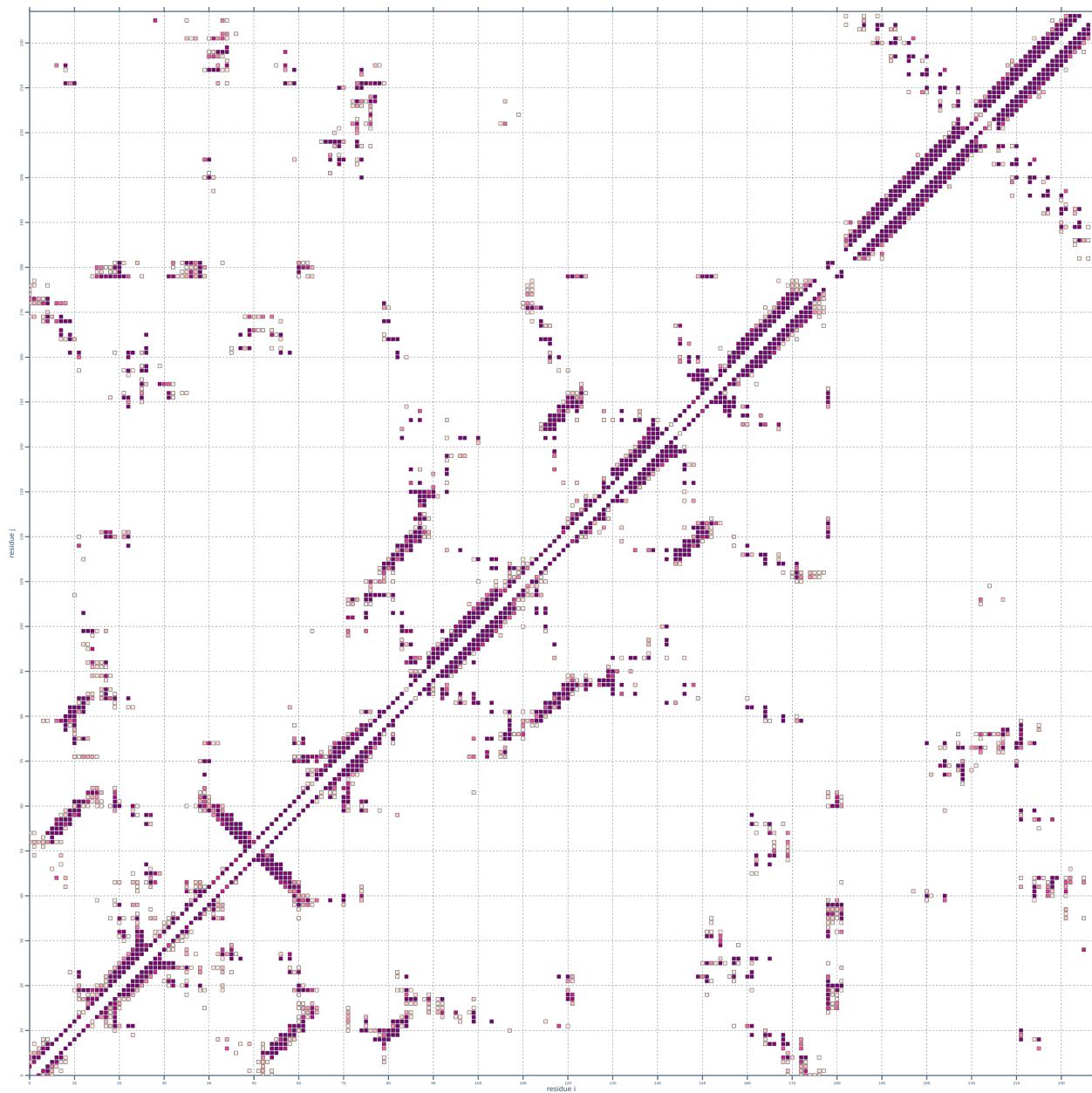
```

(continues on next page)

(continued from previous page)

```

26 ) ";
27
28 /** @brief Calculates a contact map for a given protein structure
29 *
30 * CATEGORIES: core:::calc::structural::ContactMap
31 * KEYWORDS: PDB input; contact map
32 * GROUP: Structure calculations;
33 * IMG: ap_contact_map.png
34 * IMG_ALT: Contact map calculated for 2KWI protein structure solved by NMR. The 2KWI_
35 ↪deposit holds 51 models, the color scale shows how popular is a given contact among_
36 ↪the models
37 */
38 int main(const int argc, const char* argv[]) {
39
40     if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about_
41 ↪missing program parameter
42
43     core:::data::structural::selectors::AtomSelector_SP selector
44         = std::make_shared<core:::data::structural::selectors::IsSC>();
45     core:::data::io:::PdbLineFilter filter = core:::data::io:::is_not_water;
46     if (std::strcmp(argv[1], "CA") == 0) {
47         selector = std::make_shared<core:::data::structural::selectors::IsNamedAtom>(" CA
48 ↪");
49         core:::data::io:::PdbLineFilter filter = core:::data::io:::is_ca;
50     }
51     if (std::strcmp(argv[1], "CB") == 0) {
52         core:::data::io:::PdbLineFilter filter = core:::data::io:::is_cb;
53         selector = std::make_shared<core:::data::structural::selectors::IsNamedAtom>(" CB
54 ↪");
55     }
56
57     double cutoff = utils::from_string<double>(argv[3]); // The third parameter is the_
58 ↪contact distance (in Angstroms)
59     core:::data::io:::Pdb reader(argv[2], filter); // --- file name (PDB format, may be_
60 ↪gzip-ped)
61
62     core:::data::structural::Structure_SP structure = reader.create_structure(0);
63     core:::calc::structural::interactions::ContactMap cmap(*structure, cutoff, selector);
64     for (int i_model = 1; i_model < reader.count_models(); ++i_model) {
65         reader.fill_structure(i_model, *structure);
66         cmap.add(*structure);
67     }
68
69     std::vector<std::pair<core:::index2, core:::index2>> contacts;
70     cmap.nonempty_indexes(contacts);
71
72     for(const std::pair<core:::index2,core:::index2> ij : contacts) {
73         core:::index2 i_res = ij.first;
74         core:::index2 j_res = ij.second;
75         std::cout << utils::string_format("%4d %4s %4d%c %4d %4s %4d%c %d\n", i_res,
76             cmap.residue_index(i_res).chain_id.c_str(),
77             cmap.residue_index(i_res).residue_id, cmap.residue_index(i_res).i_code,
78             j_res, cmap.residue_index(j_res).chain_id.c_str(),
79             cmap.residue_index(j_res).residue_id, cmap.residue_index(j_res).i_code,
80             cmap.at(i_res, j_res, 0));
81     }
82 }
```



ap_fit_VonMises_mixture

ap_fit_VonMises_mixture reads a text file with 1D arbitrary observations in degrees and fits a mixture of VonMises distributions to the data. The number of distributions to fit is determined by the starting parameters: f\$mu\$ and f\$kappa\$ for each distribution. Alternatively, the program can scan the parameter space automatically, when only the number of distributions is given at the input.

USAGE:

```
ap_fit_VonMises_mixture chi_angles.dat mu kappa [mu2 kappa2 ...]
ap_fit_VonMises_mixture chi_angles.dat n_dist
```

EXAMPLES:

```
ap_fit_VonMises_mixture chi_angles.dat -1.05 30 -3.0 30 1.05 30
ap_fit_VonMises_mixture chi_angles.dat 3
```

Keywords:

- *statistics*
- *estimation*
- *expectation-maximization*

Categories:

- core::calc::statistics::VonMisesDistribution

Input files:

- THR_chi1.dat

Output files:

- stdout.out

Program source:

```
1 #include <math.h>
2
3 #include <iostream>
4 #include <random>
5
6 #include <core/algorithms/basic_algorithms.hh>
7 #include <core/algorithms/Combinations.hh>
8 #include <core/calc/statistics/NormalDistribution.hh>
9 #include <core/calc/statistics/expectation_maximization.hh>
10 #include <core/calc/numeric/numerical_integration.hh>
11 #include <core/calc/statistics/VonMisesDistribution.hh>
12 #include <core/calc/statistics/RobustDistributionDecorator.hh>
13 #include <core/data/io/DataTable.hh>
14
15 #include <utils/exit.hh>
16
17 std::string program_info = R"(
18
19 ap_fit_VonMises_mixture reads a text file with 1D arbitrary observations in degrees
20 and fits a mixture of VonMises distributions to the data. The number of
21 ↵distributions to fit
22 is determined by the starting parameters:  $\mu$  and  $\kappa$  for each
23 ↵distribution. Alternatively,
24 the program can scan the parameter space automatically, when only the number of
25 ↵distributions is given
26 at the input.
27 )"
```

(continues on next page)

(continued from previous page)

```

25 USAGE:
26     ap_fit_VonMises_mixture chi_angles.dat mu kappa [mu2 kappa2 ...]
27     ap_fit_VonMises_mixture chi_angles.dat n_dist
28
29 EXAMPLES:
30     ap_fit_VonMises_mixture chi_angles.dat -1.05 30 -3.0 30 1.05 30
31     ap_fit_VonMises_mixture chi_angles.dat 3
32
33 ) ";
34
35 /** @brief Reads a file with 1D data and estimates a mixture of VonMisesDistribution,
36  * based on these observations.
37 *
38 * CATEGORIES: core::calc::statistics::VonMisesDistribution
39 * KEYWORDS: statistics; estimation; expectation-maximization
40 * GROUP: Statistics;
41 * IMG: ap_fit_VonMises_mixture.png
42 * IMG_ALT: Distribution of Chil angles of THR side chains approximated with a
43 mixture of three von Mises distribution
44 */
45 int main(const int argc, const char *argv[]) {
46
47     if (argc < 3) utils::exit_OK_with_message(program_info); // --- complain about
48 missing program parameter
49     using namespace core::calc::statistics;
50
51     double deg_to_rad = M_PI / 180.0;
52
53     // ----- Read in data points (your observations) from a file
54     std::vector<std::vector<double>> data_points;
55     core::data::io::DataTable in(argv[1]);
56     double min = 180, max = -180; // This is to detects whether angle values are in
57 radians or in degrees
58     for (const auto &row : in) {
59         std::vector<double> d;
60         double v = row.get<double>(0);
61         if (v < min) min = v;
62         if (v > max) max = v;
63         d.push_back(v);
64         data_points.push_back(d);
65     }
66     if (((min < -M_PI) || (max > M_PI))) std::cerr << "Converting from degrees to
67 radians!\n";
68     else deg_to_rad = 1.0;
69     for (std::vector<double> &d : data_points) d[0] *= deg_to_rad;
70
71     std::vector<double> default_params{0.0, 100.0};
72
73     // ----- RobustDistributionDecorator object for each distribution
74     core::index1 n_distributions = atoi(argv[2]);
75     std::vector<RobustDistributionDecorator<VonMisesDistribution>> r_distributions_1D;
76     std::vector<RobustDistributionDecorator<VonMisesDistribution>> r_best_distributions;
77
78     std::vector<std::vector<std::vector<double>>> initial_parameters; // --- Initial
79 parameters for fitting
80     std::vector<core::index1> distribution_index; // --- Resulting assignment of every
81 data point to a distribution

```

(continues on next page)

(continued from previous page)

```

75     std::vector<core::index1> best_assignment; // --- Resulting assignment of every_
→data point to a distribution
76
77     // --- This is the case when user provided initial parameters for fitting Von Mises_
→distribution
78     if (argc >= 4) {
79         // ----- Read parameters from cmdline and create distribution objects
80         std::vector<std::vector<double>> params;
81         for (int i = 2; i < argc; i += 2) {
82             params.push_back(std::vector<double>{atof(argv[i]) * deg_to_rad, atof(argv[i +_
→1])});
83             r_distributions_1D.emplace_back(RobustDistributionDecorator
→<VonMisesDistribution>(params.back()));
84             r_best_distributions.emplace_back(RobustDistributionDecorator
→<VonMisesDistribution>(default_params));
85             initial_parameters.push_back(params);
86         }
87         initial_parameters.push_back(params);
88     } else {
89         // --- This is the case when user provided the number of distributions to be fit_
→automatically
90         n_distributions = atoi(argv[2]);
91         for (size_t i = 0; i < n_distributions; ++i) {
92             r_distributions_1D.emplace_back(RobustDistributionDecorator
→<VonMisesDistribution>(default_params));
93             r_best_distributions.emplace_back(RobustDistributionDecorator
→<VonMisesDistribution>(default_params));
94         }
95
96         std::vector<double> random_starts{-1.0, -0.8, -0.6, -0.4, -0.2, 0.0, 0.2, 0.4, 0.
→6, 0.8, 1.0}; // multiplicity of PI
97
98         core::algorithms::Combinations<double> generator(n_distributions, random_starts);
99         std::vector<double> combination(n_distributions);
100        std::vector<std::vector<double>> params;
101        while (generator.next(combination)) {
102            params.clear();
103            for (size_t i_distr = 0; i_distr < n_distributions; ++i_distr)
104                params.push_back(std::vector<double>{combination[i_distr] * M_PI, 100.0});
105            initial_parameters.push_back(params);
106        }
107    }
108
109    double best_likelihood = -std::numeric_limits<double>::max();
110    // ----- Run Expectation-Maximization algorithm
111    for (size_t i_start = 0; i_start < initial_parameters.size(); ++i_start) { // ---_
→iterate over starting points
112        for (size_t i_distr = 0; i_distr < r_distributions_1D.size(); ++i_distr) // ---_
→loop over distributions to set each starting point
113            r_distributions_1D[i_distr].copy_parameters_from(initial_parameters[i_start][i_
→distr]);
114        double score = expectation_maximization(data_points, r_distributions_1D,
→distribution_index, 0.1, 100);
115        if (score > best_likelihood) {
116            best_likelihood = score;
117            for (size_t i = 0; i < n_distributions; ++i)
118                r_best_distributions[i].copy_parameters_from(r_distributions_1D[i].
→parameters());

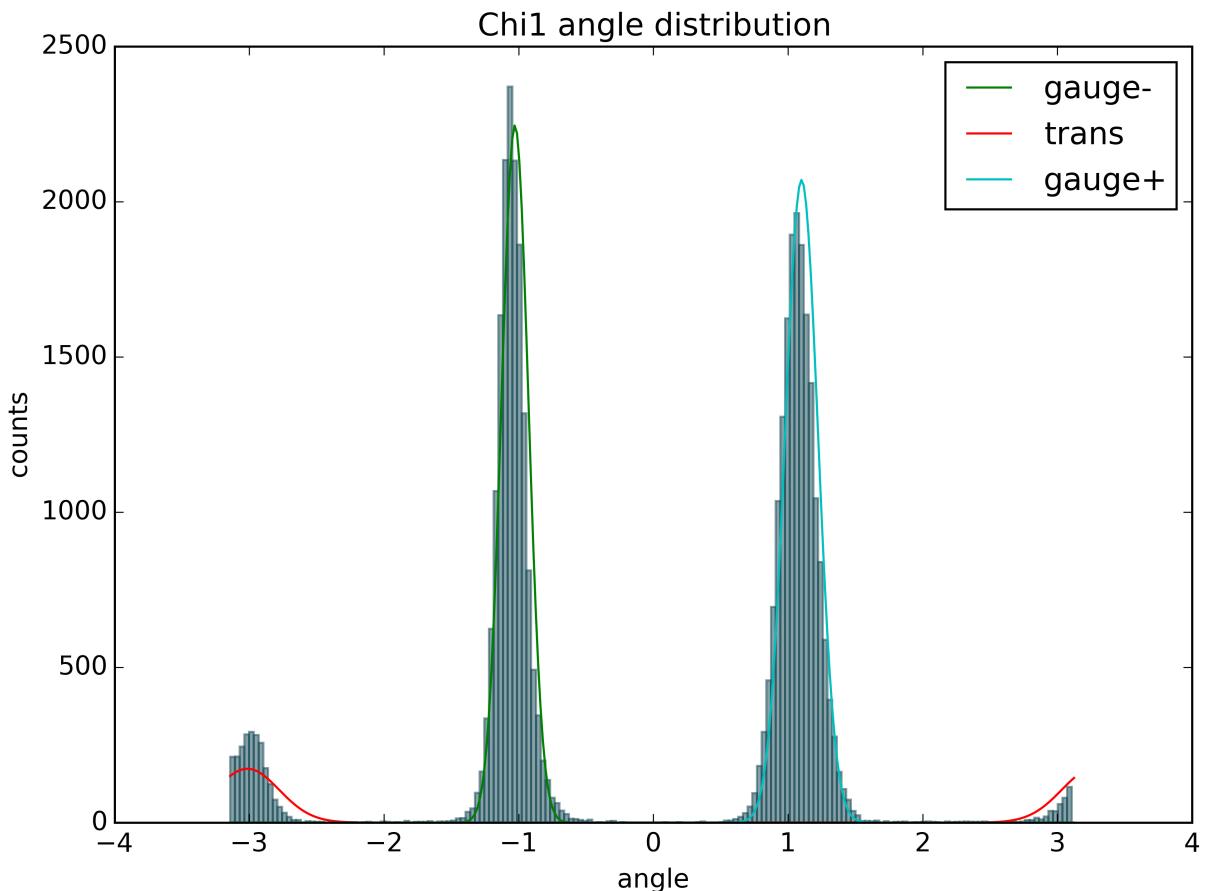
```

(continues on next page)

(continued from previous page)

```

119     best_assignment.swap(distribution_index);
120     std::cerr << "# Best likelihood so far: " << best_likelihood << "\n";
121   }
122 }
123 std::map<core::index1, core::index4> counts;
124 core::algorithms::count_distinct(best_assignment, counts);
125 const double total = std::accumulate(std::begin(counts), std::end(counts), 0,
126                                     [](const size_t previous, decltype(*counts.
127                                     begin()) p) {
128   return previous + p.second;
129 });
130 std::cout << "# Best likelihood " << best_likelihood << "\n# Estimated_
131 distributions:\n";
132 for (size_t i_distr = 0; i_distr < r_distributions_1D.size(); ++i_distr) {
133   std::cout << counts[i_distr] / total << " " << r_best_distributions[i_distr] <<
134   "\n";
135 }
136 }
```



ap_hbonds

ap_hbonds finds all hydrogen bonds in a given protein structure, including side chain interactions. For each bond the program lists residues involved and describes its geometry (bond length and respective angles). Backbone hydrogen bonds are reported separately from those involving side chains. Detection of hydrogen bond donors and acceptors in a given PDB deposit is based on the definition of respective monomers. The most popular monomers including amino acids and nucleotides are provided with the BioShell distribution. Others must be provided by a user, either in CIF or in PDB format. The input protein must include hydrogen atoms. Crystal structures should be protonated before using this app

USAGE:

```
ap_hbonds input.pdb [ligand_1.cif [ ligand_2.pdb ...] ]
```

EXAMPLE:

```
ap_hbonds 2gb1.pdb
```

OUTPUT (fragment):**Keywords:**

- *PDB input*
- *interactions*

Categories:

- core::calc::structural::interactions::HydrogenBondInteraction

Input files:

- 2kwi-1.pdb
- 2gb1.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <map>
3
4 #include <core/data/io/Pdb.hh>
5 #include <core/calc/structural/interactions/HydrogenBondInteraction.hh>
6 #include <core/calc/structural/interactions/BackboneHBondInteraction.hh>
7 #include <core/calc/structural/interactions/HydrogenBondCollector.hh>
8 #include <core/chemical/MonomerStructureFactory.hh>
9 #include <utils/string_utils.hh>
10 #include <utils/exit.hh>
```

(continues on next page)

(continued from previous page)

```

11 std::string program_info = R"(
12
13 ap_hbonds finds all hydrogen bonds in a given protein structure, including side chain_
14 ↪interactions.
15
16 For each bond the program lists residues involved and describes its geometry (bond_
17 ↪length and respective angles).
18 Backbone hydrogen bonds are reported separately from those involving side chains.
19
20 Detection of hydrogen bond donors and acceptors in a given PDB deposit is based on_
21 ↪the definition
22 of respective monomers. The most popular monomers including amino acids and_
23 ↪nucleotides are provided
24 with the BioShell distribution. Others must be provided by a user, either in CIF or_
25 ↪in PDB format.
26 The input protein must include hydrogen atoms. Crystal structures should be_
27 ↪protonated before using this app
28
29
30
31
32 USAGE:
33     ap_hbonds input.pdb [ligand_1.cif [ ligand_2.pdb ...] ]
34
35
36 EXAMPLE:
37     ap_hbonds 2gb1.pdb
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
) ";

/** @brief Finds all hydrogen bonds in a given protein structure
 *
 * CATEGORIES: core:::calc::structural:::interactions:::HydrogenBondInteraction
 * KEYWORDS: PDB input; interactions
 * GROUP: Structure calculations;
 * IMG: ap_hbonds_sq.png
 * IMG_ALT: Hydrogen bonds
 */
int main(const int argc, const char *argv[]) {
    if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
    ↪missing program parameter

    using namespace core:::data::io; // --- Pdb reader and PdbLineFilter lives there
    using namespace core:::chemical;
    using namespace core:::calc::structural:::interactions;

    // ----- Register additional monomers, provided by a user from a command line,_
    ↪either .pdb or .cif
    for (int i = 2; i < argc; ++i)
        MonomerStructureFactory::get_instance().register_monomer(argv[i]);

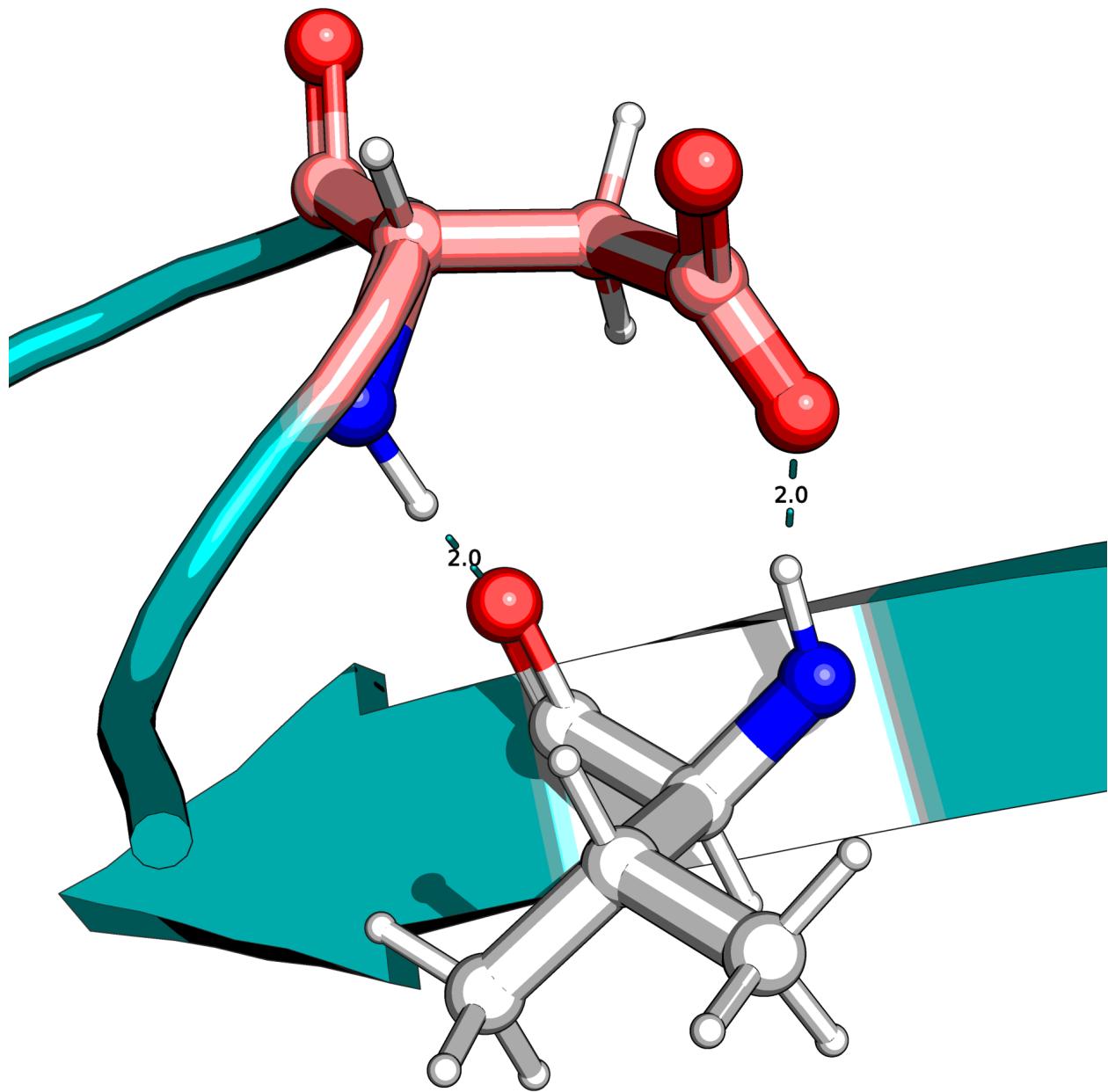
    // ----- Read a PDB file given as an argument to this program
    Pdb reader(argv[1]);
    HydrogenBondCollector collector;

```

(continues on next page)

(continued from previous page)

```
60     std::vector<ResiduePair_SP> sink;
61     // ----- Iterate over all models in the input file
62     for (size_t i_protein = 0; i_protein < reader.count_models(); ++i_protein) {
63         sink.clear();
64         core::data::structural::Structure_SP s = reader.create_structure(0);
65         collector.collect(*s, sink);
66         // ----- The first loop prints backbone hydrogen bonds
67         std::cout << BackboneHBondInteraction::output_header() << "\n";
68         for (const ResiduePair_SP ri:sink) {
69             BackboneHBondInteraction_SP bi = std::dynamic_pointer_cast
70             <BackboneHBondInteraction>(ri);
71             if (bi) std::cout << *bi << "\n";
72         }
73         // ----- The second loop prints hydrogen bonds involving side chain atoms
74         std::cout << HydrogenBondInteraction::output_header() << "\n";
75         for (const ResiduePair_SP ri:sink) {
76             BackboneHBondInteraction_SP bi = std::dynamic_pointer_cast
77             <BackboneHBondInteraction>(ri);
78             if (!bi) std::cout << *std::dynamic_pointer_cast<HydrogenBondInteraction>(ri) <
79             << "\n";
79 }
```



ap_interdigitated_strands

Reads a PDB file, creates a BetaStructuresGraph for it and finds all interdigitated strands. A strand is interdigitated when its hydrogen-bonded neighbors within a beta sheet come from different protein chains than that strand.

EXAMPLE:

```
ap_interdigitated_strands 2fdo.pdb
```

REFERENCE: Wang S. et al. "Crystal Structure of the Conserved Protein of Unknown Function AF2331 from Archaeoglobus fulgidus DSM 4304 Reveals a New Type of Alpha/Beta Fold" Protein Sci. (2009) 18 2410–2419.

Keywords:

- *PDB input*

Categories:

- core::data::structural::BetaStructuresGraph

Input files:

- 2fdo.pdb

Output files:

- stdout.out

Program source:

```
1 #include <core/data/io/Pdb.hh>
2 #include <core/algorithms/graph_algorithms.hh>
3 #include <core/data/structural/BetaStructuresGraph.hh>
4 #include <core/calc/structural/ProteinArchitecture.hh>
5
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Reads a PDB file, creates a BetaStructuresGraph for it and finds all interdigitated_
11 ↪strands.
12 A strand is interdigitated when its hydrogen-bonded neighbors within a beta sheet_
13 ↪come from
14 different protein chains than that strand.
15
16 EXAMPLE:
17     ap_interdigitated_strands 2fdo.pdb
18
19 REFERENCE:
20 Wang S. et al. "Crystal Structure of the Conserved Protein of Unknown Function AF2331_
21 ↪from Archaeoglobus fulgidus
22 DSM 4304 Reveals a New Type of Alpha/Beta Fold" Protein Sci. (2009) 18 2410-2419.
23 )";
24
25 void index_strands(core::data::structural::BetaStructuresGraph_SP g) {
26
27     using core::data::structural::Strand_SP;
28
29     // ----- Firstly, let's find the first strand on the path: the one with just_
30     ↪one partner
31     Strand_SP first_strand = nullptr;
32     for (auto it = g->begin_strand(); it != g->end_strand(); ++it) {
33         if (g->count_partners(*it) == 1) {
34             if (first_strand == nullptr) first_strand = *it;
35             else if (first_strand->length() > (*it)->length()) // there are two edge_
36             ↪strands, take the shorter one
```

(continues on next page)

(continued from previous page)

```

32         first_strand = *it;
33     }
34   }
35
36 // ----- If it's a barrel, take the shortest one
37 if (first_strand == nullptr) {
38     Strand_SP first_strand = *g->begin_strand();
39     for (auto it = g->begin_strand(); it != g->end_strand(); ++it) {
40         if (first_strand->length() > (*it)->length()) first_strand = *it;
41     }
42   }
43
44 std::set<Strand_SP> visited;
45 std::vector<Strand_SP> stack;
46 std::vector<Strand_SP> scratch;
47 stack.push_back(first_strand);
48 core::index2 idx = 0;
49 while (stack.size() > 0) {
50     // --- pop a strand from stack, mark as visited
51     Strand_SP s = stack.back();
52     s->strand_index_in_sheet = (++idx);
53     visited.insert(s);
54     stack.pop_back();
55
56     // --- get its neighbors, push to scratch if not visited yet
57     scratch.clear();
58     for (auto it = g->begin_strand(s); it != g->end_strand(s); ++it)
59         if (visited.find(*it) == visited.cend())
60             scratch.push_back(*it);
61
62     // --- sort neighbors
63     std::sort(scratch.begin(), scratch.end(),
64               [] (Strand_SP lhs, Strand_SP rhs) { return rhs->length() < lhs->length();
65   });
66     // --- push from the shortest
67     for (Strand_SP si : scratch) stack.push_back(si);
68   }
69 }
70
71 struct OrderStandsInSheet {
72
73     bool operator()(core::data::structural::Strand_SP lhs,
74     ~core::data::structural::Strand_SP rhs) { return lhs->strand_index_in_sheet < rhs->
75     strand_index_in_sheet; }
76 };
77
78 /** @brief Creates a BetaStructuresGraph and finds interdigitated sheets
79 */
80 * CATEGORIES: core::data::structural::BetaStructuresGraph
81 * KEYWORDS: PDB input
82 * GROUP: Structure calculations;
83 * IMG: 2fdo-7-sq.png
84 * IMG_ALT: Interdigitated beta-sheet of 2FDO deposit; the two chains A and B shown
85     with different colors
86 */
87 int main(const int argc, const char* argv[]) {

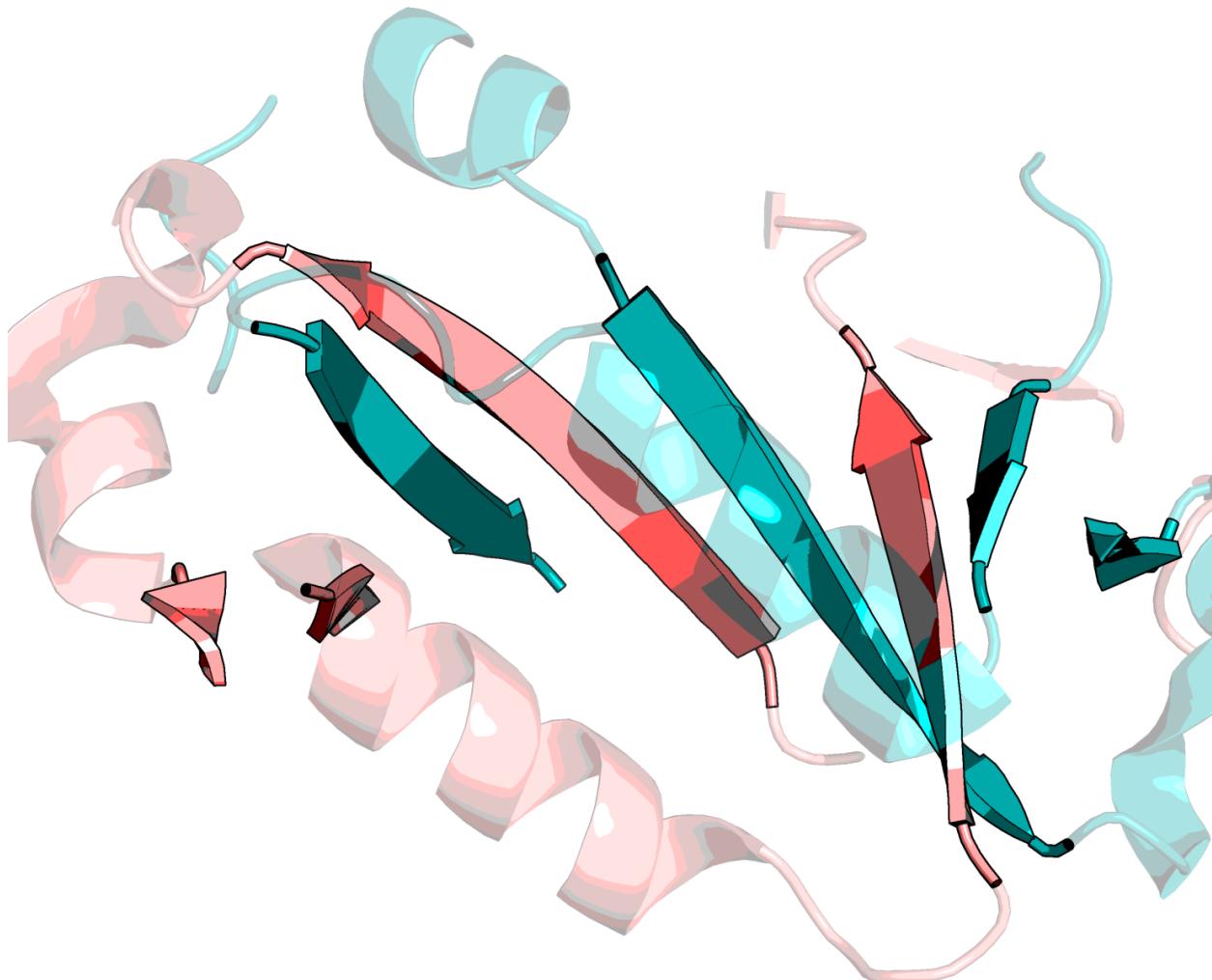
```

(continues on next page)

(continued from previous page)

```

85
86     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
87     ↵missing program parameter
88     using namespace core::data::structural;
89     using namespace core::data::io;
90
91 // core::data::io::Pdb reader(argv[1], (is_not_alternative), true);
92     core::data::io::Pdb reader(argv[1], core::data::io::all_true(is_not_alternative, is_
93     ↵not_water), core::data::io::keep_all, true);
94     core::data::structural::Structure_SP strctr = reader.create_structure(0);
95
96     std::string sec_str;
97     for (auto chain : *strctr) sec_str += chain->create_sequence()->str();
98     core::calc::structural::ProteinArchitecture a(*strctr, false);
99     BetaStructuresGraph_SP g = a.create_strand_graph();
100
101    g->print_adjacency_matrix(std::cerr);
102
103    for (auto s_it = g->begin(); s_it != g->end(); ++s_it) {
104        auto s = *s_it;
105        for (auto nbr = g->begin_strand(s); nbr != g->end_strand(s); ++nbr) {
106            core::data::structural::Strand_SP strnd = *nbr;
107        }
108
109        std::vector<StrandPairing_SP> edges;
110        for (auto it = g->cbegin_pairings(); it != g->cend_pairings(); ++it) edges.push_
111        ↵back((*it).second);
112        for (StrandPairing_SP sp:edges) {
113            Strand_SP first = sp->first_strand;
114            Strand_SP second = sp->second_strand;
115            if ((*first)[0]->owner()->id() == (*second)[0]->owner()->id()) {
116                g->remove_strand_pairing(first, second);
117            }
118
119        auto sheets = core::algorithms::connected_components<BetaStructuresGraph, Strand_SP,
120        ↵ StrandPairing_SP>(*g, 2);
121        int cnt = 0;
122        for(const auto & sheet: sheets) {
123            std::cout << utils::string_format("----- Sheet %d -----\\n",
124            ↵++cnt);
125            auto strnd = sheet->cbegin_strand();
126
127            index_strands(g); // --- index strands in a current sheet
128
129            std::vector<Strand_SP> strands;
130            for(auto it=sheet->cbegin_strand();it!=sheet->cend_strand();++it)
131                strands.push_back(*it);
132            std::sort(strands.begin(), strands.end(), OrderStandsInSheet{});
133            for (auto s:strands) std::cout << *s << ", has " << g->count_partners(s) << " "
134            ↵edges\\n";
135        }
136    }
137 }
```



ap_ligand_clustering

ap_ligand_clustering performs clustering analysis on small molecule docking poses. The default settings for this program are: clustering_cutoff: 5.0 Angstroms and min_cluster_size: 5 Every line of the output contains a single cluster: the first is number that cluster size, followed by PDB file names that belong to that cluster SEE: pdb_from_clustering.py example script is a tool to create PDB files based on output from ap_ligand_clustering and input PDB files

USAGE:

```
ap_ligand_clustering code list_of_files.txt [ min_cluster_size clustering_cutoff1 .. ↵ ]
```

SEE ALSO: ap_docking_crmsd - for a flexible docking crmsd calculations ap_stiff_docking_crmsd - for a rigid dock-

ing crmsd calculations ap_LigandsOnGridProtocol - simple clustering by projecting ligands on a 3D grid; crude but fast; can handle very large poolsof models

EXAMPLE:

```
ap_ligand_clustering CLO  list.txt 10 2.0 5.0
```

Keywords:

- *PDB input*
- *clustering*
- :ref:“

Categories:

- core::calc::clustering::HierarchicalClustering1B

Input files:

- 00040_HEM_Clotrimazole.pdb_9149.pdb
- 00040_HEM_Clotrimazole.pdb_4439.pdb
- 00040_HEM_Clotrimazole.pdb_5452.pdb
- 00040_HEM_Clotrimazole.pdb_4313.pdb
- 00040_HEM_Clotrimazole.pdb_6297.pdb
- 00040_HEM_Clotrimazole.pdb_8420.pdb
- 00040_HEM_Clotrimazole.pdb_2785.pdb
- 00040_HEM_Clotrimazole.pdb_5724.pdb
- 00040_HEM_Clotrimazole.pdb_6861.pdb
- 00040_HEM_Clotrimazole.pdb_1281.pdb
- 00040_HEM_Clotrimazole.pdb_3187.pdb
- file_list.txt
- 00040_HEM_Clotrimazole.pdb_6644.pdb
- 00040_HEM_Clotrimazole.pdb_4215.pdb
- 00040_HEM_Clotrimazole.pdb_3818.pdb
- 00040_HEM_Clotrimazole.pdb_2412.pdb
- 00040_HEM_Clotrimazole.pdb_528.pdb
- 00040_HEM_Clotrimazole.pdb_2169.pdb
- 00040_HEM_Clotrimazole.pdb_1614.pdb
- 00040_HEM_Clotrimazole.pdb_987.pdb
- 00040_HEM_Clotrimazole.pdb_4992.pdb
- 00040_HEM_Clotrimazole.pdb_4687.pdb

- 00040_HEM_Clotrimazole.pdb_9108.pdb
 - 00040_HEM_Clotrimazole.pdb_8537.pdb
 - 00040_HEM_Clotrimazole.pdb_7150.pdb
 - 00040_HEM_Clotrimazole.pdb_4295.pdb

Output files:

- `stdout.out`
 - `clusters-10.00.txt`
 - `clusters-6.00.txt`

Program source:

```

1 #include <iostream>
2 #include <map>
3
4 #include <core/data/io/Pdb.hh>
5 #include <utils/exit.hh>
6 #include <utils/io_utils.hh>
7 #include <core/data/structural/selectors/structure_selectors.hh>
8 #include <core/protocols/PairwiseLigandCrmsd.hh>
9 #include <core/calc/clustering/DistanceByValues1B.hh>
10 #include <core/calc/clustering/HierarchicalClustering1B.hh>
11
12 std::string program_info = R"(
13
14 ap_ligand_clustering performs clustering analysis on small molecule docking poses.
15 The default settings for this program are: clustering_cutoff: 5.0 Angstroms and
16 min_cluster_size: 5
17
18 Every line of the output contains a single cluster: the first is number that cluster_
19 ↵size,
20 followed by PDB file names that belong to that cluster
21
22 SEE:
23     pdb_from_clustering.py example script is a tool to create PDB files based on_
24 ↵output from
25         ap_ligand_clustering and input PDB files
26
27 USAGE:
28     ap_ligand_clustering code list_of_files.txt [ min_cluster_size clustering_cutoff1_
29 ↵... ]
30
31 SEE ALSO:
32     ap_docking_crmsd - for a flexible docking crmsd calculations
33     ap_stiff_docking_crmsd - for a rigid docking crmsd calculations
34     ap_LigandsOnGridProtocol - simple clustering by projecting ligands on a 3D grid;_
35 ↵crude but fast; can handle very large poolsof models
36
37 EXAMPLE:
38     ap_ligand_clustering CLO  list.txt 10 2.0 5.0

```

(continues on next page)

(continued from previous page)

```

36 ) ";
37
38 /**
39 * @brief Performs clustering analysis on small molecule docking poses
40 *
41 * CATEGORIES: core::calc::clustering::HierarchicalClustering1B
42 * KEYWORDS: PDB input; clustering;
43 * GROUP: Structure calculations; Docking;
44 * IMG:
45 * IMG_ALT:
46 */
47
48 int main(const int argc, const char* argv[]) {
49
50     if (argc < 3) utils::exit_OK_with_message(program_info); // --- complain about_
51     ↪missing program parameter
52
53     utils::Logger l("ap_ligand_clustering");
54
55     using namespace core::data::structural::selectors;
56     AtomSelector_SP select_ligand =
57         std::static_pointer_cast<AtomSelector>(std::make_shared<SelectResidueByName>
58     ↪(argv[1]));
59     AtomSelector_SP select_ca =
60         std::static_pointer_cast<AtomSelector>(std::make_shared<IsCA>());
61
62     core::protocols::PairwiseLigandCrmsd crmsd_calculator(select_ligand, select_ca);
63
64     std::vector<std::string> fnames = utils::read_listfile(argv[2]);
65     for(const std::string & f:fnames) {
66         core::data::io::Pdb reader(f, "is_not_hydrogen", false, false);
67         crmsd_calculator.add_input_structure(reader.create_structure(0), f);
68     }
69
70     core::index2 min_cluster_size = (argc < 4) ? 5 : atof(argv[3]);
71     std::vector<double> clustering_cutoffs;
72     double max_clustering_cutoff = 0.0;
73     if (argc < 4) {
74         max_clustering_cutoff = 15.0;
75         clustering_cutoffs.push_back(15.0);
76     } else {
77         for (int i = 4; i < argc; ++i) {
78             clustering_cutoffs.push_back(atof(argv[i]));
79             max_clustering_cutoff = std::max(max_clustering_cutoff, clustering_cutoffs.
80             ↪back());
81         }
82     }
83     double evaluate_cutoff = max_clustering_cutoff * 1.5;
84     double conversion_factor = 255 / evaluate_cutoff;
85
86     crmsd_calculator.crmsd_cutoff(evaluate_cutoff);
87     crmsd_calculator.set_out_matrix();
88     crmsd_calculator.calculate();
89     auto out = crmsd_calculator.out_matrix();
90
91     core::calc::clustering::DistanceByValues1B distances(crmsd_calculator.tags());
92     for (core::index4 i = 1; i < fnames.size(); ++i) {
93         for (core::index4 j = 0; j < i; ++j) {
94             if (out->has_element(i, j)) {

```

(continues on next page)

(continued from previous page)

```

90     double v = out->at(i, j) * conversion_factor;
91     distances.set(i, j, core::index1(v));
92     distances.set(j, i, core::index1(v));
93     // --- uncomment the line below to see the actual distance values together
94     // with their converted counterparts
95     // std::cerr << i << " " << j << " " << out->at(i, j) << " " << int(v) << "\n"
96     //";
97   }
98 }
99
100 core::calc::clustering::HierarchicalClustering1B hac(distances.labels(), "");
101 hac.run_clustering(distances, "COMPLETE_LINK");
102 for (double clustering_cutoff:clustering_cutoffs) {
103   std::ofstream out(utils::string_format("clusters-%.2f.txt", clustering_cutoff));
104   auto clusters = hac.get_clusters(clustering_cutoff * conversion_factor, min_
105   cluster_size);
106   l << utils::LogLevel::INFO << clusters.size() << " clusters created for cutoff " <
107   << clustering_cutoff << "\n";
108
109   for (const auto &c : clusters) {
110     std::vector<std::string> el = c->cluster_items(c);
111     out << el.size() << " ";
112     for (const std::string &s:el) out << s << " ";
113     out << "\n";
114   }
115   out.close();
116 }
117 }
```

ap_ligand_contacts

ap_ligand_contacts finds contacts between a ligand molecule and a protein. It reads a multi-model PDB file and for each of the models detects contacts between a particular ligand and the rest of the complex. The ligand must be identified by its three-letter code. The output provides the interacting residues (name and residueId) along - separately for each model

USAGE:

```
ap_ligand_contacts input.pdb ligand-code cutoff-distance
```

EXAMPLE:

```
ap_ligand_contacts 5edw.pdb TTP 7.0
```

where 5edw.pdb id an input file, TTP the ligand code and 7.0 - contact distance in Angstroms

OUTPUT (fragment): — ligand — | —— partner —— | distance c res id atname | c res id type atname | in Angstrom A TTP 404 C5' A ASP 105 protein OD1 3.371 A TTP 404 C5' A ASP 105 protein OD2 3.149 A TTP 404 O2G A LYS 159 protein CE 2.958 A TTP 404 O2G A LYS 159 protein NZ 2.936 A TTP 404 O3G A LYS 159 protein NZ 3.455 A TTP 404 O2A A CA 401 unknown CA 2.316 A TTP 404 O2B A CA 401 unknown CA 2.375 A TTP 404 O2G A CA 401 unknown CA 2.325 A TTP 404 O2A A CA 402 unknown CA 2.356 A TTP 404 O1A A HOH 510 unknown O 3.150 A TTP 404 O3A A HOH 510 unknown O 2.782 A TTP 404 O1G A HOH 510 unknown O 3.373 A TTP 404 O2 T DG 6 nucleic N1 3.048 A TTP 404 O2 T DG 6 nucleic C2 3.467 A TTP 404 O2 T DG 6 nucleic N2 2.931 A TTP 404 N3 T DG 6 nucleic O6 3.129

Keywords:

- *PDB input*
- *contact map*
- *ligand*

Categories:

- core::data::io::Pdb

Input files:

- 2kwi.pdb
- 5edw.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <map>
3
4 #include <core/data/io/Pdb.hh>
5 #include <utils/string_utils.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(

9 ap_ligand_contacts finds contacts between a ligand molecule and a protein.

10 It reads a multi-model PDB file and for each of the models detects contacts between a
11 ↪particular ligand
12 and the rest of the complex. The ligand must be identified by its three-letter code.
13 The output provides the interacting residues (name and residueId) along - separately
14 ↪for each model

15 USAGE:
16     ap_ligand_contacts input.pdb ligand-code cutoff-distance
17
18 EXAMPLE:
19     ap_ligand_contacts 5edw.pdb TTP 7.0
20
21 where 5edw.pdb id an input file, TTP the ligand code and 7.0 - contact distance in
22 ↪Angstroms
23
24 OUTPUT (fragment):
25     ---- ligand ---- | ----- partner ----- | distance
26     c res id atname | c res id type atname | in Angstrom
27     A TTP 404 C5'    A ASP 105 protein OD1    3.371
```

(continues on next page)

(continued from previous page)

```

28 A TTP 404 C5'      A ASP 105 protein OD2    3.149
29 A TTP 404 O2G      A LYS 159 protein CE     2.958
30 A TTP 404 O2G      A LYS 159 protein NZ     2.936
31 A TTP 404 O3G      A LYS 159 protein NZ     3.455
32 A TTP 404 O2A      A CA  401 unknown CA    2.316
33 A TTP 404 O2B      A CA  401 unknown CA    2.375
34 A TTP 404 O2G      A CA  401 unknown CA    2.325
35 A TTP 404 O2A      A CA  402 unknown CA    2.356
36 A TTP 404 O1A      A HOH 510 unknown O     3.150
37 A TTP 404 O3A      A HOH 510 unknown O     2.782
38 A TTP 404 O1G      A HOH 510 unknown O     3.373
39 A TTP 404 O2       T   DG   6 nucleic N1    3.048
40 A TTP 404 O2       T   DG   6 nucleic C2    3.467
41 A TTP 404 O2       T   DG   6 nucleic N2    2.931
42 A TTP 404 N3       T   DG   6 nucleic O6    3.129

43 )
44 */

45 /**
46  * @brief Finds contacts between a ligand molecule and a protein.
47  *
48  * @CATEGORIES: core::data::io::Pdb
49  * @KEYWORDS: PDB input; contact map; ligand
50  * @GROUP: Structure calculations;
51  * @IMG: ap_ligand_contacts.png
52  * @IMG_ALT: Contacts found between 5EDW protein and its ligand TTP
53 */
54 int main(const int argc, const char* argv[]) {
55
56     if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about
57     //missing program parameter
58
59     core::data::io::Pdb reader(argv[1]); // --- file name (PDB format, may be gzip-ped)
60
61     const std::string code(argv[2]); // --- The ligand code is the second parameter
62     //of the program
63     double cutoff = utils::from_string<double>(argv[3]); // The third parameter is the
64     //contact distance (in Angstroms)
65
66     std::cout << " ---- ligand ---- | ----- partner ----- | distance\n";
67     std::cout << "c res id atname | c res id type atname | in Angstrom\n";
68
69     for (size_t i = 0; i < reader.count_models(); ++i) { // --- Iterate over all models
70     //in the input file
71         core::data::structural::Structure_SP strctr = reader.create_structure(i);
72
73         // --- Here we use a standard <code>find_if</code> algorithm to find the ligand
74         //residue by its 3-letter code
75         auto ligand = std::find_if(strctr->first_residue(), strctr->last_residue(), [&
76         //code](core::data::structural::Residue_SP res) {return (res->residue_type() ==
77         //code3==code);});
78         if(ligand== strctr->last_residue()) { // --- If no ligand - print a message and
79         //take next structure
80             std::cerr << "Model " << i << " of " << argv[1] << " has no " << argv[2] << "
81             //residue\n";
82             continue;
83         }

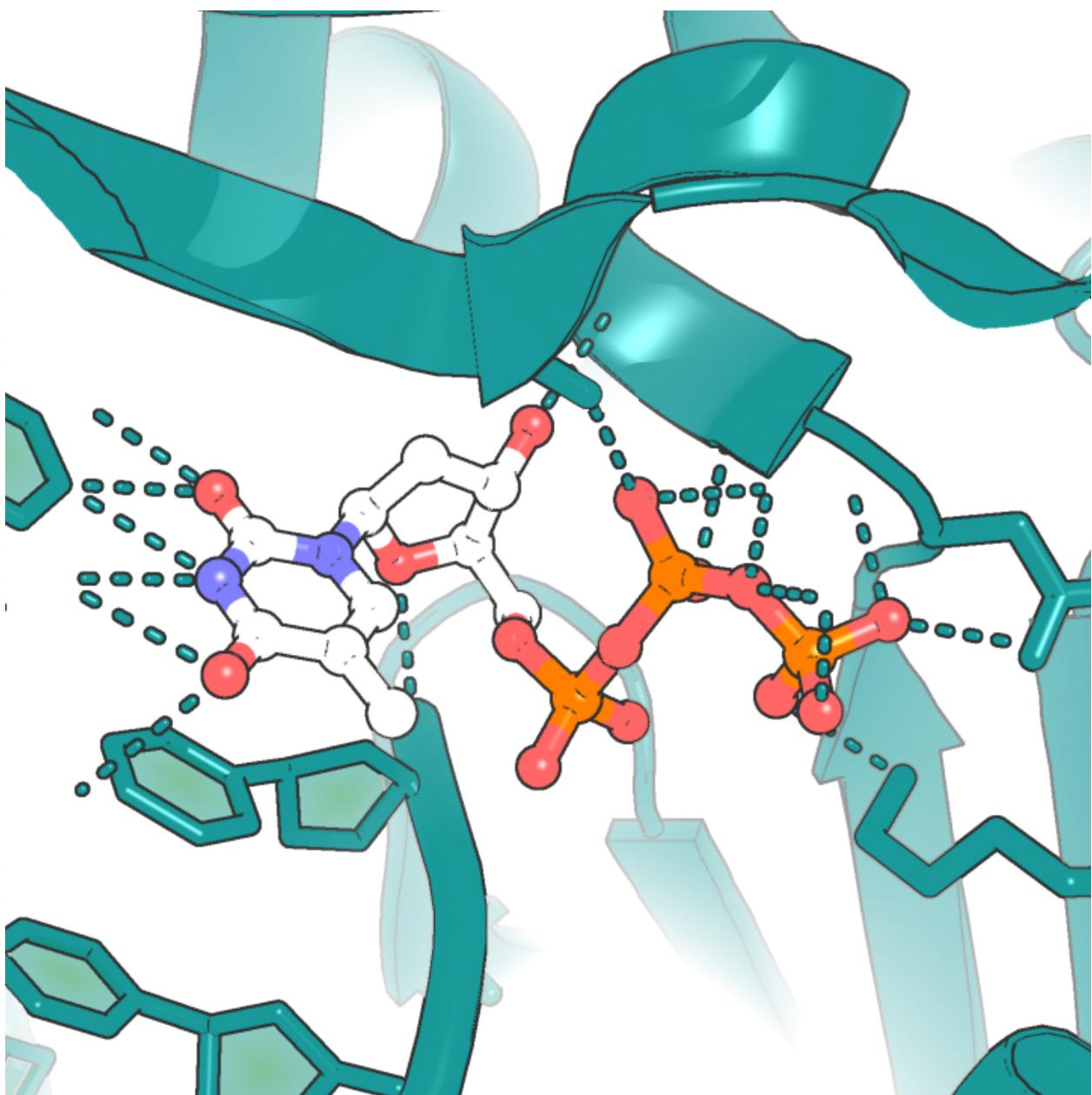
```

(continues on next page)

(continued from previous page)

```

76     if (reader.count_models() > 1) std::cout << "# Model " << i + 1 << "\n";
77     for (auto it_resid = strctr->first_residue(); it_resid != strctr->last_residue(); ++it_resid) {
78         if (*it_resid == *ligand) continue;
79         double d = (*it_resid)->min_distance(*ligand);
80         if (d < cutoff) { // --- if this is close enough,
81             for(auto const & ligand_atom : **ligand) {
82                 for(auto const & other_atom : **it_resid) {
83                     if(ligand_atom->distance_to(*other_atom) <= cutoff) {
84                         std::cout << utils::string_format("%4s %3s %4d %4s %4s %3s %4d %6.3f\n",
85                                         (**ligand).owner()->id().c_str(),
86                                         (**ligand).residue_type().code3.c_
87                                         str(), (**ligand).id(),
88                                         ligand_atom->atom_name().c_str(),
89                                         (**it_resid).owner()->id().c_str(),
89                                         (**it_resid).residue_type().code3.c_
89                                         str(), (**it_resid).id(),
90                                         core::chemical::monomer_type_
91                                         name((**it_resid).residue_type()).c_str(),
92                                         other_atom->atom_name().c_str(),
92                                         ligand_atom->distance_to(*other_
93                                         atom));
94                     }
95                 }
96             }
97         }
98     }
99 }
```



ap_orient_pdb

ap_orient_pdb reads a PDB file and orients the atoms along the axes so the longest protein dimension is along X and the second longest along Y. This example also creates a second transformation, that repeatedly rotate a structure fragment around Z axis by 45 degrees. The first (mandatory) argument is a PDB file name. User can also specify a structural fragment by providing a respective chain-ID and a residue range.

USAGE:

```
ap_orient_pdb input.pdb [chain-id first-resid last-resid]
```

EXAMPLE:

```
ap_orient_pdb input.pdb B 419 446
```

where 2kwi.pdb is the name of an input file and 419 446 are the first and last of the reoriented residues of chain B, respectively

Keywords:

- *PDB input*
- structural fragment
- *structure selectors*
- PCA
- transformations

Categories:

- core/calc/numeric/PCA.hh

Input files:

- 2kwi.pdb

Output files:

- stdout.out
- helix_rotated.pdb

Program source:

```
1 #include <iostream>
2 #include <random>
3
4 #include <core/index.hh>
5 #include <core/data/io/Pdb.hh>
6 #include <core/data/structural/selectors/structure_selectors.hh>
7 #include <core/calc/structural/transformations/Rototranslation.hh>
8 #include <core/calc/numeric/Pca3.hh>
9 #include <utils/exit.hh>
10 #include <core/calc/structural/angles.hh>
11
12 std::string program_info = R"
13
14 ap_orient_pdb reads a PDB file and orients the atoms along the longest_
15 ↴protein dimension is along X
16 and the second longest along Y.
17
18 This example also creates a second transformation, that repeatedly rotate a structure_
19 ↴fragment around Z axis by 45 degrees
20 The first (mandatory) argument is a PDB file name. User can also specify a structural_
21 ↴fragment by providing a respective
```

(continues on next page)

(continued from previous page)

```

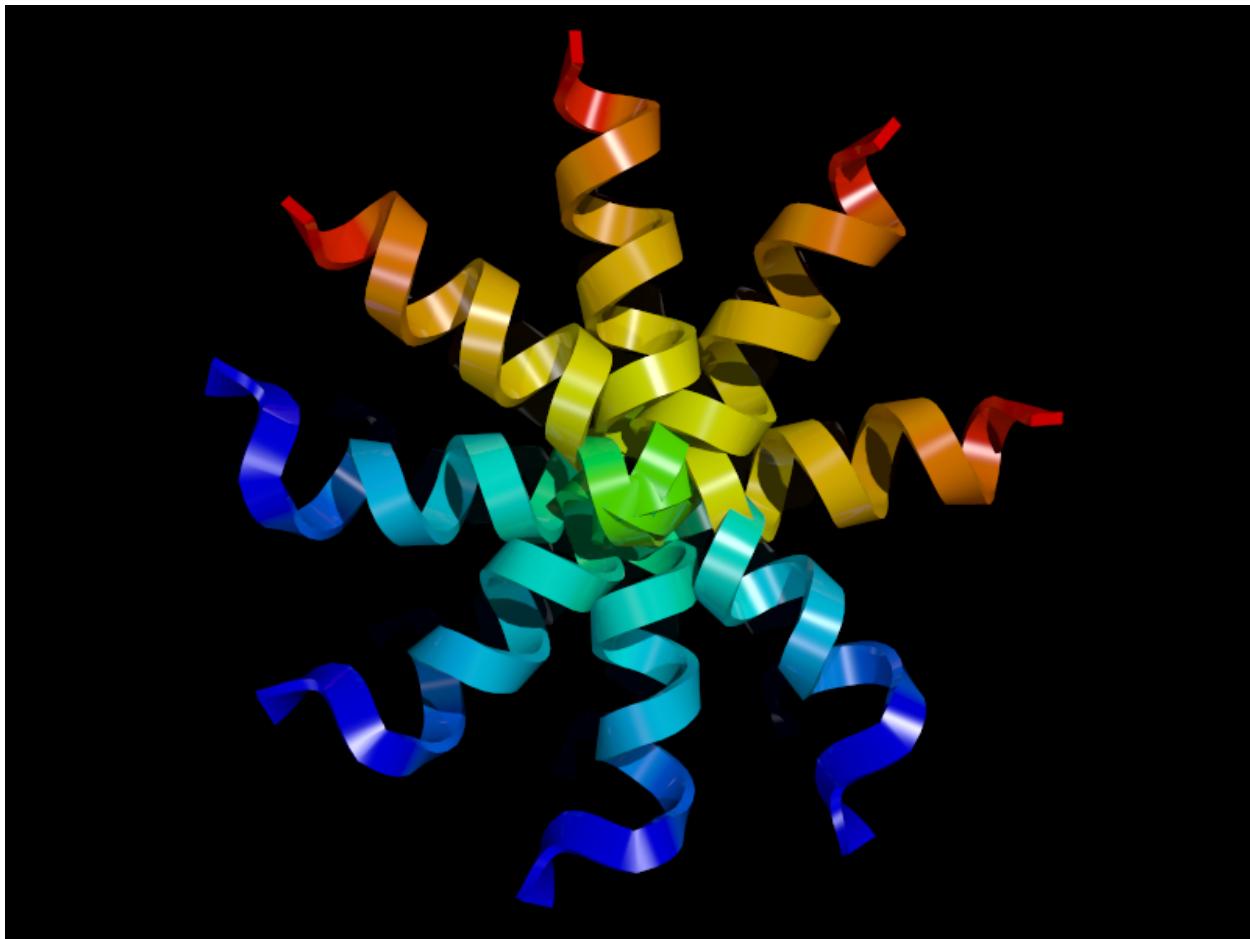
19 chain-ID and a residue range.
20
21 USAGE:
22     ap_orient_pdb input.pdb [chain-id first-resid last-resid]
23 EXAMPLE:
24     ap_orient_pdb input.pdb B 419 446
25
26 where 2kwi.pdb is the name of an input file and 419 446 are the first and last
27 of the reoriented residues of chain B, respectively
28
29 )";
30
31 /** @brief Shows how to rotate a piece of a protein structure
32 *
33 * CATEGORIES: core/calc/numeric/PCA.hh
34 * KEYWORDS: PDB input; structural fragment; structure selectors; PCA; transformations
35 * GROUP: Structure calculations;
36 * IMG: helices.png
37 * IMG_ALT: Alpha helix rotated a few times by a fixed angle
38 */
39 int main(const int argc, const char *argv[]) {
40
41     using namespace core::data::basic; // --- for Vec3 and Array2D
42
43     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
44     ↪missing program parameter
45
46     core::data::io::Pdb reader(argv[1]); // --- read the input PDB file
47     core::data::structural::Structure_SP structure_sp = reader.create_structure(0); // -
48     ↪--- create a structure corresponding to the first model
49
50     std::vector<core::data::structural::PdbAtom_SP> points3d;
51
52     // --- if a chain-ID and residue range were also given, create a selector to_
53     ↪extract the relevant part of the input
54     if (argc > 4) {
55         std::string selection_string = utils::string_format("%c:%d-%d", argv[2][0],_
56         ↪atoi(argv[3]), atoi(argv[4]));
57         core::data::structural::selectors::SelectChainResidues selector(selection_string);
58         // --- if selector selects (returns true), copy the atoms
59         for (auto atom_it = structure_sp->first_atom(); atom_it != structure_sp->last_
60             ↪atom(); ++atom_it)
61             if (selector((*atom_it)->owner())) points3d.push_back(*atom_it);
62         } else { // --- If there is no selection, copy all the atoms from the given_
63             ↪structure
64             for (auto atom_it = structure_sp->first_atom(); atom_it != structure_sp->last_
65                 ↪atom(); ++atom_it)
66                 points3d.push_back(*atom_it);
67             }
68         core::calc::numeric::Pca3 pca3(points3d);
69         auto rt = pca3.create_transformation();
70         std::cout << "MODEL      1\n";
71         for (auto atom : points3d) {
72             rt.apply(*atom);
73             std::cout << (atom)->to_pdb_line() << "\n";
74         }
75         std::cout << "ENDMDL\n";

```

(continues on next page)

(continued from previous page)

```
69 auto rt2 = core::calc::structural::transformations::Rototranslation::around_axis(
70     Vec3(0,0,1),core::calc::structural::to_radians(45.0),Vec3(0,0,0));
71 for(int i=2;i<5;i++) {
72     std::cout << "MODEL" << i << "\n";
73     for (auto atom : points3d) {
74         rt2.apply(*atom);
75         std::cout << atom->to_pdb_line() << "\n";
76     }
77     std::cout << "ENDMDL\n";
78 }
79
80 }
```



ap_shuffled_sequence_alignment

Reads a FASTA file with two sequences and calculate global sequence alignment scores with one of the two sequences randomly shuffled N_shuffles times (1000 by default). Each time the reshuffled sequence is aligned to the other one. The statistics of scores from randomised alignments is then used to estimate p-value of the global alignment. The default substitution-matrix is BLOSUM62. The program prints all the randomized alignment scores and estimated p-value of the alignment.

USAGE:

```
ap_shuffled_sequence_alignment input.fasta [ [substitution_matrix] N_shuffles]
```

EXAMPLE:

```
ap_shuffled_sequence_alignment input2.fasta BLOSUM80 10000
```

Keywords:

- *FASTA input*
- *Needleman-Wunsch*
- *sequence alignment*
- *statistics*

Categories:

- core::alignment::NWAligner

Input files:

- input2.fasta

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <chrono>
3
4 #include <core/data/io/fasta_io.hh>
5
6 #include <core/alignment/NWAligner.hh>
7 #include <core/alignment/scoring/SimilarityMatrix.hh>
8 #include <core/alignment/scoring/SimilarityMatrixScore.hh>
9 #include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>
10 #include <core/calc/statistics/OnlineStatistics.hh>
11 #include <core/calc/statistics/NormalDistribution.hh>
12 #include <core/calc/statistics/Random.hh>
13 #include <core/protocols/PairwiseSequenceIdentityProtocol.hh>
14 #include <utils/exit.hh>
15
16 std::string program_info = R"(
17
18 Reads a FASTA file with two sequences and calculate global sequence alignment scores
19 with one of the two sequences randomly shuffled N_shuffles times (1000 by default).
20 Each time the reshuffled sequence is aligned to the other one. The statistics of
21 ↵scores
from randomised alignments is then used to estimate p-value of the global alignment.
)
```

(continues on next page)

(continued from previous page)

```

22 The default substitution-matrix is BLOSUM62
23
24 The program prints all the randomized alignment scores and estimated p-value of the ↵
25 alignment
26
27 USAGE:
28     ap_shuffled_sequence_alignment input.fasta  [[substitution_matrix] N_shuffles]
29
30 EXAMPLE:
31     ap_shuffled_sequence_alignment input2.fasta  BLOSUM80 10000
32 ) ";
33
34 /** @brief Calculate global sequence alignment scores with one sequence randomly ↵
35 shuffled and estimates alignment p-value
36 *
37 * CATEGORIES: core:::alignment::NWAligner
38 * KEYWORDS: FASTA input; Needleman-Wunsch; sequence alignment; statistics
39 * GROUP: Alignments
40 * IMG: ap_shuffled_sequence_alignment.png
41 * IMG_ALT: Statistics of random sequence alignment between IBC6 and SFL95851.1
42 */
43
44 int main(const int argc, const char *argv[]) {
45
46     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about ↵
47     missing program parameter
48
49     using namespace core:::data:::io;
50     using namespace core:::alignment:::scoring;
51
52     core:::index2 n_shuffles = (argc > 3) ? atoi(argv[3]) : 1000; // --- The number of ↵
53     random shuffles
54
55     // --- Read the query sequence
56     std::vector<std::shared_ptr<Sequence>> input_sequences;
57     read_fasta_file(argv[1], input_sequences);
58
59     // --- find longest sequence to initialize aligner object large enough
60     unsigned max_len = std::max(input_sequences[0]->length(), input_sequences[1]->
61     length());
62
63     // --- create aligner object
64     core:::alignment::NWAligner<short, SimilarityMatrixScore<short>> aligner(max_len);
65
66     // --- read similarity matrix from a file (i.e. BLOSUM62)
67     std::string substitution_matrix_name = (argc > 2) ? argv[2] : "BLOSUM62";
68     NcbiSimilarityMatrix_SP sim_m = NcbiSimilarityMatrixFactory::get().get_matrix(
69     "BLOSUM62");
70
71     // --- go through all db sequences and align them with the given query
72     auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
73
74     std::string j_seq_copy = input_sequences[1]->sequence;
75     SimilarityMatrixScore<short> score(input_sequences[0]->sequence, j_seq_copy, *sim_
76     m);
77     // --- find score of the alignment; just the score - this is faster than aligning ↵
78     and keeping backtracking info

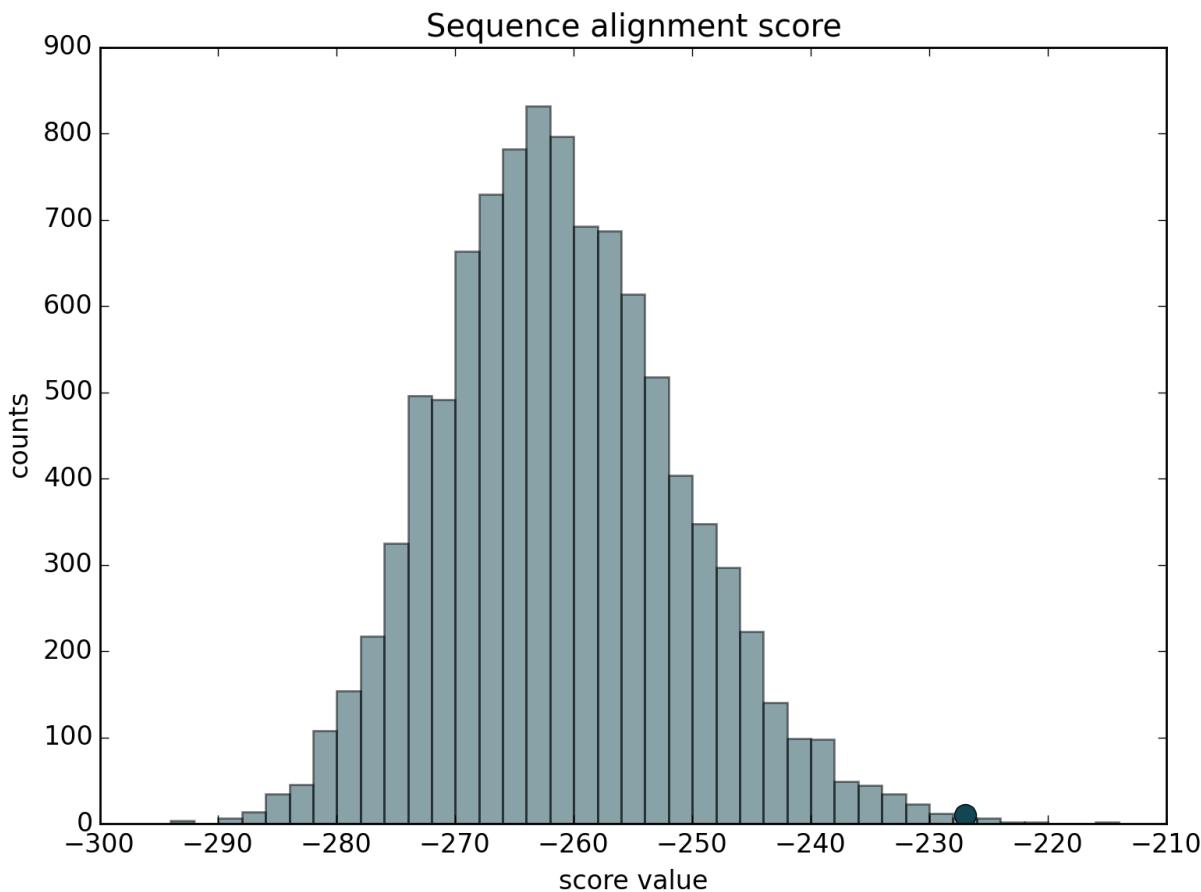
```

(continues on next page)

(continued from previous page)

```

71 short result = aligner.align_for_score(-10, -1, score);
72
73 core::calc::statistics::Random & r = core::calc::statistics::Random::get();
74 r.seed(12345); // --- seed the generator for repeatable results
75 core::calc::statistics::OnlineStatistics stats; // --- online (on-the fly)
76 // statistics calculator
77 for (size_t i = 0; i < n_shuffles; ++i) {
78     shuffle(j_seq_copy.begin(), j_seq_copy.end(), r);
79     SimilarityMatrixScore<short> score(input_sequences[0]->sequence, j_seq_copy, *sim_
80     m);
81     short res = aligner.align_for_score(-10, -1, score);
82     stats(res);
83     std::cout << res << "\n";
84 }
85 auto end = std::chrono::high_resolution_clock::now();
86 std::chrono::duration<double> time_span = std::chrono::duration_cast
87 //<std::chrono::duration<double>>(end - start);
88 std::cerr << "# " << n_shuffles << " alignment shuffled scores computed within "
89 << time_span.count() << " [s]\n";
90 std::cout << "# alignment score: " << result << "\n";
91 std::cout << "# normal p-value, avg, sdev: "
92 << 1 - core::calc::statistics::NormalDistribution::cdf(result, stats.
93 //avg(), sqrt(stats.var())) << " "
94 << stats.avg()
95 << " " << sqrt(stats.var()) << "\n";
96 core::protocols::PairwiseSequenceIdentityProtocol protocol;
97 protocol.substitution_matrix("BLOSUM62").gap_open(-10).gap_extend(-1);
98 protocol.add_input_sequence(input_sequences[0]);
99 protocol.add_input_sequence(input_sequences[1]);
100 protocol.run();
101 std::cout << "# same value calculated by a library function: " << protocol.count_
102 //identical(0, 1) << "\n";
103 }
```



ap_stacking_interactions

Finds stacking interactions in a given PDB file. The program reports all stacking interactions detected in a given PDB file. A plausible stacking interaction is detected when two aromatic rings are found to be close in space. Detection of aromatic rings in a given PDB deposit is based on the definition of respective monomers. The most popular monomers including amino acids and nucleotides are provided with the BioShell distribution. Others must be provided by a user, either in CIF or in PDB format.

USAGE:

```
ap_stacking_interactions input.pdb [ligand1.cif [ligand2.pdb ...] ]
```

EXAMPLE:

```
ap_stacking_interactions 5edw.pdb
```

Keywords:

- *PDB input*
- *PDB line filter*
- stacking interactions

Categories:

- core::calc::structural::interactions::StackingInteraction

Input files:

- 2gb1.pdb
- 5edw.pdb
- 3dcg.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <core/data/io/Pdb.hh>
3 #include <core/calc/structural/interactions/StackingInteraction.hh>
4 #include <core/calc/structural/interactions/StackingInteractionCollector.hh>
5 #include <utils/exit.hh>
6 #include <utils/LogManager.hh>
7
8 std::string program_info = R"(

9 Finds stacking interactions in a given PDB file.

10 The program reports all stacking interactions detected in a given PDB file.

11 A plausible stacking interaction is detected when two aromatic rings are found to be close in space.

12 Detection of aromatic rings in a given PDB deposit is based on the definition of respective monomers.
13 The most popular monomers including amino acids and nucleotides are provided with the BioShell distribution.
14 Others must be provided by a user, either in CIF or in PDB format.

15 USAGE:
16     ap_stacking_interactions input.pdb [ligand1.cif [ligand2.pdb ...] ]
17
18 EXAMPLE:
19     ap_stacking_interactions 5edw.pdb
20
21 )";

22
23 using namespace core::data::io; // --- Pdb reader and PdbLineFilter lives there
24 using namespace core::chemical;
25 using namespace core::calc::structural::interactions;
26
27 /**
28 * @brief Finds stacking interactions in a given PDB file.
29 */
30
31 /**
32 * @CATEGORIES: core::calc::structural::interactions::StackingInteraction
33 */
34

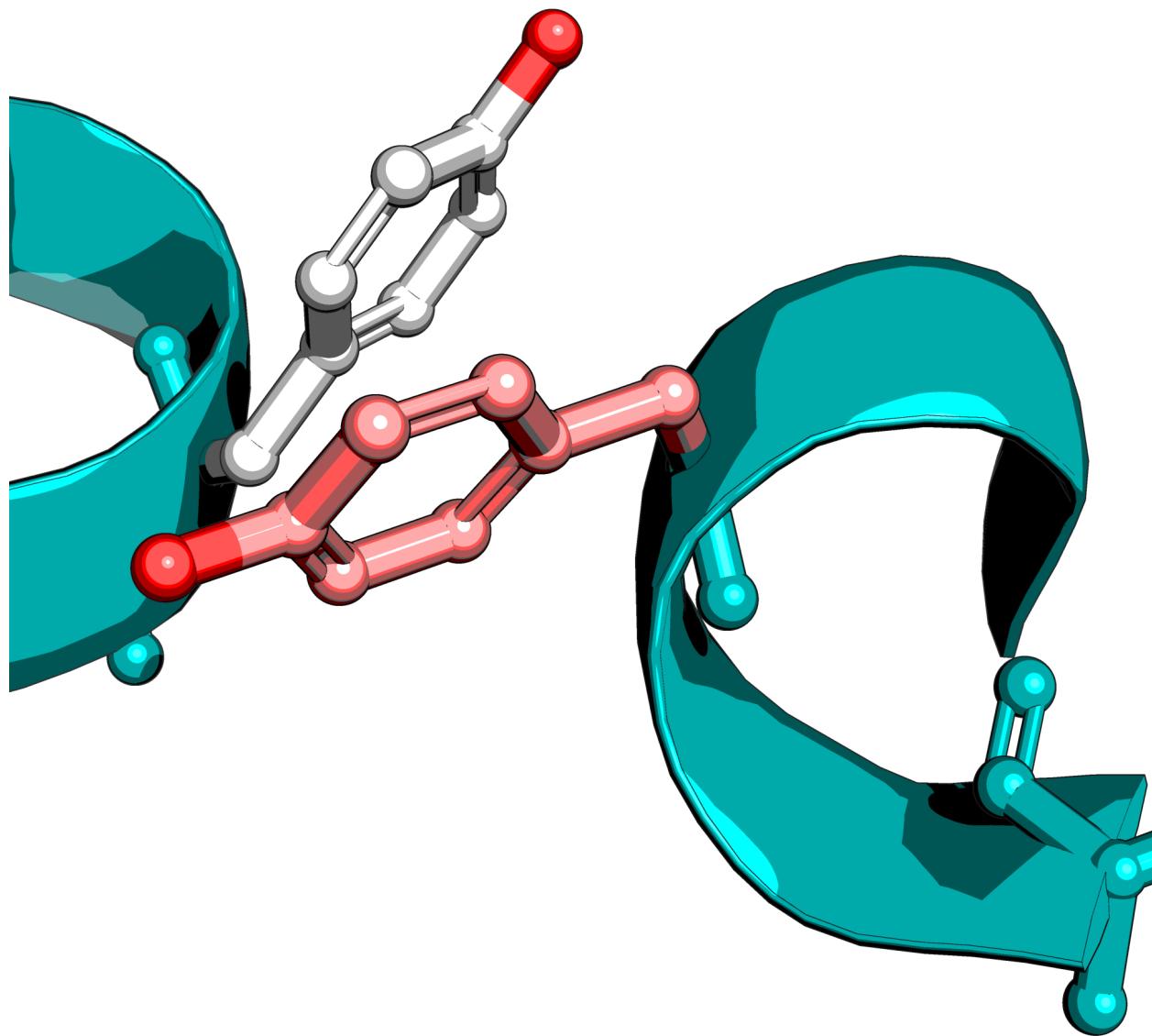
```

(continues on next page)

(continued from previous page)

```

35 * KEYWORDS: PDB input; PDB line filter; stacking interactions
36 * GROUP: Structure calculations;
37 * IMG: ap_stacking_interactions_sq.png
38 * IMG_ALT: Two tyrosine residues in stacking interaction
39 */
40 int main(const int argc, const char *argv[]) {
41
42     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
43     //missing program parameter
44     utils::LogManager::INFO(); // --- INFO is the default logging
45     //level; set it to FINE to see more
46
47     // ----- Read a PDB file given as an argument to this program
48     Pdb reader(argv[1], // --- input PDB file
49     all_true(is_not_water, is_not_alternative, is_not_hydrogen,
50     invert_filter(is_bb)), // --- Inverted backbone selector
51     //reads only side chains
52     core::data::io::only_ss_from_header, true); // --- yes,
53     //read header
54
55     core::data::structural::Structure_SP s = reader.create_structure(0);
56
57     // ----- Register additional monomers, provided by a user from a command line,
58     //either .pdb or .cif
59     for (int i = 2; i < argc; ++i)
60         MonomerStructureFactory::get_instance().register_monomer(argv[i]);
61
62     StackingInteractionCollector collector=StackingInteractionCollector();
63     std::vector<ResiduePair_SP> sink;
64     collector.collect(*s,sink);
65     std::cout << StackingInteraction::output_header() << "\n";
66
67     for (const ResiduePair_SP ri:sink) {
68         StackingInteraction_SP bi = std::dynamic_pointer_cast<StackingInteraction>(ri);
69         if (bi) std::cout << *bi << "\n";
70     }
71 }
```



ap_vdw_interactions

ap_vdw_interactions finds all van der Waals interactions in a given protein structure.

USAGE:

```
ap_vdw_interactions input.pdb [input2.pdb ...]
```

EXAMPLE:

```
ap_vdw_interactions 2gb1.pdb
```

OUTPUT (fragment):

Keywords:

- *PDB input*
- *interactions*

Categories:

- core::calc::structural::interactions::VdWInteraction

Input files:

- 2kwi.pdb
- 2gb1.pdb
- 5edw.pdb

Output files:

- stdout.out

Program source:

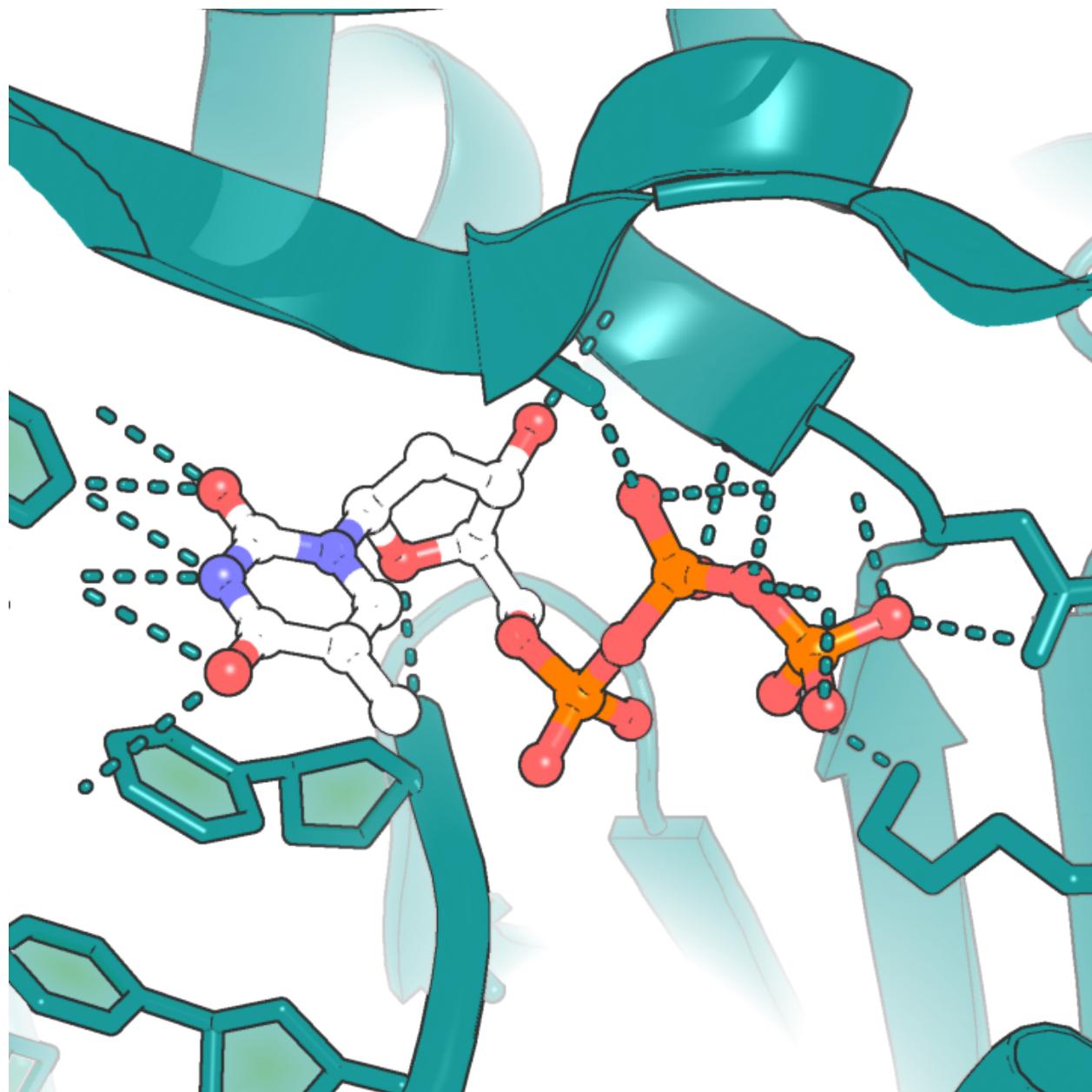
```
1 #include <iostream>
2 #include <map>
3
4 #include <core/data/io/Pdb.hh>
5 #include <core/calc/structural/interactions/VdWInteraction.hh>
6 #include <core/calc/structural/interactions/VdWInteractionCollector.hh>
7 #include <utils/string_utils.hh>
8 #include <utils/exit.hh>
9
10 std::string program_info = R"(
11
12 ap_vdw_interactions finds all van der Waals interactions in a given protein structure.
13
14
15 USAGE:
16     ap_vdw_interactions input.pdb [input2.pdb ...]
17
18 EXAMPLE:
19     ap_vdw_interactions 2gb1.pdb
20
21 OUTPUT (fragment):
22
23
24 )";
25
26 /** @brief Finds all van der Waals interactions in a given protein structure.
27 *
28 *  * CATEGORIES: core::calc::structural::interactions::VdWInteraction
29 *  * KEYWORDS: PDB input; interactions
30 *  * GROUP: Structure calculations;
```

(continues on next page)

(continued from previous page)

```

31 * IMG: ap_ligand_contacts.png
32 * IMG_ALT: Contacts found between 5EDW protein and its ligand TTP
33 */
34 int main(const int argc, const char* argv[]) {
35
36     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
37     ↵missing program parameter
38
39     using namespace core::calc::structural::interactions;
40
41     VdWInteractionCollector collector;
42     for (size_t i_protein = 1; i_protein < argc; ++i_protein) { // --- Iterate over all_
43     ↵models in the input file
44     core::data::io::Pdb reader(argv[i_protein]); // --- file name (PDB format, may be_
45     ↵gzip-ped)
46
47     for (size_t i_model = 0; i_model < reader.count_models(); ++i_model) { // ---_
48     ↵Iterate over all models in the input file
49     std::vector<ResiduePair_SP> sink;
50     core::data::structural::Structure_SP strctr = reader.create_structure(i_model);
51     collector.collect(*strctr, sink);
52
53     std::cout << VdWInteraction::output_header() << "\n";
54     for (const ResiduePair_SP ri:sink) {
55         VdWInteraction_SP bi = std::dynamic_pointer_cast<VdWInteraction>(ri);
56         if (bi) std::cout << *bi << "\n";
57     }
58 }
59 }
```



ap_AAHydrophobicity

Reads a PDB file and substitutes b-factor column with hydrophobicity values according to Kyte-Doolittle scale. If just a PDB file is given as an input, all b-factors will be replaced by respective KD hydrophobicity values. User can also provide a Multiple Sequence Alignment (MSA) in ClustalO format (.aln); hydrophobicity values will be averaged over a corresponding column of the MSA. In that case the sequence from the given PDB file must also be included in the alignment; its name is third argument of the program.

USAGE:

```
ap_AAHydrophobicity input.pdb  
ap_AAHydrophobicity input.pdb input.aln sequence-id
```

EXAMPLE

```
ap_AAHydrophobicity 2gb1.pdb
ap_AAHydrophobicity 2gb1.pdb 2gb1.aln 2GB1
```

REFERENCE: Kyte, Jack, and Russell F. Doolittle. "A simple method for displaying the hydropathic character of a protein." Journal of molecular biology 157.1 (1982): 105-132. doi: 10.1016/0022-2836(82)90515-0

Keywords:

- *PDB input*
- hydrophobicity
- *structure selectors*
- *PDB line filter*
- *sequence alignment*
- MSA input

Categories:

- core/chemical/AAHydrophobicity

Input files:

- 2gb1.pdb
- CYP109B1.aln
- 4rm4.pdb

Output files:

- stdout.out
- 2gb1.out.pdb

Program source:

```

1 #include <iostream>
2 #include <iomanip>
3
4 #include <core/algorithms/predicates.hh>
5 #include <core/alignment/NWAligner.hh>
6 #include <core/alignment/scoring/SimilarityMatrix.hh>
7 #include <core/alignment/scoring/SimilarityMatrixScore.hh>
8 #include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>
9 #include <core/chemical/AAHydrophobicity.hh>
10 #include <core/data/io/Pdb.hh>
11 #include <core/data/io/clustalw_io.hh>
12
13 #include <utils/exit.hh>
14 #include <core/data/structural/selectors/structure_selectors.hh>
```

(continues on next page)

(continued from previous page)

```
15 std::string program_info = R"(  
16  
17 Reads a PDB file and substitutes b-factor column with hydrophobicity values according to Kyte-Doolittle scale.  
18  
19 If just a PDB file is given as an input, all b-factors will be replaced by respective KD hydrophobicity values.  
20 User can also provide a Multiple Sequence Alignment (MSA) in ClustalO format (.aln); hydrophobicity values will be  
21 averaged over a corresponding column of the MSA. In that case the sequence from the given PDB file  
22 must also be included in the alignment; its name is third argument of the program.  
23  
24 USAGE:  
25     ap_AAHydrophobicity input.pdb  
26     ap_AAHydrophobicity input.pdb input.aln sequence-id  
27  
28 EXAMPLE:  
29     ap_AAHydrophobicity 2gb1.pdb  
30     ap_AAHydrophobicity 2gb1.pdb 2gb1.aln 2GB1  
31  
32 REFERENCE:  
33 Kyte, Jack, and Russell F. Doolittle. "A simple method for displaying the hydropathic character of a protein."  
34 Journal of molecular biology 157.1 (1982): 105-132. doi: 10.1016/0022-2836(82)90515-0  
35 )";  
36  
37 /** @brief Reads a PDB file and substitutes b-factor column with hydrophobicity values according to Kyte-Doolittle scale. This example prints atoms for each side chain in a protein  
38 *  
39 * CATEGORIES: core/chemical/AAHydrophobicity;  
40 * KEYWORDS: PDB input; hydrophobicity; structure selectors; PDB line filter; sequence alignment; MSA input  
41 * GROUP: Sequence calculations  
42 */  
43 int main(const int argc, const char *argv[]) {  
44  
45     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter  
46  
47     using namespace core::data::io; // is_not_alternative, Pdb and read_clustalw_file() are from this namespace  
48     using namespace core::data::sequence;  
49     using namespace core::data::structural; // Structure and Residue come from here  
50  
51     using namespace core::alignment::scoring;  
52  
53     Pdb reader(argv[1], all_true(is_not_alternative, is_not_water));  
54     Structure_SP strctr = reader.create_structure(0); // create a Structure object from the first model found in the input file  
55     Chain & first_chain = *(strctr)[0]; // --- We assume the first chain is the one used in MSA  
56     first_chain.erase(std::remove_if(first_chain.begin(), first_chain.end(),  
57                         core::algorithms::Not<selectors::IsAA>(selectors::IsAA())), first_chain.end());  
58  
59 }
```

(continues on next page)

(continued from previous page)

```

60     std::vector<double> kd_values;
61     const core::chemical::AAHydrophobicity &kd_scale =
62     ↪core::chemical::AAHydrophobicity::KyteDoolittle;
63
64     std::ofstream out("out.pdb");
65     // ----- The case when we have both a PDB file and a multiple sequence
66     ↪alignment (.aln file)
67     if (argc == 4) {
68         std::vector<Sequence_SP> msa; // --- placeholder for aligned sequences
69         core::data::io::read_clustalw_file(argv[2], msa); // --- read the MSA and store
70     ↪sequences in a vector
71
72         // ----- Find the reference sequence in the alignment
73         std::string ref_sequence_name(argv[3]); // --- the name of the sequence
74         auto s = std::find_if(msa.begin(), msa.end(),
75             [&ref_sequence_name](Sequence_SP s) { return s->header().find(ref_sequence_
76             name) != std::string::npos; });
77         if (s == msa.end())
78             utils::exit_OK_with_message(
79                 "Can't find the reference sequence in the given MSA. Is the name correct: " +
80             ref_sequence_name);
81         Sequence_SP ref_sequence = *s;
82
83         // --- Create a sequence object for the first chain of the PDB deposit
84         core::data::sequence::SecondaryStructure_SP pdb_seq = first_chain.create_
85     ↪sequence();
86
87         // ----- we have to align the reference sequence with the sequence found in
88     ↪the given PDB file
89         // ----- as they might differ; we set PDB sequence to be a query and the
90     ↪reference - as a template
91         unsigned max_len = std::max(pdb_seq->length(), ref_sequence->length());
92         core::alignment::NWAligner<short, SimilarityMatrixScore<short>> aligner(max_len);
93         NcbiSimilarityMatrix_SP sim_m = NcbiSimilarityMatrixFactory::get().get_matrix(
94     ↪"BLOSUM62");
95         SimilarityMatrixScore<short> score(pdb_seq->sequence, ref_sequence->sequence,
96     ↪*sim_m);
97         aligner.align(-10, -1, score);
98         auto alignment = aligner.backtrace();
99
100        std::cout << "#msa_col aa_col aa_res_id : avg_KD n_aa\n";
101        // ----- Iterate over all columns of the MSA
102        for (core::index2 i_res = 0; i_res < ref_sequence->length(); ++i_res) {
103            if (ref_sequence->get_monomer(i_res).is_gap()) continue;
104            int j = alignment->which_query_for_template(i_res); // --- -1 denotes a gap,
105     ↪otherwise the index is non-negative
106            if (j < 0) continue;
107            double avg_kd = 0;
108            double n = 0;
109            for (Sequence_SP si:msa) {
110                if (!si->get_monomer(i_res).is_gap()) {
111                    avg_kd += kd_scale.hydrophobicity(si->get_monomer(i_res));
112                    ++n;
113                }
114            }
115            std::cout
116                << utils::string_format("%4d %4d %c %4d : %5.2f %3d\n", i_res, j, first_
117     ↪chain[j]->residue_type().code1,
```

(continues on next page)

(continued from previous page)

```

106     first_chain[j]->id(), avg_kd / n, int(n));
107     avg_kd = avg_kd / n + 5.0; // --- we add 5.0 because KD scale is from -4.5 to 4.
108     ↪5 and b-factor can't be negative
109     for (const PdbAtom_SP &a : *(first_chain[j])) {
110         a->b_factor(avg_kd);
111         out << a->to_pdb_line() << "\n";
112     }
113
114     return 0;
115 }
116
117 // ----- The case when we have only PDB file : iIterate over all residues in
118 ↪the structure
119 for (auto res_it = strctr->first_residue(); res_it != strctr->last_residue(); ++res_
120 ↪it) {
121     double val = kd_scale.hydrophobicity((*res_it)->residue_type()) + 5.0;
122
123     for (const PdbAtom_SP &a : **res_it) {
124         a->b_factor(val);
125         out << a->to_pdb_line() << "\n";
126     }
127     out.close();
}

```



ap_AlignmentPValuesProtocol

ap_AlignmentPValuesProtocol calculates each-vs-each pairwise semiglobal alignments between protein sequences read from a given input file. p-value for every alignment is estimated based on re-shuffled statistics (30 randomly shuffled alignments are calculated)

USAGE:

```
ap_AlignmentPValuesProtocol input.fasta
```

EXAMPLE:

```
ap_AlignmentPValuesProtocol small50_95identical.fasta
```

Keywords:

- *FASTA input*
- *sequence alignment*
- *statistics*

Categories:

- core/protocols/AlignmentPValuesProtocol.hh

Input files:

- small50_95identical.fasta
- small500_95identical.fasta

Output files:

- stdout.out

Program source:

```
1 #include <utils/exit.hh>
2 #include <core/alignment/aligner_factory.hh>
3 #include <core/data/basic/Array2D.hh>
4 #include <core/data/sequence/Sequence.hh>
5 #include <core/protocols/AlignmentPValuesProtocol.hh>
6 #include <core/data/io/fasta_io.hh>
7 #include <core/data/basic/SparseMap2D.hh>
8
9 std::string program_info = R"(
10
11 ap_AlignmentPValuesProtocol calculates each-vs-each pairwise semiglobal alignments
12 between protein sequences read from a given input file. p-value for every alignment
13 is estimated based on re-shuffled statistics (30 randomly shuffled alignments are
14 calculated)
15 )"
```

(continues on next page)

(continued from previous page)

```

16 USAGE:
17     ap_AlignmentPValuesProtocol input.fasta
18
19 EXAMPLE:
20     ap_AlignmentPValuesProtocol small1500_95identical.fasta
21
22 ) ";
23
24 /** @brief Uses AlignmentPValuesProtocol protocol to calculate all pairwise p-values
25  * for a given set of sequences
26 *
27 * CATEGORIES: core/protocols/AlignmentPValuesProtocol.hh
28 * KEYWORDS: FASTA input; sequence alignment; statistics
29 * GROUP: Alignments
30 */
31
32 int main(const int argc, const char* argv[]) {
33
34     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
35     //missing program parameter
36
37     using namespace core::data::sequence;
38     using namespace core::protocols;
39     using namespace core::alignment;
40
41     core::protocols::AlignmentPValuesProtocol protocol;
42     protocol.gap_open(-10).gap_extend(-1).substitution_matrix("BLOSUM62").keep_
43     //alignments(true).
44     alignment_method(AlignmentType::SEMIGLOBAL_ALIGNMENT).keep_alignments(true).n_
45     //threads(4);
46     protocol.n_shuffles(30).p_value_cutoff(0.01);
47
48     std::vector<Sequence_SP> input_sequences;
49     core::data::io::read_fasta_file(argv[1], input_sequences);
50     for(Sequence_SP si: input_sequences) protocol.add_input_sequence(si);
51
52     auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
53     protocol.run();
54     auto end = std::chrono::high_resolution_clock::now();
55     std::chrono::duration<double> time_span = std::chrono::duration_cast
56     //<std::chrono::duration<double>>(end - start);
57     std::cerr << input_sequences.size() * (input_sequences.size() - 1) / 2.0
58     << " global alignment sequence similarities calculated within " << time_
59     //span.count() << " [s]\n";
60
61     protocol.print_p_values(std::cout);
62 }
```



ap_LigandsOnGridProtocol

ap_LigandsOnGridProtocol reads a list of pdb files and creates grid with ligands in it.

USAGE:

```
ap_LigandsOnGridProtocol box_grid_width models-list
```

Keywords:

- *PDB input*
- :ref:``

Categories:

- core::protocols::LigandsOnGridProtocol

Input files:

- 1001_0001.pdb
- 1001_0003.pdb
- 1001_0002.pdb
- cat_list
- 2hpl_ref_pepdock.pdb

Output files:

- stdout.out
- all.pdb

Program source:

```

1 #include <vector>
2 #include <string>
3 #include <fstream>
4 #include <iostream>
5
6 #include <cstring>
7
8
9 #include <core/data/io/Pdb.hh>
10 #include <utils/io_utils.hh>
11
12 #include <utils/options/output_options.hh>
13 #include <utils/options/input_options.hh>
14 #include <core/protocols/LigandsOnGridProtocol.hh>
15 #include <utils/exit.hh>
16

```

(continues on next page)

(continued from previous page)

```

17  using namespace core::data::io;           // PDB is from this namespace
18  using namespace core::data::structural;
19  using namespace core::data::structural::selectors;
20  using namespace core::calc::structural;
21  using namespace utils;
22  using namespace std;
23
24
25  std::string program_info = R"(
26
27 ap_LigandsOnGridProtocol reads a list of pdb files and creates grid with ligands in it.
28
29 USAGE:
30     ap_LigandsOnGridProtocol box_grid_width models-list
31 )";
32
33 /** @brief Reads list of pdb files and creates grid with ligands in it.
34 */
35 * The first model on list (index = 0) is the representative one.
36 *
37 * CATEGORIES: core::protocols::LigandsOnGridProtocol
38 * GROUP: Structure calculations; Docking;
39 * KEYWORDS: PDB input;
40 */
41 int main(const int argc, const char *argv[]) {
42
43     if (argc < 3) utils::exit_OK_with_message(program_info);
44
45     std::vector<std::string> pdb_files; //vector of string file names
46     utils::read_listfile(argv[2], pdb_files);
47     // assumes that ligand is in B chain
48     AtomSelector_SP select_ligand = std::static_pointer_cast<AtomSelector>(std::make_
49     ↪shared<ChainSelector>("B"));
49     // assumes that receptor is in A chain
50     AtomSelector_SP select_receptor = std::static_pointer_cast<AtomSelector>(std::make_
51     ↪shared<ChainSelector>("A"));
51     // creating LigandsOnGridProtocol object
52     core::protocols::LigandsOnGridProtocol ligpro =_
53     ↪core::protocols::LigandsOnGridProtocol(select_ligand, select_receptor);
53     // setting box_grid_size
54     ligpro.box_grid_size(atof(argv[1]));
55     PdbLineFilter filter = core::data::io::is_ca;
56     Pdb pdb = Pdb(pdb_files[0], filter);
57     //creating structure from first file on a list
58     Structure_SP strctr = pdb.create_structure(0);
59     // adding structure to LigandsOnGridProtocol object
60     ligpro.add_input_structure(strctr);
61     // loading and adding rest of structures from cat_list
62     for (int i = 1; i < pdb_files.size(); i++) {
63         Pdb pdb = Pdb(pdb_files[i], filter);
64         pdb.fill_structure(0, *strctr);
65         // std::cout << pdb_files[i] << "\n";
66         ligpro.add_input_structure(strctr);
67     }
68
69     // running the calculation to put ligands into grid

```

(continues on next page)

(continued from previous page)

```

70 ligpro.calculate();
71 // creating a copy of a vector with hashes from filled grid cells
72 std::vector<core::index4> grid_cells = ligpro.grid()->filled_cells();
73
74 core::index4 index = 0; // variable to remember iterator for biggest cell
75 int size = 100; //variable to remember SIZE
76 while (grid_cells.size() > 0 and size >= 10) { //until vector is not empty and
    ↪there are cells bigger than 10
77     size = 0;
78     for (core::index4 i = 0; i < grid_cells.size(); i++) { // iterating over cells
79         //std::cout<<grid_cells[i]<<" "<<ligpro.grid()->get_cell(grid_cells[i]).size()<<
    ↪"\n";
80         if (ligpro.grid()->get_cell(grid_cells[i]).size() > size) { //checking if
    ↪current cell size is bigger than SIZE
81             size = ligpro.grid()->get_cell(grid_cells[i]).size(); //if yes, changing size
    ↪and index values
82             index = grid_cells[i];
83         }
84     }
85     if (size >= 10) {
86         // std::cout<<index<<" "<<ligpro.grid()->get_cell(index).size()<<"\n";
87
88         std::vector<core::index4> hashes; //vector to store neighbor cells
89         ligpro.grid()->get_neighbor_cells(index, hashes); //getting all hashes for
    ↪neighbors cells
90         for (core::index4 ind = 0; ind < hashes.size(); ind++) {
91             grid_cells.erase(std::remove(grid_cells.begin(), grid_cells.end(),
    ↪hashes[ind]),
92                             grid_cells.end()); //attepmt to erase biggest cell from the
    ↪vector
93         }
94
95         std::ofstream of(utils::to_string(index) + ".out");
96         std::vector<core::data::structural::PdbAtom_SP> sink;
97         ligpro.grid()->get_neighors(index, sink);
98         for (core::index4 a = 0; a < sink.size(); a++) { //iterating over Atoms in sink
99             of << sink[a]->id() << "\n"; //writing to file
100        }
101        of.close();
102    }
103 }
104 }
```



ap_LocalStructureMatch

Finds contiguous structural segments that are similar between two structures. The program creates contiguous structural segments of 5 or 7 CA atoms based on C-alpha coordinates from file1 and file2 (PDB format). The segment size must be given as the first input parameter. Then it looks for segments that are structurally similar by computing LocalStructureMatch distance between them. This value is defined as a squared difference between local inter-atomic distances. A small value means local structural similarity between respective segments. The last (optional) parameter is the maximum value of a LocalStructureMatch distance to be printed.

USAGE:

```
./ap_LocalStructureMatch (5 or 7) file1.pdb file2.pdb [max_distance]
```

EXAMPLE:

```
./ap_LocalStructureMatch 7 4rm4A.pdb 5ofqA.pdb 9.0
```

Keywords:

- *PDB input*
- structure match

Categories:

- core/alignment/scoring/LocalStructureMatch

Input files:

- 5ofq.pdb
- 4rm4.pdb

Output files:

- stdout.out

Program source:

```

1 #include <ctime>
2 #include <iostream>
3 #include <sstream>
4
5 #include <utils/Logger.hh>
6 #include <utils/io_utils.hh>
7
8 #include <utils/options/Option.hh>
9 #include <utils/options/OptionParser.hh>
10 #include <utils/options/input_options.hh>
11
12 #include <core/data/basic/Vec3.hh>
13 #include <core/data/io/Pdb.hh>
```

(continues on next page)

(continued from previous page)

```

14 #include <core/alignment/scoring/LocalStructure7.hh>
15 #include <core/alignment/scoring/LocalStructure5.hh>
16 #include <core/alignment/scoring/LocalStructureMatch.hh>
17 #include <utils/exit.hh>
18
19 utils::Logger l("ap_LocalStructureMatch");
20
21 std::string program_info = R"
22
23 Finds contiguous structural segments that are similar between two structures.
24
25 The program creates contiguous structural segments of 5 or 7 CA atoms based on C-
26 ↵alpha coordinates from file1 and file2
27 (PDB format). The segment size must be given as the first input parameter. Then it_
28 ↵looks for segments
29 that are structurally similar by computing LocalStructureMatch distance between them.↵
30 ↵This value is defined as
31 a squared difference between local inter-atomic distances. A small value means local_
32 ↵structural similarity between
33 respective segments. The last (optional) parameter is the maximum value of a_
34 ↵LocalStructureMatch distance to be printed.
35
36 USAGE:
37   ./ap_LocalStructureMatch (5 or 7) file1.pdb file2.pdb [max_distance]
38
39 EXAMPLE:
40   ./ap_LocalStructureMatch 7 4rm4A.pdb 5ofqA.pdb 9.0
41
42 ) ";
43
44 /**
45 * @brief Finds contiguous structural segments that are similar between two_
46 ↵structures
47 *
48 * CATEGORIES: core/alignment/scoring/LocalStructureMatch;
49 * KEYWORDS: PDB input; structure match
50 * GROUP: Alignments
51 */
52 int main(const int argc, const char *argv[]) {
53
54   if(argc < 4) utils::exit_OK_with_message(program_info); // --- complain about_
55 ↵missing program parameter
56   int match_size = atoi(argv[1]);
57
58   double max_print_distance = (argc==5) ? atof(argv[4]) : 100000.0;
59   using namespace core::alignment::scoring; // --- for LocalStructure7,_
60 ↵LocalStructure5 and LocalStructureMatch
61   using namespace core::data::basic; // --- for Coordinates_SP and Vec3
62   Coordinates_SP xyz_q = std::make_shared<std::vector<Vec3>>();
63   Coordinates_SP xyz_t = std::make_shared<std::vector<Vec3>>();
64   core::data::io::Pdb::read_coordinates(argv[2], *xyz_q, true, core::data::io::is_ca);
65   if (match_size == 7) {
66     LocalStructure7 local_query(xyz_q);
67     core::data::io::Pdb::read_coordinates(argv[3], *xyz_t, true, core::data::io::is_
68 ↵ca);
69     LocalStructure7 local_tmplt(xyz_t);
70     LocalStructureMatch<LocalStructure7, 8> lm(local_query, local_tmplt);
71     lm.print(std::cout, max_print_distance);
72 }
```

(continues on next page)

(continued from previous page)

```
62 } else if (match_size == 5) {  
63     LocalStructure5 local_query(xyz_q);  
64     core::data::io::Pdb::read_coordinates(argv[3], *xyz_t, true, core::data::io::is_  
65     ↵ca);  
66     LocalStructure5 local_tmplt(xyz_t);  
67     LocalStructureMatch<LocalStructure5, 8> lm(local_query, local_tmplt);  
68     lm.print(std::cout, max_print_distance);  
69 }
```



ap_MC_water

The program runs an isothermal MC simulation of water. By default it starts from a regular lattice conformation unless an input file (PDB) with initial conformation is provided

USAGE:

```
ap_MC_water n_molecules temperature small_cycles big_cycles
ap_MC_water starting.pdb temperature small_cycles big_cycles
```

Keywords:

- no_keywords

Categories:

- no_categories

Output files:

- water_tra.pdb
- movers.dat
- final.pdb

Program source:

```

1 #include <iostream>
2 #include <vector>
3 #include <string>
4
5 #include <core/data/basic/Vec3I.hh>
6 #include <core/BioShellVersion.hh>
7
8 #include <utils/string_utils.hh>
9 #include <utils/options/Option.hh>
10 #include <utils/options/OptionParser.hh>
11 #include <utils/options/output_options.hh>
12 #include <utils/options/sampling_options.hh>
13
14 #include <simulations/systems/CartesianChains.hh>
15 #include <simulations/systems/BuildFluidSystem.hh>
16 #include <simulations/movers/RotateRigidMolecule.hh>
17 #include <simulations/movers/TranslateMolecule.hh>
18 #include <simulations/movers/MoversSetSweep.hh>
19 #include <simulations/forcefields/mm/Water3PointEnergy.hh>
20 #include <simulations/forcefields/mm/WaterModelParameters.hh>
21
22 #include <simulations/sampling/IsothermalMC.hh>
23 #include <simulations/observers/ObserveEvaluators.hh>
24 #include <simulations/observers/cartesian/PdbObserver.hh>
25 #include <simulations/observers/ObserveMoversAcceptance.hh>
```

(continues on next page)

(continued from previous page)

```

26 #include <simulations/observers/TriggerEveryN.hh>
27 #include <simulations/observers/AdjustMoversAcceptance.hh>
28 #include <simulations/observers/cartesian/ExplicitPdbFormatter.hh>
29 #include <simulations/evaluators/CallEvaluator.hh>
30 #include <simulations/systems/SimpleAtomTyping.hh>
31
32
33 using namespace core::data::basic;
34
35 utils::Logger logs("ap_MC_water");
36
37 std::string program_info = R"
38
39 The program runs an isothermal MC simulation of water. By default it starts from a_
40 ↪regular lattice conformation
41 unless an input file (PDB) with initial conformation is provided
42 USAGE:
43     ap_MC_water n_molecules temperature small_cycles big_cycles
44     ap_MC_water starting.pdb temperature small_cycles big_cycles
45 ) ";
46
47 /** @brief Isothermal Monte Carlo simulation of water.
48 */
49
50 int main(const int argc,const char* argv[]) {
51
52     using core::data::basic::Vec3Cubic;
53     using namespace simulations::systems;
54     using namespace simulations::movers; // for MoversSet
55     using namespace simulations::observers::cartesian; // for all observers
56     using simulations::forcefields::WaterModelParameters;
57
58     logs << utils::LogLevel::INFO << "BioShell version:\n" << core::BioShellVersion().to_string() << "\n";
59
60     core::index4 n_outer_cycles = 1000;
61     core::index4 n_inner_cycles = 10;
62     double temperature = 298; // in Kelvins
63     core::index4 n_molecules = 216; // 216
64     core::calc::statistics::Random::seed(1234);
65
66     double water_density = 0.99823;
67     double water_mass = 18.01528;
68     core::data::structural::Structure_SP water_structure = nullptr;
69
70     if (argc < 5) std::cerr << program_info;
71     else {
72         if (utils::is_integer(argv[1])) n_molecules = atoi(argv[1]);
73         else { // --- read an input file if given
74             core::data::io::Pdb reader(argv[1]);
75             water_structure = reader.create_structure();
76             n_molecules = water_structure->count_residues();
77         }
78         temperature = atof(argv[2]);
79         n_inner_cycles = atoi(argv[3]);
80         n_outer_cycles = atoi(argv[4]);

```

(continues on next page)

(continued from previous page)

```

81 }
82 double water_volume = n_molecules * 10 * water_mass/6.02214;
83 double box_len = pow(water_volume / water_density, 0.333333333333333);
84
85 // --- Initialize periodic boundary conditions
86 core::data::basic::Vec3I::set_box_len(box_len);
87 logs << utils::LogLevel::INFO << "box width for " << int(n_molecules) << " molecules : " << box_len << "\n";
88
89 WaterModelParameters::load_models();
90 WaterModelParameters tip3p = WaterModelParameters::get_model("TIP3P");
91
92 // --- Create water structure if not loaded from PDB
93 if (water_structure == nullptr) {
94     core::data::structural::Residue_SP hoh = tip3p.create_residue();
95     core::data::structural::Residue_SP water_molecule = std::make_shared<core::data::structural::Residue>(1, "HOH");
96     PointGridGenerator_SP grid = std::make_shared<SimpleCubicGrid>(box_len, n_molecules);
97     water_structure = BuildFluidSystem::build_structure(*hoh, grid);
98 }
99
100 // SimpleAtomTyping tip3p_typing({"HOH"}, {"O", "H"}, {" O ", " H "});
101 // --- Create the system to be sampled
102 std::vector<std::string> res_types {"HOH"};
103 std::vector<std::string> atom_types {"O", "H"};
104 std::vector<std::string> pdb_types {" O ", " H "};
105 AtomTypingInterface_SP tip3p_typing = std::make_shared<SimpleAtomTyping>(res_types, atom_types, pdb_types);
106 CartesianChains system(tip3p_typing, *water_structure);
107 CartesianChains backup(system);
108
109 // --- Create energy function - TIP3P potential
110 simulations::forcefields::mm::Water3PointEnergy en(tip3p);
111
112 // --- Create a mover, which is a random perturbation of an atom in this case, and place it in a movers' set
113 std::shared_ptr<RotateRigidMolecule> rot = std::make_shared<RotateRigidMolecule>(system, backup, en, 0);
114 std::shared_ptr<TranslateMolecule> trs = std::make_shared<TranslateMolecule>(system, backup, en);
115 MoversSet_SP movers = std::make_shared<simulations::movers::MoversSetSweep>();
116 movers->add_mover(rot, n_molecules);
117 movers->add_mover(trs, n_molecules);
118
119 // --- create an isothermal Monte Carlo sampler
120 simulations::sampling::IsothermalMC mc(movers, temperature);
121
122 // ----- Create an observer which calls energy calculation and prints it on the screen
123 std::shared_ptr<simulations::observers::ObserveEvaluators> obs = std::make_shared<simulations::observers::ObserveEvaluators>("");
124 // ----- Create an observer which calls energy calculation and prints it to a file
125 // std::shared_ptr<simulations::observers::ObserveEvaluators> obs = std::make_shared<simulations::observers::ObserveEvaluators>"energy.dat");
126 std::function<double(void)> recent_energy = [&en, &system] () { return en.energy(system) / (system.count_residues() * 1000); };

```

(continues on next page)

(continued from previous page)

```
127     obs->add_evaluator(
128         std::make_shared<simulations::evaluators::CallEvaluator<std::function
129             ↪<double (void)>>>(recent_energy, "energy", 8));
130
131     std::shared_ptr<simulations::observers::AdjustMoversAcceptance> observe_moves
132         = std::make_shared<simulations::observers::AdjustMoversAcceptance>(*movers,
133             ↪"movers.dat", 0.4);
134     observe_moves->observe_header();
135
136     std::shared_ptr<AbstractPdbFormatter> fmt = std::make_shared<ExplicitPdbFormatter>
137             ↪(*water_structure);
138     auto observe_trajectory = std::make_shared
139             ↪<simulations::observers::cartesian::PdbObserver>(system, fmt, "water_tra.pdb");
140 //   observe_trajectory->trigger(std::make_shared
141 //       ↪<simulations::observers::TriggerEveryN>(10));
142     mc.outer_cycle_observer(observe_trajectory); // --- commented out to save disk space
143     mc.outer_cycle_observer(observe_moves);
144     mc.outer_cycle_observer(obs);
145     mc.cycles(n_inner_cycles,n_outer_cycles,1);
146
147     mc.run();
148
149     simulations::observers::cartesian::PdbObserver final(system, fmt, "final.pdb");
150     final.observe();
151 //   logs << utils::LogLevel::INFO << "Final energy " << lj_energy.calculate() << "\n";
152 }
```



ap_MSAColumnConservation

Reads a Multiple Sequence Alignment (MSA) in ClustalW or FASTA format and evaluates sequence conservation for every column.

USAGE:

```
./ap_MSAColumnConservation msa-file [sequence-id]
```

EXAMPLE:

```
./ap_MSAColumnConservation cyped.CYP109.aln M5R670_9BACI
```

where cyped.CYP109.aln is the name of input MSA file (.aln or .fasta format). If the sequence identifier is given as a second optional argument (here: M5R670_9BACI), program will attempt to find the sequence annotated with this name. When such a sequence is found, additional column will be added to provide residue for every position in that sequence (gaps are also shown).

Keywords:

- *clustal input*
- *MSA*
- *FASTA input*

Categories:

- core::alignment::MSAColumnConservation

Input files:

- conservation_test_msa.aln
- CYP109B1.aln

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/alignment/MSAColumnConservation.hh>
4 #include <core/data/io/clustalw_io.hh>
5 #include <utils/exit.hh>
6 #include <utils/io_utils.hh>
7 #include <core/data/io/fasta_io.hh>
8
9 std::string program_info = R"(
10
11 Reads a Multiple Sequence Alignment (MSA) in ClustalW or FASTA format and evaluates_
12 sequence conservation for every column.
```

(continues on next page)

(continued from previous page)

```

12
13 USAGE:
14 ./ap_MSAColumnConservation msa-file [sequence-id]
15
16 EXAMPLE:
17 ./ap_MSAColumnConservation cyped.CYP109.aln M5R670_9BACI
18
19 where cyped.CYP109.aln is the name of input MSA file (.aln or .fasta format). If the_
20 ↪sequence identifier
21 is given as a second optional argument (here: M5R670_9BACI), program will attempt to_
22 ↪find the sequence
23 annotated with this name. When such a sequence is found, additional column will be_
24 ↪added to provide residue
25 for every position in that sequence (gaps are also shown).
26
27 )";
28
29 /**
30 * CATEGORIES: core::alignment::MSAColumnConservation
31 * KEYWORDS: clustal input; MSA; FASTA input
32 * GROUP: Alignments
33 */
34 int main(const int argc, const char* argv[]) {
35
36     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
37 ↪missing program parameter
38
39     using namespace core::data::io;
40     using namespace core::data::sequence;
41
42     std::vector<Sequence_SP> msa;    // --- Sequence_SP is just a shorter name for_
43 ↪std::shared_ptr<Sequence>
44     const std::pair<std::string, std::string> name_ext = utils::root_extension(argv[1]);
45     if((name_ext.second=="fasta") || (name_ext.second=="FASTA") || (name_ext.second=="fast"
46 ↪"))
47         core::data::io::read_fasta_file(argv[1], msa, true);
48     else
49         core::data::io::read_clustalw_file(argv[1],msa);
50
51     std::string seq_str( msa[0]->length(), ' ');
52     std::string seq_name = (argc > 2) ? argv[2] : "";
53     bool sequence_found = false;
54     if (seq_name.size() > 0) {
55         for (const auto &seq:msa)
56             if (seq->header().find(argv[2]) != std::string::npos) {
57                 seq_str = seq->sequence;
58                 sequence_found = true;
59             }
60         if (!sequence_found) std::cerr << "Warning: the sequence >" << seq_name << "< can
61 ↪'t be located!\n";
62     }
63
64     core::alignment::MSAColumnConservation consrv(msa);
65     if (sequence_found)
66         std::cout << "#pos a gaps Shanon Relative Variation SumOfPairs JensenShannon\n"
67 ↪";

```

(continues on next page)

(continued from previous page)

```
61     else
62         std::cout << "#pos    gaps    Shanon Relative Variation SumOfPairs JensenShannon\n";
63
64     for (size_t ipos = 0; ipos < msa[0]->length(); ++ipos)
65         std::cout << utils::string_format("%4d %c %7.3f %7.3f %7.3f %7.3f %7.3f %7.3f\n",
66                                         ipos, seq_str[ipos],
67                                         consrv.evaluate(core::alignment::ColumnConservationScores::GapPercent, ipos),
68                                         consrv.evaluate(core::alignment::ColumnConservationScores::ShannonEntropy,
69                                         ipos),
70                                         consrv.evaluate(core::alignment::ColumnConservationScores::RelativeEntropy,
71                                         ipos),
72                                         consrv.evaluate(core::alignment::ColumnConservationScores::Variation, ipos),
73                                         consrv.evaluate(core::alignment::ColumnConservationScores::SumOfPairs, ipos),
74                                         consrv.
75                                         evaluate(core::alignment::ColumnConservationScores::JensenShannonDivergence, ipos));
76 }
```



ap_NWAligner

Calculates global sequence alignments (Needleman–Wunsch algorithm) between sequences read from a FASTA file. For every query - subject pair of sequences prints the alignment in the Edinburgh format. The default substitution-matrix is BLOSUM62

USAGE:

```
ap_NWAligner query.fasta database.fasta [substitution-matrix]
```

EXAMPLE:

```
ap_NWAligner 5fd1.fasta ferrodoxins.fasta
```

REFERENCE: Needleman, Saul B., and Christian D. Wunsch. “A general method applicable to the search for similarities in the amino acid sequence of two proteins.” JMB 48.3 (1970): 443-453. [https://doi.org/10.1016/0022-2836\(70\)90057-4](https://doi.org/10.1016/0022-2836(70)90057-4)

Keywords:

- *FASTA input*
- *Needleman-Wunsch*
- *sequence alignment*

Categories:

- core/alignment/NWAligner

Input files:

- ferrodoxins.fasta
- input2.fasta

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <chrono>
3 #include <algorithm>
4
5 #include <core/data/io/fasta_io.hh>
6
7 #include <core/alignment/NWAligner.hh>
8 #include <core/alignment/scoring/SimilarityMatrix.hh>
9 #include <core/alignment/scoring/SimilarityMatrixScore.hh>
10 #include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>
11 #include <core/alignment/on_alignment_computations.hh>
```

(continues on next page)

(continued from previous page)

```

12 #include <core/alignment/PairwiseSequenceAlignment.hh>
13 #include <core/data/io/alignment_io.hh>
14 #include <core/data/sequence/Sequence.hh>
15
16 #include <utils/exit.hh>
17
18 std::string program_info = R"(
19
20 Calculates global sequence alignments (Needleman-Wunsch algorithm) between sequences_
21 ↪read from
22 a FASTA file. For every query - subject pair of sequences prints the alignment in the_
23 ↪Edinburgh
24 format. The default substitution-matrix is BLOSUM62
25
26
27
28 USAGE:
29 ap_NWAligner query.fasta database.fasta [substitution-matrix]
30
31
32 EXAMPLE:
33 ap_NWAligner 5fd1.fasta ferrodoxins.fasta
34
35
36 REFERENCE:
37 Needleman, Saul B., and Christian D. Wunsch.
38 "A general method applicable to the search for similarities in the amino acid_
39 ↪sequence of two proteins."
40 JMB 48.3 (1970): 443-453. https://doi.org/10.1016/0022-2836\(70\)90057-4
41
42 )";
43
44 /**
45 * @brief Calculate all pairwise sequence alignments between sequences read from two_
46 ↪FASTA files : query and database
47 *
48 * CATEGORIES: core/alignment/NWAligner
49 * KEYWORDS: FASTA input; Needleman-Wunsch; sequence alignment
50 * GROUP: Alignments
51 */
52 int main(const int argc, const char* argv[]) {
53
54     if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about_
55 ↪missing program parameter
56
57     using namespace core::data::io;
58     using namespace core::data::sequence;
59     using namespace core::alignment::scoring;
60
61     // --- the query sequence
62     std::vector<std::shared_ptr<Sequence>> query_sequences;
63     read_fasta_file(argv[1], query_sequences);
64     // --- container for the sequence database
65     std::vector<std::shared_ptr<Sequence>> db_sequences;
66     read_fasta_file(argv[2], db_sequences);
67
68     // --- find longest sequence to initialize aligner object large enough
69     unsigned max_len = 0;
70     std::for_each(query_sequences.begin(), query_sequences.end(),
71         [&max_len](const Sequence_SP s) { max_len = std::max(max_len, unsigned(s->
72             length())); });
73
74 }
```

(continues on next page)

(continued from previous page)

```

63     std::for_each(db_sequences.begin(), db_sequences.end(),
64     [&max_len](const Sequence_SP s) { max_len = std::max(max_len, unsigned(s->
65     length())); });
66
66 // --- create aligner object
67 core::alignment::NWAligner<short, SimilarityMatrixScore<short>> aligner(max_len);
68 // --- read similarity matrix from a file (e.g. BLOSUM62)
69 NcbiSimilarityMatrix_SP sim_m = NcbiSimilarityMatrixFactory::get().get_matrix((argc_
70 > 3) ? argv[3] : "BLOSUM62");
71 // --- go through all db sequences and align them with the given query
72 auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
73 for (size_t i = 0; i < query_sequences.size(); ++i) {
74     for (size_t j = 0; j < db_sequences.size(); ++j) {
75         // --- Here we create a sequence similarity object that will score a match
76         // --- between individual positions from the two sequences being aligned
77         SimilarityMatrixScore<short> score(query_sequences[i]->sequence, db_
78         >sequences[j]->sequence, *sim_m);
79
80         // ----- calculate local alignment
81         aligner.align(-14, -2, score);
82
83         // ----- Convert the abstract alignment to a pairwise sequence alignment_
84         >object
85         const core::alignment::PairwiseAlignment_SP ali = aligner.backtrace();
86         core::alignment::PairwiseSequenceAlignment seq_ali(ali, query_sequences[i], db_
87         >sequences[j]);
88
88         // ----- check basics statistics of the alignment
89         core::index2 identical = core::alignment::sum_identical(seq_ali);
90         core::index2 n_aligned = seq_ali.alignment->n_aligned();
91         std::cout << utils::string_format("# %s %s id: %6.3f cov: %6.3f\n",
92             utils::split(query_sequences[i]->header())[0].c_str(), utils::split(db_
93             >sequences[j]->header())[0].c_str(),
94             identical / double(query_sequences[i]->length()), n_aligned / double(query_
95             >sequences[i]->length()) );
96         // ----- Print the alignment in Edinburgh format
97         core::data::io::write_edinburgh(seq_ali, std::cout, 80);
98
99         // --- Alternatively one can find only the score of the alignment;
100        // --- just the score - this is faster than aligning and keeping backtracking_
101        >info
102        short result = aligner.align_for_score(-10, -1, score);
103    }
104 }
```



ap_OnlineStatistics

ap_OnlineStatistics reads a file with real values and calculates simple statistics: min, mean, stdev, max. The program uses method of Knuth and Welford for computing average and standard deviation in one pass through the data If no input file is provided, the program calculates the statistics from a random sample.

USAGE:

```
ap_OnlineStatistics infile
```

EXAMPLE:

```
ap_WeightedOnlineStatistics random_normal.txt
```

REFERENCE: https://www.johndcook.com/blog/skewness_kurtosis/ https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance#Welford%27s_online_algorithm

Keywords:

- *random numbers*
- *statistics*

Categories:

- core::calc::statistics::OnlineStatistics; core::calc::statistics::Random

Input files:

- random_normal.txt

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/index.hh>
4 #include <core/calc/statistics/Random.hh>
5 #include <core/calc/statistics/OnlineStatistics.hh>
6
7 std::string program_info = R"(
8
9 ap_OnlineStatistics reads a file with real values and calculates simple statistics:_
10 ↵min, mean, stdev, max.
11 The program uses method of Knuth and Welford for computing average and standard_
12 ↵deviation in one pass through the data
13 If no input file is provided, the program calculates the statistics from a random_
14 ↵sample.
15 )"
```

(continues on next page)

(continued from previous page)

```

13 USAGER:
14     ap_OnlineStatistics infile
15
16 EXAMPLE:
17     ap_WeightedOnlineStatistics random_normal.txt
18
19 REFERENCE:
20     https://www.johndcook.com/blog/skewness_kurtosis/
21     https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance#Welford%27s_
22     ↪online_algorithm
23 ) ";
24
25 /** @brief Reads a file with real values and calculates simple statistics: min, mean, ↪
26     ↪stdev, max.
27 * If no input file is provided, the program calculates the statistics from a random ↪
28     ↪sample
29 *
30 * CATEGORIES: core::calc::statistics::OnlineStatistics; ↪
31     ↪core::calc::statistics::Random
32 * KEYWORDS: random numbers; statistics
33 * GROUP: Statistics;
34 */
35
36 int main(const int argc, const char *argv[]) {
37
38     core::calc::statistics::OnlineStatistics stats;
39     if(argc < 2) {
40         // --- complain about missing program parameter
41         std::cerr << program_info;
42         // ----- Use the random engine if no data is provided
43         core::calc::statistics::Random r = core::calc::statistics::Random::get();
44         r.seed(12345); // --- seed the generator for repeatable results
45         core::calc::statistics::UniformRealRandomDistribution<double> uniform_random;
46         for(core::index4 n = 0; n < 100000; ++n) stats(uniform_random(r));
47     } else {
48         std::ifstream in(argv[1]);
49         double r;
50         while(in) {
51             in >> r;
52             stats(r);
53         }
54     }
55
56     std::cout << "#cnt      min          avg          sdev      skewness      kurtosis      max      ↪
57     ↪bimodalitycoefficient\n";
58     std::cout << utils::string_format("%d %f %f %f %f %f %f %f\n", stats.cnt(), stats.
59     ↪min(), stats.avg(),
60     sqrt(stats.var()), stats.skewness(), stats.kurtosis(), stats.max(), stats.bimodality_
61     ↪coefficient());
62 }
```



ap_PairwiseCrmsd

ap_PairwiseCrmsd calculates crmsd value between every pair of protein structures given at the input (at least two structures must be provided). Only values smaller than 20 Angstroms are printed. This example evaluates crmsd for each pair of proteins twice: on C-alpha atoms and on all backbone atoms

USAGE:

```
ap_PairwiseCrmsd structureA.pdb structureB.pdb [structureC.pdb ... ]
```

EXAMPLE:

```
ap_PairwiseCrmsd 2gb1.pdb 2gb1-model11.pdb 2gb1-model12.pdb
```

Keywords:

- *PDB input*
- *crmsd*
- *structure selectors*

Categories:

- core::protocols::PairwiseCrmsd

Input files:

- 2gb1-model3.pdb
- 2gb1-model11.pdb
- 2gb1-model4.pdb
- 2gb1.pdb
- 2gb1-model2.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/protocols/PairwiseCrmsd.hh>
4 #include <core/data/io/Pdb.hh>
5 #include <core/data/structural/selectors/structure_selectors.hh>
6
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10

```

(continues on next page)

(continued from previous page)

```

11 ap_PairwiseCrmsd calculates crmsd value between every pair of protein structures,
12   ↪given at the input
13 (at least two structures must be provided). Only values smaller than 20 Angstroms are,
14   ↪printed.
15
16 This example evaluates crmsd for each pair of proteins twice: on C-alpha atoms and on,
17   ↪all backbone atoms
18
19 USAGE:
20   ap_PairwiseCrmsd structureA.pdb structureB.pdb [structureC.pdb ... ]
21
22 EXAMPLE:
23   ap_PairwiseCrmsd 2gb1.pdb 2gb1-model1.pdb 2gb1-model2.pdb
24
25 ) ";
26
27 /**
28 * @brief Calculates crmsd value for a set of protein structures (at least two)
29 *
30 * CATEGORIES: core::protocols::PairwiseCrmsd
31 * KEYWORDS: PDB input; crmsd; structure selectors
32 * GROUP: Structure calculations;
33 */
34
35 int main(const int argc, const char *argv[]) {
36
37     if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about,
38   ↪missing program parameter
39
40     using core::data::basic::Vec3;
41     using namespace core::data::structural::selectors; // --- for all AtomSelector types
42     using namespace core::data::io;
43     using namespace core::protocols;
44
45     std::vector<core::data::structural::Structure_SP> structures;
46     std::vector<std::string> tags;
47     for (int i = 1; i < argc; ++i) {
48         core::data::io::Pdb reader(argv[i], all_true(is_not_alternative, is_not_water),
49           ↪keep_all, false); // --- note we read all atoms but skip alternate locators and
50           ↪waters
51         structures.push_back(reader.create_structure(0));
52         tags.push_back(structures.back()>code());
53     }
54
55     // ----- crmsd on C-alpha : this is the
56     std::cout << "# crmsd on alpha carbons:\n";
57     std::shared_ptr<AtomSelector> is_CA = std::make_shared<IsCA>();
58     PairwiseCrmsd rmsd_ca(structures, is_CA, tags);
59     rmsd_ca.crmsd_cutoff(20.0); // crmsd cutoff large enough to get some output
60     rmsd_ca.calculate();
61
62     // ----- crmsd on backbone
63     std::cout << "# crmsd on heavy backbone atoms:\n";
64     std::shared_ptr<AtomSelector> is_bb = std::make_shared<IsBB>();
65     std::shared_ptr<AtomSelector> not_h = std::make_shared<NotHydrogen>();
66     std::shared_ptr<LogicalANDSelector> heavy_bb = std::make_shared<LogicalANDSelector>
67       ();
68     heavy_bb->add_selector(is_bb);
69     heavy_bb->add_selector(not_h);

```

(continues on next page)

(continued from previous page)

```
61 PairwiseCrmsd rmsd_bb(structures, heavy_bb, tags);
62 rmsd_bb.crmsd_cutoff(20.0); // crmsd cutoff large enough to get some output
63 rmsd_bb.calculate();
64 }
```



ap_PairwiseSequenceIdentityProtocol

Evaluates pairwise sequence identity between sequences found in a given FASTA file. The calculations may be performed for a single sequence (against all the other sequences) or for a range of sequences. Calculations may be executed in several parallel threads, calculated values are printed on the screen if they are greater than given cutoff. In addition, the query sequence or sequence range may be provided as fourth, or fourth and fifth parameters, respectively. By default, the program runs on 4 threads, with cutoff 0.28, i.e. printing only these pairs where sequence identity is higher than 28%

USAGE:

```
./ap_PairwiseSequenceIdentityProtocol in.fasta [n_threads [cutoff] ]
./ap_PairwiseSequenceIdentityProtocol in.fasta n_threads cutoff query-sequence-index
./ap_PairwiseSequenceIdentityProtocol in.fasta n_threads cutoff first-sequence-index_
↪last-sequence-index
```

EXAMPLES:

```
./ap_PairwiseSequenceIdentityProtocol small50_95identical.fasta 4 0.28
./ap_PairwiseSequenceIdentityProtocol small50_95identical.fasta 4 0.28 0
./ap_PairwiseSequenceIdentityProtocol small50_95identical.fasta 4 0.28 0 5
```

First example calculates identity for every pair of sequences. Next one between the first sequence (index 0) all others sequences. Finally the third uses sequences from 0 to 5 (both inclusive) as queries against all the other sequences.

Keywords:

- *FASTA input*
- *sequence alignment*

Categories:

- core/protocols/PairwiseSequenceIdentityProtocol.hh

Input files:

- small50_95identical.fasta
- small500_95identical.fasta

Output files:

- stdout.out

Program source:

```
1 #include <core/index.hh>
2 #include <utils/exit.hh>
3 #include <utils/Logger.hh>
4 #include <core/data/io/fasta_io.hh>
5 #include <core/protocols/PairwiseSequenceIdentityProtocol.hh>
```

(continues on next page)

(continued from previous page)

```

6 std::string program_info = R"(

7 Evaluates pairwise sequence identity between sequences found in a given FASTA file. ↴
8 ↪The calculations may be performed
9 for a single sequence (against all the other sequences) or for a range of sequences.
10
11 Calculations may be executed in several parallel threads, calculated values are ↴
12 ↪printed on the screen if they
13 are greater than given cutoff. In addition, the query sequence or sequence range may ↴
14 ↪be provided as fourth,
15 or fourth and fifth parameters, respectively.

16 By default, the program runs on 4 threads, with cutoff 0.28, i.e. printing only these ↴
17 ↪pairs where sequence identity
18 is higher than 28%


19 USAGE:
20 ./ap_PairwiseSequenceIdentityProtocol in.fasta [n_threads [cutoff] ]
21 ./ap_PairwiseSequenceIdentityProtocol in.fasta n_threads cutoff query-sequence-index
22 ./ap_PairwiseSequenceIdentityProtocol in.fasta n_threads cutoff first-sequence-index ↴
23 ↪last-sequence-index


24 EXAMPLES:
25 ./ap_PairwiseSequenceIdentityProtocol small150_95identical.fasta 4 0.28
26 ./ap_PairwiseSequenceIdentityProtocol small150_95identical.fasta 4 0.28 0
27 ./ap_PairwiseSequenceIdentityProtocol small150_95identical.fasta 4 0.28 0 5


28 First example calculates identity for every pair of sequences. Next one between the ↴
29 ↪first sequence (index 0)
30 all others sequences. Finally the third uses sequences from 0 to 5 (both inclusive) ↴
31 ↪as queries against
32 all the other sequences.


33 )

34 ) ";


35 /**
36  * @brief Uses PairwiseSequenceIdentityProtocol protocol to calculate all pairwise ↴
37  * sequence identity values for a set of sequences
38  *
39  * @CATEGORIES: core/protocols/PairwiseSequenceIdentityProtocol.hh
40  * @KEYWORDS: FASTA input; sequence alignment
41  * @GROUP: Alignments
42  */
43 int main(const int argc, const char* argv[]) {

44     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about ↴
45     ↪missing program parameter


46     using namespace core::data::sequence;
47     using namespace core::protocols;
48     using namespace core::alignment;

49


50     utils::Logger logs("ap_PairwiseSequenceIdentityProtocol");
51     int my_argc = argc;
52     bool if_use_fasta = false;
53 //    if (stristr(argv[my_argc-1], "fasta") !=NULL) {

```

(continues on next page)

(continued from previous page)

```

54 //      if_use_fasta = true;
55 //      --my_argc;
56 }
57 core::index2 n_threads = (my_argc > 2) ? atoi(argv[2]) : 4;
58 float cutoff = (my_argc > 3) ? atof(argv[3]) : 0.25;
59
60 logs << utils::LogLevel::INFO << "number of threads used : " << n_threads << "\n";
61 logs << utils::LogLevel::INFO << "seq. similarity cutoff : " << cutoff << "\n";
62
63 core::protocols::PairwiseSequenceIdentityProtocol protocol;
64 protocol.printed_seqname_length(20).gap_open(-10).gap_extend(-1).substitution_
65   ↪matrix("BLOSUM62").
66   keep_alignments(false).alignment_method(AlignmentType::SEMIGLOBAL_ALIGNMENT).n_
67   ↪threads(n_threads);
68 protocol.if_use_fasta_filter(if_use_fasta).seq_identity_cutoff(cutoff).batch_
69   ↪size(10000);
70 protocol.printed_seqname_length(10);
71
72 if (my_argc == 5) {
73     protocol.select_query(atoi(argv[4]));
74     logs << utils::LogLevel::INFO << "Using sequence at index " << atoi(argv[4]) << "_
75   ↪as a query\n";
76 }
77 if (my_argc == 6) {
78     for (core::index4 i = atoi(argv[4]); i <= atoi(argv[5]); ++i) protocol.add_
79   ↪query(i);
80     logs << utils::LogLevel::INFO << "Using " << atoi(argv[5]) - atoi(argv[4]) << "_
81   ↪query sequences\n";
82 }
83
84 std::vector<Sequence_SP> input_sequences;
85 core::data::io::read_fasta_file(argv[1], input_sequences);
86 for(Sequence_SP si: input_sequences) protocol.add_input_sequence(si);
87
88 auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
89 protocol.run();
90 auto end = std::chrono::high_resolution_clock::now();
91 std::chrono::duration<double> time_span = std::chrono::duration_cast
92   ↪<std::chrono::duration<double>>(end - start);
93 logs << utils::LogLevel::INFO << (size_t) protocol.n_jobs_completed()
94   << " global alignment sequence identities calculated within " << time_
95   ↪span.count() << " [s]\n";
96
97 protocol.print_header(std::cout);
98 protocol.print_sequence_identity(std::cout);
99 }
```



ap_ProteinArchitecture

ap_ProteinArchitecture reads a PDB file and describes its architecture in terms of secondary structure elements (SSEs) and their connectivity (i.e. how strands are connected in sheets). The SSEs themselves are defined based on data from PDB file header. If DSSP flag has been given, the app will detect secondary structure elements using BioShell's implementation of DSSP algorithm.

USAGE:

```
ap_ProteinArchitecture input.pdb [DSSP]
```

EXAMPLE:

```
ap_ProteinArchitecture 5edw.pdb [DSSP]
```

REFERENCE: Kabsch, Wolfgang, and Christian Sander. "Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features." *Biopolymers* 22 (1983): 2577-2637. doi:10.1002/bip.360221211

Keywords:

- *PDB input*
- *Hydrogen bonds*
- *Protein structure features*

Categories:

- core/calc/structural/ProteinArchitecture

Input files:

- 2gb1.pdb
- 5edw.pdb
- 2fdo.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <core/calc/structural/ProteinArchitecture.hh>
5 #include <utils/exit.hh>
6 #include <utils/LogManager.hh>
7
8 using namespace core::data::structural;
9 using namespace core::data::io;
```

(continues on next page)

(continued from previous page)

```

10  using namespace core::data::basic;
11
12 std::string program_info = R"
13
14 ap_ProteinArchitecture reads a PDB file and describes its architecture in terms of ↵
15 ↵secondary structure elements (SSEs) and their connectivity (i.e. how strands are connected in sheets). The SSEs ↵
16 ↵themselves are defined based on data from PDB file header. If DSSP flag has been given, the app will detect secondary ↵
17 ↵structure elements using BioShell's implementation of DSSP algorithm.
18
19 USAGE:
20     ap_ProteinArchitecture input.pdb [DSSP]
21
22 EXAMPLE:
23     ap_ProteinArchitecture 5edw.pdb [DSSP]
24
25 REFERENCE:
26 Kabsch, Wolfgang, and Christian Sander.
27 "Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded ↵
28 ↵and geometrical features."
29 Biopolymers 22 (1983): 2577-2637. doi:10.1002/bip.360221211
30 )
31
32 /**
33 * @brief Calculates a map of backbone hydrogen bonds.
34 * @CATEGORIES: core/calc/structural/ProteinArchitecture;
35 * @KEYWORDS: PDB input; Hydrogen bonds; Protein structure features
36 * @GROUP: Structure calculations;
37 */
38 int main(const int argc, const char* argv[]) {
39
40     using namespace core::calc::structural;
41
42     utils::LogManager::INFO(); // --- Turn it to FINE to see a lot more of messages, e.g. about missed h-bonds
43
44     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
45     core::data::io::Pdb reader(argv[1], core::data::io::all_true(is_not_alternative, is_not_water),
46         core::data::io::keep_all, true); // --- Read in a PDB file
47     Structure_SP strctr = reader.create_structure(0); // --- create a Structure object from the first model
48
49     bool if_dssp = (argc > 2) && (strcmp(argv[2], "DSSP") == 0);
50     core::calc::structural::ProteinArchitecture pa(*strctr, if_dssp);
51
52     std::cout << "# ----- Secondary structure elements -----" << std::endl;
53     for (const auto sse : pa.sse_vector())
54         std::cout << sse << "\n";
55
56     std::cout << "# ----- Beta strand connectivity -----" << std::endl;
57     auto sse_graph = pa.create_strand_graph();
58     sse_graph->print_adjacency_matrix(std::cerr);

```

(continues on next page)

(continued from previous page)

```
59 for(auto e_it = sse_graph->cbegin_strand(); e_it!=sse_graph->cend_strand();++e_it) {  
60     std::cout << (*e_it)->info()<<" paired with:\n";  
61     for(auto partner_it = sse_graph->cbegin_strand(*e_it); partner_it != sse_graph->  
62         cend_strand(*e_it); ++partner_it) {  
63         auto pairing_sp = sse_graph->get_strand_pairing(*e_it, *partner_it);  
64         std::cout << "\t" << (*partner_it)->name() << " " << Strand::strand_type_  
65         name(pairing_sp->pairing_type)  
66             << " by " << pairing_sp->hydrogen_bonds().size() << " hbonds\n";  
67     }  
68 }  
69 }
```



ap_Rubik_simulation

The program runs a Replica Exchange Monte Carlo simulation of a Rubik's cube system

Keywords:

- *Monte Carlo*
- *sampling*
- *observer*
- *simulation*

Categories:

- simulations/sampling/ReplicaExchangeMC

Program source:

```

1 #include <iostream>
2 #include <string>
3 #include <stdexcept>
4 #include <stdlib.h>
5 #include <fstream>
6 #include <vector>
7
8 #include <simulations/evaluators/CallEvaluator.hh>
9 #include <simulations/forcefields/CalculateEnergyBase.hh>
10 #include <simulations/evaluators/Evaluator.hh>
11 #include <simulations/forcefields/TotalEnergy_OBSOLETE.hh>
12
13 #include <simulations/observers/ObserveEvaluators.hh>
14 #include <simulations/observers/ObserveMoversAcceptance.hh>
15 #include <simulations/observers/TriggerEveryN.hh>
16 #include <simulations/observers/ObserveReplicaFlow.hh>
17 #include <simulations/observers/ObserveWLSampling.hh>
18
19 #include <simulations/sampling/IsothermalMC.hh>
20 #include <simulations/sampling/ReplicaExchangeMC.hh>
21 #include <simulations/systems/ising/RubikCube.hh>
22
23 #include <utils/options/sampling_options.hh>
24 #include <utils/options/sampling_from_cmdline.hh>
25 #include <simulations/sampling/WangLandauSampler.hh>
26
27
28 using namespace simulations;
29 using namespace simulations::systems::ising;
30
31 utils::Logger logs("ap_Rubik_simulation");
32
33 std::string program_info = R"(
34
35 The program runs a Replica Exchange Monte Carlo simulation of a Rubik's cube system

```

(continues on next page)

(continued from previous page)

```

36
37 ) ";
38
39 /** @brief Turns energy of a system into an energy bin index (integer)
40 * @param energy - system's energy
41 * @return integer assigned to a bin; may be negative
42 */
43 inline int bfe(double energy) { return (int) energy; }
44
45 std::shared_ptr<simulations::sampling::WangLandauSampler> prepare_wl_simulation(const_
46 → simulations::SimulationSettings &settings) {
47
48     using namespace utils::options;
49     int system_size = settings.get<int>("cube_size");
50     Rubik_SP system = std::make_shared<Rubik>(system_size);
51     logs << "Minimum energy: " << system->calculate() << "\n";
52     system->scramble(); // Set the cube to a random conformation
53     logs << "starting energy: " << system->calculate() << "\n";
54
55     // ----- Movers definition -----
56     simulations::movers::MoversSet_SP movers = std::make_shared_
57 → <simulations::movers::MoversSetSweep>();
58     movers->add_mover(std::static_pointer_cast<simulations::movers::Mover>(system),_
59 → system_size * system_size);
60
61     // ----- Create the sampler -----
62     std::shared_ptr<simulations::sampling::WangLandauSampler> sampler = std::make_shared_
63 → <simulations::sampling::WangLandauSampler>(
64         movers, system->calculate(), bfe, system_size * system_size * 6);
65     sampler->reset(settings);
66
67     simulations::forcefields::CalculateEnergyBase_SP energies;
68
69     simulations::observers::ObserveEvaluators_SP observations
70         = std::make_shared<simulations::observers::ObserveEvaluators>(utils::string_
71 → format("energy-wl.dat"));
72     observations->add_evaluator(system);
73     sampler->outer_cycle_observer(observations);
74     sampler->outer_cycle_observer(std::make_shared_
75 → <simulations::observers::ObserveWLSampling>(*sampler, "wl.dat"));
76
77     simulations::observers::ObserveMoversAcceptance_SP obs_ms
78         = std::make_shared<simulations::observers::ObserveMoversAcceptance>(*movers,
79
80 → utils::string_format("movers-wl.dat"));
81     obs_ms->observe_header();
82     sampler->outer_cycle_observer(obs_ms);
83
84     return sampler;
85 }
86
87 std::shared_ptr<simulations::sampling::ReplicaExchangeMC> prepare_replica_
88 → simulation(const simulations::SimulationSettings& settings) {
89
90     using namespace utils::options;
91     std::vector<Rubik_SP> systems;
92     std::vector<simulations::sampling::IsothermalMC_SP> replica_samplers;

```

(continues on next page)

(continued from previous page)

```

85     std::vector<simulations::forcefields::CalculateEnergyBase_SP> energies;
86     std::vector<double> temperatures;
87     utils::split(settings.get<std::string>(replicas), temperatures, ',');
88     core::index4 n_outer_cycles = settings.get<core::index4>(mc_outer_cycles);
89     core::index4 n_inner_cycles = settings.get<core::index4>(mc_inner_cycles);
90     core::index4 n_exchanges = settings.get<core::index4>(replica_exchanges);
91
92     int system_size = settings.get<int>("cube_size");
93     for (core::index2 irepl = 0; irepl < temperatures.size(); ++irepl) {
94
95         // ----- Create the systems to be sampled -----
96         Rubik_SP system = std::make_shared<Rubik>(system_size);
97         system->scramble();      // Set the cube to a random conformation
98         systems.push_back(system);
99         energies.push_back(system);
100
101        // ----- Movers definition -----
102        simulations::movers::MoversSet_SP movers = std::make_shared
103        <simulations::movers::MoversSetSweep>();
104        movers->add_mover(std::static_pointer_cast<simulations::movers::Mover>(system),
105        system_size * system_size);
106
107        // ----- Create the sampler -----
108        auto sampler = std::make_shared<simulations::sampling::IsothermalMC>(movers,
109        temperatures[irepl]);
110        replica_samplers.push_back(sampler);
111        sampler->cycles(n_inner_cycles, n_outer_cycles);
112
113        simulations::observers::ObserveEvaluators_SP observations
114        = std::make_shared<simulations::observers::ObserveEvaluators>(utils::string_
115        &format("energy-%.3f.dat", temperatures[irepl]));
116        observations->add_evaluator(system);
117        sampler->outer_cycle_observer(observations);
118        simulations::observers::ObserveMoversAcceptance_SP obs_ms
119        = std::make_shared<simulations::observers::ObserveMoversAcceptance>(*movers,
120        &utils::string_format("movers-%.3f.dat", temperatures[irepl]));
121        obs_ms->observe_header();
122        sampler->outer_cycle_observer(obs_ms);
123    }
124
125    auto remc = std::make_shared<simulations::sampling::ReplicaExchangeMC>(replica_
126    &samplers, energies, true);
127    auto remc_flow = std::make_shared<simulations::observers::ObserveReplicaFlow>(*remc,
128    &"replica_flow.dat");
129    remc->exchange_observer(remc_flow);
130    remc->replica_exchanges(n_exchanges);
131
132    return remc;
133}
134
135/** @brief The program runs a Replica Exchange Monte Carlo simulation of a Rubik's
136cube system.
137 */
138/* This example shows how to simulate a system using BioShell library
139 */
140/* CATEGORIES: simulations/sampling/ReplicaExchangeMC;
141 * KEYWORDS: Monte Carlo; sampling; observer; simulation

```

(continues on next page)

(continued from previous page)

```

134 * IMG_ALT: Example results from a Rubik's cube simulations
135 */
136 int main(const int argc, const char *argv[]) {
137
138     using namespace utils::options; // --- All the options are in this namespace
139     static Option cube_size("-c", "-cube_size", "size of the Rubik's cube");
140     static Option sampler("-s", "-sampler", "MC sampler: 'remc' or 'wl'");
141
142     utils::options::OptionParser &cmd = utils::options::OptionParser::get();
143     cmd.register_option(utils::options::help, verbose, rnd_seed, cube_size(3), sampler);
144     cmd.register_option(mc_outer_cycles(10000), mc_inner_cycles(10), mc_cycle_factor(1),
145     ↪ replica_exchanges(10));
146     cmd.register_option(begin_temperature(2.0), end_temperature(0.5), temp_steps(0.1),
147                         replicas("2,1.75,1.5,1.25,1.0,0.8,0.7,0.6,0.5"));
148
149     if (!cmd.parse_cmdline(argc, argv)) return 1;
150
151     if (rnd_seed.was_used()) {
152         auto rnd = option_value<core::calc::statistics::Random::result_type>(rnd_seed);
153         core::calc::statistics::Random::seed(rnd);
154         logs << utils::LogLevel::SEVERE << "Pseudorandom start: " << rnd << "\n";
155     } else {
156         core::calc::statistics::Random::get().seed(12345); // --- seed the generator for
157     ↪ repeatable results
158         logs << utils::LogLevel::SEVERE << "Pseudorandom start with seed: 12345\n";
159     //     core::calc::statistics::Random::seed(time(0));
160     //     logs << utils::LogLevel::SEVERE << "Pseudorandom start with time(0) seed: \n";
161     }
162
163     simulations::SimulationSettings settings;
164     settings.insert_or_assign(cmd, true);
165
166     if ((option_value<std::string>(sampler) == "wl") || (option_value<std::string>
167     ↪ (sampler) == "WL")) {
168         auto wl = prepare_wl_simulation(settings);
169         wl->run();
170     } else {
171         auto remc = prepare_replica_simulation(settings);
172         remc->run();
173     }
174 }
```



ap_SWAligner

Calculates local sequence alignments (Smith-Waterman algorithm) between sequences read from a FASTA file. For every query - subject pair of sequences prints the alignment in the Edinburgh format. The default substitution-matrix is BLOSUM62.

USAGE:

```
ap_SWAligner query.fasta database.fasta [substitution-matrix]
```

EXAMPLE:

```
ap_SWAligner 5fd1.fasta test_inputs/ferrodoxins.fasta
```

REFERENCE: Smith, Temple F., and Michael S. Waterman. "Identification of common molecular subsequences." JMB 147.1 (1981): 195-197. doi:10.1016/0022-2836(81)90087-5

Keywords:

- *FASTA input*
- *sequence alignment*

Categories:

- core/alignment/SWAligner

Input files:

- ferrodoxins.fasta
- input2.fasta

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <chrono>
3 #include <algorithm>
4
5 #include <core/data/io/fasta_io.hh>
6
7 #include <core/data/sequence/Sequence.hh>
8 #include <core/alignment/SWAligner.hh>
9 #include <core/alignment/scoring/SimilarityMatrix.hh>
10 #include <core/alignment/scoring/SimilarityMatrixScore.hh>
11 #include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>
12 #include <core/alignment/PairwiseSequenceAlignment.hh>
13 #include <core/data/io/alignment_io.hh>
```

(continues on next page)

(continued from previous page)

```

14
15 #include <utils/exit.hh>
16
17 std::string program_info = R"(
18
19 Calculates local sequence alignments (Smith-Waterman algorithm) between sequences_
20 ↪read from
21 a FASTA file. For every query - subject pair of sequences prints the alignment in the_
22 ↪Edinburgh
23 format. The default substitution-matrix is BLOSUM62.
24
25
26 USAGE:
27 ap_SWAligner query.fasta database.fasta [substitution-matrix]
28
29 EXAMPLE:
30 ap_SWAligner 5fd1.fasta test_inputs/ferrodoxins.fasta
31
32 REFERENCE:
33 Smith, Temple F., and Michael S. Waterman. "Identification of common molecular_
34 ↪subsequences."
35 JMB 147.1 (1981): 195-197. doi:10.1016/0022-2836(81)90087-5
36
37 )";
38
39 /**
40 * @brief Calculate all pairwise sequence alignments between sequences read from two_
41 ↪FASTA files : query and database
42 */
43 * CATEGORIES: core/alignment/SWAligner
44 * KEYWORDS: FASTA input; sequence alignment
45 * GROUP: Alignments
46 */
47 int main(const int argc, const char* argv[]) {
48
49     if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about_
50 ↪missing program parameter
51
52     using namespace core::data::io;
53     using namespace core::data::sequence;
54     using namespace core::alignment::scoring;
55
56     // --- the query sequence
57     std::vector<std::shared_ptr<Sequence>> query_sequences;
58     read_fasta_file(argv[1], query_sequences);
59     // --- container for the sequence database
60     std::vector<std::shared_ptr<Sequence>> db_sequences;
61     read_fasta_file(argv[2], db_sequences);
62
63     // --- find longest sequence to initialize aligner object large enough
64     unsigned max_len = 0;
65     std::for_each(query_sequences.begin(), query_sequences.end(),
66         [&max_len](const Sequence_SP s) { max_len = std::max(max_len, unsigned(s->
67             length())); });
68     std::for_each(db_sequences.begin(), db_sequences.end(),
69         [&max_len](const Sequence_SP s) { max_len = std::max(max_len, unsigned(s->
70             length())); });
71
72     // ----- Create aligner object

```

(continues on next page)

(continued from previous page)

```

64 core::alignment::SWAligner<short, SimilarityMatrixScore<short>> aligner(max_len);
65
66 // ----- read similarity matrix from a file (e.g. BLOSUM62)
67 NcbiSimilarityMatrix_SP sim_m = NcbiSimilarityMatrixFactory::get().get_matrix((argc_
68 ↪ > 3) ? argv[3] : "BLOSUM62");
69
70 // ----- Go through all db sequences and align them with the given query
71 auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
72 for (size_t i = 0; i < query_sequences.size(); ++i) {
73     for (size_t j = 0; j < db_sequences.size(); ++j) {
74
75         // ----- Here we create a sequence similarity object that will score a_
76         ↪ match
77         // ----- between individual positions from the two sequences being aligned
78         SimilarityMatrixScore<short> score(query_sequences[i]->sequence, db_
79         ↪ sequences[j]->sequence, *sim_m);
80
81         // ----- calculate local alignment
82         aligner.align(-10, -1, score);
83
84         // ----- Convert the abstract alignment to a pairwise sequence alignment_
85         ↪ object
86         const core::alignment::PairwiseAlignment_SP ali = aligner.backtrace();
87         core::alignment::PairwiseSequenceAlignment seq_ali(ali, query_sequences[i], db_
88         ↪ sequences[j]);
89
90         // ----- Print the alignment in Edinburgh format
91         core::data::io::write_edinburgh(seq_ali, std::cout, 80);
92     }
93 }
```



ap_SequenceProfile

ap_SequenceProfile reads a Multiple Sequence Alignment (MSA) in ClustalO or FASTA format and prints a sequence profile made from it. The program detects the format of an input file by its extension: use either .fasta or .aln, for FASTA and ClustalO, respectively. If the optional argument -w is used, sequences will be weighted before profile calculations. The profile probabilities will be therefore weighted counts rather than just raw observations.

USAGE:

```
./ap_SequenceProfile infile.aln [-w]
```

EXAMPLE:

```
./ap_SequenceProfile cyped.CYP109.aln  
./ap_SequenceProfile cyped.CYP109.fasta -w
```

Keywords:

- *sequence profile*
- *clustal input*
- *MSA*

Categories:

- core/data/sequence/SequenceProfile; core/protocols/SequenceWeightingProtocol

Input files:

- 1crn.hssp
- cyped.CYP109.aln

Output files:

- cyped.CYP109.profile
- 1crn.profile
- cyped.CYP109.wght.profile
- stdout.out

Program source:

```
1 #include <iostream>  
2  
3 #include <core/data/io/hssp_io.hh>  
4 #include <core/data/io/fasta_io.hh>  
5 #include <core/data/io/clustalw_io.hh>  
6 #include <core/data/sequence/SequenceProfile.hh>  
7 #include <core/protocols/SequenceWeightingProtocol.hh>  
8 #include <utils/exit.hh>
```

(continues on next page)

(continued from previous page)

```

9 #include <utils/io_utils.hh>
10
11 std::string program_info = R"(
12
13 ap_SequenceProfile reads a Multiple Sequence Alignment (MSA) in ClustalO or FASTA_
14 ↴format
15 and prints a sequence profile made from it. The program detects the format of ain_
16 ↴input file
17 by its extension: use either .fasta or .aln, for FASTA and ClustalO, respectively.
18
19 If the optional argument -w is used, sequences will be weighted before profile_
20 ↴calculations.
21 The profile probabilities will be therefore weighted counts rather than just raw_
22 ↴observations.
23
24 USAGE:
25     ./ap_SequenceProfile infile.aln [-w]
26
27 EXAMPLE:
28     ./ap_SequenceProfile cyped.CYP109.aln
29     ./ap_SequenceProfile cyped.CYP109.fasta -w
30
31 )";
32
33 /**
34 * @brief Reads a MSA in ClustalW format and prints a sequence profile
35 *
36 * CATEGORIES: core/data/sequence/SequenceProfile; core/protocols/
37 ↴SequenceWeightingProtocol
38 * KEYWORDS: sequence profile; Clustal input; MSA
39 * GROUP: Sequence calculations;
40 */
41
42 int main(const int argc, const char* argv[]) {
43
44     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
45 ↴missing program parameter
46
47     using namespace core::data::io;
48     using namespace core::data::sequence;
49
50     // ----- Load all sequences into a vector
51     std::vector<Sequence_SP> msa; // --- Sequence_SP is just a shorter name for_
52 ↴std::shared_ptr<Sequence>
53     auto root_extn = utils::root_extension(argv[1]);
54     if ((root_extn.second == "aln") || (root_extn.second == "clustalw")) {
55         core::data::io::read_clustalw_file(argv[1], msa, true);
56     } else if (root_extn.second == "hssp") {
57         core::data::io::read_hssp_file(argv[1], msa, true, true);
58     } else
59         core::data::io::read_fasta_file(argv[1], msa);
60
61     std::vector<double> seq_weights{1,1.0}; // --- just one weight of value 1.0
62     // ----- Set up and run sequence weighting protocol if needed
63     if ((argc == 3) && (strcmp(argv[2], "-w") == 0)) {
64         core::protocols::HenikoffSequenceWeights protocol;
65         protocol.n_threads(4).add_input_sequences(msa);
66         protocol.run();
67         seq_weights.clear();
68     }
69 }
```

(continues on next page)

(continued from previous page)

```
59     for (core:::index2 i = 0; i < msa.size(); ++i) seq_weights.push_back(protocol.get_
60     ↵weight(i));
61
62     // ----- Create a sequence profile and print it on the screen
63     SequenceProfile profile(*msa[0], SequenceProfile::aaOrderByPropertiesGapped(), msa,
64     ↵seq_weights);
65     profile.write_table(std::cout);
66 }
```



ap_SequenceWeightingProtocol

ap_SequenceWeightingProtocol reads a set of protein sequences and computes a real weight for each of those sequences. If the FASTA file is the input, every pair of sequences will be aligned and sequence identity values will be evaluated based on these alignments. If .aln is the input (i.e. ClustalO MSA file format), it is assumed the sequences are already aligned and sequence identity values will be computed based on the MSA. Sequence identity values will be transformed into real weights. These weights may be further used e.g. in sequence profile construction

USAGE:

```
ap_SequenceWeightingProtocol input-file
```

EXAMPLES:

```
ap_SequenceWeightingProtocol input.fasta  
ap_SequenceWeightingProtocol input.aln
```

Keywords:

- *FASTA input*
- *sequence alignment*
- sequence identity
- sequence weighting

Categories:

- core/protocols/SequenceWeightingProtocol

Input files:

- ferrodoxins.fasta
- identical.fasta
- cyped.CYP109.aln

Output files:

- stdout.out

Program source:

```
1 #include <utils/exit.hh>  
2 #include <core/data/basic/Array2D.hh>  
3 #include <core/data/sequence/Sequence.hh>  
4 #include <core/data/io/fasta_io.hh>  
5 #include <core/data/io/clustalw_io.hh>  
6 #include <core/protocols/SequenceWeightingProtocol.hh>  
7 #include <utils/io_utils.hh>
```

(continues on next page)

(continued from previous page)

```

9 std::string program_info = R"(
10
11 ap_SequenceWeightingProtocol reads a set of protein sequences and computes a real_
12 ↴weight for each of those sequences.
13
14 If the FASTA file is the input, every pair of sequences will be aligned and sequence_
15 ↴identity values will be evaluated
16 based on these alignments. If .aln is the input (i.e. ClustalO MSA file format), it_
17 ↴is assumed the sequences are already
18 aligned and sequence identity values will be computed based on the MSA.
19
20 Sequence identity values will be transformed into real weights. These weights may be_
21 ↴further used e.g.
22 in sequence profile construction
23
24 USAGE:
25     ap_SequenceWeightingProtocol input-file
26
27 EXAMPLES:
28     ap_SequenceWeightingProtocol input.fasta
29     ap_SequenceWeightingProtocol input.aln
30
31 )";
32
33 /**
34 * @brief Shows how to use SequenceWeightingProtocol class
35 *
36 * @CATEGORIES: core/protocols/SequenceWeightingProtocol
37 * @KEYWORDS: FASTA input; sequence alignment; sequence identity; sequence weighting
38 * @GROUP: Sequence calculations;
39 */
40 int main(const int argc, const char* argv[]) {
41
42     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
43     ↴missing program parameter
44
45     using namespace core::data::sequence;
46     using namespace core::protocols;
47
48     bool if_align = true;
49     std::vector<Sequence_SP> input_sequences;
50     auto root_extn = utils::root_extension(argv[1]);
51     if ((root_extn.second == "aln") || (root_extn.second == "clustalw")) {
52         core::data::io::read_clustalw_file(argv[1], input_sequences);
53         if_align = false;
54     } else
55         core::data::io::read_fasta_file(argv[1], input_sequences);
56
57     core::protocols::HenikoffSequenceWeights protocol;
58     protocol.n_threads(1);
59     protocol.add_input_sequences(input_sequences);
60     auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
61     protocol.run();
62     auto end = std::chrono::high_resolution_clock::now();
63     std::chrono::duration<double> time_span = std::chrono::duration_cast
64     ↴<std::chrono::duration<double>>(end - start);
65     std::cerr << input_sequences.size() * (input_sequences.size() - 1) / 2.0
66     ↴<< " sequence similarities calculated within " << time_span.count() << "_
67     ↴[s]\n";

```

(continues on next page)

(continued from previous page)

```
60     protocol.print_weights(std::cout);  
61 }  
62 }
```



ap_WeightedOnlineStatistics

ap_WeightedOnlineStatistics reads a file with two columns: real values and their weights. It calculates average value and standard deviation of the data using an online algorithm (Welford method). If no input file is provided, the program calculates the statistics from a random sample.

USAGE:

```
ap_WeightedOnlineStatistics infile
```

EXAMPLE:

```
ap_WeightedOnlineStatistics random_normal_weighted.txt
```

REFERENCE: https://www.johndcook.com/blog/skewness_kurtosis/ https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance#Welford%27s_online_algorithm

Keywords:

- *statistics*

Categories:

- core/calc/statistics/ap_WeightedOnlineStatistics; core/calc/statistics/Random

Input files:

- random_normal_weighted.txt

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <cmath>
3
4 #include <core/index.hh>
5 #include <core/calc/statistics/Random.hh>
6 #include <core/calc/statistics/WeightedOnlineStatistics.hh>
7
8 std::string program_info = R"(
9
10 ap_WeightedOnlineStatistics reads a file with two columns: real values and their
11 ↪weights.
12 It calculates average value and standard deviation of the data using an online
13 ↪algorithm (Welford method).
14
15 If no input file is provided, the program calculates the statistics from a random
16 ↪sample.
17 USAGE:
```

(continues on next page)

(continued from previous page)

```

15    ap_WeightedOnlineStatistics infile
16
17 EXAMPLE:
18     ap_WeightedOnlineStatistics random_normal_weighted.txt
19
20 REFERENCE:
21     https://www.johndcook.com/blog/skewness\_kurtosis/
22     https://en.wikipedia.org/wiki/Algorithms\_for\_calculating\_variance#Welford%27s\_online\_algorithm
23 ) ";
24
25 /** @brief Reads a file with two columns: real values and their weights, and
26  * calculates their mean and stdev.
27  *
28  * If no input file is provided, the program calculates the statistics from a random
29  * sample
30  *
31  * CATEGORIES: core/calc/statistics/ap_WeightedOnlineStatistics; core/calc/statistics/
32  * Random
33  * KEYWORDS: statistics
34  * GROUP: Statistics;
35  */
36
37 int main(const int argc, const char *argv[]) {
38
39     core::calc::statistics::WeightedOnlineStatistics stats;
40     if (argc < 2) {
41         // --- complain about missing program parameter
42         std::cerr << program_info;
43         // ----- Use the random engine if no data is provided
44         core::calc::statistics::Random r = core::calc::statistics::Random::get();
45         r.seed(12345); // --- seed the generator for repeatable results
46         std::normal_distribution<double> normal_random;
47         for (core::index4 n = 0; n < 10000; ++n) {
48             double x = normal_random(r);
49             if (x <= 2.0) stats(x, 0.1); // --- insert the random point with an arbitrary
50             weight = 0.1
51             else for (int i = 0; i < 10; ++i) stats(x, 0.01); // in the tail insert points
52             ten times with weight 1/10
53         }
54     } else {
55         std::ifstream in(argv[1]);
56         double x, w;
57         while (in) {
58             in >> x >> w;
59             stats(x, w);
60         }
61     }
62
63     std::cout << "#count sum_wghts    avg      sdev\n";
64     std::cout << utils::string_format("%d  %lf %f %f \n", stats.cnt(), double(stats.sum_
65     of_weights()), stats.avg(), sqrt(stats.var())));
66 }
```



ap_align_profiles

Read two files with sequence profiles (BioShell's tabular format) and calculates global alignment between them. The gap penalty function depends on observed gap probabilities. Prints sequence alignment as an output. The default for values for base gap penalty is -10 and -1 for gap_open and gap_extend, respectively.

USAGE:

```
ap_align_profiles <file1.profile> <file2.profile> [gap_open gap_extend]
```

EXAMPLE:

```
ap_align_profiles d4proc1-A1.profile d4proc1-A2.profile -11 -2
```

Keywords:

- *FASTA input*
- *Needleman-Wunsch*
- *sequence alignment*

Categories:

- core/alignment/NWAlignerAnyGap

Input files:

- d1exja2-A2.profile
- d4proc1-A2.profile
- d1exja2-A1.profile
- d4proc1-A1.profile

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <chrono>
3 #include <algorithm>
4
5 #include <core/data/io/fasta_io.hh>
6
7 #include <core/alignment/NWAlignerAnyGap.hh>
8 #include <core/alignment/PairwiseSequenceAlignment.hh>
9 #include <core/alignment/scoring/Picasso3.hh>
10 #include <core/alignment/scoring/FrequencyScaledGapPenalty.hh>
11 #include <core/data/io/alignment_io.hh>
```

(continues on next page)

(continued from previous page)

```

12 #include <core/data/sequence/Sequence.hh>
13
14 #include <utils/exit.hh>
15
16 std::string program_info = R"(
17
18 Read two files with sequence profiles (BioShell's tabular format) and calculates_
19 ↪global alignment between them.
20 The gap penalty function depends on observed gap probabilities. Prints sequence_
21 ↪alignment as an output.
22 The default for values for base gap penalty is -10 and -1 for gap_open and gap_extend,
23 ↪respectively.
24
25 USAGE:
26 ap_align_profiles <file1.profile> <file2.profile> [gap_open gap_extend]
27
28 )";
29
30 /**
31 * @brief Calculate all pairwise sequence alignments between sequence profiles
32 */
33 * CATEGORIES: core/alignment/NWAlignerAnyGap
34 * KEYWORDS: FASTA input; Needleman-Wunsch; sequence alignment
35 * GROUP: Alignments
36 */
37 int main(const int argc, const char* argv[]) {
38
39     if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about_
40 ↪missing program parameter
41
42     using namespace core::data::io;
43     using namespace core::data::sequence;
44     using namespace core::alignment::scoring;
45
46     float gap_open = -10;
47     float gap_extend = -1;
48     if(argc == 5) {
49         gap_open = atof(argv[3]);
50         gap_extend = atof(argv[4]);
51     }
52
53     utils::Logger logs("ap_align_profiles");
54
55     // --- the query profile
56     SequenceProfile_SP query = read_profile_table(argv[1]);
57     std::vector<float> query_gap_open, query_gap_extend;
58     query->get_probabilities(core::chemical::Monomer::GAP, query_gap_open);
59     query->get_probabilities(core::chemical::Monomer::GPE, query_gap_extend);
60
61     logs << utils::LogLevel::INFO << "Query sequence is: " << query->sequence << "\n";
62
63     // --- the template profile
64     SequenceProfile_SP tmplt = read_profile_table(argv[2]);
65     std::vector<float> tmplt_gap_open, tmplt_gap_extend;
66     tmplt->get_probabilities(core::chemical::Monomer::GAP, tmplt_gap_open);

```

(continues on next page)

(continued from previous page)

```

65 tmplt->get_probabilities(core:::chemical::Monomer:::GPE,tmplt_gap_extend);
66
67 logs << utils::LogLevel::INFO << "Template sequence is: " << tmplt->sequence<<"\n";
68
69 // --- scoring system
70 const Picasso3 scoring(query,tmplt);
71 const FrequencyScaledGapPenalty gaps(gap_open,gap_extend,query_gap_open,query_gap_
72 →extend,tmplt_gap_open,tmplt_gap_extend);
73
74 // --- create aligner object
75 core:::alignment::NWAlignerAnyGap<Picasso3,FrequencyScaledGapPenalty>_
76 →aligner(std::max(query->length(),tmplt->length()));
77
78 auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
79 float score = aligner.align(scoring,gaps);
80 auto ali = aligner.backtrace();
81 core:::alignment::PairwiseSequenceAlignment seq_ali(ali, query, tmplt);
82 std::cout << seq_ali.get_aligned_query('*') << "\n";
83 std::cout << seq_ali.get_aligned_template('*') << "\n";
84
85 auto end = std::chrono::high_resolution_clock::now();
86 std::chrono::duration<double> time_span = std::chrono::duration_cast
87 →<std::chrono::duration<double>>(end - start);
88 }
```



ap_atom_correlations

ap_atom_correlations reads a multimodel PDB trajectory and calculates correlations between atomic coordinates

USAGE:

```
ap_atom_correlations 2kwi.pdb
```

where 2kwi.pdb is the input file. The output, printed on the screen, provides nine columns: i-atom j-atom covariance(i,j)

where the covariance between is computed

Keywords:

- *PDB input*
- *statistics*

Categories:

- core::data::io::Pdb::fill_structure; core::calc::statistics::OnlineMultivariateStatistics

Input files:

- 2kwi.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <iomanip>
3
4 #include <core/data/io/Pdb.hh>
5 #include <core/calc/statistics/OnlineMultivariateStatistics.hh>
6 #include <utils/string_utils.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 ap_atom_correlations reads a multimodel PDB trajectory and calculates correlations_
12 ↵between atomic coordinates
13
14 USAGE:
15     ap_atom_correlations 2kwi.pdb
16
17 where 2kwi.pdb is the input file. The output, printed on the screen, provides nine_
18 ↵columns:
    i-atom j-atom covariance(i,j)

```

(continues on next page)

(continued from previous page)

```

19 where the covariance between is computed
20
21 );
22
23 /** @brief Reads a multimodel PDB trajectory and calculates correlation between
24   atomic coordinates
25
26 * CATEGORIES: core:::data:::io:::Pdb:::fill_structure; ▾
27   core:::calc:::statistics:::OnlineMultivariateStatistics
28 * KEYWORDS: PDB input; statistics
29 * GROUP: Structure calculations;
30 */
31
32 int main(const int argc, const char *argv[]) {
33
34     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
35   missing program parameter
36
37     core:::data:::io:::Pdb reader(argv[1], core:::data:::io:::is_ca); // --- Read PDB file,
38   may be gzip-ped; take only the lines with C-alphas
39     std::vector<core:::data:::basic:::Vec3> atoms(reader.count_atoms(0));
40     std::vector<double> xyz(atoms.size() * 3);
41     core:::calc:::statistics:::OnlineMultivariateStatistics stats(xyz.size());
42
43     // --- Read all models from the deposit, store alpha carbons from each model as a
44   separate vector of double values
45     for (size_t i = 0; i < reader.count_models(); ++i) { // --- Iterate over all models
46   in the input file
47         reader.fill_structure(i, atoms);
48         // --- utilize coordinates of the new pose
49         for (size_t j = 0; j < atoms.size(); ++j) {
50             xyz[j * 3] = atoms[j].x;
51             xyz[j * 3 + 1] = atoms[j].y;
52             xyz[j * 3 + 2] = atoms[j].z;
53         }
54         stats(xyz);
55     }
56
57     std::vector<std::string> labels;
58     const auto structure = reader.create_structure(0);
59     for(auto it = structure->first_const_residue(); it!=structure->last_const_residue();
60   ++it)
61         labels.push_back(utils::string_format("%4d %3s CA", (**it).id(), (**it).residue_
62   type().code3.c_str()));
63
64
65     std::cout << "# i-resid coord j-resid coord i j correlation\n";
66     std::string xyz_chars = "XYZ";
67     std::cout << "#ipos j-pos correlation\n";
68     for (size_t i = 0; i < xyz.size(); ++i) {
69         for (size_t j = 0; j < xyz.size(); ++j) {
70             std::cout << labels[int(i / 3)] << "-" << xyz_chars[i % 3] << " " <<
71   labels[int(j / 3)] << "-" << xyz_chars[j % 3];
72             std::cout << " " << std::setw(4) << i << " " << std::setw(4) << j << " " <<
73   stats.covar(i, j) << "\n";
74         }
75     }
76 }
```



ap_blast_nonredundant

ap_blast_nonredundant reads output from blast search (XML format) and selects a non-redundant subset of sequences. The subset is selected by hierarchical clustering (complete-linkage approach) of the sequences extracted from the given input file generated by psiblast - last iteration only. Distance between any two sequences is defined as (1 - sequence identity fraction) calculated over alignment extracted from blast results.

USAGE:

```
ap_blast_nonredundant blast-out.xml identity_ratio
```

EXAMPLE:

```
ap_blast_nonredundant 1K25_01+PBP_C2.psi 0.5
```

Keywords:

- *hierarchical clustering*
- blast

Categories:

- core::calc::clustering::HierarchicalClustering; core::data::io::BlastXMLReader

Input files:

- 1K25_01+PBP_C2.psi

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/BlastXMLReader.hh>
4 #include <core/alignment/on_alignment_computations.hh>
5 #include <core/calc/clustering/DistanceByValues1B.hh>
6 #include <core/calc/clustering/HierarchicalCluster.hh>
7 #include <core/calc/clustering/HierarchicalClustering1B.hh>
8
9 #include <utils/exit.hh>
10 #include <utils/LogManager.hh>
11
12 std::string program_info = R"(
13
14 ap_blast_nonredundant reads output from blast search (XML format) and selects a non-
15 redundant subset of sequences.
16
17 The subset is selected by hierarchical clustering (complete-linkage approach) of the_
18 sequences extracted
19 )"
```

(continues on next page)

(continued from previous page)

```

17 from the given input file generated by psiblast - last iteration only. Distance_
18 ↪between any two sequences is defined as
19 (1 - sequence identity fraction) calculated over alignment extracted from blast_
20 ↪results.

21 USAGE:
22     ap_blast_nonredundant blast-out.xml identity_ratio

23 EXAMPLE:
24     ap_blast_nonredundant 1K25_01+PBP_C2.psi 0.5
25
26 ) ";

27
28 /** @brief Reads output from blast search (XML format) and selects a non-redundant_
29 ↪subset of sequences
30
31 * CATEGORIES: core:::calc:::clustering:::HierarchicalClustering;_
32 ↪core:::data:::BlastXMLReader
33 * KEYWORDS: hierarchical clustering; blast
34 * GROUP: File processing; Data filtering
35
36 int main(const int argc, const char *argv[]) {
37
38     utils::LogManager::INFO();
39
40     if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about_
41 ↪missing program parameter
42
43     using namespace core:::data:::io;
44     using namespace core:::calc:::clustering;
45
46     BlastXMLReader blast_reader;
47     auto hits = blast_reader.parse(argv[1]);
48     std::vector<std::string> sequences; // --- vector containing all sequences from the_
49 ↪last iteration of psiblast
50     std::vector<std::string> seq_ids; // --- vector containing identifiers of these_
51 ↪sequences; sequences.size() equals to seq_ids.size()
52     std::map<std::string, std::string> seq_by_id; // --- maps sequence IDs (keys) to the_
53 ↪respective sequences (values)
54     for (const auto hsp: hits.back()) {
55         seq_ids.push_back(hsp.hit_accession());
56         sequences.push_back(std::string(hsp.query_start() - 1, '-') + hsp.sbjct());
57         seq_by_id[seq_ids.back()] = sequences.back();
58     }
59
60     DistanceByValues1B dist(seq_ids, 254, 255);
61     for(core:::index4 i=1;i<sequences.size();++i)
62         for(core:::index4 j=0;j<i;++j) {
63             double val = core:::alignment::sum_identical(sequences[i], sequences[j]);
64             val /= std::min(sequences[i].length(), sequences[j].length());
65             core:::index1 d = core:::index1(250 * (1-val));
66             // std::cout << i << " " << j << " " << core:::alignment::sum_
67             ↪identical(sequences[i], sequences[j])
68             // << " " << val << " " << int(d) << "\n";
69             dist.set(i, j, d);
70             dist.set(j, i, d);
71         }
72     }

```

(continues on next page)

(continued from previous page)

```

65 HierarchicalClustering1B hac(dist.labels(), "");
66 CompleteLink1B merge;
67 hac.run_clustering(dist, merge);
68
69 // --- Uncomment the line below to print the clustering tree (may be a lot of_
70 // output)
71 // hac.write_merging_steps(std::cerr);
72
73 std::vector<std::string> elements; // --- vector used to store elements of each_
74 // cluster
75 core::index1 cutoff = core::index1((1.0 - atof(argv[2])) * 250);
76 std::cerr << "# clustering cutoff set to " << int(cutoff) << "\n";
77 auto clusters = hac.get_clusters(cutoff, 1);
78 std::cerr << "# " << sequences.size() << " hits' set reduced to " << clusters.
79 //size() << " representatives\n";
80 for (core::index2 i = 0; i < clusters.size(); i++) {
81     const auto & c = clusters[i];
82     std::string medoid_id = medoid_by_average_distance<core::index1, std::string,>(
83         DistanceByValues1B >(c, dist).medoid;
84     elements.clear();
85     collect_leaf_elements(std::static_pointer_cast<BinaryTreeNode<std::string>>(c),
86     //elements);
87     std::cout << "> " << medoid_id;
88     if(elements.size() > 1) {
89         std::cout << " represents also:";
90         for(const std::string & e: elements)
91             if(e!=medoid_id) std::cout << " "<<e;
92     }
93     core::data::sequence::remove_gaps(seq_by_id[medoid_id]);
94     std::cout << "\n" << seq_by_id[medoid_id] << "\n\n";
95 }
96 }
```



ap_blastxml_to_fasta

ap_blastxml_to_fasta reads a XML file produced by PsiBlast and extracts sequences of all hits. The list of hits is divided into sections, according to the psiblast iteration when a given subject sequence was detected. The sequences are written on the screen in FASTA format

USAGE:

```
ap_blastxml_to_fasta blastout.xml
```

EXAMPLE:

```
ap_blastxml_to_fasta "1K25_01+PBP_C2.psi"
```

Keywords:

- *XML*
- *data structures*

Categories:

- core::data::io::XML; core::algorithms::trees::TreeNode

Input files:

- 1K25_01+PBP_C2.psi

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <fstream>
3
4 #include <stdexcept>
5 #include <core/data/io/BlastXMLReader.hh>
6 #include <core/algorithms/trees/algorithms.hh>
7 #include <core/index.hh>
8 #include <core/data/io/XML.hh>
9
10 #include <core/data/io/XMLElement.hh>
11 #include <core/data/io/Hsp.hh>
12 #include <utils/exit.hh>
13
14 std::string program_info = R"(
15
16 ap_blastxml_to_fasta reads a XML file produced by PsiBlast and extracts sequences of_
17 ↵all hits.
```

(continues on next page)

(continued from previous page)

```

18 The list of hits is divided into sections, according to the psiblast iteration when a_
19 ↪given subject
20 sequence was detected. The sequences are written on the screen in FASTA format
21
22 USAGE:
23   ap_blastxml_to_fasta blastout.xml
24 EXAMPLE:
25   ap_blastxml_to_fasta "1K25_01+PBP_C2.psi"
26 )
27
28 using namespace core::data::io;
29
30 struct BlastXMLVisitor {
31
32   void operator() (std::shared_ptr<core::algorithms::trees::TreeNode<XMLElementData>>_
33   ↪n) {
34
35     if (n->element.name() == "Hsp") {
36       auto xmlel = std::static_pointer_cast<XMLElement>(n);
37       auto xmlel_root = std::static_pointer_cast<XMLElement>(n->get_root()->get_
38       ↪root());
39
40       const std::string &sequence = xmlel->find_value("Hsp_hseq");
41       const std::string &seq_name = xmlel_root->find_value("Hit_accession");
42       std::cout << "> " << seq_name << "\n" << sequence << "\n";
43     }
44   }
45 }
46
47 /** @brief Reads XML produced by psiblast and creates FASTA file containing all hits
48  *
49  * CATEGORIES: core::data::io::XML; core::algorithms::trees::TreeNode
50  * KEYWORDS: XML; data structures
51  * GROUP: File processing;Format conversion
52  */
53 int main(int argc, char *argv[]) {
54
55   if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
56   ↪missing program parameter
57
58   XML xxx;
59   std::shared_ptr<XMLElement> root = xxx.load_data(argv[1]);
60
61   auto it = root->begin();
62   while ((*it)->element.name() != "BlastOutput_iterations") ++it; // --- Visit_
63   ↪branches until you find BlastOutput_iterations
64
65   core::index2 iteration_counter = 0;
66   for (const auto &v : **it) {
67     if (v->element.name() == "Iteration") {
68       std::cout << "\n# ----- iteration " << ++iteration_counter << " ----- \n";
69       core::algorithms::trees::depth_first_preorder((*it)->get_right(),_
70       ↪BlastXMLVisitor());
71     }
72   }
73 }
```

(continues on next page)

(continued from previous page)

```
69     return 0;  
70 }
```



ap_blastxml_to_hsp

Reads a XML file produced by PsiBlast and extracts High Scoring Pairs (HSP). Program prints a table where each row corresponds to a single HSP found in the input file. The table's columns provide: - hit sequence ID - hit length - alignment score - number of gaps - gap percentage - number of identical positions - identity percentage - e-value - query start position - subject start position - subject sequence

USAGE:

```
ap_blastxml_to_hsp blastout.xml
```

EXAMPLE:

```
ap_blastxml_to_hsp "1K25_01+PBP_C2.psi"
OUTPUT:
hit sequence ID      len score gaps  gap% ident ident%      evalue   qpos tpos sequence
[      UniRef50_A0A0E9GHR2]  139    220    0 ( 0%) 28 ( 50%) 3.56e-22    3 ↴84 --ELPD MYGWTKENVQVF GKTGIEVTYQGNGSHVTAQSSDTGTALKLKKLTITLGE
[      UniRef50_A0A111B192]  151    221    0 ( 0%) 48 ( 82%) 4.26e-22    1 ↴94 AEEVPD MYGWTKETAETLAKWL NIELEFQGSGSTVQKQDVR RANTAIKDIKKITL LGD
[      UniRef50_P59676]     750    229    0 ( 0%) 48 ( 82%) 2.43e-21    1 ↴693 AEEVPD MYGWTKETAETLAKWL NIELEFQGSGSTVQKQDVR RANTAIKDIKKITL LGD
[      UniRef50_T0UT66]     466    227    0 ( 0%) 29 ( 50%) 3.87e-21    2 ↴410 -DAVPD MYGWTKNADIFGEWTGIEITYKGSGKKVTQSVKMNTSLNKT KITL LGD
[      UniRef50_A0A0T8ADZ4]  322    223    0 ( 0%) 58 (100%) 5.32e-21    1 ↴265 VEEIPD MYGWKKETAETFAKWL DIELEFEGSGS VVQKQDVR TNAIKNIKKI KLT LGD
[      UniRef50_A0A139PMG7]  412    222    0 ( 0%) 48 ( 82%) 1.37e-20    1 ↴355 AEEVPD MYGWTKATAETLAKWL NIELEFEGSGSTVQKQDVR RANTAIKDIKKITL LGD
[      UniRef50_A0A0E9EQ17]  236    212    0 ( 0%) 29 ( 51%) 5.33e-20    3 ↴181 --EMPDMYGWTKNVETFGEWLGIKVHVKS KGSKVVAQSVKTNASLKKIKEITITL LGD
```

Keywords:

- *XML*
- *data structures*
- HSP

Categories:

- core::data::io::XML; core::algorithms::trees::TreeNode

Input files:

- 1K25_01+PBP_C2.psi

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <fstream>
3
4 #include <stdexcept>
5 #include <core/data/io/BlastXMLReader.hh>
6 #include <core/algorithms/trees/algorithms.hh>
7 #include <core/index.hh>
8 #include <core/data/io/XML.hh>
9
10 #include <core/data/io/XMLElement.hh>
11 #include <core/data/io/Hsp.hh>
12 #include <utils/exit.hh>
13
14 std::string program_info = R"(

15
16 Reads a XML file produced by PsiBlast and extracts High Scoring Pairs (HSP). Program_
17 ↪prints a table
18 where each row corresponds to a single HSP found in the input file. The table's_
19 ↪columns provide:
20   - hit sequence ID
21   - hit length
22   - alignment score
23   - number of gaps
24   - gap percentage
25   - number of identical positions
26   - identity percentage
27   - e-value
28   - query start position
29   - subject start position
30   - subject sequence

31 USAGE:
32   ap_blastxml_to_hsp blastout.xml
33 EXAMPLE:
34   ap_blastxml_to_hsp "1K25_01+PBP_C2.psi"
35 OUTPUT:
36     hit sequence ID      len score gaps  gap% ident ident%      evalue    qpos tpos_
37   ↪sequence
38   [      UniRef50_A0A0E9GHR2]  139    220    0 ( 0%)  28 ( 50%)  3.56e-22    3 ↴
39   ↪84 --ELPD MYGWTKENVQVFGKWTGIEVTYQGNNGSHVTAQSSDTGTALKLKKLTITLGE
40   [      UniRef50_A0A111B192]  151    221    0 ( 0%)  48 ( 82%)  4.26e-22    1 ↴
41   ↪94 AEEVPD MYGWTKETAETLAKWLNLIELEFQGSGSTVQKQDVRANTAIKDIKKITLTLGD
42   [      UniRef50_P59676]    750    229    0 ( 0%)  48 ( 82%)  2.43e-21    1 ↴
43   ↪693 AEEVPD MYGWTKETAETLAKWLNLIELEFQGSGSTVQKQDVRANTAIKDIKKITLTLGD
44   [      UniRef50_TOUT66]    466    227    0 ( 0%)  29 ( 50%)  3.87e-21    2 ↴
45   ↪410 -DAVPD MYGWTKNADIFGEWTGIEITYKGSGKKVTKQSVKMNTSLNKTKitLTLGD
46   [      UniRef50_A0A0T8ADZ4]  322    223    0 ( 0%)  58 (100%)  5.32e-21    1 ↴
47   ↪265 VEEIPD MYGWKKETAETFAKWL DIELEFEGSGS VVQKQDVRNTAIKNIKKIKLTLGD
48   [      UniRef50_A0A139PMG7]  412    222    0 ( 0%)  48 ( 82%)  1.37e-20    1 ↴
49   ↪355 AEEVPD MYGWTKATAETLAKWLNLIELEFEGSGSTVQKQDVRANTAIKDIKKITLTLGD
50   [      UniRef50_A0A0E9EQ17]  236    212    0 ( 0%)  29 ( 51%)  5.33e-20    3 ↴
51   ↪181 --EMPDMYGWTKNVETFGEWLGIKVHVSKGSKVVAQSVKTNASLKKIKEITITLGD
52
53 ) ";

```

(continues on next page)

(continued from previous page)

```
46 using namespace core::data::io;
47 /** @brief Reads XML produced by psiblast and creates High Scoring FASTA Pair
48 *
49 * CATEGORIES: core::data::io::XML; core::algorithms::trees::TreeNode
50 * KEYWORDS: XML; data structures; HSP
51 * GROUP: File processing;Format conversion
52 */
53
54 int main(int argc, char *argv[]) {
55
56     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
missing program parameter
57
58     // ----- creates BlastXMLReader object
59     BlastXMLReader p;
60     // ----- parse data from XML file and returns it to variable
61     auto iterations = p.parse(argv[1]);
62     std::cout << Hsp::output_header << "\n";
63     // print Hsp line for every hit for every iteration
64     for (core::index2 i = 0; i < iterations.size(); i++) {
65         for (core::index2 j = 0; j < iterations[i].size(); j++) std::cout <<
iterations[i][j] << "\n";
66     }
67
68     return 0;
69 }
```



ap_bootstrap_quantile

ap_bootstrap_quantile reads a file with real values and calculates statistics for a given quantile. The statistics: expected quantile value and its standard deviation are computed by 100-fold bootstrap procedure. If no input file is provided, the program calculates the statistics of a random sample withdrawn from a normal distribution (mean=0.0, variance = 1.0)

USAGE:

```
ap_bootstrap_quantile quantile_value infile  
ap_bootstrap_quantile quantile_value
```

Keywords:

- *random numbers*
- *statistics*

Categories:

- core:::calc::statistics::simple_statistics.hh; core:::calc::statistics::Random

Input files:

- random_normal.txt

Output files:

- stdout.out

Program source:

```
1 #include <iostream>  
2  
3 #include <core/index.hh>  
4 #include <core/calc/statistics/Random.hh>  
5 #include <core/calc/statistics/OnlineStatistics.hh>  
6 #include <core/calc/statistics/simple_statistics.hh>  
7 #include <utils/exit.hh>  
8  
9 std::string program_info = R"(  
10  
11 ap_bootstrap_quantile reads a file with real values and calculates statistics for a  
12 ↪given quantile.  
13  
14 The statistics: expected quantile value and its standard deviation are computed by  
15 ↪100-fold bootstrap  
16 procedure.  
17  
18 If no input file is provided, the program calculates the statistics of a random  
19 ↪sample withdrawn  
20 from a normal distribution (mean=0.0, variance = 1.0)
```

(continues on next page)

(continued from previous page)

```

18 USAGE:
19     ap_bootstrap_quantile quantile_value infile
20     ap_bootstrap_quantile quantile_value
21
22
23 ) ";
24
25 /** @brief Reads a file with real values and calculates statistics for a given_
26    quantile
27
28 * If no input file is provided, the program calculates the statistics from a random_
29    sample
30
31 * CATEGORIES: core::calc::statistics::simple_statistics.hh;_
32    core::calc::statistics::Random
33 * KEYWORDS: random numbers; statistics
34 * GROUP: Statistics;
35 */
36
37 int main(const int argc, const char *argv[]) {
38
39     if(argc ==1)    utils::exit_OK_with_message(program_info);
40
41     double quantile_level = atof(argv[1]);
42
43     core::calc::statistics::OnlineStatistics stats;
44     if(argc < 3) {
45         // --- complain about missing program parameter
46         //std::cerr << program_info;
47         // ----- Use the random engine if no data is provided - for testing purposes
48         size_t n_data = 10000;
49         std::vector<double> data(n_data);
50         core::calc::statistics::Random r = core::calc::statistics::Random::get();
51         r.seed(12345); // --- seed the generator for repeatable results
52         core::calc::statistics::NormalRandomDistribution<double> normal_random;
53         for (core::index4 n = 0; n < n_data; ++n) data[n] = normal_random(r);
54         const auto out = core::calc::statistics::bootstrap_quantile(data, quantile_level,
55             100);
56         std::cout << "q-level value stdev\n" << quantile_level << " " << out.first << " "
57             << out.second << "\n";
58     } else {
59         std::vector<double> data;
60         for(size_t i_file=2;i_file<argc;++i_file) {
61             data.clear();
62             std::ifstream in(argv[i_file]);
63             double r;
64             while(in) {
65                 in >> r;
66                 data.push_back(r);
67             }
68             in.close();
69             const auto out = core::calc::statistics::bootstrap_quantile(data, quantile_
70             level, 100);
71             std::cout << "fname q-level value stdev\n" << argv[i_file] << " " << quantile_
72             level << " " << out.first << " "
73                 << out.second << "\n";
74         }
75     }
76 }
```

(continues on next page)

(continued from previous page)

68
69

}



ap_build_crystal

ap_create_crystal reads a given PDB file and prints all atoms in a unit cell.

USAGE:

```
ap_create_crystal 5edw.pdb
```

Keywords:

- *PDB input*
- *PDB line filter*
- *Structure*

Categories:

- core/data/io/Pdb

Input files:

- 5edw.pdb
- 3dcg.pdb

Output files:

- 3dcg_uc.pdb
- stdout.out
- 5edw_uc.pdb

Program source:

```
1 #include <iostream>
2 #include <core/data/io/Pdb.hh>
3 #include <utils/exit.hh>
4 #include <iomanip>
5
6 std::string program_info = R"(
7
8 ap_create_crystal reads a given PDB file and prints all atoms in a unit cell.
9
10 USAGE:
11     ap_create_crystal 5edw.pdb
12
13 )";
14
15 /** @brief ap_create_crystal reads a given PDB file and prints all atoms in a unit
16  * cell.
17  * * CATEGORIES: core/data/io/Pdb;
```

(continues on next page)

(continued from previous page)

```

18 * KEYWORDS: PDB input; PDB line filter; Structure
19 * GROUP: Structure calculations;
20 */
21 int main(const int argc, const char *argv[]) {
22
23     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
24     ↵missing program parameter
25
26     core::data::io::Pdb reader(argv[1], // file name (PDB format, may be gzip-ped)
27     core::data::io::keep_all,           // a predicate - now read all atoms
28     core::data::io::keep_all, true);   // parse PDB header
29
30     std::shared_ptr<core::data::io::Remark290> r290 = reader.symmetry_operators();
31     core::data::structural::Structure_SP s = reader.create_structure(0);
32     std::cout << "# Symmetry operators found: " << r290->count_operators() << "\n";
33     core::data::basic::Vec3 tmp;
34     core::index2 im = 0;
35     for (const auto &rt: *r290) {
36         std::cout << "MODEL " << std::setw(6) << ++im << "\n";
37         for (auto a_it = s->first_atom(); a_it != s->last_atom(); ++a_it) {
38             tmp.set(**a_it);
39             rt.apply(**a_it);
40             std::cout << (*a_it)->to_pdb_line() << "\n";
41             (*a_it)->set(tmp);
42         }
43         std::cout << "ENDMDL\n";
44     }
}

```



ap_calc_rdf

ap_calc_rdf calculates Radial Distribution Function (RDF) over a trajectory If a multi-model PDB file was given, the program combines the data from all models

USAGE:

```
ap_calc_rdf  trajectory.pdb HOH O box_side
```

where trajectory.pdb is the input file multimodel-PDB file, HOH and O defines the atom in a molecules for which the RDF will be evaluated

Keywords:

- *PDB input*
- *simulation*

Categories:

- core::data::basic::Vec3

Program source:

```

1 #include <iostream>
2
3 #include <core/index.hh>
4 #include <core/data/io/Pdb.hh>
5 #include <core/data/basic/Vec3I.hh>
6 #include <core/calc/statistics/Histogram.hh>
7
8 #include <utils/exit.hh>
9
10 std::string program_info = R"(
11
12 ap_calc_rdf calculates Radial Distribution Function (RDF) over a trajectory
13
14 If a multi-model PDB file was given, the program combines the data from all models
15
16 USAGE:
17     ap_calc_rdf  trajectory.pdb HOH O box_side
18
19 where trajectory.pdb is the input file multimodel-PDB file, HOH and O defines the_
20 ↪atom in a molecules for which
21 the RDF will be evaluated
22 )
23
24 /**
25 * @brief Calculates Radial Distribution Function
26 *
27 * @CATEGORIES: core::data::basic::Vec3
28 * @KEYWORDS: PDB input; simulation
29 * @GROUP: Structure calculations;
30 */
31 int main(const int argc, const char *argv[]) {

```

(continues on next page)

(continued from previous page)

```

31
32     if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about_
33     ↪missing program parameter
34
34     double L = utils::from_string<double>(argv[4]); // The third parameter is the box_
35     ↪width (in Angstroms)
36     core::data::basic::Vec3I::set_box_len(L);
37     core::data::io::Pdb reader(argv[1]); // --- file name (PDB format, may be gzip-ped)
38
38     core::index4 n_atoms = reader.count_atoms(0);
39     core::index4 n_frmes = reader.count_models();
40
41     std::vector<core::data::basic::Vec3I> frame_i(n_atoms);
42
43     core::calc::statistics::Histogram<double, core::index4> h(0.01, 0, L/4.0);
44     // ----- Load coordinates to memory and accumulate RDF -----
45     for (int i_start = 0; i_start < n_frmes; ++i_start) {
46         reader.fill_structure(i_start, frame_i);
47         for(core::index4 i=1;i<n_atoms;++i) {
48             for(core::index4 j=0;j<i;++j) {
49                 double d = frame_i[i].distance_to(frame_i[j]);
50                 h.insert(d);
51             }
52         }
53     }
54
55     double norm = 4 * M_PI * n_atoms / pow(L, 3.0) * n_frmes;
56     for(core::index4 i=0;i<h.count_bins();++i) {
57         std::cout << h.bin_middle_val(i) << " " << h.get_bin(i) / (norm * h.bin_middle_
58         ↪val(i) * h.bin_middle_val(i)) << " "
59         << h.get_bin(i) << "\n";
60     }
61     // ----- Calculate displacement -----
62 }
```



ap_caonly_multimodel

Reads a file with names of PDB files and creates a single multimodel PDB file. Each model is stored as a separate model within that file. Only C-alpha atoms are written to the output PDB

EXAMPLE:

```
ap_caonly_multimodel cat_list
where cat_list is a file with a content like:
```

2gb1-model1.pdb 2gb1-model2.pdb 2gb1-model3.pdb 2gb1-model4.pdb

Keywords:

- *PDB input*
- CA only
- *structure selectors*
- *PDB output*
- *PDB line filter*

Categories:

- core::data::io::is_ca

Input files:

- 2gb1-model3.pdb
- 2gb1-model1.pdb
- 2gb1-model4.pdb
- cat_list
- 2gb1-model2.pdb

Output files:

- stdout.out
- all.pdb

Program source:

```
1 #include <vector>
2 #include <string>
3 #include <fstream>
4 #include <iostream>
5 #include <cstring>
6
7 #include <core/data/io/Pdb.hh>
```

(continues on next page)

(continued from previous page)

```

8 #include <utils/options/output_options.hh>
9 #include <utils/options/input_options.hh>
10 #include <utils/exit.hh>
11
12 using namespace core::data::io; // PDB is from this namespace
13 using namespace core::data::structural;
14 using namespace core::calc::structural;
15 using namespace utils;
16
17 std::string program_info = R"(
18
19 Reads a file with names of PDB files and creates a single multimodel PDB file. Each_
20 ↴model
21 is stored as a separate model within that file. Only C-alpha atoms are written to the_
22 ↴output PDB
23
24 EXAMPLE:
25   ap_caonly_multimodel cat_list
26 where cat_list is a file with a content like:
27
28 2gb1-model1.pdb
29 2gb1-model2.pdb
30 2gb1-model3.pdb
31 2gb1-model4.pdb
32 )";
33
34 /**
35 * @brief Reads cat_list of pdb files and creates multimodel pdb with CA only
36 */
37
38 * CATEGORIES: core::data::io::is_ca
39 * KEYWORDS: PDB input; CA only; structure selectors;PDB output; PDB line filter
40 * GROUP: File processing;Format conversion
41 */
42
43 int main(const int argc, const char *argv[]) {
44
45   if (argc < 2) utils::exit_OK_with_message(program_info);
46
47   PdbLineFilter filter = core::data::io::is_ca;
48   std::ofstream out;
49   out.open("all.pdb");
50   std::ifstream pdb_list(argv[1]);
51   std::string pdb_file;
52   std::getline(pdb_list, pdb_file);
53   Pdb pdb = Pdb(pdb_file, filter);
54   Structure_SP strctr = pdb.create_structure(0);
55   out << "MODEL 1\n";
56   for (auto it = strctr->first_atom(); it != strctr->last_atom(); it++) out << (*it)->
57   ↴to_pdb_line() << "\n";
58   out << "ENDMDL\n";
59   core::index4 i = 2;
60   while (std::getline(pdb_list, pdb_file)) {
61     Pdb pdb = Pdb(pdb_file, filter);
62     pdb.fill_structure(0, *strctr);
63     out << utils::string_format("MODEL %6d\n", i);
64     for (auto it = strctr->first_atom(); it != strctr->last_atom(); it++)
65       out << (*it)->to_pdb_line() << "\n";
66     out << "ENDMDL\n";
67     i++;
68   }
69 }
```

(continues on next page)

(continued from previous page)

```
62    }
63
64    out.close();
65 }
```



ap_contact_map_overlap

ap_contact_map_overlap calculates overlap between contact maps calculated for two (or more) structures. The overlap, defined as Jaccard coefficient, is computed between the native structure and every model found in models.pdb; map-type can take one of the following values: CA CB and SC for C-alpha, C-beta and all atom side chain, respectively. Contact is recorded when any selected atoms from two different residues are closer to each other than the give cutoff. If only one PDB file is given, the program computes calculates overlap between the first and any other model found in that file

USAGE:

```
ap_contact_map_overlap map-type native.pdb models.pdb cutoff
```

EXAMPLE:

```
ap_contact_map_overlap SC 2gb1.pdb 2gb1-model1.pdb 4.5
```

REFERENCE: https://en.wikipedia.org/wiki/Jaccard_index

Keywords:

- *PDB input*
- *contact map*

Categories:

- core::calc::structural::ContactMap

Input files:

- 2gb1-model1.pdb
- 2kwi.pdb
- 2gb1.pdb
- 2gb1-model2.pdb

Output files:

- stdout.out

Program source:

```
1 #include <vector>
2 #include <algorithm>
3
4 #include <core/data/io/Pdb.hh>
5 #include <core/calc/structural/interactions/ContactMap.hh>
6
7 #include <utils/exit.hh>
```

(continues on next page)

(continued from previous page)

```

9 std::string program_info = R"
10
11 ap_contact_map_overlap calculates overlap between contact maps calculated for two (or_
12 ↵more) structures.
13
14 The overlap, defined as Jaccard coefficient, is computed between the native structure
15 and every model found in models.pdb; map-type can take one of the following values:
16 CA CB and SC for C-alpha, C-beta and all atom side chain, respectively.
17 Contact is recorded when any selected atoms from two different residues are closer to_
18 ↵each other than
19 the give cutoff.
20
21 If only one PDB file is given, the program computes calculates overlap between the_
22 ↵first and any other
23 model found in that file
24
25 USAGE:
26     ap_contact_map_overlap map-type native.pdb models.pdb cutoff
27
28 EXAMPLE:
29     ap_contact_map_overlap SC 2gb1.pdb 2gb1-model1.pdb 4.5
30
31 REFERENCE:
32     https://en.wikipedia.org/wiki/Jaccard_index
33 )";
34
35 /**
36 * @brief Calculates overlap between contact maps calculated for two (or more)_
37 ↵structures
38 *
39 * @CATEGORIES: core::calc::structural::ContactMap
40 * @KEYWORDS: PDB input; contact map
41 * @GROUP: Structure calculations;
42 */
43 int main(const int argc, const char *argv[]) {
44
45     if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about_
46 ↵missing program parameter
47
48     using namespace core::data::io; // PDB is from this namespace
49     using namespace core::data::structural;
50     using namespace core::data::structural::selectors; // --- for all structural_
51 ↵selectors
52     using namespace core::calc::structural::interactions;
53
54     AtomSelector_SP selector = std::make_shared<IsSC>();
55     core::data::io::PdbLineFilter filter = core::data::io::is_not_water;
56     if (std::strcmp(argv[1], "CA") == 0) {
57         selector = std::make_shared<IsNamedAtom>(" CA ");
58         core::data::io::PdbLineFilter filter = core::data::io::is_ca;
59     }
60     if (std::strcmp(argv[1], "CB") == 0) {
61         core::data::io::PdbLineFilter filter = core::data::io::is_cb;
62         selector = std::make_shared<IsNamedAtom>(" CB ");
63     }
64     double cutoff = utils::from_string<double>(argv[(argc==5) ? 4 : 3]); // The third/
65 ↵fourth parameter is the contact distance (in Angstroms)
66
67 }
```

(continues on next page)

(continued from previous page)

```

59 // --- This is the case when user gave a reference structure (e.g. the native one)
60 if (argc == 5) {
61     Pdb pdb_native = Pdb(argv[2], filter);
62     core::data::structural::Structure_SP reference_structure = pdb_native.create_
63     ↪structure(0);
64     ContactMap_SP reference_map = std::make_shared<ContactMap>(*reference_structure,_
65     ↪cutoff, selector);
66
67     Pdb models_pdb = Pdb(argv[3], filter);
68     ContactMap cmap(*reference_structure, cutoff, selector);
69     std::vector<std::pair<core::index2, core::index2>> contacts;
70     for (core::index4 i = 0; i < models_pdb.count_models(); ++i) {
71         models_pdb.fill_structure(i,*reference_structure);
72         ContactMap cmap(*reference_structure, cutoff, selector);
73         std::cout << i << " " << reference_map->jaccard_overlap_coefficient(cmap) << "\n"
74         ↪";
75     }
76 } else {
77     Pdb models_pdb = Pdb(argv[2], filter);
78     core::data::structural::Structure_SP structure = models_pdb.create_structure(0);
79     std::vector<std::vector<std::pair<core::index2, core::index2>>> models(models_pdb.
80     ↪count_models());
81     for (int i = 0; i < models_pdb.count_models(); ++i) {
82         models_pdb.fill_structure(i,*structure);
83         ContactMap cmap(*models_pdb.create_structure(i), cutoff, selector);
84         cmap.nonempty_indexes(models[i]);
85     }
86
87     for (core::index4 i = 1; i < models.size(); ++i) {
88         for (core::index4 j = 0; j < i; ++j)
89             std::cout << i << " " << j << " "
90             ↪<< core::calc::structural::interactions::jaccard_overlap_
91             ↪coefficient(models[i], models[j]) << "\n";
92     }
93 }
94 }
```



ap_crmsd_on_common_subset

Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on common subset of atoms Program ensures the same number of atoms.

USAGE:

```
./ap_crmsd_on_common_subset -in:pdb:native=file.pdb -select:chains=A -select:bb -  
→in:pdb=rebuilt.pdb
```

EXAMPLES:

```
./ap_crmsd_on_common_subset -in:pdb:native=6h60.pdb -select:chains=A -select:bb -  
→in:pdb=6h60_A_rebuilt.pdb
```

REFERENCE: Kabsch, W. "A Solution for the Best Rotation to Relate Two Sets of Vectors." Acta Cryst (1976) 32 922-923

Keywords:

- *PDB input*
- *crmsd*

Categories:

- core/calc/structural/transformations/Crmsd

Input files:

- 6h60_A_rebuilt.pdb
- 6h60.pdb

Program source:

```
1 #include <iostream>  
2  
3 #include <core/data/io/Pdb.hh>  
4 #include <core/calc/structural/transformations/Crmsd.hh>  
5 #include <utils/options/Option.hh>  
6 #include <utils/options/OptionParser.hh>  
7 #include <utils/options/input_options.hh>  
8 #include <utils/options/structures_from_cmdline.hh>  
9 #include <utils/options/select_options.hh>  
10 #include <utils/options/selector_from_cmdline.hh>  
11 #include <utils/exit.hh>  
12 #include <core/algorithms/basic_algorithms.hh>  
13 #include <utils/options/output_options.hh>  
14 #include <core/data/structural/selectors>SelectChainBreaks.hh>  
15  
16 std::string program_info = R"(  
17  
18 Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on common subset of  
→atoms
```

(continues on next page)

(continued from previous page)

```

19 Program ensures the same number of atoms.
20
21 USAGE:
22 ./ap_crmsd_on_common_subset -in:pdb:native=file.pdb -select:chains=A -select:bb -
23   ↪in:pdb=rebuilt.pdb
24
25 EXAMPLES:
26 ./ap_crmsd_on_common_subset -in:pdb:native=6h60.pdb -select:chains=A -select:bb -
27   ↪in:pdb=6h60_A_rebuilt.pdb
28
29 REFERENCE:
30 Kabsch, W. "A Solution for the Best Rotation to Relate Two Sets of Vectors."
31 Acta Cryst (1976) 32 922-923
32 )
33
34 void extract_atom_by_name(const core::data::structural::Structure_SP s, std::vector<
35   ↪std::string> & atom_names,
36   std::map<std::string, core::data::structural::PdbAtom_SP> & atoms_by_name,
37   bool aa_only = true, bool skip_chainbreaks=true) {
38
39   using namespace core::data::structural;
40
41   // --- select only amino acids
42   selectors::IsAA is_aa;
43   // --- remove atoms at chain breaks
44   selectors::ProperlyConnectedCA at_gap;
45   for (auto c: *s) {
46     for (auto r: *c) {
47       if (r == nullptr) continue;
48       if (aa_only && ! is_aa(*r)) continue;
49       if (skip_chainbreaks && ! at_gap(*r)) continue;
50       for (auto a: (*r)) {
51         std::string code = c->id() + (*r).residue_type().code3 + (*r).residue_
52         ↪id() + a->atom_name();
53         atom_names.push_back(code);
54         atoms_by_name[code] = a;
55       }
56     }
57   }
58
59 /**
60  * @brief Calculates crmsd value on C-alpha coordinates. The program prints just the
61  ↪crmsd value.
62  *
63  * @CATEGORIES: core/calc/structural/transformations/Crmsd
64  * @KEYWORDS: PDB input; crmsd
65  * @GROUP: Structure calculations;
66  */
67 int main(const int argc, const char *argv[]) {
68
69   if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
70   ↪missing program parameter
71
72   using core::data::basic::Vec3;
73   using namespace core::calc::structural::transformations;

```

(continues on next page)

(continued from previous page)

```

70     using namespace core::data::io;
71     using namespace utils::options;
72     using namespace core::data::basic;
73     using namespace core::data::structural;
74     using namespace core::data::structural::selectors;
75     using namespace core::calc::structural;
76     using namespace core::calc::structural::transformations;
77
78     Crmsd<std::vector<Vec3>, std::vector<Vec3>> rms;
79
80     utils::options::OptionParser &cmd = OptionParser::get("ap_crmsd_on_common_subset"
81     );
82     // ----- input PDB structures
83     cmd.register_option(input_pdb, input_pdb_native, input_pdb_list, input_pdb_path,
84     input_pdb_header);
85     // ----- selecting options
86     cmd.register_option(select_ca, select_bb, select_bb_cb, select_cb, select_atoms_
87     by_name, select_aa, select_chains, all_models);
88     // ----- output PDB structures
89     cmd.register_option(output_pdb, output_name_prefix);
90     cmd.register_option(verbose, db_path);
91
92     if (!cmd.parse_cmdline(argc, argv)) return 1;
93     Structure_SP native = native_from_cmdline();
94     std::vector<core::data::structural::Structure_SP> structures;
95     std::vector<std::string> structure_ids;
96     utils::options::structures_from_cmdline(structure_ids, structures);
97
98     // ----- Load atoms from the native structure
99     std::vector<std::string> native_names;
100    std::map<std::string, PdbAtom_SP> native_name_to_atom;
101    extract_atom_by_name(native, native_names, native_name_to_atom, select_aa.was_
102    used());
103    std::sort(native_names.begin(), native_names.end());
104
105    std::vector<Vec3> q, t;
106    std::cout << "native nres, atoms " << native->count_residues() << " "
107    << native->count_atoms() << "\n";
108    std::cout << "model nres, atoms " << structures[0]->count_residues() << " "
109    << structures[0]->count_atoms() << "\n";
110
111    for (auto i = 0; i < structures.size(); ++i) {
112        // ----- Load atoms from a query structure
113        std::vector<std::string> q_names;
114        std::map<std::string, PdbAtom_SP> q_name_to_atom;
115        extract_atom_by_name(structures[i], q_names, q_name_to_atom, select_aa.was_
116        used());
117        std::sort(q_names.begin(), q_names.end());
118
119        // ----- find the common subset
120        std::vector<std::string> common_atom_names;
121        core::algorithms::intersect_sorted(native_names.begin(), native_names.end(),
122        q_
123        names.begin(), q_names.end(),
124        common_atom_names);
125
126        // ----- get the two subset of atoms
127        q.clear();

```

(continues on next page)

(continued from previous page)

```

119     t.clear();
120     std::shared_ptr<std::vector<double>> errors = std::make_shared<std::vector<double>>
121     ();
122     for (const std::string &name: common_atom_names) {
123         t.push_back(*q_name_to_atom[name]);
124         q.push_back(*native_name_to_atom[name]);
125     }
126
127     double rms_val = rms.crmsd(q, t, q.size(), true);
128     rms.calculate_crmsd_value(q, t, q.size(), errors);
129     // ----- This is the moment when we can dump the transformed structure into_
130     // a PDB file
131     if(output_pdb.was_used()) {
132         int iatm = -1;
133         std::string fname;
134         if(input_pdb_list.was_used())
135             fname = option_value<std::string>(output_name_prefix)+utils::split(structure_
136             </ids[i], {'/'})</>.back() + ".pdb";
137         else fname = option_value<std::string>(output_pdb);
138         std::ofstream out(fname);
139         for (const std::string &name: common_atom_names) {
140             q_name_to_atom[name]->b_factor((*errors)[++iatm]);
141             out << q_name_to_atom[name]->to_pdb_line() << "\n";
142             std::cout << name << " " << (*errors)[++iatm] << "\n";
143         }
144         std::cout << native->code() << " " << i << " crmsd: " << rms_val << "\n";
145     }
146 }
```



ap_docking_crmsd

ap_docking_crmsd calculates crmsd between ligand positions after flexible docking to a receptor. The program reads in a native pose and at least one PDB file with a computed pose (i.e. a model), each of them must contain a ligand molecule bound to a protein receptor. The ligand can be a small molecule, peptide or even a protein. The program finds a small-molecule ligand by residue ID (a three-letter code, such as CAM) Peptide ligands (or proteins) are identified by a chain ID (a single letter code, such as X). If the reference structure is not given and dash ‘-’ character is used instead (as in the last example), the program evaluates pairwise all-vs-all crMSD calculations. The output provides:

- ligand name (and possibly model ID) - crmsd on receptor
- no. of atoms of a receptor - crmsd on a ligand
- no. of atoms of a ligand

USAGE:

```
./ap_docking_crmsd reference.pdb ligand_def model.pdb [model2.pdb ...]
```

SEE ALSO: ap_ligand_clustering - for clustering of ligand docking poses ap_stiff_docking_crmsd - for a rigid docking crmsd calculations

EXAMPLEs:

```
./ap_docking_crmsd 2m56-ref.pdb CAM 00199.pdb 00963.pdb 04473.pdb
./ap_docking_crmsd 2kwi-1.pdb B 2kwi.pdb
./ap_docking_crmsd - B 2kwi.pdb
```

where 2m56-ref.pdb is the native and CAM is the three-letter PDB code of the ligand for which crmsd will be evaluated and 00199.pdb and the two other files are conformation after docking. In the second and third examples, B is the ID of the chain containing a ligand peptide.

Keywords:

- *PDB input*
- *crmsd*
- *docking*
- *structure selectors*

Categories:

- core::protocols::PairwiseCrmsd

Input files:

- 04473.pdb
- 2kwi.pdb
- 2m56-ref.pdb
- 00199.pdb
- 00963.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <map>
3
4 #include <core/data/io/Pdb.hh>
5 #include <core/protocols/PairwiseCrmsd.hh>
6 #include <core/data/structural/selectors/structure_selectors.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 ap_docking_crmsd calculates crmsd between ligand positions after flexible docking to_
12 ↪ a receptor.
13
14 The program reads in a native pose and at least one PDB file with a computed pose (i.
15 ↪. e. a model), each of them must
16 contain a ligand molecule bound to a protein receptor. The ligand can be a small_
17 ↪molecule, peptide or even a protein.
18
19 The program finds a small-molecule ligand by residue ID (a three-letter code, such as_
20 ↪CAM) Peptide ligands
21 (or proteins) are identified by a chain ID (a single letter code, such as X). If the_
22 ↪reference structure is not given
23 and dash '--' character is used instead (as in the last example), the program_
24 ↪evaluates pairwise
25 all-vs-all cRMSD calculations. The output provides:
26   - ligand name (and possibly model ID)
27   - crmsd on receptor
28   - no. of atoms of a receptor
29   - crmsd on a ligand
30   - no. of atoms of a ligand
31
32 USAGE:
33 ./ap_docking_crmsd reference.pdb ligand_def model.pdb [model2.pdb ...]
34
35 SEE ALSO:
36   ap_ligand_clustering - for clustering of ligand docking poses
37   ap_stiff_docking_crmsd - for a rigid docking crmsd calculations
38
39 EXAMPLES:
40 ./ap_docking_crmsd 2m56-ref.pdb CAM 00199.pdb 00963.pdb 04473.pdb
41 ./ap_docking_crmsd 2kwi-1.pdb B 2kwi.pdb
42 ./ap_docking_crmsd - B 2kwi.pdb
43
44 where 2m56-ref.pdb is the native and CAM is the three-letter PDB code of the ligand_
45 ↪for which crmsd will be evaluated
46 and 00199.pdb and the two other files are conformation after docking. In the second_
47 ↪and third examples,
48 B is the ID of the chain containing a ligand peptide.
49
50 )";
51
52 using namespace core::data::structural;
53
54 /**
55  * @brief ap_docking_crmsd calculates crmsd between two ligand positions after_
56 ↪docking to a receptor.
```

(continues on next page)

(continued from previous page)

```

47
48 * CATEGORIES: core::protocols::PairwiseCrmsd
49 * KEYWORDS: PDB input; crmsd; docking; structure selectors
50 * GROUP: Structure calculations; Docking;
51 */
52 int main(const int argc, const char* argv[]) {
53
54     using namespace core::data::structural::selectors;
55
56     if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about_
57     ↪missing program parameter
58
59     const std::string code(argv[2]);    // --- The ligand code is the second parameter_
60     ↪of the program
61     AtomSelector_SP select_ligand = nullptr; // --- Ligand selector object
62     if (code.size() == 3) // --- If the code is 3 characters long, its a residue code
63         select_ligand = std::static_pointer_cast<AtomSelector>(std::make_shared
64         ↪<SelectResidueByName>(code));
65     else {
66         AtomSelector_SP select_chain = std::static_pointer_cast<AtomSelector>(std::make_
67         ↪shared<ChainSelector>(code[0]));
68         std::shared_ptr<LogicalANDSelector> select_chain_ca = std::make_shared
69         ↪<LogicalANDSelector>();
70         select_chain_ca->add_selector( std::make_shared<IsCA>() );
71         select_chain_ca->add_selector(select_chain);
72         select_ligand = select_chain_ca;
73     }
74
75     std::shared_ptr<LogicalANDSelector> select_receptor = std::make_shared
76     ↪<LogicalANDSelector>(); // --- Receptor selector object
77     AtomSelector_SP select_not_ligand = std::static_pointer_cast<AtomSelector>
78     ↪(std::make_shared<InverseAtomSelector>(*select_ligand));
79     select_receptor->add_selector(std::make_shared<IsCA>());
80     select_receptor->add_selector(select_not_ligand);
81
82     core::protocols::PairwiseCrmsd crmsd_protocol(select_receptor, select_ligand);
83
84     for(core::index2 i=3;i<argc;++i) {
85         core::data::io::Pdb reader(argv[i], core::data::io::is_not_hydrogen);
86         if (reader.count_models()>1) {
87             for (core::index2 j = 0; j < reader.count_models(); ++j)
88                 crmsd_protocol.add_input_structure(reader.create_structure(j),_
89                 ↪utils::string_format("%s:%4d",argv[i], j));
90             } else
91                 crmsd_protocol.add_input_structure(reader.create_structure(0), argv[i]);
92         }
93
94         crmsd_protocol.crmsd_cutoff(50.0); // crmsd cutoff large enough to get some output
95         crmsd_protocol.output_stream( std::shared_ptr<std::ostream>(&std::cout, [](void*) { } )
96         ↪ );
97         if(std::string(argv[1]) != "-") {
98             core::data::io::Pdb reader(argv[1], core::data::io::is_not_hydrogen);
99             Structure_SP native = reader.create_structure(0);
100             crmsd_protocol.calculate(native);
101         } else crmsd_protocol.calculate();
102     }

```



ap_download_pdb

Simple app downloads a requested pdb file from RCSB website; it expects a four-letter PDB code of the deposit

USAGE:

```
ap_download_pdb PDB_code
```

EXAMPLE:

```
ap_download_pdb 2gb1
```

Keywords:

- PDB file
- download

Categories:

- utils/read_properties_file

Output files:

- 2gb1.pdb
- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <utils/exit.hh>
4 #include <utils/io_utils.hh>
5
6 std::string program_info = R"(
7
8 Simple app downloads a requested pdb file from RCSB website; it expects a four-letter PDB code of the deposit
9 USAGE:
10     ap_download_pdb PDB_code
11 EXAMPLE:
12     ap_download_pdb 2gb1
13 )
14 ;
15
16 /**
17 * @brief Simple app downloads a pdb file from RCSB website
18 *
19 * CATEGORIES: utils/read_properties_file
20 * KEYWORDS: PDB file; download
21 * GROUP: File processing
22 */
23 int main(const int argc, const char* argv[]) {

```

(continues on next page)

(continued from previous page)

```
24  if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
25  ↵missing program parameter
26
27  std::ofstream out(std::string(argv[1])+".pdb");
28  out << utils::download_pdb(argv[1]);
29  out.close();
{}
```



ap_dssp

Detects secondary structure using BioShell's implementation of the DSSP algorithm.

USAGE:

```
ap_dssp input.pdb
```

EXAMPLE:

```
ap_dssp 5edw.pdb
```

REFERENCE: Kabsch, Wolfgang, and Christian Sander. "Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features." *Biopolymers* 22 (1983): 2577-2637. doi:10.1002/bip.360221211

Keywords:

- *PDB input*
- *Hydrogen bonds*
- *secondary structure*
- *DSSP*
- *Protein structure features*

Categories:

- core::calc::structural::ProteinArchitecture

Input files:

- 2gb1.pdb
- 5edw.pdb
- 3dcg.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <core/calc/structural/ProteinArchitecture.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Detects secondary structure using BioShell's implementation of the DSSP algorithm.
```

(continues on next page)

(continued from previous page)

```

10 USAGE:
11     ap_dssp input.pdb
12
13 EXAMPLE:
14     ap_dssp 5edw.pdb
15
16 REFERENCE:
17 Kabsch, Wolfgang, and Christian Sander. "Dictionary of protein secondary structure:  

18     ↪pattern recognition  

19     of hydrogen-bonded and geometrical features." Biopolymers 22 (1983): 2577-2637.  

20     ↪doi:10.1002/bip.360221211
21 ) ";
22 /**
23 * @brief DSSP implementation
24 *
25 * CATEGORIES: core::calc::structural::ProteinArchitecture;
26 * KEYWORDS: PDB input; Hydrogen bonds; secondary structure; DSSP; Protein,  

27     ↪structure features
28 * GROUP: Structure calculations;
29 */
30 int main(const int argc, const char* argv[]) {
31
32     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about  

33     ↪missing program parameter
34
35     core::data::io::Pdb reader(argv[1]); // file name (PDB format, may be gzip-ped)
36     core::data::structural::Structure_SP strctr = reader.create_structure(0);
37
38     core::calc::structural::ProteinArchitecture pa(*strctr);
39     std::cout << pa.hec_string() << "\n";
40 }
```



ap_dssp_to_ss2

Reads a DSSP file and writes secondary structure in SS2 format. To convert DSSP to FASTA format use ap_DsspData
EXAMPLE:

```
ap_dssp_to_ss2 5edw.dssp
```

Keywords:

- *DSSP*
- *Structure*
- *secondary structure*
- *Format conversion*

Categories:

- core::data::io::DsspData

Input files:

- 5edw.dssp

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <core/data/io/ss2_io.hh>
3 #include <core/data/io/DsspData.hh>
4 #include <utils/exit.hh>
5
6 std::string program_info = R"(
7
8 Reads a DSSP file and writes secondary structure in SS2 format.
9 To convert DSSP to FASTA format use ap_DsspData
10
11 EXAMPLE:
12     ap_dssp_to_ss2 5edw.dssp
13
14 )";
15
16 /**
17  * @brief Reads a DSSP file and prints the secondary structure of each chain in SS2_
18  * format.
19  *
20  * @see ap_DsspData.cc converts DSSP to FASTA format
21  * @CATEGORIES: core::data::io::DsspData
22  * @KEYWORDS: DSSP; Structure; secondary structure; Format conversion

```

(continues on next page)

(continued from previous page)

```
21 * GROUP:      File processing; Format conversion
22 */
23 int main(const int argc, const char* argv[]) {
24
25     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
26     // missing program parameter
27
28     // --- read a DSSP file - the first command line argument of the program
29     core::data::io::DsspData dssp(argv[1], true);
30     for (const auto & ss2 : dssp.sequences())           // --- for each protein
31     // sequence found in the DSSP data ...
32         core::data::io::write_ss2(*ss2, std::cout);      // --- print it as SS2!
33 }
```



ap_filter_fasta

ap_find_in_fasta reads a file in FASTA format and prints only these sequences which satisfy the following filters:
- sequence must a protein
- sequence must not be shorter than 20 aa
- sequence must contain at most 10 UNK residues
The output sequences are sorted.

USAGE:

```
ap_filter_fasta input.fasta [input2.fasta ...]
```

EXAMPLE:

```
ap_filter_fasta ferrodoxins.fasta
```

Keywords:

- *FASTA input*
- *FASTA output*
- *sequence*
- *FASTA*
- *pre-processing*
- sequence filters

Categories:

- core/data/io/fasta_io

Input files:

- ferrodoxins.fasta
- 5edw.fasta

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/algorithms/UnionFind.hh>
4 #include <core/data/io/fasta_io.hh>
5 #include <utils/exit.hh>
6 #include <utils/io_utils.hh>
7
8 std::string program_info = R"(
9
10 ap_find_in_fasta reads a file in FASTA format and prints only these sequences which_
11 <> satisfy the following filters:
12 )"
```

(continues on next page)

(continued from previous page)

```

11 - sequence must a protein
12 - sequence must not be shorter than 20 aa
13 - sequence must contain at most 10 UNK residues
14 The output sequences are sorted.
15
16 USAGE:
17     ap_filter_fasta input.fasta [input2.fasta ...]
18 EXAMPLE:
19     ap_filter_fasta ferrodoxins.fasta
20
21 ) ";
22
23 /** @brief This program reads a file with sequences in FASTA format and sorts them by_
24 length.
25 * DNA sequences, sequences that are shorter than 15 residues and those having more_
26 than 10 Xs are removed
27 *
28 * CATEGORIES: core/data/io/fasta_io;
29 * KEYWORDS: FASTA input; FASTA output; sequence; FASTA; pre-processing; sequence_
30 filters
31 * GROUP: File processing;Data filtering
32 */
33
34 int main(const int argc, const char* argv[]) {
35
36     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
37     //missing program parameter
38
39     // --- Sequence_SP is just a std::shared_ptr to core::data::sequence::Sequence type
40     using core::data::sequence::Sequence_SP;
41
42     // --- Create a container where the sequences will be stored
43     std::vector<Sequence_SP> sequences;
44
45     // --- Read a file (or files) with FASTA sequences; sequences are appended to the_
46     //given vector
47     for (int i = 1; i < argc; ++i) core::data::io::read_fasta_file(argv[i], sequences);
48
49     // --- Remove sequences that do not come from proteins
50     sequences.erase(std::remove_if(sequences.begin(), sequences.end(),
51         [] (const Sequence_SP s){ return !s->is_protein_sequence; }), sequences.end());
52
53     // --- Remove sequences that are too short
54     sequences.erase(std::remove_if(sequences.begin(), sequences.end(),
55         [] (const Sequence_SP s){ return s->length()<20; }), sequences.end());
56
57     // --- Remove sequences that contain 10 or more 'X' characters (i.e. unknown amino_
58     //acids)
59     sequences.erase(std::remove_if(sequences.begin(), sequences.end(),
60         [] (const Sequence_SP s){ return std::count(s->sequence.begin(), s->sequence.
61             end(), 'X')>10; }), sequences.end());
62
63     // --- Now, sort the sequences by length
64     std::sort(sequences.begin(), sequences.end(),
65         [] (const Sequence_SP si,const Sequence_SP sj){ return si->length()<sj->length();
66         });
67
68     // --- Remove duplicates

```

(continues on next page)

(continued from previous page)

```
60 core:::algorithms::UnionFind<Sequence_SP, core:::index4> uf;
61 uf.add_element(sequences[0]);
62 for (size_t i = 1; i < sequences.size(); ++i) {
63     uf.add_element(sequences[i]);
64     for (int j = i - 1; j >= 0; --j) {
65         if (sequences[i]->length() - sequences[j]->length() > 20) break;
66         if (sequences[i]->sequence.find(sequences[j]->sequence) != std::string::npos) ↴
67             uf.union_set(i, j);
68     }
69     for (size_t i = 0; i < sequences.size(); ++i) {
70         core:::index4 set_id = uf.find_set(i);
71         if (set_id != i) {
72             std::string new_header = sequences[set_id]->header() + " " + sequences[i]->header();
73             sequences[set_id]->header(new_header);
74         }
75     }
76
77 // --- Print sequences to stdout
78 for (size_t i = 0; i < sequences.size(); ++i) {
79     if (uf.find_set(i) == i) std::cout << core:::data::io::create_fasta_
80             string(*sequences[i]) << "\n";
81 }
```



ap_filter_msa

Reads a Multiple Sequence Alignment (MSA) in ClustalW or FASTA format and removes these sequences who produce highly gapped columns. This filter first identifies Highly Gapped Columns (HGCs) as those MSA columns that have at most HG-fraction*N_SEQ letters and all remaining characters are gaps. Then each sequence that participates in at least sum_gap gapped columns is removed

USAGE:

```
./ap_filter_msa msa-file HG-fraction sum_gap
```

EXAMPLE:

```
./ap_filter_msa cyped.CYP109.aln 0.01 10
```

where cyped.CYP109.aln is the name of input MSA file; 0.01 means that in gapped columns 99% of sequences have a gap and 1% has a letter. Finally, 10 means that sequences that participate in at least 10 HGP are removed from the input MSA

Keywords:

- *clustal input*
- *MSA*
- *FASTA input*

Categories:

- core::alignment::MSAColumnConservation

Input files:

- conservation_test_msa.aln
- CYP109B1.aln

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/alignment/FilterByHighlyGappedColumns.hh>
4 #include <core/data/io/clustalw_io.hh>
5 #include <utils/exit.hh>
6 #include <utils/io_utils.hh>
7 #include <core/data/io/fasta_io.hh>
8
9
10 std::string program_info = R" (
```

(continues on next page)

(continued from previous page)

```

11 Reads a Multiple Sequence Alignment (MSA) in ClustalW or FASTA format and removes_
12 ↪these sequences who produce
13 highly gapped columns. This filter first identifies Highly Gapped Columns (HGCs) as_
14 ↪those MSA columns that have
15 at most HG-fraction*N_SEQ letters and all remaining characters are gaps. Then each_
16 ↪sequence that participates
17 in at least sum_gap gapped columns is removed
18
19
20 USAGE:
21 ./ap_filter_msa msa-file HG-fraction sum_gap
22
23 EXAMPLE:
24 ./ap_filter_msa cyped.CYP109.aln 0.01 10
25
26 where cyped.CYP109.aln is the name of input MSA file; 0.01 means that in gapped_
27 ↪columns 99% of sequences have a gap
28 and 1% has a letter. Finally, 10 means that sequences that participate in at least 10_
29 ↪HGP are removed from the
30 input MSA
31
32 ) ";
33
34 /**
35 * @brief Reads a MSA in ClustalW format and removes these sequences who produce
36 * highly gapped columns
37 *
38 * CATEGORIES: core::alignment::MSAColumnConservation
39 * KEYWORDS: clustal input; MSA; FASTA input
40 * GROUP: Alignments
41 */
42
43 int main(const int argc, const char* argv[]) {
44
45     if(argc < 4) utils::exit_OK_with_message(program_info); // --- complain about_
46     ↪missing program parameter
47
48     using namespace core::data::io;
49     using namespace core::data::sequence;
50
51     double f_nnc = atof(argv[2]);
52     core::index2 sum_gap = atoi(argv[3]);
53
54     std::vector<Sequence_SP> msa;    // --- Sequence_SP is just a shorter name for_
55     ↪std::shared_ptr<Sequence>
56     const std::pair<std::string, std::string> name_ext = utils::root_extension(argv[1]);
57     if((name_ext.second=="fasta") || (name_ext.second=="FASTA") || (name_ext.second=="fast
58     "))
59         core::data::io::read_fasta_file(argv[1], msa, true);
60     else
61         core::data::io::read_clustalw_file(argv[1], msa);
62
63     core::alignment::FilterByHighlyGappedColumns filter{msa};
64     filter.f_non_gapped(f_nnc);
65     core::index2 n_seq = msa.size();
66     filter.run(sum_gap);
67     std::cout << "#" << filter.msa().size() << " sequences remained, " << (n_seq -
68     ↪filter.msa().size()) << " removed\n";
69     std::cout << "#" << filter.highly_gapped_positions().size() << " highly gapped_
70     ↪positions found\n";

```

(continues on next page)

(continued from previous page)

```
59   for (const core:::data::sequence::Sequence_SP &seq:filter.msa())
60     std::cout << core:::data::io::create_fasta_string(*seq);
61   int i_seq = -1;
62   for (core:::index2 cnt:filter.highly_gapped_for_sequence())
63     std::cout << "# " << (++i_seq) << " " << cnt << "\n";
64 }
```



ap_filter_pdb

Shows the concept of PDB line filters in BioShell: creates a PDB reader which accepts only desired atoms/groups. The filter used by this example to read PDB file is created based on filter names (space separated). For each string a distinct filter will be created; all filters will be joined with logical AND operation i.e. all must be true to read in a PDB line. Therefore the last example below will return an empty set of atoms because the two filters it uses are contradictory.

USAGE:

```
ex_filter_pdb 5edw.pdb filter-names
```

EXAMPLES:

```
ex_filter_pdb 5edw.pdb is_standard_atom  
ex_filter_pdb 5edw.pdb is_bb is_cb
```

Keywords:

- *PDB input*
- *PDB line filter*

Categories:

- core::data::io::Pdb

Input files:

- 2gb1.pdb
- 5edw.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>  
2 #include <core/data/io/Pdb.hh>  
3 #include <utils/exit.hh>  
4  
5 std::string program_info = R"(  
6  
7 Shows the concept of PDB line filters in BioShell: creates a PDB reader which accepts  
8 ↳only desired atoms/groups.  
9 The filter used by this example to read PDB file is created based on filter names  
10 ↳(space separated). For each string  
11 a distinct filter will be created; all filters will be joined with logical AND  
12 ↳operation i.e. all must be true  
13 to read in a PDB line. Therefore the last example below will return an empty set of  
14 ↳atoms because the two filters
```

(continues on next page)

(continued from previous page)

```

11 it uses are contradictory.
12
13 USAGE:
14   ex_filter_pdb 5edw.pdb filter-names
15
16 EXAMPLES:
17   ex_filter_pdb 5edw.pdb is_standard_atom
18   ex_filter_pdb 5edw.pdb is_bb is_cb
19
20 ) ";
21
22 /** @brief Filters a PDB file by a given filter
23 *
24 * CATEGORIES: core::data::io::Pdb;
25 * KEYWORDS: PDB input; PDB line filter
26 */
27 int main(const int argc, const char* argv[]) {
28
29   if(argc < 3) {
30     program_info += "\nKnown filters:\n";
31     for (const std::string &name: core::data::io::Pdb::pdb_filter_names)
32       program_info += "\t" + name;
33     utils::exit_OK_with_message(program_info); // --- complain about missing program_
34   } //parameter
35   std::string filter_names(argv[2]);
36   for (int i = 3; i < argc; ++i) filter_names += " " + std::string(argv[i]);
37   core::data::io::Pdb reader(argv[1], // file name (PDB format, may be gzip-ped)
38                             filter_names, // filter names combined into a single_
39   string //string
40                             false); // don't parse header to achieve highest speed
41
42   for (int im = 0; im < reader.count_models(); ++im)
43     for (const core::data::io::Atom &pdb_line : (*reader.atoms[im]))
44       std::cout << pdb_line.to_pdb_line() << "\n";
45 }
```



ap_find_in_fasta

Program reads a sequence database in FASTA format and a text file with sequence identifiers, and prints the requested sequences on the screen.

USAGE:

```
ap_find_in_fasta input.fasta seq_id_list.txt
```

EXAMPLE:

```
ap_find_in_fasta uniref90.fasta seq_id_list.txt
ap_find_in_fasta ferrodoxins.fasta selected_list.txt
```

Keywords:

- *FASTA input*
- *FASTA output*
- *sequence*
- *FASTA*
- *pre-processing*

Categories:

- core/data/io/fasta_io.hh

Input files:

- ferrodoxins.fasta
- seq_id_list.txt

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/data/io/fasta_io.hh>
4 #include <utils/exit.hh>
5 #include <utils/io_utils.hh>
6
7 std::string program_info = R"(
8
9 Program reads a sequence database in FASTA format and a text file with sequence_
10 ↵identifiers, and prints
the requested sequences on the screen.

```

(continues on next page)

(continued from previous page)

```

11
12 USAGE:
13     ap_find_in_fasta input.fasta seq_id_list.txt
14 EXAMPLE:
15     ap_find_in_fasta uniref90.fasta seq_id_list.txt
16     ap_find_in_fasta ferrodoxins.fasta selected_list.txt
17
18 ) ";
19
20 bool is_good_sequence(const core::data::sequence::Sequence_SP seq, const std::vector<std::string> & wanted_seq_id) {
21
22     for(const std::string & s : wanted_seq_id) if(seq->header().find(s) != std::string::npos) return true;
23
24     return false;
25 }
26
27
28 /** @brief ap_find_in_fasta reads a sequence database in FASTA format and looks for sequences by given IDs
29 *
30 * CATEGORIES: core/data/io/fasta_io.hh;
31 * KEYWORDS: FASTA input; FASTA output; sequence; FASTA; pre-processing
32 * GROUP: File processing; Data filtering
33 */
34 int main(const int argc, const char *argv[]) {
35
36     if (argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
37
38     using core::data::sequence::Sequence_SP; // --- Sequence_SP is just a std::shared_ptr to core::data::sequence::Sequence type
39     using namespace core::data::io; // --- for FASTA I/O
40
41     std::vector<std::string> wanted_seq_id;
42     utils::read_listfile(argv[2], wanted_seq_id);
43     std::vector<core::data::sequence::Sequence_SP> sink;
44
45     // --- Read a file with FASTA sequences
46     core::data::sequence::Sequence_SP seq = nullptr;
47     std::ifstream infile;
48     utils::in_stream(argv[1], infile);
49     size_t n = 0;
50     infile >> seq;
51     while (seq != nullptr) {
52         if (is_good_sequence(seq, wanted_seq_id)) std::cout << create_fasta_string(*seq) << '\n';
53         if ((++n) % 10000 == 10000) std::cerr << n << " sequences tested\n";
54         infile >> seq;
55     }
56 }
```



ap_hhpred_converter

Reads alignments from HHpred output and prints then in Edinburgh, FASTA or PIR format, according to given flag.
The list of available flags: -e for Edinburgh output format -f for FASTA output format -p for PIR output format

USAGE:

```
ap_hhpred_converter hhpred-file flag
```

EXAMPLE:

```
ap_hhpred_converter hhpred.out -p
```

REFERENCE: Soding, J and Biegert, A and Lupas, A. N., “The HHpred interactive server for protein homology detection and structure prediction.” Nucleic acids research (2005) 33 W244–W248

Keywords:

- *sequence alignment*
- *FASTA*
- *PIR*
- Edinburgh

Categories:

- core:::data::io::read_hhpred

Input files:

- CYP51F.hhpred

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <core/data/io/alignment_io.hh>
3 #include <core/data/io/faasta_io.hh>
4 #include <core/data/io/pir_io.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Reads alignments from HHpred output and prints then in Edinburgh, FASTA or PIR format,
10 ↵ according to given flag.
11 The list of available flags:
12 -e for Edinburgh output format
```

(continues on next page)

(continued from previous page)

```

13 -f for FASTA output format
14 -p for PIR output format
15
16 USAGE:
17     ap_hhpred_converter hhpred-file flag
18 EXAMPLE:
19     ap_hhpred_converter hhpred.out -p
20
21 REFERENCE:
22 Soding, J and Biegert, A and Lupas, A. N.,
23 "The HHpred interactive server for protein homology detection and structure_
24 ↪prediction."
25 Nucleic acids research (2005) 33 W244--W248
26 )
27 /**
28 * @brief Extract alignments from HHpred output
29 *
30 * CATEGORIES: core:::data:::io:::read_hhpred;
31 * KEYWORDS: sequence alignment; FASTA; PIR; Edinburgh
32 * GROUP: File processing; Format conversion
33 */
34 int main(const int argc, const char* argv[]) {
35
36     if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about_
37 ↪missing program parameter
38
39     std::vector<core:::alignment::PairwiseSequenceAlignment_SP> alignments;
40     core:::data:::io:::read_hhpred(argv[1], alignments);
41
42     char flag = argv[2][1]; // --- E, e, F, f, P, p
43     switch(flag) {
44         case 'E' :
45         case 'e' :
46             for(const auto & seq_ali : alignments)
47                 core:::data:::io:::write_edinburgh(*seq_ali, std::cout, 80);
48             break;
49         case 'F' :
50         case 'f' :
51             for(const auto & seq_ali : alignments)
52                 std::cout << core:::data:::io:::create_fasta_string(*seq_ali, 80) << "\n";
53             break;
54         case 'P' :
55         case 'p' :
56             for(const auto & seq_ali : alignments)
57                 std::cout << core:::data:::io:::create_pir_string(*seq_ali, 80) << "\n";
58             break;
59         default: std::cerr << "Incorrect output format requested!\n";
60     }
61 }
```



ap_ligand_interactions

Finds ligand - protein interactions in a given PDB file. Ligand code must also be provided. The program prints interactions between protein and ligand including stacking, hydrogen bonds and van der Waals interactions.

USAGE:

```
ap_ligand_interactions input.pdb ligand_code
```

EXAMPLE:

```
ap_ligand_interactions 5ldk.pdb ATP
```

Keywords:

- *PDB input*
- *PDB line filter*
- *interactions*

Categories:

- core::data::io::Pdb; core::calc::structural::interactions

Input files:

- 5ldk.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <numeric>
3 #include <core/data/structural/selectors/structure_selectors.hh>
4 #include <core/data/io/Pdb.hh>
5 #include <core/calc/structural/interactions/StackingInteraction.hh>
6 #include <core/calc/structural/interactions/StackingInteractionCollector.hh>
7 #include <core/calc/structural/interactions/VdWInteractionCollector.hh>
8 #include <core/calc/structural/interactions/VdWInteraction.hh>
9 #include <core/calc/structural/interactions/HydrogenBondInteraction.hh>
10 #include <core/calc/structural/interactions/HydrogenBondCollector.hh>
11 #include <utils/LogManager.hh>
12
13 #include <utils/exit.hh>
14
15 std::string program_info = R"(
16 Finds ligand - protein interactions in a given PDB file. Ligand code must also be_
17 ↵provided

```

(continues on next page)

(continued from previous page)

```

18
19 The program prints interactions between protein and ligand including stacking, ↵
20 ↵hydrogen bonds and van der Waals interactions.
21
22 USAGE:
23     ap_ligand_interactions input.pdb ligand_code
24
25 EXAMPLE:
26     ap_ligand_interactions 5ldk.pdb ATP
27
28 ) ";
29
30 using namespace core::data::io; // --- Pdb reader and PdbLineFilter lives there
31 using namespace core::calc::structural::interactions;
32
33 /** @brief Finds stacking, hbonds and van der Waals interactions for ligand in a
34 given PDB file.
35
36 *
37 * CATEGORIES: core::data::io::Pdb; core::calc::structural::interactions
38 * KEYWORDS: PDB input; PDB line filter; interactions
39 * GROUP: Structure calculations;
40
41 int main(const int argc, const char *argv[]) {
42
43     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
44     ↵missing program parameter
45     utils::LogManager::FINER(); // --- INFO is the default logging level; set it to
46     ↵FINE to see more
47
48     // --- Read a PDB file given as an argument to this program
49     Pdb reader(argv[1], // --- input PDB file
50                 all_true(is_not_water, is_not_alternative, is_not_hydrogen,
51                         invert_filter(is_bb)), // --- Inverted backbone selector
52     ↵reads only side chains
53                 keep_all, true); // --- yes, read header
54
55     core::data::structural::Structure_SP s = reader.create_structure(0);
56     std::string code_3 = argv[2];
57
58     VdWInteractionCollector vdw_collector = VdWInteractionCollector();
59     HydrogenBondCollector hb_collector = HydrogenBondCollector();
60     StackingInteractionCollector stack_collector = StackingInteractionCollector();
61
62     std::vector<ResiduePair_SP> v_sink;
63     std::vector<ResiduePair_SP> hb_sink;
64     std::vector<ResiduePair_SP> s_sink;
65
66     hb_collector.collect(*s, hb_sink);
67     vdw_collector.collect(*s, v_sink);
68     stack_collector.collect(*s, s_sink);
69
70     std::cout << VdWInteraction::output_header() << "\n";
71     for (const ResiduePair_SP ri:v_sink) {
72         VdWInteraction_SP bi = std::dynamic_pointer_cast<VdWInteraction>(ri);
73         if (bi && bi->first_residue()->residue_type().code3.c_str() == code_3)
74             std::cout << *bi << "\n";

```

(continues on next page)

(continued from previous page)

```
69     }
70
71     std::cout << HydrogenBondInteraction::output_header() << "\n";
72     for (const ResiduePair_SP ri:hb_sink) {
73         HydrogenBondInteraction_SP bi = std::dynamic_pointer_cast
74             <HydrogenBondInteraction>(ri);
75         if (bi && bi->first_residue()->residue_type().code3.c_str() == code_3)
76             std::cout << *bi << "\n";
77     }
78
79     std::cout << StackingInteraction::output_header() << "\n";
80     for (const ResiduePair_SP ri:s_sink) {
81         StackingInteraction_SP bi = std::dynamic_pointer_cast<StackingInteraction>
82             (ri);
83         if (bi && bi->first_residue()->residue_type().code3.c_str() == code_3)
84             std::cout << *bi << "\n";
85     }
86 }
```



ap_ligand_trajectory

Finds contacts between a ligand molecule and a protein. Reads a multi-model PDB file and detects contacts in every model (e.g. a frame of an MD simulation). The output provides the interacting residues (name and residue ID) along with the number of observations for this contact. Requires PDB input file, three-letter ligand code and contact distance in Angstroms.

USAGE:

```
./ap_ligand_trajectory input.pdb ligand-code contact-distance
```

EXAMPLE:

```
./ap_ligand_trajectory test_inputs/2kwi.pdb GNP 3.5
```

Keywords:

- *PDB input*
- *contact map*
- *ligand*

Categories:

- core::data::io::Pdb

Input files:

- 2kwi.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <map>
3
4 #include <core/data/io/Pdb.hh>
5 #include <utils/string_utils.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Finds contacts between a ligand molecule and a protein.
11
12 Reads a multi-model PDB file and detects contacts in every model (e.g. a frame of an
13 ↵MD simulation).
13 The output provides the interacting residues (name and residue ID) along with the
13 ↵number of observations

```

(continues on next page)

(continued from previous page)

```

14 for this contact. Requires PDB input file, three-letter ligand code and contact_
→distance in Angstroms.
15
16 USAGE:
17     ./ap_ligand_trajectory input.pdb ligand-code contact-distance
18
19 EXAMPLE:
20     ./ap_ligand_trajectory test_inputs/2kwi.pdb GNP 3.5
21
22 );
23
24 /** @brief Finds contacts between a ligand molecule and a multimodel-protein.
25 *
26 * CATEGORIES: core::data::io::Pdb
27 * KEYWORDS: PDB input; contact map; ligand
28 * GROUP: Structure calculations;
29 */
30 int main(const int argc, const char* argv[]) {
31
32     if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about_
→missing program parameter
33
34     core::data::io::Pdb reader(argv[1]); // --- file name (PDB format, may be gzip-ped)
35
36     const std::string code(argv[2]); // --- The ligand code is the second parameter_
→of the program
37     double cutoff = utils::from_string<double>(argv[3]); // The third parameter is the_
→contact distance (in Angstroms)
38
39     std::map<std::string, int> results; // --- Map used to store results
40     for (size_t i = 0; i < reader.count_models(); ++i) { // --- Iterate over all models_
→in the input file
41         core::data::structural::Structure_SP strctr = reader.create_structure(i);
42
43         // --- Here we use a standard <code>find_if</code> algorithm to find the ligand_
→residue by its 3-letter code
44         auto ligand = std::find_if(strctr->first_residue(), strctr->last_residue(), [&
45             code](core::data::structural::Residue_SP res) {return (res->residue_type() .
46             code3==code);});
47         if(ligand== strctr->last_residue()) { // --- If no ligand - print a message and_
→take next structure
48             std::cerr << "Model " << i << " has no " << argv[2] << " residue\n";
49             continue;
50         }
51         for (auto it_resid = strctr->first_residue(); it_resid != strctr->last_residue();_
→++it_resid) {
52             double d = (*it_resid)->min_distance(*ligand);
53             if (d < cutoff) { // --- if this is close enough,
54                 std::string key = utils::string_format("%3s %4s %4d", (*it_resid)->residue_
→type().code3.c_str(),
55                 (*it_resid)->owner()->id().c_str(), (*it_resid)->id());
56                 if (results.find(key) == results.end()) results[key] = 1;
57                 else results[key]++;
58             }
59         }
60     }
61 }
```

(continues on next page)

(continued from previous page)

```
60 // --- print results
61 std::cout << "#resn chain resid counts\n";
62 for(const auto & p:results)
63     std::cout << p.first << " " << utils::string_format("%5d", p.second) << "\n";
64 }
```



ap_molecule_diffusion

ap_molecule_diffusion calculates average displacement of a small molecule as a function of time over a trajectory If a multi-model PDB file was given, the program prints contact count observed in all models

USAGE:

```
ap_molecule_diffusion trajectory.pdb HOH box_side
```

where trajectory.pdb is the input file multimodel-PDB file HOH is the PDB-id of molecules for which the displacement will be evaluated

Keywords:

- *PDB input*
- *simulation*

Categories:

- core::data::basic::Vec3Cubic

Input files:

- ar_tra.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/index.hh>
4 #include <core/data/io/Pdb.hh>
5 #include <core/data/basic/Vec3I.hh>
6 #include <core/calc/statistics/OnlineStatistics.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 ap_molecule_diffusion calculates average displacement of a small molecule as a_
12 ↵function of time over a trajectory
13
14 If a multi-model PDB file was given, the program prints contact count observed in all_
15 ↵models
16
17 USAGE:
18     ap_molecule_diffusion trajectory.pdb HOH box_side
19
20 where trajectory.pdb is the input file multimodel-PDB file HOH is the PDB-id of_
21 ↵molecules for which the displacement

```

(continues on next page)

(continued from previous page)

```

19 will be evaluated
20
21 )
22
23 /**
24 * @brief Calculates average displacement of a small molecule as a function of time
25 * over a trajectory
26 *
27 * CATEGORIES: core:::data:::basic:::Vec3Cubic
28 * KEYWORDS: PDB input; simulation
29 * GROUP: Structure calculations;
30 */
31
32 int main(const int argc, const char *argv[]) {
33
34     if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about
35     // missing program parameter
36
37     double L = utils::from_string<double>(argv[3]); // The third parameter is the box
38     // width (in Angstroms)
39     core:::data:::basic:::Vec3I:::set_box_len(L);
40     core:::data:::io:::Pdb reader(argv[1]); // --- file name (PDB format, may be gzip-ped)
41
42     core:::index4 n_atoms = reader.count_atoms();
43     core:::index4 n_frmes = reader.count_models();
44     core:::index4 t_max = n_frmes / 5;
45     std::vector<std::vector<core:::data:::basic:::Vec3I>> v;
46
47     // ----- Load coordinates to memory -----
48     for (int i_start = 0; i_start < n_frmes; ++i_start) {
49         std::vector<core:::data:::basic:::Vec3I> vi(n_atoms);
50         reader.fill_structure(i_start, vi);
51         v.push_back(vi);
52     }
53
54     // ----- Calculate displacement -----
55     std::vector<core:::calc:::statistics:::OnlineStatistics> avg(t_max);
56     for (size_t i_start = 0; i_start < n_frmes - t_max - 1; ++i_start) {
57         const std::vector<core:::data:::basic:::Vec3I> &v0 = v[i_start];
58         for (size_t i_t = 1; i_t <= t_max; ++i_t) {
59             const std::vector<core:::data:::basic:::Vec3I> &vi = v[i_start + i_t];
60             for (size_t i_atom = 0; i_atom < n_atoms; ++i_atom)
61                 avg[i_t - 1](sqrt(v0[i_atom].distance_square_to(vi[i_atom])));
62         }
63     }
64
65     for (size_t i_t = 1; i_t <= t_max; ++i_t) {
66         std::cout << utils::string_format("%5d %f %f\n", i_t, avg[i_t - 1].avg(),
67         //sqrt(avg[i_t - 1].var()));
68     }
69 }
```



ap_pdb_to_fasta_ss

Reads a PDB file and writes protein sequence(s) in FASTA format. The program also writes secondary structure in FASTA format, if this data is available from PDB headers. The sequence comprise only these amino acid residues which have C-alpha atom User can select a chain by providing its code as the second argument of the program. The program also writes a PDB file that corresponds to the sequence.

USAGE:

```
ap_pdb_to_fasta_ss input.pdb chain-code
```

EXAMPLE:

```
ap_pdb_to_fasta_ss 5edw.pdb A
```

OUTPUT: >2GB1 A MTYKLILNGKTLKGETTTEAVDAATAEKVFKQYANDNGVDGEWTYDDATKTFTVTE
>2GB1 A - secondary structure CEEEEEECCCCCCEEEEEECCHHHHHHHHHHHCCCCCEEEECCC-
CEEEECC

Keywords:

- *PDB input*
- *FASTA output*
- *secondary structure*
- predicates

Categories:

- core:::data::io::Pdb; core:::algorithms::Not; core:::data::sequence::SecondaryStructure

Input files:

- 2gb1.pdb
- 5edw.pdb
- 5d95.pdb

Output files:

- 5D95A.pdb
- 5EDWA.pdb
- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/algorithms/predicates.hh>
4 #include <core/data/io/Pdb.hh>
5 #include <core/data/io/fasta_io.hh>
6 #include <core/data/structural/selectors/structure_selectors.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 Reads a PDB file and writes protein sequence(s) in FASTA format.
12
13 The program also writes secondary structure in FASTA format, if this data is ↴
14 ↴available from PDB headers.
15 The sequence comprise only these amino acid residues which have C-alpha atom
16 User can select a chain by providing its code as the second argument of the program. ↴
17 ↴The program also writes a PDB file
18 that corresponds to the sequence.
19
20
21 USAGE:
22     ap_pdb_to_fasta_ss input.pdb chain-code
23
24
25 EXAMPLE:
26     ap_pdb_to_fasta_ss 5edw.pdb A
27
28
29 OUTPUT:
30     >2GB1 A
31     MTYKLILNGKTLKGTTTEAVDAATAEKVFKQYANDNGVDGEWTYDDATKTFTVTE
32
33     >2GB1 A - secondary structure
34     CEEEEEECCCCCCEEEEEECCHHHHHHHHHHHCCCCEEEECCCEEEEEC
35
36 )";
37
38 /**
39 * @brief Reads a PDB file and writes protein sequence(s) in FASTA format.
40 *
41 * The program also writes secondary structure in FASTA format, if this data is ↴
42 ↴available from PDB headers.
43 * User can select a chain by providing its code as the second argument of the program
44 * USAGE:
45 *     ap_pdb_to_fasta_ss 5edw.pdb A
46 *
47 * CATEGORIES: core:::data:::io:::Pdb; core:::algorithms:::Not; ↴
48 ↴core:::data:::sequence:::SecondaryStructure
49 * KEYWORDS: PDB input; FASTA output; secondary structure; predicates
50 * GROUP: File processing; Format conversion
51 */
52 int main(const int argc, const char* argv[]) {
53
54     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about ↴
55     ↴missing program parameter
56
57     using namespace core:::data:::io; // Pdb and create_fasta_string lives there
58     using namespace core:::data:::structural; // Chain and
59
60     Pdb reader(argv[1],is_not_alternative,core:::data:::io:::keep_all,true);
61     Structure_SP strctr = reader.create_structure(0);

```

(continues on next page)

(continued from previous page)

```
53
54 // Iterate over all chains
55 for (auto it_chain = strctr->begin(); it_chain!=strctr->end(); ++it_chain) {
56     Chain & c = **it_chain; // --- dereference iterator for easier access
57     if ((argc > 2) && ((*it_chain)->id() != argv[2])) continue;
58
59     // --- The line below uses STL algorithm with BioShell predicate to remove all
60     // the residues lacking c-alpha
61     c.erase(std::remove_if(c.begin(), c.end(), core::algorithms::Not
62     <selectors::ResidueHasCA>(selectors::ResidueHasCA())), c.end());
63
64     if(c.size()>0) {
65         // --- Create a sequence object (including secondary structure information)
66         core::data::sequence::SecondaryStructure_SP s = (*it_chain)->create_sequence();
67         // --- Write sequence as FASTA
68         std::cout << create_fasta_string(*s) << "\n";
69         // --- Write secondary structure as FASTA
70         std::cout << create_fasta_secondary_string(*s) << "\n";
71     }
72 }
```



ap_pdb_to_pir

Reads a PDB file and writes protein sequence(s) in PIR format

USAGE:

```
ap_pdb_to_pir 5edw.pdb
```

Keywords:

- *PDB input*
- *PIR*

Categories:

- core::data::io::PirEntry

Input files:

- 2kwi.pdb
- 2gb1.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <core/data/io/pir_io.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Reads a PDB file and writes protein sequence(s) in PIR format
10 USAGE:
11     ap_pdb_to_pir 5edw.pdb
12
13 )";
14
15 /** @brief Reads a PDB file and writes protein sequence(s) in PIR format
16 *
17 * CATEGORIES: core::data::io::PirEntry
18 * KEYWORDS: PDB input; PIR
19 * GROUP: File processing;Format conversion
20 */
21 int main(const int argc, const char* argv[]) {
22
```

(continues on next page)

(continued from previous page)

```
23  if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
→missing program parameter
24
25  using namespace core::data::io;
26  using namespace core::data::sequence;
27
28  core::data::io::Pdb reader(argv[1],is_not_alternative, only_ss_from_header, true);
29  core::data::structural::Structure_SP strctr = reader.create_structure(0);
30
31  // Iterate over all chains
32  for (auto it_chain = strctr->begin(); it_chain!=strctr->end(); ++it_chain) {
33      PirEntry e("",(*it_chain)->create_sequence()->sequence);
34      e.type(PirEntryType::STRUCTURE_X);
35      e.code(strctr->code());
36      e.first_residue_id((*it_chain)->front()->id());
37      e.first_chain_id((*it_chain)->char_id());
38      e.last_residue_id((*it_chain)->back()->id());
39      e.last_chain_id((*it_chain)->char_id());
40
41      std::cout << e<<"\n";
42  }
43 }
```



ap_pir_to_fasta

Reads a file with sequences in PIR format and converts them to FASTA.

USAGE:

```
ap_pir_to_fasta example.pir
```

REFERENCE: <https://salilab.org/modeller/9v8/manual/node454.html>

Keywords:

- *PIR*
- *FASTA output*

Categories:

- core/data/io/pir_io

Input files:

- az.pir
- example.pir

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/data/io/pir_io.hh>
4 #include <core/data/io/fasta_io.hh>
5
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Reads a file with sequences in PIR format and converts them to FASTA.
11
12 USAGE:
13     ap_pir_to_fasta example.pir
14
15 REFERENCE:
16     https://salilab.org/modeller/9v8/manual/node454.html
17
18 )";
19
20 /**
21  * @brief Reads a file with sequences in PIR format and converts them to FASTA.
22 */

```

(continues on next page)

(continued from previous page)

```
22 * CATEGORIES: core/data/io/pir_io;
23 * KEYWORDS: PIR; FASTA output
24 * GROUP: File processing;Format conversion
25 */
26 int main(const int argc, const char* argv[]) {
27
28     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
29     //missing program parameter
30
31     using core::data::sequence::Sequence_SP; // --- Sequence_SP is just a std::shared_
32     //ptr to core::data::sequence::Sequence type
33     using namespace core::data::io;           // --- This is required so PirEntry can be
34     //printed with << operator
35
36     // --- Create a container where the sequences will be stored
37     std::vector<Sequence_SP> sequences;
38
39     // --- Read a file with PIR sequences
40     core::data::io::read_pir_file(argv[1], sequences);
41
42     // --- Write them in FASTA
43     for (const Sequence_SP s : sequences)
44         std::cout << core::data::io::create_fasta_string(*s) << "\n";
45
46     // --- The sequence data is actually in FASTA format; just upper-casted to Sequence_
47     //SP
48     // --- Here we down-cast it back to the derived type
49     std::cout << "The source PIR data was:\n";
50     for (const Sequence_SP s : sequences)
51         std::cout << *std::dynamic_pointer_cast<core::data::sequence::PirEntry>(s) << "\n"
52     //";
53 }
```



ap_reordered_profile_columns

Reads a sequence profile (ASN.1 file format) and shuffles profile's columns as requested. Resulting profile is written in a tabular text format. If the new column order is not specified, amino acids will appear in the order: GAP VILMC HWFY KR QD NQST, i.e. small, aromatic, positive, negative and other-polar

USAGE:

```
./ap_reordered_profile_columns input asn1 [column-order]
```

EXAMPLE:

```
./ap_reordered_profile_columns d1or4A_.asn1
```

Keywords:

- output file
- *sequence profile*

Categories:

- core::data::sequence::SequenceProfile

Input files:

- **d1or4A_.asn1**

Output files:

- stdout.out
- d1or4A_reordered.txt

Program source:

```
1 #include <iostream>
2
3 #include <core/chemical/Monomer.hh>
4 #include <core/data/sequence/SequenceProfile.hh>
5 #include <utils/exit.hh>
6 #include <utils/LogManager.hh>
7 #include <utils/io_utils.hh>
8
9 std::string program_info = R"(
10
11 Reads a sequence profile (ASN.1 file format) and shuffles profile's columns as_
12 ↪requested.
13 Resulting profile is written in a tabular text format. If the new column order is not_
14 ↪specified, amino acids
15 will appear in the order: GAP VILMC HWFY KR QD NQST, i.e. small, aromatic, positive,_
16 ↪negative and other-polar
```

(continues on next page)

(continued from previous page)

```

14
15 USAGE:
16   ./ap_reorder_profile_columns input.asn1 [column-order]
17 EXAMPLE:
18   ./ap_reorder_profile_columns dlor4A_.asn1
19
20 ) ";
21
22 // small aromatic positive negative other-polar
23 const std::string nice_order = "GAP"  "VILMC"  "HWFY"  "KR"  "QD"  "NQST";
24
25 /** @brief Reads a sequence profile (ASN.1 file format) and shuffles profile's columns
26 *
27 * CATEGORIES: core::data::sequence::SequenceProfile
28 * KEYWORDS: output file; sequence profile
29 * GROUP: File processing
30 */
31 int main(const int argc, const char* argv[]) {
32
33   if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
34   ↵missing program parameter
35
36   using namespace core::data::io;
37   using namespace core::data::sequence;
38   utils::Logger logs("ap_reorder_profile_columns");
39   std::string order_string = (argc>2) ? argv[2] : nice_order;
40   logs << utils::LogLevel::INFO << "new aa order is: " << order_string << "\n";
41   SequenceProfile_SP profile_in = core::data::sequence::read ASN1_checkpoint(argv[1]);
42   SequenceProfile_SP profile_out = profile_in->create_reordered(order_string);
43   profile_out->write_table(std::cout);
44 }
```



ap_rescore_alignment

Reads sequence alignment(s) in the FASTA format and recalculates scores. The input file may contain more than two sequences, i.e. may provide a Multiple Sequence Alignment. Every pair of aligned sequences is rescored in this case. Output values are printed on the screen. The default scoring parameters are: BLOSUM62, -10, -1

USAGE:

```
./ap_rescore_alignment input.fasta [substitution-matrix [gap_open [gap_extend] ] ]
```

EXAMPLE:

```
./ap_rescore_alignment test_inputs/2azaA_2pcyA-ali.fasta
```

Keywords:

- *sequence alignment*
- *FASTA input*
- sequence alignment score

Categories:

- core/alignment/on_alignment_computations.cc

Input files:

- optimal_global.ali.fasta
- 2azaA_2pcyA-ali.fasta

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <utils/exit.hh>
3
4 #include <core/data/io/fasta_io.hh>
5 #include <core/alignment/PairwiseSequenceAlignment.hh>
6 #include <core/alignment/on_alignment_computations.hh>
7 #include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>
8
9 using namespace core::data::io;
10 using namespace core::alignment::scoring;
11 using namespace core::data::sequence;
12
13 std::string program_info = R"(
14
15 Reads sequence alignment(s) in the FASTA format and recalculates scores. The input_

```

(continues on next page)

(continued from previous page)

```

16 may contained more than two sequences, i.e. may provide a Multiple Sequence Alignment.
17 Every pair of aligned sequences is rescored in this case. Output values are printed ↵
18 →on the screen.
19 The default scoring parameters are: BLOSUM62, -10, -1
20
21 USAGE:
22 ./ap_rescore_alignment input.fasta [substitution-matrix [gap_open [gap_extend] ] ]
23
24 EXAMPLE:
25 ./ap_rescore_alignment test_inputs/2azaA_2pcyA-ali.fasta
26 )
27
28 /** @brief Estimates pairwise sequence similarity for a set of sequences given in a
29 →FASTA format
30 *
31 * CATEGORIES: core/alignment/on_alignment_computations.cc;
32 * KEYWORDS: sequence alignment; FASTA input; sequence alignment score
33 * GROUP: Alignments
34 */
35 int main(const int argc, const char *argv[]) {
36
37     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about ↵
38 →missing program parameter
39
40     // --- read input database fasta file
41     std::vector<Sequence_SP> ali.fasta; // --- stores sequences (should be already ↵
42 →aligned)
43     core::data::io::read_fasta_file(argv[1], ali.fasta);
44
45     std::string matrix_name = (argc>2) ? argv[2] : "BLOSUM62";
46     short gap_open = (argc > 3) ? atoi(argv[3]) : -10;
47     short gap_extend = (argc > 4) ? atoi(argv[4]) : -1;
48     NcbiSimilarityMatrix_SP sim_m = NcbiSimilarityMatrixFactory::get().get_
49 →matrix(matrix_name);
50
51     // --- prints both fasta sequences names and their recalculated score
52     for (size_t i = 1; i < ali.fasta.size(); ++i)
53         for (size_t j = 0; j < i; ++j)
54             std::cout << std::setw(10) << ali.fasta[i]->header().substr(0, 10) << " "
55             << std::setw(10) << ali.fasta[j]->header().substr(0, 10)
56             << " " << core::alignment::calculate_score(*ali.fasta[i], *ali_
57 →fasta[j], *sim_m, gap_open, gap_extend)
58             << "\n";
59 }
```



ap_sasa

Calculates Solvent Accessible Surface Area (SASA) for every atom in the input structure for the probe sphere with the given radius (in Angstroms) and number of dots (n_dots) used to approximate surface area. Resulting values will be stored as B-factor values in PDB file. Default probe radius is 1.6 Angstroms, the program uses 960 dots by default

USAGE:

```
./ap_sasa input.pdb probe-radius n-dots
```

EXAMPLE:

```
./ap_sasa 2gb1.pdb 1.6
```

REFERENCE: Lee, Byungkook, Frederic M. Richards. "The interpretation of protein structures: estimation of static accessibility." JMB 55 (1971): 379-IN4. doi:10.1016/0022-2836(71)90324-X

Shrake, A., J. A. Rupley. "Environment and exposure to solvent of protein atoms. Lysozyme and insulin." JMB 79(1973): 351-371. doi:10.1016/0022-2836(73)90011-9.

Keywords:

- *PDB input*
- *structural properties*

Categories:

- core::calc::structural::shrake_rupley_sasa

Input files:

- 2gb1.pdb
- 5edw.pdb

Output files:

- 5edw_sasa.pdb
- 2gb1_sasa.pdb
- stdout.out

Program source:

```
1 #include <iostream>
2 #include <algorithm>
3 #include <set>
4
5 #include <core/data/io/Pdb.hh>
6 #include <core/data/structural/selectors/structure_selectors.hh>
7 #include <core/calc/structural/sasa.hh>
```

(continues on next page)

(continued from previous page)

```

8 #include <utils/exit.hh>
9
10 std::string program_info = R"(
11
12 Calculates Solvent Accessible Surface Area (SASA) for every atom in the input_
13 ↵structure for the probe sphere
14 with the given radius (in Angstroms) and number of dots (n_dots) used to approximate_
15 ↵surface area.
16 Resulting values will be stored as B-factor values in PDB file.
17
18 Default probe radius is 1.6 Angstroms, the program uses 960 dots by default
19
20 USAGE:
21     ./ap_sasa input.pdb probe-radius n-dots
22
23 EXAMPLE:
24     ./ap_sasa 2gb1.pdb 1.6
25
26 REFERENCE:
27 Lee, Byungkook, Frederic M. Richards. "The interpretation of protein structures:_
28 ↵estimation of static accessibility."
29 JMB 55 (1971): 379-IN4. doi:10.1016/0022-2836(71)90324-X
30
31 Shrake, A., J. A. Rupley. "Environment and exposure to solvent of protein atoms._
32 ↵Lysozyme and insulin."
33 JMB 79(1973): 351-371. doi:10.1016/0022-2836(73)90011-9.
34
35 )";
36
37 /**
38 * @brief Calculates Solvent Accessible Surface Area for every atom in the input_
39 ↵structure
40 *
41 *
42 * CATEGORIES: core::calc::structural::shrake_rupley_sasa
43 * KEYWORDS: PDB input; structural properties
44 * GROUP: Structure calculations;
45 */
46 int main(const int argc, const char* argv[]) {
47
48     if (argc < 3) utils::exit_OK_with_message(program_info); // --- complain about_
49 ↵missing program parameter
50
51     using namespace core::data::io;
52     Pdb reader(argv[1], all_true(is_not_water,is_not_alternative)); // --- file name_
53 ↵(PDB format, may be gzip-ped)
54     double probe_radius = (argc > 2) ? atof(argv[2]) : 1.6;
55     int n_dots = (argc > 3) ? atoi(argv[3]) : 960;
56     using namespace core::data::structural;
57     Structure_SP strctr = reader.create_structure(0);
58     std::vector<double> sasa;
59     core::calc::structural::shrake_rupley_sasa(*strctr, sasa, probe_radius, n_dots);
60     int i = -1;
61     for (auto it_atom_i = strctr->first_const_atom(); it_atom_i != strctr->last_const_
62 ↵atom(); ++it_atom_i) {
63         (**it_atom_i).b_factor(sasa[++i]);

```

(continues on next page)

(continued from previous page)

```
57     std::cout << (**it_atom_i).to_pdb_line() << "\n";
58 }
59 }
```



ap_scorefile_columns

Reads a score file or a silent file (produced by Rosetta) and extracts requested columns of scores

USAGE:

```
ap_scorefile_columns default.out
ap_scorefile_columns score.fsc
ap_scorefile_columns 1pgxA-abinitio.fsc rms ss_pair rsigma
```

Keywords:

- *Rosetta scorefile*
- :ref:“

Categories:

- core::data::io::scorefile_io

Input files:

- 1pgxA-abinitio.fsc

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/scorefile_io.hh>
4 #include <utils/exit.hh>
5 #include <utils/Logger.hh>
6
7 std::string program_info = R"(
8
9 Reads a score file or a silent file (produced by Rosetta) and extracts requested_
10    ↪columns of scores
11 USAGE:
12     ap_scorefile_columns default.out
13     ap_scorefile_columns score.fsc
14     ap_scorefile_columns 1pgxA-abinitio.fsc rms ss_pair rsigma
15 )";
16
17 /** @brief Reads a score-file or a silent file (produced by Rosetta) and extracts_
18    ↪requested columns of scores
19 *
20 * CATEGORIES: core::data::io::scorefile_io
21 * KEYWORDS: Rosetta scorefile;
```

(continues on next page)

(continued from previous page)

```

21 * GROUP:      File processing;Data filtering
22 */
23 int main(const int argc, const char* argv[]) {
24
25     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
26     ↪missing program parameter
27
28     using namespace core::data::io;
29
30     utils::Logger logs("ap_scorefile_columns");
31
32     std::shared_ptr<NamedDataTable> fsc = read_scorefile(argv[1]);
33     std::vector<core::index1> columns;
34     if (argc > 2) {
35         for (core::index1 i = 2; i < argc; ++i)
36             if (fsc->has_column(argv[i]))
37                 columns.push_back(fsc->column_index(argv[i]));
38             else
39                 logs << utils::LogLevel::WARNING << "Unknown column ID: " << argv[i] << "\n";
40     } else {
41         columns.push_back(fsc->column_index("score"));
42         columns.push_back(fsc->column_index("rms"));
43     }
44     std::vector<std::string> tags;
45     std::cout << "#";
46     for (core::index1 icol : columns) std::cout << " " << fsc->column_name(icol) ;
47     std::cout << "\n";
48     for (const auto &row: *fsc) {
49         for (core::index1 icol : columns) std::cout << row[icol] << " ";
50         std::cout << "\n";
51     }
}

```



ap_stiff_docking_crmsd

Reads a PDB file with a ligand docked to a protein receptor and a native (reference) protein-ligand complex and calculates cRMSD (coordinate Root-Mean-Square Deviation) on a ligand molecule between the two conformations. The file with structural models may contain more than one conformation (multi-model PDB file). The program assumes that the receptor structure doesn't change significantly during docking (stiff or semi-flexible docking scenario) and superimposes all models on the first one, which significantly reduces calculation time. The ligand may be a small molecule compound, peptide or even a protein. The program evaluates cRMSD based solely on ligand coordinates. The program finds a small-molecule ligand by residue ID (a three-letter code, such as CAM) Peptide ligands (or proteins) are identified by a chain ID (a single letter code, such as X). If the reference structure is not given and dash '-' character is used instead (as in the last example), the program evaluates pairwise all-vs-all cRMSD calculations. The output provides: - ligand name (and possibly model ID) - crmsd on receptor (to confirm that is rigid) - no. of atoms of a receptor - crmsd on a ligand - no. of atoms of a ligand

USAGE:

```
./ap_stiff_docking_crmsd reference.pdb ligand_def model.pdb [model2.pdb ...]
```

SEE ALSO: ap_docking_crmsd - for a flexible docking analysis ap_ligand_clustering - for clustering of ligand docking poses

EXAMPLES:

```
./ap_stiff_docking_crmsd 2m56-ref.pdb CAM 00199.pdb 00963.pdb 04473.pdb
./ap_stiff_docking_crmsd 2kwi-1.pdb B 2kwi.pdb
./ap_stiff_docking_crmsd - B 2kwi.pdb
```

where 2m56-ref.pdb is the native and CAM is the three-letter PDB code of the ligand for which crmsd will be evaluated and 00199.pdb and the two other files are conformation after docking. In the second and third examples, B is the ID of the chain containing a ligand peptide.

Keywords:

- *PDB input*
- *crmsd*
- *docking*
- *structure selectors*

Categories:

- core::protocols::PairwiseLigandCrmsd

Input files:

- 04473.pdb
- 2kwi-1.pdb
- 2kwi.pdb
- 2m56-ref.pdb
- 00199.pdb
- 00963.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <map>
3
4 #include <core/data/io/Pdb.hh>
5 #include <core/calc/structural/transformations/Crmsd.hh>
6 #include <utils/exit.hh>
7 #include <core/data/structural/selectors/structure_selectors.hh>
8 #include <core/protocols/PairwiseLigandCrmsd.hh>
9
10 std::string program_info = R"(

11 Reads a PDB file with a ligand docked to a protein receptor and a native (reference) ↴
12 ↪protein-ligand complex
13 and calculates crMSD (coordinate Root-Mean-Square Deviation) on a ligand molecule ↴
14 ↪between the two conformations.
15 The file with structural models may contain more than one conformation (multi-model ↴
16 ↪PDB file).

17 The program assumes that the receptor structure doesn't change significantly during ↴
18 ↪docking
19 (stiff or semi-flexible docking scenario) and superimposes all models on the first ↴
20 ↪one, which significantly reduces
21 calculation time. The ligand may be a small molecule compound, peptide or even a ↴
22 ↪protein.
23 The program evaluates crMSD based solely on ligand coordinates.

24 The program finds a small-molecule ligand by residue ID (a three-letter code, such as ↴
25 ↪CAM) Peptide ligands
26 (or proteins) are identified by a chain ID (a single letter code, such as X). If the ↴
27 ↪reference structure is not given
28 and dash '-' character is used instead (as in the last example), the program ↴
29 ↪evaluates pairwise
30 all-vs-all crMSD calculations. The output provides:
31   - ligand name (and possibly model ID)
32   - crmsd on receptor (to confirm that is rigid)
33   - no. of atoms of a receptor
34   - crmsd on a ligand
35   - no. of atoms of a ligand

36 USAGE:
37 ./ap_stiff_docking_crmsd reference.pdb ligand_def model.pdb [model2.pdb ...]

38 SEE ALSO:
39   ap_docking_crmsd - for a flexible docking analysis
40   ap_ligand_clustering - for clustering of ligand docking poses

41 EXAMPLES:
42 ./ap_stiff_docking_crmsd 2m56-ref.pdb CAM 00199.pdb 00963.pdb 04473.pdb
43 ./ap_stiff_docking_crmsd 2kwi-1.pdb B 2kwi.pdb
44 ./ap_stiff_docking_crmsd - B 2kwi.pdb
```

(continues on next page)

(continued from previous page)

```

42
43 where 2m56-ref.pdb is the native and CAM is the three-letter PDB code of the ligand,
44 ↪for which crmsd will be evaluated
45 and 00199.pdb and the two other files are conformation after docking. In the second
46 ↪and third examples,
47 B is the ID of the chain containing a ligand peptide.
48
49 )";
50
51 using namespace core::data::structural;
52
53 /** @brief ap_peptide_docking_crmsd calculates crmsd of a peptide that is bound to a
receptor
54 * CATEGORIES: core::protocols::PairwiseLigandCrmsd
* KEYWORDS: PDB input; crmsd; docking; structure selectors
* GROUP: Structure calculations; Docking;
55 */
56
57 int main(const int argc, const char* argv[]) {
58
59     using namespace core::data::structural::selectors;
60
61     if (argc < 3) utils::exit_OK_with_message(program_info); // --- complain about
missing program parameter
62
63     const std::string code(argv[2]); // --- The ligand code is the second parameter
of the program
64     AtomSelector_SP select_ligand = nullptr; // --- Ligand selector object
65     if (code.size() == 3) // --- If the code is 3 characters long, its a residue code
66         select_ligand = std::static_pointer_cast<AtomSelector>(std::make_shared
<SelectResidueByName>(code));
67     else {
68         AtomSelector_SP select_chain = std::static_pointer_cast<AtomSelector>(std::make_
shared<ChainSelector>(code[0]));
69         std::shared_ptr<LogicalANDSelector> select_chain_ca = std::make_shared
<LogicalANDSelector>();
70         select_chain_ca->add_selector(std::make_shared<IsCA>());
71         select_chain_ca->add_selector(select_chain);
72         select_ligand = select_chain_ca;
73     }
74
75     std::shared_ptr<LogicalANDSelector> select_receptor = std::make_shared
<LogicalANDSelector>(); // --- Receptor selector object
76     AtomSelector_SP select_not_ligand = std::static_pointer_cast<AtomSelector>
(std::make_shared<InverseAtomSelector>(*select_ligand));
77     select_receptor->add_selector(std::make_shared<IsCA>());
78     select_receptor->add_selector(select_not_ligand);
79
80     core::protocols::PairwiseLigandCrmsd crmsd_protocol(select_ligand, select_receptor);
81
82     for(core::index2 i=3;i<argc;++i) {
83         core::data::io::Pdb reader(argv[i], core::data::io::is_not_hydrogen);
84         if (reader.count_models()>1) {
85             for (core::index2 j = 0; j < reader.count_models(); ++j)
86                 crmsd_protocol.add_input_structure(reader.create_structure(j),
utils::string_format("%s:%4d", argv[i], j));
87         } else

```

(continues on next page)

(continued from previous page)

```
88         crmsd_protocol.add_input_structure(reader.create_structure(0), argv[i]);
89     }
90
91     crmsd_protocol.crmsd_cutoff(20.0); // crmsd cutoff large enough to get some output
92     crmsd_protocol.output_stream( std::shared_ptr<std::ostream>(&std::cout, [](){}));
93     if(std::string(argv[1]) != "-") {
94         core::data::io::Pdb reader(argv[1], core::data::io::is_not_hydrogen);
95         Structure_SP native = reader.create_structure(0);
96         crmsd_protocol.calculate(native);
97     } else crmsd_protocol.calculate();
98
99 }
```



ap_superimpose_pdb_by_ca

Superimposes protein structures by matching C-alphas. All atoms of the second (and subsequent) protein structures will be superimposed on the first protein based on the CA positions. All structures must contain the same number of C-alphas atoms.

USAGE:

```
./ap_superimpose_pdb_by_ca reference pdb_file_1 [pdb_file_2 ...]
```

EXAMPLE:

```
./ap_superimpose_pdb_by_ligand 4rm4A.pdb model.pdb
```

Keywords:

- *PDB input*
- *rototranslation*
- superimposition
- *crmsd*
- *docking*

Categories:

- core/calc/structural/transformations/PairwiseCrmsd

Program source:

```
1 #include <memory>
2 #include <iostream>
3 #include <vector>
4
5 #include <core/data/io/Pdb.hh>
6 #include <core/data/structural/selectors/structure_selectors.hh>
7 #include <core/protocols/PairwiseCrmsd.hh>
8 #include <utils/LogManager.hh>
9
10 using namespace core::data::structural;
11
12 std::string program_info = R"(

13 Superimposes protein structures by matching C-alphas.

14 All atoms of the second (and subsequent) protein structures will be superimposed on
15 the first protein based on the CA
16 positions. All structures must contain the same number of C-alphas atoms.

17 USAGE:
18     ./ap_superimpose_pdb_by_ca reference pdb_file_1 [pdb_file_2 ...]

19 EXAMPLE:
20
21
22 )"
```

(continues on next page)

(continued from previous page)

```

23     . /ap_superimpose_pdb_by_ligand 4rm4A.pdb  model.pdb
24 ) ";
25
26 /** @brief Superimposes protein structures by matching ligand molecules.
27 * *
28 * CATEGORIES: core/calc/structural/transformations/PairwiseCrmsd
29 * KEYWORDS: PDB input; rototranslation; superimposition; crmsd; docking
30 * GROUP: Structure calculations;
31 */
32 int main(const int argc, const char *argv[]) {
33
34     if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about_
35     ↪missing program parameter
36
37     utils::LogManager::get().FINE();
38     selectors::AtomSelector_SP selector = std::make_shared<selectors::IsCA>(); // ---_
39     ↪select a ligand residue by its 3-letter code
40
41     core::data::io::Pdb read_native(argv[1], core::data::io::keep_all); // --- Read the_
42     ↪native (reference) structure, keep all atoms
43     Structure_SP native = read_native.create_structure(0);
44     std::vector<Structure_SP> models; // --- Container for targets to be superimposed
45     for (int i = 2; i < argc; ++i) {
46         core::data::io::Pdb reader(argv[i], core::data::io::keep_all);
47         for (int j = 0; j < reader.count_models(); ++j) // --- Read all models from each_
48         ↪target PDB file
49         models.push_back(reader.create_structure(j));
50     }
51
52     selectors::AtomSelector_SP select_all = std::make_shared<selectors::AtomSelector>();
53     core::protocols::PairwiseCrmsd rms_calc(models, selector);
54     std::shared_ptr<std::ostream> out = std::make_shared<std::ofstream>("out.pdb");
55     rms_calc.calculate(native, out);
56 }
```



ap_superimpose_pdb_by_ligand

Superimposes protein structures by matching ligand molecules. All the given protein structures must contain the same ligand molecule, every time in the same conformation. The program calculates a transformation (rotation-translation) that superimposes that ligand from input structures on the same ligand molecule found in the native PDB. The transformation is then used to rototranslate whole protein structures. Results is written to “out.pdb” file

USAGE:

```
./ap_superimpose_pdb_by_ligand native_pdb ligand_name pdb_file_1 [pdb_file_2 ...]
```

EXAMPLE:

```
./ap_superimpose_pdb_by_ligand 4rm4A.pdb HEM 5ofqA.pdb
```

Keywords:

- *PDB input*
- *rototranslation*
- superimposition
- *crmsd*
- *docking*

Categories:

- core/calc/structural/transformations/PairwiseCrmsd

Input files:

- 5ofq.pdb
- 4rm4.pdb

Output files:

- stdout.out
- out.pdb

Program source:

```

1 #include <memory>
2 #include <iostream>
3 #include <vector>
4
5 #include <core/data/io/Pdb.hh>
6 #include <core/data/structural/selectors/structure_selectors.hh>
7 #include <core/protocols/PairwiseCrmsd.hh>
8 #include <utils/LogManager.hh>
9

```

(continues on next page)

(continued from previous page)

```

10  using namespace core::data::structural;
11
12 std::string program_info = R"(
13
14 Superimposes protein structures by matching ligand molecules.
15
16 All the given protein structures must contain the same ligand molecule, every time in_
17 ↪the same conformation.
18 The program calculates a transformation (rotation-translation) that superimposes that_
19 ↪ligand from input structures
20 on the same ligand molecule found in the native PDB. The transformation is then used_
21 ↪to rototranslate whole protein
22 structures. Results is written to "out.pdb" file
23
24 USAGE:
25     ./ap_superimpose_pdb_by_ligand native_pdb ligand_name pdb_file_1 [pdb_file_2 ...]
26
27 EXAMPLE:
28     ./ap_superimpose_pdb_by_ligand 4rm4A.pdb HEM 5ofqA.pdb
29 )";
30
31 /**
32  * @brief Superimposes protein structures by matching ligand molecules.
33  * *
34  * @CATEGORIES: core/calc/structural/transformations/PairwiseCrmsd
35  * @KEYWORDS: PDB input; rototranslation; superimposition; crmsd; docking
36  * @GROUP: Structure calculations;
37  */
38 int main(const int argc, const char *argv[]) {
39
40     if(argc < 4) utils::exit_OK_with_message(program_info); // --- complain about_
41 ↪missing program parameter
42
43     utils::LogManager::get().FINE();
44     selectors::AtomSelector_SP selector = std::make_shared<
45         <selectors::SelectResidueByName>(argv[2]); // --- select a ligand residue by its 3-
46 ↪letter code
47
48     core::data::io::Pdb read_native(argv[1], core::data::io::keep_all); // --- Read the_
49 ↪native (reference) structure, keep all atoms
50     Structure_SP native = read_native.create_structure(0);
51     std::vector<Structure_SP> models; // --- Container for targets to be superimposed
52     for (int i = 3; i < argc; ++i) {
53         core::data::io::Pdb reader(argv[i], core::data::io::keep_all);
54         for (int j = 0; j < reader.count_models(); ++j) // --- Read all models from each_
55 ↪target PDB file
56             models.push_back(reader.create_structure(j));
57     }
58
59     selectors::AtomSelector_SP select_all = std::make_shared<selectors::AtomSelector>();
60     core::protocols::PairwiseCrmsd rms_calc(models, selector);
61     std::shared_ptr<std::ostream> out = std::make_shared<std::ofstream>("out.pdb");
62     rms_calc.calculate(native, out);
63 }

```



ap_symmetry_in_pdb

Example shows how to access symmetry operators stored in a PDB file header. For every operation found, it creates a Rototranslation object and prints it on a screen

USAGE:

```
c input.pdb
```

EXAMPLE:

```
ex_Remark290 5edw.pdb
```

Keywords:

- *PDB input*
- *PDB line filter*
- *Structure*

Categories:

- core/data/io/Pdb

Input files:

- 5edw.pdb
- 3dcg.pdb
- 5lpc.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <core/data/io/Pdb.hh>
3 #include <utils/options/OptionParser.hh>
4 #include <utils/exit.hh>
5
6 std::string program_info = R"(
7
8 Example shows how to access symmetry operators stored in a PDB file header. For every ↵
9 ↵operation found,
10 ↵it creates a Rototranslation object and prints it on a screen
11
12 USAGE:
13   c input.pdb
14 EXAMPLE:
```

(continues on next page)

(continued from previous page)

```

14     ex_Remark290 5edw.pdb
15
16 ) ";
17
18 /** @brief ex_Remark290 demo shows how to access symmetry operators stored in a PDB_
19   file header.
20 *
21 * CATEGORIES: core/data/io/Pdb;
22 * KEYWORDS: PDB input; PDB line filter; Structure
23 */
24
25 int main(const int argc, const char* argv[]) {
26
27     if ((argc > 1) && utils::options::call_for_help(argv[1]))
28         utils::exit_OK_with_message(program_info);
29
30     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
31   ↪missing program parameter
32     for (int i = 1; i < argc; ++i) {
33         core::data::io::Pdb reader(argv[i], // file name (PDB format, may be gzip-ped)
34           core::data::io::is_bb,           // a predicate to read only the ATOM lines_
35   ↪corresponding to backbone atoms
36           core::data::io::keep_all,      // keep all header lines
37           true);                      // parse PDB header
38
39     std::shared_ptr<core::data::io::Remark290> r290 = reader.symmetry_operators();
40     std::shared_ptr<core::data::io::Remark350> r350 = reader.biomolecule_symmetry();
41
42     std::cout << "# Symmetry operators found: " << r290->count_operators() << "\n";
43     for (const auto &rt: *r290) {
44         std::cout << rt << "\n";
45     }
46     std::cout << "# Biological symmetry biomolecules found: " << r350->count_
47   ↪biomolecules() << "\n";
48     for (const auto &sym: *r350) {
49         std::cout << "For chains: ";
50         for (auto c: sym.first) std::cout << c << " ";
51         std::cout << "with size " << sym.second.size() << "\n";
52         for (auto rt: sym.second) std::cout << rt << " ";
53         std::cout << "\n";
54     }
55 }
56 }
```



7.1.2 .py scripts

These group contains Python simple examples, which shows how to use PyBioShell package.

contact_map.py

Calculates contact map for a number of models from a single PDB file and prints how often any two residues are in contact

USAGE:

```
python3 contact_map.py input.pdb cutoff
```

EXAMPLE:

```
python3 contact_map.py 2kwi.pdb 4.5
```

Keywords:

- *PDB input*
- *contact map*

Categories:

- core/calc/structural/ContactMap

Input files:

- 2kwi.pdb

Output files:

- stdout.out

Program source:

```

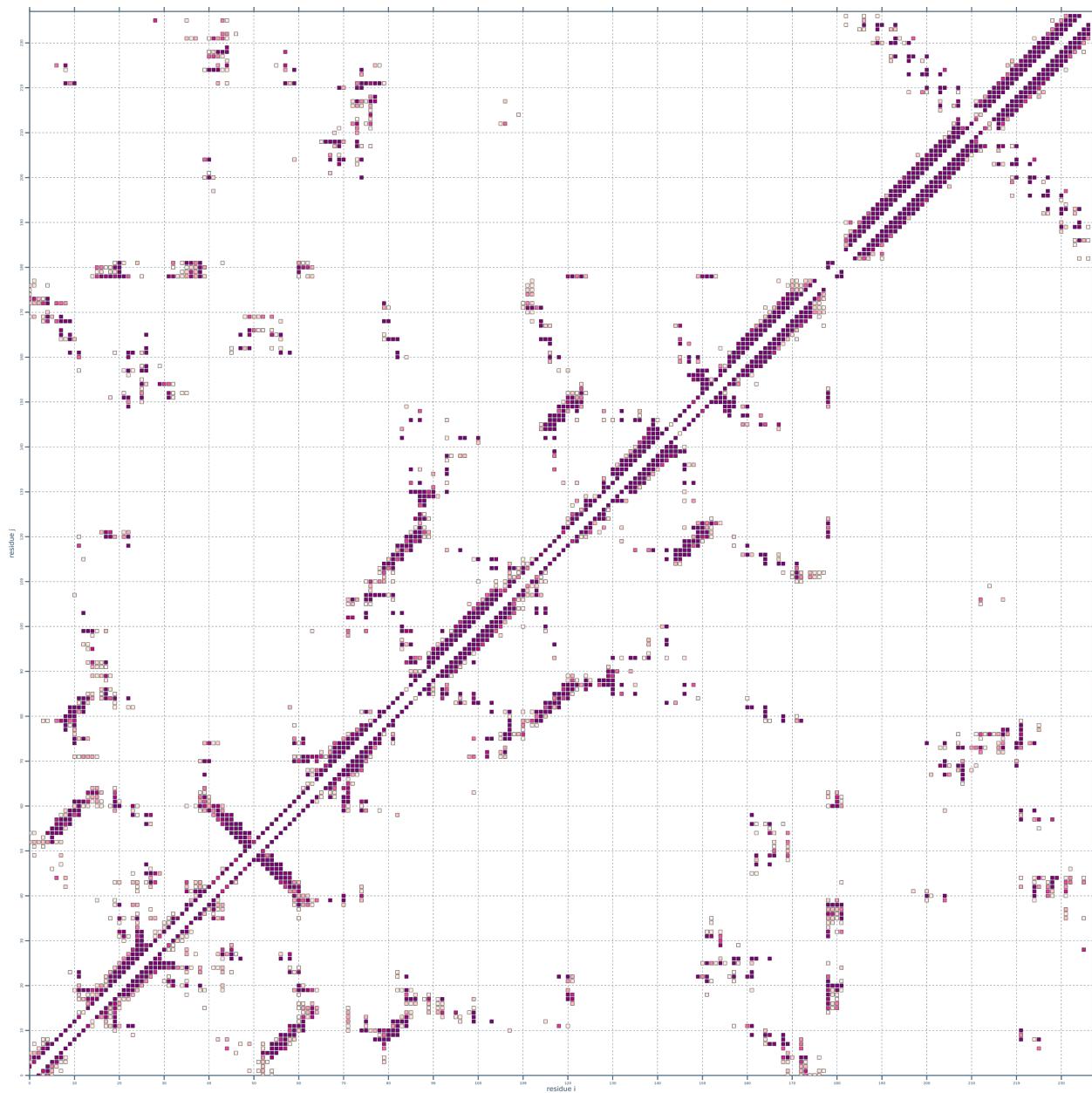
1 import sys
2
3 from pybioshell.core.data.io import Pdb
4 from pybioshell.core.calc.structural import evaluate_phi, evaluate_psi, ContactMap
5 import time
6
7 if len(sys.argv) < 3 :
8     print("""
9
10    Calculates contact map for a number of models from a single PDB file and prints how
11    ↪often any two residues
12    ↪are in contact
13

```

(continues on next page)

(continued from previous page)

```
14 USAGE:
15     python3 contact_map.py input.pdb cutoff
16
17
18 EXAMPLE:
19     python3 contact_map.py 2kwi.pdb 4.5
20
21
22 CATEGORIES: core/calc/structural/ContactMap
23 KEYWORDS: PDB input; contact map
24 GROUP: Structure calculations;
25 IMG: ap_contact_map.png
26
27 """
28     sys.exit()
29
30 pdb = Pdb(sys.argv[1], "")
31 cutoff = float(sys.argv[2])
32 contact_map = ContactMap(pdb.create_structure(0), cutoff)
33 for i_model in range(1,pdb.count_models()) :
34     strctr = pdb.create_structure(i_model)
35     contact_map.add(strctr)
36     print("model",i_model,"added", file=sys.stderr)
37
38 res_max = contact_map.max_row_index()
39 print(" i ci res_i j cj resj n_cont")
40 for i in range(res_max) :
41     # --- Get residue index for residue indexed by i; residue index is a structure_
42     # holding chain ID, residue ID and an insertion code
43     ri = contact_map.residue_index(i)
44     for j in range(i-1) :
45         rj = contact_map.residue_index(j)
46         if contact_map.has_element(i,j) :
47             print("%4d %c %4d%c %4d %c %4d%c %4d" %(i,ri.chain_id, ri.residue_id, ri.i_
48             code,
49                 j,rj.chain_id, rj.residue_id, rj.i_code,contact_map.at(i,j)))
```



crmsd_on_c-alpha.py

Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on C-alpha coordinates. If one file is given, each-vs-each cRMSD between models is calculated. If two or more file is given, crmsd for first pdb vs. the rest is calculated.

USAGE:

```
python3 crmsd_on_c-alpha.py file1.pdb [file2.pdb...]
```

EXAMPLE:

```
python3 crmsd_on_c-alpha.py 1cey.pdb
```

Keywords:

- *PDB input*
- *crmsd*

Categories:

- core/calc/structural/transformations/Crmsd

Input files:

- 2gb1-model3.pdb
- 2gb1-model1.pdb
- 2gb1-model4.pdb
- 1cey.pdb
- 2gb1-model2.pdb

Output files:

- stdout.out

Program source:

```
1 import sys, math
2
3 from pybioshell.core.data.io import Pdb
4 from pybioshell.core.data.basic import Vec3
5 from pybioshell.std import vector_core_data_basic_Vec3
6
7 from pybioshell.core.calc.structural.transformations import *
8 from pybioshell.utils import LogManager
9
10 LogManager.INFO()
11
12 if len(sys.argv) < 2:
13     print("""
14
15 Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on C-alpha coordinates.
16 ↵
17 If one file is given, each-vs-each cRMSD between models is calculated.
18 If two or more file is given, crmsd for first pdb vs. the rest is calculated.
19
20 USAGE:
21     python3 crmsd_on_c-alpha.py file1.pdb [file2.pdb...]
22
23
24 EXAMPLE:
25     python3 crmsd_on_c-alpha.py 1cey.pdb
```

(continues on next page)

(continued from previous page)

```

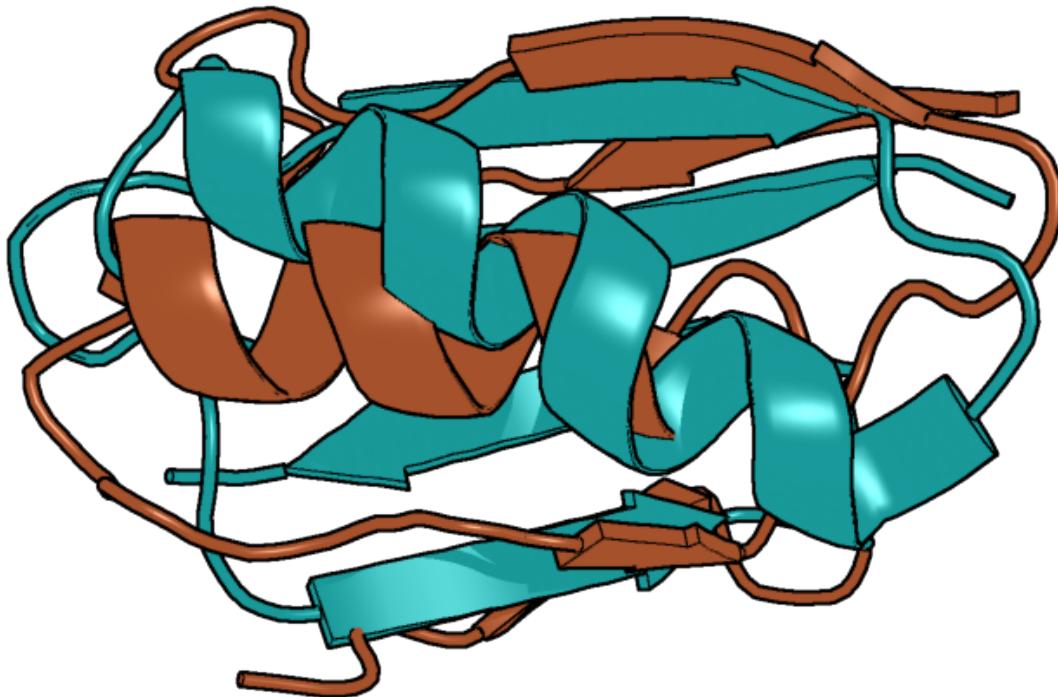
27
28 CATEGORIES: core/calc/structural/transformations/Crmsd
29 KEYWORDS: PDB input; crmsd
30 GROUP: Structure calculations
31 IMG: ap_Crmsd_deepteal_brown_1.png
32
33
34 """)
35     sys.exit()
36
37 rms = CrmsdOnVec3()
38
39 if len(sys.argv) == 2: # --- The case of each-vs-each calculations between models of_
40   ↪a single PDB file
41
42     pdb = Pdb(sys.argv[1], "is_ca", False)
43     n_atoms = pdb.count_atoms(0)
44
45     structure = pdb.create_structure(0)
46     models = []
47
48     for i_model in range(0, pdb.count_models()):
49       xyz = vector_core_data_basic_Vec3()
50       for i in range(n_atoms): xyz.append(Vec3())
51       models.append(xyz)
52
53     for i_model in range(0, pdb.count_models()):
54       pdb.fill_structure(i_model, models[i_model])
55       for j in range(i_model):
56         try:
57           print("%2d %2d %6.3f" % (i_model, j, rms.crmsd(models[i_model],_
58             ↪models[j], len(models[j]))))
59         except:
60           sys.stderr.write(str(sys.exc_info()[0]) + " " + str(sys.exc_
61             ↪info()[1]))
62
63 else: # --- The case when two or more PDB files are given, calculates first vs. the_
64   ↪rest
65
66     pdb = Pdb(sys.argv[1], "is_ca", False)
67     n_atoms = pdb.count_atoms(0)
68     structure = pdb.create_structure(0)
69
70     xyz = vector_core_data_basic_Vec3()
71     for i in range(n_atoms): xyz.append(Vec3())
72     pdb.fill_structure(0, xyz)
73
74     for pdb_fname in sys.argv[2:]:
75       other_pdb = Pdb(pdb_fname, "is_ca", False)
76       other_structure = other_pdb.create_structure(0)
77       if n_atoms != other_pdb.count_atoms():
78         print("The two structures have different number of CA atoms!\n")
79       other_xyz = vector_core_data_basic_Vec3()
80       for i in range(n_atoms): other_xyz.append(Vec3())
81       other_pdb.fill_structure(0, other_xyz)
82       try:
83         print("%s: %6.3f" % (pdb_fname.split("/")[-1].split(".")[0], rms.
84           ↪crmsd(xyz, other_xyz, n_atoms)))

```

(continues on next page)

(continued from previous page)

```
80     except:  
81         sys.stderr.write(str(sys.exc_info()[0]) + " " + str(sys.exc_info()[1]))
```



hist_from_scorefile.py

Reads Rosetta scorefile and plot histogram of given column name (default is rms) and energy plot.

EXAMPLE:

```
python3 hist_from_scorefile.py -f 1pgx-abinitio.fsc 1pgx-abinitio-bis.fsc -c  
score -min -50.0 -max 40.0
```

(continues on next page)

(continued from previous page)

Call <code>python3 hist_from_scorefile.py -h</code> for full help

Keywords:

- Rosetta scorefile
- histogram
- energy plot

Categories:

- core/calc/statistics/HistogramDD; core/data/io/read_scorefile

Input files:

- 1pgxA-abinitio.fsc
- 1pgxA-abinitio-bis.fsc

Output files:

- score_hist.svg
- rms_hist.svg
- stderr.out
- stdout.out
- stdout

Program source:

```

1 import sys, argparse
2
3 from pybioshell.core.data.io import find_pdb, read_scorefile
4
5 from os.path import expanduser, join
6 home_dir = expanduser("~/")
7 # It is assumed VisualLife library is installed in your $HOME/src.git/visualife/src/
8 # directory
9 sys.path.append(join(home_dir, "src.git/visualife/src/"))
10
11 from core.Plot import Plot
12 from core.SvgViewport import SvgViewport
13 from core.styles import ColorRGB, color_by_name
14
15 from pybioshell.core.calc.statistics import HistogramDD
16
17 if len(sys.argv) < 2 :
18     print(""""

```

(continues on next page)

(continued from previous page)

```

19 Reads Rosetta scorefile and plot histogram of given column name (default is rms) and
20   ↵energy plot.
21
22 EXAMPLE:
23   python3 hist_from_scorefile.py -f 1pgx-abinitio.fsc 1pgx-abinitio-bis.fsc -c
24   ↵score -min -50.0 -max 40.0
25
26
27 CATEGORIES: core/calc/statistics/HistogramDD; core/data/io/read_scorefile
28 KEYWORDS: Rosetta scorefile; histogram; energy plot
29 GROUP: Statistics;
30 IMG: rms_hist.svg
31
32 """
33     sys.exit()
34
35 # -----argument parsing
36 parser = argparse.ArgumentParser(description='Reads Rosetta scorefiles and plot
37   ↵histogram of given column name (default is rms) for all files in one plot')
38
39 parser.add_argument('-f', '--file', help="input .fsc file", nargs='+', required=True)
40 parser.add_argument('-c', '--column', help="column name for histogram", nargs=1,
41   ↵required=False, default=["rms"])
42 parser.add_argument('-min', '--min', help="minimum data value", nargs=1,
43   ↵required=False)
44 parser.add_argument('-max', '--max', help="maximum data value", nargs=1,
45   ↵required=False)
46 parser.add_argument('-b', '--bin_width', help="bin width", nargs=1, required=False,
47   ↵default=[1.0])
48
49
50 parser = parser.parse_args()
51
52 xmin = float(parser.min[0]) if parser.min else min(alldata)
53 xmax = float(parser.max[0]) if parser.max else max(alldata)
54 step = float(parser.bin_width[0])
55
56 data = []
57 alldata = []
58
59 # -----filling lists with data from file
60 for i in range(len(parser.file)):
61     p = read_scorefile(parser.file[i])
62     indx = p.column_index(parser.column[0])
63     print("#Making histogram for ",parser.column[0]," column at index ",indx)
64     lhist = []
65     for row in p:
66         lhist.append(float(row[indx]))
67     data.append(lhist)
68     alldata.extend(lhist)
69
70 # -----plotting-----
71 drawing = SvgViewport("%s_hist.svg"%(parser.column[0]), 0, 0, 800, 650,color="white")
72 pl = Plot(drawing,100,700,100,550,xmin, xmax,0,0.5,axes_definition="UBLR")
73
74 stroke_color = color_by_name("SteelBlue").create_darker(0.3)

```

(continues on next page)

(continued from previous page)

```

69 pl.axes["B"].label = parser.column[0]
70
71 for key,ax in pl.axes.items() :
72     ax.fill, ax.stroke, ax.stroke_width = stroke_color, stroke_color, 2.0
73     ax.tics(0,5)
74 pl.axes["U"].tics(5,0)
75 pl.axes["R"].tics(5,0)
76 pl.draw_axes()
77 pl.plot_label = "%s histogram" %(parser.column[0])
78 pl.draw_plot_label()

79
80
81 box_width = 0.9 * step if len(data) == 1 else step/(len(data)+1.0)
82 for i in range(len(data)):
83     # ----- here we actually make a histogram -----
84     h = HistogramDD(step,xmin,xmax)
85     for j in data[i]: h.insert(j)
86     h.normalize()
87     x_data = []
88     y_data = []
89     for b in range(h.count_bins()):
90         x_data.append(h.bin_min_val(b)+box_width*i)
91         y_data.append(h.get_bin(b))

92     print(h)
93
94     pl.bars(x_data, y_data, width=step, title="hist %s" %(i))
95
96
97 drawing.close()

```

phi_psi.py

Calculates Phi, Psi dihedral angles of a given input PDB structure.

USAGE:

```
python3 phi_psi.py input.pdb
```

EXAMPLE:

```
python3 phi_psi.py 2gb1.pdb
```

Keywords:

- *PDB input*
- *structural properties*

Categories:

- core/calc/structural/evaluate_phi; core/calc/structural/evaluate_psi

Input files:

- 2kwi.pdb
- 2gb1.pdb
- 4mcb.pdb
- 5edw.pdb

Output files:

- stdout.out

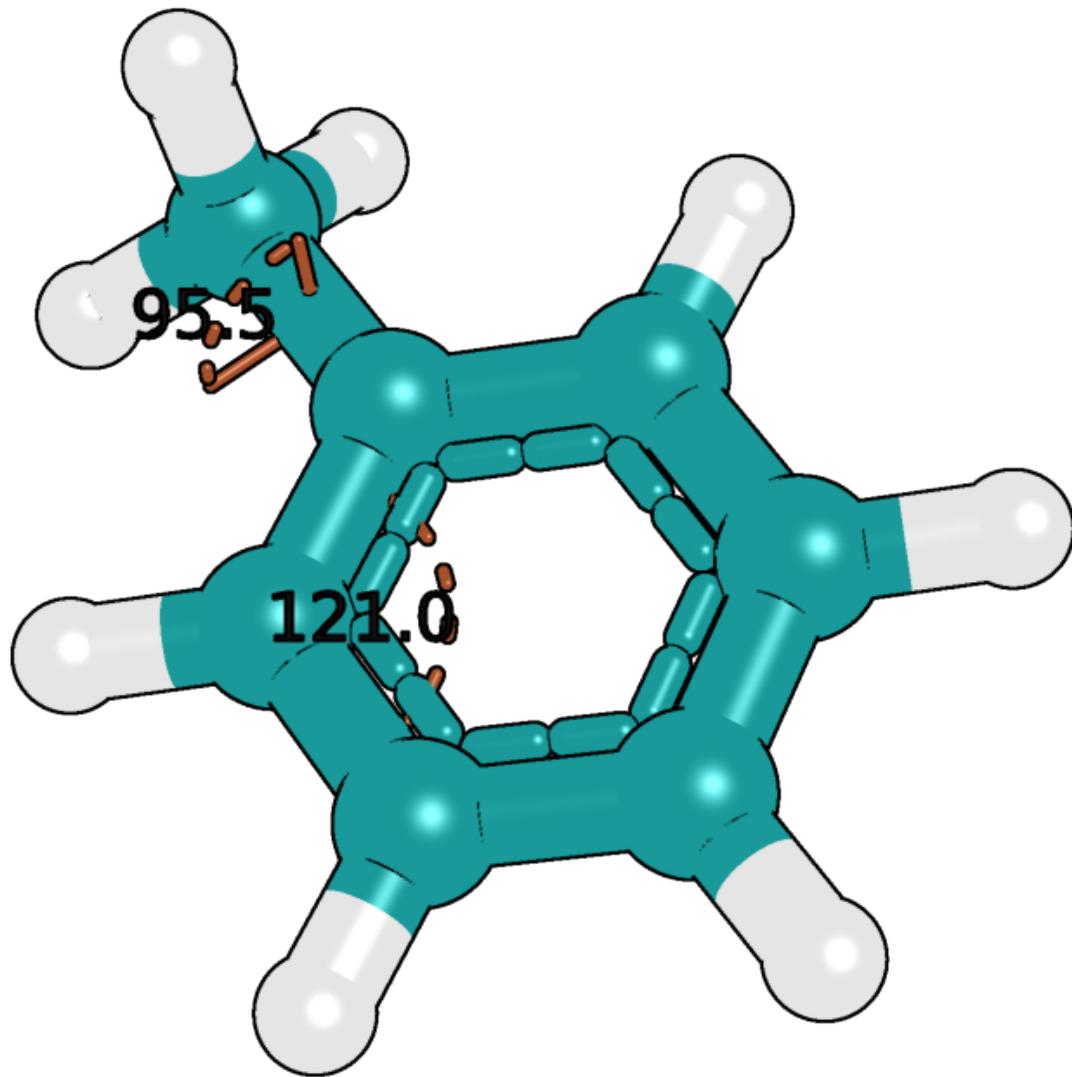
Program source:

```
1 import sys
2
3 from pybioshell.core.data.io import Pdb
4 from pybioshell.core.calc.structural import evaluate_phi, evaluate_psi
5
6 if len(sys.argv) < 2 :
7     print("""
8
9 Calculates Phi, Psi dihedral angles of a given input PDB structure.
10
11
12 USAGE:
13     python3 phi_psi.py input.pdb
14
15
16 EXAMPLE:
17     python3 phi_psi.py 2gb1.pdb
18
19
20 CATEGORIES: core/calc/structural/evaluate_phi; core/calc/structural/evaluate_psi
21 KEYWORDS: PDB input; structural properties
22 GROUP: Structure calculations
23 IMG: Toluene_dihedral_flat_angle.png
24
25 """)
26     sys.exit()
27
28 factor = 180.0/3.14159
29 structure = Pdb(sys.argv[1], "", False).create_structure(0)
30 for code in structure.chain_codes() :
31     chain = structure.get_chain(code)
32     n_res = chain.count_aa_residues()
33     for i_res in range(1,n_res-1) :
34         try :
35             r = chain[i_res]
36             r_prev = chain[i_res-1]
37             r_next = chain[i_res+1]
38             phi = evaluate_phi(r_prev,r)
39             psi = evaluate_psi(r,r_next)
40             print("%d %s %c %7.2f %7.2f" % (r.id(), r.residue_type().code3, r.owner().id(),
41             ↴phi*factor, psi*factor))
```

(continues on next page)

(continued from previous page)

```
41 except :  
42     print("can't evaluate Phi/Psi at position",i_res, file=sys.stderr)
```



score_rms_plot.py

Reads Rosetta scorefile and make an energy to rms plot.

USAGE:

```
python3 score_rms_plot.py -f file1 file2 ... [-x from to -y from to]
```

EXAMPLE:

```
python3 score_rms_plot.py -f 1pgx-abinitio.fsc 1pgx-abinitio-bis.fsc
```

Keywords:

- Rosetta scorefile
- every vs. rms plot

Categories:

- core/data/io/read_scorefile

Input files:

- 1pgxA-abinitio.fsc
- 1pgxA-abinitio-bis.fsc

Output files:

- stderr.out
- stdout.out
- score_to_rms.svg

Program source:

```
1 import sys, argparse
2
3 from pybioshell.core.data.io import find_pdb, read_scorefile
4
5 from os.path import expanduser, join
6 home_dir = expanduser("~/")
7 # It is assumed VisualLife library is installed in your $HOME/src.git/visualife/src/
8 # directory
9 sys.path.append(join(home_dir, "src.git/visualife/src/"))
10
11 from core.Plot import Plot
12 from core.SvgViewport import SvgViewport
13 from core.styles import ColorRGB, color_by_name
14
15 from pybioshell.core.calc.statistics import HistogramD4
16
17 if len(sys.argv) < 2 :
18     print("""
19 Reads Rosetta scorefile and make an energy to rms plot.
20
21 USAGE:
22     python3 score_rms_plot.py -f file1 file2 ... [-x from to -y from to]
```

(continues on next page)

(continued from previous page)

```

24
25
26 EXAMPLE:
27     python3 score_rms_plot.py -f lpgx-abinitio.fsc lpgx-abinitio-bis.fsc
28
29 CATEGORIES: core/data/io/read_scorefile
30 KEYWORDS: Rosetta scorefile; every vs. rms plot
31 GROUP: Statistics
32 IMG: score_to_rms.svg
33
34     """)
35     sys.exit()
36
37 #-----argument parsing
38 parser = argparse.ArgumentParser(description='Reads scorefile from Rosetta and'
39     'prepares score to rms plot for all given files as a series in one picture')
40
41 parser.add_argument('-f', '--file', help="input .fsc file", nargs='+', required=True)
42 parser.add_argument('-x', '--x_range', help="range for X axis of a plot (two values:"
43     'min max)', nargs=2, required=False)
44 parser.add_argument('-y', '--y_range', help="range for X axis of a plot (two values:"
45     'min max)', nargs=2, required=False)
46
47 parser = parser.parse_args()
48
49
50 energy = []
51 rms = []
52 allenergy = []
53 allrms = []
54 S=""
55
56 #-----filling lists with data from file
57 print("Plotting score vs. rms for:",end="")
58 for i in range(len(parser.file)):
59     S+=" "+parser.file[i]
60
61 p = read_scorefile(parser.file[i])
62
63 en = p.column_index("score")
64 xrms = p.column_index("rms")
65
66 e = []
67 r = []
68
69 for row in p:
70     e.append(float(row[en]))
71     r.append(float(row[xrms]))
72
73 energy.append(e)
74 rms.append(r)
75 allenergy.extend(e)
76 allrms.extend(r)
77
78 #-----plotting energy plot
79
80 xfrom = float(parser.x_range[0]) if parser.x_range else min(allrms)-1

```

(continues on next page)

(continued from previous page)

```

78 xto = float(parser.x_range[1]) if parser.x_range else max(allrms)+1
79 yfrom = float(parser.y_range[0]) if parser.y_range else min(allenergy)-10
80 yto = float(parser.y_range[1]) if parser.y_range else max(allenergy)+10
81
82
83 drawing = SvgViewport("outputs_from_test/score_to_rms.svg", 0, 0, 800, 650,color=
84     ↪"white")
85 pl = Plot(drawing,100,700,100,600,xfrom,yfrom,yto,axes_definition="UBLR")
86
87 stroke_color = color_by_name("SteelBlue").create_darker(0.3)
88 pl.axes["B"].label = "rmsd"
89 pl.axes["L"].label = "score"
90
91 for key,ax in pl.axes.items() :
92     ax.fill, ax.stroke, ax.stroke_width = stroke_color, stroke_color, 2.0
93     ax.tics(0,5)
94 pl.axes["U"].tics(5,0)
95 pl.axes["R"].tics(5,0)
96 pl.draw_axes()
97 pl.plot_label = "score_to_rms"
98 pl.draw_plot_label()
99 print(s)
100 for i in range(len(energy)):
101     pl.scatter(rms[i],energy[i], markersize=2, markerstyle='c', title="serie-%s"%(i))
102 drawing.close()

```

seq_identity.py

Reads a .fasta file with a set of amino acid sequences and calculates each-vs-each pairwise alignments using semi-global aligner. Prints only these pairs for which sequence identity is higher than a given cutoff.

USAGE:

```
python333 seq_identity.py input.fasta cutoff
```

EXAMPLE:

```
python3 seq_identity.py cyped.CYP109.fasta 0.3
```

REFERENCES: Smith, Temple F., and Michael S. Waterman. “Identification of common molecular subsequences.” Journal of molecular biology 147.1 (1981): 195-197. [https://doi.org/10.1016/0022-2836\(81\)90087-5](https://doi.org/10.1016/0022-2836(81)90087-5)

Needleman, Saul B., and Christian D. Wunsch. “A general method applicable to the search for similarities in the amino acid sequence of two proteins.” JMB 48.3 (1970): 443-453. [https://doi.org/10.1016/0022-2836\(70\)90057-4](https://doi.org/10.1016/0022-2836(70)90057-4)

Keywords:

- *FASTA input*
- *sequence alignment*

Categories:

- core/protocols/PairwiseSequenceIdentityProtocol

Input files:

- cyped.CYP109.fasta

Output files:

- stdout.out

Program source:

```

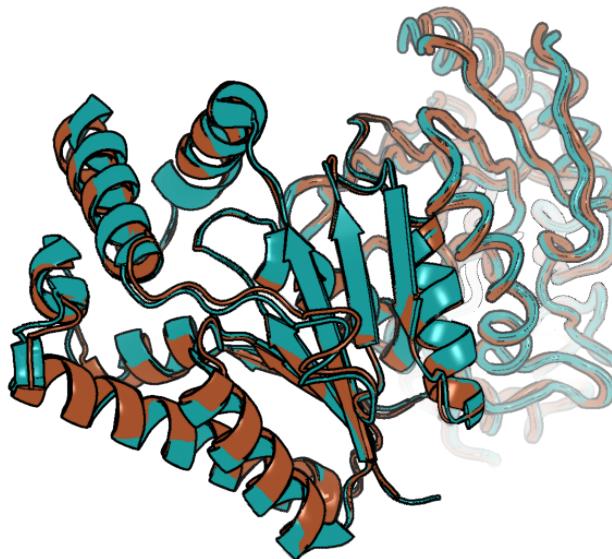
1 import sys
2
3 from pybioshell.std import vector_std_shared_ptr_core_data_sequence_Sequence_t
4 from pybioshell.core.data.io import read_fasta_file
5 from pybioshell.core.alignment import AlignmentType
6 from pybioshell.core.protocols import PairwiseSequenceIdentityProtocol
7
8
9 if len(sys.argv) < 3 :
10     print("""
11
12 Reads a .fasta file with a set of amino acid sequences and calculates each-vs-each_
13 ↪pairwise alignments
14 using semi-global aligner. Prints only these pairs for which sequence identity is_
15 ↪higher than a given cutoff.
16
17 USAGE:
18     python333 seq_identity.py input.fasta cutoff
19
20 EXAMPLE:
21     python3 seq_identity.py cyped.CYP109.fasta 0.3
22
23
24 REFERENCES:
25     Smith, Temple F., and Michael S. Waterman.
26     "Identification of common molecular subsequences." Journal of molecular biology_
27 ↪147.1 (1981): 195-197.
28     https://doi.org/10.1016/0022-2836\(81\)90087-5
29
30     Needleman, Saul B., and Christian D. Wunsch.
31     "A general method applicable to the search for similarities in the amino acid_
32 ↪sequence of two proteins."
33     JMB 48.3 (1970): 443-453. https://doi.org/10.1016/0022-2836\(70\)90057-4
34
35 CATEGORIES: core/protocols/PairwiseSequenceIdentityProtocol
36 KEYWORDS: FASTA input; sequence alignment
37 GROUP: Alignments

```

(continues on next page)

(continued from previous page)

```
37     IMG:      ap_aligned-1k6m-1bif.png
38     """
39
40     sys.exit()
41
42 cutoff = float(sys.argv[2])
43 sequences = vector_std_shared_ptr_core_data_sequence_Sequence_t()
44 read_fasta_file(sys.argv[1], sequences, True)
45 align_protocol = PairwiseSequenceIdentityProtocol()
46 n_seq = align_protocol.add_input_sequences(sequences)
47 align_protocol.n_threads(4).alignment_method(AlignmentType.SEMIGLOBAL_ALIGNMENT)
48 align_protocol.run()
49
50 for i in range(1,n_seq) :
51     for j in range(i) :
52         seq_id = align_protocol.get_sequence_identity(i,j)
53         if seq_id > cutoff : print( i, j, seq_id)
```



unwrap_pdb.py

Reads PDB file with wrapped coordinates (from simulation with periodic boundary conditions), unwraps them and generates PDB with it.

USAGE:

```
python3 unwrap_pdb.py input_pbc.pdb cutoff
```

EXAMPLE:

```
python3 unwrap_pdb.py out_pbc.pdb 40
```

Keywords:

- *PDB input*
- PBC
- SURPASS
- Vec3Cubic

Categories:

- core/data/basic/Vec3Cubic

Input files:

- out_pbc.pdb

Output files:

- stdout.out

Program source:

```

1 import sys, math,copy
2
3 from pybioshell.core.data.io import find_pdb
4 from pybioshell.core.data.basic import Vec3Cubic
5
6 from pybioshell.std import vector_core_data_basic_Vec3Cubic
7
8 if len(sys.argv) < 3 :
9     print("""
10
11 Reads PDB file with wrapped coordinates (from simulation with periodic boundary_ ↵conditions), unwraps them and generates PDB with it.
12
13
14 USAGE:
15     python3 unwrap_pdb.py input_pbc.pdb cutoff
16
17
18 EXAMPLE:
19     python3 unwrap_pdb.py out_pbc.pdb 40
20
21
22 CATEGORIES: core/data/basic/Vec3Cubic
23 KEYWORDS: PDB input; PBC; SURPASS; Vec3Cubic
24 IMG: unwrapped.gif
25
26     """)
27     sys.exit()
28
29

```

(continues on next page)

(continued from previous page)

```

30 pdb = find_pdb(sys.argv[1], "./")
31 n_atoms = pdb.count_atoms(0)
32 cutoff = float(sys.argv[2])
33
34 Vec3Cubic.set_box_len(cutoff)
35 xyz = vector_core_data_basic_Vec3Cubic()
36 for i in range(n_atoms): xyz.append(Vec3Cubic())
37
38 for i_model in range(0, pdb.count_models()) :
39     xyz = vector_core_data_basic_Vec3Cubic()
40     for i in range(n_atoms): xyz.append(Vec3Cubic())
41     pdb.fill_structure(i_model, xyz)
42     structure = pdb.create_structure(i_model)
43     n_res = 0
44     print("MODEL      ", i_model+1 )
45     for ia in range(structure.count_chains()):
46         chain = structure[ia]
47         #wrapping first atom of every chain to the first box
48         if xyz[n_res ].x > cutoff: xyz[n_res ].x -= cutoff
49         if xyz[n_res ].x < 0: xyz[n_res ].x += cutoff
50         if xyz[n_res ].y > cutoff: xyz[n_res ].y -= cutoff
51         if xyz[n_res ].y < 0: xyz[n_res ].y += cutoff
52         if xyz[n_res ].z > cutoff: xyz[n_res ].z -= cutoff
53         if xyz[n_res ].z < 0: xyz[n_res ].z += cutoff
54         chain[0][0].set(xyz[n_res ])
55         #calculating unwraped coordinates
56         for ir in range(chain.count_residues()-1):
57             ax = xyz[n_res+ir+1].closest_delta_x(xyz[n_res+ir])
58             ay = xyz[n_res+ir+1].closest_delta_y(xyz[n_res+ir])
59             az = xyz[n_res+ir+1].closest_delta_z(xyz[n_res+ir])
60             xyz[n_res+ir+1].x = xyz[n_res+ir].x + ax
61             xyz[n_res+ir+1].y = xyz[n_res+ir].y + ay
62             xyz[n_res+ir+1].z = xyz[n_res+ir].z + az
63
64             chain[ir+1][0].set(xyz[n_res+ir+1])
65     n_res+=ir+2
66     #writing to PDB file
67     for ir in range(chain.count_residues()) :
68         resid = chain[ir]
69         print(resid[0].to_pdb_line())
70     print("ENDMDL")
71

```

align_sequences.py

Computes pairwise sequence alignment

USAGE:

python3 align_sequences.py input-1.fasta input-2.fasta [gap_open gap_cont]

EXAMPLE:

```
python3 align_sequences.py 2azaA.pdb 2pcyA.pdb
```

Keywords:

- alignment

Categories:

- core/alignment/SequenceNWAligner

Input files:

- 2pcyA.fasta
- 2azaA.fasta

Output files:

- stdout.out

Program source:

```

1 import sys
2
3 from pybioshell.core.alignment import SequenceNWAligner, SequenceSWAligner
4 from pybioshell.core.data.io import read_fasta_file
5
6 if len(sys.argv) < 3 :
7     print("""
8
9 Computes pairwise sequence alignment
10
11
12 USAGE:
13     python3 align_sequences.py input-1.fasta input-2.fasta [gap_open gap_cont]
14
15
16 EXAMPLE:
17     python3 align_sequences.py 2azaA.pdb 2pcyA.pdb
18
19
20 CATEGORIES: core/alignment/SequenceNWAligner
21 KEYWORDS: alignment
22 GROUP: Sequence calculations
23
24     """)
25     sys.exit()
26 q = read_fasta_file(sys.argv[1])[0]
27 t = read_fasta_file(sys.argv[2])[0]
28 # ----- calculate a global alignment
29 aligner = SequenceNWAligner(max(q.length(), t.length()))

```

(continues on next page)

(continued from previous page)

```
30 # ----- calculate a local alignment
31 aligner = SequenceSWAligner(max(q.length(), t.length()))
32 if len(sys.argv) < 4 :
33     score = aligner.align(q, t, -10, -1, "BLOSUM62")
34 else:
35     score = aligner.align(q, t, int(sys.argv[3]), int(sys.argv[4]), "BLOSUM62")
36
37 alignment = aligner.backtrace_sequence_alignment()
38 print("# score:", score)
39 print("> query\n" + alignment.get_aligned_query())
40 print("> template\n" + alignment.get_aligned_template())
```



asn1_to_profile.py

Converts a sequence profile (in ASN.1 format) produced by psiblast to a flat tabular format

USAGE:

```
python asn1_to_profile.py input.asn1
```

EXAMPLE:

```
python asn1_to_profile.py d1or4A_.asn1
```

Keywords:

- *sequence profile*
- *Format conversion*

Categories:

- core/data/sequence/read ASN1_checkpoint

Input files:

- **d1or4A_.asn1**

Output files:

- stdout.out

Program source:

```
1 import sys
2
3 from pybioshell.core.data.sequence import read ASN1_checkpoint
4
5 if len(sys.argv) < 2 :
6     print """
7
8 Converts a sequence profile (in ASN.1 format) produced by psiblast to a flat tabular_
9 →format
10
11 USAGE:
12     python asn1_to_profile.py input.asn1
13
14
15 EXAMPLE:
16     python asn1_to_profile.py d1or4A_.asn1
17
18
19 CATEGORIES: core/data/sequence/read ASN1_checkpoint
```

(continues on next page)

(continued from previous page)

```
20 KEYWORDS: sequence profile; Format conversion
21 GROUP: File processing; Format conversion
22
23 """
24     sys.exit()
25
26 profile = read ASN1_checkpoint(sys.argv[1])
27 profile.write_table_header()
28 profile.write_table()
```



betastructures_graph.py

Shows how to use BetaStructuresGraph class from Python

USAGE:

```
python3 betastructures_graph.py input.pdb
```

EXAMPLE:

```
python3 betastructures_graph.py 5edw.pdb
```

Keywords:

- *PDB input*
- *graphs*

Categories:

- core/calc/structural/ProteinArchitecture

Input files:

- 2gb1.pdb
- 5edw.pdb

Output files:

- stdout.out

Program source:

```

1 import sys
2
3 from pybioshell.core.data.io import Pdb
4 from pybioshell.core.calc.structural import ProteinArchitecture
5
6
7 if len(sys.argv) < 2 :
8     print("""
9
10 Shows how to use BetaStructuresGraph class from Python
11
12
13 USAGE:
14     python3 betastructures_graph.py input.pdb
15
16
17 EXAMPLE:
18     python3 betastructures_graph.py 5edw.pdb

```

(continues on next page)

(continued from previous page)

```
19
20 CATEGORIES: core/calc/structural/ProteinArchitecture
21 KEYWORDS: PDB input; graphs
22
23 """
24     sys.exit()
25
26 pdb_fname = sys.argv[1]
27 structure = Pdb(pdb_fname, "").create_structure(0)
28 architecture = ProteinArchitecture(structure)
29 beta_graph = architecture.create_strand_graph()
30 strands = beta_graph.get_strands_copy()
31
32 print("    ",end="")
33 for i in range(len(strands)) : print(" %2d " % (i),end="")
34 print()
35
36 for i in range(len(strands)) :
37     pairings = []
38     print("S%2d: " % (i),end="")
39     for j in range(len(strands)) :
40         are_paired = False
41         try :
42             p = beta_graph.get_strand_pairing(strands[i],strands[j])
43             are_paired = True
44         except: pass
45         if are_paired :
46             print(" X ")
47             pairings.append(j)
48         else : print("      ")
49     print("| %-45s paired with %d" % (str(strands[i]),pairings[0]),end="")
50     for pi in range(1,len(pairings)) :
51         print(", %d" % (pairings[pi]),end="")
52     print()
```



caonly_multimodel.py

Reads multiple PDB files and writes C-alpha atom of all structures into a single multimodel pdb file. The input file is a simple text file providing PDB file names (one string per line).

USAGE:

```
python3 caonly_multimodel.py.py input_structres_list [output_fname.pdb]
```

EXAMPLE:

```
python3 caonly_multimodel.py.py cat_lits o.pdb
```

Keywords:

- *PDB input*
- *structure selectors*
- *PDB output*

Categories:

- core/data/io/Pdb

Input files:

- 2gb1-model3.pdb
- 2gb1-model1.pdb
- 2gb1-model4.pdb
- cat_list
- 2gb1-model2.pdb

Output files:

- o.pdb
- stderr.out
- stdout.out

Program source:

```
1 import sys
2
3 from pybioshell.core.data.io import Pdb, write_pdb
4
5 if len(sys.argv) < 2 :
6     print("""
7
```

(continues on next page)

(continued from previous page)

```

8 Reads multiple PDB files and writes C-alpha atom of all structures into a single multimodel pdb file.
9
10 The input file is a simple text file providing PDB file names (one string per line).
11
12
13 USAGE:
14     python3 caonly_multimodel.py.py input_structres_list [output_fname.pdb]
15
16 EXAMPLE:
17     python3 caonly_multimodel.py.py cat_lits o.pdb
18
19
20 CATEGORIES: core/data/io/Pdb
21 KEYWORDS: PDB input; structure selectors; PDB output
22 GROUP: File processing
23
24 """
25     sys.exit()
26
27 input_fnames = open(sys.argv[1])
28 reader = Pdb(input_fnames.readline().strip(),"is_ca",False)
29 structure = reader.create_structure(0)
30 out_fname = "out.pdb" if len(sys.argv) == 2 else sys.argv[2]
31 write_pdb(structure, out_fname, 1)
32 i_model = 2
33 print("Reading PDB files: ",end="")
34 for pdb_fname in input_fnames :
35     print(pdb_fname.strip().split("/")[-1],end=" ")
36     reader = Pdb(pdb_fname.strip(),"is_ca",False)
37     reader.fill_structure(0,structure)
38     write_pdb(structure, out_fname, i_model)
39     i_model += 1

```



center_protein.py

Moves a given protein structure so its geometric center is located at (0,0,0).

USAGE:

```
python3 center_protein.py input.pdb [which_model]
```

EXAMPLE:

```
python3 center_protein.py 2kwi.pdb 51
```

Keywords:

- *PDB input*
- center protein
- *internal coordinates*

Categories:

- core/data/io/find_pdb

Input files:

- 2kwi.pdb

Output files:

- stdout.out

Program source:

```

1 import sys
2
3 from pybioshell.core.data.io import find_pdb
4
5 if len(sys.argv) < 2 :
6     print("""
7
8 Moves a given protein structure so its geometric center is located at (0,0,0).
9
10 USAGE:
11     python3 center_protein.py input.pdb [which_model]
12
13
14 EXAMPLE:
15     python3 center_protein.py 2kwi.pdb 51
16
17     CATEGORIES: core/data/io/find_pdb
18     KEYWORDS: PDB input; center protein; internal coordinates

```

(continues on next page)

(continued from previous page)

```
19 GROUP: Structure calculations;
20
21 """
22     sys.exit()
23
24 which_model = 0 if len(sys.argv) == 2 else (int(sys.argv[2])-1)
25
26 pdb_reader = find_pdb(sys.argv[1], "./")
27 structure = pdb_reader.create_structure(which_model)
28 cx, cy, cz, n = 0, 0, 0, 0
29 for ic in range(structure.count_chains()) :
30     chain = structure[ic]
31     for ir in range(chain.count_residues()) :
32         resid = chain[ir]
33         for ai in range(resid.count_atoms()) :
34             cx += resid[ai].x
35             cy += resid[ai].y
36             cz += resid[ai].z
37             n+=1.0
38
39 cx /= n
40 cy /= n
41 cz /= n
42 print("# Center was:",cx,cy,cz)
43
44 for ic in range(structure.count_chains()) :
45     chain = structure[ic]
46     for ir in range(chain.count_residues()) :
47         resid = chain[ir]
48         for ai in range(resid.count_atoms()) :
49             resid[ai].x -= cx
50             resid[ai].y -= cy
51             resid[ai].z -= cz
52             print(resid[ai].to_pdb_line())
```



check_structure.py

Checks if a given structure has chain breaks

USAGE:

```
python3 check_structure.py input.pdb
```

EXAMPLE:

```
python3 check_structure.py 2gb1.pdb
```

Keywords:

- *PDB input*
- *structural properties*

Categories:

- core/calc/structural/

Input files:

- 2gb1.pdb
- 3dcg.pdb

Output files:

- stdout.out

Program source:

```
1 import sys
2
3 sys.path.append("/Users/dgront/src.git/bioshell/bin")
4 from pybioshell.core.data.io import Pdb
5
6 if len(sys.argv) < 2 :
7     print("""
8
9 Checks if a given structure has chain breaks
10
11
12 USAGE:
13     python3 check_structure.py input.pdb
14
15
16 EXAMPLE:
17     python3 check_structure.py 2gb1.pdb
```

(continues on next page)

(continued from previous page)

```

19
20 CATEGORIES: core/calc/structural/
21 KEYWORDS: PDB input; structural properties
22 GROUP: Structure calculations
23
24     """)
25     sys.exit()
26
27 N_gaps = 0
28 structure = Pdb(sys.argv[1], "", False).create_structure(0)
29 for code in structure.chain_codes() :
30     chain = structure.get_chain(code)
31     r_prev, r = chain[0], chain[0]
32     prev_ca = r.find_atom(" CA ")
33     the_ca = prev_ca
34     for ires in range(1,chain.size()):
35         r_prev = r
36         prev_ca = the_ca
37         if not prev_ca:
38             continue
39         r = chain[ires]
40         the_ca = r.find_atom(" CA ")
41         if not the_ca:
42             continue
43
44         d = the_ca.distance_to(prev_ca)
45         if d > 4.0:
46             N_gaps += 1
47             print("chain %c: too long distance between CA of %s%d and %s%d residue:
48             ↪%6.3f" %
49             (r.owner().id(), r_prev.residue_type().code3, r_prev.id(), r.
50             ↪residue_type().code3, r.id(), d))
51
52 print("# Summary for %s: n_gaps - %d" % (sys.argv[1], N_gaps))

```



cif_to_mol2.py

Converts a small molecule structure from CIF to MOL2 file format. The last, optional parameter of the script provides the name of a given molecule, that will be stored in MOL2 file

USAGE:

```
python3 cif_to_mol2.py input.cif [molecule_name]
```

EXAMPLE:

```
python3 cif_to_mol2.py HEM.cif [molecule_name]
python3 cif_to_mol2.py HEM.cif HAEM
```

Keywords:

- *CIF input*
- MOL2 output
- *Format conversion*

Categories:

- core/data/io/write_mol2

Input files:

- HEM.cif

Output files:

- stdout.out

Program source:

```

1 import sys
2
3 from pybioshell.core.data.io import Cif, write_mol2
4 from pybioshell.core.chemical import MonomerStructure
5
6
7 if len(sys.argv) < 2 :
8     print("""
9
10 Converts a small molecule structure from CIF to MOL2 file format.
11 The last, optional parameter of the script provides the name of a given
12 molecule, that will be stored in MOL2 file
13
14
15 USAGE:
16     python3 cif_to_mol2.py input.cif [molecule_name]
```

(continues on next page)

(continued from previous page)

```
17
18
19 EXAMPLE:
20     python3 cif_to_mol2.py HEM.cif [molecule_name]
21     python3 cif_to_mol2.py HEM.cif HAEM
22
23 CATEGORIES: core/data/io/write_mol2
24 KEYWORDS: CIF input; MOL2 output; Format conversion
25 GROUP: File processing; Format conversion
26 """
27     sys.exit()
28
29 mm = MonomerStructure.from_cif(sys.argv[1])
30 if len(sys.argv) > 2:                      # --- set molecule name if provided from_
31     ↪ command line
32     mm.molecule_name = sys.argv[2]
33 write_mol2(mm, "stdout")
```



convert_msa.py

Converts multiple sequence alignment (MSA) data from one format to another. Known input formats: FASTA (.fasta), HSSP (.hssp) and ClustalW/ClustalO (.aln) Known output formats: FASTA (.fasta) Input and output file formats are detected by file extension

USAGE:

```
python3 convert_msa.py input_file output_file
```

EXAMPLE:

```
python3 convert_msa.py cyped.CYP109.aln cyped.CYP109.fasta
```

Keywords:

- *MSA*
- *Format conversion*

Categories:

- core/data/io/read_hssp_file

Input files:

- CYP109B1.aln
- 1crn.hssp

Output files:

- 1crn.fasta
- CYP109B1.fasta

Program source:

```
1 import sys
2
3 from pybioshell.std import vector_std_shared_ptr_core_data_sequence_Sequence_t
4 from pybioshell.core.data.io import read_hssp_file, write_clustalo_file, read_fasta_
5   file, read_clustalw_file
6
7 from pybioshell.utils import LogManager
8
9 LogManager.FINEST()
10
11 if len(sys.argv) < 3 :
12   print("""
13   Converts multiple sequence alignment (MSA) data from one format to another.
14   Known input formats: FASTA (.fasta), HSSP (.hssp) and ClustalW/ClustalO (.aln)
```

(continues on next page)

(continued from previous page)

```
15 Known output formats: FASTA (.fasta)
16
17 Input and output file formats are detected by file extension
18
19 USAGE:
20     python3 convert_msa.py input_file output_file
21
22
23 EXAMPLE:
24     python3 convert_msa.py cyped.CYP109.aln cyped.CYP109.fasta
25
26
27 CATEGORIES: core/data/io/read_hssp_file
28 KEYWORDS: MSA; Format conversion
29 GROUP: File processing;
30
31 """
32     sys.exit()
33
34 msa = vector_std_shared_ptr_core_data_sequence_Sequence_t() # --- vector to hold
35     ↪sequences obtained from an input file
35 extension_in = sys.argv[1].split('.')[−1]                      # --- detect the input
36     ↪format
36 if extension_in == 'aln':
37     read_clustalw_file(sys.argv[1], msa)
38 elif extension_in == 'hssp':
39     read_hssp_file(sys.argv[1], msa)
40 elif extension_in == 'fasta':
41     read_fasta_file(sys.argv[1], msa)
42
43 f = open(sys.argv[2], "w")
44 for seq in msa:
45     print(">", seq.header(), file=f)
46     print(seq.sequence, file=f)
47 f.close()
```



crmsd_on_ligands.py

Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on all atoms of a ligand conformations. This script reads one or more PDB files, extracts all ligands that matches a given three-letter code and calculates cRMSD between them on all atoms.

USAGE:

```
python3 crmsd_on_ligands.py ligand input1.pdb [input2.pdb]
```

EXAMPLE:

```
python3 crmsd_on_ligands.py HEM 5ofq.pdb 4rm4.pdb
```

Keywords:

- *PDB input*
- *ligand*
- *crmsd*

Categories:

- core/calc/structural/transformations/CrmsdOnVec3

Input files:

- 5ofq.pdb
- 4rm4.pdb

Output files:

- stdout.out

Program source:

```

1 import sys, math
2
3 from pybioshell.core.data.io import Pdb
4 from pybioshell.std import vector_core_data_basic_Vec3
5 from pybioshell.core.calc.structural.transformations import *
6 from pybioshell.utils import LogManager
7 LogManager.INFO()
8
9 if len(sys.argv) < 3 :
10     print("""
11
12 Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on all atoms of a
13     ↵ligand conformations.
14
15 This scripts reads one or more PDB files, extracts all ligands that matches a given

```

(continues on next page)

(continued from previous page)

```

15 three-letter code and calculates cRMSD between them on all atoms.
16
17
18 USAGE:
19     python3 crmsd_on_ligands.py ligand input1.pdb [input2.pdb]
20
21
22 EXAMPLE:
23     python3 crmsd_on_ligands.py HEM 5ofq.pdb 4rm4.pdb
24
25
26 CATEGORIES: core/calc/structural/transformations/CrmsdOnVec3
27 KEYWORDS: PDB input; ligand; crmsd
28 GROUP: Structure calculations
29
30 """
31     sys.exit()
32
33 rms = CrmsdOnVec3()
34
35 pdb_codes = []          # --- PDB code for every input structure: for informative_
36             ↵output
37 structures = []          # --- contains all input structures
38 residues = []           # --- ligand residue objects (to keep information about_
39             ↵residue number and chain)
40 atoms_by_ligand = []    # --- a list of atoms for every ligand
41 for pdb_code in sys.argv[2:] :
42     pdb = Pdb(pdb_code,"",False)
43     structure = pdb.create_structure(0)
44     structures.append(structure)
45     for ic in range(structure.count_chains()) :
46         chain = structure[ic]
47         for ir in range(chain.terminal_residue_index() + 1,chain.size()) :
48             resid = chain[ir]
49             code3 = resid.residue_type().code3
50             if code3 != sys.argv[1] : continue # Skip other ligands
51             residues.append(resid)
52             pdb_codes.append(pdb_code)
53             atoms = vector_core_data_basic_Vec3()
54             for ia in range(resid.count_atoms()):
55                 atoms.append(resid[ia])
56             atoms_by_ligand.append(atoms)
57
58 for i_ligand in range(0, len(atoms_by_ligand)) :
59     ir = residues[i_ligand]
60     for j_ligand in range(i_ligand):
61         jr = residues[j_ligand]
62         crmsd_val = rms.crmsd(atoms_by_ligand[i_ligand], atoms_by_ligand[j_ligand],_
63             ↵len(atoms_by_ligand[j_ligand]))
64         print("%s %4d %3s %c - %s %4d %3s %c : %7.3f" %
65               (pdb_codes[i_ligand], ir.id(), ir.residue_type().code3, ir.owner().id(),
66                pdb_codes[j_ligand], jr.id(), jr.residue_type().code3, jr.owner().id(),_
67                ↵crmsd_val))

```



fasta_subset.py

Reads a multiple FASTA files and print a randomly selected fraction of sequences.

USAGE:

```
python3 read_fasta.py faction input.fasta
```

EXAMPLE:

```
python3 read_fasta.py 0.01 small500_95identical.fasta
```

Keywords:

- *FASTA input*
- *sequence*

Categories:

- core/data/io/read_fasta_file

Input files:

- 5edw.fasta
- small500_95identical.fasta

Output files:

- stdout.out

Program source:

```
1 import sys
2 from random import random, seed
3
4 from pybioshell.core.data.io import read_fasta_file, create_fasta_string
5
6
7 if len(sys.argv) < 3 :
8     print("""
9
10 Reads a multiple FASTA files and print a randomly selected fraction of sequences.
11
12
13 USAGE:
14     python3 read_fasta.py faction input.fasta
15
16
17 EXAMPLE:
18     python3 read_fasta.py 0.01 small500_95identical.fasta
```

(continues on next page)

(continued from previous page)

```
19
20
21 CATEGORIES: core/data/io/read_fasta_file
22 KEYWORDS:   FASTA input; sequence
23 GROUP:      File processing; Data filtering
24
25     """)
26     sys.exit()
27
28 seed(0)
29 fasta = read_fasta_file(sys.argv[2])
30 for fname in sys.argv[3:] : read_fasta_file(fname,fasta)
31
32 fraction = float(sys.argv[1])
33 for seq in fasta:
34     if random() < fraction : print(create_fasta_string(seq))
```



filter_scorefile.py

Reads Rosetta scorefile and prints only it's requested part

EXAMPLE:

```
python3 filter_scorefile.py 1pgx-abinitio.fsc 1pgx-abinitio-bis.fsc score -50.0 40.0
```

Call python3 filter_scorefile.py -h for full help

Keywords:

- *Rosetta scorefile*
- :ref:``

Categories:

- core/data/io/read_scorefile

Input files:

- 1pgxA-abinitio.fsc
- 1pgxA-abinitio-bis.fsc

Output files:

- stdout.out

Program source:

```

1 import sys, argparse
2
3 from pybioshell.core.data.io import read_scorefile
4
5 if len(sys.argv) < 2 :
6     print("""
7
8 Reads Rosetta scorefile and prints only it's requested part
9
10 EXAMPLE:
11     python3 filter_scorefile.py 1pgx-abinitio.fsc 1pgx-abinitio-bis.fsc score -50.0
12     ↪40.0
13
14 Call python3 filter_scorefile.py -h for full help
15
16 CATEGORIES: core/data/io/read_scorefile
17 KEYWORDS: Rosetta scorefile;
18 GROUP: Statistics;
19 """
20     )
21     sys.exit()
```

(continues on next page)

(continued from previous page)

```
21
22 # -----argument parsing
23 parser = argparse.ArgumentParser(description="Reads Rosetta scorefile and prints only the requested part")
24
25 parser.add_argument('-f', '--file', help="input .fsc file", nargs='+', required=True)
26 parser.add_argument('-c', '--column', help="column name(s) to keep", nargs='+', required=False, default=["score", "rms"])
27 parser = parser.parse_args()
28
29 columns = [col_name for col_name in parser.column]
30
31 # ----- Print scorefile header
32 print("SCORE: ", end="")
33 for col_name in columns:
34     print(col_name, end=" ")
35 print()
36
37 # ----- Print scorefile data
38 for file_name in parser.file:
39     sf = read_scorefile(file_name)
40     for i_row in range(len(sf)):
41         row = sf[i_row]
42         print("SCORE: ", end="")
43         for col_name in columns:
44             print(row[sf.column_index(col_name)], end=" ")
45         print()
```



find_rings.py

Reads in a small molecule (PDB file format) and prints all cycles (i.e. rings) that can be found. Note, that rings may be nested, e.g. naphthalene molecule has actually three rings!

USAGE:

```
python3 find_rings.py molecule.pdb
```

EXAMPLE:

```
python3 find_rings.py 9ZB_ideal.pdb
```

Keywords:

- *PDB input*
- small molecules

Categories:

- core/chemical/

Input files:

- 9ZB_ideal.pdb
- MES_ideal.pdb

Output files:

- stdout.out

Program source:

```
1 import sys
2
3 from pybioshell.core.chemical import PdbMolecule
4 from pybioshell.core.chemical import find_rings
5
6 if len(sys.argv) < 2 :
7     print("""
8
9 Reads in a small molecule (PDB file format) and prints all cycles (i.e. rings) that
10    ↪can be found.
11 Note, that rings may be nested, e.g. naphthalene molecule has actually three rings!
12
13 USAGE:
14     python3 find_rings.py molecule.pdb
15
16 EXAMPLE:
```

(continues on next page)

(continued from previous page)

```
17 python3 find_rings.py 9ZB_ideal.pdb
18
19 CATEGORIES: core/chemical/
20 KEYWORDS: PDB input; small molecules
21 GROUP: small molecules;
22
23 """)
24 sys.exit()
25
26 mol = PdbMolecule.from_pdb(sys.argv[1])
27 rings = find_rings(mol)
28 for ring in rings:
29     print("# -----")
30     for atom in ring:
31         print(mol.get_atom(atom).to_pdb_line())
32
```



hhpred_to_modeller.py

Reads an output file produced by HHpred, that contains alignments between a query protein and template protein structures. Writes PIR input files necessary for Modeller to build structural models of the query based on a given alignment (by default the first alignment is used)

USAGE:

```
python3 hhpred_to_modeller.py hhpred_output [which-alignment [other alignments ...] ]
```

EXAMPLE:

```
python3 hhpred_to_modeller.py CYP51F.hhpred 1 2
```

Keywords:

- HHpred
- comparative modelling

Categories:

- core/data/io/read_hhpred

Input files:

- CYP51F.hhpred

Output files:

- stdout.out
- 1.pir
- 2.pir

Program source:

```

1 import sys
2
3 from pybioshell.core.data.io import read_hhpred, create_pir_string
4
5
6 if len(sys.argv) < 2 :
7     print("""
8
9     Reads an output file produced by HHpred, that contains alignments between a query_
10    ↪protein and template protein structures.
11    Writes PIR input files necessary for Modeller to build structural models of the_
12    ↪query based on a given alignment
13        (by default the first alignment is used)

```

(continues on next page)

(continued from previous page)

```
14 USAGE:  
15     python3 hhpred_to_modeller.py hhpred_output [which-alignment [other alignments ...  
16     ] ]  
17  
18 EXAMPLE:  
19     python3 hhpred_to_modeller.py CYP51F.hhpred 1 2  
20  
21  
22 CATEGORIES: core/data/io/read_hhpred  
23 KEYWORDS: HHPred; comparative modelling  
24 GROUP: File processing; Format conversion  
25  
26 """)  
27     sys.exit()  
28  
29 alignments = read_hhpred(sys.argv[1])  
30 which_ali = sys.argv[2:] if len(sys.argv) > 2 else [1]  
31 print(len(alignments), "alignments found in", sys.argv[1])  
32 for i in which_ali :  
33     i = int(i)  
34     print("retriving alignment:", i, "as %d.pir" % (i))  
35     f = open("%d.pir" % (i), "w")  
36     f.write(create_pir_string(alignments[i-1], 80))  
37     f.close()
```



ligand_contacts.py

Finds contacts between a ligand molecule and a protein. This script reads one or more PDB files, extracts all ligands that matches a given three-letter code and finds contacts between a ligand molecule and a protein for given cutoff. The script can also detect plausible hydrogen bonds between a ligand and a protein, but user must provide two JSON dictionaries: of hydrogen bond donors and acceptors. Use ‘-’ (dash character) to omit either of the two files and provide just one of them Note, that both files with JSON must have .json extension, otherwise the script will attempt to load them as PDB

USAGE:

```
python3 ligand_contacts.py ligand distance [donors.json acceptors.json] input.pdb  
↪ [input2.pdb]
```

EXAMPLE:

```
python3 ligand_contacts.py HEM 3.5 5ofq.pdb 4rm4.pdb  
python3 ligand_contacts.py TDZ 3.5 donors.json acceptors.json 2vn0.pdb
```

Keywords:

- *PDB input*
- *ligand*
- *structural properties*

Categories:

- core/data/io/Pdb

Input files:

- 5ofq.pdb
- 4rm4.pdb
- 2vn0.pdb

Output files:

- stdout.out

Program source:

```
1 import sys, math, json  
2  
3 from pybioshell.core.data.io import Pdb  
4 from pybioshell.utils import LogManager  
5 from pybioshell.core.chemical import monomer_type_name, HydrogenBondFilter  
6  
7 LogManager.INFO()  
8
```

(continues on next page)

(continued from previous page)

```

9  if len(sys.argv) < 4:
10     print("""
11
12 Finds contacts between a ligand molecule and a protein.
13
14 This script reads one or more PDB files, extracts all ligands that matches a given
15 three-letter code and finds contacts between a ligand molecule and a protein for ↴
16 ↴given cutoff.
17 The script can also detect plausible hydrogen bonds between a ligand and a protein, ↴
18 ↴but user
19 must provide two JSON dictionaries: of hydrogen bond donors and acceptors. Use '-' ↴
20 ↴(dash character)
21 to omit either of the two files and provide just one of them
22
23 Note, that both files with JSON must have .json extension, otherwise the script will ↴
24 ↴attempt
25 to load them as PDB
26
27 USAGE:
28     python3 ligand_contacts.py ligand distance [donors.json acceptors.json] input.pdb ↴
29 ↴[input2.pdb]
30
31
32 EXAMPLE:
33     python3 ligand_contacts.py HEM 3.5 5ofq.pdb 4rm4.pdb
34     python3 ligand_contacts.py TDZ 3.5 donors.json acceptors.json 2vn0.pdb
35
36 CATEGORIES: core/data/io/Pdb
37 KEYWORDS: PDB input; ligand; structural properties
38 GROUP: Structure calculations
39
40 """
41     sys.exit()
42
43 cutoff = float(sys.argv[2])
44
45 first_pdb = 3
46 extra_acceptors, extra_donors = {}, {}
47 if sys.argv[3] == '-':
48     first_pdb = 5
49 elif sys.argv[3].endswith(".json"):
50     extra_donors = json.loads(open(sys.argv[3]).read())
51     first_pdb = 5
52
53 if sys.argv[4].endswith(".json"):
54     extra_acceptors = json.loads(open(sys.argv[4]).read())
55
56 hb_filter = HydrogenBondFilter()
57 for code3 in extra_acceptors.keys():
58     for atom_name in extra_acceptors[code3]:
59         hb_filter.add_acceptor_definition(code3, atom_name)
60 for code3 in extra_donors.keys():
61     for atom_name in extra_donors[code3]:
62         hb_filter.add_donor_definition(code3, atom_name)
63
64 for pdb_code in sys.argv[first_pdb:]: # --- Iterate over PDB input files

```

(continues on next page)

(continued from previous page)

```

61     if len(sys.argv[first_pdb:]) > 1: print("# Pdb file %s" % (pdb_code.split("/")[-
62         1].split(".") [0]))
63     pdb = Pdb(pdb_code, "", False)
64     print(" ---- ligand --- | ----- partner ----- | distance")
65     print("c res id atname | c res id type atname | in Angstrom")
66
66     for m in range(pdb.count_models()): # --- Iterate over all models in the input_
67         file
68         if pdb.count_models() > 1: print("# Model %d" % (i + 1))
69         structure = pdb.create_structure(m)
70
70         for ic in range(structure.count_chains()):
71             lig_chain = structure[ic]
72             for ir in range(lig_chain.count_residues()):
73                 ligand = lig_chain[ir]
74                 code3 = ligand.residue_type().code3
75
76                 if code3 != sys.argv[1]: continue # Skip other ligands
77                 for iic in range(structure.count_chains()):
78                     other_chain = structure[iic]
79                     for r in range(other_chain.count_residues()): # ----Iterate over_
80                         residues
81                         res = other_chain[r]
82                         if res == ligand: continue
83                         d = res.min_distance(ligand) # ---- If residue is close_
84                         enough to ligand
85                         if d < cutoff:
86                             for ilig in range(ligand.count_atoms()):
87                                 for ioth in range(res.count_atoms()):
88                                     ligand_atom = ligand[ilig]
89                                     other_atom = res[ioth]
90                                     if ligand_atom.distance_to(other_atom) <= cutoff:
91                                         extras = ""
92                                         if hb_filter(ligand_atom, other_atom, ligand_
93                                         atom.distance_to(other_atom)):
94                                             extras = "HYDROGEN_BOND"
95                                             print("%s %3s %4d %4s      %s %3s %4d %6s
96                                         %4s %6.3f %s" % (ligand.owner().id(),
97                                         ligand.residue_type().code3, ligand.
98                                         id(), ligand_atom.atom_name(),
99                                         res.owner().id(), res.residue_type().
100                                         code3, res.id(),
101                                         other_atom.atom_name(),
102                                         ligand_atom.distance_to(other_atom),
103                                         extras))

```



ligand_crmsd_on_cofactor.py

Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on all atoms of a ligand conformations after superimposition based on another group, e.g. a cofactor. This script has been used in P450 analysis project: all PDB deposits with a drug (e.g. itraconazole) were pulled from PDB. For each pair of structures, the optimal superimposition for haeme groups is found. Then the very transformation is used to transform coordinates a molecule of a drug and to compute crmsd on the two itraconazole molecules.

USAGE:

```
python3 ligand_crmsd_on_cofactor.py cofactor-code3 ligand-code3 input1.pdb [input2.  
→pdb]
```

EXAMPLE:

```
python3 crmsd_on_ligands.py HEM 1YN 5ofq.pdb 4rm4.pdb
```

Keywords:

- *PDB input*
- *ligand*
- *crmsd*

Categories:

- core/calc/structural/transformations/CrmsdOnVec3

Input files:

- 5ESK.pdb
- 5JLC.pdb
- 5ESL.pdb
- 4ZE2.pdb
- 4ZDY.pdb
- 5V5Z.pdb
- 5ESG.pdb
- 5EQB.pdb
- 5ESH.pdb
- 6AYC.pdb

Output files:

- 1YN-by-HEM.pdb
- stdout.out

Program source:

```

1 import sys, math
2
3 sys.path.append("../../../../../bin/")
4
5 from pybioshell.core.data.basic import Vec3
6 from pybioshell.core.data.io import Pdb
7 from pybioshell.std import vector_core_data_basic_Vec3
8 from pybioshell.core.calc.structural.transformations import *
9 from pybioshell.utils import LogManager
10
11 LogManager.INFO()
12
13 if len(sys.argv) < 3:
14     print("""
15
16 Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on all atoms of a
17 ↪ligand conformations
18 after superimposition based on another group, e.g. a cofactor.
19
20 This scripts has been used in P450 analysis project: all PDB deposits with a drug (e.
21 ↪g. itraconazole) were pulled from PDB
22 For each pair of structures, the optimal superimposition for haeme groups is found.
23 ↪Then the very transformation
24 is used to transfrom coordinates a molecule of a drug and to compute crsmd on the two
25 ↪itraconazole molecules
26
27 USAGE:
28     python3 ligand_crmsd_on_cofactor.py cofactor-code3 ligand-code3 input1.pdb
29 ↪[input2.pdb]
30
31 EXAMPLE:
32     python3 crmsd_on_ligands.py HEM 1YN 5ofq.pdb 4rm4.pdb
33
34 CATEGORIES: core/calc/structural/transformations/CrmsdOnVec3
35 KEYWORDS: PDB input; ligand; crmsd
36 GROUP: Structure calculations
37
38 """
39
40 rms = CrmsdOnVec3()
41
42 class Entry:
43
44     def __init__(self, code, structure):
45
46         self.pdb_code = code # --- PDB code for every input structure: for
47 ↪informative output
48         self.structure = structure # --- contains all input structures
49         self.ligand = None
50         self.cofactor = None
51         self.atoms_superimposed = vector_core_data_basic_Vec3() # --- a list of
52 ↪atoms to define rototranslation

```

(continues on next page)

(continued from previous page)

```

50         self.atoms_crmsd = vector_core_data_basic_Vec3() # --- a list of atoms to_
→compute crmsd
51
52     def is_OK(self):
53         return self.ligand and self.cofactor
54
55     def add_ligand(self,ligand):
56         for ia in range(ligand.count_atoms()):
57             e.atoms_crmsd.append(ligand[ia])
58         self.ligand = ligand
59
60     def add_cofactor(self,cofactor):
61         for ia in range(cofactor.count_atoms()):
62             e.atoms_superimposed.append(cofactor[ia])
63         self.cofactor = cofactor
64
65 rototranslation_code3 = sys.argv[1]
66 crmsd_code3 = sys.argv[2]
67
68 entries = []
69 for pdb_code in sys.argv[3:]:
70     pdb = Pdb(pdb_code, "is_not_alternative is_not_water", False)
71     structure = pdb.create_structure(0)
72     for ic in range(structure.count_chains()):
73         chain = structure[ic]
74         # start from chain.terminal_residue_index() + 1 if you are sure the PDB file_
→has TER lines
75         e = Entry(pdb_code.split("/")[-1], structure)
76         for ir in range(0, chain.size()):
77             code3 = chain[ir].residue_type().code3
78             if code3 == rototranslation_code3: e.add_cofactor(chain[ir])
79             elif code3 == crmsd_code3: e.add_ligand(chain[ir])
80             if e.is_OK(): entries.append(e)
81
82 tmp_vec = Vec3()
83 output = open("%s-by-%s.pdb" % (crmsd_code3, rototranslation_code3), "w")
84 n_superimposed = len(entries[0].atoms_superimposed)
85 n_crmsd = len(entries[0].atoms_crmsd)
86 for ei in entries:
87     for ej in entries:
88         if ei == ej: continue
89         try :
90             if len(ei.atoms_superimposed) != len(ej.atoms_superimposed):
91                 print("superimposed sets differ in size")
92                 continue
93             crmsd_val_1 = rms.crmsd(ei.atoms_superimposed, ej.atoms_superimposed, n_
→superimposed, True)
94             if len(ei.atoms_crmsd) != len(ej.atoms_crmsd):
95                 print("crmsd sets differ in size")
96                 continue
97             crmsd_val_2 = rms.calculate_crmsd_value(ei.atoms_crmsd, ej.atoms_crmsd, n_
→crmsd)
98             print("%s %4d %3s %c - %s %4d %3s %c : %7.3f %7.3f" %
99                  (ei.pdb_code, ei.ligand.id(), ei.ligand.residue_type().code3, ei.
→ligand.owner().id(),
100                 ej.pdb_code, ej.ligand.id(), ej.ligand.residue_type().code3, ej.
→ligand.owner().id(), crmsd_val_1, crmsd_val_2))

```

(continues on next page)

(continued from previous page)

```
101     if ej == entries[0]:
102         output.write("MODEL      %1\n")
103         for ai in range(ei.ligand.count_atoms()):
104             rms.apply(ei.ligand[ai])
105             output.write(ei.ligand[ai].to_pdb_line() + "\n")
106             output.write("ENDMDL\n")
107     except:
108         pass
109
110 output.write("MODEL      %d\n" % (1))
111 for ai in range(entries[0].ligand.count_atoms()):
112     output.write(entries[0].ligand[ai].to_pdb_line() + "\n")
113 output.write("ENDMDL\n")
114
115 output.write("MODEL      %d\n" % (2))
116 for ai in range(entries[0].cofactor.count_atoms()):
117     output.write(entries[0].cofactor[ai].to_pdb_line() + "\n")
118 output.write("ENDMDL\n")
119
120 output.close()
```



ligand_rototranslation.py

Calculates rototranslation transformation that superimposes a ligand molecule from one reference frame to another.
As an output, prints the rototranslation

USAGE:

```
python3 ligand_rototranslation.py ligand-code3 reference.pdb input1.pdb [input2.pdb  
↔...]
```

EXAMPLE:

```
python3 ligand_rototranslation.py CAM 2m56-ref.pdb 00199.pdb 00963.pdb 04473.pdb
```

Keywords:

- *PDB input*
- *ligand*
- *crmsd*

Categories:

- core/calc/structural/transformations/CrmsdOnVec3

Input files:

- 04473.pdb
- 2m56-ref.pdb
- 00199.pdb
- 00963.pdb

Output files:

- stdout.out

Program source:

```
1 import sys, math
2
3 sys.path.append("../../../../../bin/")
4
5 from pybioshell.core.data.basic import Vec3
6 from pybioshell.core.data.io import Pdb
7 from pybioshell.core.data.structural.selectors import SelectResidueByName
8 from pybioshell.core.protocols import copy_selected_atoms, copy_selected_coordinates
9 from pybioshell.core.calc.structural.transformations import *
10 from pybioshell.utils import LogManager
11
```

(continues on next page)

(continued from previous page)

```

12 LogManager.INFO()
13 IF_ROW_OUTPUT = False
14
15 if len(sys.argv) < 3:
16     print("""
17
18 Calculates rototranslation transformation that superimposes a ligand molecule from
19 one reference frame to another.
20 As an output, prints the rototranslation
21
22 USAGE:
23     python3 ligand_rototranslation.py  ligand-code3  reference.pdb input1.pdb [input2.
24     pdb ...]
25
26 EXAMPLE:
27     python3 ligand_rototranslation.py CAM 2m56-ref.pdb 00199.pdb 00963.pdb 04473.pdb
28
29 CATEGORIES: core/calc/structural/transformations/CrmsdOnVec3
30 KEYWORDS: PDB input; ligand; crmsd
31 GROUP: Structure calculations
32
33 """
34     sys.exit()
35
36 rms = CrmsdOnVec3()
37 select_ligand = SelectResidueByName(sys.argv[1])
38 ref_strctr = Pdb(sys.argv[2], "").create_structure(0)
39 ref_atoms = copy_selected_coordinates(ref_strctr, select_ligand)
40
41 for f_model in sys.argv[3:]:
42     strctr = Pdb(f_model, "").create_structure(0)
43     ligand_atoms = copy_selected_coordinates(strctr, select_ligand)
44     crsmd_val = rms.crmsd(ligand_atoms, ref_atoms, len(ref_atoms), True) #_
45     # crsmd_val = rms.crmsd(ref_atoms, ligand_atoms, len(ref_atoms), True) #_
46     # superimpose a model onto a reference
47     if IF_ROW_OUTPUT:
48         print("%7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %8.3f %8.3f %8.
49         # superimpose a reference onto a model
50         # crsmd_val = rms.crmsd(ref_atoms, ligand_atoms, len(ref_atoms), True) #_
51         # superimpose a reference onto a model
52         if IF_ROW_OUTPUT:
53             print("%7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %8.3f %8.3f %8.
54             # superimpose a reference onto a model
55             print(rms)

```



list_pdb_ligands.py

Prints names of ligand molecules found in a given PDB file.

A ligand is defined as a residue located after TER field in a PDB chain

USAGE:

```
python3 list_pdb_ligands.py input.pdb
```

EXAMPLE:

```
python3 list_pdb_ligands.py 5edw.pdb
```

Keywords:

- *PDB input*
- *ligand*

Categories:

- core/data/io/find_pdb

Input files:

- 5ofq.pdb
- 5edw.pdb

Output files:

- stdout.out

Program source:

```
1 import sys
2
3 from pybioshell.core.data.io import find_pdb
4
5
6 if len(sys.argv) < 2 :
7     print("""
8
9 Prints names of ligand molecules found in a given PDB file.
10
11 A ligand is defined as a residue located after TER field in a PDB chain
12
13
14 USAGE:
15     python3 list_pdb_ligands.py input.pdb
16
17
```

(continues on next page)

(continued from previous page)

```

18 EXAMPLE:
19     python3 list_pdb_ligands.py 5edw.pdb
20
21
22 CATEGORIES: core/data/io/find_pdb
23 KEYWORDS: PDB input; ligand
24 GROUP: File processing; Data filtering
25
26 """)
27     sys.exit()
28
29 for pdb_fname in sys.argv[1:]:
30     structure = find_pdb(pdb_fname, "./").create_structure(0)
31     for ic in range(structure.count_chains()):
32         chain = structure[ic]
33         #print(chain.terminal_residue_index())
34
35         for ir in range(chain.terminal_residue_index() + 1, chain.size()):
36             resid = chain[ir]
37             code3 = resid.residue_type().code3
38             if resid.residue_type().code3 == "HOH": continue # Skip water molecules,
→they are so obvious and abundant
39             formula = structure.formula(code3)
40             hetname = structure.hetname(code3)
41             print("%3s %c %4d %s %s" %(code3, chain.id(), resid.id(), formula.strip(),
→ hetname.strip()))

```



msa_to_profile.py

Reads a multiple sequence alignment (MSA) (in .aln format) produced by ClustalO, calculates a sequence profile and prints in a flat tabular format

USAGE:

```
python3 msa_to_profile.py input.aln
```

EXAMPLE:

```
python3 msa_to_profile.py cyped.CYP109.aln
```

Keywords:

- *MSA*
- *sequence profile*
- *Format conversion*

Categories:

- core/data/io/read_clustalw_file

Input files:

- cyped.CYP109.aln

Output files:

- stdout.out

Program source:

```

1 import sys
2
3 from pybioshell.std import vector_std_shared_ptr_core_data_Sequence_t
4 from pybioshell.core.data.io import read_clustalw_file
5 from pybioshell.core.data.sequence import SequenceProfile
6
7
8 if len(sys.argv) < 2 :
9     print("""
10
11 Reads a multiple sequence alignment (MSA) (in .aln format) produced by ClustalO,
12 calculates a sequence profile and prints in a flat tabular format
13
14
15 USAGE:
16     python3 msa_to_profile.py input.aln
17

```

(continues on next page)

(continued from previous page)

```
18
19 EXAMPLE:
20     python3 msa_to_profile.py cyped.CYP109.aln
21
22
23 CATEGORIES: core/data/io/read_clustalw_file
24 KEYWORDS: MSA; sequence profile; Format conversion
25 GROUP: Sequence calculations;
26
27 """)
28     sys.exit()
29
30 msa = vector_std_shared_ptr_core_data_sequence_Sequence_t()
31 read_clustalw_file(sys.argv[1], msa)
32 profile = SequenceProfile(msa[0], SequenceProfile.aaOrderByPropertiesGapped(), msa)
33 profile.write_table()
```



partial_thread.py

Reads a FASTA file with two aligned sequences: a query and a template, and a template structure. Prints a partial thread of the template, i.e. the fragment of a template structure that is aligned with a query

USAGE:

```
python3 partial_thread.py ali.fasta template.pdb [chain-id]
```

EXAMPLE:

```
python3 partial_thread.py 2azaA_2pcyA-ali.fasta 2aza.pdb A
```

Keywords:

- *FASTA input*
- *sequence*

Categories:

- core/alignment

Input files:

- 2azaA_2pcyA-ali.fasta
- 2pcy.pdb
- 2aza.pdb

Output files:

- 2azaA-thread.pdb
- 2pcyA-thread.pdb

Program source:

```
1 import sys
2
3 from pybioshell.core.data.io import read_fasta_file, Pdb
4
5
6 if len(sys.argv) < 3 :
7     print("""
8
9 Reads a FASTA file with two aligned sequences: a query and a template, and a template_
10    ↪structure.
11 Prints a partial thread of the template, i.e. the fragment of a template structure_
12    ↪that is
aligned with a query
```

(continues on next page)

(continued from previous page)

```

13
14 USAGE:
15     python3 partial_thread.py ali.fasta template.pdb [chain-id]
16
17
18 EXAMPLE:
19     python3 partial_thread.py 2azaA_2pcyA-ali.fasta 2aza.pdb A
20
21
22 CATEGORIES: core/alignment
23 KEYWORDS: FASTA input; sequence
24 GROUP: File processing; Data filtering
25
26 """)
27 sys.exit()
28
29 def select_aligned_residues(query_seq, template_seq, template_chain):
30     j = 0
31     out = []
32     for i in range(len(query_seq)):
33         if template_seq[i] != '-':
34             if query_seq[i] != '-': out.append(template_chain[j])
35             j += 1
36     return out
37
38
39 def print_atoms(residues):
40     for r in residues:
41         for i in range(r.count_atoms()):
42             print(r[i].to_pdb_line())
43
44
45 fasta = read_fasta_file(sys.argv[1])
46 seq1 = fasta[0].sequence
47 seq1_gapless = fasta[0].create_ungapped_sequence().sequence
48 seq2 = fasta[1].sequence
49 seq2_gapless = fasta[1].create_ungapped_sequence().sequence
50
51 print(fasta[0].sequence,fasta[1].sequence)
52
53 strctr = Pdb(sys.argv[2], "").create_structure(0)
54 if len(sys.argv) > 3:
55     chain = strctr.get_chain(sys.argv[3])
56 else:
57     chain = strctr[0]
58 seq_pdb = chain.create_sequence().sequence
59 if seq_pdb == seq1_gapless:
60     atoms = select_aligned_residues(seq2, seq1, chain)
61     print_atoms(atoms)
62 elif seq_pdb == seq2_gapless:
63     atoms = select_aligned_residues(seq1, seq2, chain)
64     print_atoms(atoms)
65 else:
66     print("template sequence can't be identified in the given alignment")
67
68
69

```



pdb_from_clustering.py

Extracts PDB clusters from clustering results produced by ap_cluster_ligands output SEE:

ap_cluster_ligands program to see how to run clustering

USAGE:

```
python3 pdb_from_clusters.py clustering_output.txt ligand_code
```

EXAMPLE:

```
python3 pdb_from_clustering.py clustering_output.txt Clo
```

Keywords:

- *PDB output*
- *clustering*

Categories:

- core/data/io/Pdb

Input files:

- 00040_HEM_Clotrimazole.pdb_9149.pdb
- 00040_HEM_Clotrimazole.pdb_4439.pdb
- 00040_HEM_Clotrimazole.pdb_5452.pdb
- 00040_HEM_Clotrimazole.pdb_4313.pdb
- 00040_HEM_Clotrimazole.pdb_6297.pdb
- 00040_HEM_Clotrimazole.pdb_8420.pdb
- 00040_HEM_Clotrimazole.pdb_2785.pdb
- 00040_HEM_Clotrimazole.pdb_5724.pdb
- 00040_HEM_Clotrimazole.pdb_6861.pdb
- 00040_HEM_Clotrimazole.pdb_1281.pdb
- 00040_HEM_Clotrimazole.pdb_3187.pdb
- 00040_HEM_Clotrimazole.pdb_6644.pdb
- 00040_HEM_Clotrimazole.pdb_4215.pdb
- 00040_HEM_Clotrimazole.pdb_3818.pdb
- 00040_HEM_Clotrimazole.pdb_2412.pdb
- 00040_HEM_Clotrimazole.pdb_528.pdb
- 00040_HEM_Clotrimazole.pdb_2169.pdb
- clusters-10.00.txt

- 00040_HEM_Clotrimazole.pdb_1614.pdb
- 00040_HEM_Clotrimazole.pdb_987.pdb
- 00040_HEM_Clotrimazole.pdb_4992.pdb
- 00040_HEM_Clotrimazole.pdb_4687.pdb
- 00040_HEM_Clotrimazole.pdb_9108.pdb
- 00040_HEM_Clotrimazole.pdb_8537.pdb
- 00040_HEM_Clotrimazole.pdb_7150.pdb
- 00040_HEM_Clotrimazole.pdb_4295.pdb

Output files:

- c29.pdb
- c18.pdb
- c38.pdb

Program source:

```
1 import sys
2
3 from pybioshell.core.data.io import Pdb
4
5 if len(sys.argv) < 2 :
6     print("""
7
8 Extracts PDB clusters from clustering results produced by ap_cluster_ligands output
9
10 SEE:
11     ap_cluster_ligands program to see how to run clustering
12
13 USAGE:
14     python3 pdb_from_clusters.py clustering_output.txt ligand_code
15
16
17 EXAMPLE:
18     python3 pdb_from_clustering.py clustering_output.txt Clo
19
20
21 CATEGORIES: core/data/io/Pdb
22 KEYWORDS:    PDB output; clustering
23 GROUP:       File processing; Structure calculations
24
25 """)
26     sys.exit()
27
28 chain_ids = "ABCDEFGHIJKLMNPQRSTUVWXYZ1234567890abcdefghijklmnopqrstuvwxyz"
29 clusters_file = open(sys.argv[1])
30 ligand_code = sys.argv[2]
31 iline = 0
32 for line in clusters_file:
33     iline += 1
```

(continues on next page)

(continued from previous page)

```

34 tokens = line.strip().split()
35 outp = open("c"+str(iLine)+tokens[0]+".pdb", "w")
36 p = Pdb(tokens[1], "", False)
37 s = p.create_structure(0)
38 i_chain = 0
39 for ic in range(s.count_chains()):
40     c = s[ic]
41     for ir in range(c.count_residues()):
42         r = c[ir]
43         for ia in range(r.count_atoms()):
44             a = r[ia]
45             outp.write(a.to_pdb_line() + "\n")
46
47 for fname in tokens[2:]:
48     p = Pdb(fname, "", False)
49     s = p.create_structure(0)
50     for ic in range(s.count_chains()):
51         c = s[ic]
52         for ir in range(c.count_residues()):
53             r = c[ir]
54             if r.residue_type().code3 != ligand_code: continue
55             r.owner().id(chain_ids[i_chain])
56             for ia in range(r.count_atoms()):
57                 a = r[ia]
58                 outp.write(a.to_pdb_line() + "\n")
59
outp.close()

```



pdb_info.py

Reads a PDB file and extracts some basic information from its header

USAGE:

```
python3 pdb_info.py input.pdb [input2.pdb]
```

EXAMPLE:

```
python3 pdb_info.py 2kwi.pdb
```

Keywords:

- *PDB input*
- :ref:``

Categories:

- core/data/io/Pdb

Input files:

- 2kwi.pdb
- 2gb1.pdb
- 4mcb.pdb
- 5edw.pdb
- 6xra.pdb

Output files:

- stdout.out

Program source:

```

1 import sys
2
3 from pybioshell.core.data.io import Pdb
4
5 if len(sys.argv) < 2 :
6     print("""
7
8 Reads a PDB file and extracts some basic information from its header
9
10
11 USAGE:
12     python3 pdb_info.py input.pdb [input2.pdb]
13

```

(continues on next page)

(continued from previous page)

```
14
15 EXAMPLE:
16     python3 pdb_info.py 2kwi.pdb
17
18
19 CATEGORIES: core/data/io/Pdb
20 KEYWORDS: PDB input;
21 GROUP: File processing;
22
23 """
24     sys.exit()
25
26 for pdb_fname in sys.argv[1:]:
27     s = Pdb(pdb_fname, "", True).create_structure(0)
28
29     print(s.classification())
30     print("protein", s.code(), "has", s.count_chains(), "chain(s)", s.count_
31 →residues(),
32         "residues and", s.count_atoms(), "atoms\n")
33     print("deposited : ", s.deposition_date())
34     print("Is XRAY? : ", (s.is_xray()))
35     print("Is NMR? : ", (s.is_nmr()))
36     print("Is EM? : ", (s.is_em()))
37     print("resolution: ", s.resolution())
38     print("R-value : ", s.r_value())
39     print("R-free : ", s.r_free())
40     if len(s.keywords()) > 0:
41         print("Keywords : ", s.keywords()[0], end="")
42         for k in s.keywords()[1:]:
43             print(", ", k, end="")
44     print()
```



pdb_to_fasta.py

Extracts amino acid (or nucleotide) sequence from a PDB file. Note, that by default ligands are not included in the output sequence even if they are amino acids (e.g. 7dk3 deposit)

USAGE:

```
python3 pdb_to_fasta.py input.pdb [input2.pdb]
```

EXAMPLE:

```
python3 pdb_to_fasta.py 2kwi.pdb
```

Keywords:

- *PDB input*
- *FASTA*
- *Format conversion*

Categories:

- core/data/io/find_pdb

Input files:

- 2kwi.pdb
- 2gb1.pdb
- 4mcb.pdb
- 5edw.pdb

Output files:

- stdout.out

Program source:

```
1 import sys
2
3 from pybioshell.core.data.io import find_pdb
4
5 # change that setting to False to include ligands in the output sequence
6 IF_EXCLUDE_LIGANDS = True
7
8 if len(sys.argv) < 2 :
9     print("""
10
11 Extracts amino acid (or nucleotide) sequence from a PDB file.
```

(continues on next page)

(continued from previous page)

```
13 Note, that by default ligands are not included in the output sequence even if they
14 are amino acids (e.g. 7dk3 deposit)
15
16 USAGE:
17     python3 pdb_to_fasta.py input.pdb [input2.pdb]
18
19
20 EXAMPLE:
21     python3 pdb_to_fasta.py 2kwi.pdb
22
23
24 CATEGORIES: core/data/io/find_pdb
25 KEYWORDS: PDB input; FASTA; Format conversion
26 GROUP: File processing; Format conversion
27
28     """
29     sys.exit()
30
31 for pdb_fname in sys.argv[1:] :
32     structure = find_pdb(pdb_fname, "./").create_structure(0)
33     for ic in range(structure.count_chains()) :
34         chain = structure[ic]
35         print(">", structure.code(), chain.id())
36         print(chain.create_sequence(IF_EXCLUDE_LIGANDS).sequence)
```



pdb_to_seq.py

Converts sequence in a PDB format to SEQ format.

USAGE:

```
python3 pdb_to_ss2.py input.pdb [input.ss2]
```

EXAMPLE:

```
python3 pdb_to_ss2.py 2gb1.pdb 2gb1.out
python3 pdb_to_ss2.py 2kwi.pdb
```

Keywords:

- *PDB input*
- *secondary structure*
- *Format conversion*

Categories:

- core/data/io/write_seq

Input files:

- 2kwi.pdb
- 2gb1.pdb
- 4mcb.pdb
- 5edw.pdb

Output files:

- 2gb1.ss2
- stdout.out

Program source:

```

1 import sys
2
3 from pybioshell.core.data.io import Pdb,create_seq_string
4
5 if len(sys.argv) < 2 :
6     print("""
7
8 Converts sequence in a PDB format to SEQ format.
9
10 USAGE:
11
```

(continues on next page)

(continued from previous page)

```
12     python3 pdb_to_ss2.py input.pdb [input.ss2]
13
14
15 EXAMPLE:
16     python3 pdb_to_ss2.py 2gb1.pdb 2gb1.out
17     python3 pdb_to_ss2.py 2kwi.pdb
18
19 CATEGORIES: core/data/io/write_seq
20 KEYWORDS: PDB input; secondary structure; Format conversion
21 GROUP: File processing; Format conversion
22 """
23     sys.exit()
24
25 structure = Pdb(sys.argv[1], "").create_structure(0)
26 outname = sys.argv[2] if len(sys.argv) > 2 else "stdout"
27 for ic in range(structure.count_chains()):
28     chain = structure[ic]
29     ss = chain.create_sequence()
30     a = create_seq_string(ss)
31     print(a)
32
```



pdb_to_ss2.py

Extracts amino acid (or nucleotide) sequence from a PDB file.

USAGE:

```
python3 pdb_to_ss2.py input.pdb [input.ss2]
```

EXAMPLE:

```
python3 pdb_to_ss2.py 2gb1.pdb 2gb1.out  
python3 pdb_to_ss2.py 2kwi.pdb
```

Keywords:

- *PDB input*
- *secondary structure*
- *Format conversion*

Categories:

- core/data/io/write_ss2

Input files:

- 2kwi.pdb
- 2gb1.pdb
- 4mcb.pdb
- 5edw.pdb

Output files:

- 2gb1.ss2
- stdout.out

Program source:

```
1 import sys  
2  
3 from pybioshell.core.data.io import find_pdb, write_ss2  
4  
5  
6 if len(sys.argv) < 2 :  
7     print("""  
8 Extracts amino acid (or nucleotide) sequence from a PDB file.  
9  
10  
11
```

(continues on next page)

(continued from previous page)

```
12 USAGE:  
13     python3 pdb_to_ss2.py input.pdb [input.ss2]  
14  
15  
16 EXAMPLE:  
17     python3 pdb_to_ss2.py 2gb1.pdb 2gb1.out  
18     python3 pdb_to_ss2.py 2kwi.pdb  
19  
20 CATEGORIES: core/data/io/write_ss2  
21 KEYWORDS: PDB input; secondary structure; Format conversion  
22 GROUP: File processing; Format conversion  
23 """)  
24     sys.exit()  
25  
26 structure = find_pdb(sys.argv[1], "./").create_structure(0)  
27 outname = sys.argv[2] if len(sys.argv) > 2 else "stdout"  
28 for ic in range(structure.count_chains()):  
29     chain = structure[ic]  
30     ss = chain.create_sequence()  
31     write_ss2(ss,outname)  
32     print() # Print empty line to separate chain: note that it works only when  
→printed to stdout  
33
```



radial_distribution_function.py

Calculates radial distribution function for a trajectory from a molecular simulation.

USAGE:

```
python3 radial_distribution_function.py input_tra.pdb cutoff
```

EXAMPLE:

```
python3 radial_distribution_function.py ar_tra.pdb 27.6214
```

Keywords:

- *PDB input*
- *structural properties*

Categories:

- core/data/basic/Vec3Cubic

Input files:

- ar_tra.pdb

Output files:

- stdout.out

Program source:

```

1 import sys, math
2
3 from pybioshell.core.data.io import find_pdb
4 from pybioshell.core.data.basic import Vec3Cubic
5
6 from pybioshell.std import vector_core_data_basic_Vec3Cubic
7
8 if len(sys.argv) < 3 :
9     print("""
10
11 Calculates radial distribution function for a trajectory from a molecular simulation. ↵
12
13
14 USAGE:
15     python3 radial_distribution_function.py input_tra.pdb cutoff
16
17
18 EXAMPLE:
19     python3 radial_distribution_function.py ar_tra.pdb 27.6214

```

(continues on next page)

(continued from previous page)

```
20
21
22 CATEGORIES: core/data/basic/Vec3Cubic
23 KEYWORDS: PDB input; structural properties
24 GROUP: Structure calculations;
25
26 """)
27     sys.exit()
28
29
30 pdb = find_pdb(sys.argv[1], "./")
31 n_atoms = pdb.count_atoms(0)
32 cutoff = float(sys.argv[2])
33
34 Vec3Cubic.set_box_len(cutoff)
35 xyz = vector_core_data_basic_Vec3Cubic()
36 for i in range(n_atoms) : xyz.append( Vec3Cubic() )
37 histogram = [0 for i in range(121)]
38 for i_model in range(0, pdb.count_models()) :
39     # print i_model
40     pdb.fill_structure(i_model, xyz)
41     for i_atom in range(n_atoms) :
42         for j_atom in range(i_atom) :
43             d = xyz[i_atom].closest_distance_square_to(xyz[j_atom], 12*12)
44             if d < 144 : histogram[ int(math.sqrt(d)*10) ] += 1
45
46 for i in range(1,120) : print("%5f %7.2f" % (i*0.1, histogram[i]/(i*i*0.01)))
```



read_scorefile.py

Simple example that parses a score file (Rosetta output)

USAGE:

```
python3 read_scorefile.py score-file
```

EXAMPLE:

```
python3 read_scorefile.py scores.sf
```

Keywords:

- scorefile input
- :ref:``

Categories:

- core/data/io/read_scorefile

Input files:

- 1pgxA-abinitio.fsc
- 1pgxA-abinitio-bis.fsc

Output files:

- stdout.out

Program source:

```
1 import sys
2 from pybioshell.core.data.io import read_scorefile
3
4 if len(sys.argv) < 2 :
5     print("""
6
7 Simple example that parses a score file (Rosetta output)
8
9
10 USAGE:
11     python3 read_scorefile.py score-file
12
13 EXAMPLE:
14     python3 read_scorefile.py scores.sf
15
16 CATEGORIES: core/data/io/read_scorefile
17 KEYWORDS:   scorefile input;
18 GROUP:      File processing; Format conversion
```

(continues on next page)

(continued from previous page)

```
19      """)
20      sys.exit()
21
22 sf = read_scorefile(sys.argv[1])
23
24 print("Number of rows: %d" % len(sf))
25 print("Number of columns: %d" % sf[0].size())
26 print("Known columns:")
27 for i in range(sf[0].size()):
28     print(sf.column_name(i))
29
```



rg.py

Calculates the radius of gyration from given pdb file coordinates.

USAGE:

```
python3 rg.py input.pdb
```

EXAMPLE:

```
python3 rg.py 1cey.pdb
```

Keywords:

- *PDB input*
- *structural properties*

Categories:

- core/calc/structural/calculate_Rg_square

Input files:

- 1cey.pdb

Output files:

- stdout.out

Program source:

```

1 import sys, math
2
3 from pybioshell.core.data.io import find_pdb
4 from pybioshell.core.data.basic import Vec3
5 from pybioshell.std import vector_core_data_basic_Vec3
6
7 from pybioshell.core.calc.structural import *
8 from pybioshell.utils import LogManager
9 LogManager.INFO()
10
11
12
13 if len(sys.argv) < 2 :
14     print("""
15
16 Calculates the radius of gyration from given pdb file coordinates.
17
18
19 USAGE:
20     python3 rg.py input.pdb

```

(continues on next page)

(continued from previous page)

```
21
22
23 EXAMPLE:
24     python3 rg.py 1cey.pdb
25
26
27 CATEGORIES: core/calc/structural/calculate_Rg_square
28 KEYWORDS: PDB input; structural properties
29 GROUP: Structure calculations;
30
31     """)
32     sys.exit()
33
34 for pdb_fname in sys.argv[1:] :
35     pdb=find_pdb(pdb_fname, "./")
36     n_atoms = pdb.count_atoms(0)
37
38     structure = pdb.create_structure(0)
39     models=[ ]
40
41     for i_model in range(0, pdb.count_models()) :
42         xyz=vector_core_data_basic_Vec3()
43         for i in range(n_atoms) : xyz.append( Vec3() )
44         models.append(xyz)
45
46     for i_model in range(0, pdb.count_models()) :
47         pdb.fill_structure(i_model, models[i_model])
48         try:
49             print("Rg for %s, model # %5d : %7.3f" % (pdb_fname.split("/")[-1].split("."
50             ")[0],i_model,
51             math.sqrt(calculate_Rg_square(models[i_model][0], models[i_model][n_atoms-
52             1]))))
53         except:
54             sys.stderr.write(str(sys.exc_info()[0])+" "+str(sys.exc_info()[1]))
55
```



superimpose_by_fragment.py

Superimposes protein structures based on a structural fragment.

This script superimposes all models given at command line (at least one) on the reference structure. The superimposition is based on C-alpha atoms of residues from %d to %d. If you need another fragment, change these values in the script!

USAGE:

```
python3 superimpose_by_fragment.py reference.pdb model1.pdb [model2.pdb...]
```

EXAMPLE:

```
python3 superimpose_by_fragment.py 2gb1.pdb 2gb1-model1.pdb 2gb1-model2.pdb
```

Keywords:

- *PDB input*
- *crmsd*

Categories:

- core/calc/structural/transformations/Crmsd

Input files:

- 2gb1-model3.pdb
- 2gb1-model1.pdb
- 2gb1-model4.pdb
- 2gb1.pdb
- 2gb1-model2.pdb

Output files:

- rot-2gb1-model1.pdb
- rot-2gb1-model4.pdb
- rot-2gb1-model2.pdb
- rot-2gb1-model3.pdb

Program source:

```
1 import sys, math, os
2
3 from pybioshell.core.data.io import Pdb, write_pdb
4 from pybioshell.core.data.basic import Vec3
5 from pybioshell.std import vector_core_data_basic_Vec3
```

(continues on next page)

(continued from previous page)

```

6   from pybioshell.core.calc.structural.transformations import *
7   from pybioshell.utils import LogManager
8
9   REFERENCE_FROM, REFERENCE_TO = 23, 32 # 25, 390
10
11 LogManager.INFO()
12
13 if len(sys.argv) < 3:
14     print("""
15
16 Superimposes protein structures based on a structural fragment.
17
18
19 This script superimposes all models given at command line (at least one) on the_
20     ↪reference structure.
21 The superimposition is based on C-alpha atoms of residues from %d to %d. If you need_
22     ↪another fragment,
23 change these values in the script!
24
25 USAGE:
26     python3 superimpose_by_fragment.py reference.pdb model1.pdb [model2.pdb...]
27
28 EXAMPLE:
29     python3 superimpose_by_fragment.py 2gb1.pdb 2gb1-model1.pdb 2gb1-model2.pdb
30
31
32 CATEGORIES: core/calc/structural/transformations/Crmsd
33 KEYWORDS: PDB input; crmsd
34 GROUP: Structure calculations
35
36
37 """ % (REFERENCE_FROM, REFERENCE_TO))
38 sys.exit()
39
40 rms = CrmsdOnVec3()
41
42 pdb = Pdb(sys.argv[1], "", False)                      # --- read the reference PDB file -_
43     ↪only C-alphas
44 structure = pdb.create_structure(0)
45 n_atoms = REFERENCE_TO - REFERENCE_FROM + 1
46 xyz = vector_core_data_basic_Vec3()                     # --- std::vector<
47     ↪of Vec3 object is required to calculate superimposition
48 for i in range(REFERENCE_FROM, REFERENCE_TO+1):          # --- fill the vector_
49     ↪with the selected reference coordinates
50     r = structure[0][i]                                    # --- i-th residue of the first_
51     ↪chain
52     xyz.append(r.find_atom(" CA "))
53
54 #out_fname = "rot.pdb"
55 for pdb_fname in sys.argv[2:]:                           # --- iterate_
56     ↪over all models
57     out_fname = "rot-" + pdb_fname.split(os.path.sep)[-1]
58     other_pdb = Pdb(pdb_fname, "", False)
59     other_structure = other_pdb.create_structure(0)
60     other_xyz = vector_core_data_basic_Vec3()             # --- container for_
61     ↪coordinates of a model

```

(continues on next page)

(continued from previous page)

```
56
57     try:
58         for i in range(REFERENCE_FROM, REFERENCE_TO+1):           # --- fill the vector
59             →with the selected coordinates
60             r = other_structure[0][i]                           # --- i-th residue of the
61             →first chain
62             other_xyz.append(r.find_atom(" CA "))
63             rms_val = rms.crmsd(xyz, other_xyz, n_atoms, True)
64             rms.apply_inverse(other_structure)
65             write_pdb(other_structure, out_fname, 0)
66     except:
67         sys.stderr.write(str(sys.exc_info()[0]) + " " + str(sys.exc_info()[1]))
```



tmscore.py

Calculates TMScore value on two or more structures. First file is a reference structure and the second can be multi-model pdb. Calculations is running between reference structure and every model from the second file.

USAGE:

```
python3 tmscore.py file1.pdb [file2.pdb...]
```

EXAMPLE:

```
python3 tmscore.py 2gb1-model1.pdb 2gb1-model2.pdb
```

Keywords:

- *PDB input*
- TMscore

Categories:

- core/calc/structural/transformations/TMscore

Input files:

- 2gb1-model3.pdb
- 2gb1-model1.pdb
- 2gb1-model4.pdb
- 1cey.pdb
- 2gb1-model2.pdb

Output files:

- stdout.out

Program source:

```
1 import sys, math
2
3 from pybioshell.core.data.io import Pdb
4 from pybioshell.core.data.basic import Vec3
5 from pybioshell.std import vector_core_data_basic_Vec3
6
7 from pybioshell.core.calc.structural.transformations import *
8 from pybioshell.utils import LogManager
9
10 LogManager.INFO()
11
12 if len(sys.argv) < 2:
```

(continues on next page)

(continued from previous page)

```

13     print """
14
15 Calculates TMScore value on two or more structures.
16 First file is a reference structure and the second can be multimodel pdb.
17 Calculations is running between reference structure and every model from the second
18 →file.
19
20 USAGE:
21     python3 tmscore.py file1.pdb [file2.pdb...]
22
23
24 EXAMPLE:
25     python3 tmscore.py 2gb1-model1.pdb 2gb1-model2.pdb
26
27
28 CATEGORIES: core/calc/structural/transformations/TMScore
29 KEYWORDS: PDB input; TMScore
30 GROUP: Structure calculations
31
32
33 """
34     sys.exit()
35
36
37 if len(sys.argv) == 3:
38
39     pdb = Pdb(sys.argv[1], "is_not_water", False)
40     n_atoms = pdb.count_atoms(0)
41     ref_structure = pdb.create_structure(0)
42     ref_xyz = vector_core_data_basic_Vec3()
43     for i in range(n_atoms): ref_xyz.append(Vec3())
44     pdb.fill_structure(0, ref_xyz)
45
46     pdb = Pdb(sys.argv[2], "is_not_water", False)
47     n_atoms = pdb.count_atoms(0)
48
49     structure = pdb.create_structure(0)
50     models = []
51
52     for i_model in range(0, pdb.count_models()):
53         xyz = vector_core_data_basic_Vec3()
54         for i in range(n_atoms): xyz.append(Vec3())
55         models.append(xyz)
56
57     for i_model in range(0, pdb.count_models()):
58         pdb.fill_structure(i_model, models[i_model])
59         tmscore = TMScore(models[i_model], ref_xyz)
60         try:
61             print("%2d %6.3f" % (i_model, tmscore.tmscore()))
62         except:
63             sys.stderr.write(str(sys.exc_info()[0]) + " " + str(sys.exc_info()[1]))

```



validate_saturated_ring6.py

Validates a hexagonal saturated ring, defined by 6 atoms.

USAGE:

```
python3 validate_saturated_ring6.py input.pdb ligand _atom1_ _atom2_ _atom3_ _atom4_ _
→_atom5_ _atom6_
```

EXAMPLE:

```
python3 validate_saturated_ring6.py 4jm3.pdb EPE _N1_ _C2_ _C3_ _N4_ _C5_ _C6_
```

Keywords:

- *PDB input*
- *structural properties*

Categories:

- core/calc/structural/SaturatedRing6Geometry

Input files:

- pdb6bge.ent.gz
- 4jm3.pdb

Output files:

- stdout.out

Program source:

```

1 import sys, math
2
3 from pybioshell.core.data.io import find_pdb
4 from pybioshell.core.calc.structural import SaturatedRing6Geometry
5
6 if len(sys.argv) < 3 :
7     print("""
8
9 Validates a hexagonal saturated ring, defined by 6 atoms.
10
11 USAGE:
12     python3 validate_saturated_ring6.py input.pdb ligand _atom1_ _atom2_ _atom3_ _
→_atom4_ _atom5_ _atom6_
13
14
15 EXAMPLE:
16     python3 validate_saturated_ring6.py 4jm3.pdb EPE _N1_ _C2_ _C3_ _N4_ _C5_ _C6_

```

(continues on next page)

(continued from previous page)

```
17
18 CATEGORIES: core/calc/structural/SaturatedRing6Geometry
19 KEYWORDS: PDB input; structural properties
20 GROUP: Structure calculations
21
22     """)
23     sys.exit()
24
25
26 pdb = find_pdb(sys.argv[1], "./")
27 strctr = pdb.create_structure(0)
28 for i_chain in range(strctr.count_chains()) :
29     chain = strctr[i_chain]
30     for i_res in range(chain.count_residues()) :
31         if chain[i_res].residue_type().code3 == sys.argv[2] :
32             atoms = []
33             for at_name in sys.argv[3:] :
34                 try :
35                     at_name_fixed = at_name.replace("_", " ")
36                     atoms.append( chain[i_res].find_atom(at_name_fixed) )
37                     if not atoms[-1] :
38                         sys.stderr.write("Can't find atom "+at_name_fixed+" in "+sys.argv[2]+"
→residue\n")
39                 except :
40                     sys.stderr.write("Can't find atom "+at_name+" in "+sys.argv[2]+"
residue\n")
41                     s = SaturatedRing6Geometry(atoms[0],atoms[1],atoms[2],atoms[3],atoms[4],
→atoms[5])
42                     print(s.first_wing_angle(),s.second_wing_angle())
43
```



7.1.3 ex_* programs

These group contains unit test, i.e. programs that tests a single class of a function.

ex_BinaryTreeNode

Simple demo for BinaryTreeNode class

Keywords:

- *algorithms*
- *data structures*
- *graphs*

Categories:

- core/algorithms/trees/BinaryTreeNode

Output files:

- stdout.out

Program source:

```
1 #include <memory>
2 #include <iostream>
3
4 #include <core/algorithms/trees/TreeNode.hh>
5 #include <core/algorithms/trees/algorithms.hh>
6 #include <core/algorithms/trees/trees_io.hh>
7
8
9 /** @brief Simple demo for BinaryTreeNode class
10 *
11 * This program creates a small tree with 6 nodes and performs various operations on it
12 *
13 * CATEGORIES: core/algorithms/trees/BinaryTreeNode
14 * KEYWORDS: algorithms; data structures; graphs
15 * IMG: ex_BinaryTreeNode_1.png
16 * IMG_ALT: Example tree node
17 */
18 int main(const int argc, const char* argv[]) {
19
20     using namespace core::algorithms::trees;
21
22     typedef std::shared_ptr<BinaryTreeNode<char>> Node_SP; // --- Let's make the typename shorter
23     Node_SP p1(new BinaryTreeNode<char>(0, 'A'));
24     Node_SP p2(new BinaryTreeNode<char>(1, 'B'));
25     Node_SP p3(new BinaryTreeNode<char>(2, 'C'));
```

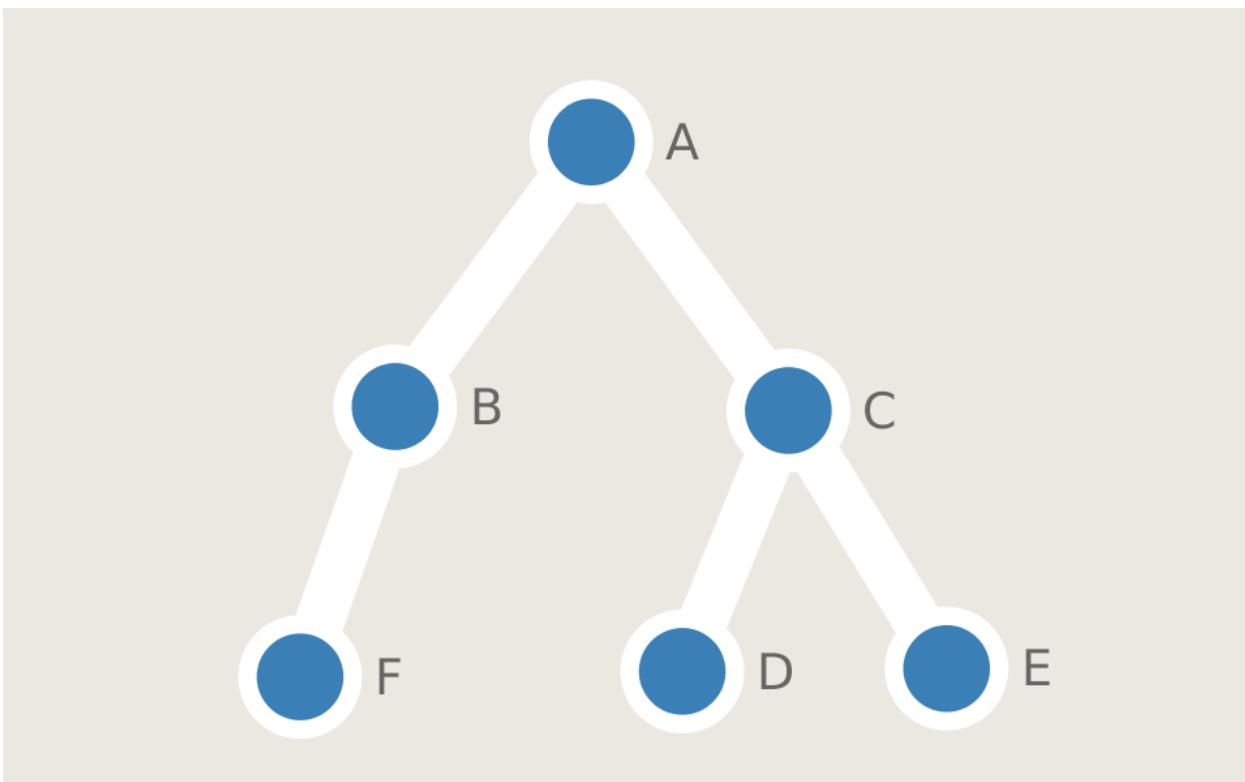
(continues on next page)

(continued from previous page)

```

26 Node_SP p4(new BinaryTreeNode<char>(3, 'D'));
27 Node_SP p5(new BinaryTreeNode<char>(4, 'E'));
28 Node_SP p6(new BinaryTreeNode<char>(5, 'F'));
29
30 p1->set_left_right(p2, p3);
31 p3->set_left_right(p4, p5);
32 p2->set_left(p6);
33 std::cout << "Size of the whole tree and its right branch: " << size(p1) << " "
34 << size(p3)
35 << " (should be 6 and 3)\n";
36
37 std::vector<char> elements;
38 collect_leaf_elements(p1, elements);
39 std::cout << "Leaf-only elements (E D F):\n";
40 for (std::vector<char>::const_iterator i = elements.begin(); i != elements.end(); i++)
41     std::cout << *i << ' ';
42 std::cout << "\n";
43
44 std::cout << "All elements stored on the tree (A C E D B F):\n";
45 elements.clear();
46 collect_elements(p1, elements);
47 for (std::vector<char>::const_iterator i = elements.begin(); i != elements.end(); i++)
48     std::cout << *i << ' ';
49 std::cout << "\n";
50
51 std::cout << "Leaf-only nodes (E D F):\n";
52 elements.clear();
53 std::vector<Node_SP> nodes;
54 collect_leaf_nodes(p1, nodes);
55 for (Node_SP & i : nodes)
56     std::cout << i->element << ' ';
57 std::cout << "\n";
58
59 std::cout << "The tree was:\n";
60 XMLFormatters<Node_SP> xml(std::cout);
61 write_tree(p1, xml.start, xml.leaf, xml.stop);
62
63 return 0;
}

```



ex_Molecule

Demonstrates how to create a Molecule object based on PdbAtom data type (as nodes of the graph)

Keywords:

- *molecule*

Categories:

- core::chemical::Molecule

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <memory>
3
4 #include <core/index.hh>
5 #include <core/algorithms/graph_algorithms.hh>
6 #include <core/chemical/Molecule.hh>
```

(continues on next page)

(continued from previous page)

```

7 #include <core/chemical/molecule_utils.hh>
8 #include <core/calc/structural/angles.hh>
9 #include <core/data/structural/PdbAtom.hh>
10 #include <core/chemical/PdbMolecule.hh>
11 #include <core/chemical/Bond.hh>
12
13 core::chemical::PdbMolecule_SP create_toluene_molecule() {
14
15     using namespace core::chemical;
16     using namespace core::data::structural;
17
18     // --- Define atoms that we use to build a molecule
19     PdbAtom_SP atoms[] = {std::make_shared<PdbAtom>(1, "C1 ", 0, 0, 0),
20                           std::make_shared<PdbAtom>(2, "C2 ", 1.24, 0.72, 0),
21                           std::make_shared<PdbAtom>(3, "C3 ", 1.24, 2.16, 0),
22                           std::make_shared<PdbAtom>(4, "C4 ", 0, 2.88, 0),
23                           std::make_shared<PdbAtom>(5, "C5 ", -1.24, 2.16, 0),
24                           std::make_shared<PdbAtom>(6, "C6 ", -1.24, 0.72, 0),
25                           std::make_shared<PdbAtom>(7, "C7 ", 0, -1.52, 0)};
26     PdbMolecule_SP toluene = std::make_shared<PdbMolecule>();
27
28     // --- Insert atoms into the molecule
29     for (PdbAtom_SP ai : atoms) toluene->add_atom(ai);
30     // --- Create bonds between them
31     toluene->bind_atoms(0, 1, BondType::AROMATIC);
32     toluene->bind_atoms(1, 2, BondType::AROMATIC);
33     toluene->bind_atoms(2, 3, BondType::AROMATIC);
34     toluene->bind_atoms(3, 4, BondType::AROMATIC);
35     toluene->bind_atoms(4, 5, BondType::AROMATIC);
36     toluene->bind_atoms(0, 5, BondType::AROMATIC);
37     toluene->bind_atoms(0, 6, BondType::SINGLE);
38
39     return toluene;
40 }
41
42 /** @brief Demonstrates how to create a Molecule object based on PdbAtom data type
43  * (as nodes of the graph)
44  */
45
46 * This demo is similar to ex_Molecule_vec3, the difference is that here PdbAtom
47 * instances are used as graph nodes
48 * rather than Vec3 instance. It creates a toluene molecule and detects planar and
49 * dihedral angles.
50
51 */
52 int main(const int argc, const char *argv[]) {
53
54     using namespace core::chemical;
55     using namespace core::data::structural;
56
57     PdbMolecule_SP molecule;
58     if (argc == 1)
59         molecule = create_toluene_molecule();
60     else {

```

(continues on next page)

(continued from previous page)

```

61 // --- Read structure that we use to build a molecule
62 core::data::io::Pdb reader(argv[1]); // file name (PDB format, may be gzip-ped)
63 core::data::structural::Structure_SP strctr = reader.create_structure(0);
64 // --- Create molecule object
65 molecule = structure_to_molecule(*strctr);
66 //molecule = create_molecule<Structure::atom_iterator>(strctr->first_atom(),_
67 //strctr->last_atom(), 0.1);
68 }

69 // --- Print some info about the molecule
70 std::cout << molecule->count_atoms() << " atoms, " << molecule->count_bonds() << "_"
71 //<< bonds\n";
72 for (auto atom_it=molecule->begin_atom();atom_it!=molecule->end_atom();++atom_it) {
73     PdbAtom_SP ai = *atom_it;
74     std::cout << "atom " << ai->id() << " bonded to " << molecule->count_bonds(ai) <<
75     //<< " atoms:";
76     for (auto n_it = molecule->begin_atom(ai); n_it != molecule->end_atom(ai); ++n_it)
77         std::cout << " " << (*n_it)->id();
78     std::cout << "\n";
79 }
80

81 // --- Find all planar angles in the molecule
82 std::vector<std::tuple<PdbAtom_SP, PdbAtom_SP, PdbAtom_SP>> planars;
83 find_planar_angles(*molecule, planars);
84 // --- Sort the angles just to make the output stable i.e. every time in the same_
85 //order so it can be used for benchmarking
86 std::sort(planars.begin(), planars.end(), ComparePlanarAngles());
87

88 std::cout << "Detected planar angles:\n"; // --- Evaluate and print all the planars
89 for (auto pi : planars) {
90     PdbAtom &a1 = *std::get<0>(pi);
91     PdbAtom &a2 = *std::get<1>(pi);
92     PdbAtom &a3 = *std::get<2>(pi);
93     std::cout << a1.id() << " -- " << a2.id() << " -- " << a3.id() << " " <<
94     core::calc::structural::evaluate_planar_angle(a1, a2, a3) * 180.0 / 3.14159 << "\n"
95     ;
96 }
97

98 // --- Find all torsion angles in the molecule
99 std::vector<std::tuple<PdbAtom_SP, PdbAtom_SP, PdbAtom_SP, PdbAtom_SP>> torsions;
100 find_torsion_angles(*molecule, torsions);
101 // --- Sort also dihedral angles
102 std::sort(torsions.begin(), torsions.end(), CompareDihedralAngles());
103 std::cout << "Detected dihedral angles:\n"; // --- Evaluate and print all the_
104 //planars
105 for (auto ti : torsions) {
106     PdbAtom &a1 = *std::get<0>(ti);
107     PdbAtom &a2 = *std::get<1>(ti);
108     PdbAtom &a3 = *std::get<2>(ti);
109     PdbAtom &a4 = *std::get<3>(ti);
110     std::cout << a1.id() << " -- " << a2.id() << " -- " << a3.id() << " -- " << a4.
111     id() << " " <<
112     core::calc::structural::evaluate_dihedral_angle(a1, a2, a3, a4) * 180.0 / 3.14159
113     << "\n";
114 }
115

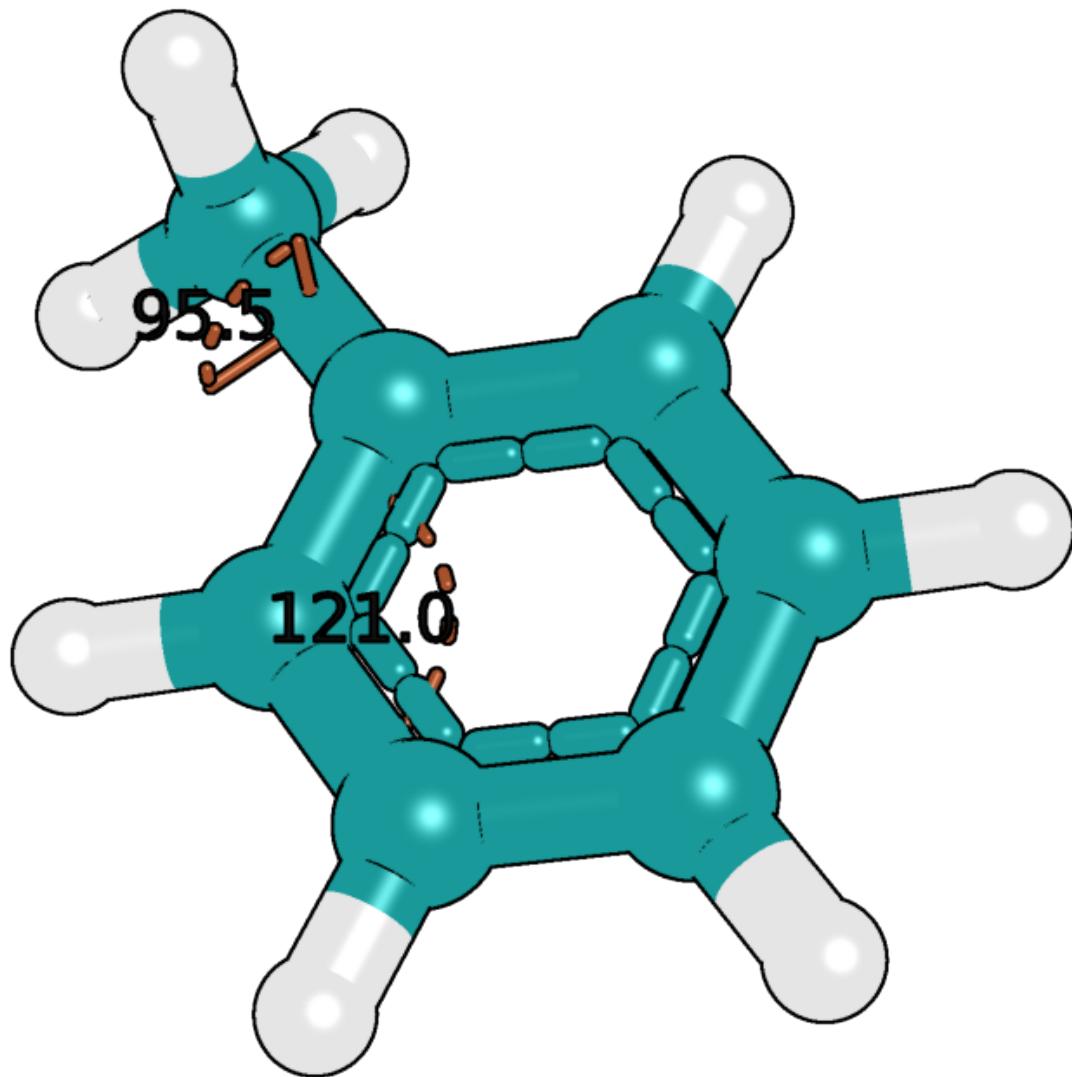
116 // --- Here we find the benzene ring in the molecule - a cycle in a graph

```

(continues on next page)

(continued from previous page)

```
110     std::vector<std::vector<core::index4>> cycles =
111         core::algorithms::find_cycles<PdbMolecule, PdbAtom_SP, std::shared_ptr<BondType>_>>(*molecule);
112
113     std::cout << "Atoms in a cycle:";
114     for (core::index4 i:cycles[0]) std::cout << " " << molecule->get_atom(i)->atom_
115     >name();
116     std::cout << "\n";
117 }
```



ex_Molecule_Vec3

Unit test which shows how to create a Molecule object based on Vec3 data type (Vec3 objects are nodes of the graph).

USAGE:

```
./ex_Molecule_Vec3
```

Keywords:

- *molecule*

Categories:

- core::chemical::Molecule

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <memory>
3
4 #include <core/chemical/Molecule.hh>
5 #include <core/chemical/molecule_utils.hh>
6 #include <core/calc/structural/angles.hh>
7 #include <core/data/basic/Vec3.hh>
8 #include <utils/options/OptionParser.hh>
9 #include <utils/exit.hh>
10
11 std::string program_info = R"(

12 Unit test which shows how to create a Molecule object based on Vec3 data type
13 (Vec3 objects are nodes of the graph).

14
15 USAGE:
16     ./ex_Molecule_Vec3
17
18 )";

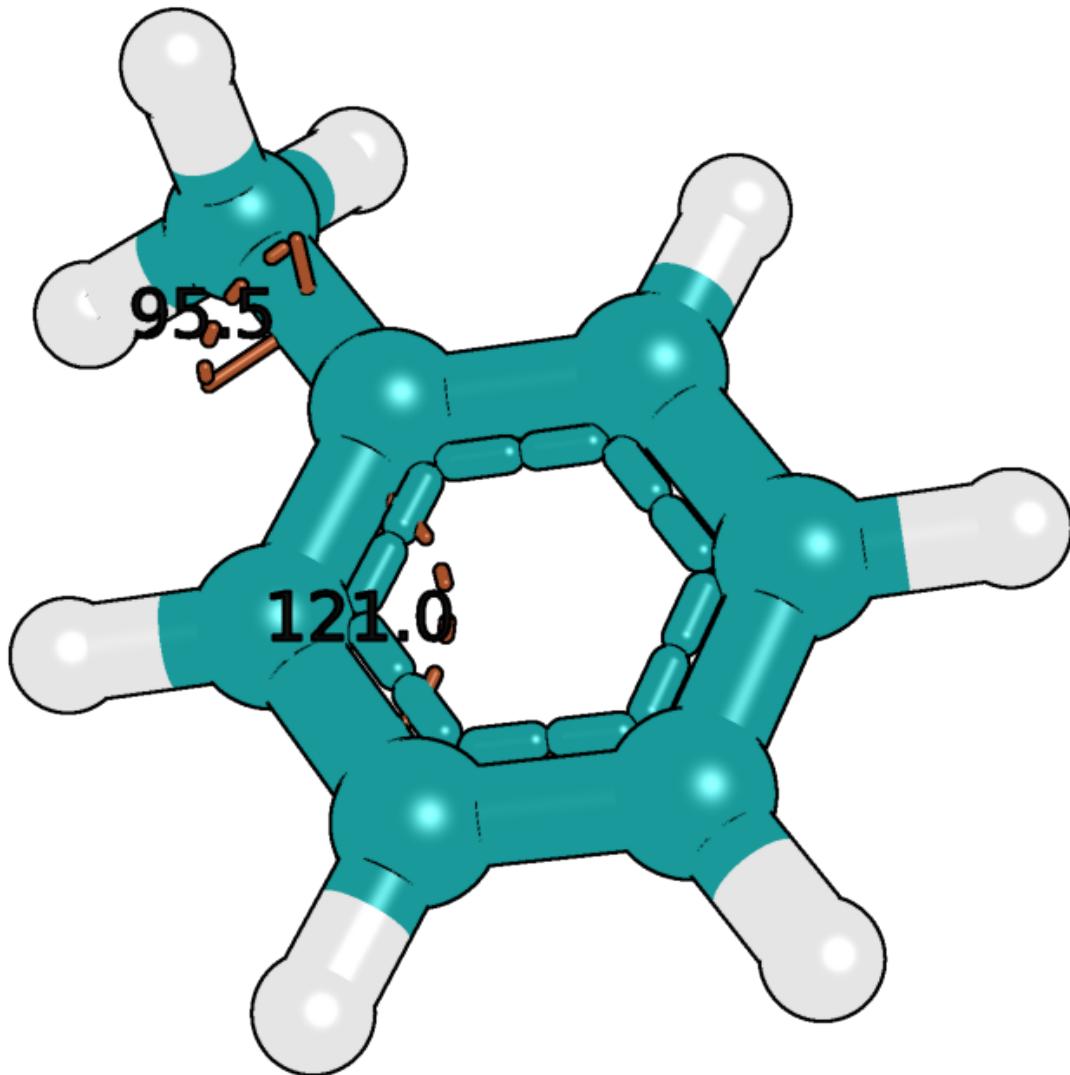
19 /**
20 * @brief Demonstrates how to create a Molecule object based on Vec3 data type (Vec3_
21 *       ↪are nodes of the graph)
22 *
23 * This demo is similar to ex_Molecule, the difference is that here Vec3 instances_
24 *       ↪are used as graph nodes
25 * rather than PdbAtom instance. It creates a toluene molecule and detects planar_
26 *       ↪angles.
27 *
28 * CATEGORIES: core::chemical::Molecule
29 * KEYWORDS: molecule
30 * IMG: Toluen_dihedral_flat_angle.png
31 * IMG_ALT: Planar angles in a toluen molecule
32 */
```

(continues on next page)

(continued from previous page)

```

33 if ((argc > 1) && utils::options::call_for_help(argv[1]))
34     utils::exit_OK_with_message(program_info);
35
36 using namespace core::chemical;
37 using namespace core::data::structural;
38
39 Molecule<Vec3> toluene;
40 Vec3 atoms[] = {Vec3(0, 0, 0),
41                 Vec3(1.24, 0.72, 0),
42                 Vec3(1.24, 2.16, 0),
43                 Vec3(0, 2.88, 0),
44                 Vec3(-1.24, 2.16, 0),
45                 Vec3(-1.24, 0.72, 0),
46                 Vec3(0, -1.52, 0)};
47
48 // --- Mark atom numbers to check if the molecule is correct
49 for (core::index2 i = 0; i < 7; ++i) atoms[i].register_ = i;
50
51 // --- Insert atoms into the molecule
52 for (Vec3 & ai : atoms) toluene.add_atom(ai);
53 // --- Create bonds between them
54 toluene.bind_atoms(0, 1, BondType::AROMATIC);
55 toluene.bind_atoms(1, 2, BondType::AROMATIC);
56 toluene.bind_atoms(2, 3, BondType::AROMATIC);
57 toluene.bind_atoms(3, 4, BondType::AROMATIC);
58 toluene.bind_atoms(4, 5, BondType::AROMATIC);
59 toluene.bind_atoms(0, 5, BondType::AROMATIC);
60 toluene.bind_atoms(0, 6, BondType::SINGLE);
61
62 std::cout << "Connectivity (bonds):\n";
63 for(auto atom_it=toluene.cbegin_atom();atom_it!=toluene.cend_atom();++atom_it) {
64     std::cout << (*atom_it).register_ << " : ";
65     for(auto atom_it2=toluene.cbegin_atom(*atom_it);atom_it2!=toluene.cend_atom(*atom_
66     ↪it);++atom_it2)
67         std::cout << " " << (*atom_it2).register_;
68     std::cout << "\n";
69 }
70
71 // --- Find all planar angles in the molecule
72 std::vector<std::tuple<Vec3, Vec3, Vec3>> planars;
73 find_planar_angles(toluene, planars);
74
75 std::vector<double> planar_values;
76 std::cout << "Detected planar angles:\n"; // --- Evaluate and print all the planars
77 for (auto pi : planars) {
78     Vec3 &a1 = std::get<0>(pi);
79     Vec3 &a2 = std::get<1>(pi);
80     Vec3 &a3 = std::get<2>(pi);
81     planar_values.push_back(core::calc::structural::evaluate_planar_angle(a1, a2, a3) ↪
82     ↪* 180.0 / 3.14159);
83 }
84
85 // --- Sort the values before printing them to make the output stable
86 std::sort(planar_values.begin(),planar_values.end());
87 for (double value:planar_values) std::cout << value << "\n";
88 }
```



ex_NcbiSimilarityMatrixFactory

Test for loading substitution matrices available in BioShell. The program reads a substitution matrix from a given file (NCBI file format) and prints it back on the screen. The program can either load the input matrix from Biohell database (data alignments directory) or from a file specified by a user. One can manually install custom matrices just by copying them to data alignments/

USAGE:

```
./ex_NcbiSimilarityMatrixFactory subst-matrix-name
```

EXAMPLES:

```
./ex_NcbiSimilarityMatrixFactory BLOSUM45
./ex_NcbiSimilarityMatrixFactory ./BLOSUM45.txt
```

Keywords:

- *sequence alignment*
- substitution matrix

Categories:

- core::alignment::scoring::NcbiSimilarityMatrixFactory

Output files:

- stdout.out

Program source:

```

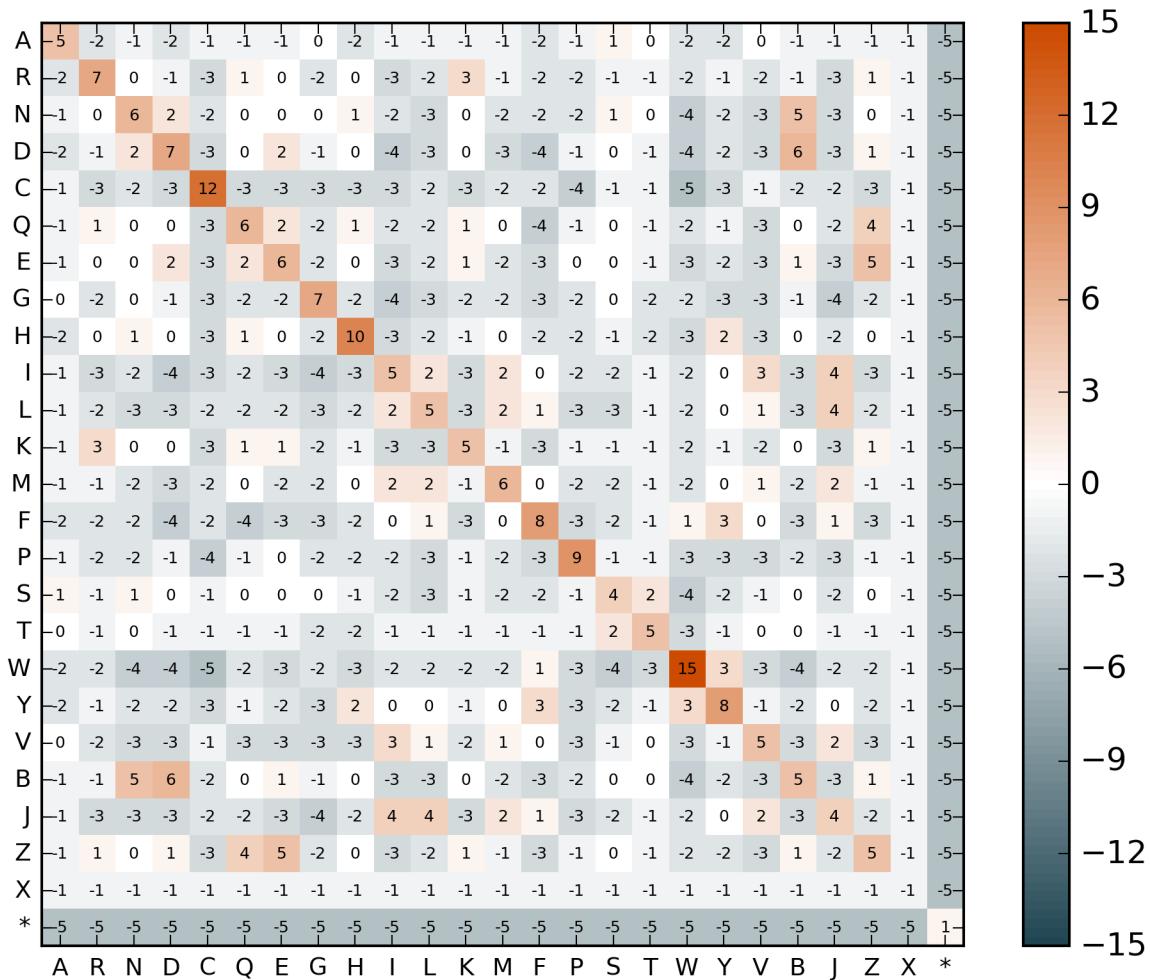
1 #include <iostream>
2
3 #include <core/alignment/scoring/SimilarityMatrix.hh>
4 #include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Test for loading substitution matrices available in BioShell.
10
11 The program reads a substitution matrix from a given file (NCBI file format) and prints
12 it back on the screen. The program can either load the input matrix from Biohell_
13 ↪database
14 (data/alignments directory) or from a file specified by a user.
15 One can manually install custom matrices just by copying them to data/alignments/
16
17 USAGE:
18 ../ex_NcbiSimilarityMatrixFactory subst-matrix-name
19
20 EXAMPLES:
21 ../ex_NcbiSimilarityMatrixFactory BLOSUM45
22 ../ex_NcbiSimilarityMatrixFactory ./BLOSUM45.txt
23 )";
24
25
26 /** @brief Test for loading substitution matrices available in BioShell
27 *
28 * CATEGORIES: core::alignment::scoring::NcbiSimilarityMatrixFactory
29 * KEYWORDS: sequence alignment; substitution matrix
30 * IMG: heatmap_1.png
31 * IMG_ALT: BLOSUM62 matrix plotted
32 */

```

(continues on next page)

(continued from previous page)

```
33 int main(const int argc, const char *argv[]) {
34
35     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
36     ↪missing program parameter
37
38     using namespace core::alignment::scoring;
39
40     NcbiSimilarityMatrixFactory sim_factory = NcbiSimilarityMatrixFactory::get();
41
42     if (argc == 1) {
43         std::vector<std::string> names;
44         sim_factory.get().matrix_names(names);
45         std::cout << "\nMatrices defined in BioShell:\n";
46         for (const auto &n : names)
47             std::cout << "\t" << n;
48         std::cout << "\n";
49     } else {
50         NcbiSimilarityMatrix_SP m = sim_factory.load_matrix(argv[1]);
51         std::stringstream out;
52         out << "\n";
53         m->print("%4d", 4, out);
54         std::cout << out.str() << "\n";
55     }
56 }
```



ex_REMC_Ising

The program runs a Replica Exchange Monte Carlo simulation of a simple 2D Ising system (a spin glass). The simulation performs N_INNER x N_OUTER MC cycles and then a replica exchange is attempted.

USAGE:

```
ex_REMC_Ising [system_size inner_cycles outer_cycles n_exchanges]
```

EXAMPLE:

```
ex_REMC_Ising 32 50 100 100
```

Keywords:

- REMC
- Ising2D
- *observer*
- *simulation*

Categories:

- simulations/sampling/ReplicaExchangeMC; simulations/systems/ising/Ising2D

Output files:

- movers-2.000.dat
- movers-3.000.dat
- movers-1.000.dat
- replica_flow.dat
- energy-2.250.dat
- energy-2.000.dat
- energy-1.500.dat
- movers-2.250.dat
- energy-2.500.dat
- movers-7.500.dat
- movers-4.000.dat
- energy-3.000.dat
- energy-100.000.dat
- movers-100.000.dat
- energy-1.000.dat
- movers-5.000.dat
- movers-1.500.dat
- energy-5.000.dat
- energy-7.500.dat
- energy-1.750.dat
- movers-2.500.dat
- movers-1.750.dat
- energy-4.000.dat

Program source:

```
1 #include <iostream>
2
3 #include <simulations/evaluators/CallEvaluator.hh>
4 #include <simulations/forcefields/CalculateEnergyBase.hh>
5
6 #include <simulations/movers/ising/SingleFlip2D.hh>
7 #include <simulations/movers/ising/WolffMove2D.hh>
8
9 #include <simulations/observers/ObserveEvaluators.hh>
```

(continues on next page)

(continued from previous page)

```

10 #include <simulations/observers/ObserveMoversAcceptance.hh>
11 #include <simulations/observers/TriggerEveryN.hh>
12 #include <simulations/observers/ObserveReplicaFlow.hh>
13
14 #include <simulations/sampling/IsothermalMC.hh>
15 #include <simulations/sampling/ReplicaExchangeMC.hh>
16 #include <simulations/systems/isising/Ising2D.hh>
17
18 using namespace core::data::basic;
19
20 utils::Logger logs("ex_REMC_Ising");
21
22 std::string program_info = R"(
23
24 The program runs a Replica Exchange Monte Carlo simulation of a simple 2D Ising_
25 system (a spin glass).
26
27 The simulation performs N_INNER x N_OUTER MC cycles and then a replica exchange is_
28 attempted.
29
30 USAGE:
31     ex_REMC_Ising [system_size inner_cycles outer_cycles n_exchanges]
32
33 EXAMPLE:
34     ex_REMC_Ising 32 50 100 100
35 )";
36
37 /**
38 * @brief The program runs a Replica Exchange Monte Carlo simulation of a simple 2D_
39 * Ising system (spin glass).
40 */
41
42 /**
43 * This example shows how to set up a REMC simulation
44 *
45 * CATEGORIES: simulations/sampling/ReplicaExchangeMC; simulations/systems/isising/
46 * Ising2D
47 * KEYWORDS: REMC; Ising2D; observer; simulation
48 * IMG: Energy_plot.png
49 * IMG_ALT: Energy over time in Ising model
50 */
51
52 int main(const int argc,const char* argv[]) {
53
54     using namespace simulations::systems::isising;
55     using namespace simulations::movers::isising;
56
57     core::index4 n_outer_cycles = 10;
58     core::index4 n_inner_cycles = 10;
59     core::index4 n_exchanges = 10;
60     std::vector<double> temperatures = {100.0, 7.5, 5, 4, 3, 2.5, 2.25, 2, 1.75, 1.5, 1}
61     ;
62
63     core::index2 system_size = 32;
64     if (argc < 2) std::cerr << program_info;
65     else {
66         system_size = atoi(argv[1]);
67         n_inner_cycles = atoi(argv[2]);
68         n_outer_cycles = atoi(argv[3]);
69     }
70 }
```

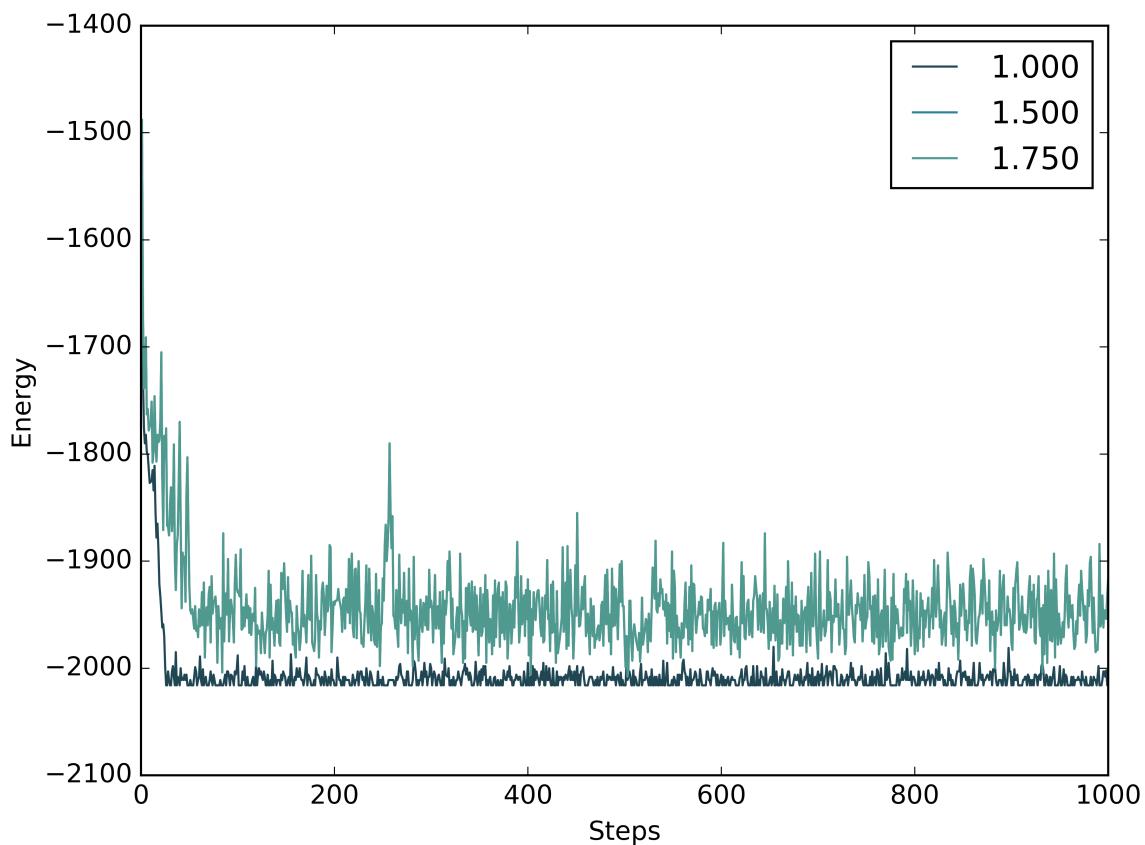
(continues on next page)

(continued from previous page)

```

62     n_exchanges = atoi(argv[4]);
63 }
64
65 core::calc::statistics::Random::get().seed(12345); // --- seed the generator for
66 // repeatable results
67
68 std::vector<std::shared_ptr<Ising2D<core::index1,core::index2>>> systems;
69 std::vector<simulations::sampling::IsothermalMC_SP> replica_samplers;
70 std::vector<simulations::forcefields::TotalEnergy_SP> energies;
71
72 for (core::index2 irepl = 0; irepl < temperatures.size(); ++irepl) {
73
74     // ----- Create the systems to be sampled -----
75     std::shared_ptr<Ising2D<core::index1,core::index2>> system
76         = std::make_shared<Ising2D<core::index1,core::index2>>(system_size, system_
77 //size);
78     system->initialize(); // Populate system with random spins
79     systems.push_back(system);
80     energies.push_back(system);
81
82     // ----- Movers definition -----
83     simulations::movers::MoversSet_SP movers = std::make_shared
84 //<simulations::movers::MoversSetSweep>();
85     movers->add_mover( std::make_shared<SingleFlip2D<core::index1,core::index2>>
86 //(*system),system->count_spins());
87     movers->add_mover( std::make_shared<WolffMove2D<core::index1,core::index2>>
88 //(*system),system->count_spins()*0.2);
89
90     // ----- Create the sampler -----
91     auto sampler = std::make_shared<simulations::sampling::IsothermalMC>(movers,
92 //temperatures[irepl]);
93     replica_samplers.push_back(sampler);
94     sampler->cycles(n_inner_cycles,n_outer_cycles);
95
96     simulations::observers::ObserveEvaluators_SP observations
97         = std::make_shared<simulations::observers::ObserveEvaluators>(utils::string_
98 //format("energy-%.3f.dat",temperatures[irepl]));
99     observations->add_evaluator(system);
100    sampler->outer_cycle_observer(observations);
101    simulations::observers::ObserveMoversAcceptance_SP obs_ms
102        = std::make_shared<simulations::observers::ObserveMoversAcceptance>(*movers,
103 //utils::string_format("movers-%.3f.dat",temperatures[irepl]));
104    obs_ms->observe_header();
105    sampler->outer_cycle_observer(obs_ms);
106 }

107 bool replica_isothermal_observation_mode = true;
108 auto remc = std::make_shared<simulations::sampling::ReplicaExchangeMC>(replica_
109 //samplers, energies, replica_isothermal_observation_mode);
110 auto remc_flow = std::make_shared<simulations::observers::ObserveReplicaFlow>(*remc,
111 //"replica_flow.dat");
112 remc->exchange_observer(remc_flow);
113 remc->replica_exchanges(n_exchanges);
114 remc->run();
115 }
```



ex_SelectChainResidueAtom

Extracts a fragment of a PDB file by applying a SelectChainResidueAtom selector. The selection string consists of chain code and residue range, separated by a colon, e.g.: - A:-1-10 - AB:

USAGE:

```
ex_SelectChainResidueAtom input.pdb selector-string
```

EXAMPLES:

```
ex_SelectChainResidueAtom 2gb1.pdb A:23-32
ex_SelectChainResidueAtom 1ofz.pdb A:aa
ex_SelectChainResidueAtom 2gb1.pdb A:1-20:_CA+_N+_O+_C_
ex_SelectChainResidueAtom 1ofz.pdb *:*:_CA_
```

Keywords:

- *structure selectors*
- *PDB input*
- *PDB output*

Categories:

- core::data::structural::StructureSelector

Input files:

- 1ofz.pdb
- 2gb1.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <core/data/io/Pdb.hh>
3 #include <core/data/structural/selectors/structure_selectors.hh>
4 #include <utils/exit.hh>
5
6 std::string program_info = R"(
7
8 Extracts a fragment of a PDB file by applying a SelectChainResidueAtom selector.
9
10 The selection string constists of chain code and residue range, separated by a colon, ↵
11 ↵e.g.:
12   - A:-1-10
13   - AB:
14
15 USAGE:
16   ex_SelectChainResidueAtom input.pdb selector-string
17 EXAMPLES:
18   ex_SelectChainResidueAtom 2gb1.pdb A:23-32
19   ex_SelectChainResidueAtom 1ofz.pdb A:aa
20   ex_SelectChainResidueAtom 2gb1.pdb A:1-20:_CA+_N__+_O__+_C__
21   ex_SelectChainResidueAtom 1ofz.pdb *:*:_CA_
22 )
23
24 /**
25  * @brief Extracts a fragment of a PDB file.
26  *
27  * CATEGORIES: core::data::structural::StructureSelector
28  * KEYWORDS: structure selectors; PDB input; PDB output
29  * IMG: ex_SelectChainResidueAtoms_1.png
30  * IMG_ALT: Proline residue selected from 1OFZ deposit
31  */
32 int main(const int argc, const char* argv[]) {
33
34     using namespace core::data::structural;
35
36     if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about ↵
37     ↵missing program parameter
```

(continues on next page)

(continued from previous page)

```

37 // ----- Read a PDB file and create a Structure object
38 core::data::io::Pdb reader(argv[1],      // --- data file
39     core::data::io::keep_all);          // --- a predicate to read ALL the ATOM lines
40 // (by default hydrogens are excluded)
41 Structure_SP strctr = reader.create_structure(0);
42
43 // --- Create a selector object from a selector string
44 selectors::SelectChainResidueAtom sel(argv[2]);
45 Structure_SP full_copy = strctr->clone(sel); // --- cloning with this selector
46 //makes a deep copy of everything
47 for(auto atom_it=full_copy->first_atom();atom_it!=full_copy->last_atom();++atom_it)
48     std::cout << (*atom_it)->to_pdb_line() << "\n";
}

```



ex_SelectPlanarCAGeometry

Reads a PDB file and tests whether geometry at CA atom is tetrahedral or not. The program also prints the actual values of the N-CA-C-CB dihedral angle.

USAGE:

```
./ex_SelectPlanarCAGeometry input.pdb
```

EXAMPLE:

```
./ex_SelectPlanarCAGeometry 5edw.pdb
```

OUTPUT (fragment): 112 CYS D OK -2.22 3dcg 140 ASN E WRONG -2.42 3dcg 141 LYS E OK -2.23 3dcg 142 VAL E OK -2.17 3dcg 144 SER E OK -2.16 3dcg 145 LEU E OK -2.19 3dcg

Keywords:

- residue geometry
- *structure selectors*
- *PDB input*
- *structure validation*

Categories:

- core::data::structural::ResidueHasBBCB;
core::data::structural::SelectPlanarCAGeometry
- core::data::structural::SelectResidueByName;

Input files:

- 2gb1.pdb
- 5edw.pdb
- 3dcg.pdb

Output files:

- stdout.out

Program source:

```
1 #include <core/data/io/Pdb.hh>
2 #include <core/calc/structural/angles.hh>
3 #include <core/data/structural/selectors/structure_selectors.hh>
4 #include <core/data/structural/selectors>SelectPlanarCAGeometry.hh>
5
6 #include <utils/exit.hh>
7 #include <utils/io_utils.hh>
8
9 std::string program_info = R" (
```

(continues on next page)

(continued from previous page)

```

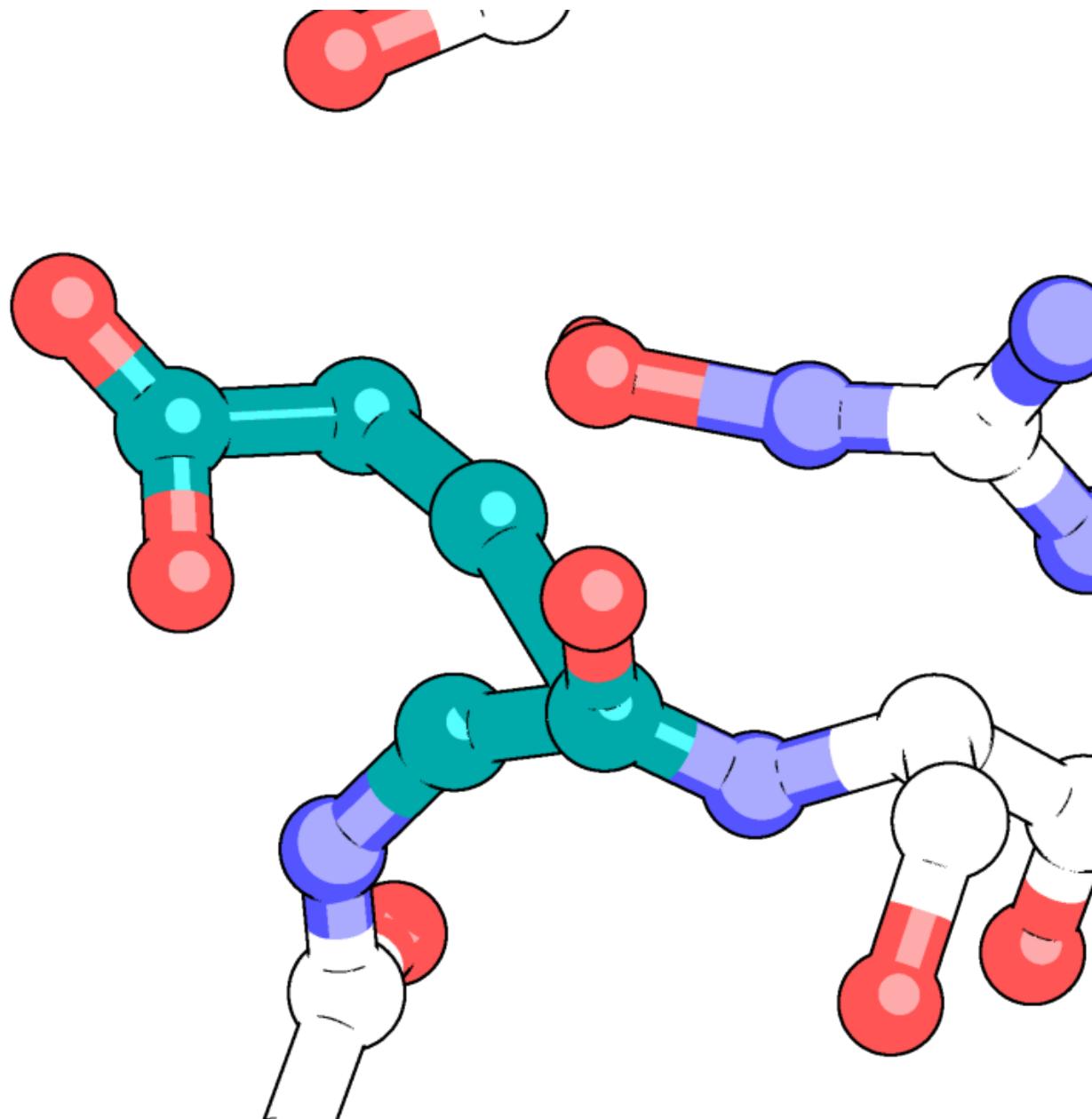
10
11 Reads a PDB file and tests whether geometry at CA atom is tetrahedral or not.
12 The program also prints the actual values of the N-CA-C-CB dihedral angle.
13
14 USAGE:
15   ./ex_SelectPlanarCAGeometry input.pdb
16
17 EXAMPLE:
18   ./ex_SelectPlanarCAGeometry 5edw.pdb
19
20 OUTPUT (fragment):
21   112 CYS D  OK      -2.22 3dcg
22   140 ASN E  WRONG   -2.42 3dcg
23   141 LYS E  OK      -2.23 3dcg
24   142 VAL E  OK      -2.17 3dcg
25   144 SER E  OK      -2.16 3dcg
26   145 LEU E  OK      -2.19 3dcg
27
28 ) ";
29
30 /** @brief Tests whether alpha-carbons actually have tetrahedral geometry as they_
31  ↪should.
32 *
33 * CATEGORIES: core:::data::structural::ResidueHasBBCB;_
34  ↪core:::data::structural::SelectResidueByName;_
35  ↪core:::data::structural::SelectPlanarCAGeometry
36 * KEYWORDS: residue geometry; structure selectors; PDB input; structure validation
37 * IMG: 1NXB_A_56_Glu_pymol_5.png
38 * IMG_ALT: GLU56 of 1NXB deposit has planar geometry of alpha-carbon (witch_
39  ↪contradicts basic chemical knowledge)
40 */
41 int main(const int argc, const char *argv[]) {
42
43   if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
44  ↪missing program parameter
45   using namespace core:::data::io;
46   using core:::calc::structural::to_degrees;
47
48   core:::data::io::Pdb reader(argv[1], is_not_alternative); // file name (PDB format,_
49  ↪may be gzip-ped)
50   core:::data::structural::Structure_SP strctr = reader.create_structure(0);
51
52   // --- Selector that returns true if a residue has beta-carbon
53   core:::data::structural::selectors::ResidueHasBBCB has_bb_cb;
54   // --- Selector that test the geometry on alpha carbon
55   core:::data::structural::selectors::SelectPlanarCAGeometry tester;
56   // --- Selector that selects GLY residues
57   core:::data::structural::selectors::SelectResidueByName is_gly("GLY");
58   for (auto res_it = strctr->first_residue(); res_it != strctr->last_residue(); ++res_
59  ↪it) {
60     // --- If a residue is not GLY and has C-beta ...
61     if ((has_bb_cb(**res_it)) && (!is_gly(**res_it)))
62       std::cout << utils::string_format("%4d %3s %4s %s %7.2f %s\n", (*res_it)->id(),
63                                         (*res_it)->residue_type().code3.c_str(), (*res_it)->owner()->id().c_str(),
64                                         (tester(**res_it)) ? "WRONG" : "OK ",
65                                         to_degrees(tester.evaluate_angle(**res_it)), utils::basename(strctr->code()) .
66                                         c_str());

```

(continues on next page)

(continued from previous page)

```
58 }  
59 }
```



ex_VonMisesDistribution

`ex_VonMisesDistribution` withdraws N random values (by default N = 1000) from a Normal distribution and fits Von Mises distribution to the data. If exactly two arguments are provided (mu and kappa, respectively) the program tabulates Von Mises distribution for that parameters.

USAGE:

```
1 ex_VonMisesDistribution N
2 ex_VonMisesDistribution mu kappa
```

EXAMPLES:

```
1 ex_VonMisesDistribution 10000
2 ex_VonMisesDistribution 1.5708 100.0
```

Keywords:

- *statistics*

Categories:

- core/calc/statistics/VonMisesDistribution

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <random>
3
4 #include <core/calc/statistics/VonMisesDistribution.hh>
5 #include <core/calc/statistics/Random.hh>
6
7 std::string program_info = R"(

8
9 ex_VonMisesDistribution withdraws N random values (by default N = 1000) from a Normal_
10 ↴distribution
11 and fits Von Mises distribution to the data.

12 If exactly two arguments are provided (mu and kappa, respectively) the program_
13 ↴tabulates Von Mises distribution
14 for that parameters.

15 USAGE:
16     ex_VonMisesDistribution N
17     ex_VonMisesDistribution mu kappa

18 EXAMPLES:
19     ex_VonMisesDistribution 10000
20     ex_VonMisesDistribution 1.5708 100.0

21 )
22

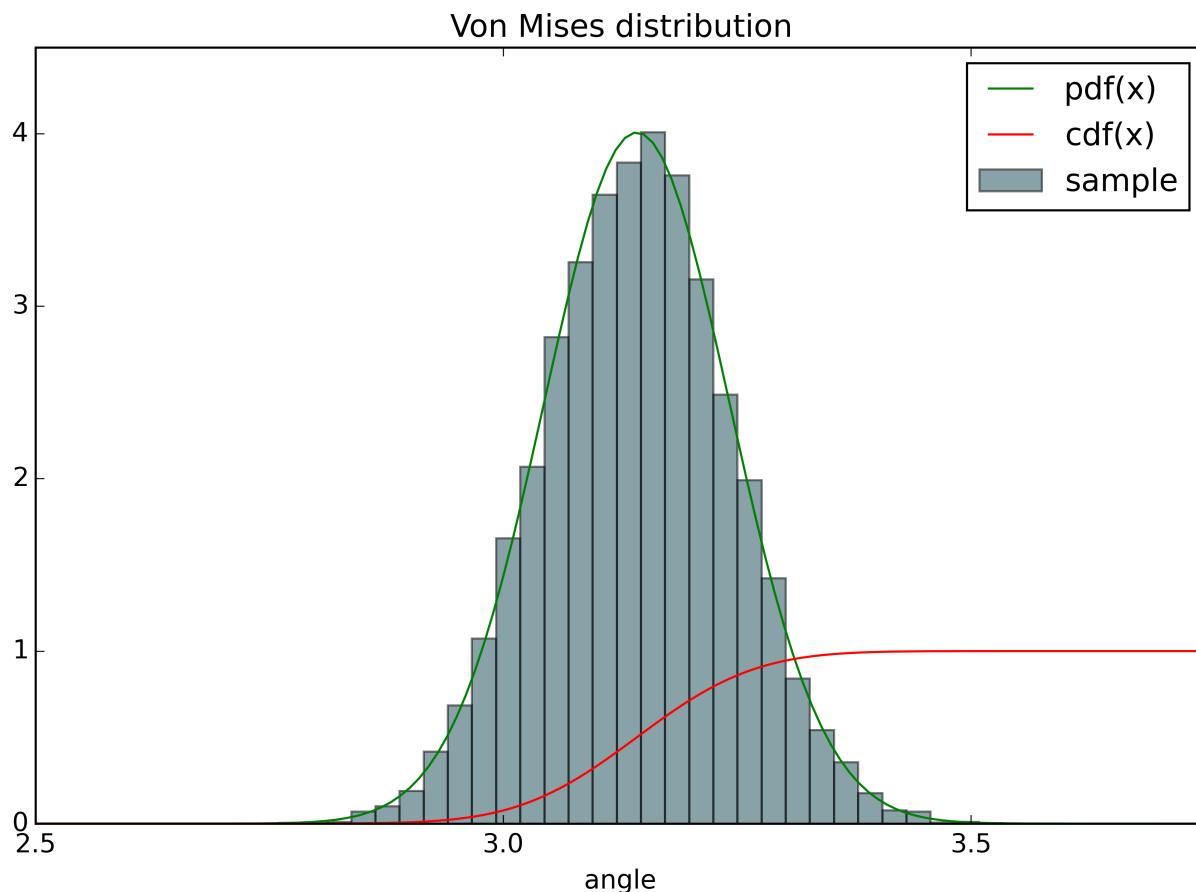
23 /**
24  * @brief Example which estimates parameters of von Mises distribution and tabulates_
25  * its values
26  * @CATEGORIES: core/calc/statistics/VonMisesDistribution
27  * @KEYWORDS: statistics
28  * @IMG: von_mises.png
```

(continues on next page)

(continued from previous page)

```

28 * IMG_ALT: Example von Mises distribution: histogram of a sample, pdf(x) and cdf(x)
29 */
30 int main(const int argc, const char *argv[]) {
31
32     using namespace core::calc::statistics;
33
34     core::calc::statistics::VonMisesDistribution f(std::vector<double>{0.0, 1.0}); // --
35     // initial distribution
36     if (argc == 3) {
37         double mu = atof(argv[1]);
38         double kappa = atof(argv[2]);
39         f.copy_parameters_from(std::vector<double>{mu, kappa});
40         std::cout << "# tabulating VonMisesDistribution: " << f << "\n";
41         for (double x = -M_PI; x <= M_PI; x += M_PI / 25.0)
42             std::cout
43                 << utils::string_format("%6.3f %9f %5.3f\n", x, f.evaluate(x),
44                                         VonMisesDistribution::cdf(x, f.mu(), f.kappa()));
45     }
46
47     core::index4 N = 1000;
48     if (argc < 2) std::cerr << program_info;
49     else N = atoi(argv[1]);
50     std::vector<std::vector<double>> input_data;
51     Random r = Random::get();
52     r.seed(9876543);
53     std::normal_distribution<double> dist(M_PI / 2.0, 0.1);
54
55     // ----- prepare data that will be use to estimate the distribution
56     for (core::index4 i = 0; i < N; ++i) {
57         std::vector<double> v({dist(r)});
58         input_data.push_back(v);
59     }
60     f.estimate(input_data); // --- run the estimation
61     std::cout << f << "\n";
62     // ----- now prepare weighted data: just copy some points ten times and insert
63     // them with weight 0.1
64     input_data.clear();
65     std::vector<double> weights;
66     for (core::index4 i = 0; i < N; ++i) {
67         double x = dist(r);
68         std::vector<double> v({x});
69         if (x < 2) {
70             input_data.push_back(v);
71             weights.push_back(1.0);
72         } else {
73             for (int j = 0; j < 10; ++j) {
74                 input_data.push_back(v);
75                 weights.push_back(0.1);
76             }
77         }
78     }
79     f.estimate(input_data, weights); // --- run the estimation based on weighted
80     // observations
81     std::cout << f << "\n";
82 }
```



ex_bf_by_residue

`ex_bf_by_residue` reads a PDB file and prints per-residue statistics of B-factors. The output provides: amino acid type (1-letter code), residue ID, and minimum, average and maximum b-factors for that residue

USAGE:

```
ex_bf_by_residue input.pdb
```

EXAMPLE:

```
ex_bf_by_residue 2gb1.pdb
```

Keywords:

- *PDB input*
- B-factors
- *structure selectors*

Categories:

- core::data::io::Pdb

Input files:

- 2gb1.pdb

Output files:

- stdout.out

Program source:

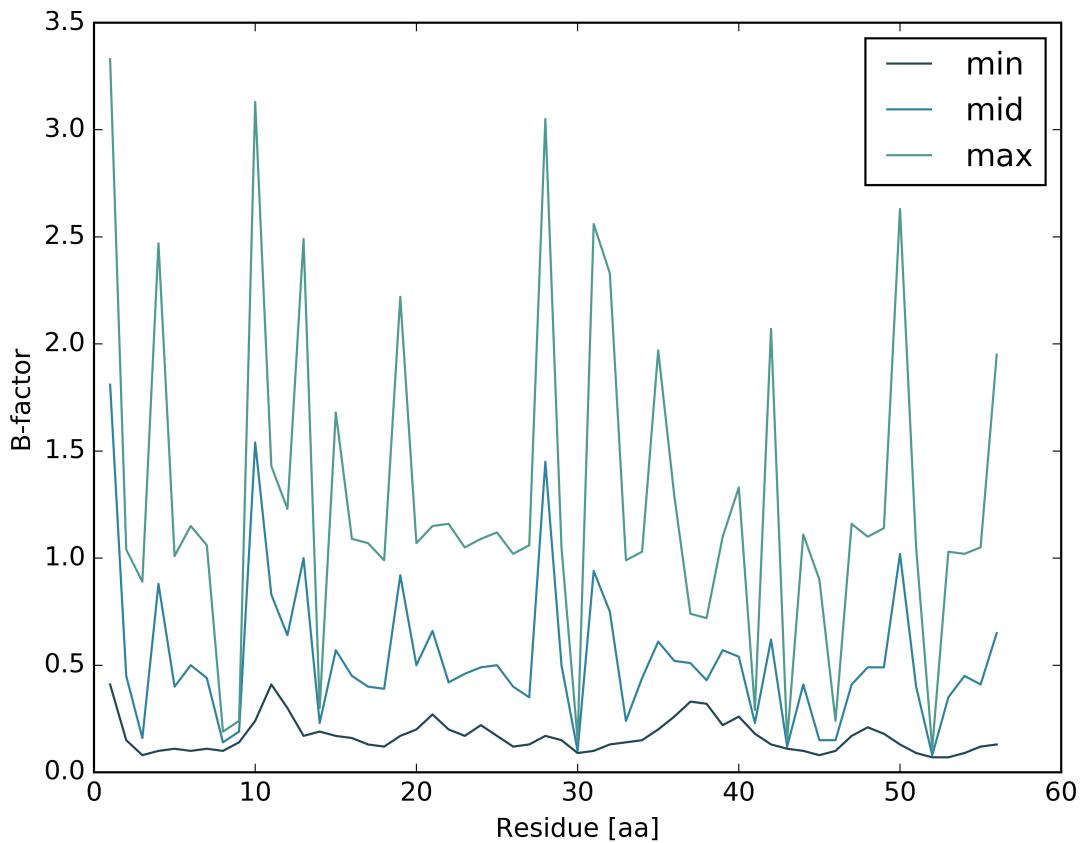
```
1 #include <core/data/io/Pdb.hh>
2 #include <core/data/structural/selectors/structure_selectors.hh>
3 #include <utils/string_utils.hh>
4 #include <utils/exit.hh>
5
6 std::string program_info = R"(
7
8 ex_bf_by_residue reads a PDB file and prints per-residue statistics of B-factors. The
9 ↪output provides:
10 amino acid type (1-letter code), residue ID, and minimum, average and maximum b-
11 ↪factors for that residue
12
13 USAGE:
14   ex_bf_by_residue input.pdb
15 EXAMPLE:
16   ex_bf_by_residue 2gb1.pdb
17
18 )";
19
20 /**
21 * CATEGORIES: core::data::io::Pdb;
22 * KEYWORDS: PDB input; B-factors; structure selectors
23 * IMG: Bfactor_plot.png
24 * IMG_ALT: B-factors of 2GB1 PDB deposit
25 */
26 int main(const int argc, const char *argv[]) {
27
28   if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
29   ↪missing program parameter
30
31   core::data::io::Pdb reader(argv[1]); // file name (PDB format, may be gzip-ped)
32   core::data::structural::Structure_SP strctr = reader.create_structure(0);
33   core::data::structural::selectors::IsBB is_bb;
34   core::data::structural::selectors::IsAA is_aa;
35   for (auto res_it = strctr->first_residue(); res_it != strctr->last_residue(); ++res_
36   ↪it) {
37     if (!is_aa(**res_it)) continue;
38     double min = 9999.0, max = -999.0, avg = 0.0, n = 0.0;
```

(continues on next page)

(continued from previous page)

```

37     for (auto atom : **res_it) {
38 //      if (is_bb(*atom)) continue; // --- uncomment that line to compute statistics
39 //      for side chain only
40         double bf = atom->b_factor();
41         if (min > bf) min = bf;
42         if (max < bf) max = bf;
43         avg += bf;
44         n += 1.0;
45     }
46     std::cout << (*res_it)->residue_type().code1 << " " <<
47     utils::string_format("%4d %5.2f %5.2f %5.2f\n", (*res_it)->id(), min, avg / n,
48 //      max);
49   }
50 }
```



ex_chi_correlation

Unit test which calculates Chi dihedral angles for every pair of amino acid side chains measured in two different homologous protein structures which are assumed to be aligned.

USAGE:

```
ex_chi_correlation file-1.pdb file-2.pdb
```

EXAMPLE:

```
ex_chi_correlation 1bgx_aligned.pdb 1xo1_aligned.pdb
```

Keywords:

- *PDB input*
- *structural properties*
- rotamers

Categories:

- core/calc/structural/evaluate_chi

Input files:

- 5fd1.pdb
- 1bgx_aligned.pdb
- 2fd2.pdb
- 1xo1_aligned.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <iomanip>
3 #include <core/data/io/Pdb.hh>
4
5 #include <core/calc/structural/protein_angles.hh>
6 #include <core/chemical/ChiAnglesDefinition.hh>
7 #include <utils/exit.hh>
8 #include <core/calc/structural/angles.hh>
9
10 std::string program_info = R"(
11
12 Unit test which calculates Chi dihedral angles for every pair of amino acid side_
13 ↴chains measured in two different
14 homologous protein structures which are assumed to be aligned.
15
16 USAGE:
17     ex_chi_correlation file-1.pdb file-2.pdb
18
19 EXAMPLE:
```

(continues on next page)

(continued from previous page)

```

19    ex_chi_correlation 1bgx_aligned.pdb 1xol_aligned.pdb
20
21 ) ";
22
23 /** @brief Calculates correlation between Chi dihedral angles measured in two
24  ↪different protein structures.
25 *
26 * CATEGORIES: core/calc/structural/evaluate_chi;
27 * KEYWORDS: PDB input; structural properties; rotamers
28 * IMG: ex_chi_correlation_plot.png
29 * IMG_ALT: Example correlation of chi angles between two homologous structures
30 */
31
32 int main(const int argc, const char* argv[]) {
33
34     if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about
35  ↪missing program parameter
36
37     using namespace core::data::structural;
38     core::data::io::Pdb readerA(argv[1]); // file name (PDB format, may be gzip-ped)
39     Structure_SP proteinA = readerA.create_structure(0);
40     core::data::io::Pdb readerB(argv[2]);
41     Structure_SP proteinB = readerB.create_structure(0);
42
43     std::vector<std::vector<double>> chiA, chiB;
44     std::vector<std::string> labels; // for nice output on a screen
45     std::vector<std::string> mpt_annotationsA; // MPT string - one per residue only
46  ↪(for other lines of a given residue empty strings are inserted)
47     std::vector<std::string> mpt_annotationsB;
48
49     if (proteinA->count_residues() != proteinB->count_residues()) {
50         std::cerr << "The two input PDB files should contain only the aligned parts of
51  ↪input proteins and be equal in length!\n";
52         return 0;
53     }
54     auto a_res_it = proteinA->first_const_residue();
55     auto b_res_it = proteinB->first_const_residue();
56     while(a_res_it!=proteinA->last_const_residue()) {
57
58         if ((*a_res_it)->residue_type() == (*b_res_it)->residue_type()) { // check chi
59  ↪angles
60
61             std::vector<double> ca, cb;
62             for (core::index2 k = 1; k <= core::chemical::ChiAnglesDefinition::count_chi_
63  ↪angles((*a_res_it)->residue_type()); ++k) {
64                 try {
65                     double aA = core::calc::structural::evaluate_chi((**a_res_it), k);
66                     double aB = core::calc::structural::evaluate_chi((**b_res_it), k);
67                     ca.push_back(aA);
68                     cb.push_back(aB);
69                     labels.push_back(utils::string_format("%4d%c %4d%c %c %ld", (*a_res_it)->
70  ↪id(), (*a_res_it)->icode(),
71                     (*b_res_it)->id(), (*b_res_it)->icode(), (*a_res_it)->residue_type().
72  ↪code1, k));
73                 } catch (utils::exceptions::AtomNotFound e) {
74                     std::cerr << e.what() << "\n";
75                     std::cerr << "Can't define chi angle for residue " << (**a_res_it) << "\n";
76                 }
77             }
78         }
79     }

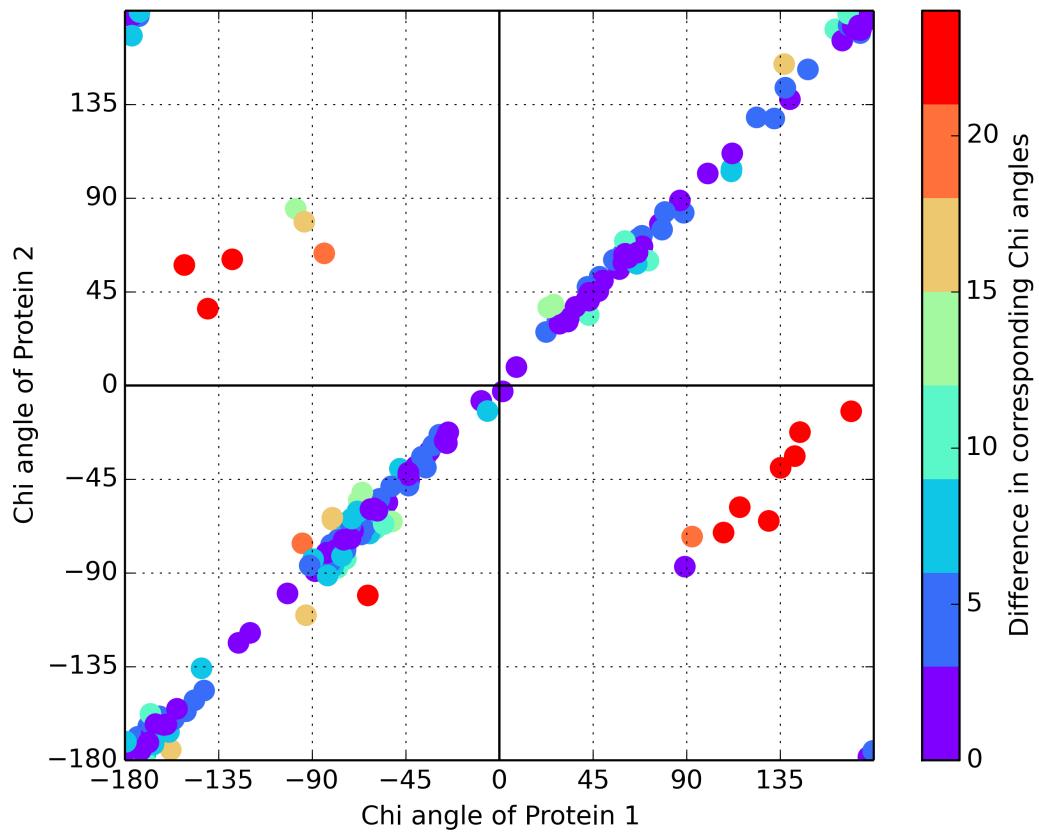
```

(continues on next page)

(continued from previous page)

```

68     }
69     if (ca.size() == cb.size() && ca.size() > 0) {
70         chiA.push_back(ca);
71         chiB.push_back(cb);
72         mpt_annotationsA.push_back(core::calc::structural::define_rotamer(ca));
73         mpt_annotationsB.push_back(core::calc::structural::define_rotamer(cb));
74     }
75 }
76 ++a_res_it;
77 ++b_res_it;
78 } // end of while loop over residues
79
80 std::cout << "#ires jres aa k ichi_k jchi_k delta(chi) irot jrot\n";
81 double err = 0.0;
82 core::index2 n_reoriented = 0;
83
84 size_t ilabel=0;
85 for(size_t ires=0;ires<chiA.size();++ires) {
86     for (size_t i = 0; i < chiA[ires].size(); ++i) {
87         double e = fabs(chiA[ires][i] - chiB[ires][i]);
88         if (e > M_PI) e = 2.0 * M_PI - e;
89         if (e > 0.523) ++n_reoriented; // --- i.e. 30 degrees of error
90         std::cout << labels[ilabel] << utils::string_format("%8.2f %8.2f %8.2f",
91             core::calc::structural::to_degrees(chiA[ires][i]), core::calc::structural::to_
92             degrees(chiB[ires][i]),
93             core::calc::structural::to_degrees(e));
94         if (i == 0)
95             std::cout << std::setw(5) << mpt_annotationsA[ires] << std::setw(5) << mpt_
96             annotationsB[ires] << "\n";
97         else std::cout << "\n";
98         err += e;
99         ++ilabel;
100    }
101 }
102 std::cout << "# avg_err, n_diff, n_res: " << err / double(chiA.size()) << " "
103             << n_reoriented << "#<<chiA.size()<<"\n";
104 }
```



ex_evaluate_chi

Calculates all side chain Chi dihedral angles for the input protein structure

USAGE:

```
ex_evaluate_chi input.pdb
```

EXAMPLE:

```
ex_evaluate_chi 2kwi.pdb
```

Keywords:

- *PDB input*
- *structural properties*
- *structure validation*

Categories:

- core::chemical::ChiAnglesDefinition; core::calc::structural::evaluate_chi()

Input files:

- 2kwi.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4
5 #include <core/chemical/ChiAnglesDefinition.hh>
6 #include <core/calc/structural/protein_angles.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(

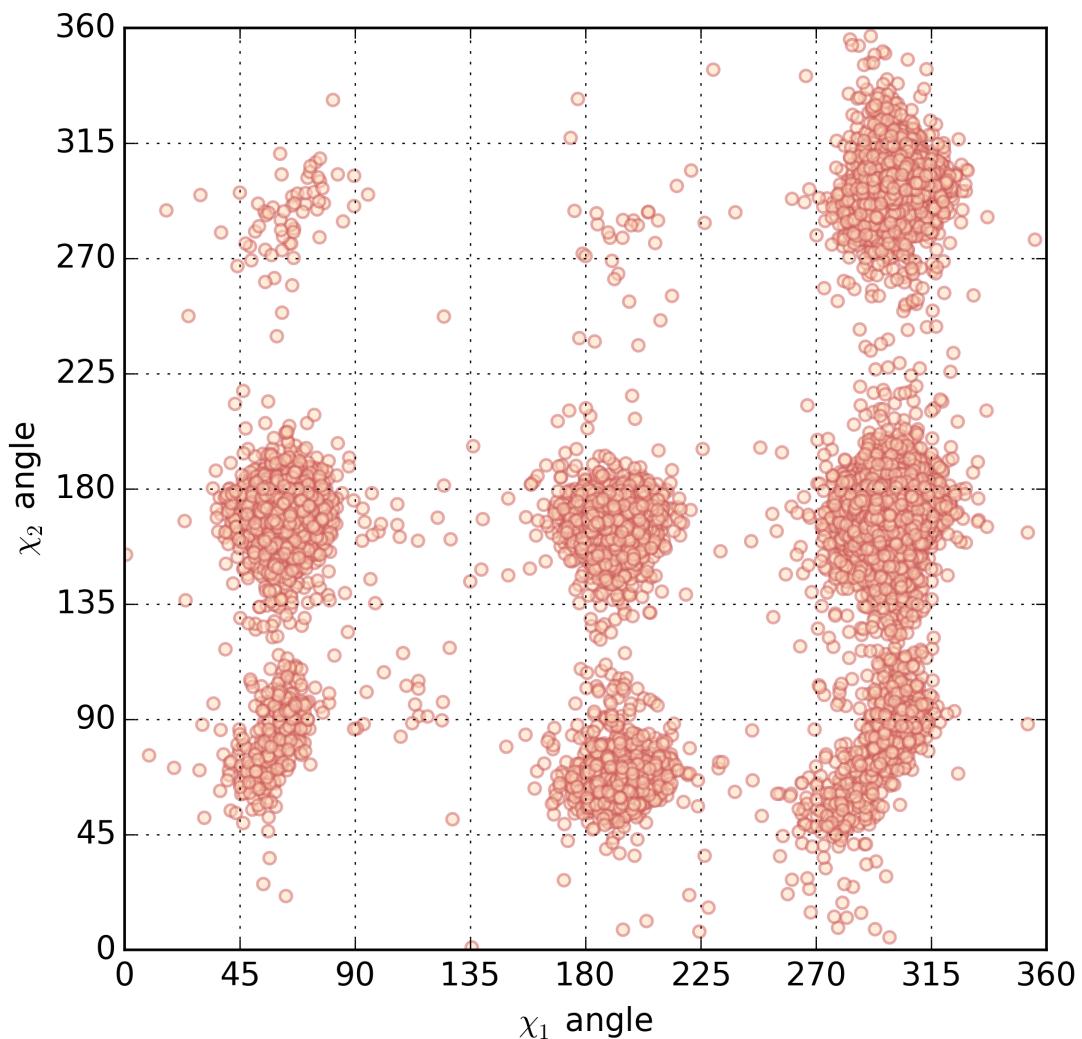
10 Calculates all side chain Chi dihedral angles for the input protein structure
11 USAGE:
12     ex_evaluate_chi input.pdb
13 EXAMPLE:
14     ex_evaluate_chi 2kwi.pdb
15 )
16
17 /**
18 * @brief Calculates all side chain Chi dihedral angles for the input protein
19 * structure
20 *
21 * CATEGORIES: core::chemical::ChiAnglesDefinition; core::calc::structural::evaluate_
22 * chi()
23 * KEYWORDS: PDB input; structural properties; structure validation
24 * IMG: ex_evaluate_chi.png
25 * IMG_ALT: Ch1-Chi2 statistics for ILE residue
26 */
27 int main(const int argc, const char *argv[]) {
28
29     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
30     // missing program parameter
31
32     core::data::io::Pdb reader(argv[1]); // Create a PDB reader for a given file
33     core::data::structural::Structure_SP str = reader.create_structure(
34         0); // Create a structure object from the first model
35
36     for (auto ires = str->first_residue(); ires != str->last_residue(); ++ires) { //_
37     // iterate over all residues
38         std::string line = utils::string_format("%4d %3s %4s", (*ires)->id(), (*ires)->
39         //residue_type().code3.c_str(), (*ires)->owner()->id().c_str());
40
41         try {
42             for (unsigned short i = 1; i <= core::chemical::ChiAnglesDefinition::count_chi_
43             angles((*ires)->residue_type()); ++i)
44                 line += utils::string_format(" %6.1f", core::calc::structural::evaluate_
45             chi(**ires, i) * 180.0 / 3.1415);
46         }
47     }
48 }
```

(continues on next page)

(continued from previous page)

```

40     std::cout << line << "\n";
41 } catch(const std::exception& e) {
42     std::cerr << "Skipping incomplete residue: "<<line<<"\n";
43 }
44 }
45 }
```



ex_evaluate_phi_psi

Calculates Phi,Psi angles (Ramachandran map) for every model found in the input protein structure

USAGE:

```
ex_evaluate_phi_psi input.pdb [chain-id]
```

EXAMPLE:

```
ex_evaluate_phi_psi 2kwi.pdb B
```

Keywords:

- *PDB input*
- *structural properties*
- *structure validation*

Categories:

- core::calc::structural::LocalBackboneProperties

Input files:

- 2kwi.pdb

Output files:

- stdout.out

Program source:

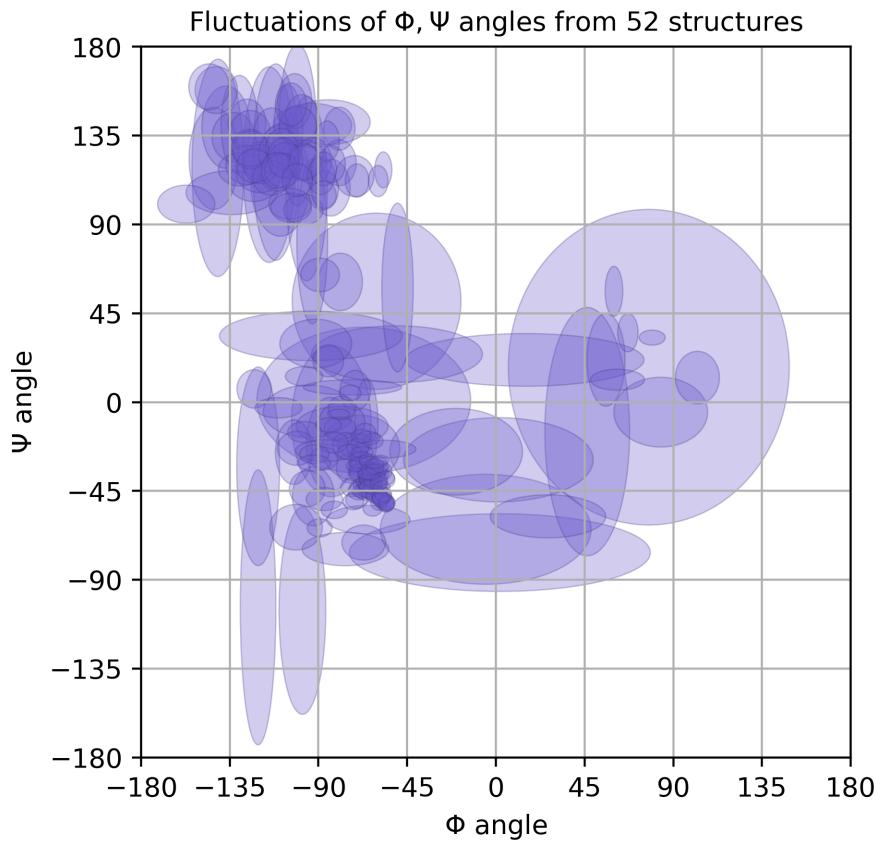
```
1 #include <iostream>
2
3 #include <iostream>
4 #include <core/data/io/Pdb.hh>
5 #include <core/calc/structural/local_backbone_geometry.hh>
6 #include <utils/exit.hh>
7 #include <core/data/structural/ResidueSegmentProvider.hh>
8 #include <core/data/structural/selectors/ResidueSegmentSelector.hh>
9 #include <core/data/structural/selectors>SelectPlanarCAGeometry.hh>
10 #include <core/protocols/selection_protocols.hh>
11
12 std::string program_info = R"(
13
14 Calculates Phi,Psi angles (Ramachandran map) for every model found in the input_
15 ↴protein structure
16 USAGE:
17     ex_evaluate_phi_psi input.pdb [chain-id]
18
19 EXAMPLE:
20     ex_evaluate_phi_psi 2kwi.pdb B
21 )";
22
23 /** @brief Calculates Phi,Psi angles (Ramachandran map) for the input protein_
24 ↴structure
25 *
26 * CATEGORIES: core::calc::structural::LocalBackboneProperties
27 * KEYWORDS: PDB input; structural properties; structure validation
28 * IMG: phi_psi_scatterplot.png
29 * IMG_ALT: Phi,Psi values plotted for every residue of 2KWI PDB deposit; radius of a_
30 ↴circle denotes standard deviation calculated from the NMR ensemble
31 */
32
```

(continues on next page)

(continued from previous page)

```

30 int main(const int argc, const char *argv[]) {
31
32     using namespace core::calc::structural;
33     using namespace core::data::structural;
34     using namespace core::data::io;
35
36     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
37     ↵missing program parameter
38
39     core::data::io::Pdb reader(argv[1], keep_all, keep_all, true); // Create a PDB
40     ↵reader for a given file
41     selectors::HasProperlyConnectedCA is_connected;
42     core::data::structural::selectors::ResidueHasAllHeavyAtoms check_atoms;
43     core::data::structural::selectors::SelectPlanarCAGeometry if_flat;
44     Phi phi(1);
45     Psi psi(1);
46     for(int i=0;i<reader.count_models();++i) {
47         core::data::structural::Structure_SP str = reader.create_structure(i);
48         if (argc==3) {
49             core::data::structural::selectors::ChainSelector pick_chain(argv[2]);
50             core::protocols::keep_selected_chains(pick_chain, *str);
51         }
52         ResidueSegmentProvider rsp(str, 3);
53         while (rsp.has_next()) {
54             const ResidueSegment_SP seg = rsp.next();
55             if (is_connected(*seg)) {
56                 for(int i=0;i<3;++i) {
57                     if (if_flat((*seg)[i])) break;
58                     if (!check_atoms((*seg)[i])) break;
59                 }
60                 const auto &res = *(*seg)[1];
61                 std::cout << utils::string_format("%4s %s %4d %3s ", str->code().c_str(), ↵
62                 ↵res.owner()->id().c_str(), res.id(), res.residue_type().code3.c_str());
63                 std::cout << seg->sequence()->sequence << " " << seg->sequence()->str() << " ";
64                 std::cout << utils::string_format(phi.format(), phi(*seg)) << " ";
65                 std::cout << utils::string_format(psi.format(), psi(*seg)) << "\n";
66             }
67         }
68     }
69 }
```



ex_plot_VonMises_mixture

ex_plot_VonMises_mixture evaluates a mixture of Von Mises distribution so it can be plotted nicely

USAGE:

```
ex_plot_VonMises_mixture scaling mu kappa [scaling2 mu2 kappa2 ...]
```

EXAMPLE:

```
ex_plot_VonMises_mixture 0.487862 -3.00582 17.4059 0.0794212 -1.02886 112.164
```

where the six numbers are scaling, mean and spread of two VonMises distributions

REFERENCE: <http://mathworld.wolfram.com/vonMisesDistribution.html> https://en.wikipedia.org/wiki/Von_Mises_distribution

Keywords:

- *statistics*

Categories:

- core/calc/statistics/VonMisesDistribution

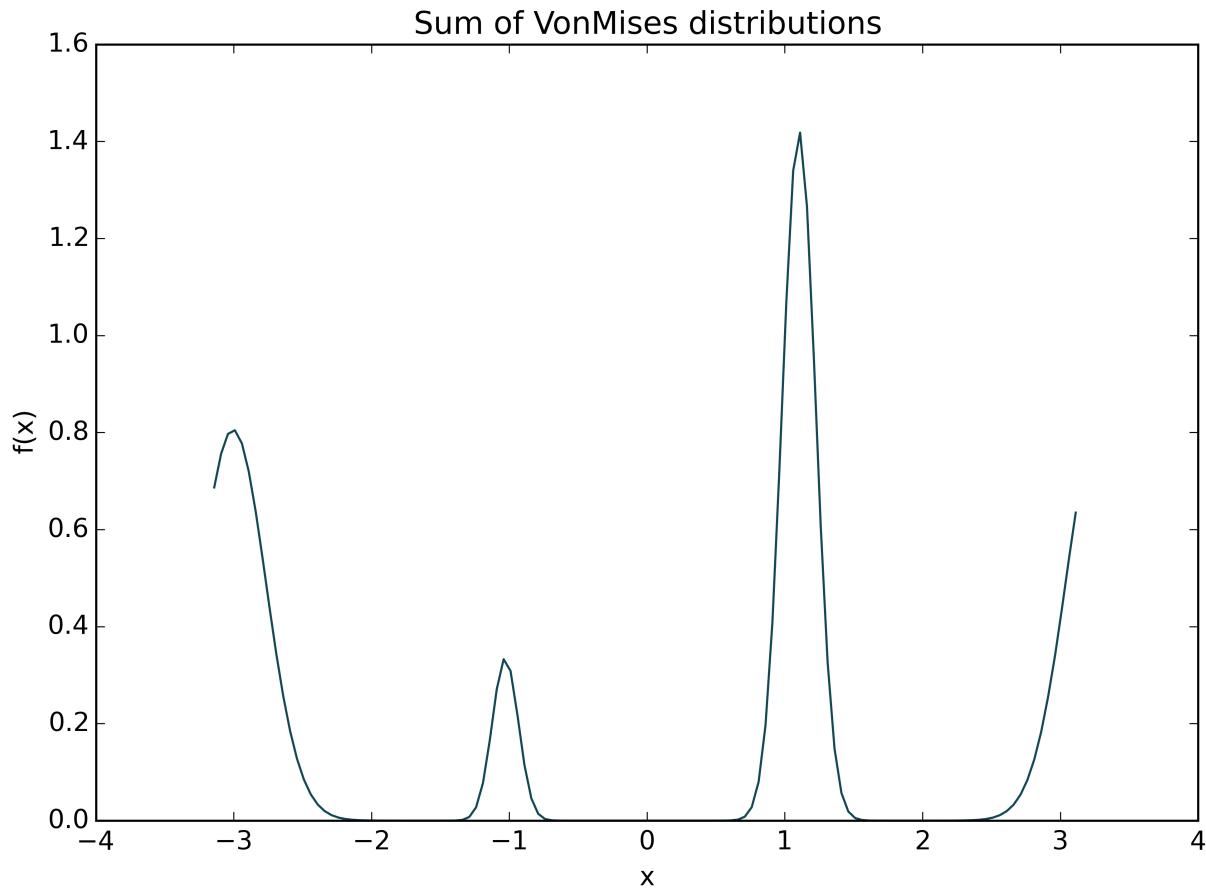
Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <random>
3 #include <cstdlib>
4
5 #include <core/calc/statistics/VonMisesDistribution.hh>
6 #include <core/calc/statistics/Random.hh>
7
8 std::string program_info = R"(
9
10 ex_plot_VonMises_mixture evaluates a mixture of Von Mises distribution so it can be_
11 ↪plotted nicely
12
13 USAGE:
14     ex_plot_VonMises_mixture scaling mu kappa [scaling2 mu2 kappa2 ...]
15
16 EXAMPLE:
17     ex_plot_VonMises_mixture 0.487862 -3.00582 17.4059 0.0794212 -1.02886 112.164
18
19 where the six numbers are scaling, mean and spread of two VonMises distributions
20
21 REFERENCE:
22     http://mathworld.wolfram.com/vonMisesDistribution.html
23     https://en.wikipedia.org/wiki/Von_Mises_distribution
24 )";
25
26 /**
27  * @brief Example which evaluates a mixture of Von Mises distribution so it can be_
28  * ↪plotted nicely
29  * @CATEGORIES: core/calc/statistics/VonMisesDistribution
30  * @KEYWORDS: statistics
31  * @IMG: ex_plot_VonMises_mixture.png
32  * @IMG_ALT: Mixture of von Mises functions plotted
33  */
34
35 int main(const int argc, const char *argv[]) {
36
37     using namespace core::calc::statistics;
38
39     std::vector<double> factors;
40     std::vector<VonMisesDistribution> components;
41     for (size_t i = 1; i < argc; i += 3) {
42         factors.push_back(atof(argv[i]));
43         components.push_back(VonMisesDistribution(atof(argv[i + 1]), atof(argv[i + 2])));
44     }
45
46     for (double x = -M_PI; x <= M_PI; x += M_PI / 62.8) {
47         double val = 0;
48         for (size_t i = 0; i < components.size(); ++i) val += factors[i] * components[i].
49             ↪evaluate(x);
50         std::cout << utils::string_format("%6.3f %9f\n", x, val);
51     }
52 }
```



ex_Array2DSymmetric

Unit test which demonstrates how to use Array2DSymmetric class. The test fills a matrix with random data and prints it on the screen.

USAGE:

```
./ex_Array2DSymmetric
```

Keywords:

- *data structures*
- *random numbers*

Categories:

- core:::data::basic::Array2DSymmetric

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <core/data/basic/Array2DSymmetric.hh>
3 #include <core/calc/statistics/Random.hh>
4 #include <utils/options/OptionParser.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Unit test which demonstrates how to use Array2DSymmetric class. The test fills a_
10 ↵matrix with random data
11 and prints it on the screen.
12
13 USAGE:
14     ./ex_Array2DSymmetric
15 )";
16
17 /** @brief Simple test for Array2DSymmetric class.
18 *
19 * The test fills a matrix with random data and prints it on the screen.
20 *
21 * CATEGORIES: core::data::basic::Array2DSymmetric
22 * KEYWORDS: data structures; random numbers
23 */
24 int main(const int argc, const char* argv[]) {
25
26     if ((argc > 1) && utils::options::call_for_help(argv[1]))
27         utils::exit_OK_with_message(program_info);
28
29     core::calc::statistics::Random r = core::calc::statistics::Random::get();
30     std::uniform_int_distribution<core::index1> uniform_bytes;
31     r.seed(12345);
32     core::data::basic::Array2DSymmetric<core::index1> m(10);
33     for (core::index4 n = 0; n < 1000; ++n) {
34         core::index1 i = uniform_bytes(r) % 10;
35         core::index1 j = uniform_bytes(r) % 10;
36         m.set(i, j, uniform_bytes(r));
37     }
38     m.print("%4d", std::cout);
39 }
```



ex_AtomSelector

Simple example showing how to use atom selectors. Each selector returns true or false. This example uses selector to check, if:

- an atom is an alpha-carbon (core::data::structural::IsCA)
- an atom is a beta-carbon (core::data::structural::IsCB)
- an atom is in backbone (core::data::structural::IsBB)
- an atom is of the specified element (core::data::structural::IsElement)
- an atom is either beta-carbon or a backbone atom (core::data::structural::IsBBCB)
- an atom is of the specified name (core::data::structural::IsNamedAtom)
- an atom is neither beta-carbon nor a backbone atom (core::data::structural::InverseAtomSelector of core::data::structural::IsBBCB)

USAGE:

```
./ex_AtomSelector
```

```
)";
```

```
std::string thr = R"(ATOM 726 N THR A 49 16.822 -5.118 -7.249 1.00 0.00 N ATOM 727 CA THR A 49 18.249  
-4.825 -7.180 1.00 0.00 C ATOM 728 C THR A 49 18.495 -3.354 -6.872 1.00 0.00 C ATOM 729 O THR A 49 19.599  
-2.845 -7.066 1.00 0.00 O ATOM 730 CB THR A 49 18.965 -5.191 -8.493 1.00 0.00 C ATOM 731 OG1 THR A 49  
18.016 -5.723 -9.426 1.00 0.00 O ATOM 732 CG2 THR A 49 20.053 -6.223 -8.238 1.00 0.00 C ATOM 733 H THR  
A 49 16.231 -4.547 -7.836 1.00 0.00 H ATOM 734 HA THR A 49 18.702 -5.391 -6.366 1.00 0.00 H ATOM 735 HB  
THR A 49 19.411 -4.291 -8.916 1.00 0.00 H ATOM 736 HG1 THR A 49 17.144 -5.733 -9.024 1.00 0.00 H ATOM  
737 1HG2 THR A 49 20.548 -6.468 -9.177 1.00 0.00 H ATOM 738 2HG2 THR A 49 20.782 -5.816 -7.538 1.00 0.00  
H ATOM 739 3HG2 THR A 49 19.607 -7.123 -7.817 1.00 0.00 H
```

Keywords:

- *PDB input*
- *structure selectors*

Categories:

- core/data/structural/structure_selectors.hh

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <iomanip>           // for std::setw()
3 #include <ios>                // for std::boolalpha
4 #include <sstream>
5 #include <core/data/io/Pdb.hh>
6 #include <core/data/structural/selectors/structure_selectors.hh>
7 #include <utils/options/OptionParser.hh>
8 #include <utils/exit.hh>
9
10 std::string program_info = R"(
11
12 Simple example showing how to use atom selectors.
```

(continues on next page)

(continued from previous page)

```

13
14 Each selector returns true or false. This example uses selector to check, if:
15     - an atom is an alpha-carbon (core::data::structural::IsCA)
16     - an atom is a beta-carbon (core::data::structural::IsCB)
17     - an atom is in backbone (core::data::structural::IsBB)
18     - an atom is of the specified element (core::data::structural::IsElement)
19     - an atom is either beta-carbon or a backbone atom ↵
20     ↵(core::data::structural::IsBBCB)
21         - an atom is of the specified name (core::data::structural::IsNamedAtom)
22         - an atom is neither beta-carbon nor a backbone atom
23             (core::data::structural::InverseAtomSelector of ↵
24             ↵core::data::structural::IsBBCB)

25
26 USAGE:
27 ./ex_AtomSelector
28
29 ) ";
30 std::string thr = R"(ATOM      726   N    THR A  49      16.822  -5.118  -7.249  1.00  0.
31   ↵00
32       N
33 ATOM    727   CA    THR A  49      18.249  -4.825  -7.180  1.00  0.00      C
34 ATOM    728   C     THR A  49      18.495  -3.354  -6.872  1.00  0.00      C
35 ATOM    729   O     THR A  49      19.599  -2.845  -7.066  1.00  0.00      O
36 ATOM    730   CB    THR A  49      18.965  -5.191  -8.493  1.00  0.00      C
37 ATOM    731   OG1   THR A  49      18.016  -5.723  -9.426  1.00  0.00      O
38 ATOM    732   CG2   THR A  49      20.053  -6.223  -8.238  1.00  0.00      C
39 ATOM    733   H     THR A  49      16.231  -4.547  -7.836  1.00  0.00      H
40 ATOM    734   HA    THR A  49      18.702  -5.391  -6.366  1.00  0.00      H
41 ATOM    735   HB    THR A  49      19.411  -4.291  -8.916  1.00  0.00      H
42 ATOM    736   HG1   THR A  49      17.144  -5.733  -9.024  1.00  0.00      H
43 ATOM    737   1HG2  THR A  49      20.548  -6.468  -9.177  1.00  0.00      H
44 ATOM    738   2HG2  THR A  49      20.782  -5.816  -7.538  1.00  0.00      H
45 ATOM    739   3HG2  THR A  49      19.607  -7.123  -7.817  1.00  0.00      H
46 ) ";
47
48 /**
49 * @brief Demonstrates how to use atom selectors.
50 */
51
52 int main(const int argc, const char* argv[]) {
53
54     if ((argc > 1) && utils::options::call_for_help(argv[1]))
55         utils::exit_OK_with_message(program_info);
56
57     std::stringstream in(thr);                                // Create an input stream that will
58     ↵provide data from a string
59     core::data::io::Pdb reader(in,                         // data stream
60                               core::data::io::keep_all());           // a predicate to read ALL the ATOM lines
61     ↵(hydrogens are excluded by default)
62     core::data::structural::Structure_SP strctr = reader.create_structure(0);
63
64     core::data::structural::selectors::IsCA ca_test;
65     core::data::structural::selectors::IsCB cb_test;
66     core::data::structural::selectors::IsBB bb_test;
67     core::data::structural::selectors::IsElement is_H("H");
68     core::data::structural::selectors::IsBBCB bb_cb_test;

```

(continues on next page)

(continued from previous page)

```

65 core:::data::structural::selectors::InverseAtomSelector not_bb_cb(bb_cb_test);
66 core:::data::structural::selectors::IsNamedAtom is_ogl(" OG1"); // note the u
67 →padding for four characters!
68 std::cout << "atom is_CA is_CB is_BB is_H is_OG1 is_bb_CB !is_bb_
69 →CB\n; 
70     for(auto ai = strctr->first_atom(); ai != strctr->last_atom(); ++ai)
71         std::cout << (*ai)->atom_name() << " "
72             << std::setw(5) << std::boolalpha << ca_test(**ai) << " "
73             << std::setw(5) << std::boolalpha << cb_test(**ai) << " "
74             << std::setw(5) << std::boolalpha << bb_test(**ai) << " "
75             << std::setw(5) << std::boolalpha << is_H(**ai) << " "
76             << std::setw(5) << std::boolalpha << is_ogl(**ai) << " "
77             << std::setw(5) << std::boolalpha << bb_cb_test(**ai) << " "
78             << std::setw(5) << std::boolalpha << not_bb_cb(**ai) << "\n";
79 }
```



ex_AtomicElement

Unit test which shows how to use AtomicElement class. It prints all the element names

USAGE:

```
./ex_AtomicElement
```

Keywords:

- chemical elements

Categories:

- core::chemical::AtomicElement

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <iomanip>
3
4 #include <core/chemical/AtomicElement.hh>
5 #include <utils/options/OptionParser.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Unit test which shows how to use AtomicElement class. It prints all the element names
11
12 USAGE:
13 ./ex_AtomicElement
14
15 )";
16
17 /** @brief Example showing how to use AtomicElement class
18 *
19 * CATEGORIES: core::chemical::AtomicElement
20 * KEYWORDS: chemical elements
21 */
22 int main(int argc, char *argv[]) {
23
24     if ((argc > 1) && utils::options::call_for_help(argv[1]))
25         utils::exit_OK_with_message(program_info);
26
27     using namespace core::chemical;
28
29     if (argc < 2)
30         for (const std::pair<std::string, AtomicElement> &e : AtomicElement::elements_by_
31             symbol)
```

(continues on next page)

(continued from previous page)

```
31     std::cout << e.second << std::endl;
32
33     else
34         for (int i = 1; i < argc; i++) std::cout << AtomicElement::by_symbol(argv[i]) <<
35             "\n";
36 }
```



ex_BetaStructuresGraph

Reads a PDB file, creates a BetaStructuresGraph for it and finds all strands as connected components of that graph

USAGE:

```
ex_BetaStructuresGraph 5edw.pdb
```

Keywords:

- *PDB input*

Categories:

- core::data::structural::BetaStructuresGraph

Input files:

- 2fdo.pdb

Output files:

- stdout.out

Program source:

```
1 #include <core/data/io/Pdb.hh>
2 #include <utils/exit.hh>
3 #include <core/algorithms/graph_algorithms.hh>
4 #include <core/data/structural/BetaStructuresGraph.hh>
5 #include <core/calc/structural/ProteinArchitecture.hh>
6
7 std::string program_info = R"(
8
9 Reads a PDB file, creates a BetaStructuresGraph for it and finds all strands as
10    ↵connected components of that graph
11 USAGE:
12     ex_BetaStructuresGraph 5edw.pdb
13 )";
14
15 /** @brief Creates a BetaStructuresGraph and finds all strands
16 *
17 * CATEGORIES: core::data::structural::BetaStructuresGraph
18 * KEYWORDS:   PDB input
19 */
20 int main(const int argc, const char* argv[]) {
21
22     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
23    ↵missing program parameter
24
25     using namespace core::data::structural;
```

(continues on next page)

(continued from previous page)

```
25  using namespace core::data::io;
26
27  core::data::io::Pdb reader(argv[1],is_not_alternative,only_ss_from_header, true);
28  core::data::structural::Structure_SP strctr = reader.create_structure(0);
29
30  core::calc::structural::ProteinArchitecture a(*strctr);
31  BetaStructuresGraph_SP g = a.create_strand_graph();
32  auto sheets = core::algorithms::connected_components<BetaStructuresGraph, Strand_SP,
33  ↪ StrandPairing_SP>(*g);
34  int cnt = 0;
35  for(const auto & sheet: sheets) {
36      std::cout << utils::string_format("----- Sheet %d -----\\n",
37      ↪ ++cnt);
38      for(auto it=sheet->cbegin_strand();it!=sheet->cbegin_strand();++it)
39          std::cout << **it<<"\\n";
    }
}
```



ex_BioShellVersion

Unit test for BioShellVersion class which prints the BioShell version info - a string that unambiguously describes code version (Git SHA and branch) and compilation time (Git timestamp). Note, that the output changes with every git / cmake operation

USAGE:

```
./ex_BioShellVersion
```

Keywords:

- *BioShell*

Categories:

- core/BioShellVersion

Program source:

```

1 #include <iostream>
2 #include <core/BioShellVersion.hh>
3 #include <utils/exit.hh>
4 #include <utils/options/OptionParser.hh>
5
6 std::string program_info = R"(
7
8 Unit test for BioShellVersion class which prints the BioShell version info - a string
9 ↪that unambiguously
10 describes code version (Git SHA and branch) and compilation time (Git timestamp).
11 Note, that the output changes with every git / cmake operation
12
13 USAGE:
14 ./ex_BioShellVersion
15 )";
16
17 /**
18 * @brief Test for BioShellVersion class prints the BioShell version info
19 *
20 * CATEGORIES: core/BioShellVersion
21 * KEYWORDS: bioshell
22 */
23 int main(const int argc, const char* argv[]) {
24
25     if ((argc > 1) && utils::options::call_for_help(argv[1]))
26         utils::exit_OK_with_message(program_info);
27
28     std::cout << "BioShell boilerplate:\n";
29     std::cout << core::BioShellVersion() << "\n";
30 }
```



ex_BivariateNormal

Estimates parameters of a two-dimensional Gaussian distribution. The program expects a file with columns of real values; based on them parameters of the distributions are estimated. Otherwise the example withdraws 10000 random numbers from a normal distribution and later it estimates a normal distribution from the sample.

USAGE:

```
ex_BivariateNormal infile [x_column y_column]
```

EXAMPLE:

```
./ex_BivariateNormal bivariate_normal.dat 0 1
```

where x_column y_column are optional parameters that indicate which columns should be used for estimation; by default columns 0 and 1 are used.

Keywords:

- *statistics*
- *random numbers*
- *estimation*

Categories:

- core::calc::statistics::BivariateNormal; core::calc::statistics::Random

Input files:

- bivariate_normal.dat

Output files:

- stdout.out

Program source:

```

1 #include <math.h>
2
3 #include <iostream>
4 #include <random>
5
6 #include <core/data/io/DataTable.hh>
7 #include <core/calc/statistics/Random.hh>
8 #include <core/calc/statistics/BivariateNormal.hh>
9 #include <core/calc/statistics/RobustDistributionDecorator.hh>
10
11 std::string program_info = R"(
12
13 Estimates parameters of a two-dimensional Gaussian distribution

```

(continues on next page)

(continued from previous page)

```

14
15 The program expects a file with columns of real values; based on them parameters of ↵
16 ↵the distributions are estimated.
17 Otherwise the example withdraws 10000 random numbers from a normal distribution and ↵
18 ↵later it estimates
19 a normal distribution from the sample.
20
21 USAGE:
22   ex_BivariateNormal infile [x_column y_column]
23
24 EXAMPLE:
25   ./ex_BivariateNormal bivariate_normal.dat 0 1
26
27 where x_column y_column are optional parameters that indicate which columns should be ↵
28 ↵used for estimation;
29 by default columns 0 and 1 are used.
30
31 ) ";
32
33 /**
34 * @brief Estimates parameters of a two-dimensional Gaussian distribution
35 *
36 * CATEGORIES: core::calc::statistics::BivariateNormal; core::calc::statistics::Random
37 * KEYWORDS: statistics; random numbers; estimation
38 */
39
40 int main(const int argc, const char* argv[]) {
41
42     using namespace core::calc::statistics;
43
44     std::vector<std::vector<double> > data_2D;
45     std::vector<double> row(2);
46     if (argc == 1) { // --- No input file? Generate random data for the test
47
48         std::cerr << program_info;
49
50         Random rd = core::calc::statistics::Random::get();
51         rd.seed(12345); // --- seed the generator for repeatable results
52         unsigned N = 100000; //--- the number of random points to use in tests
53         core::calc::statistics::NormalRandomDistribution<double> nX(1.0, 2.5);
54         core::calc::statistics::NormalRandomDistribution<double> nY(2.0, 0.7);
55
56         for (unsigned i = 0; i < N; ++i) { // --- get a random sample in 2D
57             double x = nX(rd);
58             double y = nY(rd);
59             row[0] = x - y; // --- make X variable correlated with Y
60             row[1] = x + y;
61             data_2D.push_back(row);
62         }
63     } else {
64         core::data::io::DataTable in_data(argv[1]);
65         int column_x_id = 0, column_y_id = 1;
66         if (argc > 3) {
67             column_x_id = utils::from_string<int>(argv[2]);
68             column_y_id = utils::from_string<int>(argv[3]);
69         }
70         for (const auto &data_row : in_data) {
71             row[0] = data_row.get<double>(column_x_id);
72             row[1] = data_row.get<double>(column_y_id);
73         }
74     }
75 }
```

(continues on next page)

(continued from previous page)

```

68     data_2D.push_back(row);
69 }
70 }
71
72 std::vector<double> initial_parameters{0.0, 0.0, 1.0, 1.0, 1.0};
73 // --- Here we declare a 2D normal distribution ...
74 core::calc::statistics::BivariateNormal n(initial_parameters);
75 // ... and estimate its parameters
76 const std::vector<double> & params = n.estimate(data_2D);
77 // show the estimated parameters of the distribution
78 std::cout << "estimated parameters: " << params[0] << " " << params[1] <<
79 " " << params[2] << " " << params[3] << " " << params[4] << "\n";
80 // ... and estimate its parameters
81 core::calc::statistics::RobustDistributionDecorator<BivariateNormal> rn(initial_
82 -parameters, 0.05);
83 const std::vector<double> & params_r = rn.estimate(data_2D);
84 // show the estimated parameters of the distribution
85 std::cout << "estimated parameters (robust): " << params_r[0] << " " << params_
86 -r[1] << " " << params_r[2] << " " << params_r[3] << " " << params_r[4] << "\n";
87 }
```



ex_BoundedPriorityQueue

Unit test which shows how to use the BoundedPriorityQueue data structure. BoundedPriorityQueue is a sorted queue with pre-defined maximum capacity. It's purpose is to keep N best elements of what was inserted to the queue. Overflow elements are removed from the queue

USAGE:

```
./ex_BoundedPriorityQueue
```

Keywords:

- *algorithms*
- *data structures*
- BoundedPriorityQueue

Categories:

- core::data::basic::BoundedPriorityQueue

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <random>
3
4 #include <core/index.hh>
5 #include <core/calc/statistics/Random.hh>
6 #include <core/data/basic/BoundedPriorityQueue.hh>
7 #include <utils/options/OptionParser.hh>
8 #include <utils/exit.hh>
9
10 std::string program_info = R"(
11
12 Unit test which shows how to use the BoundedPriorityQueue data structure.
13
14 BoundedPriorityQueue is a sorted queue with pre-defined maximum capacity. It's_
15 ↵purpose is to keep N best elements
16 of what was inserted to the queue. Overflow elements are removed from the queue
17
18 USAGE:
19 ./ex_BoundedPriorityQueue
20
21 )";
22
23 using namespace core::data::basic;
24
25 /**
26  * @brief Simple demo for BoundedPriorityQueue class

```

(continues on next page)

(continued from previous page)

```

25
26 * This program creates a BoundedPriorityQueue and fills it with random numbers.
27 * When printed, they should be ordered descending
28 *
29 * CATEGORIES: core:::data:::basic:::BoundedPriorityQueue
30 * KEYWORDS: algorithms; data structures; BoundedPriorityQueue
31 */
32 int main(int cnt, char *argv[]) {
33
34     if ((cnt > 1) && utils::options::call_for_help(argv[1]))
35         utils::exit_OK_with_message(program_info);
36
37     // ----- test on real values
38     typedef std::function<bool(const double, const double)> ComparatorType;
39     core:::data:::basic:::BoundedPriorityQueue<double, ComparatorType, ComparatorType> q(
40         [&](double x, double y) { return x > y; },
41         [&](double x, double y) { return x == y; }, 10, 20, -std::numeric_limits<double>
42         ::max());
43     core:::calc:::statistics::Random r = core:::calc:::statistics::Random::get();
44     std::uniform_real_distribution<float> flat_f(0, 10.0);
45     for (core:::index4 i = 0; i < 30; ++i)
46         q.push(flat_f(r));
47     for (core:::index4 i = 1; i < q.size(); ++i) {
48         std::cout << q[i] << " ";
49         if (q[i - 1] < q[i])
50             std::cerr
51                 << utils::string_format("Incorrect ordering in a bounded priority queue, %f<
52                 before %f\n", q[i - 1], q[i]);
53     }
54
55     std::cout << "\n";
56
57     // ----- test on integers
58     typedef std::function<bool(int, int)> ComparatorTypeI;
59     core:::data:::basic:::BoundedPriorityQueue<int, ComparatorTypeI, ComparatorTypeI> q_i(
60         [&](int x, int y) { return x > y; },
61         [&](int x, int y) { return x == y; }, 10, 20, -std::numeric_limits<int>::max());
62     std::uniform_int_distribution<int> flat(0, 20);
63     for (core:::index4 i = 0; i < 30; ++i)
64         q_i.push(flat(r));
65     for (core:::index4 i = 1; i < q_i.size(); ++i) {
66         std::cout << q_i[i] << " ";
67         if (q_i[i - 1] < q_i[i])
68             std::cerr
69                 << utils::string_format("Incorrect ordering in a bounded priority queue, %f<
70                 before %f\n", q_i[i - 1], q_i[i]);
71     }
72 }
```



ex_BuildPolymerChain

Creates a mixture of simple polymer chains in a periodic box

USAGE:

```
./ex_BuildPolymerChain box_width n_chains n_atoms_in_chain
```

Keywords:

- *Chain*
- ResidueChain
- Polymer
- Vec3Cubic

Categories:

- simulations::systems::BuildPolymerChain

Output files:

- out.pdb

Program source:

```
1 #include <iostream>
2 #include <core/data/basic/Vec3Cubic.hh>
3 #include <simulations/systems/BuildPolymerChain.hh>
4 #include <simulations/systems/SingleAtomType.hh>
5 #include <simulations/systems/CartesianChains.hh>
6 #include <simulations/observers/cartesian/PdbObserver.hh>
7 #include <simulations/observers/cartesian/ExplicitPdbFormatter.hh>
8
9
10 #include <utils/exit.hh>
11
12 using namespace simulations::systems;
13 using namespace core::data::structural;
14 using core::data::basic::Vec3Cubic;
15
16 std::string program_info = R"(
17
18 Creates a mixture of simple polymer chains in a periodic box
19
20 USAGE:
21   ./ex_BuildPolymerChain box_width n_chains n_atoms_in_chain
22
23 )";
24
25 /* @brief Creates a mixture of simple polymer chains in a periodic box
```

(continues on next page)

(continued from previous page)

```

27
28 * CATEGORIES: simulations::systems::BuildPolymerChain
29 * KEYWORDS: Chain; ResidueChain; Polymer; Vec3Cubic
30 */
31 int main(const int argc, const char *argv[]) {
32
33     if (argc < 4) utils::exit_OK_with_message(program_info);
34
35     double box = atof(argv[1]);
36     core::index2 n_chains = atoi(argv[2]);
37     core::index2 n_res_each = atoi(argv[3]);
38
39     // --- here we create a Structure object of n_chains polyalanine chains
40     Structure_SP starting_structure = std::make_shared<Structure>("");
41     std::string sequence(n_res_each, 'A'); // --- We create a polyalanine chain, the_
42     //sequence is made by many 'A's
43     for (core::index2 i = 0; i < n_chains; ++i)
44         starting_structure->push_back(Chain::create_ca_chain(sequence, std::string
45         //utils::letters[i] )); // --- A + 1 makes the chain code
46
47     // --- we have to renumber atoms so the indexes are consistent in the whole_
48     //structure
49     core::index4 i_atom = 0;
50     for(Chain_SP m : *starting_structure)
51         std::for_each(m->first_atom(), m->last_atom(), [&](PdbAtom_SP e) { (e)->id(++i_
52         atom); });
53     // Vec3Cubic::set_box_len(box); // --- set periodic box width
54     std::shared_ptr<AtomTypingInterface> atom_typing = std::make_shared<SingleAtomType>
55     //(); // --- simplest atom typing possible
56     CartesianChains chains(atom_typing,*starting_structure);
57     BuildPolymerChain chain_builder(chains);
58     chain_builder.generate(3.8,5.5);
59     core::index4 ai = 0;
60     std::shared_ptr<simulations::observers::cartesian::AbstractPdbFormatter> fmt =
61     std::make_shared<simulations::observers::cartesian::ExplicitPdbFormatter>(*starting_
62     structure);
63     simulations::observers::cartesian::PdbObserver start(chains, fmt, "");
64     start.observe();
65 }

```



ex_Cart

Shows how to use CART classification model

Keywords:

- CART
- *observer*

Categories:

- core::calc::statistics::Cart

Input files:

- LEU_chi1_chi2_rad.dat

Output files:

- out_tree.txt
- stdout.out

Program source:

```

1 #include <iostream>
2 #include <vector>
3 #include <map>
4
5 #include <core/calc/statistics/Cart.hh>
6 #include <core/algorithms/basic_algorithms.hh>
7
8 using namespace core::calc::statistics;
9 using namespace core::data::io;
10 using namespace core::data::basic;
11
12 utils::Logger logs("ex_Cart");
13
14 /** @brief Shows how to use CART classification model
15 *
16 * CATEGORIES: core::calc::statistics::Cart
17 * KEYWORDS: CART; observer
18 */
19 int main(const int argc, const char *argv[]) {
20
21     DataTable dt;
22     dt.load(argv[1]);
23
24     std::vector<LabelledObservationVector_SP> observations;
25     std::vector<std::string> class_names;
26     std::vector<core::index2> class_ids;
27     std::map<std::string, core::index2> class_to_id;

```

(continues on next page)

(continued from previous page)

```

28
29 // --- First find distinct labels
30 core::index2 i_class = 0;
31 for (const TableRow &tr : dt) {
32     if (class_to_id.find(tr.back()) == class_to_id.end()) {
33         class_ids.push_back(i_class);
34         class_to_id[tr.back()] = i_class;
35         ++i_class;
36     }
37 }
38
39 for (const TableRow &tr : dt)
40     observations.push_back(std::make_shared<LabelledObservationVector>(tr, class_to_
41     ↪id[tr.back()], 0, tr.size() - 2));
42
43 // --- print some debug info : known classes etc.
44 logs << utils::LogLevel::INFO << "classification into " << class_ids.size() << "\u
45 ↪classes\n";
46 if (logs.is_logable(utils::LogLevel::INFO)) {
47     logs << utils::LogLevel::INFO << "Known classes:\n";
48     core::index1 icol = 0;
49     for (auto c:class_to_id) {
50         logs << c.first << " ";
51         ++icol;
52         if (icol % 10 == 0) logs << "\n";
53     }
54     logs << "\n";
55 }
56
57 // --- create the CART classifier and train it
58 Cart cart(class_ids);
59 cart.train(observations);
60 std::cout << cart;
61
62 // --- test the classifier for the training data set
63 core::index4 n_ok = 0;
64 for(const auto o : observations) {
65     n_ok += (cart.classify(o) == o->label());
66 }
67 std::cout << utils::string_format("# classification test:\n# success rate: %d of %d
68 ↪(%6.2f%%)\n", n_ok, observations.size(),
69     100.0*n_ok / float(observations.size())));
70 }
```



ex_CartesianToSpherical

Unit test that calculates spherical coordinates from a few points in the Cartesian space using BioShell

USAGE:

```
./ex_CartesianToSpherical
```

)";

```
std::string input_pdb = R"(ATOM 201 N SER A 12 25.081 -7.330 -14.416 1.00 0.00 N ATOM 202 CA SER A 12 25.875 -6.648 -15.435 1.00 0.00 C ATOM 203 C SER A 12 25.030 -6.429 -16.700 1.00 0.00 C ATOM 204 O SER A 12 25.187 -5.429 -17.400 1.00 0.00 O ATOM 205 CB SER A 12 27.126 -7.492 -15.717 1.00 0.00 C ATOM 206 OG SER A 12 27.645 -8.029 -14.500 1.00 0.00 O ATOM 207 H SER A 12 25.486 -8.177 -14.049 1.00 0.00 H
```

Keywords:

- *internal coordinates*

Categories:

- core/calc/structural/CartesianToSpherical

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <core/data/structural/PdbAtom.hh>
5 #include <core/calc/structural/transformations/Rototranslation.hh>
6 #include <core/calc/structural/transformations/transformation_utils.hh>
7 #include <core/calc/structural/transformations/CartesianToSpherical.hh>
8 #include <utils/options/OptionParser.hh>
9
10 std::string program_info = R"(
11
12 Unit test that calculates spherical coordinates from a few points in the Cartesian_
13     ↪space using BioShell
14
15 USAGE:
16 ./ex_CartesianToSpherical
17 )";
18
19 std::string input_pdb = R"(ATOM      201   N    SER A  12      25.081  -7.330 -14.416  1.
20     ↪00  0.00
21             N
22 ATOM      202   CA   SER A  12      25.875  -6.648 -15.435  1.00  0.00
23 ATOM      203   C    SER A  12      25.030  -6.429 -16.700  1.00  0.00
24 ATOM      204   O    SER A  12      25.187  -5.429 -17.400  1.00  0.00
25
26 )";
```

(continues on next page)

(continued from previous page)

```

23 ATOM    205 CB   SER A  12      27.126  -7.492 -15.717  1.00  0.00      C
24 ATOM    206 OG   SER A  12      27.645  -8.029 -14.500  1.00  0.00      O
25 ATOM    207 H    SER A  12      25.486  -8.177 -14.049  1.00  0.00      H) ";
26
27 /** @brief Calculates spherical coordinates using BioShell and 'by hand' to check if it works
28 *
29 * CATEGORIES: core/calc/structural/CartesianToSpherical;
30 * KEYWORDS: internal coordinates
31 */
32 int main(const int argc, const char *argv[]) {
33
34     using namespace core::data::structural;
35     using namespace core::calc::structural::transformations;
36
37     std::stringstream in_stream(input_pdb); // --- Create an input stream from the text
38     core::data::io::Pdb reader(in_stream, core::data::io::keep_all); // --- read from
39     →this stream
40     core::data::structural::Structure_SP strctr = reader.create_structure(0); // ---_
41     →create a structure object
42
43     auto residue = *(strctr->first_residue()); // --- get the residue ...
44     PdbAtom_SP n = residue->find_atom(" N "); // --- and extract three atoms
45     PdbAtom_SP ca = residue->find_atom(" CA "); // --- to form a local coordinate_
46     →system (LCS)
47     PdbAtom_SP c = residue->find_atom(" C ");
48     PdbAtom_SP cb = residue->find_atom(" CB "); // --- CB will be transformed
49
50     Rototranslation_SP rt = local_coordinates_three_atoms(*n, *ca, *c);
51     CartesianToSpherical to_spherical;
52
53     Vec3 cb_local, cb_spherical;
54     rt->apply(*cb, cb_local);
55     to_spherical.apply(cb_local, cb_spherical);
56
57     double r = cb_local.length();
58     double theta = acos(cb_local.z / r);
59     double phi = atan2(cb_local.y, cb_local.x);
60
61     std::cout << "local computed by BioShell: " << cb_local << "\n";
62     std::cout << "spherical by BioShell: " << cb_spherical << "\n";
63     std::cout << "spherical computed here: " << utils::string_format("%8.3f %8.3f %8.
64     →3f\n", r, theta, phi);
65     double x = r * sin(theta) * cos(phi);
66     double y = r * sin(theta) * sin(phi);
67     double z = r * cos(theta);
68     std::cout << "local from inversion: " << utils::string_format("%8.3f %8.3f %8.
69     →3f\n", x, y, z);
70     to_spherical.apply_inverse(cb_spherical);
71     std::cout << "local by BioShell : " << cb_spherical << "\n";
72 }

```



ex_ChiAnglesDefinition

Unit test that shows how to look up information on Chi angle definitions. It prints how many Chi angles are defined for ARG and which atoms define Chi2 of TRP

USAGE:

```
./ex_ChiAnglesDefinition
```

Keywords:

- *structural properties*

Categories:

- core::chemical::ChiAnglesDefinition

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/chemical/ChiAnglesDefinition.hh>
4 #include <utils/options/OptionParser.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Unit test that shows how to look up information on Chi angle definitions. It prints_
10 ↵how many Chi angles
11 are defined for ARG and which atoms define Chi2 of TRP
12
13 USAGE:
14 ./ex_ChiAnglesDefinition
15 )";
16
17 /** @brief Shows how to look up information on Chi angle definitions
18 *
19 * This example prints how many Chi angles are defined for ARG and wish atoms define_
20 ↵Chi2 of TRP
21 *
22 * CATEGORIES: core::chemical::ChiAnglesDefinition
23 * KEYWORDS: structural properties
24 */
25 int main(const int argc, const char *argv[]) {
26
27     if ((argc > 1) && utils::options::call_for_help(argv[1]))
28         utils::exit_OK_with_message(program_info);
29 }
```

(continues on next page)

(continued from previous page)

```
29 // List atoms that define the second Chi angle in TRP residue
30 std::cout << "Chi_2 in TRP:";
31 for (const std::string &a : core::chemical::ChiAnglesDefinition::chi_angle_atoms(
32     "TRP", 2)) // "TRP" defines a residue, "2" stands for Chi_2
33     std::cout << a << " ";
34     std::cout << "\n";
35 const core::chemical::Monomer &m = core::chemical::Monomer::ARG; // Create a local_
36     // reference to ARG monomer (just to make the following lines shorter)
37     std::cout << "\nAll Chi angles for in " << m.code3 << " :\n";
38     for (unsigned short i = 1; i <= core::chemical::ChiAnglesDefinition::count_chi_
39     angles(m); ++i) { // Count how many Chi angles ARG has
40         std::cout << "Chi" << i << " ";
41         for (const std::string &a : core::chemical::ChiAnglesDefinition::chi_angle_
42             atoms(m, i)) // List atoms for each of them
43             std::cout << a << " ";
44             std::cout << "\n";
45     }
46 }
```



ex_Cif

Unit test which shows how to read CIF files.

USAGE:

```
ex_Cif file.cif
```

EXAMPLE:

```
ex_Cif AA3.cif
```

Keywords:

- *CIF input*

Categories:

- core/data/io/Cif

Input files:

- AA3.cif

Output files:

- stdout.out

Program source:

```
1 #include <core/data/io/Cif.hh>
2 #include <utils/Logger.hh>
3 #include <utils/LogManager.hh>
4
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Unit test which shows how to read CIF files.
10
11 USAGE:
12     ex_Cif file.cif
13 EXAMPLE:
14     ex_Cif AA3.cif
15
16 )";
17
18 /**
19 * CATEGORIES: core/data/io/Cif
20 * KEYWORDS:    CIF input
21 */
22
```

(continues on next page)

(continued from previous page)

```
23 int main(const int argc, const char *argv[]) {
24
25     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
26     ↵missing program parameter
27
28     utils::LogManager::FINEST(); // --- INFO is the default logging level; set it to
29     ↵FINE to see more
30     core::data::io::Cif reader(argv[1]);
31     std::cout << reader;
32 }
```



ex_Combinations

Unit test for BioShell's combination generator prints all possible tripeptides by taking all 3-element combinations of 20-elements set.

USAGE:

```
./ex_Combinations
```

Keywords:

- *algorithms*
- *random*

Categories:

- core:::algorithms::Combination

Output files:

- stdout.out

Program source:

```

1 #include <memory>
2 #include <iostream>
3 #include <random>
4 #include <core/algorithms/Combinations.hh>
5
6 #include <utils/options/OptionParser.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 Unit test for BioShell's combination generator prints all possible tripeptides by taking all 3-element
12 combinations of 20-elements set.
13
14 USAGE:
15 ./ex_Combinations
16
17 )";
18
19
20 /** @brief A simple example shows how to generate Combination
21 *
22 * The program generates all possible tripeptides as ${20}\choose{3} combinations
23 *
24 * CATEGORIES: core:::algorithms::Combination;
25 * KEYWORDS: algorithms; random
26 */
27 int main(const int argc, const char *argv[]) {

```

(continues on next page)

(continued from previous page)

```
28
29     if ((argc > 1) && utils::options::call_for_help(argv[1]))
30         utils::exit_OK_with_message(program_info);
31
32     std::vector<std::string> amino_acids{"ALA", "ARG", "ASP", "ASN", "CYS", "PHE", "GLU", "GLN"
33     ↪, "GLY", "HIS", "ILE", "LEU", "LYS", "MET", "PRO", "SER",
34     "THR", "TYR", "TRP", "VAL"};
35
36     std::vector<std::string> a_combination(3);
37     core::algorithms::Combinations<std::string> generator(3, amino_acids);
38     int cnt = 0;
39     while (generator.next(a_combination)) {
40         std::cout << a_combination[0] << " " << a_combination[1] << " " << a_
41     ↪combination[2] << "\n";
42         ++cnt;
43     }
44     std::cout << "# " << cnt << " combinations generated\n";
45 }
```



ex_DsspData

ex_DsspData reads a DSSP file and writes secondary structure in FASTA format

USAGE:

```
ex_DsspData input.dssp
```

EXAMPLE:

```
ex_DsspData 5edw.dssp
```

Keywords:

- *DSSP*
- *FASTA output*
- *Structure*
- *secondary structure*
- *Format conversion*

Categories:

- core/data/io/DsspData

Input files:

- 5edw.dssp

Output files:

- stderr.out
- stdout.out

Program source:

```
1 #include <iostream>
2 #include <core/data/io/fasta_io.hh>
3 #include <core/data/io/DsspData.hh>
4 #include <utils/exit.hh>
5
6 std::string program_info = R"(
7
8 ex_DsspData reads a DSSP file and writes secondary structure in FASTA format
9
10 USAGE:
11     ex_DsspData input.dssp
12 EXAMPLE:
13     ex_DsspData 5edw.dssp
```

(continues on next page)

(continued from previous page)

```

14
15 ) ";
16
17 /** @brief Reads a DSSP file and prints the sequence and the secondary structure of
18  * each chain in FASTA format.
19  *
20  * @see ex_dssp_to_ss2.cc converts DSSP to SS2 format
21  *
22  * CATEGORIES: core/data/io/DsspData
23  * KEYWORDS: DSSP; FASTA output; Structure; secondary structure; Format conversion
24 */
25
26 int main(const int argc, const char* argv[]) {
27
28     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
29     // missing program parameter
30
31     core::data::io::DsspData dssp(argv[1], true); // --- read a DSSP file - the first
32     // command line argument of the program
33     for (const auto & ss2 : dssp.sequences()) // --- for each protein sequence found in
34     // the DSSP data ...
35         std::cout << core::data::io::create_fasta_string(*ss2, 80) << "\n" // --- print
36         // the sequence as FASTA
37         << core::data::io::create_fasta_secondary_string(*ss2, 80) << "\n"; // --- print
38         // the secondary structure as FASTA
39 }

```



ex_FastaMatchProtocol

ex_FastaMatchProtocol finds similar substrings between two amino acid sequences. FastaMatchProtocol implements FAST algorithm to detect similar subsequences. This example just prints the list of FAST matches found between any two sequences from the input set.

USAGE:

```
ex_FastaMatchProtocol input.fasta [n_threads]
```

EXAMPLES:

```
ex_FastaMatchProtocol small1500_95identical.fasta 4
```

REFERENCE: Smith, Temple F., and Michael S. Waterman. "Identification of common molecular subsequences." PNAS 85 (1988): 2444-8 doi:10.1073/pnas.85.8.2444

Keywords:

- *FASTA input*
- *sequence alignment*
- *statistics*

Categories:

- core/protocols/PairwiseSequenceIdentityProtocol.hh

Input files:

- unique100.fasta

Output files:

- stdout.out

Program source:

```

1 #include <core/index.hh>
2 #include <utils/exit.hh>
3 #include <core/data/io/fasta_io.hh>
4 #include <core/protocols/PairwiseSequenceIdentityProtocol.hh>
5 #include <core/protocols/FastaMatchProtocol.hh>
6
7 std::string program_info = R"(
8
9 ex_FastaMatchProtocol finds similar substrings between two amino acid sequences.
10
11 FastaMatchProtocol implements FAST algorithm to detect similar subsequences. This
12 ↵example just prints the list
of FAST matches found between any two sequences from the input set.
)
```

(continues on next page)

(continued from previous page)

```

13
14 USAGE:
15   ex_FastaMatchProtocol input.fasta [n_threads]
16
17 EXAMPLES:
18   ex_FastaMatchProtocol small1500_95identical.fasta 4
19
20 REFERENCE:
21   Smith, Temple F., and Michael S. Waterman. "Identification of common molecular_
22   ↪ subsequences."
23   PNAS 85 (1988): 2444-8 doi:10.1073/pnas.85.8.2444
24 )
25
26 /** @brief Uses FastaMatchProtocol protocol to find similar substrings between two_
27   ↪ amino acid sequences
28 *
29 * CATEGORIES: core/protocols/PairwiseSequenceIdentityProtocol.hh
30 * KEYWORDS: FASTA input; sequence alignment; statistics
31 */
32 int main(const int argc, const char* argv[]) {
33
34   if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
35   ↪ missing program parameter
36
37   using namespace core::data::sequence;
38   using namespace core::protocols;
39   using namespace core::alignment;
40
41   utils::Logger logs("ex_FastaMatchProtocol");
42   core::index2 n_threads = (argc > 2) ? atoi(argv[2]) : 4;
43
44   bool if_store_diagonals = false;
45   logs << utils::LogLevel::INFO << "number of threads used : " << n_threads << "\n";
46
47   core::protocols::FastaMatchProtocol protocol;
48   protocol.minimum_diagonal_coverage(0.9).shortest_match_recorded(20).minimum_
49   ↪ identity(0.9).longest_gap(8);
50   protocol.batch_size(10000).n_threads(n_threads).keep_alignments(if_store_diagonals).
51   ↪ printed_seqname_length(5);
52
53   std::vector<Sequence_SP> input_sequences;
54   core::data::io::read_fasta_file(argv[1], input_sequences);
55   for(Sequence_SP si: input_sequences) protocol.add_input_sequence(si);
56
57   auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
58   protocol.run();
59   auto end = std::chrono::high_resolution_clock::now();
60   std::chrono::duration<double> time_span = std::chrono::duration_cast
61   ↪ <std::chrono::duration<double>>(end - start);
62   logs << utils::LogLevel::INFO << (size_t) protocol.n_jobs_completed()
63   << " FASTA matches calculated within " << time_span.count() << " [s]\n";
64
65   if(if_store_diagonals) {
66     std::cout << "Diagonals:\n";
67     protocol.print_header(std::cout);
68     protocol.print_diagonals(std::cout);

```

(continues on next page)

(continued from previous page)

```
64
65     std::cout << "Hits:\n";
66     std::vector<core::index4> hits;
67     for(core::index4 i_seq=0;i_seq< input_sequences.size();++i_seq) {
68         if (protocol.matches(i_seq, hits) > 0) {
69             std::cout << i_seq << " :";
70             for (core::index4 j:hits) std::cout << " " << j;
71             std::cout << "\n";
72         }
73     }
74 }
```



ex_GraphWithData

A unit test for SimpleGraph and GraphWithData classes

Keywords:

- *algorithms*
- *data structures*

Categories:

- core/algorithms/GraphWithData; core/algorithms/SimpleGraph

Program source:

```

1 #include <memory>
2 #include <iostream>
3 #include <iomanip>
4
5 #include <core/algorithms/GraphWithData.hh>
6 #include <core/algorithms/SimpleGraph.hh>
7
8 /** @brief A unit test for SimpleGraph and GraphWithData classes
9 *
10 * This program creates small graph data structures and test their methods
11 *
12 * CATEGORIES: core/algorithms/GraphWithData; core/algorithms/SimpleGraph
13 * KEYWORDS: algorithms; data structures
14 * IMG_ALT: Example tree node
15 */
16 int main(const int argc, const char* argv[]) {
17
18     using namespace core::algorithms;
19
20     GraphWithData<SimpleGraph,int,std::string> g;
21     g.add_vertex(0);
22     g.add_vertex(1);
23     g.add_vertex(2);
24     g.add_vertex(3);
25     g.add_edge(0,2,"0-2");
26     g.add_edge(1,2,"1-2");
27     g.add_edge(3,2,"3-2");
28     g.add_edge(core::index4(3),core::index4(0),"0-3");
29
30     std::cout << "# adjacency matrix\n";
31     g.print_adjacency_matrix(std::cout);
32     std::cout << "# are 0 and 3 connected?\n" << std::boolalpha << g.are_connected(0,3) << "\n";
33
34     g.remove_edge(0,3);
35     std::cout << "# are 0 and 3 still connected?\n" << std::boolalpha << g.are_
36     << connected(0,3)<<"\n";
37     std::cout << "# adjacency matrix\n";

```

(continues on next page)

(continued from previous page)

```
37 g.print_adjacency_matrix(std::cout);  
38  
39 std::cout << "# neighbors of 3\n";  
40 for(auto iter = g.begin(3);iter!=g.end(3);++iter)  
41     std::cout << *iter<<" ";  
42     std::cout << "\n";  
43  
44 return 0;  
45 }
```



ex_HierarchicalClustering

Example showing how to use hierarchical clustering method. The program uses Single Link method to cluster letters. Once clustering is done, it prints the clustering tree.

USAGE:

```
./ex_HierarchicalClustering
```

Keywords:

- *clustering*
- *hierarchical clustering*

Categories:

- core::calc::clustering::HierarchicalClustering

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <sstream>
3 #include <vector>
4 #include <numeric> // for std::accumulate
5
6 #include <core/algorithms/trees/algorithms.hh>
7 #include <core/calc/clustering/DistanceByValues.hh>
8 #include <core/calc/clustering/HierarchicalCluster.hh>
9 #include <core/calc/clustering/HierarchicalClustering.hh>
10 #include <utils/options/OptionParser.hh>
11
12 std::string program_info = R"(
13
14 Example showing how to use hierarchical clustering method. The program uses Single_
15 ↴Link method to cluster letters.
16 Once clustering is done, it prints the clustering tree.
17
18 USAGE:
19 . ./ex_HierarchicalClustering
20 )
21
22 using namespace core::calc::clustering;
23
24 static utils::Logger l("ex_HierarchicalClustering");
25
26 /// Data points to be clustered
27 std::vector<std::string> points = {"A", "B", "C", "E", "G", "L", "M", "Q", "R", "T",
28 ↴"X", "Y", "Z"};
```

(continues on next page)

(continued from previous page)

```

28
29 /// Distance function defined for the data above; here the alphabetic distance is used
30 DistanceByValues<float> calc_distance_matrix(std::vector<std::string> points) {
31
32     DistanceByValues<float> d(points, 99.0, 99.0);
33     for (size_t i = 1; i < d.n_data(); i++) {
34         for (size_t j = 0; j < i; j++) {
35             float v = std::sqrt((points[j][0] - points[i][0]) * (points[j][0] - points[i][0]));
36             d.set(i, j, v);
37             d.set(j, i, v);
38         }
39
40     return d;
41 }
42
43 /** @brief Example showing how to use hierarchical clustering method.
44 *
45 * CATEGORIES: core::calc::clustering::HierarchicalClustering
46 * KEYWORDS: clustering; hierarchical clustering
47 */
48 int main(int cnt, char *argv[]) {
49
50     if ((cnt > 1) && utils::options::call_for_help(argv[1]))
51         utils::exit_OK_with_message(program_info);
52
53     DistanceByValues<float> d = calc_distance_matrix(points);
54
55     HierarchicalClustering<float, std::string> hac(d.labels(), "");
56     hac.run_clustering(d, "");
57     std::vector<std::string> elements;
58     for (size_t i = 0; i < hac.count_steps(); i++) {
59         elements.clear();
60         std::shared_ptr<BinaryTreeNode<std::string>> c_node(std::static_pointer_cast<BinaryTreeNode<std::string>>(hac.clustering_step(i)));
61         core::algorithms::trees::collect_leaf_elements(c_node, elements);
62         std::string a = std::accumulate(elements.begin(), elements.end(), std::string(""));
63         a.erase(std::remove(a.begin(), a.end(), ' '), a.end());
64         std::sort(a.begin(), a.end());
65         std::cout << "Clustering step: "<<i<<" : ";
66         std::cout << a << "\n";
67     }
68
69     // --- write the clustering steps to a stream
70     std::ostringstream sso;
71     hac.write_merging_steps(sso);
72     std::cout <<sso.str();
73
74     // --- get medoid element for of the clusters created at distance d = 1.0
75     auto clusters = hac.get_clusters(1.0, 2);
76     for (core::index2 i = 0; i < 4; i++)
77         std::cout << medoid_by_average_distance<float, std::string, DistanceByValues<float>>(clusters[i], d).medoid << "\n";
78 }
```



ex_HierarchicalClustering1B

Example showing how to use hierarchical clustering method - 1 byte version. HierarchicalClustering1B is a specialized version of HierarchicalClustering which uses as least memory as possible. Distance values must be an integer in the range 0-255 (both inclusive); user is responsible for an appropriate and relevant conversion. The program uses Complete Link strategy. Once clustering is done, it prints medoids - elements located in centers of their clusters, corresponding to the given distance cutoff. The default cutoff value is set to 195. The clustering tree is printed on stderr.

USAGE:

```
ex_HierarchicalClustering1B input.txt
```

EXAMPLE:

```
ex_HierarchicalClustering1B fasta_distances
```

REFERENCE: Dominik Gront, Andrzej Koliński. “HCPM—program for hierarchical clustering of protein models.” Bioinformatics, 21 (2005):3179–80 doi:10.1093/bioinformatics/bti450

Keywords:

- *clustering*
- *hierarchical clustering*

Categories:

- core:::calc::clustering::HierarchicalClustering

Input files:

- fasta_distances

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <sstream>
3 #include <vector>
4 #include <numeric> // for std::accumulate
5
6 #include <core/algorithms/trees/algorithms.hh>
7 #include <core/algorithms/UnionFind.hh>
8 #include <core/calc/clustering/DistanceByValues1B.hh>
9 #include <core/calc/clustering/HierarchicalCluster.hh>
10 #include <core/calc/clustering/HierarchicalClustering1B.hh>
11 #include <core/BioShellEnvironment.hh>
12 #include <utils/LogManager.hh>
```

(continues on next page)

(continued from previous page)

```

13
14 using namespace core::calc::clustering;
15
16 static utils::Logger l("ex_HierarchicalClustering1B");
17
18 std::string program_info = R"
19
20 Example showing how to use hierarchical clustering method - 1 byte version.
21
22 HierarchicalClustering1B is a specialized version of HierarchicalClustering which
23 ↪uses as least memory
24 as possible. Distance values must be an integer in the range 0-255 (both inclusive);
25 ↪user is responsible
26 for an appropriate and relevant conversion.
27 The program uses Complete Link strategy. Once clustering is done, it prints medoids -
28 ↪elements located
29 in centers of their clusters, corresponding to the given distance cutoff. The default
30 ↪cutoff value is set to 195.
31 The clustering tree is printed on stderr.
32
33 USAGE:
34   ex_HierarchicalClustering1B input.txt
35
36 EXAMPLE:
37   ex_HierarchicalClustering1B fasta_distances
38
39 REFERENCE:
40 Dominik Gront, Andrzej Koliński. "HCPM-program for hierarchical clustering of protein
41 ↪models."
42 Bioinformatics, 21 (2005):3179-80 doi:10.1093/bioinformatics/bti450
43
44 )";
45
46 const int MAX = 10; // --- longest sequence id string
47
48 DistanceByValues1B read_distance_matrix(const std::string &distance_file, std::set<std::string> & labels) {
49
50     core::index1 val;
51     std::string line;
52     std::ifstream infile(distance_file);
53     char name_i[MAX], name_j[MAX];
54     while (std::getline(infile, line)) {
55         if(scanf_row(line, name_i, name_j, val)==3) {
56             labels.insert(name_i);
57             labels.insert(name_j);
58         }
59     }
60     infile.close();
61     infile.open(distance_file);
62
63     std::vector<std::string> v( labels.begin(), labels.end() );
64     core::algorithms::UnionFindSI4 uf(labels.size());
65     for (const std::string &s:v) uf.add_element(s);
66     DistanceByValues1B d(v);
67     while (std::getline(infile, line)) {
68         if (scanf_row(line, name_i, name_j, val) == 3) {
69             labels.insert(name_i);
70             labels.insert(name_j);
71             uf.union_set(name_i, name_j);
72         }
73     }
74     return uf;
75 }
```

(continues on next page)

(continued from previous page)

```

64     size_t i_index = d.at(name_i);
65     size_t j_index = d.at(name_j);
66     d.set(i_index, j_index, val);
67     d.set(j_index, i_index, val);
68     if(val==255)
69         uf.union_set(i_index, j_index);
70     }
71   }
72
73   std::cout << "# UnionFind groups larger than 2 (representative PDB-ID, group_"
74   << size) : \n";
75   const auto sets = uf.retrieve_sets();
76   for (const auto &set: sets) {
77     if (set.second.size() > 2)
78       std::cout << uf.element(set.first) << " : " << set.second.size() << "\n";
79   }
80   std::cout << sets.size() << "\n";
81
82   return d;
83 }
84
85 /**
86 * CATEGORIES: core::calc::clustering::HierarchicalClustering
87 * KEYWORDS: clustering; hierarchical clustering
88 */
89 int main(int cnt, char *argv[]) {
90
91
92   std::set<std::string> labels_set;
93   DistanceByValues1B d = read_distance_matrix(argv[1], labels_set);
94   std::cout << labels_set.size() << " items for clustering\n";
95
96   utils::LogManager::INFO();
97
98   HierarchicalClustering1B hac(d.labels(), "");
99   CompleteLink1B merge;
100
101   hac.run_clustering(d, merge);
102
103   // --- write the clustering steps to a stream
104   hac.write_merging_steps(std::cerr);
105 }
```



ex_Interpolate1D

Unit test which reads a file with two columns of data and calculates interpolated values for given number of steps.

USAGE:

```
./ex_Interpolate1D file.txt n_points
```

EXAMPLE:

```
./ex_Interpolate1D input.txt 100
```

Keywords:

- *interpolation*

Categories:

- core::calc::numeric::Interpolate1D

Input files:

- input.txt

Output files:

- stdout.out

Program source:

```

1 #include <cmath>
2 #include <iostream>
3 #include <vector>
4
5 #include <core/calc/numeric/interpolators.hh>
6 #include <core/calc/numeric/Interpolate1D.hh>
7 #include <core/data/io/DataTable.hh>
8 #include <utils/exit.hh>
9
10 using namespace core::calc::numeric;
11
12 std::string program_info = R"(
13
14 Unit test which reads a file with two columns of data and calculates interpolated_
15 ↴values for given number of steps.
16
17 USAGE:
18     ./ex_Interpolate1D file.txt n_points
19 EXAMPLE:
20     ./ex_Interpolate1D input.txt 100
21 )";

```

(continues on next page)

(continued from previous page)

```
22
23  /** @brief Reads a file with two columns of data and calculates interpolated values
24  *
25  * CATEGORIES: core::calc::numeric::Interpolate1D;
26  * KEYWORDS: interpolation
27  */
28 int main(int argc, char* argv[]) {
29
30     if(argc < 3) utils::exit_OK_with_message(program_info);
31
32     std::vector<double> vx; // --- vector of function arguments: X axis
33     std::vector<double> vy; // --- vector of function arguments: Y axis
34     core::data::io::DataTable in_data(argv[1]);
35     in_data.column(0, vx);
36     in_data.column(1, vy);
37
38     // --- Create the actual interpolator object
39     CatmullRomInterpolator<double> cri;
40     Interpolate1D<std::vector<double>, double, CatmullRomInterpolator<double>> ip(vx,
41     ↪vy, cri);
42     double step = (vx.back() - vx.front()) / atof(argv[2]);
43     for (double x = vx[0]; x <= vx.back(); x += step) {
44         std::cout << x << " " << ip(x) << "\n";
45     }
46 }
```



ex_InterpolatePeriodic1D

Simple example creates an interpolating polynomial for $\sin(x)$ function and computes maximal interpolation error (i.e. the maximum of the difference between the true function and its interpolator)

USAGE:

```
./ex_InterpolatePeriodic1D
```

Keywords:

- *interpolation*

Categories:

- core::calc::numeric::InterpolatePeriodic1D

Output files:

- stdout.out

Program source:

```
1 #include <cmath>
2 #include <iostream>
3 #include <vector>
4
5 #include <core/calc/numeric/interpolators.hh>
6 #include <core/calc/numeric/InterpolatePeriodic1D.hh>
7 #include <core/calc/structural/angles.hh>
8 #include <utils/options/OptionParser.hh>
9 #include <utils/exit.hh>
10
11 std::string program_info = R"(
12
13 Simple example creates an interpolating polynomial for  $\sin(x)$  function and computes
14 maximal interpolation error (i.e. the maximum of the difference between the true_
15 ↴function and its interpolator)
16
17 USAGE:
18 ./ex_InterpolatePeriodic1D
19 )";
20
21 using namespace core::calc::numeric;
22
23 /**
24 * @brief Simple test for interpolation of a periodic 1D function
25 *
26 * @CATEGORIES: core::calc::numeric::InterpolatePeriodic1D;
27 * @KEYWORDS: interpolation
28 */
29 int main(int cnt, char *argv[]) {
```

(continues on next page)

(continued from previous page)

```

30  if ((cnt > 1) && utils::options::call_for_help(argv[1]))
31      utils::exit_OK_with_message(program_info);
32
33  std::vector<float> x, y; // --- vectors for points used for interpolation
34  double step = M_PI / 50.0; // --- interpolation data step
35  for (double ix = 0.0; ix < 2 * M_PI; ix += step) { // --- generate data for_
36  ↪interpolation knots
37      x.push_back(ix);
38      y.push_back(sin(ix));
39
40  CatmullRomInterpolator<float> cri; // --- Interpolating engine
41  InterpolatePeriodic1D<std::vector<float>, float, CatmullRomInterpolator<float> >_
42  ↪i1d2(x, y, cri, 2*M_PI);
43  double max_error = 0.0; // --- used to keep track of the maximum interpolation error
44  double worst_x = 0.0;
45  // --- The function has been defined in the range \f$[0, \pi]\f$
46  // --- interpolation goes in the range \f$[-20\pi, 40\pi]\f$
47  for (float x = -20 * M_PI; x <= 40 * M_PI; x += 0.1 * step) {
48      double error = fabs(sin(x) - i1d2(x));
49      if (error > max_error) {
50          max_error = std::max(error, max_error);
51          worst_x = x;
52      }
53  }
54  std::cout << "Maximum interpolation error:" << max_error << " for x= " << worst_x <
55  ↪< "\n";
56 }
```



ex_InterpolatePeriodic2D

Simple example creates an interpolating polynomial for $\sin(x) * \cos(y)$ 2D function and computes maximal interpolation error (i.e. the maximum of the difference between the true function and its interpolator)

USAGE:

```
./ex_InterpolatePeriodic2D
```

Keywords:

- *interpolation*

Categories:

- core::calc::numeric::InterpolatePeriodic2D

Output files:

- stdout.out

Program source:

```

1 #include <cmath>
2 #include <iostream>
3 #include <memory>
4 #include <sstream>
5 #include <vector>
6
7 #include <core/calc/numeric/interpolators.hh>
8 #include <core/calc/numeric/InterpolatePeriodic2D.hh>
9
10 #include <core/calc/structural/angles.hh>
11 #include <utils/options/OptionParser.hh>
12
13 std::string program_info = R"(
14
15 Simple example creates an interpolating polynomial for sin(x) * cos(y) 2D function_
16 ↵and computes
17 maximal interpolation error (i.e. the maximum of the difference between the true_
18 ↵function and its interpolator)
19
20
21 )";
22
23 using namespace core::calc::numeric;
24
25 /** @brief Simple test for interpolation of a periodic 2D function
26 *
27 * CATEGORIES: core::calc::numeric::InterpolatePeriodic2D;
28 * KEYWORDS: interpolation

```

(continues on next page)

(continued from previous page)

```

29 */
30 int main(int cnt, char* argv[]) {
31
32 if ((cnt > 1) && utils::options::call_for_help(argv[1]))
33     utils::exit_OK_with_message(program_info);
34
35 double inter_step = core::calc::structural::to_radians(5.0); // --- interpolation_
36 → step : 5 degrees
37 const double EPS = 0.0001;
38 std::vector<double> vx; // --- vector of function arguments: X axis
39 std::vector<double> vy; // --- vector of function arguments: Y axis
40 for (double x = -M_PI; x < M_PI-EPS; x += inter_step) {
41     vx.push_back(x);
42     vy.push_back(x); // --- we use the same values both for X and Y but in general_
43 → they may differ
44 }
45 core::index2 nx = vx.size(); // the number of grid points
46
47 // --- Prepare data to be interpolated
48 std::shared_ptr<core::data::basic::Array2D<double>> data_periodic = std::make_shared<
49 → core::data::basic::Array2D<double>>(nx,nx);
50 for (size_t ix = 0; ix < vx.size(); ++ix)
51     for (size_t iy = 0; iy < vy.size(); ++iy) data_periodic->set(ix, iy, sin(vx[ix])_
52 → * cos(vy[iy]));
53
54 // --- Create the actual interpolator object
55 CatmullRomInterpolator<double> cri;
56 InterpolatePeriodic2D<double, CatmullRomInterpolator<double>> ip(-M_PI,inter_step,
57 → nx,-M_PI,inter_step,nx, data_periodic, cri);
58 double max_error = 0.0;
59 for (double x = -5; x <= 4.0; x += inter_step / 3.0) {
60     for (double y = -5.0; y <= 4.0; y += inter_step / 3.0) {
61         double v = sin(x) * cos(y); // --- calculate the true value of the interpolated_
62 → function ...
63         double vi = ip(x, y); // --- also the interpolated value
64         double err = fabs(v - vi);
65         max_error = std::max(err, max_error);
66     }
67 }
68 std::cout << "Maximum interpolation error: " << max_error << "\n";
69 }
```



ex_Ising2D

Simple but fully functional Ising simulating program.

Keywords:

- *Mover*
- Simulated Annealing
- Ising2D

Categories:

- simulations::systems::ising::Ising2D

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <fstream>
3 #include <simulations/systems/ising/Ising2D.hh>
4 #include <simulations/movers/ising/SingleFlip2D.hh>
5 #include <simulations/movers/ising/WolffMove2D.hh>
6
7 #include <simulations/observers/ObserveEvaluators.hh>
8 #include <simulations/movers/MoversSetSweep.hh>
9 #include <simulations/sampling/SimulatedAnnealing.hh>
10
11
12 /* @brief Simple but fully functional Ising simulating program.
13 *
14 * Runs simulated annealing simulations for 100x100
15 *
16 * CATEGORIES: simulations::systems::ising::Ising2D
17 * KEYWORDS: Mover;Simulated Annealing;Ising2D
18 */
19 using namespace simulations::systems::ising;
20 using namespace simulations::movers::ising;
21
22 int main(const int argc, const char *argv[]) {
23     /* Simulation controlling variables */
24     int n_cols = 10, n_rows = 10; // size of system
25
26     /* Other settings necessary for the simulation */
27     int seed = 12345; // seed for rng
28     core::calc::statistics::Random::seed(seed);
29
30     /* Initializing the system */
31     std::shared_ptr<Ising2D<core::index1, core::index2>> system = std::make_shared
32     <Ising2D<core::index1, core::index2>>(n_rows, n_cols);
```

(continues on next page)

(continued from previous page)

```

32     system->initialize();      // Populate system with random spins
33
34     simulations::movers::MoversSet_SP movers = std::make_shared<
35         simulations::movers::MoversSetSweep>();
36     movers->add_mover( std::make_shared<SingleFlip2D<core::index1,core::index2>>
37         (*system),system->count_spins());
38     movers->add_mover( std::make_shared<WolffMove2D<core::index1,core::index2>>
39         (*system),system->count_spins()*0.2);
40
41
42     std::vector<float> temperatures = {5,4,3,2.5,2.25,2,1.75,1.5,1};
43     simulations::sampling::SimulatedAnnealing sa(movers,temperatures);
44     sa.cycles(100,100);
45
46     simulations::observers::ObserveEvaluators_SP observations = std::make_shared<
47         simulations::observers::ObserveEvaluators>( "" );
48     observations->add_evaluator(system);
49     sa.outer_cycle_observer(observations);
50
51     sa.run();
52     observations->finalize();
53 }
```



ex_IterateIJ

Unit test which shows how to use IterateIJ class, which is an iterator to a 2D container, e.g. a 2D array.

USAGE:

```
./ex_IterateIJ
```

Keywords:

- *data structures*
- *algorithms*

Categories:

- core:::algorithms::IterateIJ

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/algorithms/IterateIJ.hh>
4 #include <utils/options/OptionParser.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Unit test which shows how to use IterateIJ class, which is an iterator to a 2D_
10    ↪container, e.g. a 2D array.
11
12 USAGE:
13 ./ex_IterateIJ
14 )";
15
16 /**
17 * @brief A simple example shows how to use IterateIJ
18 *
19 * CATEGORIES: core:::algorithms::IterateIJ
20 * KEYWORDS: data structures; algorithms
21 */
22 int main(const int argc, const char *argv[]) {
23
24     if ((argc > 1) && utils::options::call_for_help(argv[1]))
25         utils::exit_OK_with_message(program_info);
26
27     // ----- Here we declare an iterator that generates indexes for a square 5x5_
28    ↪2D array

```

(continues on next page)

(continued from previous page)

```
28 core:::algorithms::IterateIJ ij1(5, false);
29     for (auto ij:ij1) std::cout << ij.first << " " << ij.second << "\n";
30     std::cout <<"\n";
31
32 // ----- Here we declare an iterator that generates indexes for the UR
33 // triangle of a 5x5 2D array
34 core:::algorithms::IterateIJ ij2(5, true);
35     for (auto ij:ij2) std::cout << ij.first << " " << ij.second << "\n";
36     std::cout <<"\n";
37
38 // ----- Now only selected rows of a square 5x5 2D array
39 core:::algorithms::IterateIJ ij3(5, false);
40 ij3.add_selected_row(1).add_selected_row(2);
41     for (auto ij:ij3) std::cout << ij.first << " " << ij.second << "\n";
42     std::cout <<"\n";
43
44 // ----- Now only selected rows and only UR triangle of a 5x5 2D array
45 core:::algorithms::IterateIJ ij4(5, true);
46 ij4.add_selected_row(2).add_selected_row(3);
47     for (auto ij:ij4) std::cout << ij.first << " " << ij.second << "\n";
48 }
```



ex_JsonNode

Demonstrates how to handle JSON data.

USAGE:

```
./ex_JsonNode
```

Keywords:

- JSON

Categories:

- core::data::io::JsonNode

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <core/data/io/json_io.hh>
3 #include <utils/options/OptionParser.hh>
4 #include <utils/exit.hh>
5
6 std::string program_info = R"(
7
8 Demonstrates how to handle JSON data.
9
10 USAGE:
11     ./ex_JsonNode
12
13 )";
14
15 /** @brief Demo for handling JSON data
16 *
17 * The example tests whether a JSON data is parsed and printed correctly.
18 *
19 * CATEGORIES: core::data::io::JsonNode
20 * KEYWORDS:   JSON
21 */
22 int main(const int argc, const char* argv[]) {
23
24     if ((argc > 1) && utils::options::call_for_help(argv[1]))
25         utils::exit_OK_with_message(program_info);
26
27     using namespace core::data::io;
28
29     // Create JsonNode object using constructors
30     JsonNode_SP n0 = std::make_shared<JsonNode>(std::make_shared<JsonValue>("options"),
31                                                 -1);
32 }
```

(continues on next page)

(continued from previous page)

```

31     n0->add_branch(std::make_shared<JsonNode>(std::make_shared<JsonValue>("size","56"),
32     ↪2));
33     n0->add_branch(std::make_shared<JsonNode>(std::make_shared<JsonValue>("color","red",
34     ↪),3));
35     std::cout << n0; // This is how to print a full JSON tree
36
37     std::string json = R"({\"options\" : {"size" : 56 , "color" : {"fg" : "red", "feature
38     ↪" : "none", "bg" : {"r":25,"g":124,"b":19} }, "do" : "all"})"
39     ;
40     JsonNode_SP root = read_json(json); // Here json string is parsed
41     std::cout << root; // and here send back to a stream
42
43     // Here a JSONArray instance is created; an empty array at first ...
44     std::shared_ptr<JSONArray> jv1 = std::make_shared<JSONArray>("res-1");
45     // and now that empty array is fileld with data; <code>"011"</code> string means,
46     ↪that the first token (i.e. 37) does not
47     // require quotes (hence logical 0) and the two latter tokens do (logical 1)
48     jv1->values("011",37,"ALA",'A');
49     JsonNode_SP jv2 = create_json_node("res-2","011",38,"PHE",'A');
50     std::cout << (*jv1) << " " << (jv2) << "\n";
51
52     // Create JsonNode object using helper methods, provided by <code>json_io.hh</code>
53     JsonNode_SP another_root = create_json_node();
54     another_root->add_branch( create_json_node(jv1) );
55     another_root->add_branch( jv2 );
56     std::cout << another_root;
57 }
```



ex_KDE_1D

Reads one column of observations and calculates Kernel Density Estimator (KDE) with given bandwidth value for the data. If optional parameters min and max are given, it defines the evaluation range. The last optional argument is the word ‘periodic’ to treat the estimated distribution as periodic.

USAGE:

```
ex_KDE_1D normal.txt 0.25 [min max periodic]
```

REFERENCE: Davis, Richard A., Keh-Shin Lii, Dimitris N. Politis. “Remarks on some nonparametric estimates of a density function.” Selected Works of Murray Rosenblatt. Springer, New York, NY, 2011. 95-100. doi:10.1214/aoms/1177728190.

Parzen, Emanuel. “On estimation of a probability density function and mode.” The annals of mathematical statistics 33.3 (1962): 1065-1076.

Keywords:

- *statistics*
- *estimation*
- *data table*

Categories:

- core/calc/statistics/KDE_1D

Input files:

- normal.txt
- THR_chi1.dat

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <iomanip>
3 #include <core/data/io/Pdb.hh>
4
5 #include <core/data/io/DataTable.hh>
6 #include <core/calc/statistics/KDE_1D.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 Reads one column of observations and calculates Kernel Density Estimator (KDE) with
12 ↪given bandwidth value for the data.

```

(continues on next page)

(continued from previous page)

```

12 If optional parameters min and max are given, it defines the evaluation range. The
13 ↪last optional argument
14 is the word 'periodic' to treat the estimated distribution as periodic.
15
16 USAGE:
17   ex_KDE_1D normal.txt 0.25 [min max periodic]
18
19 REFERENCE:
20 Davis, Richard A., Keh-Shin Lii, Dimitris N. Politis. "Remarks on some nonparametric
21 ↪estimates of a density function."
22 Selected Works of Murray Rosenblatt. Springer, New York, NY, 2011. 95-100. doi:10.
23 ↪1214/aoms/1177728190.
24
25 )";
26
27 /** @brief Reads one column of observations and calculates Kernel Density Estimator
28 ↪(KDE) for the data
29 *
30 * CATEGORIES: core/calc/statistics/KDE_1D
31 * KEYWORDS: statistics; estimation; data table
32 */
33 int main(const int argc, const char* argv[]) {
34
35   if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about
36 ↪missing program parameter
37
38   core::data::io::DataTable input_data(argv[1]); // --- Here we read a text file with
39 ↪data values in columns
40   std::vector<double> coll; // --- empty vector to hold the data from a file
41
42   bool is_periodic = (argc > 2 && argv[argc-1][0] == 'p');
43   // --- Here we read the input file and create a KDE estimator for the data in the
44 ↪first column
45   core::calc::statistics::KDE_Kernel kernel_type = core::calc::statistics::normal_
46 ↪kernel;
47   core::calc::statistics::KDE_1D kde(input_data.column(0, coll), atof(argv[2]),
48 ↪kernel_type, is_periodic);
49
50   // --- find max and min value in the file; tabulate the data in 100 steps
51   double min = (argc < 5) ? *std::min_element(coll.begin(), coll.end()) :_
52 ↪atof(argv[3]);
53   double max = (argc < 5) ? *std::max_element(coll.begin(), coll.end()) :_
54 ↪atof(argv[4]);
55   double step = (max - min) / 300.0;
56   for (double x = min; x <= max; x += step) std::cout << utils::string_format("%8.3f
57 ↪%8.3f\n", x, kde(x, is_periodic));
58 }
```



ex_LBFGS

Unit test which shows how to use Broyden–Fletcher–Goldfarb–Shanno (BFGS) function minimizer.

USAGE:

```
./ex_LBFGS
```

REFERENCE: Fletcher, Roger. Practical methods of optimization. John Wiley & Sons, 2013.

Keywords:

- *numerical methods*

Categories:

- core/calc/numeric/Bfgs

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/index.hh>
4 #include <core/calc/numeric/LBFGS.hh>
5 #include <utils/options/OptionParser.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Unit test which shows how to use Broyden-Fletcher-Goldfarb-Shanno (BFGS) function_
11 ↵minimizer.
12
13 USAGE:
14 ./ex_LBFGS
15
16 REFERENCE:
17 Fletcher, Roger. Practical methods of optimization. John Wiley & Sons, 2013.
18 )";
19
20 using namespace core::calc::numeric;
21
22 class TestFunction : public DerivableFunction<double> {
23 public:
24
25     double a, b;
26
27     TestFunction() : a(1.0), b(30.0) {}
28 }
```

(continues on next page)

(continued from previous page)

```

29     virtual double operator() (const std::vector<double> &x) {
30         return (a - x[0]) * (a - x[0]) + b * (x[1] - x[0] * x[0]) * (x[1] - x[0] * x[0]);
31     }
32
33     virtual double operator() (const std::vector<double> &x, std::vector<double> &
34     ↪gradient) {
35
36         float t1 = a - x[0];
37         float t2 = b * (x[1] - x[0] * x[0]);
38         gradient[1] = 2 * b * t2;
39         gradient[0] = -2.0 * (x[0] * gradient[1] + t1);
40
41         return t1 * t1 + t2 * t2;
42     }
43
44     core::index2 dim() const { return 2; }
45 };
46
47 /**
48 *
49 * CATEGORIES: core/calc/numeric/Bfgs
50 * KEYWORDS: numerical methods
51 */
52 int main(const int argc, const char *argv[]) {
53
54     TestFunction f;
55     LBFGS<double> minimizer(2);
56
57     std::vector<double> x{0.0, 0.0};
58     double val;
59     core::index2 nit = minimizer.minimize(f, x, val);
60
61     std::cout << "Minimum value " << val << " found at [";
62     for (const double v:x) std::cout << v << ' ';
63     std::cout << "] after " << nit << " iterations\n";
64     return 0;
65 }
```



ex_MC_Ar

The program runs an isothermal MC simulation of argon gas. By default it starts from a regular lattice conformation unless an input file (PDB) with initial conformation is provided

USAGE:

```
ex_MC_Ar n_atoms density temperature small_cycles big_cycles [max_jump]
ex_MC_Ar starting.pdb density temperature small_cycles big_cycles [max_jump]
```

Keywords:

- no_keywords

Categories:

- no_categories

Output files:

- stdout.out
- movers.dat
- final.pdb

Program source:

```

1 #include <cstdio>
2 #include <ctime>
3 #include <iostream>
4 #include <thread>
5
6 #include <core/data/basic/Vec3I.hh>
7 #include <core/BioShellVersion.hh>
8
9
10 #include <utils/string_utils.hh>
11 #include <utils/options/Option.hh>
12 #include <utils/options/OptionParser.hh>
13 #include <utils/options/output_options.hh>
14 #include <utils/options/sampling_options.hh>
15
16 #include <simulations/systems/CartesianAtoms.hh>
17 #include <simulations/systems/BuildFluidSystem.hh>
18 #include <simulations/systems/SingleAtomType.hh>
19 #include <simulations/movers/TranslateAtom.hh>
20 #include <simulations/forcefields/cartesian/LJEnergySWHomogenic.hh>
21 #include <simulations/sampling/IsothermalMC.hh>
22 #include <simulations/observers/ObserveEvaluators.hh>
23 #include <simulations/observers/cartesian/PdbObserver.hh>
24 #include <simulations/evaluators/CallEvaluator.hh>
25 #include <simulations/observers/ObserveMoversAcceptance.hh>
```

(continues on next page)

(continued from previous page)

```

26 #include <simulations/observers/TriggerEveryN.hh>
27 #include <simulations/observers/AdjustMoversAcceptance.hh>
28 #include <simulations/observers/cartesian/SimplePdbFormatter.hh>
29
30 using namespace core::data::basic;
31
32 utils::Logger logs("ex_MC_Ar");
33
34 std::string program_info = R"
35
36 The program runs an isothermal MC simulation of argon gas. By default it starts from a_
37 ↪regular lattice conformation
38 unless an input file (PDB) with initial conformation is provided
39 USAGE:
40   ex_MC_Ar n_atoms density temperature small_cycles big_cycles [max_jump]
41   ex_MC_Ar starting.pdb density temperature small_cycles big_cycles [max_jump]
42 )";
43
44 const double EPSILON = 1.654E-21;           // [J] per molecule
45 const double EPSILON_BY_K = EPSILON / 1.381E-23;      // = 119.6 in Kelvins
46 const double SIGMA = 3.4;                  // in Angstroms
47
48 /** @brief Isothermal Monte Carlo simulation of argon gas.
49 */
50
51 int main(const int argc, const char* argv[]) {
52
53     using core::data::basic::Vec3I;
54     using namespace simulations::systems;
55     using namespace simulations::movers; // for MoversSet
56     using namespace simulations::observers::cartesian; // for all observers
57
58     logs << utils::LogLevel::INFO << "BioShell version:\n" << core::BioShellVersion().to_
59     ↪string() << "\n";
60     core::index4 n_outer_cycles = 1000;
61     core::index4 n_inner_cycles = 100;
62     double density = 0.5;      // density of the system controls how many atoms will be_
63     ↪contained in the box
64     double temperature = 97; // in Kelvins
65     core::index4 n_atoms = 64;
66     double max_jump = 0.5;           // Random move range (in Angstroms)
67
68     core::data::structural::Structure_SP argon_structure = nullptr;
69     core::data::structural::PdbAtom_SP ar_atom = std::make_shared<core::data::structural::PdbAtom>(1, " AR");
70     if (argc < 6) std::cerr << program_info;
71     else {
72         if (utils::is_integer(argv[1])) n_atoms = atoi(argv[1]);
73         else { // --- read an input file if given
74             core::data::io::Pdb reader(argv[1]);
75             argon_structure = reader.create_structure(0);
76             n_atoms = argon_structure->count_atoms();
77         }
78         density = atof(argv[2]);
79         temperature = atof(argv[3]);
80         n_inner_cycles = atoi(argv[4]);
81     }
82 }
```

(continues on next page)

(continued from previous page)

```

n_outer_cycles = atoi(argv[5]);
if (argc == 7) max_jump = atof(argv[6]);
}

double ar_volume = 4.0 / 3.0 * M_PI * SIGMA * SIGMA * SIGMA * n_atoms;
double box_len = pow(ar_volume / density, 0.333333333333333);

// --- Initialize periodic boundary conditions
core::data::basic::Vec3I::set_box_len(box_len);
logs << utils::LogLevel::INFO << "box width for " << int(n_atoms) << " atoms set to "
<> " " << box_len << "\n";

// --- Create the system and distribute atoms in the box
AtomTypingInterface_SP ar_type = std::make_shared<SingleAtomType>(" AR");
CartesianAtoms ar(ar_type, n_atoms);
core::calc::statistics::Random::seed(1234);
if(argon_structure != nullptr) { // --- read coordinates from a PDB file if provided
    set_conformation(argon_structure->first_const_atom(), argon_structure->last_const_atom(), ar);
} else { // --- otherwise generate coordinates
    const auto grid = std::make_shared<SimpleCubicGrid>(box_len, n_atoms);
    BuildFluidSystem::generate(ar, *ar_atom, grid);
}
CartesianAtoms ar_backup(ar); // --- make a backup system

// --- Create energy function - just LJ potential
simulations::forcefields::cartesian::LJEnergySWHomogenic lj_energy(ar, SIGMA, _EPSILON_BY_K);

// --- Create a mover, which is a random perturbation of an atom in this case, and place it in a movers' set
std::shared_ptr<TranslateAtom> translate = std::make_shared<TranslateAtom>(ar, ar_backup, lj_energy);
translate->max_move_range_allowed(1.5);
MoversSet_SP movers = std::make_shared<simulations::movers::MoversSetSweep>();
movers->add_mover(translate, n_atoms);
translate->max_move_range(max_jump); // --- set the maximum distance a single atom can be moved by a single MC perturbation

// --- create an isothermal Monte Carlo sampler
simulations::sampling::IsothermalMC mc(movers, temperature);

// ----- Create an observer which calls energy calculation and prints it on the screen
std::shared_ptr<simulations::observers::ObserveEvaluators> obs = std::make_shared<simulations::observers::ObserveEvaluators>("");
std::function<double(void)> recent_energy = [&lj_energy, &ar] () { return lj_energy.energy(ar); };
std::function<double(void)> nbl_updates = [&lj_energy] () { return lj_energy.non_bonded_neighbors().updates_ratio(); };
obs->add_evaluator(
    std::make_shared<simulations::evaluators::CallEvaluator<std::function<double(void)>>>(recent_energy, "energy", 8));
obs->add_evaluator(
    std::make_shared<simulations::evaluators::CallEvaluator<std::function<double(void)>>>(nbl_updates, "updates_ratio", 6));

```

(continues on next page)

(continued from previous page)

```
123 // std::shared_ptr<simulations::observers::ObserveMoversAcceptance> observe_moves
124 //     = std::make_shared<simulations::observers::ObserveMoversAcceptance>(*movers,
125 //       "movers.dat");
126
127     std::shared_ptr<simulations::observers::AdjustMoversAcceptance> observe_moves
128     = std::make_shared<simulations::observers::AdjustMoversAcceptance>(*movers,
129     "movers.dat", 0.4);
130     observe_moves->observe_header();
131
132     std::shared_ptr<AbstractPdbFormatter> fmt = std::make_shared<SimplePdbFormatter>("_
133     AR ", "AR ", "AR");
134     auto observe_trajectory = std::make_shared
135     <simulations::observers::cartesian::PdbObserver>(ar, fmt, "ar_tra.pdb");
136     observe_trajectory->trigger(std::make_shared<simulations::observers::TriggerEveryN>
137     (10));
138     mc.outer_cycle_observer(observe_trajectory); // --- commented out to save disk space
139     mc.outer_cycle_observer(observe_moves);
140     mc.outer_cycle_observer(obs);
141     mc.cycles(n_inner_cycles,n_outer_cycles,1);
142
143     mc.run();
144
145     simulations::observers::cartesian::PdbObserver final(ar, fmt, "final.pdb");
146     final.observe();
147     logs << utils::LogLevel::INFO << "Final energy " << lj_energy.energy(ar) << "\n";
148 }
```



ex_MMAtomTyping

Reads a PDB file and assigns MM atom typing for every atom of the given protein according to the given force field parametrisation file. If no .par file was given, AMBER03 force field will be used.

USAGE:

```
./ex_MMAtomTyping [param-file] input.pdb
```

EXAMPLE:

```
./ex_MMAtomTyping 2gb1.pdb
./ex_MMAtomTyping amber03_atoms.par 2gb1.pdb
```

Keywords:

- *PDB input*
- atom typing
- force field

Categories:

- simulations/forcefields/mm/MMAtomTyping

Input files:

- 2gb1.pdb
- amber03_atoms.par

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <vector>
3
4 #include <core/data/io/Pdb.hh>
5 #include <core/BioShellEnvironment.hh>
6 #include <simulations/forcefields/mm/MMForceField.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 Reads a PDB file and assigns MM atom typing for every atom of the given protein,
12 ↵according to the given force field
13 parametrisation file. If no .par file was given, AMBER03 force field will be used.
14
15 USAGE:
```

(continues on next page)

(continued from previous page)

```

15   ./ex_MMAtomTyping [param-file] input.pdb
16 EXAMPLE:
17   ./ex_MMAtomTyping 2gb1.pdb
18   ./ex_MMAtomTyping amber03_atoms.par 2gb1.pdb
19
20 ) ";
21
22 /** @brief Assigns MM atom typing for every atom of the given protein according to
23  * the given force field parametrisation file
24  *
25  * CATEGORIES: simulations/forcefields/mm/MMAtomTyping
26  * KEYWORDS: PDB input; atom typing; force field
27  */
28
29 int main(const int argc, const char* argv[]) {
30
31     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
32     //missing program parameter
33
34     using namespace simulations::forcefields::mm;
35     MMForceField mmff("AMBER03");
36     const MMAtomTyping & atom_typing = *(mmff.mm_atom_typing());
37     core::data::io::Pdb pdb((argc == 2) ? argv[1] : argv[2]);
38     core::data::structural::Structure_SP s = pdb.create_structure(0);
39     for (auto chain: *s) {
40         for (core::index2 ires = 0; ires < chain->size(); ++ires) {
41             for (auto atom : *(chain)[ires]) {
42                 core::index2 t = atom_typing.atom_type(*atom);
43                 std::cout << (*atom).owner() << " " << (*atom).atom_name() << " : " << t <<
44                 " " << atom_typing.atom_internal_name(t) << "\n";
45             }
46         }
47     }
48 }
```



ex_MM_BondEnergy**Keywords:**

- no_keywords

Categories:

- no_categories

Input files:

- 2gb1.pdb
- amber03_ffbonded.itp
- amber03_atoms.par

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/data/basic/Vec3.hh>
4 #include <core/data/structural/Structure.hh>
5 #include <utils/LogManager.hh>
6
7 #include <simulations/forcefields/mm/MMBondEnergy.hh>
8 #include <simulations/forcefields/mm/MMBondedParameters.hh>
9 #include <simulations/forcefields/mm/MMForceField.hh>
10
11 int main(const int argc, const char *argv[]) {
12
13     using namespace simulations::forcefields::mm; // for all MM - related force field
14     // classes
15     using namespace core::data::structural; // for Structure, Residue, PdbAtom
16     using namespace core::data::basic; // for Vec3
17
18     utils::LogManager::get().set_level("FINE");
19
20     if (argc < 2) {
21         std::cerr << "USAGE:\n\t./ex_MM_BondEnergy atom_typing ff_bonded.par input.pdb\n\n";
22         exit(0);
23     }
24
25     // --- Create molecular mechanic bond energy object
26     MMForceField mmff("AMBER03");
27
28     MMBondedParameters bond_manager(argv[2], mmff.mm_atom_typing());

```

(continues on next page)

(continued from previous page)

```
28 // --- Read structure for which calculate bond energy
29 core::data::io::Pdb reader(argv[3]); // file name (PDB format, may be gzip-ped)
30 core::data::structural::Structure_SP strctr = reader.create_structure(0);
31 auto rc = CartesianChains(mmmff.mm_atom_typing(), *strctr);
32
33 // --- Create molecular mechanic bond energy object
34 simulations::forcefields::mm::MMBondEnergy bond_energy(rc,bond_manager);
35 const auto & bonds = bond_energy.get_bonds();
36     std::cout << "#    i      j      E(d0)      d\n";
37
38 // --- Calculate energy for each bond in the structure
39 double total = 0.0;
40 for (auto it = bonds.cbegin(); it != bonds.cend(); ++it) {
41     double en = bond_energy.calculate(*it);
42     total += en;
43     double dreal = (rc)[it->i].distance_to((rc)[it->j]);
44     std::cout << utils::string_format("%5d %5d %10.3f      %4.2f %4.2f\n", (*it).i,
45     (*it).j, en, (*it).d0,dreal);
46 }
47 // --- Calculate total bond energy by whole structure
48 double total_en = bond_energy.energy(rc);
49
50     std::cout << utils::string_format("%10.3f %10.3f\n", total, total_en);
51 }
```



ex_MMEnergy**Keywords:**

- no_keywords

Categories:

- no_categories

Input files:

- 2gb1.pdb
- amber03_ffnonbonded.itp
- amber03_ffbonded.itp
- amber03_atoms.par

Output files:

- stdout.out

Program source:

```
1 #include <string>
2 #include <chrono>
3
4 #include <core/data/basic/Vec3.hh>
5
6 #include <utils/string_utils.hh>
7 #include <utils/LogManager.hh>
8
9 #include <simulations/forcefields/mm/MMNonBonded.hh>
10 #include <simulations/forcefields/mm/MMBondType.hh>
11 #include <simulations/forcefields/mm/MMBondEnergy.hh>
12 #include <simulations/forcefields/mm/MMPlanarEnergy.hh>
13 #include <simulations/forcefields/mm/MMDihedralEnergy.hh>
14 #include <simulations/forcefields/mm/MMBondedParameters.hh>
15 #include <simulations/forcefields/mm/MMForceField.hh>
16
17 int main(const int argc, const char *argv[]) {
18
19     using namespace core::data::structural; // for Structure, Residue, PdbAtom
20     using namespace core::data::basic; // for Vec3
21     using namespace simulations::forcefields::mm; // for all MM - related force field
22     ↪ classes
23     using namespace simulations::systems; // for ResidueChain
24
25     utils::LogManager::get().set_level("INFO");
26
27     if (argc < 2) {
28         std::cerr << "USAGE:\n\t./ex_MMEnergy atoms.par ff_bonded.par 2gb1.pdb\n\n";
29     }
30 }
```

(continues on next page)

(continued from previous page)

```

28     exit(0);
29 }
30
31 // --- Read atom typing and bond parameters
32 MMForceField mmff("AMBER03");
33 simulations::forcefields::mm::MMBondedParameters ff(argv[2],mmff.mm_atom_
→typing());
34
35 // --- Read structure for which calculate bond energy
36 core::data::io::Pdb reader(argv[3]); // file name (PDB format, may be gzip-ped)
37 Structure_SP strctr = reader.create_structure(0);
38 auto rc = CartesianChains(mmff.mm_atom_typing(), *strctr);
39
40 size_t i = 0;
41 // for (auto atom_it = strctr->first_const_atom(); atom_it != strctr->last_const_
→atom(); ++atom_it) {
42 //   auto at = atom_typing->atom_type(**atom_it);
43 //   (*rc)[i].register_ = (1.0 + at.charge()) * 10000.0;
44 //   ++i;
45 // }
46
47 // --- Create molecular mechanic bond energy object
48 simulations::forcefields::mm::MMBondEnergy bond_energy(rc, ff);
49 // --- Create molecular mechanic planar energy objects
50 simulations::forcefields::mm::MMPlanarEnergy planar_energy(rc, ff, bond_energy);
51 // --- Create molecular mechanic dihedral energy objects
52 simulations::forcefields::mm::MMDihedralEnergy dihedral_energy(rc, ff, bond_energy);
53
54 // --- Create non-bonded energy; pass the reference to bond energy object so non-
→bonded energy can exclude bonds
55 MMNonBonded nb_energy(rc, bond_energy);
56 // nb_energy.get_excluded_pairs().print(std::cout);
57
58 auto start = std::chrono::high_resolution_clock::now();
59 std::cout << "#bond_energy planar_energy dihedral_energy nb_energy\n";
60 std::cout << utils::string_format(" %10.3f %10.3f %10.3f %10.3f\n",
61   bond_energy.energy(rc), planar_energy.energy(rc), dihedral_energy.energy(rc), nb_
→energy.energy(rc));
62 auto end = std::chrono::high_resolution_clock::now();
63 std::cerr << "# computed in " << std::chrono::duration<double>(end - start).count()
→<< " [s]\n";
64 }
```



ex_MMNonBonded**Keywords:**

- no_keywords

Categories:

- no_categories

Input files:

- 2gb1.pdb
- amber03_ffnonbonded.itp
- amber03_ffbonded.itp
- amber03_atoms.par

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <string>
3 #include <vector>
4 #include <tuple>
5
6 #include <utils/string_utils.hh>
7 #include <utils/LogManager.hh>
8 #include <core/data/structural/Structure.hh>
9 #include <simulations/forcefields/mm/MMBondType.hh>
10 #include <simulations/forcefields/mm/MMBondEnergy.hh>
11 #include <simulations/forcefields/mm/MMNonBondedSW.hh>
12 #include <simulations/forcefields/mm/MMNonBonded.hh>
13 #include <simulations/forcefields/mm/MMForceField.hh>
14
15 /** \todo_code Fix this demo one MM topology files parser is ready
16 *
17 */
18 int main(const int argc, const char *argv[]) {
19
20     using namespace simulations::forcefields::mm; // for all MM - related force field
21     // classes
22     using namespace core::data::structural; // for Structure, Residue, PdbAtom
23     using namespace core::data::basic; // for Vec3
24     using namespace simulations::systems; // for ResidueChain
25
26     utils::LogManager::get().set_level("FINE");
27
28     if (argc < 2) {

```

(continues on next page)

(continued from previous page)

```
28     std::cerr << "USAGE:\n\t./ex_MMNonBondEnergy data_atom_typing data_bond_typing_\n";
29     ↪input.pdb\n\n";
30     exit(0);
31 }
32
33 // --- Read atom typing and bond parameters
34 MMForceField mmff("AMBER03");
35 MMBondedParameters bonded_manager(argv[2],mmff.mm_atom_typing());
36
37 // --- Read structure for which calculate bond energy
38 core::data::io::Pdb reader(argv[3]); // file name (PDB format, may be gzip-ped)
39 Structure_SP strctr = reader.create_structure(0);
40 auto rc = CartesianChains(mmf.mmm_atom_typing(), *strctr);
41
42 // --- Create molecular mechanic bond energy object
43 MMBondEnergy bond_energy(rc, bonded_manager);
44
45 // --- Create non-bonded energy; pass the reference to bond energy object so non-
46 ↪bonded energy can exclude bonds
47 MMNonBonded nb_energy(rc,bond_energy);
48 std::cout << "# list of atoms pair excluded from pairwise calculation information\n";
49 // nb_energy.get_neighbor_list().print(std::cout);
50
51 // --- Calculate total non-bonded energy of the structure
52 double energy_by_str = nb_energy.energy(rc);
53 std::cout << utils::string_format("%10.3f\n", energy_by_str);
54 }
```



ex_MeanFieldDistributions

Prints values for a given Mean Field potential so it can be plotted nicely. The parameters of the program are: MF potential file name, pseudocounts, min_x, max_x, potential name [other potential names ..]

USAGE:

```
ex_MeanFieldDistributions "forcefield/cabs/R13_cabs.dat" 0.01 2.0 15.0 AD.HH  
(note that apostrophes may be mandatory, otherwise bash will not pass the arguments  
→correctly)
```

Keywords:

- force field

Categories:

- simulations/forcefields/mf/MeanFieldDistributions

Output files:

- stdout.out

Program source:

```
1 #include <string>  
2  
3 #include <simulations/forcefields/mf/MeanFieldDistributions.hh>  
4  
5 #include <utils/exit.hh>  
6  
7 std::string program_info = R"(  
8  
9 Prints values for a given Mean Field potential so it can be plotted nicely. The  
→parameters of the program are:  
10 MF potential file name, pseudocounts, min_x, max_x, potential name [other potential  
→names ..]  
11 USAGE:  
12     ex_MeanFieldDistributions "forcefield/cabs/R13_cabs.dat" 0.01 2.0 15.0 AD.HH  
13 (note that apostrophes may be mandatory, otherwise bash will not pass the arguments  
→correctly)  
14 )";  
15  
16 /** @brief Prints values for a given Mean Field potential so it can be plotted nicely.  
17 *  
18 * The program works for any potential stored in the Bioshell row-wise format; both  
→CABS and SURPASS potentials may be  
19 * plotted with this utility. Program usage:  
20 *  
21 * ex_MeanFieldDistributions "forcefield/cabs/R13_cabs.dat" 0.01 2.0 15.0 AD.HH  
22 *  
23 * where the parameters of the program are:  
24 *   MF potential file name, pseudocounts, min_x, max_x, potential name [other  
→potential names ..]
```

(continues on next page)

(continued from previous page)

```

25
26 * CATEGORIES: simulations/forcefields/mf/MeanFieldDistributions
27 * KEYWORDS: force field
28 */
29 int main(const int argc, const char *argv[]) {
30
31     using namespace simulations::forcefields::mf;
32
33     if(argc < 5) utils::exit_OK_with_message(program_info); // --- complain about
34     ↪missing program parameter
35
36     double pseudocounts = utils::from_string<double>(argv[2]);
37     double min_x = utils::from_string<double>(argv[3]);
38     double max_x = utils::from_string<double>(argv[4]);
39
40     std::shared_ptr<MeanFieldDistributions> mf = load_1D_distributions(argv[1], ↪
41     ↪pseudocounts);
42     std::vector<EnergyComponent_SP> terms;
43     if (argc > 5)
44         for (int i = 5; i < argc; ++i) {
45             std::string label(argv[i]);
46             if(mf->contains_distribution(label)) terms.push_back(mf->at(label));
47             else std::cerr << "Key " << label << " not found!\n";
48         }
49     else
50         for (const std::string label:mf->known_distributions())
51             terms.push_back(mf->at(label));
52
53     for (double d = min_x; d <= max_x; d += 0.0125) {
54         std::cout << utils::string_format("%7.3f",d);
55         for (const auto e : terms) std::cout << utils::string_format(" %8.3f", (*e)(d));
56         std::cout << "\n";
57     }
58 }
```



ex_Monomer

Unit test which demonstrates functionality of core::chemical::Monomer data type.

USAGE:

```
./ex_Monomer
```

Keywords:

- monomers

Categories:

- core::chemical::Monomer

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <algorithm> // for std::count_if
3 #include <iterator> // for std::distance
4
5 #include <utils/string_utils.hh>
6 #include <core/chemical/Monomer.hh>
7 #include <core/chemical/monomer_io.hh>
8 #include <utils/options/OptionParser.hh>
9 #include <utils/exit.hh>
10
11 std::string program_info = R"(
12
13 Unit test which demonstrates functionality of core::chemical::Monomer data type.
14
15 USAGE:
16 ./ex_Monomer
17
18 )";
19
20 // First we declare a function object used to count how many monomers have type = 'P'
21 // i.e. "protein"
22 bool IsAA(const core::chemical::Monomer &m) { return (m.type == 'P') || (core::chemical::get(m.parent_id).type == 'P'); }
23
24 /**
25 * CATEGORIES: core::chemical::Monomer
26 * KEYWORDS: monomers
27 */
28 int main(const int argc, const char *argv[]) {
29

```

(continues on next page)

(continued from previous page)

```

30  using namespace core::chemical;
31  // First we iterate over all monomers and count, how many of them are actually
32  // amino acids
33  // See how bioshell's iterators work together with std library
34  int n_aa = std::count_if(Monomer::cbegin(), Monomer::cend(), IsAA);
35
36  std::cout << std::distance(Monomer::cbegin(), Monomer::cend()) << " standard
37  monomers found, including "
38  << n_aa << " peptide-forming.\n";
39
40  std::cout << "The order of standard amino acid residues is:\n";
41  for (core::index2 i = 0; i < n_aa; ++i) {
42    const Monomer &m = Monomer::get(i);
43    std::cout << utils::string_format("%2d %c %3s %c\n", i, m.code1, m.code3.c_str(), 
44  m.type);
45
46  load_monomers_from_db(); // --- load the database of all known monomers
47  // Count amino acid monomers again
48  n_aa = std::count_if(Monomer::cbegin(), Monomer::cend(), IsAA);
49  std::cout << "Monomer database loaded; " << std::distance(Monomer::cbegin(), 
50  Monomer::cend())
51  << " monomers found, including " << n_aa << " peptide-forming.\n";
52
53  // Now let's count how many non-standard residues are derived from ALA
54  // Simply a parent_id of a monomer must be equal to ALA.id
55  // This time we use a lambda expression rather than a functor
56  n_aa = std::count_if(Monomer::cbegin(), Monomer::cend(), [] (const Monomer &m) { 
57  return m.parent_id == Monomer::ALA.id; });
58  std::cout << "There are " << n_aa << " derived from alanine\n";
59
}

```



ex_MonomerStructure

Unit test which shows how to read CIF files.

USAGE:

```
ex_MonomerStructure file.cif
```

EXAMPLE:

```
ex_MonomerStructure AA3.cif
```

Keywords:

- *CIF input*

Categories:

- core/chemical/MonomerStructure

Input files:

- AA3.cif

Output files:

- stdout.out

Program source:

```
1 #include <utils/Logger.hh>
2 #include <utils/LogManager.hh>
3 #include <core/chemical/MonomerStructure.hh>
4 #include <core/chemical/MonomerStructureFactory.hh>
5
6
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 Unit test which shows how to read CIF files.
12
13 USAGE:
14     ex_MonomerStructure file.cif
15 EXAMPLE:
16     ex_MonomerStructure AA3.cif
17
18 )";
19
20 /**
21 * @brief ex_MonomerStructure tests reading CIF files
22 * @CATEGORIES: core/chemical/MonomerStructure
```

(continues on next page)

(continued from previous page)

```

23 * KEYWORDS: CIF input
24 */
25 int main(const int argc, const char *argv[]) {
26
27     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
28     ↪missing program parameter
29
30     utils::LogManager::FINE(); // --- INFO is the default logging level; set it to FINE_
31     ↪to see more
32     core::chemical::MonomerStructure_SP str = core::chemical::MonomerStructure::from_
33     ↪cif(argv[1]);
34     std::cout<<"\nPOLAR H: ";
35     for (auto i: str->polar_hydrogens()) std::cout<<i->atom_name()<<" ";
36     std::cout<<"\nNONPOLAR H: ";
37
38     for (auto i: str->nonpolar_hydrogens()) std::cout<<i->atom_name()<<" ";
39     std::cout<<"\nNONPOLAR: ";
40
41     for (auto i: str->nonpolar_heavy()) std::cout<<i->atom_name()<<" ";
42     std::cout<<"\nDONORS: ";
43
44     for (auto i: str->hydrogen_donors()) std::cout<<i->atom_name()<<" ";
45     std::cout<<"\nACCEPTORS: ";
46
47     for (auto i: str->hydrogen_acceptors()) std::cout<<i->atom_name()<<" ";
48     std::cout<<"\n";
49
50     core::chemical::MonomerStructureFactory m =
51     ↪core::chemical::MonomerStructureFactory::get_instance();
52     core::chemical::MonomerStructure_SP mstr = m.get("PRO");
53     std::cout<<mstr->code3<<"\n";
54     std::cout<<"\nPOLAR H: ";
55     for (auto i: mstr->polar_hydrogens()) std::cout<<i->atom_name()<<" ";
56 }
```



ex_NeighborGrid3D

ex_NeighborGrid3D finds possible structural neighbors of a given residue using a 3D hashing grid

USAGE:

```
ex_NeighborGrid3D 2gb1.pdb 4.0 A:12
```

where 2gb1.pdb is an input file, 4.0 - grid size, A:12 - selector of a query residue

Keywords:

- *PDB input*
- *structure selectors*

Categories:

- core::calc::structural::NeighborGrid3D

Input files:

- 2gb1.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <algorithm>
3 #include <set>
4
5 #include <core/data/io/Pdb.hh>
6 #include <core/data/structural/selectors/structure_selectors.hh>
7 #include <core/calc/structural/NeighborGrid3D.hh>
8
9 #include <utils/exit.hh>
10 #include <utils/LogManager.hh>
11
12 std::string program_info = R"(
13
14 ex_NeighborGrid3D finds possible structural neighbors of a given residue using a 3D_
15 ↬hashing grid
16
17 USAGE:
18   ex_NeighborGrid3D 2gb1.pdb 4.0 A:12
19
20 where 2gb1.pdb is an input file, 4.0 - grid size, A:12 - selector of a query residue
21

```

(continues on next page)

(continued from previous page)

```

22 ) ";
23
24
25 /** @brief Finds possible structural neighbors of a given atom
26 *
27 * CATEGORIES: core::calc::structural::NeighborGrid3D
28 * KEYWORDS: PDB input; structure selectors
29 */
30 int main(const int argc, const char* argv[]) {
31
32     if (argc < 3) utils::exit_OK_with_message(program_info); // --- complain about_
33     ↪missing program parameter
34     utils::LogManager::FINE();
35
36     using namespace core::data::io;
37     Pdb reader(argv[1], all_true(is_not_water,is_not_alternative)); // --- file name_
38     ↪(PDB format, may be gzip-ped)
39
40     utils::Logger logs("ex_NeighborGrid3D");
41     using namespace core::data::structural;
42     Structure_SP strctr = reader.create_structure(0);
43     selectors::SelectChainResidues select_query(argv[3]);
44     // --- Here we find the query residue, selected by a selector string
45     Residue_SP query_resid = nullptr;
46     for(auto it=strctr->first_residue();it!=strctr->last_residue();++it) {
47         if (select_query(*it)) {
48             query_resid = *it;
49             logs << utils::LogLevel::INFO << "selecting spatial neighbors of " <<
50             " " << query_resid->residue_type().code3 << query_resid->residue_id() <<
51             "\n";
52             break;
53         }
54     }
55
56     // --- Create a 3D grid object
57     double grid_mesh = atof(argv[2]);
58     core::calc::structural::NeighborGrid3D grid(*strctr,grid_mesh);
59
60     // --- Print content of the grid
61     std::cout << "# Atoms as they are located on the grid\n";
62     std::cout << "# Grid center is: " << grid.cx() << " " << grid.cy() << " " << grid.
63     ↪cz() << "\n";
64     core::index2 ix, iy, iz;
65     for(const auto & hash : grid.filled_cells()) {
66         grid.xyz_from_hash(hash, ix, iy, iz);
67         for(const PdbAtom_SP & at : grid.get_cell(hash)) {
68             std::cout << "# " << hash << " " << at->owner()->residue_type().code3 << at->
69             ↪owner()->id() << " " << *at
70             << " ix: " << ix << " iy: " << iy << " iz: " << iz << "\n";
71     }
72 }
73
74     // --- Print neighbor cells of the selected residue
75     core::index4 hash_ca = grid.hash(*query_resid->find_atom_safe(" CA "));
76     std::vector<core::index4> neighb_hash;
77     grid.get_neighbor_cells(hash_ca, neighb_hash);
78     std::cout << "# neighbors of a cell " << hash_ca << "\n# ";

```

(continues on next page)

(continued from previous page)

```

74   for (core::index4 n:neighb_hash) std::cout << " " << n;
75   std::cout << "\n";
76
77   // --- Mark the selection on a PDB file by setting B-factor to 10.0 (all other
78   // atoms to 0.0)
79   float max_distance = 0;
80   std::vector< PdbAtom_SP> result;
81   for(auto it=strctr->first_atom();it!=strctr->last_atom();++it) (**it).b_factor(0.0);
82   for(PdbAtom_SP a : *query_resid) {
83     result.clear();
84     grid.get_neighbors(*a,result);
85     for(auto atom_sp:result) {
86       atom_sp->b_factor(10.0);
87       max_distance = std::max(max_distance,atom_sp->distance_to(*a));
88     }
89
90   for(auto it=strctr->first_atom();it!=strctr->last_atom();++it)
91     std::cout << (**it).to_pdb_line() << "\n";
92   std::cout << "# Max distance: " <<max_distance << "\n";
93 }
```



ex_NormalDistribution

Unit test for NormalDistribution class. The example withdraws 1000 random numbers from a normal distribution and later it estimates a normal distribution from the sample.

USAGE:

```
./ex_NormalDistribution
```

Keywords:

- NormalDistribution
- *random numbers*

Categories:

- core/calc/statistics/NormalDistribution; core/calc/statistics/Random

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <random>
3 #include <math.h>
4
5 #include <core/calc/statistics/NormalDistribution.hh>
6 #include <core/calc/statistics/Random.hh>
7 #include <utils/options/OptionParser.hh>
8
9 std::string program_info = R"(
10
11 Unit test for NormalDistribution class.
12
13 The example withdraws 1000 random numbers from a normal distribution and later it estimates
14 a normal distribution from the sample.
15
16 USAGE:
17 ./ex_NormalDistribution
18
19 )";
20
21 /**
22 * @brief Unit test for NormalDistribution class.
23 *
24 * CATEGORIES: core/calc/statistics/NormalDistribution; core/calc/statistics/Random
25 * KEYWORDS: NormalDistribution; random numbers
26 */
27 int main(const int argc, const char *argv[]) {

```

(continues on next page)

(continued from previous page)

```
28 if ((argc > 1) && utils::options::call_for_help(argv[1]))
29     utils::exit_OK_with_message(program_info);
30
31 using namespace core::calc::statistics;
32
33 Random rd = core::calc::statistics::Random::get();
34 rd.seed(12345); // --- seed the generator for repeatable results
35 std::mt19937 gen(rd());
36 std::normal_distribution<> d(10.5, 2.0); // --- The original distribution we take a
37 // sample from
38 // --- Note that the container for samples is two-dimensional! Each sample is
39 // placed in a separate row
40 std::vector<std::vector<double> > data;
41 std::vector<double> row(1);
42 for (unsigned short i = 0; i < 1000; ++i) {
43     row[0] = (d(gen));
44     data.push_back(row); // --- This works only because C++ makes an implicit copy of
45 // the vector we place into the outer vector
46 }
47
48 // --- Here we estimate the distribution parameters
49 core::calc::statistics::NormalDistribution n(0.0, 1.0);
50 std::vector<double> E = n.estimate(data);
51 std::cout << "True values:      average = 10.5; stdev = 2.0\n";
52 std::cout << "Estimated values: average =
53             << E[0] << " sdev = " << E[1] << "\n";           // Calculate average
54 // and stdev of values in the vector
55 }
```



ex_OptionParser

Shows how to use BioShell command line parser in your own program

Keywords:

- option parsing

Categories:

- utils::options::OptionParser

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <utils/options/Option.hh>
4 #include <utils/options/OptionParser.hh>
5 #include <utils/LogManager.hh>
6
7 using namespace utils::options;
8
9 /** @brief Shows how to use BioShell command line parser in your own program
10 */
11 /* To test this program, run:
12 * ./ex_OptionParser -n=4 -nn=1,2,3,4
13 */
14 /* CATEGORIES: utils::options::OptionParser
15 * KEYWORDS: option parsing
16 */
17 int main(const int cnt, const char *argv[]) {
18
19     // --- Limit the stdout on stderr (logging)
20     LogManager::WARNING();
21
22     // --- First get the parser instance (it's a singleton)
23     utils::options::OptionParser &cmd = OptionParser::get("ex_OptionParser");
24
25     // --- This is how to register an option that has already been declared in BioShell_
26     // library
27     cmd.register_option(utils::options::verbose, help);
28
29     // --- User can also declare non-standard options
30     Option number("-n", "-number", "returns an integer");
31     Option numbers("-nn", "-numbers", "returns a vector of integers");
32     Option value("-x", "-value_x", "returns a real value of X");
33     // --- Options that have been declared, must also be registered
34     // --- (Declaration doesn't mean automatic registration)
35     cmd.register_option(number, numbers, value);
```

(continues on next page)

(continued from previous page)

```

35 // --- after all the relevant options were registered, we parse a program command_line
36 cmd.parse cmdline(cnt, argv);
37
38 // ---- User should not check for -help and -verbose flags : this is automatically done by OptionParser
39
40 // --- Once command line has been parsed, we may check for a program parameter and retrieve its value:
41 if (numbers.was_used()) std::cout << "A number given: " << option_value<int>(number) << "\n";
42
43 // --- Options may be also accessed by their long name (but not by the abbreviated name, so cmd.was_used("-x") won't work
44 if (cmd.was_used("-value_x")) std::cout << "A number given: " << option_value<double>("-value_x") << "\n";
45
46 // --- This shows how to read a vector of values
47 if(cmd.was_used(numbers)) {
48     std::vector<int> v = option_value<std::vector<int>, int>(numbers);
49     std::cout << "Given set of values: ";
50     for (auto vi : v) std::cout << vi << " ";
51     std::cout << "\n";
52     // --- This is how the raw string given at the command line may be accessed:
53     std::cout << "The raw string associated with -value_x option was: " << cmd.value_string(numbers) << "\n";
54 }
55
56 }
```



ex_P2QuantileEstimation

ex_P2QuantileEstimation reads a file with real values and calculates a quantile using the P-square algorithm If no input file is provided, the program calculates 0.25, 0.5 and 0.75 quantiles of a random sample from normal distribution

USAGE:

```
ex_P2QuantileEstimation [infile p_value]
```

EXAMPLE:

```
ex_P2QuantileEstimation random_normal.txt 0.5
```

REFERENCE: Jain, Raj, Imrich Chlamtac. “The P2 algorithm for dynamic calculation of quantiles and histograms without storing observations.” Communications of the ACM 28.10 (1985): 1076-1085. doi:10.1145/4372.4378

Keywords:

- *statistics*

Categories:

- core/calc/statistics/OnlineStatistics; core/calc/statistics/Random

Input files:

- random_N(0,1).txt_

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/index.hh>
4 #include <core/calc/statistics/Random.hh>
5 #include <core/calc/statistics/P2QuantileEstimation.hh>
6
7 std::string program_info = R"(
8
9 ex_P2QuantileEstimation reads a file with real values and calculates a quantile using
10    ↪the P-square algorithm
11
12 If no input file is provided, the program calculates 0.25, 0.5 and 0.75 quantiles
13    of a random sample from normal distribution
14
15 USAGE:
16     ex_P2QuantileEstimation [infile p_value]
17
18 EXAMPLE:

```

(continues on next page)

(continued from previous page)

```

18   ex_P2QuantileEstimation random_normal.txt 0.5
19
20 REFERENCE:
21 Jain, Raj, Imrich Chlamtac. "The P2 algorithm for dynamic calculation of quantiles
22 and histograms without storing observations." Communications of the ACM 28.10 (1985): ↵
23 →1076-1085. doi:10.1145/4372.4378
24 );
25
26 /** @brief Reads a file with real values and calculates simple statistics: min, mean, ↵
27 →stdev, max.
28 */
29
30 * If no input file is provided, the program calculates the statistics from a random ↵
31 →sample
32 */
33
34 * CATEGORIES: core/calc/statistics/OnlineStatistics; core/calc/statistics/Random
35 * KEYWORDS: statistics
36 */
37
38 int main(const int argc, const char *argv[]) {
39
40     if(argc < 3) {
41         // --- complain about missing program parameter
42         std::cerr << program_info;
43         // ----- Use the random engine if no data is provided
44         core::calc::statistics::Random r = core::calc::statistics::Random::get();
45         r.seed(12345); // --- seed the generator for repeatable results
46         std::normal_distribution<double> normal_random;
47         core::calc::statistics::P2QuantileEstimation quartile1(0.25), quartile2(0.5), ↵
48         →quartile3(0.75);
49         for (core::index4 n = 0; n < 10000; ++n) {
50             double rr = normal_random(r);
51             quartile1(rr);
52             quartile2(rr);
53             quartile3(rr);
54         }
55         std::cout << "Quantile 0.25 :" << quartile1.p_value() << "\n"; // Should be -0.675
56         std::cout << "Quantile 0.50 :" << quartile2.p_value() << "\n"; // Should be 0.0
57         std::cout << "Quantile 0.75 :" << quartile3.p_value() << "\n"; // Should be 0.675
58
59     } else {
60         std::ifstream in(argv[1]);
61         core::calc::statistics::P2QuantileEstimation stats(atof(argv[2]));
62         double r;
63         core::index4 cnt = 0;
64         while (in >> r) {
65             ++cnt;
66             stats(r);
67         }
68         std::cout << "Quantile " << atof(argv[2]) << " " << stats.p_value() << " based on "
69         →" << cnt << " observations\n";
70     }
71 }

```



ex_PairwiseAlignment

Simple example showing how to work with PairwiseAlignment data structure, e.g. how to retrieve arbitrary data according to a sequence alignment object

USAGE:

```
./ex_PairwiseAlignment
```

Keywords:

- *sequence alignment*

Categories:

- core::alignment::PairwiseAlignment

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <iterator>
3
4 #include <core/alignment/PairwiseAlignment.hh>
5 #include <utils/options/OptionParser.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Simple example showing how to work with PairwiseAlignment data structure, e.g. how to
11 retrieve arbitrary data according to a sequence alignment object
12
13 USAGE:
14 ./ex_PairwiseAlignment
15
16 )";
17
18 /** @brief Simple example showing how to retrieve arbitrary data according to a
19  * sequence alignment object
20  *
21  * CATEGORIES: core::alignment::PairwiseAlignment;
22  * KEYWORDS: sequence alignment
23  */
24 int main(const int argc, const char *argv[]) {
25
26     if ((argc > 1) && utils::options::call_for_help(argv[1]))
27         utils::exit_OK_with_message(program_info);
28
29     // ----- The two sequences that are already globally aligned, therefore
30     // indexes in both sequences start from 0
```

(continues on next page)

(continued from previous page)

```

29 core::alignment::PairwiseAlignment ali("FTFTALILL-AVAV", 0, "--FTAL-LAAV--", 0);
30
31 // ----- query "objects" : that may be C-alpha atoms, residues, etc; here just ↵
32 // chars
33 std::vector<char> query_chars = {'F', 'T', 'F', 'T', 'A', 'L', 'I', 'L', 'L', 'A', ↵
34 // 'V', 'A', 'V'}; // --- all the characters of the query sequence
35 std::vector<char> tmplt_chars = {'F', 'T', 'A', 'L', 'L', 'L', 'A', 'A', 'V'}; ↵
36 // --- all the characters of the template sequence
37
38 // ----- container for the expected result
39 std::vector<char> query_chars_aligned;
40 std::vector<char> tmplt_chars_aligned;
41
42 // ----- set up query "objects" in the order as they appear in the alignment; ↵
43 // print result on the screen
44 ali.get_aligned_query(query_chars, '-', query_chars_aligned);
45
46 // ----- show results (it should be identical as the original alignment
47 std::copy(query_chars_aligned.begin(), query_chars_aligned.end(), std::ostream_ ↵
48 iterator<char>(std::cout, ""));
49 std::cout << "\n"; // ----- Should print FTFTALILL-AVAV
50
51 // ----- Now we extract both query and template objects; only the mutually ↵
52 // aligned positions (no gaps)
53 query_chars_aligned.clear();
54 ali.get_aligned_query_template(query_chars, tmplt_chars, query_chars_aligned, tmplt_ ↵
55 chars_aligned);
56
57 // ----- show results
58 std::copy(query_chars_aligned.begin(), query_chars_aligned.end(), std::ostream_ ↵
59 iterator<char>(std::cout, ""));
60 std::cout << "\n"; // ----- Should print FTALLAV
61 std::copy(tmplt_chars_aligned.begin(), tmplt_chars_aligned.end(), std::ostream_ ↵
62 iterator<char>(std::cout, ""));
63 std::cout << "\n"; // ----- Should also print FTALLAV
64
65 // ----- ... and finally print the alignment as a path
66 std::cout << ali.to_path() << "\n";
67
68 }

```



ex_PairwiseSequenceAlignment

Simple example showing how to work with PairwiseSequenceAlignment data structure, e.g. how to create such an object and how to print it in different formats.

USAGE:

```
./ex_PairwiseSequenceAlignment
```

Keywords:

- *sequence alignment*

Categories:

- core::alignment::PairwiseAlignment; core::alignment::PairwiseSequenceAlignment

Output files:

- `stdout.out`

Program source:

```
1 #include <iostream>
2
3 #include <core/BioShellEnvironment.hh>
4
5 #include <core/alignment/on_alignment_computations.hh>
6 #include <core/alignment/PairwiseAlignment.hh>
7 #include <core/alignment/PairwiseSequenceAlignment.hh>
8 #include <core/data/io/alignment_io.hh>
9 #include <core/data/sequence/Sequence.hh>
10 #include <utils/options/OptionParser.hh>
11
12 std::string program_info = R"(
13
14 Simple example showing how to work with PairwiseSequenceAlignment data structure, e.g.
15 ↵ how to
16 create such an object and how to print it in different formats.
17
18 USAGE:
19 ./ex_PairwiseSequenceAlignment
20 )";
21
22 std::string Q52825_1 =
23 ↵ "IDVLLGADDGSLAFVPSEFSISPGEKIVFKNNAGFPHNIVFDEDSIPSGVDASKISMSEEDLLNAKGETFEVALSNKGEYSFYCSPHQGAGMVGKV
24 ↵ ";
25 std::string P80401_2 = "VQMLNKGTGDAMVFEPGFLKIAPGDTVTFIPTDKS-HNVETFKGLIPDGV-----
26 ↵ PDFKSKPNEQYQVKFDIPGAYVLKCTPHVMGMVALIQV";
27
28 /**
29  * @brief Simple example showing how to work with PairwiseSequenceAlignment data_
30  * structure.
31  */
```

(continues on next page)

(continued from previous page)

```

26
27 * CATEGORIES: core::alignment::PairwiseAlignment; ↵
28 ↪core::alignment::PairwiseSequenceAlignment;
29 * KEYWORDS: sequence alignment
30 */
31
32 int main(const int argc, const char *argv[]) {
33
34     if ((argc > 1) && utils::options::call_for_help(argv[1]))
35         utils::exit_OK_with_message(program_info);
36
37     using namespace core::alignment;
38     using namespace core::alignment::scoring;
39     using namespace core::data::sequence; // for core::data::sequence::Sequence
40
41     // ----- Test for global alignment -----
42     // --- Alignment defined as path : '-' and '-' mean a gap in a template and in a ↵
43     ↪query sequence, respectively; '*' is a match
44     PairwiseAlignment_SP ali = std::make_shared<PairwiseAlignment>(0, 0, 0, "----*---" ↪
45     "****|**--");
46
47     // ----- The two sequences that will be aligned
48     Sequence_SP query = std::make_shared<Sequence>("query", "ITFTALILLAVAV", 1);
49     Sequence_SP tmplt = std::make_shared<Sequence>("tmplt", "FTALLAAV", 1);
50
51     PairwiseSequenceAlignment seq_ali(ali, query, tmplt);
52
53     // Show alignment as a path
54     std::cout << "Alignment path:\n" << ali->to_path() << "\n\n";
55
56     // ----- Print the alignment in Edinburgh format
57     core::index2 identity = sum_identical(seq_ali);
58     core::index2 n_gaps = seq_ali.alignment->length() - seq_ali.alignment->n_aligned();
59     std::cout << "# score: " << seq_ali.alignment_score() << " length: " << seq_ali. ↪
60     alignment->length()
61     << " n_identical: " << identity << " n_gaps: " << n_gaps << "\n";
62     core::data::io::write_edinburgh(seq_ali, std::cout, 80);
63
64     // ----- Test for local alignment -----
65     // --- Alignment defined as path : '-' and '-' mean a gap in a template and in a ↵
66     ↪query sequence, respectively; '*' is a match
67     PairwiseAlignment_SP loc_ali = std::make_shared<PairwiseAlignment>(2, 0, 0.0, "*****" ↪
68     "****|**");
69     PairwiseSequenceAlignment loc_seq_ali(loc_ali, query, tmplt);
70
71     // ----- Print the alignment in Edinburgh format
72     identity = sum_identical(loc_seq_ali);
73     n_gaps = loc_seq_ali.alignment->length() - loc_seq_ali.alignment->n_aligned();
74     std::cout << "# score: " << loc_seq_ali.alignment_score() << " length: " << loc_seq_ ↪
75     ali.alignment->length()
76     << " n_identical: " << identity << " n_gaps: " << n_gaps << "\n";
77     core::data::io::write_edinburgh(loc_seq_ali, std::cout, 80);
78
79
80     PairwiseSequenceAlignment loc_seq_ali2("Q52825_1", Q52825_1, 0, "P80401_2", P80401_ ↪
81     "2, 0, 0.0");
82     loc_seq_ali2.template_sequence->first_pos(28);
83     loc_seq_ali2.query_sequence->first_pos(1);

```

(continues on next page)

(continued from previous page)

```
75     core:::data:::io:::write_edinburgh(loc_seq_ali2, std::cout, 80);  
76 }
```



ex_Pca3

Unit test orients 3D points along the axes using the PCA algorithm.

USAGE:

```
./ex_Pca3
```

REFERENCE: Pearson, Karl. "On lines and planes of closest fit to systems of points in space." Philosophical Magazine 2 (1901): 559-572. doi:10.1080/14786440109462720.

Keywords:

- PCA
- transformations

Categories:

- core/calc/numeric/basic_algebra.hh

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <random>
3
4 #include <core/index.hh>
5 #include <core/calc/numeric/basic_algebra.hh>
6 #include <core/calc/numeric/Pca3.hh>
7 #include <utils/exit.hh>
8 #include <utils/options/OptionParser.hh>
9
10 std::string program_info = R"(
11
12 Unit test orients 3D points along the axes using the PCA algorithm.
13
14 USAGE:
15 ./ex_Pca3
16
17 REFERENCE:
18 Pearson, Karl. "On lines and planes of closest fit to systems of points in space."
19 Philosophical Magazine 2 (1901): 559-572. doi:10.1080/14786440109462720.
20
21 )";
22
23 /**
24 * @brief Orients 3D points along the axes using PCA algorithm
25 */
26 * CATEGORIES: core/calc/numeric/basic_algebra.hh
27 * KEYWORDS: PCA; transformations

```

(continues on next page)

(continued from previous page)

```
27 */
28 int main(const int argc, const char *argv[]) {
29
30     if ((argc > 1) && utils::options::call_for_help(argv[1]))
31         utils::exit_OK_with_message(program_info);
32
33     using namespace core::data::basic; // --- for Vec3 and Array2D
34
35     std::mt19937 gen;
36     std::uniform_real_distribution<double> r(0, 1);
37     std::vector<Vec3> points3d;
38     for (core::index2 i = 0; i < 100; ++i) {
39         double x = r(gen), y = r(gen), z = r(gen);
40         points3d.emplace_back(x + 0.3 * y + 0.6 * z, 0.4 * x + 1.9 * y + 0.7 * z, 0.3 * x_
41         ↪+ 0.1 * y + 0.7 * z);
42     }
43     core::calc::numeric::Pca3 pca3(points3d);
44     auto rt = pca3.create_transformation();
45     for (auto &p:points3d) {
46         std::cout << p << " ";
47         rt.apply(p);
48         std::cout << p << "\n";
49     }
}
```



ex_Pdb

Unit test which shows how to read a PDB file and create a Structure object. The program reads a given file with a PDB line filter that passes only backbone atoms; prints experimental method, resolution, R-value and R-free about the input file.

USAGE:

```
ex_Pdb 5edw.pdb
```

Keywords:

- *PDB input*
- *PDB line filter*
- *Structure*

Categories:

- core::data::io::Pdb

Input files:

- 2gb1.pdb
- 5edw.pdb
- 3dcg.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <core/data/io/Pdb.hh>
3 #include <utils/exit.hh>
4
5 std::string program_info = R"(
6
7 Unit test which shows how to read a PDB file and create a Structure object.
8
9 The program reads a given file with a PDB line filter that passes only backbone atoms;
10 prints experimental method, resolution, R-value and R-free about the input file.
11
12 USAGE:
13     ex_Pdb 5edw.pdb
14
15 )";
16
17 /**
 * @brief Reads a PDB file and creates a Structure object.
```

(continues on next page)

(continued from previous page)

```

18 *
19 * Input PDB data is filtered so only protein backbone atoms are loaded
20 *
21 * CATEGORIES: core::data::io::Pdb;
22 * KEYWORDS: PDB input; PDB line filter; Structure
23 */
24 int main(const int argc, const char* argv[]) {
25
26     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
27     ↪missing program parameter
28     for (int i = 1; i < argc; ++i) {
29         core::data::io::Pdb reader(argv[i], // file name (PDB format, may be gzip-ped)
30         core::data::io::is_bb,           // a predicate to read only the ATOM lines_
31     ↪corresponding to backbone atoms
32         core::data::io::only_ss_from_header, true);                      //_
33     ↪parse PDB header
34     core::data::structural::Structure_SP backbone = reader.create_structure(0);
35     std::cout << "protein " << reader.pdb_code() << " has " << backbone->count_
36     ↪chains()
37         << " chain(s), " << backbone->count_residues()
38         << " residues and " << backbone->count_atoms() << " backbone atoms\n";
39     std::cout << "title : " << backbone->title() << "\n";
40     std::cout << "compound : " << backbone->compound() << "\n";
41     std::cout << "classification : " << backbone->classification() << "\n";
42     std::cout << "deposited : " << backbone->deposition_date() << "\n";
43     std::cout << "Is XRAY? : " << ((backbone->is_xray()) ? "YES\n" : "No\n");
44     std::cout << "Is NMR? : " << ((backbone->is_nmr()) ? "YES\n" : "No\n");
45     std::cout << "Is EM? : " << ((backbone->is_em()) ? "YES\n" : "No\n");
46     std::cout << "resolution : " << backbone->resolution() << "\n";
47     std::cout << "R-value : " << backbone->r_value() << "\n";
48     std::cout << "R-free : " << backbone->r_free() << "\n";
49     if(backbone->keywords().size()>0)
50         std::cout << "keywords : " << backbone->keywords()[0];
51     for (auto it = ++backbone->keywords().cbegin(); it != backbone->keywords().cend();
52     ↪++it)
53         std::cout << ", " << *it;
54     std::cout << "\n";
55 }
56 }
```



ex_Quaternion

ex_Quaternion illustrates how to use Quaternion class

Keywords:

- algebra

Categories:

- core::calc::numeric::Quaternion

Input files:

- 2kwi.pdb
- 2gb1.pdb

Output files:

- peptide.pdb
- stdout.out
- peptide_rot.pdb

Program source:

```

1 #include <iostream>
2
3 #include <core/calc/numeric/Quaternion.hh>
4 #include <core/data/basic/Vec3.hh>
5 #include <core/data/io/Pdb.hh>
6 #include <core/calc/statistics/Random.hh>
7
8 /** @brief ex_Quaternion illustrates how to use Quaternion class
9 *
10 * CATEGORIES: core::calc::numeric::Quaternion
11 * KEYWORDS: algebra
12 */
13 int main(const int argc, const char* argv[]) {
14
15     using namespace core::calc::numeric;
16     using namespace core::data::basic; // --- for Vec3
17
18     Quaternion p, rot;
19     core::calc::statistics::Random::seed(0);
20     rot.random();           // --- Random rotation axis
21     //rot = Quaternion::create_from_axis_angle(1,0,0,M_PI/3.0);
22
23     // ----- Read a structure to be rotated
24     core::data::io::Pdb reader(argv[1],core::data::io::is_not_alternative,
→core::data::io::only_ss_from_header, true);

```

(continues on next page)

(continued from previous page)

```
25 const auto strctr_sp = reader.create_structure(0);
26
27 // ----- Iterate over atoms and rotate them one by one
28 for(auto a_it = strctr_sp->first_atom(); a_it != strctr_sp->last_atom(); ++a_it) {
29     // --- use quaternion rotation method
30     p.i = (**a_it).x;
31     p.j = (**a_it).y;
32     p.k = (**a_it).z;
33     p.rotate_by(rot);
34     std::cout << p.i << " " << p.j << " " << p.k << "\n";
35
36     // --- and now the same, but with apply() method - output should be numerically
37     ↪the same
38     rot.apply(**a_it);
39     std::cout << (**a_it).to_pdb_line() << "\n";
40 }
41
42 return 0;
}
```



ex_REMC_Ar

The program runs an Replica Exchange MC simulation of argon gas. By default it starts from a regular lattice conformation unless an input file (PDB) with initial conformation is provided

USAGE:

```
ex_REMC_Ar n_atoms density small_cycles big_cycles n_exchange temperature_1_
↳temperature_2 [temperature_3 ...]
ex_REMC_Ar starting.pdb density small_cycles big_cycles n_exchange temperature_1_
↳temperature_2 [temperature_3 ...]
```

Keywords:

- *Mover*
- Replica Exchange
- *Monte Carlo*
- *sampling*

Categories:

- simulations::sampling::ReplicaExchangeMC

Output files:

- energy-1.dat
- replica_flow.dat
- energy-0.dat
- stdout.out
- movers-1.dat
- final.pdb
- movers-0.dat
- movers-2.dat
- energy-2.dat

Program source:

```
1 #include <cstdio>
2 #include <ctime>
3 #include <iostream>
4
5 #include <core/data/basic/Vec3Cubic.hh>
6
7 #include <utils/string_utils.hh>
8 #include <utils/options/Option.hh>
9 #include <utils/options/OptionParser.hh>
```

(continues on next page)

(continued from previous page)

```

10 #include <utils/options/output_options.hh>
11 #include <utils/options/sampling_options.hh>
12
13 #include <simulations/systems/CartesianAtomsSimple.hh>
14 #include <simulations/systems/BuildFluidSystem.hh>
15 #include <simulations/systems/SingleAtomType.hh>
16 #include <simulations/movers/TranslateAtom.hh>
17 #include <simulations/forcefields/cartesian/LJEnergySWhomogenic.hh>
18 #include <simulations/sampling/IsothermalMC.hh>
19 #include <simulations/observers/ObserveMoversAcceptance.hh>
20 #include <simulations/observers/TriggerEveryN.hh>
21 #include <simulations/observers/ObserveReplicaFlow.hh>
22 #include <simulations/observers/ObserveEnergyComponents.hh>
23 #include <simulations/observers/UpdateSystemTags.hh>
24 #include <simulations/observers/AdjustMoversAcceptance.hh>
25 #include <simulations/observers/cartesian/SimplePdbFormatter.hh>
26
27 #include <utils/exit.hh>
28
29 using namespace core::data::basic;
30
31 utils::Logger logs("ex_REMC_Ar");
32
33 std::string program_info = R"(
34
35 The program runs an Replica Exchange MC simulation of argon gas. By default it starts_
36 ↵from a regular lattice conformation
37 unless an input file (PDB) with initial conformation is provided
38 USAGE:
39   ex_REMC_Ar n_atoms density small_cycles big_cycles n_exchange temperature_1_
40 ↵temperature_2 [temperature_3 ...]
41   ex_REMC_Ar starting.pdb density small_cycles big_cycles n_exchange temperature_1_
42 ↵temperature_2 [temperature_3 ...]
43 )";
44
45 const double EPSILON = 1.654E-21;           // [J] per molecule
46 const double EPSILON_BY_K = EPSILON / 1.381E-23;      // = 119.6 in Kelvins
47 const double SIGMA = 3.4;                   // in Angstroms
48
49 /** @brief A helper function that creates a dummy structure of 830 argon atoms.
50 *
51 * The structure is necessary to be able to save the system state in the PDB format._
52 * It will not be used
53 * in the simulation - just for output formatting. The atoms may be stored in_
54 * multiple chains, because PDB file
55 * format allows a single chain to have at most 9999 atoms.
56 */
57 core::data::structural::Structure_SP create_argon_structure(const core::index4 n_ar) {
58
59   using namespace core::data::structural;
60   Structure_SP s = std::make_shared<Structure>("");
61   core::index2 n_chains = n_ar / 9999;
62   core::index4 n_atoms = 0;

```

(continues on next page)

(continued from previous page)

```

62   for (core::index2 ic = 0; ic < n_chains; ++ic) {
63     Chain_SP chain = std::make_shared<Chain>(std::string{utils::letters[ic]});
64     for (core::index4 i = 0; i < 9999; ++i) {
65       Residue_SP res = std::make_shared<Residue>(i + 1, " AR");
66       res->push_back(std::make_shared<PdbAtom>(i + 1, " AR ", _;
67     →core::chemical::AtomicElement::ARGON.z));
68       chain->push_back(res);
69       ++n_atoms;
70     }
71     s->push_back(chain);
72   }
73   if (n_atoms < n_ar) {
74     Chain_SP chain = std::make_shared<Chain>(std::string{utils::letters[n_chains]});
75     for (core::index4 i = n_atoms; i < n_ar; ++i) {
76       Residue_SP res = std::make_shared<Residue>(i + 1 - n_atoms, " AR");
77       res->push_back(std::make_shared<PdbAtom>(i + 1 - n_atoms, " AR ", _;
78     →core::chemical::AtomicElement::ARGON.z));
79       chain->push_back(res);
80     }
81     s->push_back(chain);
82   }
83
84
85  /** @brief Isothermal Monte Carlo simulation of argon gas.
86  *
87  */
88 int main(const int argc, const char* argv[]) {
89
90   using core::data::basic::Vec3Cubic;
91   using namespace simulations::systems;
92   using namespace simulations::observers::cartesian;
93   using namespace simulations::forcefields; // for CalculateEnergyBase, NeighborList
94   using namespace simulations::movers; // for MoversSet
95
96   // ----- Define some new types so the program is easier to read and lines get_
97   →shorter
97   typedef typename cartesian::LJEnergySWHomogenic LjEnergy;
98   typedef typename simulations::observers::ObserveEnergyComponents<TotalEnergy>_
99   →EnergyObserverType;
100  typedef std::shared_ptr<EnergyObserverType> EnergyObserverType_SP;
101
102  core::index4 n_atoms;
103
104  if (argc < 8) utils::exit_OK_with_message(program_info);
105
106  std::vector<core::data::structural::Structure_SP> argon_structures;
107  bool input_from_file = false;
108  if (utils::is_integer(argv[1])) {
109    n_atoms = atoi(argv[1]);
110    argon_structures.push_back(create_argon_structure(n_atoms));
111  }
112  else { // --- read an input file if given
113    core::data::io::Pdb reader(argv[1]);
114    for(core::index2 i_str=0;i_str<reader.count_models();++i_str)
      argon_structures.push_back(reader.create_structure(i_str));

```

(continues on next page)

(continued from previous page)

```

115     n_atoms = argon_structures[0]->count_atoms();
116     input_from_file = true;
117 }
118
119 double density = atof(argv[2]);
120 core::index4 n_inner_cycles = atoi(argv[3]);
121 core::index4 n_outer_cycles = atoi(argv[4]);
122 core::index4 n_exchanges = atoi(argv[5]);
123 double ar_volume = 4.0 / 3.0 * M_PI * SIGMA * SIGMA * n_atoms;
124 double box_len = pow(ar_volume / density, 0.33333333333333);
125 core::calc::statistics::Random::seed(1234);
126
127 // --- Initialize periodic boundary conditions for the box length
128 core::data::basic::Vec3Cubic::set_box_len(box_len);
129 logs << utils::LogLevel::INFO << "box width for "<<int(n_atoms)<<" atoms : " << box_
130   << "\n";
131
132 std::vector<double> temperatures;
133 for (int i = 6; i < argc; ++i) temperatures.push_back(atof(argv[i]));
134
135 AtomTypingInterface_SP ar_type = std::make_shared<SingleAtomType>("AR");
136 std::vector<std::shared_ptr<CartesianAtomsSimple<Vec3Cubic>>> systems;
137 core::data::structural::PdbAtom_SP ar_atom = std::make_shared
138   <core::data::structural::PdbAtom>(1, "AR");
139
140 std::shared_ptr<AbstractPdbFormatter> fmt = std::make_shared<SimplePdbFormatter>("_
141   AR ", "AR ", "AR");
142
143 std::vector<simulations::sampling::IsothermalMC_SP> replica_samplers;
144 std::vector<CalculateEnergyBase_SP> energies;
145 std::vector<simulations::observers::ObserverInterface_SP> mover_adjusters;
146 for (core::index2 irepl = 0; irepl < temperatures.size(); ++irepl) {
147
148   // ----- Create the systems to be sampled -----
149   CartesianAtoms ar(ar_type, n_atoms);
150   systems.push_back(ar);
151
152   // ----- Distribute atoms in the box or use coordinates from provided PDB_
153   file
154   if(input_from_file) {
155     const auto strctr = argon_structures[irepl % argon_structures.size()];
156     core::index2 ia=0;
157     for(auto a_it = strctr->first_const_atom(); a_it != strctr->last_const_atom(); ++a_
158       it) {
159       (*ar)[ia].set(**a_it);
160       ++ia;
161     }
162   }
163   else
164     BuildFluidSystem<Vec3Cubic>::generate(*ar, *ar_atom, n_atoms);
165
166   // ----- Create energy function - just LJ potential
167   std::shared_ptr<NeighborList_OBSOLETE<Vec3Cubic>> nbl = std::make_shared
168   <NeighborList_OBSOLETE<Vec3Cubic>>(*ar, 9.0, 3.0);
169   std::shared_ptr<LjEnergy> lj_energy_term = std::make_shared<LjEnergy>(*ar, *nbl,
170   SIGMA, EPSILON_BY_K);
171   std::shared_ptr<TotalEnergy_OBSOLETE<ByAtomEnergy_OBSOLETE>> lj_energy =
172   std::make_shared<TotalEnergy_OBSOLETE<ByAtomEnergy_OBSOLETE>>();
```

(continues on next page)

(continued from previous page)

```

165     lj_energy->add_component(lj_energy_term, 1.0);
166     energies.push_back(std::static_pointer_cast<class CalculateEnergyBase>(lj_
167     ↪energy));
168
169     // --- Create a mover, which is a random perturbation of an atom in this case, ↪
170     // and place it in a movers' set
171     std::shared_ptr<TranslateAtom<Vec3Cubic>> translate = std::make_shared
172     ↪<TranslateAtom<Vec3Cubic>>(*ar, *lj_energy_term);
173     MoversSet_SP movers = std::make_shared<simulations::movers::MoversSetSweep>();
174     movers->add_mover(translate, n_atoms);
175     translate->max_move_range(0.5); // --- set the maximum distance a single atom can ↪
176     ↪be moved by a single MC perturbation
177
178     // --- Create an observer to record and adjust movers acceptance rate
179     simulations::observers::AdjustMoversAcceptance_SP adj = std::make_shared
180     ↪<simulations::observers::AdjustMoversAcceptance>(
181         *movers, utils::string_format("movers-%d.dat", irepl), 0.4);
182     mover_adjusters.push_back(adj);
183     adj->observe_header();
184
185     // ----- create an isothermal Monte Carlo sampler
186     auto mc = std::make_shared<simulations::sampling::IsothermalMC>(movers,
187     ↪temperatures[irepl]);
188
189     // ----- Create an observer for energy components
190     EnergyObserverType_SP obs_en
191     = std::make_shared<EnergyObserverType>(*lj_energy, utils::string_format("energy-
192     ↪%d.dat", irepl));
193     obs_en->observe_header();
194
195     // ----- Create an observer for trajectory in PDB format
196     auto observe_trajectory = std::make_shared
197     ↪<simulations::observers::cartesian::PdbObserver_OBSOLETE<Vec3Cubic>>(
198         *ar, fmt, utils::string_format("ar_tra-%d.pdb", irepl));
199
200     observe_trajectory->trigger(std::make_shared
201     ↪<simulations::observers::TriggerEveryN>(10));
202     mc->outer_cycle_observer(observe_trajectory);
203     mc->outer_cycle_observer(obs_en);
204     mc->cycles(n_inner_cycles, n_outer_cycles, 1);
205     replica_samplers.push_back(mc);
206 }
207
208 bool replica_isothermal_observation_mode = true;
209 simulations::sampling::ReplicaExchangeMC remc(replica_samplers, energies, replica_
210 ↪isothermal_observation_mode);
211 auto remc_flow = std::make_shared<simulations::observers::ObserveReplicaFlow>(remc,
212 ↪"replica_flow.dat");
213 remc.exchange_observer(remc_flow);
214 auto tag_updater = std::make_shared<simulations::observers::UpdateSystemTags
215 ↪<CartesianAtomsSimple<Vec3Cubic>>>(systems, remc);
216 tag_updater->observe();
217 remc.exchange_observer(tag_updater);
218
219 remc.replica_exchanges(n_exchanges);
220 for (auto o : mover_adjusters) remc.exchange_observer(o);
221 remc.run();

```

(continues on next page)

(continued from previous page)

```
210
211     simulations::observers::cartesian::PdbObserver_OBSOLETE<Vec3Cubic>_
212     ↪final(*systems[0], fmt, "final.pdb");
213     final.observe();
214     for (core::index2 i_system = 1; i_system < systems.size(); ++i_system) {
215         for (core::index4 i_atom = 0; i_atom < systems[0]->n_atoms; ++i_atom)
216             systems[0]->operator[](i_atom) = systems[i_system]->operator[](i_atom);
217         final.observe();
218     }
219     final.finalize();
}
```



ex_ReduceSequenceAlphabet

If no input is given, ex_ReduceSequenceAlphabet lists all reduced amino acid alphabets registered in BioShell library. Alternatively, user can provide an alphabet name; in this case the relevant mapping is printed on the screen.

USAGE:

```
ex_ReduceSequenceAlphabet [alphabet_name]
```

EXAMPLEs:

```
ex_ReduceSequenceAlphabet
ex_ReduceSequenceAlphabet lz-mj.16
```

Keywords:

- reduced alphabet

Categories:

- core:::data::sequence::ReduceSequenceAlphabet

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <iomanip>
3
4 #include <core/data/sequence/ReduceSequenceAlphabet.hh>
5 #include <utils/options/OptionParser.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(

9
10 If no input is given, ex_ReduceSequenceAlphabet lists all reduced amino acid_
11 ↴alphabets registered in BioShell library.
12 Alternatively, user can provide an alphabet name; in this case the relevant mapping_
13 ↴is printed on the screen.

14 USAGE:
15     ex_ReduceSequenceAlphabet [alphabet_name]

16 EXAMPLES:
17     ex_ReduceSequenceAlphabet
18     ex_ReduceSequenceAlphabet lz-mj.16

19 )
20 /**
21 ** @brief ex_ReduceSequenceAlphabet lists all reduced amino acid alphabets_
22 ↴registered in BioShell library

```

(continues on next page)

(continued from previous page)

```
23
24 * CATEGORIES: core::data::sequence::ReduceSequenceAlphabet
25 * KEYWORDS: reduced alphabet
26 */
27 int main(const int argc, const char* argv[]) {
28
29     if ((argc > 1) && utils::options::call_for_help(argv[1]))
30         utils::exit_OK_with_message(program_info);
31
32     using core::data::sequence::ReduceSequenceAlphabet;
33
34     int i = 1;
35     std::cout << "Known alphabets:\n";
36     for (auto it = ReduceSequenceAlphabet::cbegin(); it !=_
37             ReduceSequenceAlphabet::cend(); ++it) {
38         std::cout << std::setw(8) << it->first << ((i % 10 == 0) ? "\n" : " ");
39         ++i;
40     }
41     std::cout << "\n";
42     for (int i = 1; i < argc; ++i) {
43         std::cout << "Listing alphabet " << argv[i] << ":\n";
44         core::data::sequence::ReduceSequenceAlphabet_SP alph =_
45             ReduceSequenceAlphabet::get_alphabet(argv[i]);
46         std::cout << (*alph) << "\n";
47     }
48 }
```



ex_Residue

Simple example reads a PDB file and checks if all amino acid residues have complete backbone.

EXAMPLE:

```
ex_Residue 5edw.pdb
```

Keywords:

- *PDB input*
- *pre-processing*

Categories:

- core::data::structural::Structure; core::data::structural::Residue

Input files:

- 2gb1.pdb
- 5edw.pdb
- 3dcg.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <iomanip>
3 #include <core/data/io/Pdb.hh>
4 #include <utils/exit.hh>
5
6 std::string program_info = R"(
7
8 Simple example reads a PDB file and checks if all amino acid residues have complete_
9 backbone.
10 EXAMPLE:
11     ex_Residue 5edw.pdb
12 )";
13
14 /**
15  * @brief Reads a PDB file and checks if all amino acid residues have complete_
16  * backbone
17  */
18
19 * CATEGORIES: core::data::structural::Structure; core::data::structural::Residue
20 * KEYWORDS: PDB input; pre-processing
21 */
22 int main(const int argc, const char* argv[]) {
```

(continues on next page)

(continued from previous page)

```

20
21     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
↪missing program parameter
22
23     core::data::io::Pdb reader(argv[1]); // file name (PDB format, may be gzip-ped)
24     core::data::structural::Structure_SP strctr = reader.create_structure(0);
25
26     // Iterate over all residues in the structure
27     bool is_OK = true;
28     for (auto it_resid = strctr->first_residue(); it_resid!=strctr->last_residue();↪
29         ++it_resid) {
30         const core::chemical::Monomer & m = (*it_resid)->residue_type();
31         core::data::structural::PdbAtom_SP atom_sp;
32         if(m.type=='P') {
33             atom_sp = (*it_resid)->find_atom(" N ");
34             if(atom_sp==nullptr) { std::cout << "Missing backbone atom N \n"; is_OK =↪
35             false; }
36             atom_sp = (*it_resid)->find_atom(" CA ");
37             if(atom_sp==nullptr) { std::cout << "Missing backbone atom CA \n"; is_OK =↪
38             false; }
39             atom_sp = (*it_resid)->find_atom(" C ");
40             if(atom_sp==nullptr) { std::cout << "Missing backbone atom C \n"; is_OK =↪
41             false; }
42             atom_sp = (*it_resid)->find_atom(" O ");
43             if(atom_sp==nullptr) { std::cout << "Missing backbone atom O \n"; is_OK =↪
44             false; }
45         }
46         if(is_OK) std::cout << "Backbone complete!\n";
47     }

```



ex_RobustDistributionDecorator

Example showing how to create and use a RobustDistributionDecorator, which facilitates distribution estimation of any probability distribution function that is defined in BioShell. This example estimates parameters of a normal distribution from a noised data using regular and robust methods.

USAGE:

```
./ex_RobustDistributionDecorator [data.txt]
```

REFERENCE: Kim Seong-Ju "The Metrically Trimmed Mean as a Robust Estimator of Location", The Annals of Statistics (1992) 20 1534-1547

Keywords:

- *estimation*

Categories:

- core::calc::statistics::NormalDistribution; core::calc::statistics::RobustDistributionDecorator

Output files:

- stdout.out

Program source:

```

1 #include <math.h>
2
3 #include <iostream>
4 #include <random>
5 #include <vector>
6
7 #include <core/calc/statistics/NormalDistribution.hh>
8 #include <core/calc/statistics/RobustDistributionDecorator.hh>
9 #include <utils/options/OptionParser.hh>
10
11 std::string program_info = R"(

12 Example showing how to create and use a RobustDistributionDecorator, which
13   ↪ facilitates distribution estimation
14 of any probability distribution function that is defined in BioShell.

15 This example estimates parameters of a normal distribution from a noised data using
16   ↪ regular and robust methods.

17
18 USAGE:
19 ./ex_RobustDistributionDecorator [data.txt]
20
21 REFERENCE:
22 Kim Seong-Ju "The Metrically Trimmed Mean as a Robust Estimator of Location",
23 The Annals of Statistics (1992) 20 1534-1547
24 )";

```

(continues on next page)

(continued from previous page)

```

25
26  /** @brief Example showing how to create and use a RobustDistributionDecorator
27  *
28  * CATEGORIES: core::calc::statistics::NormalDistribution; ↵
29  *               core::calc::statistics::RobustDistributionDecorator
30  * KEYWORDS: estimation
31  */
32
33  int main(const int argc, const char* argv[]) {
34
35    if ((argc > 1) && utils::options::call_for_help(argv[1]))
36      utils::exit_OK_with_message(program_info);
37
38    using namespace core::calc::statistics;
39
40    unsigned int rd = 9876543;
41    std::mt19937 gen(rd);
42    core::index4 N = 10000; //--- the number of random points to use in tests
43    core::index4 Nnoise = 10; //--- the number of random points from the noise ↵
44    distribution
45
46    // ----- The two distributions used in this test
47    std::normal_distribution<> base(2.0, 0.5); // --- the "base" distribution
48    std::normal_distribution<> noise(2.0, 50); // --- the "noise" distribution
49
50    // -----
51    std::vector<std::vector<double> > random_points;
52
53    if (argc==1) { // ----- Generate random sample
54      for (core::index4 i = 0; i < N; ++i)
55        random_points.emplace_back( std::initializer_list<double>{base(gen)} );
56      for (core::index4 i = 0; i < Nnoise; ++i)
57        random_points.emplace_back( std::initializer_list<double>{noise(gen)} );
58    } else { // ----- Read data from a file
59      std::ifstream infile(argv[1], std::ios_base::in);
60      double a;
61      while (infile >> a) {
62        random_points.emplace_back(std:: initializer_list<double>{a});
63      }
64    }
65
66    std::vector<double> init_params{1.0,0.0}; // --- initial parameters of the ↵
67    estimated distribution
68    NormalDistribution n(init_params);
69    n.estimate(random_points);
70    std::cout << "Estimated: " << n << "\n";
71    RobustDistributionDecorator<NormalDistribution> rn(init_params, 0.05);
72    rn.estimate(random_points);
73    std::cout << "Estimated (robust): " << rn << "\n";
74    if (argc==1) std::cout << "True distribution: 2.0 0.5\n";
75  }

```



ex_SelectChainBreaks

Reads a PDB file and prints list of chain breaks found in every chain

EXAMPLE:

```
ex_SelectChainBreaks 4mcb.pdb
```

Keywords:

- *structure selectors*
- *PDB input*

Categories:

- core::data::structural::SelectChainBreaks

Input files:

- 2gb1.pdb
- 4mcb.pdb
- 5edw.pdb
- 2fdo.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <core/data/io/Pdb.hh>
3 #include <core/data/structural/selectors>SelectChainBreaks.hh>
4 #include <utils/exit.hh>
5 #include <utils/LogManager.hh>
6
7
8 std::string program_info = R"(
9
10 Reads a PDB file and prints list of chain breaks found in every chain
11
12 EXAMPLE:
13   ex_SelectChainBreaks 4mcb.pdb
14
15 )";
16
17 /** @brief Reads a PDB file and prints a list of chain breaks found in every chain
18 *
19 *  CATEGORIES: core::data::structural::SelectChainBreaks
```

(continues on next page)

(continued from previous page)

```

20 * KEYWORDS: structure selectors; PDB input
21 */
22 int main(const int argc, const char *argv[]) {
23
24     using namespace core::data::structural::selectors;
25     utils::LogManager::INFO();
26
27
28     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
29     ↪missing program parameter
30
31     // ----- Read a PDB file and create a Structure object
32     core::data::io::PdbLineFilter filt1 = core::data::io::all_true(core::data::io::is_
33     ↪ca, core::data::io::is_standard_atom,core::data::io::is_not_alternative);
34     core::data::io::PdbLineFilter filt2 = core::data::io::all_true(core::data::io::is_
35     ↪ca, core::data::io::is_hetero_atom,core::data::io::is_not_alternative);
36     core::data::io::PdbLineFilter filt3 = core::data::io::one_true(filt1, filt2);
37
38
39     core::data::io::Pdb reader(argv[1],filt3);
40     auto strctr = reader.create_structure(0);
41
42     SelectChainBreaks sel;
43     bool chain_is_OK = true;
44     for (const auto &chain : *strctr) {
45         if (std::distance(chain->begin(),chain->terminal_residue()) < 3) {
46             std::cerr << "Chain " << chain->id() << " of " << strctr->code() << " is too_
47             ↪short\n";
48             continue;
49         }
50         if (chain->count_aa_residues() < chain->size() * 0.8) {
51             std::cerr << "Chain " << chain->id() << " of " << strctr->code() << " is not a_
52             ↪protein\n";
53             continue;
54         }
55         std::cerr << "# Processing " << strctr->code() << " chain " << chain->id() << "\n"
56         ↪";
57         size_t first_aa = 0;
58         while ((*chain)[first_aa]->residue_type().type != 'P') ++first_aa;
59         const auto last_it = chain->terminal_residue() - 1;
60         for (auto res_it = chain->begin() + first_aa + 1; res_it != last_it; ++res_it) {
61             auto next = (**res_it).next();
62             if(next== nullptr) {
63                 next = *(+std::find(chain->begin(),chain->end(),*res_it));
64                 std::cerr << "Residue following "<<(**res_it)<<" is not an amino acid!\n";
65             }
66             auto prev = (**res_it).previous();
67             if(prev == nullptr) {
68                 prev = *(-std::find(chain->begin(),chain->end(),*res_it));
69                 std::cerr << "Residue preceding "<<(**res_it)<<" is not an amino acid!\n";
70             }
71
72             if (sel(*res_it)) {
73                 chain_is_OK = false;
74                 if (sel.last_chainbreak_type == RIGHT) {
75                     std::cout << utils::string_format("%s %4s %4d%c %4d%c %6.2f\n",
76                     ↪strctr->code().c_str(), chain->id().c_str(),

```

(continues on next page)

(continued from previous page)

```
70     (*res_it)->id(), (*res_it)->icode(), next->id(), next->icode(), sel.
71     ↪right_side_distance);
72     ++res_it;
73     if (res_it == last_it) break;
74   }
75   if (sel.last_chainbreak_type == LEFT) {
76     std::cout << utils::string_format("%s %4s %4d%c %4d %c %6.2f\n", strctr->
77     ↪code().c_str(), chain->id().c_str(),
78     prev->id(), prev->icode(), (*res_it)->id(), (*res_it)->icode(), sel.
79     ↪left_side_distance);
80   }
81   if (sel.last_chainbreak_type == BOTH) {
82     std::cout << utils::string_format("%s %4s %4d%c %4d %c %6.2f %6.2f\n", ↪
83     strctr->code().c_str(), chain->id().c_str(),
84     prev->id(), prev->icode(), (*res_it)->id(), (*res_it)->icode(), sel.
85     ↪left_side_distance);
86     std::cout << utils::string_format("%s %4s %4d%c %4d %c\n", strctr->code().c_
87     ↪str(), chain->id().c_str(),
88     (*res_it)->id(), (*res_it)->icode(), next->id(), next->icode(), sel.
89     ↪right_side_distance);
90     ++res_it;
91     if (res_it == last_it) break;
92   }
93 }
94 if(chain_is_OK) std::cout << utils::string_format("%s %4s OK\n", strctr->code().c_
95 str(), chain->id().c_str());
96 }
```



ex_SelectResidueRange

Simple example showing how to select a structural fragment based on residue IDs

USAGE:

```
./ex_SelectResidueRange
```

Keywords:

- *structure selectors*
- *PDB input*
- *STL*
- *algorithms*

Categories:

- core:::data::structural::SelectResidueRange

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <sstream>
3 #include <core/data/io/Pdb.hh>
4 #include <core/data/structural/selectors/structure_selectors.hh>
5 #include <utils/options/OptionParser.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Simple example showing how to select a structural fragment based on residue IDs
11
12 USAGE:
13 ./ex_SelectResidueRange
14
15 )";
16
17 /** @brief Shows how to select a structural fragment based on residue IDs
18 *
19 * CATEGORIES: core:::data::structural::SelectResidueRange
20 * KEYWORDS: structure selectors; PDB input; STL; algorithms
21 */
22 // --- Only C-alpha atoms are listed here to keep this example short and simple
23 std::string fragment =
24     R"(ATOM    312  CA  ALA A   -1      -10.035   4.811   1.920   1.00   0.24
25     ↪
26 ATOM    322  CA  VAL A    0       -13.437   5.248   0.258   1.00   0.33
27
28 )";
```

(continues on next page)

(continued from previous page)

```

26 ATOM    338  CA   ASP A   1    -12.201   3.975  -3.121  1.00  0.24      C
27 ATOM    350  CA   ALA A   1A   -9.237   2.226  -4.777  1.00  0.18      C
28 ATOM    360  CA   ALA A   1B   -7.956   5.461  -6.338  1.00  0.24      C
29 ATOM    370  CA   THR A   2    -7.460   7.449  -3.135  1.00  0.21      C
30 ATOM    384  CA   ALA A   3    -6.080   4.229  -1.648  1.00  0.12      C
31 ) "
32 ;
33
34 int main(const int argc, const char* argv[]) {
35
36     if ((argc > 1) && utils::options::call_for_help(argv[1]))
37         utils::exit_OK_with_message(program_info);
38
39     using namespace core::data::structural;
40
41     std::stringstream in(fragment);                                // Create an input stream that
42     // will provide data from a string
43     core::data::io::Pdb reader(in);
44     Structure_SP strctr = reader.create_structure(0);
45
46     std::cout << "IDs of the residues available for selection:";
47     std::for_each(strctr->first_residue(), strctr->last_residue(), [] (Residue_SP r)
48     // {std::cout << r->residue_id() << " ";});
49     std::cout << "\n";
50
51     selectors::SelectResidueRange range0("-1-1");
52     std::cout << "selector " << range0.selector_string() << " selects: "
53     // << std::count_if(strctr->first_residue(), strctr->last_residue(), range0) << " "
54     // residues\n";
55
56     selectors::SelectResidueRange range1("-1-1A");
57     std::cout << "selector " << range1.selector_string() << " selects: "
58     // << std::count_if(strctr->first_residue(), strctr->last_residue(), range1) << " "
59     // residues\n";
60
61     selectors::SelectResidueRange range2("*");
62     std::cout << "selector " << range2.selector_string() << " selects: "
63     // << std::count_if(strctr->first_residue(), strctr->last_residue(), range2) << " "
64     // residues\n";
65 }
```



ex_SemiglobalAligner

Example that calculates semiglobal alignment i.e. the optimal global alignment where trailing gaps are not penalized. The program also shows how one can define its own scoring function to calculate an alignment

USAGE:

```
./ex_PairwiseAlignment
```

Keywords:

- *sequence alignment*

Categories:

- core::alignment::SemiglobalAligner; core::alignment::PairwiseSequenceAlignment

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <chrono>
3 #include <algorithm>
4
5 #include <core/data/io/fasta_io.hh>
6
7 #include <core/data/sequence/Sequence.hh>
8 #include <core/alignment/SemiglobalAligner.hh>
9 #include <core/alignment/PairwiseAlignment.hh>
10 #include <core/alignment/PairwiseSequenceAlignment.hh>
11 #include <core/alignment/scoring/SimilarityMatrix.hh>
12 #include <core/alignment/scoring/SimilarityMatrixScore.hh>
13 #include <utils/options/OptionParser.hh>
14
15 std::string program_info = R"(
16
17 Example that calculates semiglobal alignment i.e. the optimal global alignment where_trailing
18 gaps are not penalized. The program also shows how one can define its own scoring_function
19 to calculate an alignment
20
21 USAGE:
22 ./ex_PairwiseAlignment
23
24 )";
25
26 using namespace core::data::sequence;
27 using namespace core::alignment::scoring;
28

```

(continues on next page)

(continued from previous page)

```

29 /// An example score function used by BioShell pairwise sequence alignment methods.
30 /** Such a scoring object must provide three components:
31 * - a scoring operator, whose arguments are positions in the scored sequences,
32 *   (query and template, respectively)
33 * - query_length() method, and
34 * - tmplt_length() method
35 */
36 struct IdentityScore {
37     IdentityScore(const Sequence & query, const Sequence & tmplt) : q(query.sequence), t(tmplt.sequence) {}
38
39     /// Alignment score is 1 when the two compared letters are identical and 0 otherwise
40     short operator()(const core::index2 i, const core::index2 j) const { return
41         q[i]==t[j]; }
42     /// Returns the length of a template sequence
43     core::index2 tmplt_length() const { return t.length(); }
44     /// Returns the length of a query sequence
45     core::index2 query_length() const { return q.length(); }
46
47     const std::string & q;
48     const std::string & t;
49 };
50
51 /** @brief Calculate a pairwise sequence alignment between two sequences with
52 * identity scoring method.
53 *
54 * The program calculates semiglobal alignment i.e. the optimal global alignment,
55 * where trailing gaps are not penalized.
56 * The program also shows how one can define its own scoring function to calculate
57 * an alignment
58 */
59 * CATEGORIES: core::alignment::SemiglobalAligner;
60 * KEYWORDS: sequence alignment
61 */
62 int main(const int argc, const char* argv[]) {
63
64     if ((argc > 1) && utils::options::call_for_help(argv[1]))
65         utils::exit_OK_with_message(program_info);
66
67     Sequence_SP query = std::make_shared<Sequence>("query", "CATACTCGACGGCT", 1);
68     Sequence_SP tmplt = std::make_shared<Sequence>("tmplt", "ACGACGT", 1);
69
70     // --- create aligner object
71     core::index2 max_len = std::max(query->length(), tmplt->length());
72     core::alignment::SemiglobalAligner<short, IdentityScore> aligner(max_len);
73
74     // --- find score of the alignment; just the score - this is faster than aligning
75     // and keeping backtracking info
76     IdentityScore s(*query, *tmplt);
77     short result1 = aligner.align_for_score(-10, -1, s);
78
79     short result2 = aligner.align(-10, -1, s);
80     core::alignment::PairwiseAlignment_SP ali = aligner.backtrace();
81     std::cerr << ali->query_length() << " " << query->sequence << " " << ali->template_
82     length() << " " << tmplt->sequence
83     << "\n";

```

(continues on next page)

(continued from previous page)

```
77 core:::alignment::PairwiseSequenceAlignment seq_ali(ali,query,tmplt);
78 IdentityScore s2(*tmplt, *query);
79 short result3 = aligner.align(-10, -1, s2);
80 std::cout << "The three scores below should be identical:\n" << result1 << " "
81     << result2 << " " << result3 << "\n"
82         << seq_ali << "\n";
}
```



ex_Seqres

Reads a PDB file and prints the sequences stored in its SEQRES fields These sequences in many cases differ from the sequences extracted from coordinates section

EXAMPLE:

```
./ex_Seqres 2kwi.pdb
```

Keywords:

- *PDB input*
- *Structure*
- *sequence*

Categories:

- core::data::io::Pdb

Input files:

- 2kwi.pdb
- 4mcb.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <core/data/io/fasta_io.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Reads a PDB file and prints the sequences stored in its SEQRES fields
10 These sequences in many cases differ from the sequences extracted from coordinates_
11    ↪section
12
13 EXAMPLE:
14     ./ex_Seqres 2kwi.pdb
15
16 )";
17
18 /**
19 * @brief Reads a PDB file and extracts its SEQRES sequence(s)
20 * @CATEGORIES: core::data::io::Pdb
21 * @KEYWORDS: PDB input; Structure; sequence

```

(continues on next page)

(continued from previous page)

```
20  */
21 int main(const int argc, const char* argv[]) {
22
23     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
24     ↵missing program parameter
25
26     using namespace core::data::io;
27
28     Pdb reader(argv[1], // file name (PDB format, may be gzip-ped)
29     is_ca,           // read only CA atoms
30     keep_all, true);          // parse PDB header !
31
32     std::shared_ptr<Seqres> seq_res = std::static_pointer_cast<Seqres>(reader.header.
33     ↵find("SEQRES")->second);
34     for(const auto & chain_seq : seq_res->sequences) {
35         const std::string header = reader.pdb_code() + " :" + chain_seq.first;
36         core::data::sequence::Sequence_SP s = seq_res->create_sequence(chain_seq.first,
37         ↵header);
38         if((s->length()>20) && (s->get_monomer(2).type=='P'))
39             std::cout << core::data::io::create_fasta_string(*s,-1)<<"\n";
40     }
41 }
```



ex_Sequence

Unit test which reads a PDB file and prints a requested sequence fragment.

USAGE:

```
./ex_Sequence input.pdb chain from to
```

EXAMPLE

```
./ex_Sequence 3wn7.pdb A 366 405
```

Keywords:

- *PDB input*
- *sequence*

Categories:

- core/data/io/Sequence

Input files:

- 3wn7.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <core/data/io/fasta_io.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(

8 Unit test which reads a PDB file and prints a requested sequence fragment.
9 USAGE:
10     ./ex_Sequence input.pdb chain from to
11 EXAMPLE
12     ./ex_Sequence 3wn7.pdb A 366 405
13 )
14
15 /**
16  * @brief Reads a PDB file and prints a fragment of its sequence.
17  *
18  * @CATEGORIES: core/data/io/Sequence
19  * @KEYWORDS: PDB input; sequence
20 )";
```

(continues on next page)

(continued from previous page)

```

21  */
22 int main(const int argc, const char* argv[]) {
23
24     if(argc < 4) utils::exit_OK_with_message(program_info); // --- complain about
25     ↪missing program parameter
26
27     using namespace core::data::io;
28
29     // --- Try this test program with 3wn7 as the input structure !
30     Pdb reader(argv[1], // file name (PDB format, may be gzip-ped)
31     all_true(is_not_hydrogen,is_not_water),           // don't read hydrogens, skip
32     ↪water molecules
33     core::data::io::keep_all, true);                  // parse PDB header !
34
35     core::index2 from = atoi(argv[3]);
36     core::index2 to = atoi(argv[4]);
37     auto structure = reader.create_structure(0);
38     auto chain = structure->get_chain(argv[2][0]);
39     auto sequence = chain->create_sequence();
40
41     // --- Should be 324 (324 is the ID of the very first residue of 3wn7 chain A)
42     std::cout << "Id of the first residue: " << sequence->first_pos() << "\n";
43     Sequence fragment(*sequence, from, to); // Cut from residue whose ID is 366
44     std::cout << fragment.first_pos() << " " << fragment.sequence << "\n";
45 }
```



ex_SimulatedAnnealing

A simple example shows how to use Monte Carlo simulated annealing.

Keywords:

- *Monte Carlo*
- *Mover*
- *sampling*
- simulated annealing
- evaluators

Categories:

- simulations/generic/evaluators/EchoEvaluator; simulations/generic/evaluators/CallEvaluator

Output files:

- stdout.out

Program source:

```

1 #include <memory>
2 #include <iostream>
3 #include <random>
4
5 #include <core/calc/statistics/Random.hh>
6
7 #include <simulations/movers/Mover.hh>
8 #include <simulations/movers/MoversSetSweep.hh>
9 #include <simulations/evaluators/EchoEvaluator.hh>
10 #include <simulations/evaluators/CallEvaluator.hh>
11 #include <simulations/sampling/SimulatedAnnealing.hh>
12
13 #include <simulations/observers/ObserveEvaluators.hh>
14
15 /** @brief Models a point particle in 1D space with harmonic energy.
16 * This simple class implements the Mover interface. One has to implement just two
17 * essential methods: <code>move()</code> and <code>undo()</code>. For simplicity, in
18 * this example energy calculation
19 * is implemented within the <code>move()</code> method.
20 */
21 class HarmonicSystemMover : public simulations::movers::Mover {
22 public:
23     const double x0;           ///< Initial position of the particle, where energy is 0.0
24     double x;                  ///< Actual position of the particle at the end of the
25     spring
26     double recent_energy;     ///< Actual energy of the spring
27
28     HarmonicSystemMover() : simulations::movers::Mover("HarmonicSystemMover"), x0(0.0)
29 }
```

(continues on next page)

(continued from previous page)

```

27  /** @brief Moves the particle randomly in either direction.
28   * This method is declared abstract in Mover class and must be implemented here
29   */
30
31 virtual bool move(simulations::sampling::AbstractAcceptanceCriterion &mc_scheme) {
32
33     double old_en = (x - x0) * (x - x0);
34     double delta_x = 0.1 - 0.2 * rand_coordinate(generator);
35     x += delta_x;
36     double new_en = (x - x0) * (x - x0);
37     inc_move_counter();
38
39     if (!mc_scheme.test(old_en, new_en)) {
40         undo();
41         recent_energy = old_en;
42         return false;
43     } else {
44         recent_energy = new_en;
45         return true;
46     }
47 }
48
49 /** @brief Back up the most recent move.
50  * This method is declared abstract in Mover class and must be implemented here
51  */
52 inline void undo() {
53     dec_move_counter();
54     x -= delta_x;
55 }
56
57 /// Yet another method inherited from the base class
58 virtual const std::string &name() const { return name_; }
59
60 /// Does nothing, but must be implemented since it's been declared as virtual in_
61 → the base class
62 void max_move_range(const double max_range) {}
63
64 /// Reads the maximum range for a move
65 virtual double max_move_range() const { return 0.1; }
66
67 private:
68     double delta_x;
69     std::uniform_real_distribution<double> rand_coordinate;
70     core::calc::statistics::Random &generator = core::calc::statistics::Random::get();
71     static std::string name_;
72 };
73
74 std::string HarmonicSystemMover::name_ = "HarmonicSystemMover";
75
76 /** @brief A simple example shows how to use Monte Carlo simulated annealing.
77  *
78  * This demo also shows how to implement a simple mover and how to hook it up to_
79  → sampling protocol.
80  * CATEGORIES: simulations/generic/movers/Mover; simulations/generic/sampling/
81  → SimulatedAnnealing; simulations/generic/evaluators/EchoEvaluator;
82  * CATEGORIES: simulations/generic/evaluators/EchoEvaluator; simulations/generic/
83  → evaluators/CallEvaluator

```

(continues on next page)

(continued from previous page)

```

80 * KEYWORDS: Monte Carlo; Mover; sampling; simulated annealing; evaluators
81 */
82 int main(const int argc, const char *argv[]) {
83
84     using namespace simulations::evaluators;
85
86     // ----- Here we create a system to me modelled
87     std::shared_ptr<HarmonicSystemMover> harmonic_ptr = std::make_shared
88     <HarmonicSystemMover>();
89     // --- You need a mover set to use SimulatedAnnealing protocol, even if the set_
89     <contains just one mover
90     simulations::movers::MoversSet_SP moves = std::make_shared
91     <simulations::movers::MoversSetSweep>();
92     moves->add_mover(harmonic_ptr,1);
93
94     simulations::sampling::SimulatedAnnealing sa(moves,{2.0,1.5,1.0});
95
96     // ----- Create an observer which calls evaluators and writes the observations_
96     <on the screen
97     std::shared_ptr<simulations::observers::ObserveEvaluators> obs = std::make_shared
97     <simulations::observers::ObserveEvaluators>("");
98     // --- Create an evaluator which just return the X position of the particle
99     // --- Add the evaluator to the observer. Obviously there might be may evaluators_
99     <added to this observer;
100    // --- then a nice table will be printed with a column correspoding to an_
100    <evaluator.
101    obs->add_evaluator(std::make_shared<EchoEvaluator<double>>(harmonic_ptr->x,
101    <"position X">));
102    obs->add_evaluator(std::make_shared<EchoEvaluator<double>>(harmonic_ptr->recent_
102    <energy,"energy"));
103
103    std::function<double(void)> get_temperature = [&sa]() { return sa.temperature(); };
104    obs->add_evaluator(
104        std::make_shared<CallEvaluator<std::function<double(void)>>>(get_temperature,
104        <"temperature"));
105
105    sa.outer_cycle_observer(obs);
106    sa.cycles(100,100);
107    sa.run();
108}
109

```



ex_Structure

ex_Structure reads a PDB file and prints a list of all atoms grouped by residues they belong to

EXAMPLE:

```
./ex_Structure 5edw.pdb
```

Keywords:

- *PDB input*
- *Structure*
- *Chain*
- Residue
- PdbAtom
- *STL*

Categories:

- core::data::structural::Structure

Input files:

- 2gb1.pdb
- 5edw.pdb
- 3dcg.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <iomanip>
3 #include <core/algorithms/predicates.hh>
4 #include <core/data/io/Pdb.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 ex_Structure reads a PDB file and prints a list of all atoms grouped by residues they_
10 ↴belong to
11 EXAMPLE:
12     ./ex_Structure 5edw.pdb
13 )";

```

(continues on next page)

(continued from previous page)

```

14
15  /** @brief Reads a PDB file and prints a list of all atoms grouped by residues they
16   * belong to.
17   *
18   * CATEGORIES: core::data::structural::Structure
19   * KEYWORDS: PDB input; Structure; Chain; Residue; PdbAtom; STL
20   */
21
22  int main(const int argc, const char* argv[]) {
23
24
25      if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
26      //missing program parameter
27      using namespace core::data::io;
28
29      core::data::io::Pdb reader(argv[1],is_not_alternative); // file name (PDB format,
30      //may be gzip-ped)
31      core::data::structural::Structure_SP strctr = reader.create_structure(0);
32
33      // Iterate over all chains
34      for (auto it_chain = strctr->begin(); it_chain!=strctr->end(); ++it_chain) {
35          std::cout << "----- chain " << (*it_chain)->id() << " -----\\n";
36          // Iterate over all residues
37          for (auto it_res = (*it_chain)->begin(); it_res!=(*it_chain)->end(); ++it_res) {
38              std::cout << std::setw(5)<<(*it_res)->id()<<" "<<(*it_res)->residue_type().
39              code3<<" :";
40              for (auto it_atom = (*it_res)->begin(); it_atom!=(*it_res)->end(); ++it_atom)
41              {
42                  if (((*it_atom)->alt_locator() == ' ') || ((*it_atom)->alt_locator() == 'A'))
43                      std::cout << " " << (*it_atom)->atom_name();
44              }
45              std::cout <<"\\n";
46          }
47      }
48  }

```



ex_ThreadPool

Unit test which shows how to use a ThreadPool class.

USAGE:

```
./ex_ThreadPool
```

Keywords:

- concurrency
- multi-threading

Categories:

- utils/ThreadPool

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <utils/ThreadPool.hh>
4 #include <utils/string_utils.hh>
5 #include <utils/options/OptionParser.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Unit test which shows how to use a ThreadPool class.
11
12 USAGE:
13 ./ex_ThreadPool
14
15 )";
16
17 /// Operator called by each thread (pretends to run very time consuming calculations)
18 struct Op { std::string operator()(int i) { return "Hello " + utils::to_string(i); } }
19
20 /** @brief Simple test for a ThreadPool class
21 *
22 * CATEGORIES: utils/ThreadPool
23 * KEYWORDS: concurrency; multi-threading
24 */
25 int main(const int cnt, const char *argv[]) {
26
27     if ((cnt > 1) && utils::options::call_for_help(argv[1]))
28         utils::exit_OK_with_message(program_info);

```

(continues on next page)

(continued from previous page)

```
29
30     utils::ThreadPool pool(4); // --- Create a pool of four threads = four jobs can be_
→executed at a time
31
32     typedef typename std::result_of<Op(int)>::type return_type;
33     Op o;
34     std::vector<std::future<return_type>> futures;
35
36     // --- Here we start 10 jobs which will be executed by four workers (threads)
37     for (int i = 0; i < 10; ++i) futures.push_back(pool.enqueue(o, i));
38
39     for (int i = 0; i < 10; ++i)
40         std::cout << std::string("Hello " + utils::to_string(i)) << " " << futures[i].
→get() << "\n";
41 }
```



ex_ThreadSafeMap

Shows how to use ThreadSafeMap class

Keywords:

- data container

Categories:

- core::data::basic::ThreadSafeMap

Program source:

```
1 #include <iostream>
2 #include <string>
3
4 #include <core/data/basic/ThreadSafeMap.hh>
5
6 /** @brief Shows how to use ThreadSafeMap class
7 *
8 * CATEGORIES: core::data::basic::ThreadSafeMap
9 * KEYWORDS: data container
10 */
11 int main(const int argc, const char* argv[]) {
12
13     core::data::basic::ThreadSafeMap<std::string,int> map;
14     int one = 1, two = 2;
15     map.insert_or_assign("one",one);
16     map.insert_or_assign("two",two);
17 }
```



ex_ThreeDTree

A simple example shows how to use BioShell kd-tree routines.

Keywords:

- neighborhood detection
- *data structures*
- *algorithms*

Categories:

- core::algorithms::trees::BinaryTreeNode; core::algorithms::trees::kd_tree.hh

Output files:

- stdout.out

Program source:

```

1 #include <memory>
2 #include <iostream>
3 #include <random>
4
5 #include <core/algorithms/trees/kd_tree.hh>
6 #include <core/algorithms/trees/BinaryTreeNode.hh>
7 #include <core/algorithms/trees/algorithms.hh>
8
9 #include <core/data/basic/Vec3.hh>
10 #include <core/calc/statistics/Random.hh>
11
12 using core::data::basic::Vec3;
13 using namespace core::algorithms::trees;
14 using namespace core::calc::statistics;
15
16
17 /// Tree traversal operation prints each node on the screen
18 struct PrintPoint {
19     void operator()(std::shared_ptr<BinaryTreeNode<KDTreeNode<Vec3>> > node) {
20         std::cout << node->element.element << " " << node->element.level << "\n";
21     }
22 };
23
24
25 /**
26  * @brief A simple example shows how to use BioShell kd-tree routines.
27  *
28  * The program generates N=500 random points and partites them into KD-tree. Later,
29  * the tree is used to find spatial
30  * neighbors in 3D space.
31  *
32  * CATEGORIES: core::algorithms::trees::BinaryTreeNode; core::algorithms::trees::kd_
33  * tree.hh

```

(continues on next page)

(continued from previous page)

```

31 * KEYWORDS: neighborhood detection; data structures; algorithms
32 */
33 int main(const int argc, const char* argv[]) {
34
35 // ----- Here we generate N random points in 3D space to be partitioned
36 const unsigned short N = 5000;
37 Random::seed(0);                                // seed random number generator
38 Random & gen = Random::get();                   // get rnd generator singleton
39 UniformRealRandomDistribution<double> rnd;    // uniform distribution will be used to_
40 →assign coordinates
41 std::vector<Vec3> atoms;                      // container for the points
42
43 // ----- We use <code>emplace_back()</code> to create Vec3 objects directly in_
44 →the container
45 for(unsigned int i=0;i<N;++i) atoms.emplace_back(rnd(gen),rnd(gen),rnd(gen));
46
47 // ----- Here the actual kd-tree is constructed
48 std::shared_ptr<BinaryTreeNode<KDTreeNode<Vec3>> > root = create_kd_tree<Vec3,
49 →std::vector<Vec3>::iterator, CompareAsReferences<Vec3>>(atoms.begin(),atoms.end());
50
51 // ----- Here we divide all the points into \f$2^2=4\f$ groups
52 std::vector<std::shared_ptr<BinaryTreeNode<KDTreeNode<Vec3>>> node_groups;
53 collect_given_level(root, 2, node_groups);
54 unsigned short subcluster_id = 0;
55 for(const auto node:node_groups) {      // then we mark all nodes in each subtree by_
56 →a distinct ID number
57     depth_first_preorder(node, [subcluster_id](std::shared_ptr<BinaryTreeNode
58 →<KDTreeNode<Vec3>> node) {
59         node->element.level = subcluster_id; });
60         ++subcluster_id;
61     }
62
63 breadth_first_preorder(root, PrintPoint()); // finally, each node is printed
64
65 // ----- Here is an example how to search the tree
66 Vec3 query(0.7,0.7,0.4); // a query point
67
68 // ----- Here is an example how to find the closes element
69 Vec3 best_point;
70 double d = search_kd_tree(root, [] (const Vec3 &v1, const Vec3 &v2) { return v1.
71 →distance_to(v2); }, query, best_point);
72 std::cout << "point closest to query: " << best_point << " : " << d << "\n";
73
74 Vec3 q_low(0.68,0.68,0.38);
75 Vec3 q_up(0.8,0.8,0.45);
76 std::vector<Vec3> hits;
77 search_kd_tree(root, q_low, q_up, 3, hits);
78 std::cout << "point within a box bounded by: " << q_low << " and " << q_up << ":\n";
79 for (const auto &p:hits) std::cout << p << "\n";
80 }
```



ex_TreeNode

Unit test which shows how to use a TreeNode data structure defined in BioShell

USAGE:

```
./ex_TreeNode
```

Keywords:

- *algorithms*
- *data structures*
- *graphs*

Categories:

- core::algorithms::trees::TreeNode

Output files:

- stdout.out

Program source:

```
1 #include <memory>
2 #include <iostream>
3 #include <string>
4
5 #include <core/algorithms/trees/TreeNode.hh>
6 #include <core/algorithms/trees/algorithms.hh>
7 #include <utils/options/OptionParser.hh>
8 #include <utils/exit.hh>
9
10 std::string program_info = R"(
11
12 Unit test which shows how to use a TreeNode data structure defined in BioShell
13
14 USAGE:
15 ./ex_TreeNode
16
17 )";
18
19 /** @brief Simple demo for TreeNode class
20 *
21 * This program creates a small tree with 7 nodes
22 *
23 * CATEGORIES: core::algorithms::trees::TreeNode
24 * KEYWORDS: algorithms; data structures; graphs
25 */
26 int main(const int argc, const char* argv[]) {
27
28     if ((argc > 1) && utils::options::call_for_help(argv[1]))

```

(continues on next page)

(continued from previous page)

```

29     utils::exit_OK_with_message(program_info);
30
31 using namespace core::algorithms::trees;
32
33     std::shared_ptr<TreeNode<std::string>> n1 = std::make_shared<TreeNode<std::string>>(
34     "A", 1);
35     std::shared_ptr<TreeNode<std::string>> n2 = std::make_shared<TreeNode<std::string>>(
36     "B", 2);
37     std::shared_ptr<TreeNode<std::string>> n3 = std::make_shared<TreeNode<std::string>>(
38     "C", 3);
39     std::shared_ptr<TreeNode<std::string>> n4 = std::make_shared<TreeNode<std::string>>(
40     "D", 4);
41     std::shared_ptr<TreeNode<std::string>> n5 = std::make_shared<TreeNode<std::string>>(
42     "E", 5);
43     std::shared_ptr<TreeNode<std::string>> n6 = std::make_shared<TreeNode<std::string>>(
44     "F", 6);
45     std::shared_ptr<TreeNode<std::string>> n7 = std::make_shared<TreeNode<std::string>>(
46     "G", 7);
47     n1->add_branch(n2);
48     n1->add_branch(n3);
49     n2->add_branch(n4);
50     n2->add_branch(n5);
51     n2->add_branch(n7);
52     n5->add_branch(n6);
53     depth_first_preorder(n1, [] (std::shared_ptr<TreeNode<std::string>> n) { std::cout << n->id << "\n"; });
54
55     std::cout << "Size of the tree: " << size(n1) << "\n";
56
57 return 0;
58 }
```



ex_UnionFind

Unit test which shows how to use the Union-Find algorithm.

USAGE:

```
./ex_UnionFind
```

REFERENCE: https://en.wikipedia.org/wiki/Disjoint-set_data_structure

Keywords:

- *data structures*
- *data structures*
- *algorithms*

Categories:

- core::algorithms::UnionFind

Output files:

- stdout.out

Program source:

```

1 #include <memory>
2 #include <iostream>
3
4 #include <core/algorithms/UnionFind.hh>
5 #include <core/calc/statistics/Random.hh>
6 #include <utils/options/OptionParser.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 Unit test which shows how to use the Union-Find algorithm.
12
13 USAGE:
14 ./ex_UnionFind
15
16 REFERENCE:
17     https://en.wikipedia.org/wiki/Disjoint-set_data_structure
18
19 )";
20
21 // ----- Data type of objects that will be clustered
22 struct Point2D {
23     float x, y;
24
25     Point2D(float nx, float ny) : x(nx), y(ny) {}
26

```

(continues on next page)

(continued from previous page)

```

27   float distance_to(const Point2D &p) { return sqrt((x - p.x) * (x - p.x) + (y - p.y)_
28   ↪* (y - p.y)); }
29 }
30
31 // ----- this operator is necessary because core:::algorithms::UnionFind keeps_
32 // std::map of data points
33 bool operator<(const Point2D &lhs, const Point2D &rhs) { return lhs.x < rhs.x; }

34 /**
35  * The program calculates greedy clustering of points in 2D. The number of points can_
36  * be provided from command line
37 */
38 /**
39  * CATEGORIES: core:::algorithms::UnionFind;
39  * KEYWORDS: data structures; data structures; algorithms
40 */
41
42 int main(const int argc, const char *argv[]) {
43
44     if ((argc > 1) && utils::options::call_for_help(argv[1]))
45         utils::exit_OK_with_message(program_info);
46
47     using namespace core:::calc::statistics;
48
49     // ----- Here we generate N random points in 3D space to be partitioned
50     const unsigned short N = (argc > 1) ? atoi(argv[1]) : 500;
51     const float cutoff = (argc > 2) ? atof(argv[2]) : 0.05;
52     std::vector<Point2D> points;           // container for the points
53
54     Random::seed(0);                      // seed random number generator
55     Random &gen = Random::get();           // get rnd generator singleton
56     UniformRealRandomDistribution<double> rnd; // uniform distribution will be used to_
57     ↪assign coordinates
58
59     core:::algorithms::UnionFind<Point2D, core:::index2> uf;
60     for (unsigned int i = 0; i < N; ++i) {
61         points.emplace_back(rnd(gen), rnd(gen)); // we use <code>emplace_back()</code> to_
62         ↪create point objects directly in the container
63         uf.add_element(points.back());
64         for (unsigned int j = 0; j < i; ++j) {
65             if (points[i].distance_to(points[j]) < cutoff) uf.union_set(i, j);
66         }
67     }
68
69     std::cout << "# x-coord      y-coord      cluster_assignment\n";
70     for (unsigned int i = 0; i < N; ++i)
71         std::cout << points[i].x << " " << points[i].y << " " << uf.find_set(i) << "\n";
72 }
```



ex_WL_Ar

The program runs a Wang-Landau MC simulation of argon gas. By default it starts from a regular lattice conformation unless an input file (PDB) with initial conformation is provided

USAGE:

```
ex_WL_Ar n_atoms density temperature small_cycles big_cycles [max_jump]
ex_WL_Ar starting.pdb density temperature small_cycles big_cycles [max_jump]
```

Keywords:

- no_keywords

Categories:

- no_categories

Program source:

```
1 #include <iostream>
2 #include <thread>
3
4 #include <core/data/basic/Vec3Cubic.hh>
5
6 #include <utils/string_utils.hh>
7 #include <utils/options/OptionParser.hh>
8
9 #include <simulations/systems/CartesianAtoms.hh>
10 #include <simulations/systems/BuildFluidSystem.hh>
11 #include <simulations/systems/SingleAtomType.hh>
12
13 #include <simulations/movers/TranslateAtom.hh>
14
15 #include <simulations/forcefields/cartesian/LJEnergySWHomogenic.hh>
16
17 #include <simulations/sampling/WangLandauSampler.hh>
18
19 #include <simulations/observers/ObserveEvaluators.hh>
20 #include <simulations/observers/cartesian/PdbObserver.hh>
21 #include <simulations/observers/ObserveWLSampling.hh>
22 #include <simulations/observers/AdjustMoversAcceptance.hh>
23 #include <simulations/observers/TriggerEveryN.hh>
24 #include <simulations/observers/cartesian/SimplePdbFormatter.hh>
25
26 #include <simulations/evaluators/CallEvaluator.hh>
27
28 using namespace core::data::basic;
29
30 utils::Logger logs("ex_WL_Ar");
31
32 std::string program_info = R"(
33
34 The program runs a Wang-Landau MC simulation of argon gas. By default it starts from a
35 regular lattice conformation
36 )"
```

(continues on next page)

(continued from previous page)

```

35 unless an input file (PDB) with initial conformation is provided
36 USAGE:
37   ex_WL_Ar n_atoms density temperature small_cycles big_cycles [max_jump]
38   ex_WL_Ar starting.pdb density temperature small_cycles big_cycles [max_jump]
39
40 );
41
42 const double EPSILON = 1.654E-21;           // [J] per molecule
43 const double EPSILON_BY_K = EPSILON / 1.381E-23;      // = 119.6 in Kelvins
44 const double SIGMA = 3.4;                  // in Angstroms
45
46 inline int bin_from_energy(double E)
47 {
48     return (int)(E / 100);
49 }
50
51 /** @brief Isothermal Monte Carlo simulation of argon gas.
52 */
53
54 int main(const int argc, const char* argv[])
55 {
56
57     using core::data::basic::Vec3Cubic;
58     using namespace simulations::systems;
59     using namespace simulations::movers; // for MoversSet
60     using namespace simulations::observers::cartesian; // for all observers
61
62     core::index4 n_outer_cycles = 1000;
63     core::index4 n_inner_cycles = 1000;
64     double density = 0.5;           // density of the system controls how many atoms will
65     // be contained in the box
66     double temperature = 97;        // in Kelvins
67     core::index4 n_atoms = 256;
68     double max_jump = 0.5;          // Random move range (in Angstroms)
69
70     core::data::structural::Structure_SP argon_structure = nullptr;
71     core::data::structural::PdbAtom_SP ar_atom = std::make_shared<core::data::structural::PdbAtom>(1, " AR");
72     if (argc < 6) std::cerr << program_info;
73     else {
74         if (utils::is_integer(argv[1])) n_atoms = atoi(argv[1]);
75         else { // --- read an input file if given
76             core::data::io::Pdb reader(argv[1]);
77             argon_structure = reader.create_structure(0);
78             n_atoms = argon_structure->count_atoms();
79         }
80         density = atof(argv[2]);
81         temperature = atof(argv[3]);
82         n_inner_cycles = atoi(argv[4]);
83         n_outer_cycles = atoi(argv[5]);
84         if (argc == 7) max_jump = atof(argv[6]);
85     }
86     double ar_volume = 4.0 / 3.0 * M_PI * SIGMA * SIGMA * SIGMA * n_atoms;
87     double box_len = pow(ar_volume / density, 0.3333333333333333);
88
89     // --- Initialize periodic boundary conditions
90     core::data::basic::Vec3Cubic::set_box_len(box_len);
91     logs << utils::LogLevel::INFO << "box width for " << int(n_atoms) << " atoms : " <
92     << box_len << "\n";

```

(continues on next page)

(continued from previous page)

```

90
91 // --- Create the system and distribute atoms in the box
92 AtomTypingInterface_SP ar_type = std::make_shared<SingleAtomType>(" AR");
93 CartesianAtoms ar(ar_type, n_atoms);
94 core::calc::statistics::Random::seed(1234);
95
96 if(argon_structure != nullptr) {           // --- read coordinates from a PDB file if provided
97     set_conformation(argon_structure->first_const_atom(), argon_structure->last_const_atom(), ar);
98 } else {                                // --- otherwise generate coordinates
99     const auto grid = std::make_shared<SimpleCubicGrid>(box_len, n_atoms);
100    BuildFluidSystem::generate(ar, *ar_atom, grid);
101 }
102 CartesianAtoms ar_backup(ar);           // --- make a backup system
103
104
105 // --- Create energy function - just LJ potential
106 simulations::forcefields::cartesian::LJEnergySWHomogenic lj_energy(ar, SIGMA,
107 EPSILON_BY_K);
108
109 // --- Create a mover, which is a random perturbation of an atom in this case, and place it in a movers' set
110 std::shared_ptr<TranslateAtom> translate = std::make_shared<TranslateAtom>(ar, ar_
111 backup, lj_energy);
112 translate->max_move_range_allowed(1.5);
113 MoversSet_SP movers = std::make_shared<simulations::movers::MoversSetSweep>();
114 movers->add_mover(translate, n_atoms);
115 translate->max_move_range(max_jump); // --- set the maximum distance a single atom can be moved by a single MC perturbation
116
117 // --- create a Wang-Landau Monte Carlo sampler
118 double initial_energy = lj_energy.energy(ar);
119 logs << utils::LogLevel::INFO << "Initial energy of the system (used to limit WL sampling) : " << initial_energy << "\n";
120
121 simulations::sampling::WangLandauSampler sampler(movers, initial_energy, bin_from_
122 energy, initial_energy + 1);
123
124 // ----- Create an observer which calls energy calculation and prints it on the screen
125 std::shared_ptr<simulations::observers::ObserveEvaluators> obs = std::make_shared<
126 <simulations::observers::ObserveEvaluators>("");
127 std::function<double(void)> recent_energy = [&lj_energy,&ar] () { return lj_energy.
128 energy(ar); };
129 obs->add_evaluator(
130     std::make_shared<simulations::evaluators::CallEvaluator<std::function<
131 <double(void)>>>(recent_energy, "energy", 8));
132
133 std::shared_ptr<AbstractPdbFormatter> fmt = std::make_shared<SimplePdbFormatter>("_
134 AR ", "AR ", "AR");
135 auto observe_trajectory = std::make_shared<
136 <simulations::observers::cartesian::PdbObserver>(ar, fmt, "ar_tra.pdb");
137
138 observe_trajectory->trigger(std::make_shared<
139 <simulations::observers::TriggerEveryN>(1));
140 sampler.outer_cycle_observer(observe_trajectory); // --- commented out to save disk space

```

(continues on next page)

(continued from previous page)

```
132
133     std::shared_ptr<simulations::observers::AdjustMoversAcceptance> observe_moves
134     = std::make_shared<simulations::observers::AdjustMoversAcceptance>(*movers,
135     ↪ "movers.dat", 0.4);
136
137     sampler.outer_cycle_observer(observe_moves);
138     sampler.outer_cycle_observer(obs);
139     sampler.cycle_size(1000);
140     sampler.inner_cycles(n_inner_cycles);
141     sampler.outer_cycles(n_outer_cycles);
142     sampler.outer_cycle_observer(std::make_shared
143     ↪ <simulations::observers::ObserveWLSampling>(sampler, "wl.dat"));
144
145     sampler.run();
146
147     simulations::observers::cartesian::PdbObserver final(ar, fmt, "final.pdb");
148     final.observe();
149     logs << utils::LogLevel::INFO << "Final energy " << lj_energy.energy(ar) << "\n";
150 }
```



ex_WL_Ising

The program runs a Wang-Landau Monte Carlo simulation of a simple 2D Ising system (spin glass)

Keywords:

- *observer*
- *simulation*

Categories:

- simulations/sampling/WangLandauSampler; simulations/systems/ising/Ising2D

Program source:

```

1 #include <iostream>
2
3 #include <simulations/observers/ObserveWLSampling.hh>
4
5 #include <simulations/movers/ising/SingleFlip2D.hh>
6 #include <simulations/movers/ising/WolffMove2D.hh>
7
8 #include <simulations/observers/ObserveMoversAcceptance.hh>
9 #include <simulations/observers/TriggerEveryN.hh>
10
11 #include <simulations/sampling/WangLandauSampler.hh>
12 #include <simulations/systems/ising/Ising2D.hh>
13
14 using namespace core::data::basic;
15
16 utils::Logger logs("ex_WL_Ising");
17
18 std::string program_info = R"(
19
20 The program runs a Wang-Landau Monte Carlo simulation of a simple 2D Ising system
21 ↪(spin glass)
22 )
23
24
25 /** @brief Turns energy of a system into an energy bin index (integer)
26 * @param energy - system's energy
27 * @return integer assigned to a bin; may be negative
28 */
29 inline int bfe(double energy) { return (int) energy; }
30
31 /** @brief The program runs a Wang-Landau Monte Carlo simulation of a simple 2D Ising
32 ↪system (spin glass).
33 *
34 * This example shows how to set up a WL simulation
35 *
36 * CATEGORIES: simulations/sampling/WangLandauSampler; simulations/systems/ising/
37 ↪Ising2D

```

(continues on next page)

(continued from previous page)

```

36 * KEYWORDS: observer; simulation
37 */
38 int main(const int argc, const char *argv[]) {
39
40     using namespace simulations::systems::ising;
41     using namespace simulations::movers::ising;
42     using namespace simulations::observers;
43
44     core::index4 n_outer_cycles = 1;
45     core::index4 n_inner_cycles = 10000;
46
47     core::index2 system_size = 10;
48     if (argc < 2) std::cerr << program_info;
49     else {
50         system_size = atoi(argv[1]);
51         if (argc == 4) {
52             n_inner_cycles = atoi(argv[2]);
53             n_outer_cycles = atoi(argv[3]);
54         }
55     }
56
57     core::calc::statistics::Random::get().seed(12345); // --- seed the generator for_
58     // repeatable results
59
60     // ----- Create the system to be sampled -----
61     std::shared_ptr<Ising2D<core::index1, core::index2>> system
62         = std::make_shared<Ising2D<core::index1, core::index2>>(system_size, system_
63         .size);
64     system->initialize(); // Populate system with random spins
65
66     // ----- Movers definition -----
67     simulations::movers::MoversSet_SP movers = std::make_shared
68     <simulations::movers::MoversSetSweep>();
69     movers->add_mover(std::make_shared<SingleFlip2D<core::index1, core::index2>>
70     (*system), system->count_spins());
71     movers->add_mover(std::make_shared<WolffMove2D<core::index1, core::index2>>
72     (*system), system->count_spins() * 0.2);
73
74     // ----- Create the sampler -----
75     const double initial_energy = system->calculate();
76     simulations::sampling::WangLandauSampler sampler(movers, initial_energy, bfe, 13);
77     sampler.inner_cycles(n_inner_cycles);
78     sampler.outer_cycles(n_outer_cycles);
79     sampler.inner_cycle_observer(std::make_shared<ObserveWLSampling>(sampler, "wl.dat
80     "));
81
82     sampler.run();
83 }
```



ex_XML

Demonstrate how to parse XML with BioShell utilities. The test runs on a predefined XML data

USAGE:

```
./ex_XML
```

```
)";
```

```
using namespace core::data::io;
```

```
std::string xml_data = R"(<product> <id>15</id> <name>Widgets</name> <description>Example text.</description> <options type="color"> <item value="Purple" shade="bright" /> <item>Green</item> <item>Orange</item> </options> </product> )";
```

Keywords:

- *XML*

Categories:

- core/data/io/XML

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/XML.hh>
4 #include <core/data/io/XMLElement.hh>
5 #include <utils/LogManager.hh>
6 #include <utils/options/OptionParser.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10 Demonstrate how to parse XML with BioShell utilities. The test runs on a predefined XML data
11
12 USAGE:
13   ./ex_XML
14
15 )";
16
17 using namespace core::data::io;
18
19 std::string xml_data =
20 R"(<product>
21   <id>15</id>
22   <name>Widgets</name>
23 )";
```

(continues on next page)

(continued from previous page)

```

24   <description>Example text.</description>
25   <options type="color">
26     <item value="Purple" shade="bright" />
27     <item>Green</item>
28     <item>Orange</item>
29   </options>
30 </product>
31 ) ";
32
33 /** @brief Simple for XML I/O utils.
34 *
35 * CATEGORIES: core/data/io/XML
36 * KEYWORDS: XML
37 */
38 int main(int argc, char *argv[]) {
39
40   if ((argc > 1) && utils::options::call_for_help(argv[1]))
41     utils::exit_OK_with_message(program_info);
42
43   utils::LogManager::FINE();
44   XML xxx;
45   if (argc == 1) {
46     std::istringstream input(xml_data);
47     xxx.load_data(input);
48   } else xxx.load_data(argv[1]);
49   std::cout << xxx.document_root();
50
51   return 0;
52 }
```



ex_alignment_io

Unit test which reads alignment in Edinburgh format or calculates a global sequence alignment for two predefined sequences. It saves output alignment in Edinburgh format.

USAGE:

```
./ex_alignment_io [alignment]
```

EXAMPLE:

```
./ex_alignment_io example.edinb
```

Keywords:

- *sequence alignment*

Categories:

- core/data/io/alignment_io.hh

Input files:

- example.edinb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/data/io/alignment_io.hh>
4 #include <core/BioShellEnvironment.hh>
5 #include <core/alignment/NWAligner.hh>
6 #include <core/alignment/on_alignment_computations.hh>
7 #include <core/alignment/PairwiseAlignment.hh>
8 #include <core/alignment/PairwiseSequenceAlignment.hh>
9 #include <core/alignment/scoring/SimilarityMatrix.hh>
10 #include <core/alignment/scoring/SimilarityMatrixScore.hh>
11 #include <utils/options/OptionParser.hh>
12 #include <utils/exit.hh>
13
14 std::string program_info = R"(
15
16 Unit test which reads alignment in Edinburgh format or calculates a global sequence
17 ↴alignment
18 for two predefined sequences. It saves output alignment in Edinburgh format.
19
20 USAGE:
21 . /ex_alignment_io [alignment]

```

(continues on next page)

(continued from previous page)

```

21
22 EXAMPLE:
23 ./ex_alignment_io example.edinb
24
25 ) ";
26
27 /** @brief Read alignment in Edinburgh format or calculate a new one from given_
28  sequences; write Edinburgh.
29 *
30 * CATEGORIES: core/data/io/alignment_io.hh
31 * KEYWORDS: sequence alignment
32 */
33
34 int main(const int argc, const char* argv[]) {
35
36     if ((argc > 1) && utils::options::call_for_help(argv[1]))
37         utils::exit_OK_with_message(program_info);
38
39     using namespace core::alignment;
40     using namespace core::alignment::scoring;
41
42     if (argc > 1) { // If there was an input alignment file given, read it!
43         std::vector<PairwiseSequenceAlignment_SP> alignments;
44         auto ali = core::data::io::read_edinburgh(argv[1], alignments);
45         for (const PairwiseSequenceAlignment_SP & ali : alignments)
46             core::data::io::write_edinburgh(*ali, std::cout, 80);
47     } else { // otherwise, align the two sequences defined below
48         // ----- The two sequences that will be aligned
49         std::string query =
50             "MKGWLFLVIAIVGEVIATSALKSSEGFTKLAPSIVIIGYGIAFYFLSLVLSIPVGVAYAVWSGLGVVITAIAWLLHGQKLDANGFVGMGLI"
51             ";";
52         std::string tmplt =
53             "MIYLYLLCAIFAEVVATSLKSTEGFTRLWPTVGCLVGYGIAFALLALSISHGMQTDVAYALWSAIGTAAIVLVAVLFLGSPISMKVVGVLI"
54             ";";
55         // ----- Gap penalties
56         short int open = -10;
57         short int extend = -2;
58         // ----- load BLOSUM matrix from bioshell's library; the directory must be_
59         //defined as a shell variable
60         const std::shared_ptr<SimilarityMatrix<short int>> b62_matrix = SimilarityMatrix<short int>::from_ncbi_file("alignments/BLOSUM62");
61         const SimilarityMatrixScore<short int> b62_score(query, tmplt, *b62_matrix);
62         NWAligner<short int, SimilarityMatrixScore<short int>> global(std::max(query.
63             length(), tmplt.length()));
64         // ----- Compute and backtrace the alignment
65         global.align(open, extend, b62_score);
66         const PairwiseAlignment_SP ali = global.backtrace();
67         // ----- Convert the abstract alignment to a pairwise sequence alignment_
68         //object
69         core::data::sequence::Sequence query_seq("Q7B1Y7_SALEN",query);
70         core::data::sequence::Sequence tmplt_seq("MMR_MYCTU",tmplt);
71         core::data::io::write_edinburgh(query_seq, *ali, tmplt_seq, std::cout, 80);
72     }
73 }
```



ex_basic_algebra

Unit test that calculates eigenvalues and eigenvectors for a 3x3 matrix

USAGE:

```
./ex_basic_algebra
```

Keywords:

- *numerical methods*

Categories:

- core/calc/numeric/basic_algebra.hh

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/basic/Vec3.hh>
4 #include <core/calc/numeric/basic_algebra.hh>
5 #include <utils/options/OptionParser.hh>
6
7 std::string program_info = R"(
8
9 Unit test that calculates eigenvalues and eigenvectors for a 3x3 matrix
10
11 USAGE:
12 ./ex_basic_algebra
13
14 )";
15
16 /** @brief ex_basic_algebra illustrates how to calculate eigenvalues and eigenvectors
17  * for a 3x3 matrix
18 *
19 * CATEGORIES: core/calc/numeric/basic_algebra.hh
20 * KEYWORDS: numerical methods
21 */
22 int main(const int argc, const char* argv[]) {
23
24     if ((argc > 1) && utils::options::call_for_help(argv[1]))
25         utils::exit_OK_with_message(program_info);
26
27     using namespace core::calc::numeric;
28     using namespace core::data::basic; // --- for Array2D
29
30     Array2D<double> m3x3(3,3,{2, -1, 0, -1, 2, -1, 0, -1, 2}); // --- input matrix to
31     // be solved
```

(continues on next page)

(continued from previous page)

```
30     std::cout << "\nOriginal matrix:\n";
31     m3x3.print("%8.3f", std::cout);
32
33     std::vector<double> eigenval;
34     core::calc::numeric::eigenvalues3(m3x3, eigenval);
35     std::cout << "\nEigenvalues: " << eigenval[0] << " " << eigenval[1] << " " <<_
36     eigenval[2] << "\n\n";
37
38     auto eigenv = eigenvectors3(m3x3, eigenval);
39     std::cout << "\nEigenvectors:\n" << eigenv[0] << "\n" << eigenv[1] << "\n" <<_
40     eigenv[2] << "\n\n";
}
```



ex_cabs_representation

Unit test which reads an all-atom structure from a PDB file and produces a structure in CABS representation.

USAGE:

```
./ex_cabs_representation input.pdb
```

EXAMPLE:

```
./ex_cabs_representation 2gb1.pdb
```

REFERENCE: Kolinski, Andrzej. “Protein modeling and structure prediction with a reduced representation.” *Acta Biochimica Polonica* 51 (2004).

Kmiecik, Sebastian, et al. “Coarse-grained protein models and their applications.” *Chemical reviews* 116.14 (2016): 7898–7936. doi: 10.1021/acs.chemrev.6b00163

Keywords:

- *PDB input*
- CABS
- representation

Categories:

- simulations::representations::cabs::cabs_utils

Input files:

- 2gb1.pdb

Output files:

- stdout.out
- out.pdb

Program source:

```

1 #include <iostream>
2 #include <iomanip>
3
4 #include <core/data/io/Pdb.hh>
5 #include <core/data/structural/selectors/structure_selectors.hh>
6 #include <simulations/representations/cabs/cabs_utils.hh>
7 #include <utils/options/OptionParser.hh>
8 #include <utils/exit.hh>
9
10 std::string program_info = R"(
11
```

(continues on next page)

(continued from previous page)

```

12 Unit test which reads an all-atom structure from a PDB file and produces a structure in CABS representation.
13
14 USAGE:
15     ./ex_cabs_representation input.pdb
16
17 EXAMPLE:
18     ./ex_cabs_representation 2gb1.pdb
19
20 REFERENCE:
21 Kolinski, Andrzej. "Protein modeling and structure prediction with a reduced representation."
22 Acta Biochimica Polonica 51 (2004).
23
24 Kmiecik, Sebastian, et al. "Coarse-grained protein models and their applications."
25 Chemical reviews 116.14 (2016): 7898-7936. doi: 10.1021/acs.chemrev.6b00163
26
27 );
28
29 using namespace core::data::structural;
30 using namespace core::data::io;
31
32 /** @brief Reads an all-atom structure from a PDB file and produces a structure in CABS representation.
33 *
34 * CATEGORIES: simulations::representations::cabs::cabs_utils
35 * KEYWORDS: PDB input; CABS; representation
36 */
37 int main(const int argc, const char* argv[]) {
38
39     if ((argc > 1) && utils::options::call_for_help(argv[1]))
40         utils::exit_OK_with_message(program_info);
41
42     // --- Read the input PDB and create a structure object
43     core::data::io::Pdb reader(argv[1], is_not_alternative, only_ss_from_header, true);
44     core::data::structural::Structure_SP strctr = reader.create_structure(0);
45
46     // --- Check whether loaded structure is in the CABS representation
47     if (simulations::representations::is_cabs_model(*strctr))
48         std::cerr<<"Loaded structure of "<<argv[1]<<" has CABS representation! Load all-atom model.\n";
49     else if (simulations::representations::is_cabsbb_model(*strctr)) {
50         std::cerr<<"Loaded structure of "<<argv[1]<<" has CABSBB representation! Load fullatom model.\n";
51     }
52     else {
53         // --- Convert the Structure into CABS representation and write the result in the PDB format
54         Structure_SP structure_sp = simulations::representations::cabs_representation(*strctr);
55         for (auto atom_sp = structure_sp->first_atom(); atom_sp != structure_sp->last_atom(); ++atom_sp)
56             std::cout << (*atom_sp)->to_pdb_line() << "\n";
57
58         // --- Here we generate CONNECT lines so the PDB file displays nicely in PyMOL
59         auto prev_ca_sp = *structure_sp->first_atom(); // --- the very first CA in the structure

```

(continues on next page)

(continued from previous page)

```

60   for (auto res_sp_it = (++(structure_sp->first_residue())); res_sp_it != structure_
61     ↪sp->last_residue(); ++res_sp_it) {
62     auto ca = *((*res_sp_it)->cbegin()); // --- iterator to the CA of this residue
63     core::data::io::Conect cn(prev_ca_sp->id(), ca->id());
64     std::cout << cn.to_pdb_line();
65     if ((*res_sp_it)->count_atoms() < 2) { // --- it has also CB
66       auto cb = *((*res_sp_it)->cbegin() + 2); // --- CB is always the third one
67       core::data::io::Conect cn(ca->id(), cb->id());
68       std::cout << cn.to_pdb_line();
69       if ((*res_sp_it)->count_atoms() < 2) { // --- it has also CB
70         auto sc = *((*res_sp_it)->cbegin() + 3); // --- SC is always the fourth one
71         core::data::io::Conect cn(cb->id(), sc->id());
72         std::cout << cn.to_pdb_line();
73       }
74     prev_ca_sp = ca;
75   }
76 }
77 }
```



ex_cabs_rotamers**Keywords:**

- no_keywords

Categories:

- no_categories

Input files:

- 2gb1.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/chemical/ChiAnglesDefinition.hh>
4
5 #include <core/data/io/Pdb.hh>
6 #include <core/data/structural/selectors/structure_selectors.hh>
7 #include <core/data/structural/selectors>SelectChainBreaks.hh>
8 #include <core/data/structural/selectors>SelectClashingResidues.hh>
9 #include <core/data/structural/selectors>SelectContiguousResidues.hh>
10
11 #include <core/calc/structural/angles.hh>
12 #include <core/calc/structural/protein_angles.hh>
13 #include <core/calc/structural/transformations/CartesianToSpherical.hh>
14 #include <core/calc/structural/transformations/transformation_utils.hh>
15 #include <core/calc/structural/transformations/Rototranslation.hh>
16
17 #include <utils/io_utils.hh>
18 #include <utils/LogManager.hh>
19
20 utils::Logger logger("ex_cabs_rotamers");
21
22 int main(const int argc, const char* argv[]) {
23
24     using namespace core::chemical;
25     using namespace core::data::structural;
26     using namespace core::calc::structural;
27     using namespace core::calc::structural::transformations;
28
29     utils::LogManager::get().set_level("FINE");
30
31     if(argc==1) {
32         return 0;
33     }

```

(continues on next page)

(continued from previous page)

```

34     core::data::io::Pdb reader(argv[1],
35         core::data::io::all_true(core::data::io::is_not_alternative,core::data::io::is_
36         ↪not_hydrogen),
37         core::data::io::keep_all, false); // Create a PDB reader for a given file
38     core::data::structural::Structure_SP str = reader.create_structure(0); // Create a_
39     ↪structure object from the first model
40
41     // ----- Selector we use to be sure that the residue is correct
42     selectors::IsAA is_aa;
43     selectors::IsBBCB atom_is_bb_cb;
44     selectors::ResidueHasBBCB has_bb_cb;
45     selectors::ResidueHasAllHeavyAtoms all_atoms;
46     selectors::SelectChainBreaks breaks;
47     selectors::SelectClashingResidues clash_test(str,2.3);
48     selectors::SelectContiguousResidues check_resids;
49
50     std::cout << "# r alpha theta alpha theta x y z ss res_"
51     ↪id aa chain prot rotamer chi angles\n";
52     CartesianToSpherical acs;
53     std::vector<std::string> lcs_atom_names = {" N ", " CA ", " C ";
54     auto ires = str->first_residue(); ++ires;
55     auto next_res = str->first_residue(); ++(++next_res);
56     for (; next_res != str->last_residue(); ++ires,++next_res) { // iterate over all_
57     ↪residues starting from the second one
58
58     if (!is_aa(**ires)) continue;
59     if (((**ires).residue_type() == Monomer::GLY) || ((**ires).residue_type() ==_
60     ↪Monomer::ALA)) continue;
61
62     if(!has_bb_cb(**ires)) {
63         logger << utils::LogLevel::WARNING << "Residue "<<(**ires) << " has incomplete_
64     ↪backbone or lacks its CB atom\n";
65         continue;
66     }
67
68     if(!all_atoms(**ires)) {
69         logger << utils::LogLevel::WARNING << "Residue side chain "<<(**ires) << " is_
70     ↪incomplete\n";
71         continue;
72     }
73
74     if(!check_resids(**ires)) {
75         logger << utils::LogLevel::WARNING << "Residue "<<(**ires) << " is broken\n";
76         continue;
77     }
78
79     if(breaks(**ires)) {
80         logger << utils::LogLevel::WARNING << "Residue "<<(**ires) << " is at chain_
81     ↪break\n";
82         continue;
83     }
84
85     if(clash_test(**ires)) {
86         logger << utils::LogLevel::WARNING << "Residue "<<(**ires) << " is in a steric_
87     ↪clash\n";
88         continue;
89     }

```

(continues on next page)

(continued from previous page)

```

82     PdbAtom_SP CA = (*ires)->find_atom_safe(" CA ");
83     PdbAtom_SP N = (*ires)->find_atom_safe(" N  ");
84     PdbAtom_SP C = (*ires)->find_atom_safe(" C  ");
85     PdbAtom_SP CB = (*ires)->find_atom_safe(" CB ");
86     double t = core::calc::structural::evaluate_dihedral_angle(*N, *C, *CB, *CA) *_
87     ↪180.0 / 3.14159;
88     if ((t < -50) || (t > -20)) {
89         logger << utils::LogLevel::WARNING << "Residue " << (**ires) << " has incorrect_"
90         ↪geometry at CA. Dihedral angle: " << t << "\n";
91         continue;
92     };
93
94     core::data::basic::Vec3 cm;
95     double n = 0;
96     std::for_each((*ires)->cbegin(), (*ires)->cbegin(), [&] (const PdbAtom_SP a) { if (!atom_
97     ↪is_bb_cb(*a)) { cm+=(*a); ++n; } });
98
99     if (n == 0) continue;
100
101     cm /= n;
102     Rototranslation_SP lcs = local_coordinates_three_atoms(**ires, lcs_atom_names);
103     Vec3 sph;
104     lcs->apply(cm);
105     acs.apply(cm, sph);
106     std::cout << utils::string_format("%7.4f %7.2f %7.2f %7.4f %7.4f    %6.3f %6.3f %6.
107     ↪3f ",_
108     sph.x, to_degrees(sph.y), to_degrees(sph.z), sph.y, sph.z, cm.x, cm.y, cm.z);
109
110     std::cout << utils::string_format("%c %6d %3s %s %s %4s",
111     (*ires)->ss(), (*ires)->id(), (*ires)->residue_type().code3.c_str(), (*ires)->
112     ↪owner()->id().c_str(),
113     utils::basename(str->code()).c_str(),
114     core::calc::structural::define_rotamer(**ires).c_str());
115     for (unsigned short i = 1; i <= core::chemical::ChiAnglesDefinition::count_chi_
116     ↪angles((*ires)->residue_type()); ++i)
117         std::cout << utils::string_format(" %6.1f", core::calc::structural::evaluate_
118     ↪chi(**ires, i) * 180.0 / 3.1415);
119         std::cout << "\n";
120     }
121 }
```



ex_cabsbb_representation

Converts all-atom protein structure to CABS-bb representation

USAGE:

```
ex_cabsbb_representation input.pdb
```

USAGE:

```
ex_cabsbb_representation 2gb1.pdb
```

Keywords:

- *PDB input*
- CABS-bb

Categories:

- simulations/representations/cabs/cabs_utils

Input files:

- 2gb1.pdb
- 1rrx.pdb
- 5l3z.pdb
- 5lpc.pdb

Output files:

- 1rrx-cabsbb.pdb
- 2gb1-cabsbb.pdb
- 5l3z-cabsbb.pdb
- stdout.out
- 5lpc-cabsbb.pdb

Program source:

```

1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <utils/exit.hh>
5
6 #include <simulations/representations/cabs/cabs_utils.hh>
7
8 using namespace core::data::structural;
9 using namespace core::data::io;
```

(continues on next page)

(continued from previous page)

```

10
11  /** @brief Reads an all-atom structure from a PDB file and produces a structure in_
12   *      ↪CABS-BB representation.
13   */
14
15 std::string program_info = R"(

16 Converts all-atom protein structure to CABS-bb representation
17 USAGE:
18     ex_cabsbb_representation input.pdb
19 USAGE:
20     ex_cabsbb_representation 2gb1.pdb
21 )";

22
23 /** @brief Converts all-atom protein structure to CABS-bb representation
24 *
25 *
26 * CATEGORIES: simulations/representations/cabs/cabs_utils;
27 * KEYWORDS: PDB input; CABS-bb
28 */int main(const int argc, const char* argv[]) {

29
30     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
31   ↪missing program parameter
32
33     // --- Read the input PDB and create a structure object
34     core::data::io::Pdb reader(argv[1], all_true(is_not_hydrogen, is_not_water, is_not_
35   ↪alternative), keep_all, true);
36     core::data::structural::Structure_SP strctr = reader.create_structure(0);

37
38     // --- Check whether loaded structure is in the CABSBB representation
39     if (simulations::representations::is_cabsbb_model(*strctr))
40         std::cerr << "Loaded structure of " << argv[1] << " has CABSBB representation!_"
41   ↪Load fullatom model.\n";
42     else if (simulations::representations::is_cabs_model(*strctr))
43         std::cerr << "Loaded structure of " << argv[1] << " has CABS representation! Load_"
44   ↪fullatom model.\n";

45
46     else {
47         // --- Convert the Structure into CABSBB representation and write the result in_
48   ↪the PDB format
49         core::data::structural::Structure_SP structure_sp =
50           simulations::representations::cabsbb_representation(*strctr);
51         for (auto atom_sp = structure_sp->first_atom(); atom_sp != structure_sp->last_
52   ↪atom(); ++atom_sp)
53             std::cout << (*atom_sp)->to_pdb_line() << "\n";
54
55         // --- Here we generate CONNECT lines so the PDB file displays nicely in PyMOL
56         for (auto it = structure_sp->first_const_residue(); it != structure_sp->last_
57   ↪const_residue(); ++it) {
58             if ((*it)->count_atoms() == 6) { // --- if this CABSBB residue has 6 atoms, it_
59   ↪must have the SC atom
60                 auto cb = *((*it)->cbegin() + 4); // --- CB is always the fifth one
61                 auto sc = *((*it)->cbegin() + 5); // --- SC is always the sixth one
62                 core::data::io::Conect cn(cb->id(), sc->id());
63                 std::cout << cn.to_pdb_line();
64             }
65         }
66     }
67 }
```

(continues on next page)

(continued from previous page)

57 }
58 }



ex_chi2_independence_test

Performs chi-square test: calculates p-value for a given number of DOFs. Alternatively, it can read a contingency matrix from a file and calculate test for independence of its two first rows When no input data is provided, the example performs Chi-square independence test on a test data

USAGE:

```
ex_chi2_independence_test [n_dofs chi2_value]
ex_chi2_independence_test [input_contingency_matrix_file]
```

EXAMPLE:

```
ex_chi2_independence_test 4 1.52
ex_chi2_independence_test matrix.dat
```

REFERENCE: Pearson, Karl. "X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling." The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 50.302 (1900): 157-175.

Keywords:

- *statistics*
- *data table*

Categories:

- core::calc::statistics::chi2_independence_test

Input files:

- chi2_test_data.txt

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/basic/Array2D.hh>
4 #include <core/calc/statistics/simple_statistics.hh>
5
6 std::string program_info = R"(
7
8 Performs chi-square test: calculates p-value for a given number of DOFs. ↴
9 Alternatively,
10 it can read a contingency matrix from a file and calculate test for independence of ↴
11 its two first rows
12 When no input data is provided, the example performs Chi-square independence test on ↴
13 a test data
14 )"
```

(continues on next page)

(continued from previous page)

```

11 USAGE:
12     ex_chi2_independence_test [n_dofs chi2_value]
13     ex_chi2_independence_test [input_contingency_matrix_file]
14
15 EXAMPLE:
16     ex_chi2_independence_test 4 1.52
17     ex_chi2_independence_test matrix.dat
18
19 REFERENCE:
20 Pearson, Karl. "X. On the criterion that a given system of deviations from the
21 ↪probable in the case  

22 of a correlated system of variables is such that it can be reasonably supposed to
23 ↪have arisen from random sampling."
24 The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 50.  

25 ↪302 (1900): 157-175.
26
27 )";
28
29 /**
30  * @brief Chi-square test for independence.
31  *
32  * @CATEGORIES: core::calc::statistics::chi2_independence_test;
33  * @KEYWORDS: statistics; data table
34  */
35 int main(const int argc, const char* argv[]) {
36
37     using core::data::basic::Array2D;
38
39     if(argc==3) { // --- calculate chi-square test for given chi-square statistics
40 ↪value and the number of DOFs
41         int k = utils::from_string<int>(argv[1]);
42         double crit = utils::from_string<double>(argv[2]);
43         std::cout << "# DOFs: " << k << "\n";
44         std::cout << "# chi2 value: " << crit << "\n";
45         std::cout << "# p-value: " << core::calc::numeric::chi_square_pvalue(k,crit) <
46             "\n";
47         return 0;
48     }
49
50     if(argc==2) { // --- calculate chi-square test for given data (test the
51 ↪independence of the two first rows)
52         Array2D<core::index4> m = Array2D<core::index4>::from_file(argv[1]);
53         int k = (m.count_rows() - 1) * (m.count_columns() - 1);
54         double crit = core::calc::statistics::chi2_independence_test(m);
55         std::cout << "# DOFs: " << k << "\n";
56         std::cout << "# chi2 value: " << crit << "\n";
57         std::cout << "# p-value: " << core::calc::numeric::chi_square_pvalue(k, crit) <
58             "\n";
59
60         return 0;
61     }
62
63     std::cerr << program_info;
64
65     std::vector<core::index4> data = {71,154,398,4992,2808,2737};
66     Array2D<core::index4> m(2,3, data);
67
68     int k = 2; // (2-1)*(3-1) = 2

```

(continues on next page)

(continued from previous page)

```
61 double crit = core:::calc::statistics::chi2_independence_test(m);  
62  
63 std::cout << "# DOFs: " << k << "\n";  
64 std::cout << "# chi2 value: " << crit << "\n";  
65 std::cout << "# p-value: " << core:::calc::numeric::chi_square_pvalue(k,crit) <<  
66 }  
67 ↪ "\n";
```



ex_consecutive_find

Unit test which shows how to find islands of consecutive elements in a container.

USAGE:

```
./ex_consecutive_find
```

Keywords:

- *algorithms*
- *data structures*

Categories:

- core/algorithms/basic_algorithms.hh

Output files:

- stdout.out

Program source:

```

1 #include <vector>
2 #include <iostream>
3 #include <iterator>
4
5 #include <core/algorithms/basic_algorithms.hh>
6 #include <utils/options/OptionParser.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 Unit test which shows how to find islands of consecutive elements in a container.
12
13 USAGE:
14 ./ex_consecutive_find
15
16 )";
17
18 struct AreConsecutive {
19     bool operator()(int i, int n) { return (n - i) == 1; }
20 };
21
22 struct SSranges {
23     bool operator()(char ci, char cn) { return (ci==cn); }
24 };
25
26 /**
27 * @brief Shows how to find islands of consecutive elements in a container
28 */
29 * CATEGORIES: core/algorithms/basic_algorithms.hh

```

(continues on next page)

(continued from previous page)

```
30 * KEYWORDS: algorithms; data structures
31 */
32 int main(const int argc, const char *argv[]) {
33
34     if ((argc > 1) && utils::options::call_for_help(argv[1]))
35         utils::exit_OK_with_message(program_info);
36
37     std::vector<int> v{-3, 1, 2, 4, 5, 7, 8, 9, 10, 12, 12, 13, 16, 18, 20, 21, 22, 23};
38     std::vector<std::pair<int, int>> islands;
39
40     int n_islands = core::algorithms::consecutive_find(v.begin(), v.end(),
41     ↪ AreConsecutive(), islands);
41
42     for (const auto &is : islands) {
43         for (int i = is.first; i <= is.second; ++i)
44             std::cout << v[i] << " ";
45         std::cout << "\n";
46     }
47
48     std::string ss("CEEEEEECCCCCCEEEEEECCCCHHHHHHHHHHHHCCCCCEEEEEECCCEEEEC");
49     std::vector<char> v_ss(ss.begin(), ss.end());
50     islands.clear();
51     n_islands = core::algorithms::consecutive_find(v_ss.begin(), v_ss.end(), SSranges(),
52     ↪ islands);
53     for (const auto &is : islands)
54         std::cout << v_ss[is.first] << " " << is.first << " " << is.second << "\n";
}
```



ex_count_residues_by_type

Reads a Multiple Sequence Alignment (MSA) in ClustalW format and counts residues by its type.

EXAMPLE:

```
./ex_count_residues_by_type cyped.CYP109.aln
```

Keywords:

- *clustal input*
- *MSA*

Categories:

- core:::data::sequence::sequence_utils

Input files:

- CYP109B1.aln

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/clustalw_io.hh>
4 #include <core/data/sequence/sequence_utils.hh>
5 #include <utils/exit.hh>
6 #include <utils/io_utils.hh>
7
8 std::string program_info = R"(
9
10 Reads a Multiple Sequence Alignment (MSA) in ClustalW format and counts residues by
11 ↴its type.
12
13 EXAMPLE:
14     ./ex_count_residues_by_type cyped.CYP109.aln
15 )
16
17 /** @brief Reads a MSA in ClustalW format and prints by-residue counts
18 *
19 * CATEGORIES: core:::data::sequence::sequence_utils
20 * KEYWORDS: clustal input; MSA
21 */
22 int main(const int argc, const char* argv[]) {
23
```

(continues on next page)

(continued from previous page)

```
24  if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
→missing program parameter
25
26  using namespace core::data::io;
27  using namespace core::data::sequence;
28
29  std::vector<Sequence_SP> msa;    // --- Sequence_SP is just a shorter name for_
→std::shared_ptr<Sequence>
30  core::data::io::read_clustalw_file(argv[1], msa);
31
32  std::map<core::chemical::Monomer, core::index4> counts = core::data::sequence::count_
→residues_by_type(msa);
33  for (const auto &key_val : counts) std::cout << key_val.first.code3 << " " << key_
→val.second << "\n";
34 }
```



ex_define_rotamer

Prints rotamer type (M-P-T code) for each amino acid residue in the input PDB structure

USAGE:

```
ex_define_rotamer input.pdb
```

EXAMPLE:

```
ex_define_rotamer 5edw.pdb
```

OUTPUT (fragment): 277 ASP 2 TP 278 LYS 4 incomplete 279 ARG 4 TTMT 280 ILE 2 MM 281 PRO 3 PMP 282
LYS 4 MTMM 283 ALA 0 284 ILE 2 TT

Keywords:

- *PDB input*
- *structural properties*
- rotamers
- *STL*

Categories:

- core::chemical::ChiAnglesDefinition; core::data::structural::ResidueHasAllHeavyAtoms

Input files:

- 5edw.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <iomanip>
3 #include <core/data/io/Pdb.hh>
4 #include <core/calc/structural/protein_angles.hh>
5 #include <core/chemical/ChiAnglesDefinition.hh>
6 #include <utils/exit.hh>
7 #include <core/data/structural/selectors/structure_selectors.hh>
8
9 std::string program_info = R"(
10
11 Prints rotamer type (M-P-T code) for each amino acid residue in the input PDB_
12 ↵structure
13 USAGE:
14     ex_define_rotamer input.pdb

```

(continues on next page)

(continued from previous page)

```

14
15 EXAMPLE:
16   ex_define_rotamer 5edw.pdb
17
18 OUTPUT (fragment):
19   277 ASP 2    TP
20   278 LYS 4 incomplete
21   279 ARG 4 TTMT
22   280 ILE 2    MM
23   281 PRO 3    PMP
24   282 LYS 4 MTMM
25   283 ALA 0
26   284 ILE 2    TT
27
28 ) ";
29
30 /** @brief Prints rotamer type (M-P-T code) for each amino acid residue in the input_
31  ↪PDB structure
32 *
33 * CATEGORIES: core::chemical::ChiAnglesDefinition; ↪
34 ↪core::data::structural::ResidueHasAllHeavyAtoms
35 * KEYWORDS: PDB input; structural properties; rotamers; STL
36 */
37 int main(const int argc, const char *argv[]) {
38
39   if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
40  ↪missing program parameter
41
42   using namespace core::data::io;
43   using namespace core::data::structural;
44
45   Pdb reader(argv[1]); // file name (PDB format, may be gzip-ped)
46   Structure_SP strctr = reader.create_structure(0);
47   selectors::ResidueHasAllHeavyAtoms has_full_sc;
48   selectors::IsAA is_aa;
49
50   // Iterate over all residues
51   for (auto ires = strctr->first_residue(); ires != strctr->last_residue(); ++ires) {
52     core::data::structural::Residue &res_sp = (**ires);
53     if (!is_aa(res_sp)) continue;
54     std::cout << std::setw(4) << res_sp.id() << " " << res_sp.residue_type().code3 <<
55  ↪" "
56  ↪           << core::chemical::ChiAnglesDefinition::count_chi_angles(res_sp.residue_
57  ↪type());
58     if (has_full_sc(res_sp)) std::cout << std::setw(5) << ↪
59  ↪core::calc::structural::define_rotamer(res_sp);
60     else std::cout << " incomplete";
61     std::cout << "\n";
62   }
63 }
```



ex_expectation_maximization

Example showing how to use expectation-maximization method retrieve arbitrary data according to a sequence alignment object

USAGE:

```
./ex_expectation_maximization
```

REFERENCE: Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm." Journal of the Royal Statistical Society: Series B 39 (1977): 1-22. doi: 10.1111/j.2517-6161.1977.tb01600.x

Keywords:

- *estimation*
- *expectation-maximization*

Categories:

- core::calc::statistics::NormalDistribution; core::calc::statistics::BivariateNormal;core/calc/statistics/expectation_maximization.hh

Output files:

- stdout.out

Program source:

```
1 #include <math.h>
2
3 #include <iostream>
4 #include <random>
5 #include <vector>
6
7 #include <core/calc/statistics/NormalDistribution.hh>
8 #include <core/calc/statistics/BivariateNormal.hh>
9 #include <core/calc/statistics/Combined_1D_2D_Normal.hh>
10 #include <core/calc/statistics/TrivariateNormal.hh>
11 #include <core/calc/statistics/expectation_maximization.hh>
12 #include <utils/options/OptionParser.hh>
13
14 std::string program_info = R"(
15
16 Example showing how to use expectation-maximization method
17 retrieve arbitrary data according to a sequence alignment object
18
19 USAGE:
20     ./ex_expectation_maximization
21
22 REFERENCE:
23     Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin.
24     "Maximum likelihood from incomplete data via the EM algorithm."
```

(continues on next page)

(continued from previous page)

```

25 Journal of the Royal Statistical Society: Series B 39 (1977): 1-22. doi: 10.1111/j.
26 ↪2517-6161.1977.tb01600.x
27 )
28
29 /** @brief Example showing how to use expectation-maximization method
30 *
31 * CATEGORIES: core::calc::statistics::NormalDistribution; ↪
32 ↪core::calc::statistics::BivariateNormal;core/calc/statistics/expectation_
33 ↪maximization.hh
34 * KEYWORDS: estimation; expectation-maximization
35 */
36 int main(const int argc, const char* argv[]) {
37
38     if ((argc > 1) && utils::options::call_for_help(argv[1]))
39         utils::exit_OK_with_message(program_info);
40
41     using namespace core::calc::statistics;
42
43     double rd = 9876543;
44     std::mt19937 gen(rd);
45     core::index4 N = 10000; //--- the number of random points to use in tests
46
47     // ----- a few distributions to play with
48     double ave1 = 2.0, ave2 = 4.0, ave3 = 6.0;
49     double std1 = 0.2, std2 = 0.3, std3 = 0.5;
50     std::normal_distribution<> m(ave1, std1);
51     std::normal_distribution<> p(ave2, std2);
52     std::normal_distribution<> t(ave3, std3);
53
54     // ----- First let's solve a 1D problem : mixture of 3 Gaussians
55     std::vector<std::vector<double>> random_points;
56     std::vector<double> r1(1), r2(1), r3(1);
57     for (core::index4 i = 0; i < N; ++i) {
58         r1[0] = (m(gen));
59         r2[0] = (p(gen));
60         r3[0] = (t(gen));
61         random_points.push_back(r1);
62         random_points.push_back(r2);
63         random_points.push_back(r3);
64     }
65
66     std::vector<core::calc::statistics::NormalDistribution> distributions_1D;
67     std::vector<core::index1> index_1D; // --- Distribution assignment computed by EM
68     ↪will be stored here
69     core::calc::statistics::NormalDistribution d1(ave1, std1), d2(ave2, std2), d3(ave3,
70     ↪std1);
71     distributions_1D.push_back(d1);
72     distributions_1D.push_back(d2);
73     distributions_1D.push_back(d3);
74
75     std::cout << "1D distributions: starting params\n"; //print
76     ↪parameters_ of all distributions
77     for (const auto & d : distributions_1D) std::cout << d << "\n";
78     double score = expectation_maximization(random_points, distributions_1D, index_1D,
79     ↪true);
80     std::cout << "1D distributions: resulting params\n"; //print
81     ↪parameters_ of all distributions

```

(continues on next page)

(continued from previous page)

```

75   for (const auto & d : distributions_1D) std::cout << d << "\n";
76   std::cout << "\n";
77
78 // ----- now a mixture of 2 Gaussians in 2D
79 std::vector<core::calc::statistics::BivariateNormal> distributions_2D;
80 std::vector<core::index1> index_2D;
81 core::calc::statistics::BivariateNormal d4(ave2, ave3, std2, std3, 0.1), d5(ave3,
82 ↪ave2, std3, std2, 0.1);
83 distributions_2D.push_back(d4);
84 distributions_2D.push_back(d5);
85 std::vector<std::vector<double> > data_2D;
86 std::vector<double> rr1(2), rr2(2);
87 for (core::index4 i = 0; i < N; ++i) {
88     rr1[0] = (p(gen));
89     rr1[1] = (t(gen));
90     rr2[0] = (t(gen));
91     rr2[1] = (p(gen));
92     data_2D.push_back(rr1);
93     data_2D.push_back(rr2);
94 }
95
96 std::cout << "2D distributions: starting params\n";           //print
97 ↪parameters_ of all distributions
98 for (const auto & d : distributions_2D) std::cout << d << "\n";
99 expectation_maximization(data_2D, distributions_2D, index_2D, true);
100 std::cout << "2D distributions: resulting params\n";          //print
101 ↪parameters_ of all distributions
102 for (const auto & d : distributions_2D) std::cout << d << "\n";
103 }
```



ex_find_side_group

Reads a PDB file and prints names of all atoms in each residue side chain. The find_side_group() function, tested by this program, creates a molecular graph to detect a side chain and returns copies side chain atoms on a vector.

USAGE:

```
ex_find_side_group 2gb1.pdb
```

Keywords:

- data_structures
- *graphs*
- residue side chains

Categories:

- core/chemical/find_side_group

Input files:

- 2gb1.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <iomanip>
3 #include <core/data/io/Pdb.hh>
4 #include <core/chemical/Molecule.hh>
5 #include <core/chemical/molecule_utils.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Reads a PDB file and prints names of all atoms in each residue side chain.
11 The find_side_group() function, tested by this program, creates a molecular graph to
12 ↩detect a side chain and returns
13 copies side chain atoms on a vector.
14
15 USAGE:
16     ex_find_side_group 2gb1.pdb
17 )
18 /**
19  * @brief A simple example shows how to select a chemical group of a molecule using
20  * ↩find_side_group() method.
```

(continues on next page)

(continued from previous page)

```

20
21 * This example prints atoms for each side chain in a protein
22 * CATEGORIES: core/chemical/find_side_group;
23 * KEYWORDS: data_structures;graphs ; residue side chains
24 */
25 int main(const int argc, const char* argv[]) {
26
27     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
28     ↪missing program parameter
29
30     using namespace core::data::structural;
31     core::data::io::Pdb reader(argv[1],core::data::io::is_not_alternative); // file_
32     ↪name (PDB format, may be gzip-ped)
33     Structure_SP strctr = reader.create_structure(0); // create a Structure object from_
34     ↪the first deposit found in the input file
35
36     // --- Here we create a molecule object; 0.1 is the tolerance for bond lengths_
37     ↪(used to detect bonds)
38     auto molecule_sp = core::chemical::create_molecule(strctr->first_atom(),strctr->
39     ↪last_atom(),0.1);
40
41     // --- Iterate over all residues in the structure
42     for(auto res_it = strctr->first_residue();res_it!=strctr->last_residue();++res_it) {
43         auto ca = (*res_it)->find_atom(" CA "); // alpha carbon is the preceding atom
44         auto cb = (*res_it)->find_atom(" CB "); // beta carbon is the atom where a side_
45         ↪chain is attached
46         if((ca== nullptr) || (cb== nullptr)) continue;
47         std::vector<PdbAtom_SP> sc;
48         core::chemical::find_side_group<PdbAtom_SP>(ca,cb,*molecule_sp,sc);
49         std::cout << utils::string_format("%4d %s : ",(*res_it)->id(),(*res_it)->residue_
        ↪type().code3.c_str());
50         for(const PdbAtom_SP & a : sc)
51             std::cout << " " << a->atom_name();
52         std::cout << "\n";
53     }
54 }
```



ex_goodman_kruskal_rank_correlation

The program read a contingency matrix from a file and calculates Goodman and Kruskal's gamma parameters which is a measure of rank correlation.

USAGE:

```
ex_goodman_kruskal_rank_correlation input_contingency_matrix_file
```

EXAMPLE:

```
ex_goodman_kruskal_rank_correlation contingency_matrix.txt
```

REFERENCE: Kruskal, William H., and Leo Goodman. "Measures of association for cross classifications." Journal of the American Statistical Association 49 (1954): 732-764. doi:10.2307/2281536.

Keywords:

- *statistics*
- *data table*

Categories:

- core::calc::statistics::goodman_kruskal_rank_correlation

Input files:

- example_contingency_matrix.txt

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/data/basic/Array2D.hh>
4 #include <core/calc/statistics/simple_statistics.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 The program read a contingency matrix from a file and calculates Goodman and Kruskal
10 ↵'s gamma parameters
11 which is a measure of rank correlation.
12
13 USAGE:
14     ex_goodman_kruskal_rank_correlation input_contingency_matrix_file
15
16 EXAMPLE:

```

(continues on next page)

(continued from previous page)

```
16     ex_goodman_kruskal_rank_correlation contingency_matrix.txt
17
18 REFERENCE:
19 Kruskal, William H., and Leo Goodman. "Measures of association for cross_
20   ↪classifications."
21 Journal of the American Statistical Association 49 (1954): 732–764. doi:10.2307/
22   ↪2281536.
23
24 )";
25
26 /** @brief Calculates Goodman and Kruskal's gamma parameters
27 *
28 * CATEGORIES: core::calc::statistics::goodman_kruskal_rank_correlation;
29 * KEYWORDS: statistics; data table
30 */
31 int main(const int argc, const char* argv[]) {
32
33     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
34   ↪missing program parameter
35
36     using core::data::basic::Array2D;
37
38     // --- Read an input file - data table format
39     Array2D<core::index4> m = Array2D<core::index4>::from_file(argv[1]);
40     std::cout << core::calc::statistics::goodman_kruskal_rank_correlation(m) << "\n";
41 }
```



ex_greedy_clustering

Example showing how to use greedy clustering method.

Keywords:

- *clustering*

Categories:

- core::calc::clustering::greedy_clustering()

Output files:

- stdout.out

Program source:

```
1 #include <vector>
2 #include <iostream>
3 #include <random>
4
5 #include <core/calc/clustering/greedy_clustering.hh>
6
7 /// A distance operator calculates the distance between two points indexed by <code>i
8 // and <code>j</code>
9 struct PointDistance {
10     std::vector<double> & points;
11
12     /// Constructor just copies the reference of a data vector
13     PointDistance(std::vector<double> & pts) : points(pts) {}
14     /// Call-operator computes the distance
15     double operator()(const size_t i, const size_t j) const { return fabs(points[i]-
16         points[j]); }
17
18 /**
19 * @brief Example showing how to use greedy clustering method.
20 */
21 * CATEGORIES: core::calc::clustering::greedy_clustering()
22 * KEYWORDS: clustering
23 */
24 int main(const int argc, const char* argv[]) {
25
26     // --- Prepare random number generators
27     std::mt19937 gen(1234567);
28     std::normal_distribution<> d1(10.5, 2.0);
29     std::normal_distribution<> d2(-0.5, 2.0);
30     std::vector<double> data;
31
32     // --- Generate 20 random values
33     for (unsigned short i = 0; i < 10; ++i) {
34         data.push_back(d1(gen));
35         data.push_back(d2(gen));
36     }
```

(continues on next page)

(continued from previous page)

```
34 }
35
36 std::vector<size_t> clusters; // --- Clusters will be stored here
37 std::vector<size_t> cluster_members; // --- vector for members assigned to clusters
38 PointDistance distance(data); // --- instance of the distance operator
39 core::calc::clustering::greedy_clustering(data,distance,5.0,clusters,cluster_
40 ←members);
41
42 // --- Show results
43 std::cout << "n_clusters: " << clusters.size() << "\n";
44 std::cout << "cluster assignment: ";
45 for (const unsigned short i : cluster_members) std::cout << i << " ";
46 std::cout << "\n";
47 }
```



ex_hssp_to_fasta

Simple test which reads a Multiple Sequence Alignment in the HSSP file format and writes it in the FASTA format.

USAGE:

```
./ex_hssp_to_fasta input.hssp
```

EXAMPLE:

```
./ex_hssp_to_fasta 1crn.hssp
```

REFERENCE: Soding, J and Biegert, A and Lupas, A. N., “The HHpred interactive server for protein homology detection and structure prediction.” Nucleic acids research (2005) 33 W244–W248

Keywords:

- *sequence alignment*
- *FASTA*
- HSSP

Categories:

- core:::data::io::hssp_io

Input files:

- 1dm1.hssp

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/data/io/hssp_io.hh>
4 #include <core/data/io/fastrn_io.hh>
5 #include <utils/exit.hh>
6 #include <utils/options/OptionParser.hh>
7
8 std::string program_info = R"(
9
10 Simple test which reads a Multiple Sequence Alignment in the HSSP file format and
11 ↪writes it
12 in the FASTA format.
13
14 USAGE:
15     ./ex_hssp_to_fasta input.hssp
16 EXAMPLE:

```

(continues on next page)

(continued from previous page)

```
16     ./ex_hssp_to_fasta lcrn.hssp
17
18 REFERENCE:
19 Soding, J and Biegert, A and Lupas, A. N.,
20 "The HHpred interactive server for protein homology detection and structureu
→prediction."
21 Nucleic acids research (2005) 33 W244--W248
22 ) ";
23
24 /** @brief Reads an MSA in HSSP format and writes a FASTA file.
25 *
26 *
27 * USAGE:
28 *      ex_hssp_to_fasta lcrn.pdb
29 *
30 * CATEGORIES: core:::data:::io:::hssp_io;
31 * KEYWORDS: sequence alignment; FASTA; HSSP
32 * GROUP: File processing; Format conversion
33 */
34 int main(const int argc, const char *argv[]) {
35
36     if ((argc > 1) && utils::options::call_for_help(argv[1]))
37         utils::exit_OK_with_message(program_info);
38
39     using namespace core:::data:::io;
40     std::vector<std::shared_ptr<core:::data:::sequence:::Sequence>> sink;
41     core:::data:::io:::read_hssp_file(argv[1], sink);
42     for(const auto & seq:sink) {
43         std::cout << create_fasta_string(seq->header(), seq->sequence) << "\n";
44     }
45 }
```



ex_intersect_sorted

Unit test shows how to find an intersection of two sorted vectors of data

USAGE:

```
./ex_intersect_sorted
```

Keywords:

- *algorithms*
- *data structures*

Categories:

- core/algorithms/basic_algorithms.hh

Output files:

- stdout.out

Program source:

```
1 #include <vector>
2 #include <iostream>
3 #include <iterator>
4 #include <core/algorithms/basic_algorithms.hh>
5 #include <utils/options/OptionParser.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Unit test shows how to find an intersection of two sorted vectors of data
11
12 USAGE:
13 ./ex_intersect_sorted
14
15 )";
16
17
18 /** @brief Shows how to find an intersection of two sorted vectors of data
19 *
20 * CATEGORIES: core/algorithms/basic_algorithms.hh
21 * KEYWORDS: algorithms; data structures
22 */
23 int main(const int argc, const char* argv[]) {
24
25     if ((argc > 1) && utils::options::call_for_help(argv[1]))
26         utils::exit_OK_with_message(program_info);
27
28     std::vector<int> range1({1,2,3,5,6,7}), range2({5,6,7,8,9,10}), repeated;
```

(continues on next page)

(continued from previous page)

```
30 // Note that both <code>rangel</code> and <code>range2</code> are already sorted!
31 core::algorithms::intersect_sorted(rangel.begin(), rangel.end(), range2.begin(),_
32 →range2.end(), repeated);
33 // Print the element found as the intersection between the two ranges
34 std::copy(repeated.begin(), repeated.end(), std::ostream_iterator<int>(std::cout, "_
35 →"));
36     std::cout << "\n";
}
```



ex_local_BBQ_coordinates

Unit test which reads a PDB file and prints local coordinates for side chain atoms. The example uses BBQ local coordinate system definition, based on three subsequent alpha carbon atoms.

USAGE:

```
./ex_local_BBQ_coordinates input.pdb
```

EXAMPLE:

```
./ex_local_BBQ_coordinates 5edw.pdb
```

REFERENCE: D. Gront, S. Kmiecik, A. Kolinski . “Backbone building from quadrilaterals: A fast and accurate algorithm for protein backbone reconstruction from alpha carbon coordinates.” J Comput Chem (2007) 1593-1597. doi:10.1002/jcc.20624

Keywords:

- *PDB input*
- local coordinates

Categories:

- core::calc::structural::transformations::local_BBQ_coordinates

Input files:

- 2gb1.pdb
- 5edw.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <core/calc/structural/transformations/Rototranslation.hh>
5 #include <core/calc/structural/transformations/transformation_utils.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Unit test which reads a PDB file and prints local coordinates for side chain atoms. ↴
11 ↪The example uses
12 BBQ local coordinate system definition, based on three subsequent alpha carbon atoms.

```

(continues on next page)

(continued from previous page)

```

13 USAGE:
14 ./ex_local_BBQ_coordinates input.pdb
15
16 EXAMPLE:
17 ./ex_local_BBQ_coordinates 5edw.pdb
18
19 REFERENCE:
20 D. Gront, S. Kmiecik, A. Kolinski . "Backbone building from quadrilaterals: A fast,
21 ↪and accurate algorithm
22 for protein backbone reconstruction from alpha carbon coordinates."
23 J Comput Chem (2007) 1593–1597. doi:10.1002/jcc.20624
24 )
25
26 /** @brief Reads a PDB file and prints local coordinates for side chain atoms
27 *
28 * CATEGORIES: core::calc::structural::transformations::local_BBQ_coordinates
29 * KEYWORDS: PDB input; local coordinates
30 */
31 int main(const int argc, const char* argv[]) {
32
33     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
34     ↪missing program parameter
35     using namespace core::data::structural;
36     using namespace core::calc::structural::transformations;
37
38     core::data::io::Pdb reader(argv[1]);
39     Structure_SP strctr = reader.create_structure(0);
40
41     Rototranslation_SP rt = nullptr;
42     for(auto a_chain : *strctr) {
43         for (core::index2 i_residue = 1; i_residue < a_chain->count_residues() - 1; ++i_
44         ↪residue) {
45             const Residue & the_residue = (*a_chain)[i_residue];
46             const Residue & prev_residue = (*a_chain)[i_residue - 1];
47             const Residue & next_residue = (*a_chain)[i_residue + 1];
48             try {
49                 rt = local_BBQ_coordinates(*prev_residue.find_atom_safe(" CA "),
50                     *the_residue.find_atom_safe(" CA "), *next_residue.find_atom_safe(" CA "));
51             } catch (utils::exceptions::AtomNotFound ex) {
52                 i_residue+=2;
53                 continue;
54             }
55             Vec3 tmp_atom;
56             for (auto i_atom : the_residue) {
57                 tmp_atom = *i_atom;
58                 rt->apply(*i_atom);
59                 std::cout << i_atom->to_pdb_line() << "\n";
60                 // --- Here we test if the inverse transformation really moves an atom to its
61                 ↪original location
62                 rt->apply_inverse(*i_atom);
63                 if(tmp_atom.distance_to(*i_atom)>0.001)
64                     throw std::runtime_error("Incorrect position after transformation!");
65             }
66         }
67     }
68 }
```



ex_local_coordinates_three_atoms

Unit test which reads a PDB file and prints local coordinates of every atom. For every residue, a local coordinate system (LCS) is constructed based on its N, C-alpha and C atoms. Then the program prints coordinates of all the atoms of that residue defined in the respective LCS.

USAGE:

```
./ex_local_coordinates_three_atoms input.pdb
```

EXAMPLE:

```
./ex_local_coordinates_three_atoms 5edw.pdb
```

Keywords:

- *PDB input*
- local coordinates
- *rototranslation*

Categories:

- core::calc::structural::transformations::local_coordinates_three_atoms

Input files:

- 2gb1.pdb
- 5edw.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <core/calc/structural/transformations/Rototranslation.hh>
5 #include <core/calc/structural/transformations/transformation_utils.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Unit test which reads a PDB file and prints local coordinates of every atom.
11
12 For every residue, a local coordinate system (LCS) is constructed based on its N, C-
13 alpha and C atoms. Then the program prints coordinates of all the atoms of that_
14 residue defined in the respective LCS.
15 )"
```

(continues on next page)

(continued from previous page)

```

14 USAGE:
15   ./ex_local_coordinates_three_atoms input.pdb
16 EXAMPLE:
17   ./ex_local_coordinates_three_atoms 5edw.pdb
18
19 ) ";
20
21 /** @brief Reads a PDB file and prints local coordinates for sidechain atoms
22 *
23 * CATEGORIES: core::calc::structural::transformations::local_coordinates_three_atoms
24 * KEYWORDS: PDB input; local coordinates; rototranslation
25 */
26 int main(const int argc, const char* argv[]) {
27
28   if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
29   ↪missing program parameter
30   using namespace core::data::structural;
31
32   core::data::io::Pdb reader(argv[1]);
33   core::data::structural::Structure_SP strctr = reader.create_structure(0);
34
35   for (auto it_resid = strctr->first_residue(); it_resid != strctr->last_residue(); ↪
36   ↪++it_resid) {
37
38     PdbAtom_SP n = (*it_resid)->find_atom(" N  ");
39     PdbAtom_SP ca = (*it_resid)->find_atom(" CA  ");
40     PdbAtom_SP c = (*it_resid)->find_atom(" C  ");
41
42     if ((n == nullptr) || (ca == nullptr) || (c == nullptr)) {
43       std::cout << "Missing backbone atom\n";
44       continue;
45     }
46
47     core::calc::structural::transformations::Rototranslation_SP rt =
48     core::calc::structural::transformations::local_coordinates_three_atoms(*n, *ca,
49     ↪*c);
50     Vec3 tmp_atom;
51     for (auto i_atom : **it_resid) {
52       tmp_atom = *i_atom;
53       rt->apply(*i_atom);
54       std::cout << i_atom->to_pdb_line() << "\n";
55       // --- Here we test if the inverse transformation really moves an atom to its
56       ↪original location
57       rt->apply_inverse(*i_atom);
58       if(tmp_atom.distance_to(*i_atom)>0.001)
59         throw std::runtime_error("Incorrect position after transformation!");
60     }
61   }
62 }
```



ex_mmCif

Unit test which shows how to read CIF files.

USAGE:

```
ex_Cif file.cif
```

EXAMPLE:

```
ex_Cif AA3.cif
```

Keywords:

- *CIF input*

Categories:

- core/data/io/Cif

Input files:

- 2GB1.cif

Output files:

- stdout.out

Program source:

```

1 #include <core/data/io/Cif.hh>
2 #include <core/data/io/mmCif.hh>
3
4 #include <utils/Logger.hh>
5 #include <utils/LogManager.hh>
6
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 Unit test which shows how to read CIF files.
12
13 USAGE:
14     ex_Cif file.cif
15 EXAMPLE:
16     ex_Cif AA3.cif
17
18 )";
19
20 /**
21 * @brief ex_Cif tests reading CIF files
22 * @CATEGORIES: core/data/io/Cif

```

(continues on next page)

(continued from previous page)

```
23 * KEYWORDS: CIF input
24 */
25 int main(const int argc, const char *argv[]) {
26
27     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
28     ↪missing program parameter
29
30     utils::LogManager::INFO(); // --- INFO is the default logging level; set it to FINE_
31     ↪to see more
32     core::data::io::mmCif reader(argv[1]);
33
34     std::cout<<reader.pdb_code()<<"\n";
35
36     core::data::structural::Structure_SP strc = reader.create_structure(0);
37     for (auto a=strc->first_atom();a!=strc->last_atom();++a)
38         std::cout<< (*a)->to_pdb_line()<<"\n";
}
```



ex_monomer_io

The program converts a monomer structure from CIF format to internal formats used by BioShell. Use it to register your own monomer which is missing in BioShell library. The program is also used to create ‘monomers.txt’ file from BioShell distribution (located in ./data/ directory). In order to do so, download the fresh repository of monomers in CIF format from: <http://ligand-expo.rcsb.org/dictionaries/Components-pub.cif> and run the program. Then replace the released monomers.txt file with the new one

USAGE:

```
./ex_monomer_io -in:::monomers::cif=HEM.cif -out::file=hem.txt  
./ex_monomer_io -in:::monomers::cif=Components-pub.cif
```

Keywords:

- monomers
- option parsing

Categories:

- core/chemical/Monomer; utils/options/OptionParser; utils/options/Option

Input files:

- HEM.cif

Output files:

- stdout.out
- monomers_new.txt

Program source:

```
1 #include <fstream>  
2 #include <iostream>  
3  
4 #include <core/chemical/Monomer.hh>  
5 #include <core/chemical/monomer_io.hh>  
6  
7 #include <utils/options/Option.hh>  
8 #include <utils/options/OptionParser.hh>  
9 #include <utils/options/input_options.hh>  
10 #include <utils/options/output_options.hh>  
11 #include <utils/exit.hh>  
12  
13 using namespace core::chemical;  
14  
15  
16 std::string program_info = R"(  
17
```

(continues on next page)

(continued from previous page)

```

18 The program converts a monomer structure from CIF format to internal formats used by BioShell.
19
20 Use it to register your own monomer which is missing in BioShell library. The program is also used to create
21 'monomers.txt' file from BioShell distribution (located in ./data/ directory). In order to do so, download
22 the fresh repository of monomers in CIF format from:
23
24 http://ligand-expo.rcsb.org/dictionaries/Components-pub.cif
25
26 and run the program. Then replace the released monomers.txt file with the new one
27
28 USAGE:
29 ../ex_monomer_io -in:::monomers::cif=HEM.cif -out::file=hem.txt
30 ../ex_monomer_io -in:::monomers::cif=Components-pub.cif
31
32 ) ";
33
34 /** @brief The program converts a monomer structure from CIF format to internal formats used by BioShell.
35 *
36 * CATEGORIES: core/chemical/Monomer; utils/options/OptionParser; utils/options/Option
37 * KEYWORDS: monomers; option parsing
38 */
39 int main(const int argc, const char* argv[]) {
40
41     using namespace utils::options;
42     utils::options::OptionParser & cmd = OptionParser::get();
43     cmd.register_option(utils::options::help);
44     cmd.register_option(verbose, mute);
45     cmd.register_option(db_path);
46     cmd.register_option(input_bin_monomers, input_cif_monomers, input_txt_monomers);
47     cmd.register_option(output_file);
48     cmd.program_info(program_info);
49
50     if (!cmd.parse_cmdline(argc, argv)) return 1;
51
52     if (input_cif_monomers.was_used()) read\_monomers\_cif\(option\_value<std::string>
53     \(input\_cif\_monomers\)\);
54
55     if (input_txt_monomers.was_used()) read\_monomers\_txt\(option\_value<std::string>
56     \(input\_txt\_monomers\)\);
57
58     if (input_bin_monomers.was_used()) read\_monomers\_binary\(option\_value<std::string>
59     \(input\_bin\_monomers\)\);
60
61     write_monomers_txt(option_value<std::string>(output_file, "monomers.txt"));
62 }

```



ex_pdb_to_fasta

Unit test which reads a PDB file and writes protein sequence(s) in FASTA format. Unlike, ap_pdb_to_fasta_ss.cc application, this doesn't print secondary structure strings

USAGE:

```
./ex_pdb_to_fasta input.pdb
```

EXAMPLE:

```
./ex_pdb_to_fasta 5edw.pdb
```

Keywords:

- *PDB input*

Categories:

- core::data::io::Pdb

Input files:

- 5edw.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <core/data/structural/Structure.hh>
5 #include <utils/options/OptionParser.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Unit test which reads a PDB file and writes protein sequence(s) in FASTA format.
11 Unlike, ap_pdb_to_fasta_ss.cc application, this doesn't print secondary structure_
12 →strings
13
14 USAGE:
15     ./ex_pdb_to_fasta input.pdb
16 EXAMPLE:
17     ./ex_pdb_to_fasta 5edw.pdb
18 )
19
20 /**
21 * @brief Reads a PDB file and writes protein sequence(s) in FASTA format.
22 */

```

(continues on next page)

(continued from previous page)

```
21
22 * This is a simplified version of ap_pdb_to_fasta_ss.cc application
23 * USAGE:
24 *     ex_pdb_to_fasta 5edw.pdb
25 *
26 * CATEGORIES: core:::data:::io:::Pdb
27 * KEYWORDS: PDB input
28 * GROUP: File processing; Format conversion
29 */
30 int main(const int argc, const char *argv[]) {
31
32     if ((argc > 1) && utils::options::call_for_help(argv[1]))
33         utils::exit_OK_with_message(program_info);
34
35     using namespace core:::data:::io; // Pdb and create_fasta_string lives there
36
37     Pdb reader(argv[1], is_not_alternative, only_ss_from_header, true);
38     core:::data:::structural::Structure_SP strctr = (reader.create_structure(0));
39
40     // Iterate over all chains
41     for (int ic = 0; ic < strctr->count_chains(); ++ic)
42         std::cout << "> " << strctr->code() << (*strctr)[ic]->id() << "\n" // --- e.g. ↵
43         prints "> 2gb1 A"
44         << (*strctr)[ic]->create_sequence()->sequence << "\n";           // --- prints ↵
        the sequence itself
    }
```



ex_peptide_hydrogen

ex_peptide_hydrogen reconstructs peptide hydrogen atoms using BioShell algorithm, where amide H is placed in reference to its N atom. Resulting coordinates are printed on the screen. The program also computes the amide-H positions using DSSP approach and calculates the average error (in Angstroms) between the two methods.

USAGE:

```
ex_peptide_hydrogen input.pdb
```

EXAMPLE:

```
ex_peptide_hydrogen 5edw.pdb
```

REFERENCE: Kabsch, Wolfgang, and Christian Sander. "Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features." *Biopolymers* 22 (1983): 2577-2637. doi:10.1002/bip.360221211

Keywords:

- *PDB input*
- hydrogen reconstruction

Categories:

- core::calc::structural::peptide_hydrogen()

Input files:

- 2gb1.pdb
- 5edw.pdb
- 3dcg.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <core/data/structural/selectors/structure_selectors.hh>
5 #include <core/calc/structural/interactions/BackboneHBondCollector.hh>
6 #include <utils/exit.hh>
7
8 using namespace core::data::structural;
9 using namespace core::data::io;
10 using namespace core::data::basic;
11
12 std::string program_info = R" (
```

(continues on next page)

(continued from previous page)

```

13
14 ex_peptide_hydrogen reconstructs peptide hydrogen atoms using BioShell algorithm,
15 where amide H is placed in reference to its N atom. Resulting coordinates are printed
16 on the screen. The program also computes the amide-H positions using DSSP approach
17 and calculates the average error (in Angstroms) between the two methods.
18
19 USAGE:
20   ex_peptide_hydrogen input.pdb
21
22 EXAMPLE:
23   ex_peptide_hydrogen 5edw.pdb
24
25 REFERENCE:
26 Kabsch, Wolfgang, and Christian Sander. "Dictionary of protein secondary structure:  

27 ↪pattern recognition  

28 of hydrogen-bonded and geometrical features." Biopolymers 22 (1983): 2577-2637.  

29 ↪doi:10.1002/bip.360221211
30
31 ) ";
32
33 /**
34 * @brief Reconstructs peptide hydrogen atoms using two methods and compares the
35 * error between them.
36 * @CATEGORIES: core::calc::structural::peptide_hydrogen()
37 * @KEYWORDS: PDB input; hydrogen reconstruction
38 */
39
40 int main(const int argc, const char* argv[]) {
41   if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
42   ↪missing program parameter
43   core::data::io::Pdb reader(argv[1], all_true(is_not_alternative, is_not_water)); //  

44   ↪--- Read in a PDB file
45   core::data::structural::Structure_SP strctr = reader.create_structure(0); // ---  

46   ↪create a Structure object from the first model
47
48   double err = 0, n = 0;
49   auto res_it = ++(strctr->first_residue()); // --- The residue being reconstructed
50   auto prev_res_it = strctr->first_residue(); // --- preceding residue
51   for (; res_it != strctr->last_residue(); ++res_it) {
52     std::cerr << "# reconstructing:" << (**prev_res_it) << " and " << (res_it) <<  

53     ↪"\n";
54     if (((*prev_res_it)->residue_type().parent_id > 20) || ((*res_it)->residue_type().  

55     ↪parent_id > 20)) { // --- its not an amino acid
56       ++prev_res_it;
57       continue;
58     }
59     try {
60       if((*prev_res_it)->owner() != (*res_it)->owner()) {
61         std::cerr << "# Chain break between residues:" << (**prev_res_it) << " and "  

62         << (*res_it) << "\n";
63         ++prev_res_it;
64         continue;
65       }
66       // --- Rebuild the peptide hydrogen in a residue pointed by res_it iterator.
67       // --- This method actually adds the newly created H atom to the residue
68       core::calc::structural::interactions::peptide_hydrogen(*prev_res_it, *res_it);
69       auto new_H = (*res_it)->find_atom_safe(" H ");
70       // --- Here we reconstruct amide H knowing the relevant atoms, but now
71       ↪according to the DSSP approach. Resulting H is not inserted

```

(continues on next page)

(continued from previous page)

```
60     auto prev_O = (*prev_res_it)->find_atom_safe(" O ");
61     auto prev_C = (*prev_res_it)->find_atom_safe(" C ");
62     auto this_N = (*res_it)->find_atom_safe(" N ");
63     PdbAtom other_H;
64     core::calc::structural::interactions::peptide_hydrogen_dssp(*prev_C, *prev_O,_
65     ↪*this_N, other_H);
66     err += other_H.distance_to(*new_H);
67     n++;
68     for (const auto &atom : **res_it)
69       std::cout << atom->to_pdb_line() << "\n"; //-- print all atoms in the current_
70     ↪residue (in PDB format)
71     ++prev_res_it; // --- advance one of the iterators by one residue; the other_
72     ↪iterator is advanced by the loop
73   } catch (utils::exceptions::AtomNotFound e) {
74     std::cerr << e.what() << "\n";
75     ++prev_res_it;
76   }
77   std::cout << "# difference between two methods: " << err / n << "\n";
78 }
```



ex_protein_peptide_interface

ex_protein_peptide_interface finds atomic contacts between a receptor and a peptide found in an input PDB file. The peptide is defined as a protein chain shorter than 35 residues, while the receptor must consist of at least 40 amino acids. Output provides: protein residue name and ID, protein chain ID, peptide protein name and ID, peptide chain ID, minimum distance between the residues, e.g.: ILE 36 A ARG 104 X 5.92977 LEU 44 A ARG 104 X 5.92685 LEU 44 A LEU 108 X 5.57779 GLU 45 A THR 102 X 6.81994

USAGE:

```
ex_protein_peptide_interface file.pdb cutoff-distance
```

EXAMPLE:

```
ex_protein_peptide_interface 1dt7.pdb 7.0
```

where 1dt7.pdb id an input file and 7.0 - contact distance in Angstroms.

Keywords:

- *PDB input*
- *contact map*
- peptide
- *STL*

Categories:

- core::data::structural::Structure

Input files:

- 1dt7.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <map>
3
4 #include <core/data/io/Pdb.hh>
5 #include <core/data/structural/selectors/structure_selectors.hh>
6
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 ex_protein_peptide_interface finds atomic contacts between a receptor and a peptide_
12 ↴found in an input PDB file.
```

(continues on next page)

(continued from previous page)

```

12 The peptide is defined as a protein chain shorter than 35 residues, while the receptor must consist
13 of at least 40 amino acids.
14
15 Output provides: protein residue name and ID, protein chain ID, peptide protein name and ID,
16 peptide chain ID, minimum distance between the residues, e.g.:
17
18 ILE 36 A ARG 104 X 5.92977
19 LEU 44 A ARG 104 X 5.92685
20 LEU 44 A LEU 108 X 5.57779
21 GLU 45 A THR 102 X 6.81994
22
23 USAGE:
24   ex_protein_peptide_interface file.pdb cutoff-distance
25
26 EXAMPLE:
27   ex_protein_peptide_interface 1dt7.pdb 7.0
28
29 where 1dt7.pdb id an input file and 7.0 - contact distance in Angstroms.
30
31 ) ";
32
33 unsigned int MAX_PEPTIDE_LENGTH = 35;
34 unsigned int MIN_PROTEIN_LENGTH = 40;
35
36 /** @brief Finds contacts atomic contacts between a receptor and a peptide.
37 * *
38 * CATEGORIES: core::data::structural::Structure
39 * KEYWORDS: PDB input; contact map; peptide; STL
40 */
41 int main(const int argc, const char* argv[]) {
42
43   if (argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
44
45   using namespace core::data::io;
46   Pdb reader(argv[1], all_true(is_not_water, is_not_alternative, is_not_hydrogen)); // -- file name (PDB format, may be gzip-ped)
47
48   core::data::structural::Structure_SP strctr = reader.create_structure(0);
49   core::data::structural::selectors::IsAA is_aaTester;
50   core::data::structural::Structure_SP sub_strctr = strctr; // = strctr->clone(is_aaTester);
51
52   double cutoff = utils::from_string<double>(argv[2]); // The second parameter is the contact distance (in Angstroms)
53
54   for (auto protein_chain_sp: *strctr) { // --- protein_chain_sp is a shared pointer to a chain
55     if (protein_chain_sp->size() < MIN_PROTEIN_LENGTH) continue;
56     for (auto i_residue_sp: *protein_chain_sp) { // --- i_residue_sp is a shared pointer to a residue
57       for (auto peptide_chain_sp: *strctr) {
58         if (peptide_chain_sp->size() > MAX_PEPTIDE_LENGTH) continue;
59         for (auto j_residue_sp: *peptide_chain_sp) {
60           double d = (i_residue_sp)->min_distance(j_residue_sp);

```

(continues on next page)

(continued from previous page)

```
61     if (d < cutoff)
62         std::cout << (*i_residue_sp) << " " << (*i_residue_sp).owner()->id() << "
63             << (*j_residue_sp) << " "
64             << (*j_residue_sp).owner()->id() << " " << d << "\n";
65     }
66 }
67 }
68 }
```



ex_ramachandran_kd_tree

ex_ramachandran_kd_tree partitions observations from a Ramachandran map

USAGE:

```
ex_ramachandran_kd_tree phi_psi.dat n_level width
```

where phi_psi.dat is an input file with two columns of data (Phi and Psi angles), width - width of a square range for counting neighbors and n_level - maximum level on the kd-tree to assign.

The output consists of lines as below: -65.08 125.25 15 684 where the first two columns contain the Phi,Psi angles, respectively, that have been loaded from the input file. The third value (15 in the example) is the number of a rectangular area resulting from the 2D-tree construction. Finally 684 is the number of points found in a square area width x width centered at the given point. That value provides in insight how probable is a given Phi, Psi observation

Keywords:

- neighborhood detection
- *data structures*
- *algorithms*

Categories:

- core::algorithms::trees::BinaryTreeNode; core::algorithms::trees::kd_tree.hh

Input files:

- val-rama.dat

Output files:

- stdout.out

Program source:

```
1 #include <memory>
2 #include <iostream>
3 #include <random>
4
5 #include <core/algorithms/trees/kd_tree.hh>
6 #include <core/algorithms/trees/BinaryTreeNode.hh>
7 #include <core/algorithms/trees/algorithms.hh>
8 #include <core/data/io/DataTable.hh>
9 #include <utils/exit.hh>
10
11 std::string program_info = R"(
12
13 ex_ramachandran_kd_tree partitions observations from a Ramachandran map
14
15 )"
```

(continues on next page)

(continued from previous page)

```

16 USAGE:
17   ex_ramachandran_kd_tree phi_psi.dat n_level width
18
19 where phi_psi.dat is an input file with two columns of data (Phi and Psi angles),
20 width - width of a square range for counting neighbors and n_level - maximum level on
21 ↴the kd-tree to assign.
22
23 The output consists of lines as below:
24   -65.08 125.25 15 684
25 where the first two columns contain the Phi,Psi angles, respectively, that have been
26 ↴loaded from the input file.
27 The third value (15 in the example) is the number of a rectangular area resulting
28 ↴from the 2D-tree construction.
29 Finally 684 is the number of points found in a square area width x width centered at
30 ↴the given point. That value
31 provides in insight how probable is a given Phi, Psi observation
32
33 ) ";
34
35 using namespace core::algorithms::trees;
36
37
38 class Point: public std::pair<float, float> {
39 public:
40
41     Point(float phi, float psi) : std::pair<float, float>(phi, psi) {}
42
43     float operator[](const size_t k) const { if (k == 0) return first; else return
44 ↴second; }
45 };
46
47 /**
48  * Operation that computes the distance between points on Ramachandran map
49  */
50 struct PhiPsiDistance {
51
52     /**
53      * This operation will be called at every tree node considered during a tree
54      * traversal
55      * @param n1, n2 are the points to be compared
56      * @return the distance between the two points
57      */
58     float operator() (const Point & n1, const Point & n2) const {
59         float d = (n1.first - n2.first);
60         float d2 = d * d;
61         d = (n1.second - n2.second);
62         d2 += d * d;
63         return sqrt(d2);
64     }
65 };
66
67 /**
68  * Tree traversal operation prints a given point along with its node level
69  * and the number of neighbors
70  */
71 struct PrintPoint {
72
73     PrintPoint(std::shared_ptr<BinaryTreeNode<KDTreeNode<Point>>> root, float width) :
74         root_(root), w_(width / 2.0) {}
75
76     /**
77      * this operator prints a visited node on the screen
78      * @param node is the node to be printed
79      */
80     void operator()(std::shared_ptr<BinaryTreeNode<KDTreeNode<Point>>> node) {
81
82         Point q_low(node->element.element[0] - w_, node->element.element[1] - w_);
83         Point q_up(node->element.element[0] + w_, node->element.element[1] + w_);
84     }
85 }
```

(continues on next page)

(continued from previous page)

```

65     std::vector<Point> hits;
66     search_kd_tree(root_, q_low, q_up, 2, hits);
67
68     std::cout << utils::string_format("%7.2f %7.2f %3d %4d\n",
69                           node->element.element.first, node->element.element.element.second, node->
70                           element.level, hits.size());
71 }
72
73 private:
74     std::shared_ptr<BinaryTreeNode<KDTreeNode<Point>>> root_;
75     float w_;
76 }
77
78 /** @brief A simple example shows how to use BioShell kd-tree routines.
79 *
80 * The program reads a file with Phi, Psi observations and partitions them in a kd-
81 * tree.
82 *
83 * CATEGORIES: core:::algorithms::trees::BinaryTreeNode; core:::algorithms::trees::kd_
84 * -tree.hh
85 * KEYWORDS: neighborhood detection; data structures; algorithms
86 */
87 int main(const int argc, const char* argv[]) {
88
89     if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
90     missing program parameters
91
92     core::index2 n_level = (argc > 2) ? atoi(argv[2]) : 4;
93     core::index2 width = (argc > 3) ? atof(argv[3]) : 5;
94     // ----- First we read a file with Phi, Psi observations
95     core::data::io::DataTable dt;
96     dt.load(argv[1]);
97     std::vector<Point> points; // container for the points
98     for(const auto & row:dt)
99         points.emplace_back(row.get<float>(0), row.get<float>(1));
100
101    // ----- Here the actual kd-tree is constructed
102    std::shared_ptr<BinaryTreeNode<KDTreeNode<Point>>> root =
103        create_kd_tree<Point, std::vector<Point>::iterator, CompareAsReferences<Point>>
104        (points.begin(), points.end(), 2);
105
106    std::vector<std::shared_ptr<BinaryTreeNode<KDTreeNode<Point>>>> node_group_
107    representatives;
108    collect_given_level(root, n_level, node_group_representatives, 0);
109    core::index2 group_id = 0;
110    for(const auto node:node_group_representatives) {
111        depth_first_preorder(node, [group_id](std::shared_ptr<BinaryTreeNode<KDTreeNode
112        <Point>>> node) {
113            node->element.level = group_id; });
114        ++group_id;
115    }
116
117    // ----- Here we print each node
118    PrintPoint pp(root, width);
119    breadth_first_preorder(root, pp); // finally, each node is printed
120
121 }
```



ex_random_vector_on_sphere

Simple test shows that random_vector_on_sphere() really produces a uniform distribution

Keywords:

- no_keywords

Categories:

- simulations/movers/random_vector_on_sphere

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <core/data/basic/Vec3.hh>
4
5 #include <simulations/movers/movers_utils.hh>
6
7 using namespace core::data::basic;
8 using namespace simulations;
9
10 /** @brief Simple test shows that random_vector_on_sphere() really produces a uniform_
11  ↪distribution
12  *
13  * CATEGORIES: simulations/movers/random_vector_on_sphere;
14  */
15 int main(int argc, char *argv[]) {
16
17     core::data::basic::Vec3 v;
18     core::data::basic::Vec3 s;
19     core::index4 N = 100000;
20     for (size_t i = 0; i < N; ++i) {
21         simulations::movers::random_vector_on_sphere(v);
22         s += v;
23         std::cout << v << "\n";
24     }
25     s /= N;
26     std::cout << "# sum: " << s << "\n";
}
```



ex_read_properties_file

Simple test for ex_read_properties_file function reads a file given from command line. The program expects a file in JAVA's .properties file format

USAGE:

```
ex_read_properties_file input_file.properties
```

REFERENCE: <https://en.wikipedia.org/wiki/.properties>

Keywords:

- file utils
- properties file

Categories:

- utils/read_properties_file

Input files:

- input.properties

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2
3 #include <utils/io_utils.hh>
4 #include <utils/exit.hh>
5
6 std::string program_info = R"(
7
8 Simple test for ex_read_properties_file function reads a file given from command line.
9 The program expects a file in JAVA's .properties file format
10
11 USAGE:
12     ex_read_properties_file input_file.properties
13
14 REFERENCE:
15 https://en.wikipedia.org/wiki/.properties
16
17 )";
18
19 /** @brief Simple test reads .properties file and prints these settings on the screen
20 *
21 *  CATEGORIES: utils/read_properties_file
```

(continues on next page)

(continued from previous page)

```

22 * KEYWORDS: file utils;properties file
23 */
24 int main(const int argc, const char* argv[]) {
25
26     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
27     ↴missing program parameter
28
29     for (int i = 1; i < argc; ++i) {
30         // In the single line below we read the properties file
31         auto mapa = utils::read_properties_file(argv[i]);
32
33         // Here we print the content of the map in the same format (i.e. .properties)
34         for (auto it = mapa.cbegin(); it != mapa.cend(); ++it) {
35             std::cout << it->first << " : ";
36             for (auto it2 = it->second.cbegin(); it2 != it->second.cend(); ++it2)
37                 std::cout << (*it2) << " ";
38             std::cout << "\n";
39         }
40     }

```



ex_selection_protocols

Simple test shows how to use AtomSelector from selection protocols set. As an example, selects atoms that belong to nucleic acid residues.

USAGE:

```
ex_selection_protocols 5edw.pdb
```

Keywords:

- *PDB input*
- Selection protocols
- *structure selectors*

Categories:

- core::protocols::keep_selected_atoms()

Input files:

- 5edw.pdb
- 3dcg.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <sstream>
3 #include <core/data/io/Pdb.hh>
4 #include <core/data/structural/selectors/structure_selectors.hh>
5 #include <core/protocols/selection_protocols.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Simple test shows how to use AtomSelector from selection protocols set.
11 As an example, selects atoms that belong to nucleic acid residues.
12
13 USAGE:
14     ex_selection_protocols 5edw.pdb
15
16 )";
17
18 /**
19 * @brief Shows how to use selection protocols functions
20 * @CATEGORIES: core::protocols::keep_selected_atoms()

```

(continues on next page)

(continued from previous page)

```
21 * KEYWORDS: PDB input; Selection protocols; structure selectors
22 */
23 int main(const int argc, const char* argv[]) {
24
25     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
26     ↪missing program parameter
27
28     using namespace core::data::structural;
29     using namespace core::data::structural::selectors;
30     using namespace core::protocols;
31
32     core::data::io::Pdb reader(argv[1]);
33
34     { // --- section which tests selecting nucleotides
35         Structure_SP strctr = reader.create_structure(0);
36
37         std::shared_ptr<AtomSelector> select_nt = std::make_shared<IsNT>();
38
39         keep_selected_atoms(*select_nt, *strctr);
40         for (auto chain_sp : *strctr)
41             std::cout << utils::string_format("\tchain %s has %3d residues satisfying the
42         ↪selector\n", chain_sp->id().c_str(),
43             chain_sp->size());
44     }
45 }
```



ex_seq_io

Unit test which reads a SEQ file and prints it's content in FASTA format.

USAGE:

```
./ex_seq_io SEQ-file
```

EXAMPLE:

```
./ex_seq_io 2gb1.seq
```

Keywords:

- *sequence*
- *FASTA output*
- *secondary structure*
- *Format conversion*

Categories:

- core/data/io/read_seq

Input files:

- 2gb1.seq

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <core/data/io/fasta_io.hh>
3 #include <core/data/io/seq_io.hh>
4 #include <utils/exit.hh>
5
6 std::string program_info = R"(
7
8 Unit test which reads a SEQ file and prints it's content in FASTA format.
9
10 USAGE:
11     ./ex_seq_io SEQ-file
12 EXAMPLE:
13     ./ex_seq_io 2gb1.seq
14
15 )";
16
17 /** @brief Example reads SEQ file and prints the data stored there in FASTA format
```

(continues on next page)

(continued from previous page)

```
18 *
19 * CATEGORIES: core/data/io/read_seq
20 * KEYWORDS: sequence; FASTA output; secondary structure; Format conversion
21 */
22 int main(const int argc, const char* argv[]) {
23
24     if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about
25     //missing program parameter
26
27     core::data::sequence::SecondaryStructure_SP ss = core::data::io::read_seq(argv[1],
28     " );
29     ss->header(argv[1]);
30     std::cout << core::data::io::create_fasta_string(*ss, 80) << "\n";
31     std::cout << core::data::io::create_fasta_secondary_string(*ss, 80) << "\n";
32 }
```



ex_set_dihedral

Sets a particular values for Phi, Psi and Omega angles at a certain residue in a protein.

USAGE:

```
ex_set_dihedral res-id file.pdb phi psi omega
```

EXAMPLE:

```
ex_set_dihedral 2gb1.pdb 18 -80.4 90.4 180.0
```

where 2gb1.pdb is the protein structure to be modified, 18 is the residue ID and the three following real values are Phi, Psi and omega dihedrals (in the range [-180.0,180.0]). The results is printed in PDB format

Keywords:

- *PDB input*
- *rototranslation*
- *structural properties*

Categories:

- core/calc/structural/transformations/Rototranslation

Input files:

- 2gb1.pdb

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <cmath>
3
4 #include <core/data/io/Pdb.hh>
5 #include <core/data/structural/Structure.hh>
6 #include <core/data/structural/selectors/structure_selectors.hh>
7
8 #include <core/calc/structural/protein_angles.hh>
9 #include <core/calc/structural/transformations/Rototranslation.hh>
10 #include <utils/exit.hh>
11
12 using namespace core::data::structural;
13 using namespace core::data::io;
14 using namespace core::data::basic;
15
16 std::string program_info = R"(

```

(continues on next page)

(continued from previous page)

```

17
18 Sets a particular values for Phi, Psi and Omega angles at a certain residue in a_
19 ↪protein.
20
21 USAGE:
22   ex_set_dihedral res-id file.pdb phi psi omega
23 EXAMPLE:
24   ex_set_dihedral 2gb1.pdb 18 -80.4 90.4 180.0
25 where 2gb1.pdb is the protein structure to be modified, 18 is the residue ID and the_
26 ↪three following
27 real values are Phi, Psi and omega dihedrals (in the range [-180.0,180.0]). The_
28 ↪results is printed in PDB format
29 )";
30
31 /** @brief Sets a particular values for Phi, Psi and Omega angles at a certain_
32 ↪residue in a protein.
33
34 int main(const int argc, const char *argv[]) {
35
36   if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about_
37 ↪missing program parameter
38
39   // --- The first parameter of the program is the PDB file name
40   core::data::io::Pdb reader(argv[1], is_not_alternative, keep_all, false); // ---
41 ↪Read in a PDB file
42   core::data::structural::Structure_SP strctr = reader.create_structure(0);
43   // --- create a Structure object from the first model and extract the first chain_
44 ↪(indexed as 'A') from it
45   core::data::structural::Chain & chain = *(strctr->get_chain('A'));
46   core::index2 res_idx = utils::from_string<core::index2>(argv[2]);
47   core::calc::structural::transformations::Rototranslation rt;
48
49   // --- Phi rotation; the new value of the Phi angle (in degrees) is the fourth_
50 ↪parameter of this program
51   if (argc > 3 && strlen(argv[3]) > 1) {
52     double phi = core::calc::structural::evaluate_phi(*chain[res_idx-1],*chain[res_
53 ↪idx]);
54     std::cerr << "Phi angle before change: " << phi * 180.0 / M_PI << " degrees\n";
55     phi = utils::from_string<double>(argv[3]) * M_PI / 180.0 - phi;
56     PdbAtom & N = *chain[res_idx]->find_atom_safe(" N ");
57     PdbAtom & CA = *chain[res_idx]->find_atom_safe(" CA ");
58     core::calc::structural::transformations::Rototranslation::around_axis(N, CA, phi,
59 ↪CA, rt);
60     for (core::index2 ires = 0 ; ires < res_idx; ++ires) {
61       for (PdbAtom_SP ai : *chain[ires]) rt.apply(*ai);
62     }
63     if(chain[res_idx]->find_atom(" H ") !=nullptr)
64       rt.apply(*chain[res_idx]->find_atom(" H "));
65   }
66
67   // --- Psi rotation; the new value of the Psi angle (in degrees) is the fifth_
68 ↪parameter of this program
69   if (argc > 4 && strlen(argv[4]) > 1) {
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
746
746
747
748
748
749
749
750
751
752
753
754
755
756
756
757
758
758
759
759
760
761
762
763
764
765
766
766
767
768
768
769
769
770
771
772
773
774
775
776
776
777
778
778
779
779
780
781
782
783
784
785
786
786
787
788
788
789
789
790
791
792
793
794
795
796
796
797
798
798
799
799
800
801
802
803
804
805
806
806
807
808
808
809
809
810
811
812
813
814
815
815
816
817
817
818
818
819
819
820
821
822
823
824
825
825
826
827
827
828
828
829
829
830
831
832
833
834
835
835
836
837
837
838
838
839
839
840
841
842
843
844
845
845
846
847
847
848
848
849
849
850
851
852
853
854
855
855
856
857
857
858
858
859
859
860
861
862
863
864
865
865
866
867
867
868
868
869
869
870
871
872
873
874
875
875
876
877
877
878
878
879
879
880
881
882
883
884
885
885
886
887
887
888
888
889
889
890
891
892
893
894
894
895
896
896
897
897
898
898
899
899
900
901
902
903
903
904
905
905
906
906
907
907
908
908
909
909
910
911
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
921
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
931
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603

```

(continued from previous page)

```

63     double psi = core::calc::structural::evaluate_psi(*chain[res_idx], *chain[res_
64     ↪idx+1]);
65     std::cerr << "Psi angle before change: " << psi * 180.0 / M_PI << " degrees\n";
66     psi = utils::from_string<double>(argv[4]) * M_PI / 180.0 - psi;
67     PdbAtom & C = *chain[res_idx]->find_atom_safe(" C ");
68     PdbAtom & CA = *chain[res_idx]->find_atom_safe(" CA ");
69     core::calc::structural::transformations::Rototranslation::around_axis(CA, C, psi, ↪
70     ↪C, rt);
71     rt.apply(*chain[res_idx]->find_atom_safe(" O "));
72     for (core::index2 ires = res_idx + 1; ires < chain.count_residues(); ++ires) {
73         for (PdbAtom_SP ai : *chain[ires]) rt.apply(*ai);
74     }
75
76 // --- Omega rotation; the new value of the Omega angle (in degrees) is the fifth_
77 // parameter of this program
78 if (argc > 5 && strlen(argv[5]) > 1) {
79     double omega = core::calc::structural::evaluate_omega(*chain[res_idx], *chain[res_
80     ↪idx+1]);
81     std::cerr << "Omega angle before change: " << omega * 180.0 / M_PI << " degrees\n
82     ↪";
83     omega = utils::from_string<double>(argv[5]) * M_PI / 180.0 - omega;
84     PdbAtom & C = *chain[res_idx]->find_atom_safe(" C ");
85     PdbAtom & N = *chain[res_idx+1]->find_atom_safe(" N ");
86     core::calc::structural::transformations::Rototranslation::around_axis(C, N, omega,
87     ↪N, rt);
88     for (core::index2 ires = res_idx + 1; ires < chain.count_residues(); ++ires) {
89         for (PdbAtom_SP ai : *chain[ires]) rt.apply(*ai);
90     }
91
92     std::for_each(chain.first_atom(), chain.last_atom(), [] (PdbAtom_SP ai){std::cout << ↪
93     ↪ai->to_pdb_line() << "\n";});
94 }
```



ex_shared_pointers

A very basic example showing how to use shared pointers (from standard C++ 11 library) when programming in BioShell.

USAGE:

```
./ex_shared_pointers
```

Keywords:

- *STL*

Categories:

- `std::make_shared`

Output files:

- `stdout.out`

Program source:

```

1 #include <memory>
2 #include <iostream>
3
4 #include <core/data/io/Pdb.hh>
5 #include <core/data/structural/Chain.hh>
6 #include <utils/options/OptionParser.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 A very basic example showing how to use shared pointers (from standard C++ 11
12 library) when programming in BioShell.
13
14 USAGE:
15 ./ex_shared_pointers
16 )
17
18 using namespace core::data::structural;
19
20 // --- An example function that takes a reference to an object as an argument
21 void show_chain_code(const Chain & c) { std::cout << c.id() << "\n"; }
22
23 /** @brief A very basic example showing how to use shared pointers (from standard C++ 11
24 library) when programming in BioShell.
25 *
26 * CATEGORIES: std::make_shared
27 * KEYWORDS: STL
28 */
29 int main(const int argc, const char* argv[]) {

```

(continues on next page)

(continued from previous page)

```
29
30     if ((argc > 1) && utils::options::call_for_help(argv[1]))
31         utils::exit_OK_with_message(program_info);
32
33     // --- This is how we create a shared pointer to a Chain object.
34     // --- The object is empty i.e. it doesn't contain any residues
35     Chain_SP chain_sp = std::make_shared<Chain>("A");
36     // --- This is the same as above, but here we create a Chain object
37     Chain chain_object("A");
38
39     // --- This method creates a chain of a given amino acid sequence and returns a
40     // shared pointer to it
41     Chain_SP longer_sp = Chain::create_ca_chain("AGGACL", "A");
42
43     // --- call a method that takes a reference to an object
44     show_chain_code(chain_object);
45     // --- here we create a reference from a shared pointer
46     show_chain_code(*chain_sp);
}
```



ex_simpson_integration

Unit test for Simpson numerical integration routine.

USAGE:

```
ex_simpson_integration
```

REFERENCE: W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1992) Numerical Recipes in C: The Art of Scientific Computing, Cambridge Univ. Press,

Keywords:

- *numerical methods*

Categories:

- core/calc/numeric/simpson_integration

Output files:

- stdout.out

Program source:

```
1 #include <math.h>
2
3 #include <iostream>
4
5 #include <core/calc/numeric/numerical_integration.hh>
6 #include <utils/options/OptionParser.hh>
7 #include <utils/exit.hh>
8
9 std::string program_info = R"(
10
11 Unit test for Simpson numerical integration routine.
12
13 USAGE:
14     ex_simpson_integration
15
16 REFERENCE:
17 W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (1992)
18 Numerical Recipes in C: The Art of Scientific Computing, Cambridge Univ. Press,
19 )";
20
21 /// First of the two functions integrated in this example
22 struct Sin {
23
24     double operator()(double x) { return sin(x); }
25 } sin_func;
26
27 /// Second of the two functions integrated in this example
28 struct X2 {
```

(continues on next page)

(continued from previous page)

```
29
30     double operator()(double x) { return x*x; }
31 } x_square_func;
32
33 /** @brief Example for numerical integration with Simpson method
34 *
35 * CATEGORIES: core/calc/numeric/simpson_integration
36 * KEYWORDS: numerical methods
37 */
38 int main(const int argc, const char *argv[]) {
39
40     if ((argc > 1) && utils::options::call_for_help(argv[1]))
41         utils::exit_OK_with_message(program_info);
42
43     std::cout << core::calc::numeric::simpson_integration(sin_func, 0, M_PI, 1000) << "\n"
44     <>;
45     std::cout << core::calc::numeric::simpson_integration(x_square_func, 0.0, 1.0, 1000)
46     <>< "\n";
47 }
```



ex_split_fasta

ex_split_fasta reads a FASTA file and writes every sequence from it in a separate file

EXAMPLE:

```
./ex_split_fasta 5edw.fasta
```

Keywords:

- *FASTA input*
- *FASTA output*
- *sequence*
- *FASTA*
- *pre-processing*

Categories:

- core/data/io/fasta_io.hh

Input files:

- 5edw.fasta

Output files:

- 5EDW_A.fasta
- 5EDW_P.fasta
- stderr.out
- stdout.out
- 5EDW_T.fasta

Program source:

```

1 #include <iostream>
2
3 #include <core/data/io/fasta_io.hh>
4 #include <utils/string_utils.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 ex_split_fasta reads a FASTA file and writes every sequence from it in a separate file
10 EXAMPLE:
11     ./ex_split_fasta 5edw.fasta
12
13 )";
```

(continues on next page)

(continued from previous page)

```

14
15  /** @brief Reads a file with sequences in FASTA format and writes each sequence to a_
16   *      separate FASTA file.
17   *
18   * CATEGORIES: core/data/io/fasta_io.hh;
19   * KEYWORDS:   FASTA input; FASTA output; sequence; FASTA; pre-processing
20   */
21
22
23  if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
24   missing program parameter
25
26  using core::data::sequence::Sequence_SP; // --- Sequence_SP is just a std::shared_
27   ptr to core::data::sequence::Sequence type
28  using namespace core::data::io;           // --- for FASTA I/O
29
30
31  // --- Create a container where the sequences will be stored
32  std::vector<Sequence_SP> sequences;
33
34  // --- Read a file with FASTA sequences
35  core::data::io::read_fasta_file(argv[1], sequences);
36
37  // --- Write them in separate FASTA files
38  for (const Sequence_SP s : sequences) {
39    std::string header = s->header();
40    std::replace(header.begin(), header.end(), '|', ' '); // --- fix ncbi-style_
41   header in FASTA files
42    auto words = utils::split(header, {' '}); // --- We take the very first word of_
43   the FASTA as a file name; hopefully it is sth meaningful, e.g. a gene name
44    std::ofstream out(words[0] + ".fasta");
45    out << "> " << s->header() << "\n" << s->sequence << "\n";
46    out.close();
47  }
48}

```



ex_structure_iterators

Example that shows how to iterate through structural components

USAGE:

```
ex_structure_iterators 1dt7.pdb
```

where 1dt7.pdb id an input file (PDB format)

Keywords:

- *PDB input*
- *Structure*
- *Chain*
- Residue
- PdbAtom
- *STL*

Categories:

- core/data/structural/Structure

Input files:

- 2gb1.pdb
- 5edw.pdb
- 3dcg.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <iomanip>
3 #include <core/data/io/Pdb.hh>
4 #include <core/chemical/Molecule.hh>
5 #include <core/chemical/molecule_utils.hh>
6 #include <core/data/structural/PdbAtom.hh>
7
8 #include <utils/exit.hh>
9
10 std::string program_info = R"(
11
12 Example that shows how to iterate through structural components
13 )"
```

(continues on next page)

(continued from previous page)

```

14 USAGE:
15   ex_structure_iterators 1dt7.pdb
16
17 where 1dt7.pdb id an input file (PDB format)
18
19 ) ";
20
21 /** @brief Shows how to iterate through structural components (residues, atoms, etc)
22 *
23 * CATEGORIES: core/data/structural/Structure
24 * KEYWORDS: PDB input; Structure; Chain; Residue; PdbAtom; STL
25 */
26 int main(const int argc, const char* argv[]) {
27
28   if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about_
29   ↪missing program parameter
30
31   using namespace core::data::structural;
32   core::data::io::Pdb reader(argv[1],core::data::io::is_not_alternative); // file_
33   ↪name (PDB format, may be gzip-ped)
34   Structure_SP strctr = reader.create_structure(0);
35
36   // ----- Directly iterate over atoms of a structure, jump over chains, residues,_
37   ↪etc.
38   // ----- atom_it is an iterator, which points to a shared pointer to an atom
39   int n_atoms_1 = 0;
40   for (auto atom_it = strctr->first_atom(); atom_it != strctr->last_atom(); ++atom_
41   ↪it) ++n_atoms_1;
42
43   // ----- Iterate over chains, residues, atoms
44   int n_atoms_2 = 0;
45   int n_chains = 0, n_residues = 0;
46   for(auto chain_sp: *strctr) { // --- chain_sp is already a shared pointer to a chain
47     ++n_chains;
48     for(auto residue_sp: *chain_sp) { // --- residue_sp is already a shared pointer to_
49     ↪a residue
50       ++n_residues;
51       for(auto atom_sp: *residue_sp) // --- atom_sp is already a shared pointer to_
52     ↪an atom
53         ++n_atoms_2;
54     }
55   }
56
56   int n_residues_2 = 0;
57   // ----- Iterate over residues of a structure, jump over chains
58   // ----- iter_res_i is an iterator, which points to a shared pointer to a residue
59   for (auto iter_res_i = strctr->first_residue(); iter_res_i != strctr->last_
60   ↪residue(); ++iter_res_i)
61     ++n_residues_2;
62
63   std::cout << "These three atom counts should be equal: " << n_atoms_1 << ", " << n_
64   ↪atoms_2 << " and "
65   << strctr->count_atoms() << "\n";
66   std::cout << "These three residue counts should be equal: " << n_residues << ", " <
67   ↪< n_residues_2 << " and "
68   << strctr->count_residues() << "\n";
69   std::cout << "These two chain counts should be equal: " << n_chains << " and " <<
70   ↪strctr->count_chains() << "\n";

```

(continues on next page)

(continued from previous page)



ex_structure_to_molecule

Unit test which creates a Molecule object from a given PDB file. As a test, the program lists all covalent bonds between a given ligand and the rest of the protein.

USAGE:

```
./ex_structure_to_molecule input.pdb ligand-code
```

EXAMPLE:

```
./ex_structure_to_molecule 4rm4.pdb HEM
```

Keywords:

- *molecule*

Categories:

- core::chemical::structure_to_molecule

Input files:

- 1lw5.pdb
- 4rm4.pdb

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <memory>
3
4 #include <core/algorithms/graph_algorithms.hh>
5 #include <core/chemical/Molecule.hh>
6 #include <core/chemical/molecule_utils.hh>
7 #include <core/calc/structural/angles.hh>
8 #include <core/data/structural/PdbAtom.hh>
9 #include <core/data/structural/selectors/structure_selectors.hh>
10 #include <utils/options/OptionParser.hh>
11 #include <utils/exit.hh>
12
13 std::string program_info = R"(
14
15
16 Unit test which creates a Molecule object from a given PDB file. As a test, the
17 ↵program lists all covalent bonds
18 between a given ligand and the rest of the protein.
```

(continues on next page)

(continued from previous page)

```

19 USAGE:
20   ./ex_structure_to_molecule input.pdb ligand-code
21
22 EXAMPLE:
23   ./ex_structure_to_molecule 4rm4.pdb HEM
24
25 ) ";
26
27 /** @brief Creates a Molecule object from a given PDB file.
28 *
29 * As a test, the program lists all covalent bonds between a given ligand and the_
30 * rest of the protein
31 *
32 * CATEGORIES: core::chemical::structure_to_molecule
33 * KEYWORDS: molecule
34 */
35
36 int main(const int argc, const char *argv[]) {
37
38   if (argc <3) utils::exit_OK_with_message(program_info);
39
40   using namespace core::chemical;
41   using namespace core::data::structural;
42
43   PdbMolecule_SP molecule;
44
45   // --- Read structure that we use to build a molecule
46   core::data::io::Pdb reader(argv[1]); // file name (PDB format, may be gzip-ped)
47   core::data::structural::Structure_SP strctr = reader.create_structure(0);
48   // --- Create molecule object
49   molecule = structure_to_molecule(*strctr);
50
51   // --- Find the ligand object(s) for a given 3-letter code
52   selectors::SelectResidueByName ligand_by_name(argv[2]);
53   std::vector<Residue_SP> ligand;
54   strctr->find_residues(ligand_by_name, ligand);
55
56   for (const auto &l:ligand) {      // --- iterate over ligands found
57     std::cout << "Bonds between " << l->residue_type().code3 << " and the rest of the_"
58     "protein:\n";
59     for (const auto &atom : *l) {    // --- iterate over atoms of a ligand, find all_
60       its bounded partners
61       for (auto it = molecule->cbegin_atom(atom); it != molecule->cend_atom(atom);_
62         ++it) {
63         if ((*it).owner() != l)      // --- if the two atoms belong to different_
64         residues - print the output
65         std::cout << (*atom).atom_name() << " - " << (**it).atom_name() << " " <<_
66         (**it).residue_type().code3
67         << " " << (*it).owner()->residue_id() << " " << (**it).owner()->
68         owner()->id() << "\n";
69       }
70     }
71   }
72 }
```



ex_test_gzip

Unit test which gzips and un-gzips a string data.

USAGE:

```
./ex_test_gzip
```

");

```
// --- The input data to be compressed std::string ala_cif_data = R"(data_ALA # _chem_comp.id ALA
_chem_comp.name ALANINE _chem_comp.type "L-PEPTIDE LINKING" _chem_comp.pdbx_type
ATOMP _chem_comp.formula "C3 H7 N O2" _chem_comp.mon_nstd_parent_comp_id ?
_chem_comp.pdbx_synonyms ? _chem_comp.pdbx_formal_charge 0 _chem_comp.pdbx_initial_date
1999-07-08 _chem_comp.pdbx_modified_date 2011-06-04 _chem_comp.pdbx_ambiguous_flag N
_chem_comp.pdbx_release_status REL _chem_comp.pdbx_replaced_by ? _chem_comp.pdbx_replaces
? _chem_comp.formula_weight 89.093 _chem_comp.one_letter_code A _chem_comp.three_letter_code
ALA _chem_comp.pdbx_model_coordinates_details ? _chem_comp.pdbx_model_coordinates_missing_flag
N _chem_comp.pdbx_ideal_coordinates_details ? _chem_comp.pdbx_ideal_coordinates_missing_flag
N _chem_comp.pdbx_model_coordinates_db_code ? _chem_comp.pdbx_subcomponent_list ?
_chem_comp.pdbx_processing_site RCSB
```

Keywords:

- GZIP

Categories:

- utils/io_utils

Output files:

- stdout.out

Program source:

```
1 #include <sstream>
2
3 #include <utils/io_utils.hh>
4 #include <utils/options/OptionParser.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Unit test which gzips and un-gzips a string data.
10
11 USAGE:
12 ./ex_test_gzip
13
14 )";
15
16 // --- The input data to be compressed
17 std::string ala_cif_data =
```

(continues on next page)

(continued from previous page)

```

18   R" (data_ALA
19
#                                         ALA
20 _chem_comp.id                         ALANINE
21 _chem_comp.name                        "L-PEPTIDE LINKING"
22 _chem_comp.type                         ATOMP
23 _chem_comp.pdbx_type                   "C3 H7 N O2"
24 _chem_comp.formula                     ?
25 _chem_comp.mon_nstd_parent_comp_id    ?
26 _chem_comp.pdbx_synonyms              ?
27 _chem_comp.pdbx_formal_charge         0
28 _chem_comp.pdbx_initial_date          1999-07-08
29 _chem_comp.pdbx_modified_date         2011-06-04
30 _chem_comp.pdbx_ambiguous_flag       N
31 _chem_comp.pdbx_release_status        REL
32 _chem_comp.pdbx_replaced_by          ?
33 _chem_comp.pdbx_replaces             ?
34 _chem_comp.formula_weight            89.093
35 _chem_comp.one_letter_code           A
36 _chem_comp.three_letter_code          ALA
37 _chem_comp.pdbx_model_coordinates_details  ?
38 _chem_comp.pdbx_model_coordinates_missing_flag N
39 _chem_comp.pdbx_ideal_coordinates_details  ?
40 _chem_comp.pdbx_ideal_coordinates_missing_flag N
41 _chem_comp.pdbx_model_coordinates_db_code ??
42 _chem_comp.pdbx_subcomponent_list    ?
43 _chem_comp.pdbx_processing_site      RCSB
44 );
45
46 /** @brief Simple test to gzip and un-gzip a string data
47 *
48 * CATEGORIES: utils/io_utils
49 * KEYWORDS: GZIP
50 */
51 int main(int cnt, char* argv[]) {
52
53     if ((cnt > 1) && utils::options::call_for_help(argv[1]))
54         utils::exit_OK_with_message(program_info);
55
56     std::string zipped,result;
57     // --- here we compress ala_cif_data string with ZIP and store the result in_
58     // another string
59     utils::zip_string(ala_cif_data,zipped);
60     // --- here we un-zip it back and store the ouput in the string "result"
61     utils::unzip_string(zipped,result);
62     if (result == ala_cif_data) {
63         std::cout << "GZIP OK :-)\n";
64         std::cout << "compressed from " << ala_cif_data.size() << " to " << zipped.size()
65         << " bytes\n";
66     } else std::cout << "GZIP ERROR !!!\n";
67
68     // --- Here we un-zip directly to a stream
69     std::stringstream ss;
70     utils::unzip_string(zipped,ss);
71
72     if (ss.str() == ala_cif_data)
73         std::cout << "GZIP OK :-)\n";
74     else std::cout << "GZIP ERROR !!!\n";

```

(continues on next page)

(continued from previous page)

73

}



ex_uniquify

Unit test for uniquify() method which removes redundant objects from a container

USAGE:

```
./ex_uniquify
```

Keywords:

- *data structures*
- *algorithms*

Categories:

- core/algorithms/basic_algorithms.hh

Output files:

- stdout.out

Program source:

```

1 #include <vector>
2
3 #include <core/algorithms/basic_algorithms.hh>
4 #include <utils/options/OptionParser.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Unit test for uniquify() method which removes redundant objects from a container
10
11 USAGE:
12 ./ex_uniquify
13
14 )";
15
16 /** @brief Tests uniquify() method which removes redundant objects from a container.
17 *
18 * CATEGORIES: core/algorithms/basic_algorithms.hh
19 * KEYWORDS: data structures; algorithms
20 */
21 int main(int cnt, char* argv[]) {
22
23     if ((cnt > 1) && utils::options::call_for_help(argv[1]))
24         utils::exit_OK_with_message(program_info);
25
26     std::vector<int> datum = std::vector<int> { 1, 8, 4, 5, 9, 4, 5 };
27     // ----- Below we define >is_equal< operator
28     auto eq = [] (std::vector<int>::iterator a, std::vector<int>::iterator b) -> bool {
29         return *a == *b; };

```

(continues on next page)

(continued from previous page)

```
29 // ----- Below we define >less_then< operator
30 auto lt = [](std::vector<int>::iterator a, std::vector<int>::iterator b) -> bool {_
31     return *a < *b; };
32
33 // ----- Below we apply uniquify() operation on a range of integers
34 datum.erase(core::algorithms::uniquify(datum.begin(), datum.end(), eq, lt), datum.
35             end());
36     for (int c:datum) std::cout << c << " ";
37     std::cout << "\n";
38
39 // ----- Below we apply uniquify() operation on a range of characters
40 std::vector<char> chars = std::vector<char> {'a', 'g', 'd', 'r', 'a', 'd'};
41 chars.erase(core::algorithms::uniquify(chars.begin(), chars.end()), chars.end());
42     for (char c:chars) std::cout << c << " ";
43     std::cout << "\n";
44 }
```



ex_web_client

Simple test for web_client methods downloads 2GB1 protein from rcsb.org website

USAGE:

```
ex_web_client
```

Keywords:

- WWW

Categories:

- ui/www/web_client

Output files:

- stdout.out

Program source:

```
1 #include <iostream>
2 #include <ui/www/web_client.hh>
3 #include <utils/exit.hh>
4 #include <utils/options/OptionParser.hh>
5 #include <utils/exit.hh>
6
7 std::string program_info = R"(
8
9 Simple test for web_client methods    downloads 2GB1 protein from rcsb.org website
10 USAGE:
11     ex_web_client
12
13 )";
14
15 /** @brief Simple test for web_client methods downloads 2GB1 protein from rcsb.org website
16 *
17 * CATEGORIES: ui/www/web_client
18 * KEYWORDS:    WWW
19 */
20 int main(const int argc, const char *argv[]) {
21
22     if ((argc > 1) && utils::options::call_for_help(argv[1]))
23         utils::exit_OK_with_message(program_info);
24
25     using namespace ui::www;
26
27     http_t *request = http_get("http://files.rcsb.org/view/2GB1.pdb", NULL);
28     if (!request) {
29         std::cerr << "Invalid request.\n";
30         return 1;
31 }
```

(continues on next page)

(continued from previous page)

```
31 }
32
33 http_status_t status = HTTP_STATUS_PENDING;
34 int prev_size = -1;
35 while (status == HTTP_STATUS_PENDING) {
36     status = http_process(request);
37     if (prev_size != (int) request->response_size) {
38         std::cout << utils::string_format("%d byte(s) received.\n", (int) request-
39             >response_size);
40         prev_size = (int) request->response_size;
41     }
42 }
43 if (status == HTTP_STATUS_FAILED) {
44     std::cerr << utils::string_format("HTTP request failed (%d): %s.\n",
45         request->status_code, request->reason_phrase);
46     http_release(request);
47     return 1;
48 }
49 std::cout << "\nContent type: " << request->content_type << "\n\n" << (char const_-
50 *) request->response_data << "\n";
51 http_release(request);
52
53 return 0;
}
```



ex_z_matrix_to_cartesian

Test for z_matrix_to_cartesian() function recovers cartesian coordinates of a fluoroethylene from Z-matrix (internal coordinates)

USAGE:

```
./ex_z_matrix_to_cartesian
```

Keywords:

- *internal coordinates*

Categories:

- core::calc::structural::z_matrix_to_cartesian

Output files:

- stdout.out

Program source:

```

1 #include <iostream>
2 #include <core/calc/structural/protein_angles.hh>
3 #include <core/calc/structural/angles.hh>
4 #include <core/data/structural/PdbAtom.hh>
5 #include <utils/options/OptionParser.hh>
6 #include <utils/exit.hh>
7
8 std::string program_info = R"(
9
10 Test for z_matrix_to_cartesian() function recovers cartesian coordinates of a
11 ↪fluoroethylene
12 from Z-matrix (internal coordinates)
13
14 USAGE:
15 ./ex_z_matrix_to_cartesian
16 )";
17
18 /** @brief Test for z_matrix_to_cartesian() function
19 *
20 * This test recovers fluoroethylene from Z-matrix (internal coordinates)
21 * CATEGORIES: core::calc::structural::z_matrix_to_cartesian
22 * KEYWORDS: internal coordinates
23 */
24 int main(const int argc, const char *argv[]) {
25
26     if ((argc > 1) && utils::options::call_for_help(argv[1]))
27         utils::exit_OK_with_message(program_info);
28
29     using core::data::structural::PdbAtom;

```

(continues on next page)

(continued from previous page)

```

30  using namespace core::calc::structural;
31
32  PdbAtom F(1, " F ", -1.0606, 0.1723, 0.0001, ↵
33  ↵core::chemical::AtomicElement::FLUORINE.z);
34  PdbAtom C1(2, " C1 ", 0.1319, -0.4627, -0.0005, ↵
35  ↵core::chemical::AtomicElement::CARBON.z);
36  PdbAtom C2(3, " C2 ", 1.2458, 0.2325, 0.0001, core::chemical::AtomicElement::CARBON. ↵
37  ↵z);
38  PdbAtom H11(2, " H11", 0.1690, -1.5420, 0.0030, ↵
39  ↵core::chemical::AtomicElement::HYDROGEN.z);
40  PdbAtom H21(3, " H21", 2.1991, -0.2751, -0.0004, ↵
41  ↵core::chemical::AtomicElement::HYDROGEN.z);
42  PdbAtom H22(3, " H22", 1.2087, 1.3119, 0.0010, ↵
43  ↵core::chemical::AtomicElement::HYDROGEN.z);
44
45  PdbAtom H11_(2, " H11", 0,0,0, core::chemical::AtomicElement::HYDROGEN.z);
46  PdbAtom H21_(2, " H21", 0,0,0, core::chemical::AtomicElement::HYDROGEN.z);
47  PdbAtom H22_(2, " H22", 0,0,0, core::chemical::AtomicElement::HYDROGEN.z);
48  core::calc::structural::z_matrix_to_cartesian(F, C2, C1, 1.0, to_radians(120), to_ ↵
49  ↵radians(180), H11_);
50  core::calc::structural::z_matrix_to_cartesian(F, C1, C2, 1.0, to_radians(120), to_ ↵
51  ↵radians(180), H21_);
52  core::calc::structural::z_matrix_to_cartesian(H21_, C1, C2, 1.0, to_radians(120), to_ ↵
53  ↵radians(180), H22_);
54  std::cout << C2.to_pdb_line() << "\n";
55  std::cout << C1.to_pdb_line() << "\n";
56  std::cout << F.to_pdb_line() << "\n";
57  std::cout << H11_.to_pdb_line() << "\n";
58  std::cout << H21_.to_pdb_line() << "\n";
59  std::cout << H22_.to_pdb_line() << "\n";
60  double error = H11_.distance_to(H11);
61  error += H21_.distance_to(H21);
62  error += H22_.distance_to(H22);
63  std::cout << "# Average error on the three hydrogen atoms: "<<error/3.0<<"\n";
64 }
```



Alphabethical list of all BioShell examples grouped by *ap_** *ex_** and **.py* category.

7.2 Examples by functionality

7.2.1 File processing

BioShell supports the following file formats, holding bioinformatics data:

- PDB
- FASTA
- CIF
- ALN (ClustalW output with multiple sequence alignment)
- HHpred output¹
- PIR
- XML (most notably these produced by blast+)
- SS2 (PsiPred output that holds secondary structure with predicted probabilities)
- CHK (legacy blast profiles, binary files)
- MAT (PSSM files produced by PsiBlast that contains PSSM)

BioShell offers reading and processing these files, which includes substructure extraction, format conversion and data filtering.

7.2.2 Alignments

Sequence alignment and multiple sequence alignment calculations includes Smith & Waterman² and Needleman & Wunsh³, both available in $O(N^2)$ and $O(N^3)$ implementations. These algotirhms are implemented as C++ templates, which facilitates alignment of virtually any kind of data, assuming that the appropriate scoring method is provided.

7.2.3 Sequence calculations

BioShell can calculate protein pI as well as hydrophobicity according to several scales. Creates, writes and handles sequence profiles. It can also convert an amino acid sequence to one of over 16 reduced alphabets⁴ obtained from teh work by Peterson at al.⁵.

¹ Soding, J and Biegert, A and Lupas, A. N., "The HHpred interactive server for protein homology detection and structure prediction." Nucleic acids research (2005) 33 W244–W248

²

T. F. Smith, and M. S. Waterman, JMB 147.1 (1981): 195-197

³

S. B. Needleman, and C. D. Wunsch, JMB 48.3 (1970)

⁴

L. R. Murphy, A. Wallqvist, R. M. Levy. (2000) "Simplified amino acid alphabets for protein fold recognition and implications for folding". Protein Eng. 13(3):149-152

⁵ Peterson, Kondev, Theriot and Phillips. "Reduced amino acid alphabets exhibit an improved sensitivity and selectivity in fold assignment". Bioinformatics 2009 25:1356-1362

7.2.4 Structure calculations

Since its origing, the main role of BioShell were structure-based calculations. The package can calculate a very broad selection of structural parameters, including:

- distances and distance maps
- contacts and contact maps
- hydrogen bonds
- dihedral angle by name (e.g. Phi or Chi1) or based on arbitrary atoms
- structural superimpositions (Kabsh algorithm) and rmsd value on arbitrary set of atoms
- structure similarity measures such as: GDT, LGS and TM-score

7.2.5 Statistical & numerical analysis

This includes:

- hierarchical agglomerative clustering with arbitrary distance and four merging scenarios: Single Link, Complete Link, Average Link and Ward's method
- spline approximation
- kernel density estimation
- expectation-maximization
- simple non-parametric statistics such as mean, variance, bootstrap estimation, robust estimation

This file has been automatically generated on Jul 19 2023 13:02:49

BioShell *ap_** examples grouped by their functionality.

7.3 Examples by keywords

7.3.1 CIF input

7.3.2 Chain

7.3.3 DSSP

7.3.4 FASTA

7.3.5 FASTA input

7.3.6 FASTA output

7.3.7 Format conversion

7.3.8 Hydrogen bonds

7.3.9 MSA

7.3.10 Monte Carlo

7.3.11 Mover

7.3.12 Needleman-Wunsch

7.3.13 PDB input

7.3.14 PDB line filter

7.3.15 PDB output

7.3.16 PIR

7.3.17 Protein structure features

7.3.18 Rosetta scorefile

7.3.19 STL

7.3.20 Structure

7.3.21 XML

7.3.22 algorithms

7.3.23 clustal input

7.3.24 clustering

7.3.25 contact map

7.3.26 crmsd

7.3. Examples by keywords

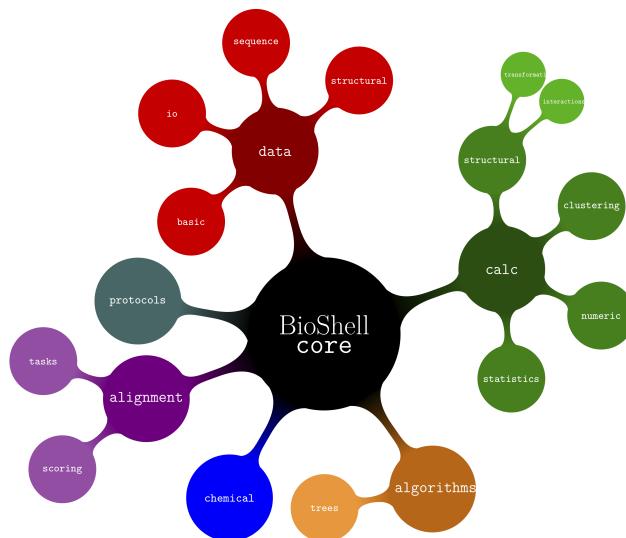
739

7.3.27 data structures

7.3.28 data table

BioShell C++ library

BioShell is a versatile C++11 library for structural bioinformatics. Its struture has been shown in the figure below:



See the [API documentation](#) generated with Doxygen.

8.1 Reading and processing PDB files

Reading PDB files into a BioShell program is divided into two steps:

- loading a text file into memory, and
- parsing its content and creating Structure object(s)

8.1.1 Loading a PDB file

You have to create a reader object to read a PDB file. In the simplest case this looks as below:

```
core::data::io::Pdb reader("infile.pdb");
```

This reader will skip water molecules and hydrogen atoms. You can control which PDB line will be omitted during reading by providing a **PdbLineFilter** instance to the constructor, e.g.

```
core::data::io::Pdb reader("infile.pdb",
    core::data::io::all_true(core::data::io::is_not_water,
    core::data::io::is_not_alternative));
```

PdbLineFilter objects can dramatically limit the number of PDB lines to be parsed and thus shorten the time spent of PDB file loading.

8.1.2 Creating Structure object

Once a file is loaded, you can create a **Structure** object from one of its models:

```
core::data::structural::Structure_SP model = reader.create_structure(0);
```

The very first model is indexed by 0. Every time `create_structure()` method is called, a new **Structure** object is created, which includes necessary memory allocation. Creating new atom objects is in fact the slowest part of this call. Sometimes it is possible to *recycle* old structure filling it with new coordinates rather than just creating a new one from scratch. This can be done as in the `ap_contact_map` program; the relevant fragment is shown below:

```
1     }
2     if (std::strcmp(argv[1], "CB") == 0) {
3         core::data::io::PdbLineFilter filter = core::data::io::is_cb;
4         selector = std::make_shared<core::data::structural::selectors::IsNamedAtom>(" CB
5             ");
6
7         double cutoff = utils::from_string<double>(argv[3]); // The third parameter is the_
8             // contact distance (in Angstroms)
9         core::data::io::Pdb reader(argv[2],filter); // --- file name (PDB format, may be_
10            // gzip-ped)
```

Coordinates of a new structure must fit into the existing stucture i.e. the new structure must be composed of the same number of chains, residues and atom as the old one. In practice this is most useful when a multi-model PDB file must be loaded, as in this example:

- in the **line 1** a PDB file is loaded with a filter instance defined someehere before
- in the **line 3** a **Structure** object is created based on the first model defined in the file
- in the **line 4** a **ContactMap** object is created and the first structure is loaded id
- finally, in **lines 5-8** a loop iterates over all the remaining models; in **line 6** coordinates of each model are loaded into the existing structure (the one created in **line 3**)

Residue, **PdbAtom** and **Chain** objects are created only once, when the structure at index 0 is loaded. After that the loop only substitutes. coordinates of this structure

CHAPTER 9

BioShell Python library

BioShell 3.0 comes also with Python bindings i.e. BioShell classes can be also used as Python modules. Let's consider the following C++ program that reads a PDB file and creates a `Structure` object that represents a biomacromolecular complex. Then it writes a FASTA sequence for every chain in the structure:

```
1 #include <iostream>
2
3 #include <core/data/io/Pdb.hh>
4 #include <core/data/structural/Structure.hh>
5 #include <utils/options/OptionParser.hh>
6 #include <utils/exit.hh>
7
8 */
9 int main(const int argc, const char *argv[]) {
10
11     using namespace core::data::io; // Pdb and create_fasta_string lives there
12
13     Pdb reader(argv[1], is_not_alternative, only_ss_from_header, true);
14     core::data::structural::Structure_SP strctr = (reader.create_structure(0));
15
16     // Iterate over all chains
17     for (int ic = 0; ic < strctr->count_chains(); ++ic)
18         std::cout << "> " << strctr->code() << (*strctr)[ic]->id() << "\n" // --- e.g. ↴
19         << (*strctr)[ic]->create_sequence()->sequence << "\n"; // --- prints ↴
20         ↴the sequence itself
21 }
```

The same program written in Python looks much simpler. It calls nearly the same BioShell C++ objects as the one above, but due to simplicity of Python, the script is a bit shorter:

```
1 import sys
2
3 from pybioshell.core.data.io import find_pdb
```

(continues on next page)

(continued from previous page)

```

5 GROUP:      File processing; Format conversion
6
7 """
8     sys.exit()
9
10 for pdb_fname in sys.argv[1:] :
11     structure = find_pdb(pdb_fname, "./").create_structure(0)
12     for ic in range(structure.count_chains()) :
13         chain = structure[ic]
14         print(">", structure.code(), chain.id())
15         print(chain.create_sequence(IF_EXCLUDE_LIGANDS).sequence)

```

9.1 Reading and writing PDB files

9.1.1 Reading PDB files

Reading PDB data is a two-stage process. First you create a reader that loads PDB content into memory:

```
pdb = Pdb(pdb_code, "", False)
```

First argument is a string which is a path to a PDB file. Second is a string which represents a `PdbLineFilter` object eg. "is_ca" - reader will read only CA atoms or "is_not_water" - will read everything what is not water. In general, filtering PDB lines may considerably speed up loading time. Third argument is a flag whether reader should read header of a PDB file. Header is neccesary to read some additional information eg. about secondary structure, connectivity (CONNECT fields), etc.

Then the content is parsed according to user's requests. In the example below the FASTA string is printed out.

```

1 structure = pdb.create_structure(0)
2 for ic in range(structure.count_chains()) :
3     chain = structure[ic]
4     print(">", structure.code(), chain.id())
5     print(chain.create_sequence().sequence)

```

9.1.2 Writing PDB files

`PdbAtom` class provides `create_pdb_line()` method. In the following example four nested loop iterate over models, chains, residues and (finally) atoms;

```

1 for i_model in range(pdb.count_models()) :
2     structure = pdb.create_structure(i_model)
3     for ic in range(structure.count_chains()):
4         chain = structure[ic]
5         for ir in range(chain.count_residues()):
6             resid = chain[ir]
7             for ia in range(resid.count_atoms()):
8                 if resid[ia].atom_name() == "CA" or resid[ia].atom_name() == "CB":
9                     print(resid[ia].to_pdb_line())

```

Also, `pybioshell.core.data.io` module contains `write_pdb()` method which writes in example below model no. 5 of a `structure` object (**note**: models count from 0) to a file with a given name. Existing files will be overwritten.

```

1 reader = Pdb(pdb_fname, "is_ca", False)
2     structure = reader.create_structure(0)
3     write_pdb(structure, out_fname, 5)

```

9.2 Help! My script crashes!

9.2.1 Getting more logs from BioShell library

There are nine levels of importance for log messages reported by BioShell methods. Seven of them are used for *general* reporting, in the order of decreasing importance: *CRITICAL*, *SEVERE*, *WARNING*, *INFO*, *FINE*, *FINER* and *FINEST*. The two additional levels: *HTTP* and *FILE* are used to report *HTTP* messages and information about disk I/O, respectively. By default, logging level is set to *INFO* which means that only *INFO* and more important messages show up. To see more logs, increase the verbosity as follows:

```

from pybioshell.utils import LogManager
LogManager.FINEST()

```

9.2.2 Checking C++ exception from Python

Occasionally PyBioShell library throws an exception, which stops a Python script. To find out the reason, wrap a bioshell call into a **try / except** block and print out execution information as below:

```

try:
    pdb = find_pdb(pdb_fname, path, True, False)
except:
    sys.stderr.write(str(sys.exc_info()[0])+" "+str(sys.exc_info()[1]))

```

9.3 Using PyBioShell in PyMOL

PyBioShell can be loaded by a Python interpreter as any other library. This also applies to the interpreter that is build in PyMOL - a molecular visualization system¹.

9.3.1 Loading PyBioShell

Load a PyBioShell module by typing a respective command in a PyMOL command input area, the same as you would use in a Python script. E.g. you can try the following:

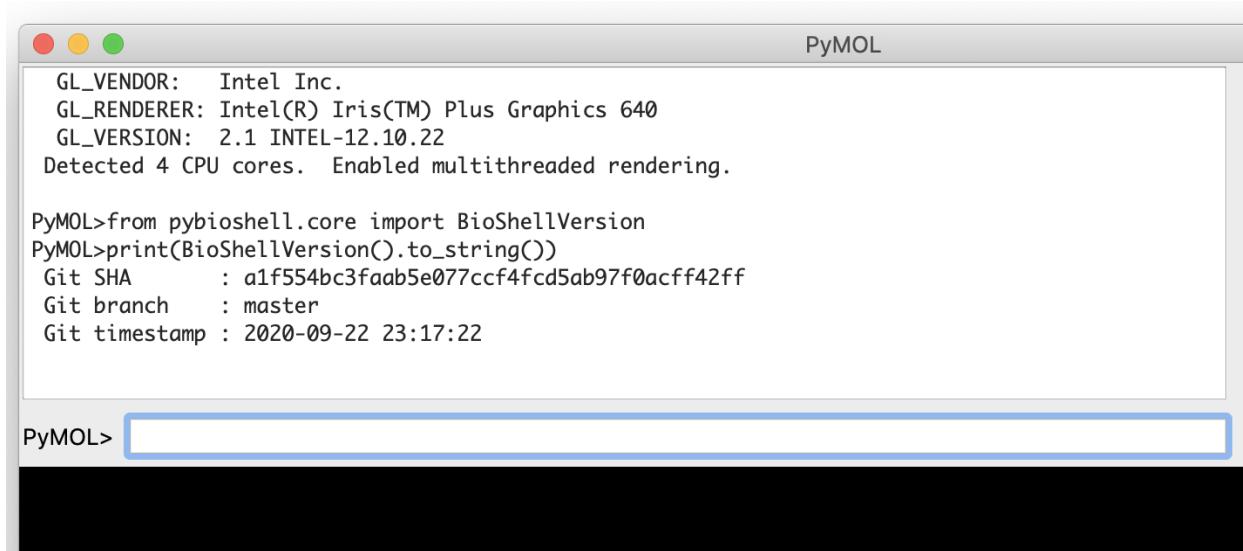
```

from pybioshell.core import BioShellVersion
print(BioShellVersion().to_string())

```

which should print information about your BioShell version, as you can see below:

¹ <https://pymol.org/>



After a successful import, you can use any PyBioShell module inside PyMOL. But how to transfer data that is visible in PyMOL 3D window to BioShell?

9.3.2 Loading PDB data from PyMOL

Fortunately PyMOL can export a desired part of a 3D view in a PDB format, which can be directly parsed by `core.data.io.Pdb` reader, e.g:

```
from pybioshell.core.data.io import Pdb
cmd.fetch("2gb1")
pdb_txt = cmd.get_pdbstr('all')
pdb = Pdb(pdb_txt, "")
strstr = pdb.create_structure(0)
```

The detailed description how to read in and process PDB data is [given here](#)

References

These pages provide documentation for BioShell package. Api documentation is given [here](#) To answer most common questions, we have a list of shortcuts below:

- **Description of BioShell package components:** [What is BioShell](#)
- **Overview of BioShell functionality:** [Examples by functionality](#)
- **How to install BioShell package:** [Installation](#)
- **How to install PyBioShell package (Python bindings to BioShell):** [PyBioShell Installation](#)

Our laboratory protocols, both related to BioShell and Rosetta, are provided on our [labnotes](#) website.

CHAPTER 10

SURPASS model

SURPASS model Single United Residue per Pre-Averaged Secondary Structure fragment is a coarse-grained low resolution model for protein simulations.

- See *SURPASS representation*
- Read about: *SURPASS force field*
- Necessary and optional *Input files*
- Resulting *Output files*
- `surpass_annealing` command line *program*

10.1 SURPASS force field

The generic force field for SURPASS model describes the most fundamental properties of globular proteins. The only sequence-dependent parameters comes from secondary structure. The background for force-field derivation define regularities observed in real protein structures. The statistics is based on a redundant set of 4600 protein chains, representing all known protein families, with resolution not lower than 1.6Å and a sequence identity not greater than 60%. Described below analysis of these statistical data defines the SURPASS force field consisting of knowledge-based statistical potentials. > [Figure 1. Schematic illustration of the terms included in the SURPASS force field.]

10.1.1 Terms to create regular secondary structure (close in sequence)

1. Short range interactions

The deficiencies of atomic details in strongly simplified and pre-averaged SURPASS chain may cause an incorrect local geometry of the structure. To avoid this, it is necessary to transfer the structural regularities of the atomistic models onto the corresponding sets of united atoms. All generic terms: R12, R13, R14 and R15 are prepared in six variants (HH, EE, CC, HE, HC, EC) depending on the secondary structure assignments for pairs of residues located at key positions. All short-range interactions have been implemented in the force field as potential of mean field

(PMF), using a one-dimensional kernel density estimator (KDE) as a method of estimating the density of the empirical distribution.

> [Table 1. Secondary structure dependent short range interactions.

term | statistic plots (6 variants) | energy plot (all-in-one) - table 4 rows x 8 columns]

> [equation and description]

2. Model of hydrogen bonding

In the SURPASS model only the hydrogen bonds between residues that are distant in the sequence, especially in extended structure fragments, are modeled more directly. Therefore, the formation of model hydrogen bonds depends on the fulfillment of a few simple geometrical conditions:

- the length of the model hydrogen bond is in a range of 3.8Å to 6.0Å, and the most probable length is 4.65Å;
- the maximum number of connections for each pseudo residue in the β -strand is 2; if there are more potential candidates for hydrogen bond formation, the best two are chosen according to the following angular criteria:
 - a hydrogen bond should be perpendicular to the main chain of both interacting β -strands and the permitted angle range is from 70 to 115;
 - the maximum allowable twist of the beta sheet, measured as the planar angle between the main chains of two adjacent β -strands, is not greater than 55;
 - for a pseudo residue that forms two hydrogen bonds (with two different β -strands), the planar angle between these bonds must be greater than 125, and 180 is the best orientation.

> [Figure 2. Statistical analysis of the geometry of the model hydrogen bond: A – length of hydrogen pseudobonds extracted from the RDF of distance between i-th and j-th pseudoresidues in two beta strands. B – angle between two β -strands connected by a hydrogen bond. C – twist of the β -sheet measured as a planar angle between the main chains of two adjacent β -strands; D – angle between two hydrogen bonds of three connecting β -strands.]

3. Helix stiffness

10.1.2 Terms to control local packing (close in space)

1. Local repulsive interactions

2. Local attractive interactions: Excluded Volume & Contacts

- pseudo atom H (helix-like) for helical (HHHH) or almost helical (HHHC, CHHH) fragments
- pseudo atom S (like β -strand) representing centers of mass of EEEE, EEEC or CEEE, fragments
- pseudo atom C (coil-like) for all remaining secondary structure combinations (H, E and C)

10.2 Input files

Secondary structure profile file (*.ss2 file format) is the only mandatory input to the program. Example input file for 2GB1 protein can be found [here](#). Optionally, a starting conformation (in the PDB format) may be provided with `-model::pdb` flag.

Please note that these input files must be in all-residues representation, even though SURPASS models are shorter by 3 residues

An input SS2 file may be conveniently generated from a PDB file as long as it contains secondary structure information in its header. The following command uses seqc program of BioShell package:

```
seqc -in:::pdb=2gb1.pdb -select:chains=A -in:pdb:header -out:ss2
```

10.3 Output files

After every *outer cycle* (see options below), **surpass_annealing** makes an observation of the current state of the simulated system. Typically this means observing energy of the system, various evaluators, topology of a protein, and the coordinates in .pdb file format.

energy.dat The file provides energy components for every observed frame

movers.dat The file provides movers acceptance ratio and range

observers.dat provides various measurements for every observed frame, such as elapsed time, temperature, radius of gyration, crsmd, etc.

topology.dat The file topology footprint

trajectory.pdb File contains coordinates of the system recorded at every observation event (file name may be changed with -out :pdb option)

10.4 SURPASS representation

SURPASS is a new coarse-grained model of protein structure. Deep reduction of the number of atoms in the representation results in a powerful computational speed-up and in this context ranks the model as a low resolution.

The number of pseudoresidues present in the modeled system corresponds to polipeptide chain size and is equal to N-3, where N is the number of amino acids in the sequence. The positions of pseudo residues are defined by averaging the coordinates of short secondary structure fragments. These fragments are replaced by a single center of interactions. The choice of four residue averaging is crucial for the local geometry of the model because leads to an almost linear shape of the SURPASS fragments representing helices or beta strands.

The SURPASS representation assumes three types of pseudo atoms depending on secondary structure assignment of the averaged fragments of protein structure:

- pseudo atom H (helix-like) for helical (HHHH) or almost helical (HHHC, CHHH) fragments
- pseudo atom S (like β -strand) representing centers of mass of EEEE, EEEC or CEEE, fragments
- pseudo atom C (coil-like) for all remaining secondary structure combinations (H, E and C)

10.5 *surpass_annealing* program

You need just a .ss2 file for your target protein to run the program. Provide it using -in:ss2 command line option. Other options are used to:

specify starting conformation

-model:random

starts the simulation from a random chain (extended conformation) while

-model:pdb

provides an input starting conformation

specify temperature set for simulated annealing run

`-sample:t_start`, `-sample:t_end` and `-sample:t_steps` define a set of N+1 temperatures distributed uniformly between the starting and the ending temperature

specify the length of the simulation (Monte Carlo steps) `-sample:mc_inner_cycles` defines the amount of sampling *between* frames that are recorded `sample:mc_cycle_factor` makes every inner MC cycle longer (multiplying them by a given factor) `-sample:mc_outer_cycles` defines the number of frames recorded for every temperature value

The total number of MC steps is then $N_{temperatures} \times N_{inner} \times N_{factor} \times N_{outer}$

Example parameters for *ab-initio* simulation of a protein:

```
./surpass_annealing -model:random \
-in:ss2=test_inputs/2gb1A.ss2 \
-sample:t_start=2.2 \
-sample:t_end=0.9 \
-sample:t_steps=15 \
-sample:mc_outer_cycles=100 \
-sample:mc_inner_cycles=10 \
-sample:mc_cycle_factor=10 \
-sample:perturb:range=0.7
```

Example parameters to relax an input structure:

```
./surpass_annealing -model:pdb=2gb1A.pdb \
-in:ss2=test_inputs/2gb1A.ss2 \
-sample:t_start=2.2 \
-sample:t_end=0.9 \
-sample:t_steps=15 \
-sample:mc_outer_cycles=100 \
-sample:mc_inner_cycles=10 \
-sample:mc_cycle_factor=10 \
-sample:perturb:range=0.7
```

CHAPTER 11

Notes for BioShell developers

Do you want to participate in the project? Have brilliant idea what would be a cool extension? Or maybe you need a specific feature?

This page will help you with rolling your own copy of BioShell!

1. Don't create branch, fork instead Make your own fork of BioShell repository

2. Work as usually

3. Merge with the upstream repository often

Remember to sync your fork of the BioShell source tree to keep it up-to-date with the upstream repository. Use the command below:

```
git pull git@bitbucket.org:dgront/bioshell.git
```

This will only update your local copy of the repository. Use `git push` to update your fork on BitBucket.

4. Create a pull request

When your work is done, you may contribute your changes to the main BioShell repository. Simply push your development branch to the forked remote repository and create the pull request.

CHAPTER 12

Indices and tables

- genindex
- search

Symbols

-model:pdb
 command line option, 749
-model:random
 command line option, 749

B

bioshell, 6
bioshell-apps, 6

C

command line option
 -model:pdb, 749
 -model:random, 749

E

examples, 6