BioShell 3.0

Release 3.0

Jul 19, 2023

Overview

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 convertion. 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 distribitions 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:

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

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 that the default one (e.g. gcc version 4.9), say:

```
cmake -DCMAKE_CXX_COMPILER=g++-4.9  -DCMAKE_C_COMPILER=gcc-4.9 -DCMAKE_
↪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 precombiled 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. Prequisities

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 utulity 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 accordint 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 actuall 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 - 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 imput 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 ouput 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 inthis case) and the number of these atoms (here 96), respectively. Finaly the last two columns provude 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

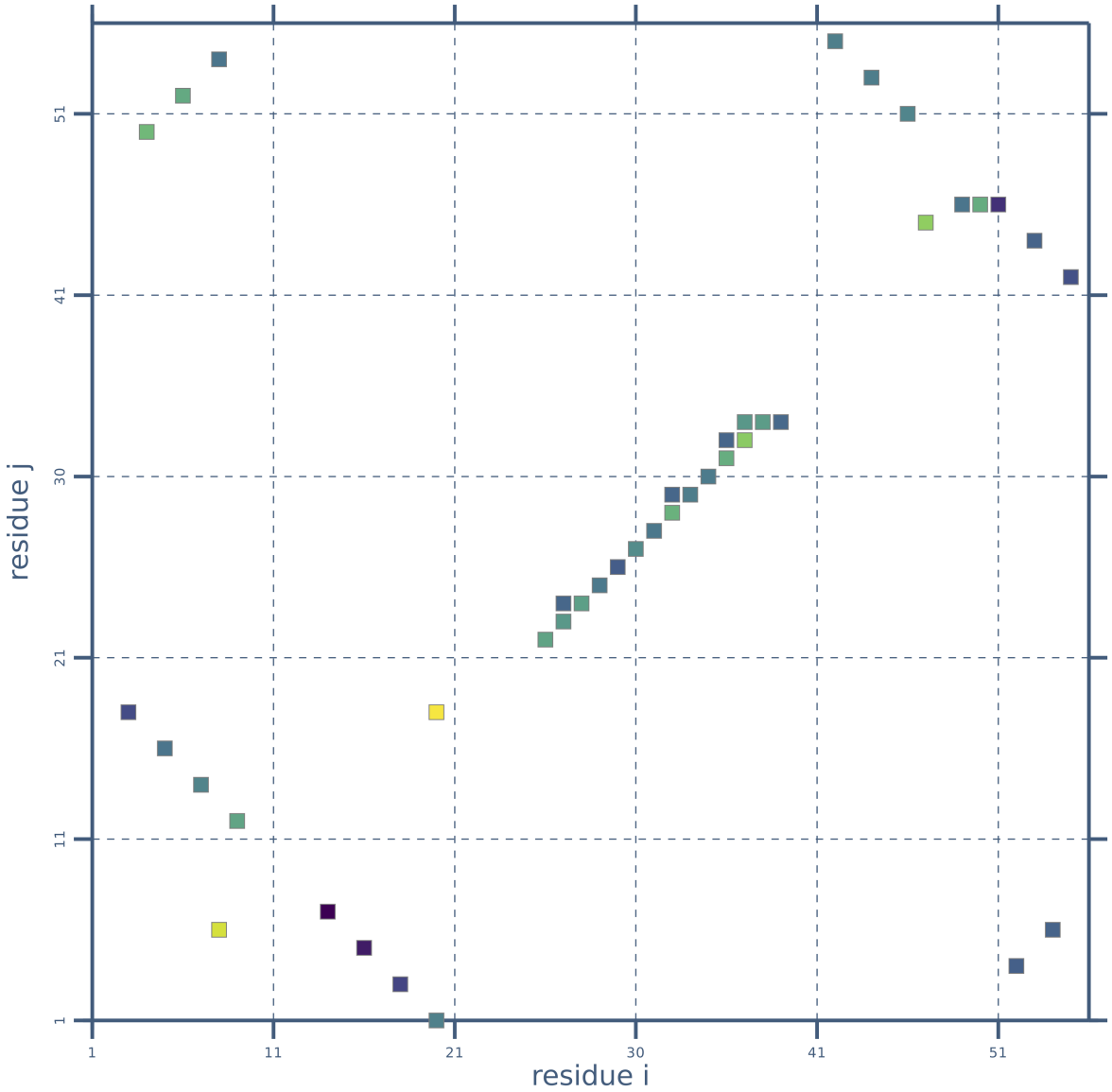
Program source:

```
#include <iostream>
#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/protocols/selection_protocols.hh>
#include <core/calc/structural/interactions/BackboneHBondMap.hh>
#include <utils/exit.hh>

using namespace core::data::structural;
using namespace core::data::io;
using namespace core::data::basic;

std::string program_info = R"(

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

)";

/** @brief Calculates a map of backbone hydrogen bonds.
 *
 * BackboneHBondMap is derived from PairwiseResidueMap class
 *
 * CATEGORIES: core/calc/structural/BackboneHBondMap;
 * KEYWORDS:   PDB input; Hydrogen bonds; data_structures; Protein structure features
 * GROUP: Structure calculations;
 * IMG: ap_BackboneHBondMap-2gb1.png
 * IMG_ALT: Map of backbone hydrogen bonds for 2GB1 protein
 */
int main(const int argc, const char* argv[]) {

  if(argc < 2) utils::exit_OK_with_message(program_info);      // --- 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); // --- Read in a PDB file
53   core::data::structural::Structure_SP strctr = reader.create_structure(0);      // --- create a Structure object from the first model
54
55   // ---------- Remove everything but amino acids from an input structure; BackboneHBondMap currently can process only AA
56   core::protocols::keep_selected_atoms(core::data::structural::selectors::IsAA{}, *strctr);
57
58   // --- The line below creates a map of backbone hydrogen bonds; -0.2 is the energy cutoff (in kcal/mol) to recognize the bond
59   core::calc::structural::interactions::BackboneHBondMap hb_map(*strctr, -0.2);
60   std::cout << "# " << hb_map.count_bonds() << " hydrogen bonds found in backbone:\n";
61   for (auto h_it = hb_map.cbegin(); h_it != hb_map.cend(); ++h_it) // --- iterate over the bonds and print each of them
62     std::cout << *((*h_it).second->donor_residue()) << " -> " << *((*h_it).second->acceptor_residue()) << " : "<< *(*h_it).second << "\n";
63
64   // --- Here we test some other BackboneHBondMap methods
65   const auto hbond = (*hb_map.cbegin()).second; // --- here we make a local copy of the first h-bond to be used in tests
66   std::cerr << "# Is residue 0 h-bonded to residue 3? " << ((hb_map.are_H_bonded(0, 3)) ? "yes\n" : "no\n");
67   std::cerr << "# Is residue 0 h-bonded to residue 5? " << ((hb_map.at(0, 5) != nullptr) ? "yes\n" : "no\n");
68   std::cerr << "# Are " << *hbond->donor_residue() << " and " << *hbond->acceptor_residue() << " really bonded? " <<
69     ((hb_map.at(*hbond->donor_residue(), *hbond->acceptor_residue()) != nullptr) ? "yes\n" : "no\n");
70 }
```



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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/calc/structural/transformations/Crmsd.hh>

#include <utils/exit.hh>

std::string program_info = R"(

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
```

(continues on next page)

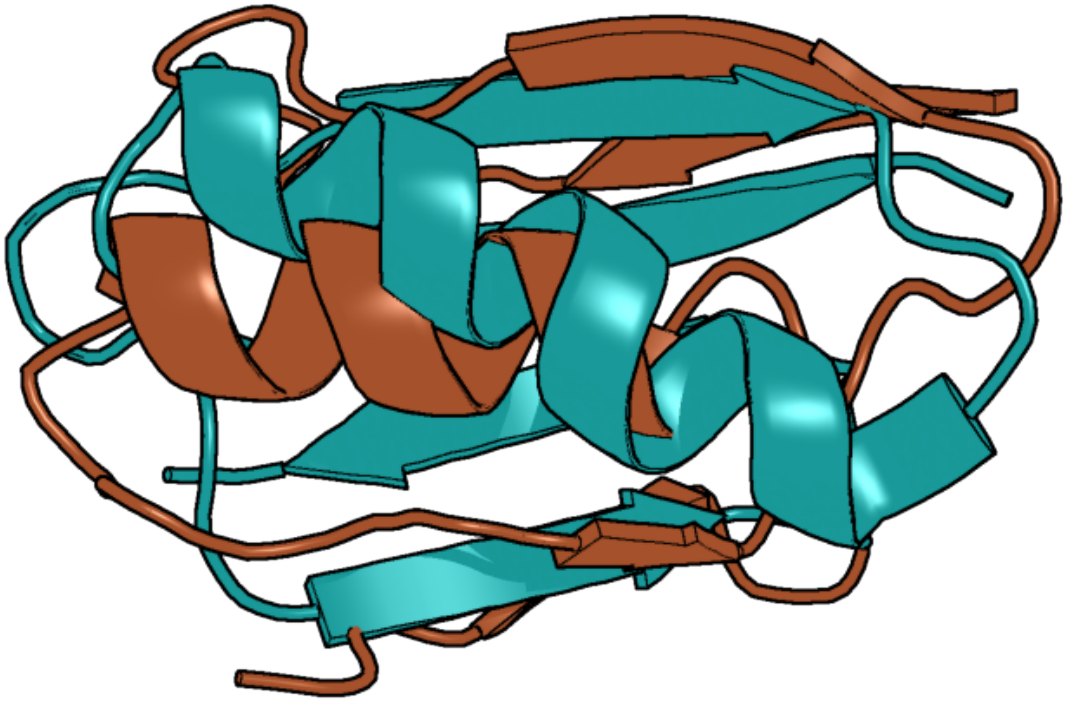
(continued from previous page)

```
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

)";

/** @brief Calculates crmsd value on C-alpha coordinates. The program prints just the crmsd value.
 *
 * CATEGORIES: core/calc/structural/transformations/Crmsd
 * KEYWORDS:   PDB input; crmsd
 * GROUP: Structure calculations;
 * IMG: ap_Crmsd_deepteal_brown_1.png
 * IMG_ALT: 2GB1 model structure superimposed on the native, crmsd = 4.93952
 */
int main(const int argc, const char *argv[]) {

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

  using core::data::basic::Vec3;
  using namespace core::calc::structural::transformations;

  Crmsd<std::vector<Vec3>,std::vector<Vec3>> rms;

  if(argc==2) { // --- The case of each-vs-each calculations between models of a single PDB file
    core::data::io::Pdb q_reader(argv[1],core::data::io::is_ca, core::data::io::keep_all, false);
    core::data::structural::Structure_SP q_strctr = q_reader.create_structure(0); // --- create a structure object

    std::vector<std::vector<core::data::basic::Vec3>> models(q_reader.count_models());
    for(int i=0;i<q_reader.count_models();++i) {
      models[i].resize(q_strctr->count_atoms());
      q_reader.fill_structure(i,models[i]);
      for (int j = 0; j < i; ++j)
        std::cout << i<<" "<<j << " "<<rms.crmsd(models[i], models[j],models[j].size()) << "\n";
    }
  } else { // --- The case when two PDB files are given
    core::data::io::Pdb q_reader(argv[1], core::data::io::is_ca, core::data::io::keep_all, false);
    core::data::structural::Structure_SP q_strctr = q_reader.create_structure(0); // --- create a structure object
```

(continues on next page)

(continued from previous page)

```

    core::data::io::Pdb t_reader(argv[2], core::data::io::is_ca, core::data::io::keep_all, false);
    core::data::structural::Structure_SP t_strctr = t_reader.create_structure(0); // -- create a structure object

    if (q_strctr->count_atoms() != t_strctr->count_atoms())
      utils::exit_OK_with_message("The two structures have different number of CA atoms!\n");

    std::vector<Vec3> q, t;
    for (auto atom_it = q_strctr->first_atom(); atom_it != q_strctr->last_atom(); ++atom_it) q.push_back(**atom_it);
    for (auto atom_it = t_strctr->first_atom(); atom_it != t_strctr->last_atom(); ++atom_it) t.push_back(**atom_it);
    std::cout << "crmsd: " << rms.crmsd(q, t, q_strctr->count_atoms()) << "\n";
  }
}
```



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

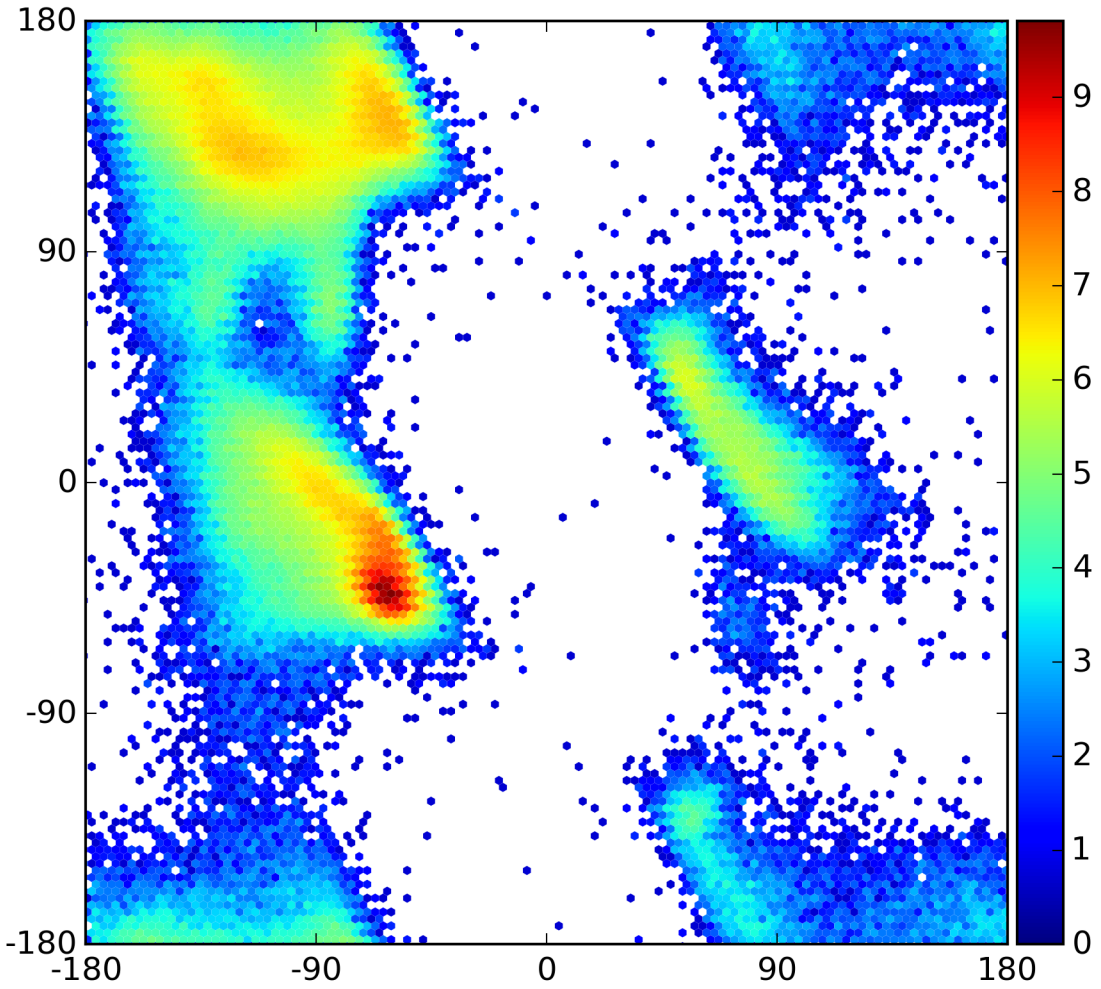
Program source:

```
#include <iostream>
#include <random>
#include <vector>

#include <core/index.hh>

#include <core/calc/statistics/Hexbins.hh>
#include <core/calc/statistics/Random.hh>
#include <core/data/io/DataTable.hh>

std::string program_info = R"(

Reads a file with 2D observations (two columns with real values) and makes hexbin histogram.
USAGE:
    ap_Hexbins input.dat [bin_side_width]

)";

/** @brief Reads a file with 2D observations (two columns) and makes hexbin histogram.
 *
 * CATEGORIES: core::calc::statistics::Hexbins
 * KEYWORDS:   histogram; statistics
 * GROUP: Statistics;
 * IMG: ramachandran_map_all.png
 * IMG_ALT: hexabin representation of Ramachandran map (histogram made from non-redundant subset of PDB)
 */
int main(const int argc, const char *argv[]) {

  using namespace core::calc::statistics;

  Hexbins<double, core::index4> hist(0.05);
  if (argc > 1) { // --- If an input file was given, make histogram using this data
    if (argc > 2) hist.bin_side(atof(argv[2]));
    float x,y;
    std::ifstream in(argv[1]);
```

(continues on next page)

(continued from previous page)

```
    std::string line;
    // --- here we read the input file using pure C API since it's faster than C++ fancy streams
    while (std::getline(in, line)) {
      sscanf(line.c_str(),"%f %f",&x,&y);
      hist.insert(x,y);
    }
  } else { // --- otherwise generate some random data
    std::cerr << program_info <<"\n";
    Random r = Random::get();
    r.seed(9876543);
    NormalRandomDistribution<double> dist_x(1.0, 0.25);
    NormalRandomDistribution<double> dist_y(3.0, 0.5);
    for (size_t i = 0; i < 100000; ++i) {
      hist.insert(dist_x(r), dist_y(r));
    }
  }
  std::cout << "# Created histogram of " << hist.count_entries() << " observations, " << hist.count_outside()
            << " were outside\n";

  std::vector<std::pair<double,double>> coordinates; // --- a vector used to retrieve coordinates of each hexagon

  for (auto it = hist.cbegin(); it != hist.cend(); ++it) {
    coordinates.clear();
    auto bin = (*it).first;
    hist.bin_vertices(bin,coordinates);
    std::cout << utils::string_format("%4d %4d %4d ",bin.first, bin.second,(*it).second);
// --- uncomment the lines below to print coordinates of hexbin vertexes in every line
// --- Note: this is a lot of (redundant) output; make_plots.py script may generate these coordinates for you based on bin indexes
//    std::cout <<" : ";
//    for(const auto & xy : coordinates) std::cout << utils::string_format("%8.3f %8.3f ",xy.first,xy.second);
    std::cout <<"\n";
  }
}
```



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

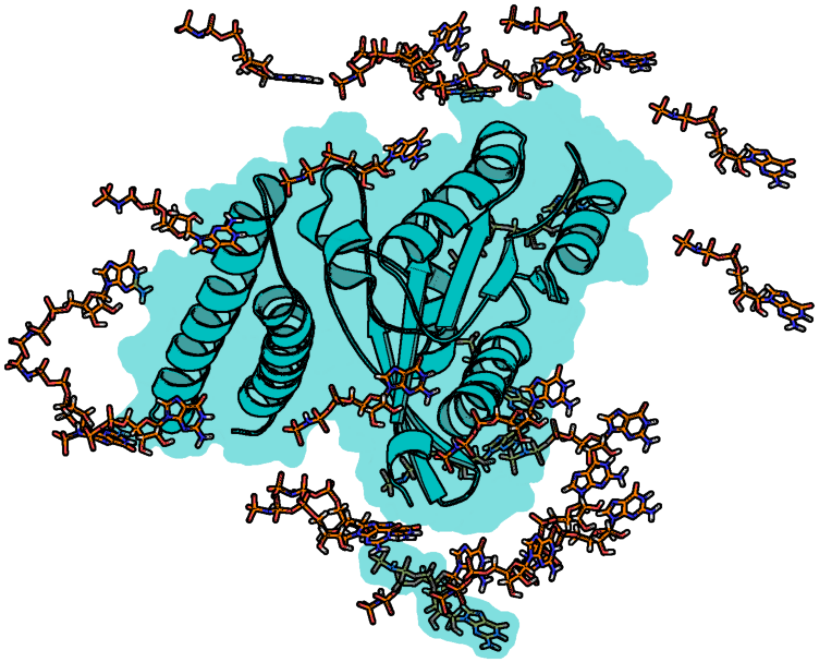
Program source:

```
#include <core/data/basic/Vec3.hh>
#include <core/data/io/Pdb.hh>
#include <simulations/movers/LigandTossingMover.hh>
#include <simulations/forcefields/ConstEnergy.hh>
#include <simulations/systems/PdbAtomTyping.hh>
#include <simulations/observers/cartesian/PdbObserver_OBSOLETE.hh>
#include <simulations/observers/cartesian/ExplicitPdbFormatter_OBSOLETE.hh>
#include <simulations/sampling/AlwaysAccept.hh>

using namespace core::data::basic;
using namespace simulations;

#include <utils/exit.hh>

std::string program_info = R"(

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)

)";

/** @brief To test LigandTossingMover mover, tosses a ligand on a proteins surface
 *
 * The program creates a multimodel PDB with random orientations of a ligand in␣
↪respect to the protein
 *
 * CATEGORIES: simulations::movers::LigandTossingMover
 * KEYWORDS:   docking; Mover
 * IMG: ap_LigandTossingMover.png
 * IMG_ALT: 25 conformations of the same ligand randomly placed on the surface of a␣
↪protein
 */
int main(int argc, char *argv[]) {

  using namespace simulations::systems; // for AtomTypingInterface and ResidueChain
  using namespace simulations::observers::cartesian;
  using namespace core::data::io; // for Pdb reader
```

(continues on next page)

(continued from previous page)

```
  using core::data::basic::Vec3;

  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  // --- Read the input PDB and create a structure object
//  Pdb reader(argv[1], all_true(is_not_hydrogen, is_not_water, is_not_alternative), true);
  Pdb reader(argv[1]);
  core::data::structural::Structure_SP strctr = reader.create_structure(0);

  // --- Prepare a modeled system from a given PDB file
  ResidueChain_OBSOLETE<Vec3> system(*strctr);

  std::string ligand_code(argv[2]);
  int from = -1, to = -1, i = 0;
  for (auto atom_it = strctr->first_atom(); atom_it != strctr->last_atom(); ++atom_it) {
    if (ligand_code.size()==3) { // truly it's a ligand code
      if (((*atom_it)->owner()->residue_type().code3 == ligand_code) && (from == -1)) from = i;
      if (((*atom_it)->owner()->residue_type().code3 != ligand_code) && (to == -1) && (from != -1)) to = i - 1;
    } else { // it should be a chain code then
        if (((*atom_it)->owner()->owner()->id() == ligand_code) && (from == -1)) from = i;
        if (((*atom_it)->owner()->owner()->id() != ligand_code) && (to == -1) && (from != -1)) to = i - 1;
    }
    ++i;
  }
  if (to == -1) to = i - 1; // --- assign the last atom if nothing has been assigned yet
  AtomRange moving(from,to);
  std::cout << "Moving atoms: " << moving << "\n";

  core::index4 n_moves = (argc==4) ? atoi(argv[3]) : 10;
  simulations::movers::LigandTossingMover<Vec3> mover(system, moving, 5.0);

  simulations::sampling::AlwaysAccept alwaysMove;
  std::shared_ptr<AbstractPdbFormatter_OBSOLETE<Vec3>> fmt = std::make_shared<ExplicitPdbFormatter_OBSOLETE<Vec3>>(*strctr);

  simulations::observers::cartesian::PdbObserver_OBSOLETE<Vec3> trajectory(system, fmt, "out.pdb");
  for (size_t i = 0; i < n_moves; i++) {
    mover.move(alwaysMove);
    trajectory.observe();
  }
}
```



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 1uox_1uox_1.pir 1uox.pdb -
ap_aligned_pdb 1uox_1uox_1.pir - 1uox.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:

```
#include <iostream>

#include <core/data/io/pir_io.hh>
#include <core/data/io/Pdb.hh>
#include <core/data/io/alignment_io.hh>
#include <core/data/io/fasta_io.hh>

#include <utils/exit.hh>
#include <core/alignment/PairwiseSequenceAlignment.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/calc/structural/transformations/Crmsd.hh>
#include <utils/Logger.hh>

std::string program_info = R"(

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
```

(continues on next page)

(continued from previous page)

```
30
31  EXAMPLE:
32      ap_aligned_pdb 1uox_1uox_1.pir 1uox.pdb -
33      ap_aligned_pdb 1uox_1uox_1.pir - 1uox.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> &
    ↪residues,
41                                 const selectors::ResidueSelector & which_part) {
42
43    selectors::IsAA aa_only;
44    // --- Read the first structure and repack its amino acid residues (the other␣
    ↪cannot be aligned)
45    core::data::io::Pdb reader(pdb_fname, core::data::io::is_not_alternative,␣
    ↪core::data::io::only_ss_from_header, true);
46    Structure_SP strctr = reader.create_structure(0);
47
48    logs << utils::LogLevel::INFO << "Selecting " << utils::to_string(which_part)<<"␣
    ↪from "<<strctr->code()<<"\n";
49    for(auto i_chain : *strctr) {
50      for(auto i_resid : *i_chain)
51        if (which_part(*i_resid) && aa_only(*i_resid)) {
52          residues.push_back(i_resid);
53        }
54    }
55
56    return strctr;
57  }
58
59  /** @brief  Reads an alignment between two proteins (PIR format) and the two␣
    ↪structures and writes PDB for the aligned parts
60   *
61   * CATEGORIES: core/data/io/pir_io;
62   * KEYWORDS:   PDB input; PIR; PDB output
63   * GROUP:      Alignments
64   * IMG: ap_aligned-1k6m-1bif.png
65   * IMG_ALT: 1K6M and 1BIF structures aligned according to HOMSTRAD database
66   */
67  int main(const int argc, const char *argv[]) {
68
69    if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about␣
    ↪missing program parameters
70
71    using core::data::sequence::Sequence_SP; // --- Sequence_SP is just a std::shared_
    ↪ptr to core::data::sequence::Sequence type
72    using namespace core::alignment;
73    using namespace core::data::structural;
74
75    // --- Create a container where the sequences will be stored
76    std::vector<Sequence_SP> sequences;
77
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)

```
  PairwiseSequenceAlignment alignment("query", sequences[0]->sequence, 0, "tmplt", sequences[1]->sequence, 0, 0.0);

  const auto select_query = core::data::structural::selectors::select_by_pir_header(*std::dynamic_pointer_cast<PirEntry>(sequences[0]));
  const auto select_tmplt = core::data::structural::selectors::select_by_pir_header(*std::dynamic_pointer_cast<PirEntry>(sequences[1]));

  // --- Read the first structure and repack its amino acid residues (the other cannot be aligned)
  Structure_SP query_structure, tmplt_structure;
  std::vector<Residue_SP> query_residues, tmplt_residues;
  if (strncmp(argv[2], "-", 1) != 0) query_structure = process_input_pdb(argv[2], query_residues,*select_query);
  if (strncmp(argv[3], "-", 1) != 0) tmplt_structure = process_input_pdb(argv[3], tmplt_residues,*select_tmplt);

  std::stringstream ss;
  core::data::io::write_edinburgh(alignment,ss,65535);
  logs << utils::LogLevel::INFO << "Input alignment\n" << ss.str() << "\n";

  // --- Retrieve aligned residues from the two structures according to the alignment object
  std::vector<Residue_SP> tmplt_residues_aligned, query_residues_aligned;   // --- container for the residues

  if (query_residues.size() == 0) alignment.alignment->get_aligned_template(tmplt_residues, tmplt_residues_aligned);
  if (tmplt_residues.size() == 0) alignment.alignment->get_aligned_query(query_residues, query_residues_aligned);

  // --- If both sets of coordinates are present - retrieve both and superimpose
  // --- Also, when both structures are given - calculate crmsd and roto-translation transformation
  if ((query_residues.size() > 0) && (tmplt_residues.size() > 0)) {
    alignment.alignment->get_aligned_query_template(query_residues, tmplt_residues, query_residues_aligned, tmplt_residues_aligned);
    std::vector<Vec3> query_xyz, tmplt_xyz;
    for (auto res:query_residues_aligned) query_xyz.push_back(*res->find_atom(" CA "));
    for (auto res:tmplt_residues_aligned) {
      tmplt_xyz.push_back(*res->find_atom(" CA "));
    }
    core::calc::structural::transformations::Crmsd<std::vector<Vec3>, std::vector<Vec3>> rms;
    std::cout << "crmsd between coordinates of " << query_xyz.size() << " CA atoms: " <<<
                 rms.crmsd(tmplt_xyz, query_xyz, query_xyz.size(), true) << "\n";
    for (auto res:tmplt_residues_aligned) for (auto atom:*res) rms.apply(*atom);
  }

  // --- Rotate the query coordinates and superimpose them on the template; print them in PDB format
  if (query_residues_aligned.size() > 0) {
    std::ofstream query_file("query.pdb");
    for (auto res:query_residues_aligned)
      for (auto atom:*res) query_file << atom->to_pdb_line() << "\n";
```

(continues on next page)

(continued from previous page)

```
    query_file.close();
  }
  // --- Print template coordinates in PDB format
  if (tmplt_residues_aligned.size() > 0) {
    std::ofstream tmplt_file("tmplt.pdb");
    for (auto res:tmplt_residues_aligned)
      for (auto atom:*res) tmplt_file << atom->to_pdb_line() << "\n";
    tmplt_file.close();
  }
}
```

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:

```
#include <math.h>

#include <iostream>
#include <random>

#include <core/calc/statistics/VonMisesDistribution.hh>
#include <core/calc/statistics/expectation_maximization.hh>
#include <core/data/io/DataTable.hh>
#include <core/calc/numeric/numerical_integration.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Reads a file with 1D data and estimates a mixture of VonMissesDistribution based on these observations.
 *
```

(continues on next page)

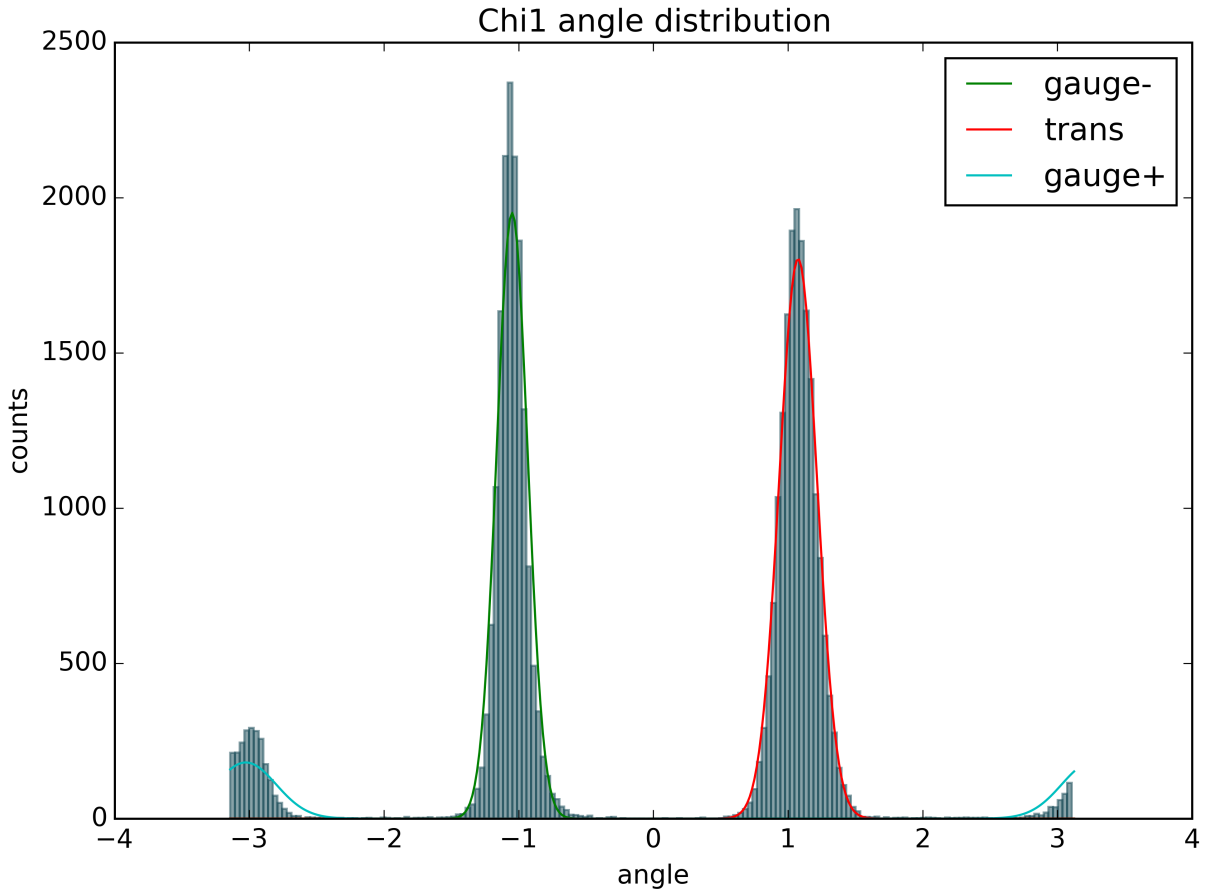
(continued from previous page)

```
 * This example may be used to approximate a $\Chi_1$ rotamer (such as VAL, THR or SER) with a mixture of
 * VonMisses distributions.
 *
 * CATEGORIES: core::calc::statistics::VonMissesDistribution
 * KEYWORDS:   von Misses distribution; estimation; expectation-maximization; statistics
 * GROUP: Statistics;
 * IMG: ap_chi1_rotamers_estimation.png
 * IMG_ALT: Distribution of Chi1 angles of THR side chains approximated with a mixture of three von Mises distribution
 */
int main(const int argc, const char *argv[]) {

  if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
  const double deg_to_rad = M_PI/180.0;
  using namespace core::calc::statistics;

  std::vector<std::vector<double>> chi_angle;
  if (argc == 2) {
    core::data::io::DataTable in(argv[1]);
    for(const auto & row : in) {
      std::vector<double> chi;
      chi.push_back( row.get<double>(0) );
      chi_angle.push_back(chi);
    }
  }
  // ---------- Three distributions - one for each rotamer
  VonMisesDistribution m(-60 * deg_to_rad, 10.0), p(60 * deg_to_rad, 10.0),
    t(-180 * deg_to_rad, 10.0); // medium, gauge-plus and gauge-minus

  std::vector<VonMisesDistribution> distributions_1D({{-60 * deg_to_rad, 10.0}, // --- gauge minus
                                                      {60 * deg_to_rad, 10.0}, // --- gauge plus
                                                      {-180 * deg_to_rad, 10.0}}); // --- trans
  std::vector<core::index1> index_1D;
  double score = expectation_maximization(chi_angle, distributions_1D, index_1D, 0.000001);
  core::index4 cnt0 = std::count(index_1D.cbegin(), index_1D.cend(), 0);
  core::index4 cnt1 = std::count(index_1D.cbegin(), index_1D.cend(), 1);
  core::index4 cnt2 = std::count(index_1D.cbegin(), index_1D.cend(), 2);
  std::cout << "# log-likelihood: " << score << "\n";
  std::cout << "# " << cnt0 << " " << distributions_1D[0]
            << " " << cnt1 << " " << distributions_1D[1]
            << " " << cnt2 << " " << distributions_1D[2] << "\n";

  for (double x = -M_PI; x < M_PI; x += 0.01)
    std::cout << utils::string_format("%6.3f %8.5f %8.5f %8.5f\n", x,
                                      cnt0 * distributions_1D[0].evaluate(x) / chi_angle.size(),
                                      cnt1 * distributions_1D[1].evaluate(x) / chi_angle.size(),
                                      cnt2 * distributions_1D[2].evaluate(x) / chi_angle.size());
```

(continues on next page)

(continued from previous page)

```
}
```



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

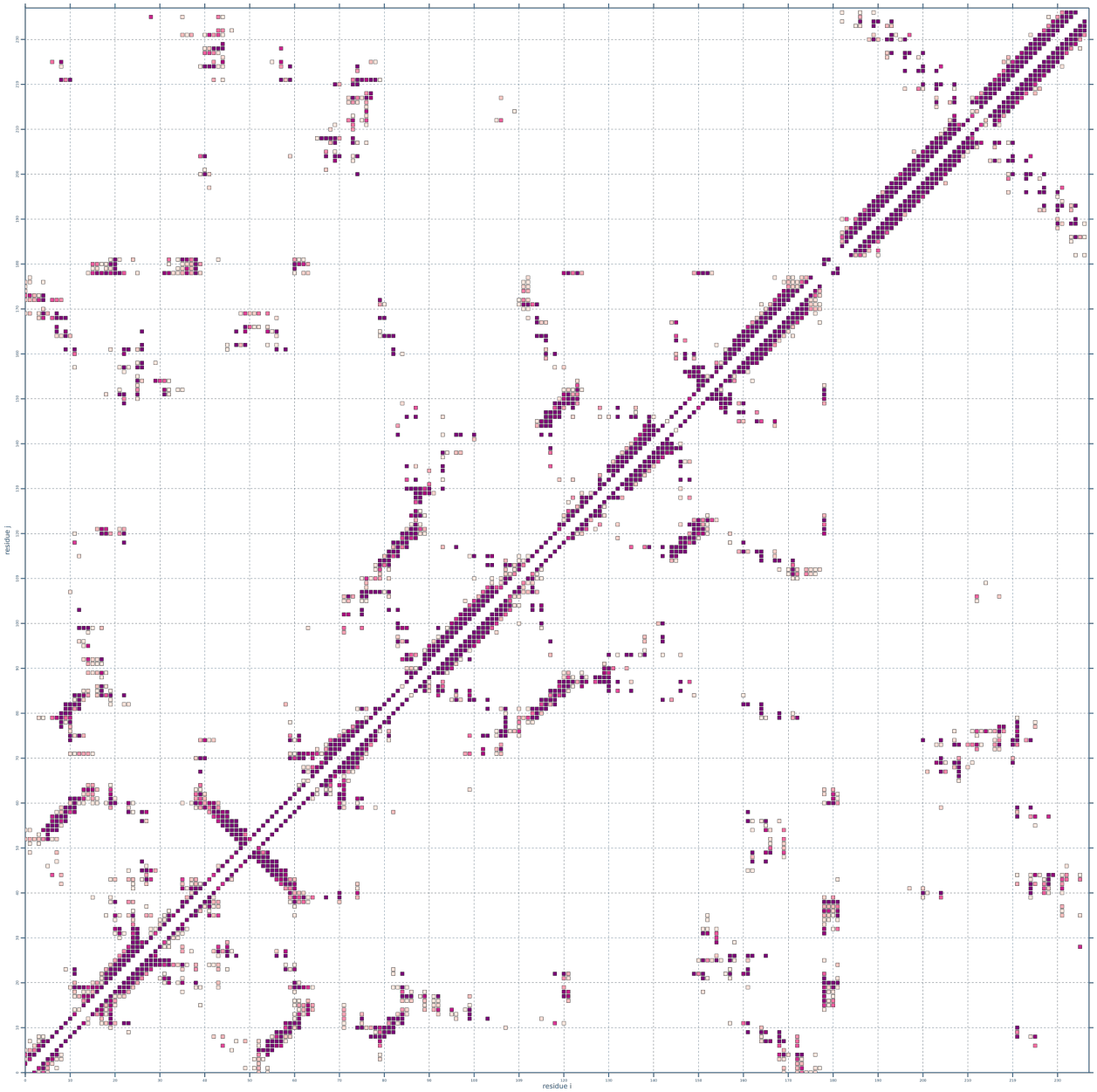
Program source:

```
#include <iostream>

#include <core/index.hh>
#include <core/data/io/Pdb.hh>
#include <core/calc/structural/interactions/ContactMap.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

```

(continues on next page)

(continued from previous page)

```
)";

/** @brief Calculates a contact map for a given protein structure
 *
 * CATEGORIES: core::calc::structural::ContactMap
 * KEYWORDS: PDB input; contact map
 * GROUP: Structure calculations;
 * IMG: ap_contact_map.png
 * IMG_ALT: Contact map calculated for 2KWI protein structure solved by NMR. The 2KWI deposit holds 51 models, the color scale shows how popular is a given contact among the models
 */
int main(const int argc, const char* argv[]) {

  if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  core::data::structural::selectors::AtomSelector_SP selector
      = std::make_shared<core::data::structural::selectors::IsSC>();
  core::data::io::PdbLineFilter filter = core::data::io::is_not_water;
  if (std::strcmp(argv[1],"CA")==0 ) {
    selector = std::make_shared<core::data::structural::selectors::IsNamedAtom>(" CA ");
    core::data::io::PdbLineFilter filter = core::data::io::is_ca;
  }
  if (std::strcmp(argv[1],"CB")==0) {
    core::data::io::PdbLineFilter filter = core::data::io::is_cb;
    selector = std::make_shared<core::data::structural::selectors::IsNamedAtom>(" CB ");
  }

  double cutoff = utils::from_string<double>(argv[3]); // The third parameter is the contact distance (in Angstroms)
  core::data::io::Pdb reader(argv[2],filter); // --- file name (PDB format, may be gzip-ped)

  core::data::structural::Structure_SP structure = reader.create_structure(0);
  core::calc::structural::interactions::ContactMap cmap(*structure, cutoff, selector);
  for (int i_model = 1; i_model < reader.count_models(); ++i_model) {
    reader.fill_structure(i_model, *structure);
    cmap.add(*structure);
  }

  std::vector<std::pair<core::index2, core::index2>> contacts;
  cmap.nonempty_indexes(contacts);

  for(const std::pair<core::index2,core::index2> ij : contacts) {
    core::index2 i_res = ij.first;
    core::index2 j_res = ij.second;
    std::cout << utils::string_format("%4d %4s %4d%c %4d %4s %4d%c %d\n", i_res,
      cmap.residue_index(i_res).chain_id.c_str(),
      cmap.residue_index(i_res).residue_id, cmap.residue_index(i_res).i_code,
      j_res, cmap.residue_index(j_res).chain_id.c_str(),
      cmap.residue_index(j_res).residue_id, cmap.residue_index(j_res).i_code,
      cmap.at(i_res, j_res, 0));
  }
}
```



ap_fit_VonMises_mixture

ap_fit_VonMisses_mixture reads a text file with 1D arbitrary observations in degrees and fits a mixture of VonMisses distributions to the data. The number of distributions to fit is determined by the starting parameters: fmuf and f$kappaf$ 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::VonMissesDistribution

Input files:

- THR_chi1.dat

Output files:

- stdout.out

Program source:

```
#include <math.h>

#include <iostream>
#include <random>

#include <core/algorithms/basic_algorithms.hh>
#include <core/algorithms/Combinations.hh>
#include <core/calc/statistics/NormalDistribution.hh>
#include <core/calc/statistics/expectation_maximization.hh>
#include <core/calc/numeric/numerical_integration.hh>
#include <core/calc/statistics/VonMisesDistribution.hh>
#include <core/calc/statistics/RobustDistributionDecorator.hh>
#include <core/data/io/DataTable.hh>

#include <utils/exit.hh>

std::string program_info = R"(

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

```

(continues on next page)

(continued from previous page)

```
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

)";

/** @brief Reads a file with 1D data and estimates a mixture of VonMisesDistribution␣
↪based on these observations.
 *
 * CATEGORIES: core::calc::statistics::VonMissesDistribution
 * KEYWORDS:   statistics; estimation; expectation-maximization
 * GROUP: Statistics;
 * IMG: ap_fit_VonMises_mixture.png
 * IMG_ALT: Distribution of Chi1 angles of THR side chains approximated with a␣
↪mixture of three von Mises distribution
 */
int main(const int argc, const char *argv[]) {

  if (argc < 3) utils::exit_OK_with_message(program_info); // --- complain about␣
↪missing program parameter
  using namespace core::calc::statistics;

  double deg_to_rad = M_PI / 180.0;

  // ---------- Read in data points (your observations) from a file
  std::vector<std::vector<double>> data_points;
  core::data::io::DataTable in(argv[1]);
  double min = 180, max = -180; // This is to detects whether angle values are in␣
↪radians or in degrees
  for (const auto &row : in) {
    std::vector<double> d;
    double v = row.get<double>(0);
    if (v < min) min = v;
    if (v > max) max = v;
    d.push_back(v);
    data_points.push_back(d);
  }
  if (((min < -M_PI) || (max > M_PI))) std::cerr << "Converting from degrees to␣
↪radians!\n";
  else deg_to_rad = 1.0;
  for (std::vector<double> &d : data_points) d[0] *= deg_to_rad;

  std::vector<double> default_params{0.0, 100.0};

  // ---------- RobustDistributionDecorator object for each distribution
  core::index1 n_distributions = atoi(argv[2]);
  std::vector<RobustDistributionDecorator<VonMisesDistribution>> r_distributions_1D;
  std::vector<RobustDistributionDecorator<VonMisesDistribution>> r_best_distributions;

  std::vector<std::vector<std::vector<double>>> initial_parameters; // --- Initial␣
↪parameters for fitting
  std::vector<core::index1> distribution_index; // --- Resulting assignment of every␣
↪data point to a distribution
```

(continues on next page)

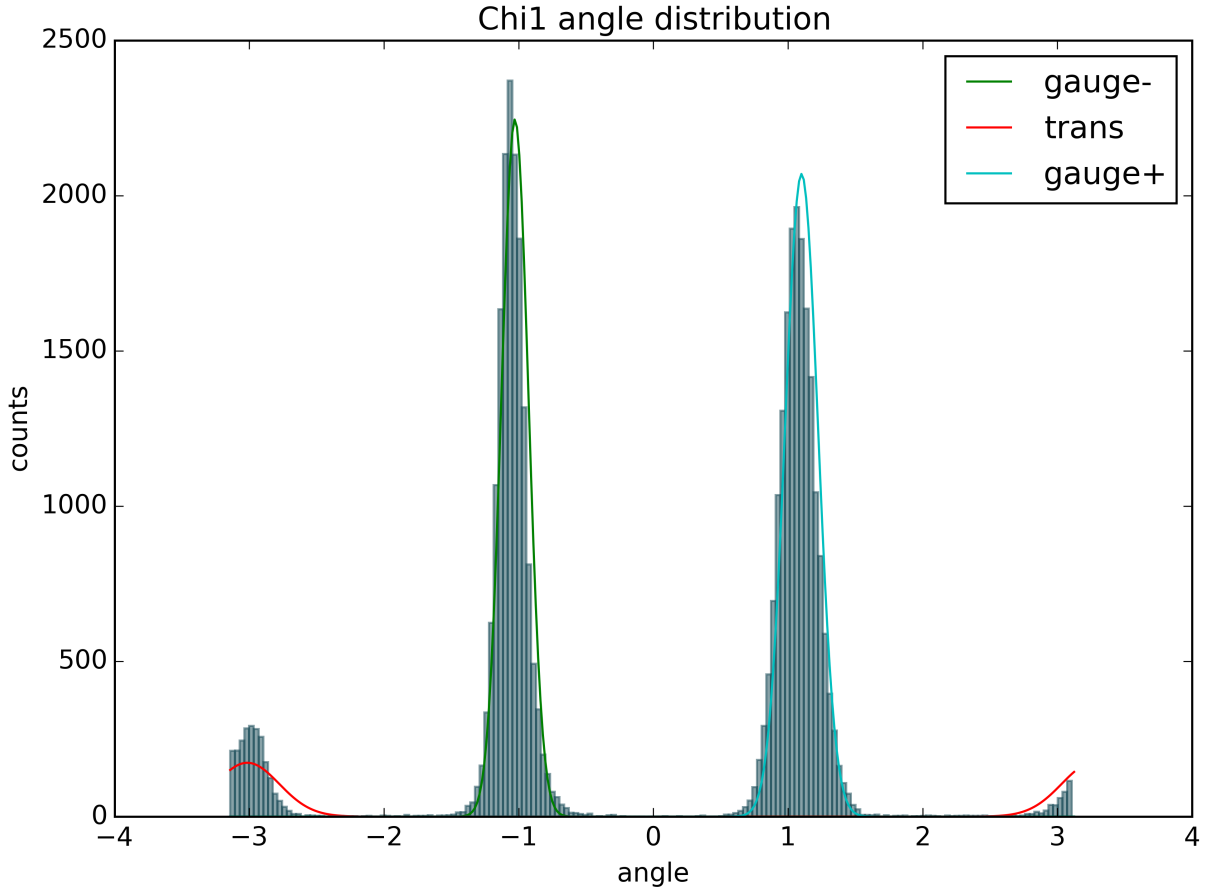
(continued from previous page)

```
  std::vector<core::index1> best_assignment; // --- Resulting assignment of every data point to a distribution

  // --- This is the case when user provided initial parameters for fitting Von Mises distribution
  if (argc >= 4) {
    // ---------- Read parameters from cmdline and create distribution objects
    std::vector<std::vector<double>> params;
    for (int i = 2; i < argc; i += 2) {
      params.push_back(std::vector<double>{atof(argv[i]) * deg_to_rad, atof(argv[i + 1])});
      r_distributions_1D.emplace_back(RobustDistributionDecorator<VonMisesDistribution>(params.back()));
      r_best_distributions.emplace_back(RobustDistributionDecorator<VonMisesDistribution>(default_params));
      initial_parameters.push_back(params);
    }
    initial_parameters.push_back(params);
  } else {
    // --- This is the case when user provided the number of distributions to be fit automatically
    n_distributions = atoi(argv[2]);
    for (size_t i = 0; i < n_distributions; ++i) {
      r_distributions_1D.emplace_back(RobustDistributionDecorator<VonMisesDistribution>(default_params));
      r_best_distributions.emplace_back(RobustDistributionDecorator<VonMisesDistribution>(default_params));
    }

    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

    core::algorithms::Combinations<double> generator(n_distributions, random_starts);
    std::vector<double> combination(n_distributions);
    std::vector<std::vector<double>> params;
    while (generator.next(combination)) {
      params.clear();
      for (size_t i_distr = 0; i_distr < n_distributions; ++i_distr)
        params.push_back(std::vector<double>{combination[i_distr] * M_PI, 100.0});
      initial_parameters.push_back(params);
    }
  }

  double best_likelihood = -std::numeric_limits<double>::max();
  // ---------- Run Expectation-Maximization algorithm
  for (size_t i_start = 0; i_start < initial_parameters.size(); ++i_start) { // --- iterate over starting points
    for (size_t i_distr = 0; i_distr < r_distributions_1D.size(); ++i_distr)  // --- loop over distributions to set each starting point
      r_distributions_1D[i_distr].copy_parameters_from(initial_parameters[i_start][i_distr]);
    double score = expectation_maximization(data_points, r_distributions_1D, distribution_index, 0.1, 100);
    if (score > best_likelihood) {
      best_likelihood = score;
      for (size_t i = 0; i < n_distributions; ++i)
        r_best_distributions[i].copy_parameters_from(r_distributions_1D[i].parameters());
```

(continues on next page)

(continued from previous page)

```
      best_assignment.swap(distribution_index);
      std::cerr << "# Best likelihood so far: " << best_likelihood << "\n";
    }
  }
  std::map<core::index1, core::index4> counts;
  core::algorithms::count_distinct(best_assignment, counts);
  const double total = std::accumulate(std::begin(counts), std::end(counts), 0,
                                       [](const size_t previous, decltype(*counts.
↪begin()) p) {
                                           return previous + p.second;
                                       });

  std::cout << "# Best likelihood " << best_likelihood << "\n# Estimated␣
↪distributions:\n";
  for (size_t i_distr = 0; i_distr < r_distributions_1D.size(); ++i_distr) {
    std::cout << counts[i_distr] / total << " " << r_best_distributions[i_distr] <<
↪"\n";
  }
}
```



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:

```
#include <iostream>
#include <map>

#include <core/data/io/Pdb.hh>
#include <core/calc/structural/interactions/HydrogenBondInteraction.hh>
#include <core/calc/structural/interactions/BackboneHBondInteraction.hh>
#include <core/calc/structural/interactions/HydrogenBondCollector.hh>
#include <core/chemical/MonomerStructureFactory.hh>
#include <utils/string_utils.hh>
#include <utils/exit.hh>
```

(continues on next page)

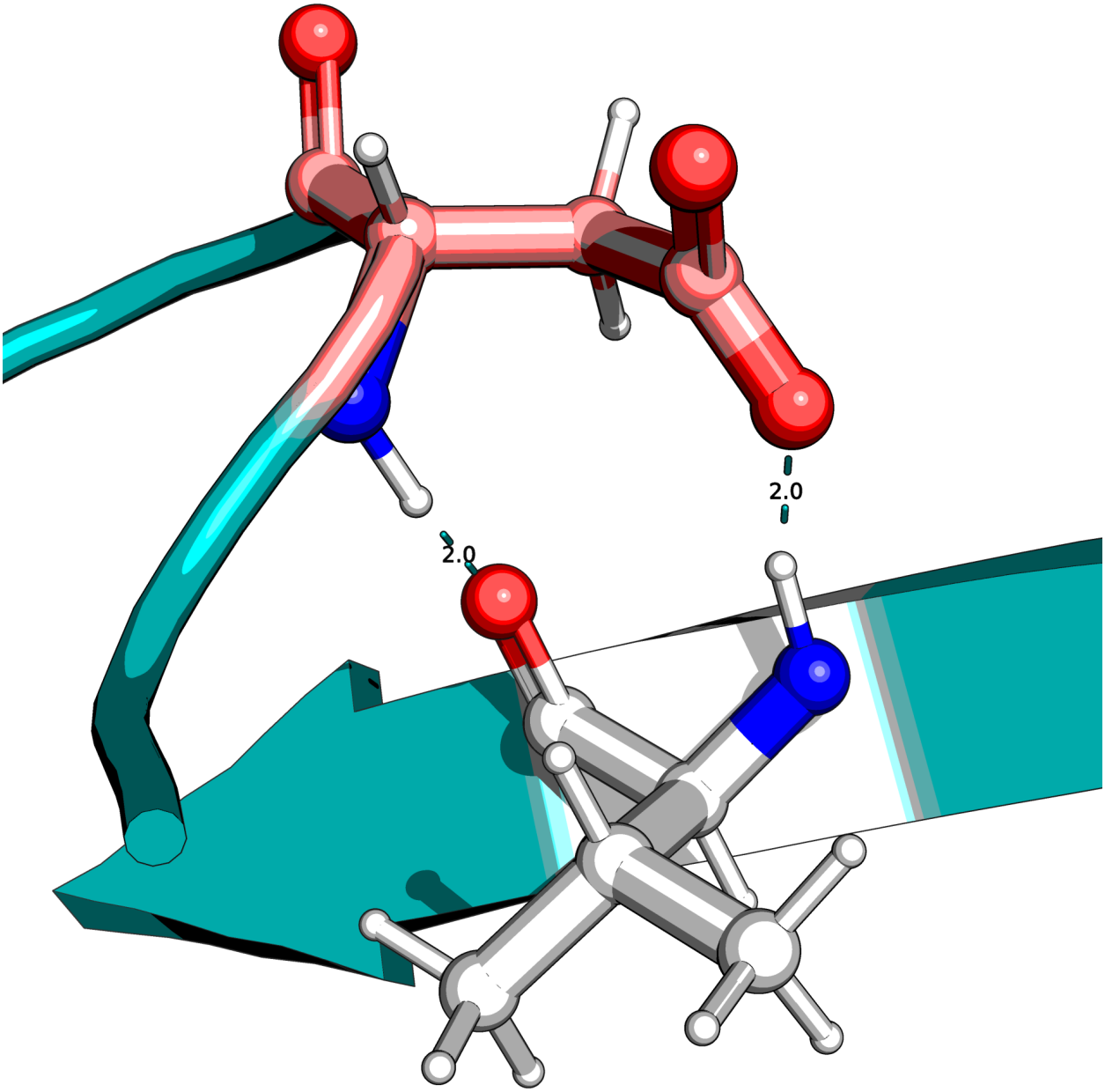
(continued from previous page)

```

std::string program_info = R"(

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):


)";

/** @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)

```
  std::vector<ResiduePair_SP> sink;
  // ---------- Iterate over all models in the input file
  for (size_t i_protein = 0; i_protein < reader.count_models(); ++i_protein) {
    sink.clear();
    core::data::structural::Structure_SP s = reader.create_structure(0);
    collector.collect(*s, sink);
    // ---------- The first loop prints backbone hydrogen bonds
    std::cout << BackboneHBondInteraction::output_header() << "\n";
    for (const ResiduePair_SP ri:sink) {
      BackboneHBondInteraction_SP bi = std::dynamic_pointer_cast
↪<BackboneHBondInteraction>(ri);
      if (bi) std::cout << *bi << "\n";
    }
    // ---------- The second loop prints hydrogen bonds involving side chain atoms
    std::cout << HydrogenBondInteraction::output_header() << "\n";
    for (const ResiduePair_SP ri:sink) {
      BackboneHBondInteraction_SP bi = std::dynamic_pointer_cast
↪<BackboneHBondInteraction>(ri);
      if (!bi) std::cout << *std::dynamic_pointer_cast<HydrogenBondInteraction>(ri) <
↪< "\n";
    }
  }
}
```



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:

```
#include <core/data/io/Pdb.hh>
#include <core/algorithms/graph_algorithms.hh>
#include <core/data/structural/BetaStructuresGraph.hh>
#include <core/calc/structural/ProteinArchitecture.hh>

#include <utils/exit.hh>

std::string program_info = R"(

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.
)";

void index_strands(core::data::structural::BetaStructuresGraph_SP g) {

  using core::data::structural::Strand_SP;

  // ---------- Firstly, let's find the first strand on the path: the one with just␣
↪one partner
  Strand_SP first_strand = nullptr;
  for (auto it = g->begin_strand(); it != g->end_strand(); ++it) {
    if (g->count_partners(*it) == 1) {
      if (first_strand == nullptr) first_strand = *it;
      else if (first_strand->length() > (*it)->length()) // there are two edge␣
↪strands, take the shorter one
```

(continues on next page)

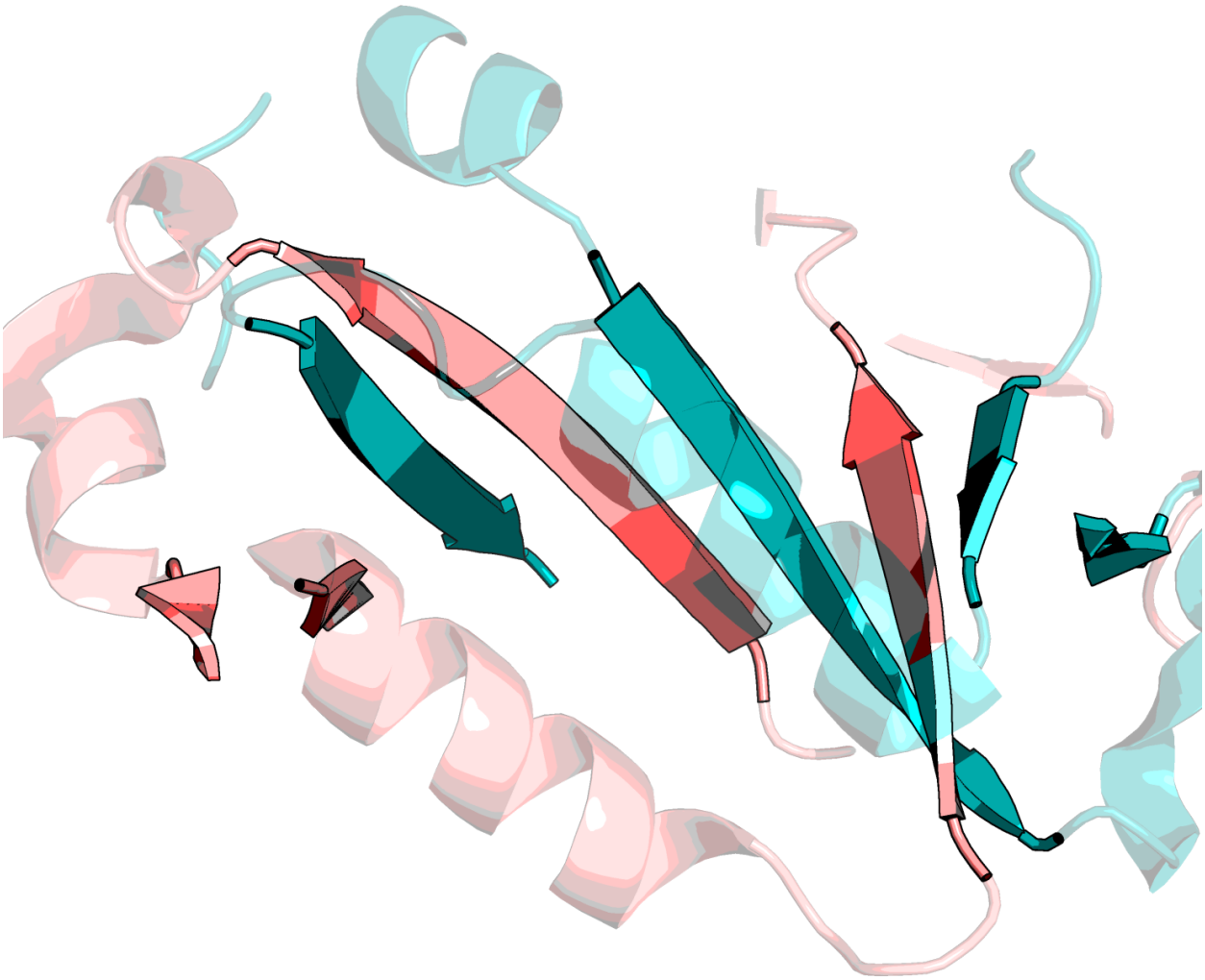
(continued from previous page)

```
        first_strand = *it;
    }
  }

  // ---------- If it's a barrel, take the shortest one
  if (first_strand == nullptr) {
    Strand_SP first_strand = *g->begin_strand();
    for (auto it = g->begin_strand(); it != g->end_strand(); ++it) {
      if (first_strand->length() > (*it)->length()) first_strand = *it;
    }
  }

  std::set<Strand_SP> visited;
  std::vector<Strand_SP> stack;
  std::vector<Strand_SP> scratch;
  stack.push_back(first_strand);
  core::index2 idx = 0;
  while(stack.size()>0) {
    // --- pop a strand from stack, mark as visited
    Strand_SP s = stack.back();
    s->strand_index_in_sheet = (++idx);
    visited.insert(s);
    stack.pop_back();

    // --- get its neighbors, push to scratch if not visited yet
    scratch.clear();
    for (auto it = g->begin_strand(s); it != g->end_strand(s); ++it)
      if(visited.find(*it)==visited.cend())
        scratch.push_back(*it);

    // --- sort neighbors
    std::sort(scratch.begin(), scratch.end(),
              [](Strand_SP lhs, Strand_SP rhs) { return rhs->length() < lhs->length();
↪ });
    // --- push from the shortest
    for(Strand_SP si:scratch) stack.push_back(si);
  }

}

struct OrderStandsInSheet {

  bool operator()(core::data::structural::Strand_SP lhs,␣
↪core::data::structural::Strand_SP rhs) { return lhs->strand_index_in_sheet < rhs->
↪strand_index_in_sheet; }
};

/** @brief Creates a BetaStructuresGraph and finds interdigitated sheets
 *
 * CATEGORIES: core::data::structural::BetaStructuresGraph
 * KEYWORDS:   PDB input
 * GROUP: Structure calculations;
 * IMG: 2fdo-7-sq.png
 * IMG_ALT: Interdigitated beta-sheet of 2FDO deposit; the two chains A and B shown␣
↪with different colors
 */
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 missing program parameter
87    using namespace core::data::structural;
88    using namespace core::data::io;
89
90  //  core::data::io::Pdb reader(argv[1], (is_not_alternative), true);
91    core::data::io::Pdb reader(argv[1], core::data::io::all_true(is_not_alternative, is_not_water), core::data::io::keep_all, true);
92    core::data::structural::Structure_SP strctr = reader.create_structure(0);
93
94    std::string sec_str;
95    for (auto chain : *strctr) sec_str += chain->create_sequence()->str();
96    core::calc::structural::ProteinArchitecture a(*strctr, false);
97    BetaStructuresGraph_SP g = a.create_strand_graph();
98
99    g->print_adjacency_matrix(std::cerr);
100
101   for (auto s_it = g->begin(); s_it != g->end(); ++s_it) {
102     auto s = *s_it;
103     for (auto nbr = g->begin_strand(s); nbr != g->end_strand(s); ++nbr) {
104       core::data::structural::Strand_SP strnd = *nbr;
105     }
106   }
107
108   std::vector<StrandPairing_SP> edges;
109   for (auto it = g->cbegin_pairings(); it != g->cend_pairings(); ++it) edges.push_back((*it).second);
110   for (StrandPairing_SP sp:edges) {
111     Strand_SP first = sp->first_strand;
112     Strand_SP second = sp->second_strand;
113     if ((*first)[0]->owner()->id() == (*second)[0]->owner()->id()) {
114       g->remove_strand_pairing(first, second);
115     }
116   }
117
118   auto sheets = core::algorithms::connected_components<BetaStructuresGraph, Strand_SP, StrandPairing_SP>(*g, 2);
119   int cnt = 0;
120   for(const auto & sheet: sheets) {
121     std::cout << utils::string_format("-------------- Sheet %d ------------------\n", ++cnt);
122     auto strnd = sheet->cbegin_strand();
123
124     index_strands(g); // --- index strands in a current sheet
125
126     std::vector<Strand_SP> strands;
127     for(auto it=sheet->cbegin_strand();it!=sheet->cend_strand();++it)
128       strands.push_back(*it);
129     std::sort(strands.begin(), strands.end(),OrderStandsInSheet{});
130     for (auto s:strands) std::cout << *s << ", has " << g->count_partners(s) << " edges\n";
131   }
132 }
```



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:

```
#include <iostream>
#include <map>

#include <core/data/io/Pdb.hh>
#include <utils/exit.hh>
#include <utils/io_utils.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/protocols/PairwiseLigandCrmsd.hh>
#include <core/calc/clustering/DistanceByValues1B.hh>
#include <core/calc/clustering/HierarchicalClustering1B.hh>

std::string program_info = R"(

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 docking 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

```

(continues on next page)

(continued from previous page)

```
)";

/** @brief Performs clustering analysis on small molecule docking poses
 *
 * CATEGORIES: core::calc::clustering::HierarchicalClustering1B
 * KEYWORDS: PDB input; clustering;
 * GROUP: Structure calculations;  Docking;
 * IMG:
 * IMG_ALT:
 */
int main(const int argc, const char* argv[]) {

  if (argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  utils::Logger l("ap_ligand_clustering");

  using namespace core::data::structural::selectors;
  AtomSelector_SP select_ligand =
      std::static_pointer_cast<AtomSelector>(std::make_shared<SelectResidueByName>(argv[1]));
  AtomSelector_SP select_ca =
      std::static_pointer_cast<AtomSelector>(std::make_shared<IsCA>());

  core::protocols::PairwiseLigandCrmsd crmsd_calculator(select_ligand, select_ca);

  std::vector<std::string> fnames = utils::read_listfile(argv[2]);
  for(const std::string & f:fnames) {
    core::data::io::Pdb reader(f, "is_not_hydrogen", false, false);
    crmsd_calculator.add_input_structure(reader.create_structure(0), f);
  }

  core::index2 min_cluster_size = (argc < 4) ? 5 : atof(argv[3]);
  std::vector<double> clustering_cutoffs;
  double max_clustering_cutoff = 0.0;
  if (argc < 4) {
    max_clustering_cutoff = 15.0;
    clustering_cutoffs.push_back(15.0);
  } else {
    for (int i = 4; i < argc; ++i) {
      clustering_cutoffs.push_back(atof(argv[i]));
      max_clustering_cutoff = std::max(max_clustering_cutoff, clustering_cutoffs.back());
    }
  }
  double evaluate_cutoff = max_clustering_cutoff * 1.5;
  double conversion_factor = 255 / evaluate_cutoff;

  crmsd_calculator.crmsd_cutoff(evaluate_cutoff);
  crmsd_calculator.set_out_matrix();
  crmsd_calculator.calculate();
  auto out = crmsd_calculator.out_matrix();

  core::calc::clustering::DistanceByValues1B distances(crmsd_calculator.tags());
  for (core::index4 i = 1; i < fnames.size(); ++i) {
    for (core::index4 j = 0; j < i; ++j) {
      if (out->has_element(i, j)) {
```

(continues on next page)

(continued from previous page)

```
        double v = out->at(i, j) * conversion_factor;
        distances.set(i, j, core::index1(v));
        distances.set(j, i, core::index1(v));
        // --- uncomment the line below to see the actual distance values together␣
↪with their converted counterparts
        // std::cerr << i << " " << j << " " << out->at(i, j) << " " << int(v) << "\n
↪";
      }
    }
  }

  core::calc::clustering::HierarchicalClustering1B hac(distances.labels(), "");
  hac.run_clustering(distances, "COMPLETE_LINK");
  for (double clustering_cutoff:clustering_cutoffs) {
    std::ofstream out(utils::string_format("clusters-%.2f.txt", clustering_cutoff));
    auto clusters = hac.get_clusters(clustering_cutoff * conversion_factor, min_
↪cluster_size);
    l << utils::LogLevel::INFO << clusters.size() << " clusters created for cutoff " <
↪< clustering_cutoff << "\n";

    for (const auto &c : clusters) {
      std::vector<std::string> el = c->cluster_items(c);
      out << el.size() << " ";
      for (const std::string &s:el) out << s << " ";
      out << "\n";
    }
    out.close();
  }
}
```

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:

```
#include <iostream>
#include <map>

#include <core/data/io/Pdb.hh>
#include <utils/string_utils.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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
```

(continues on next page)

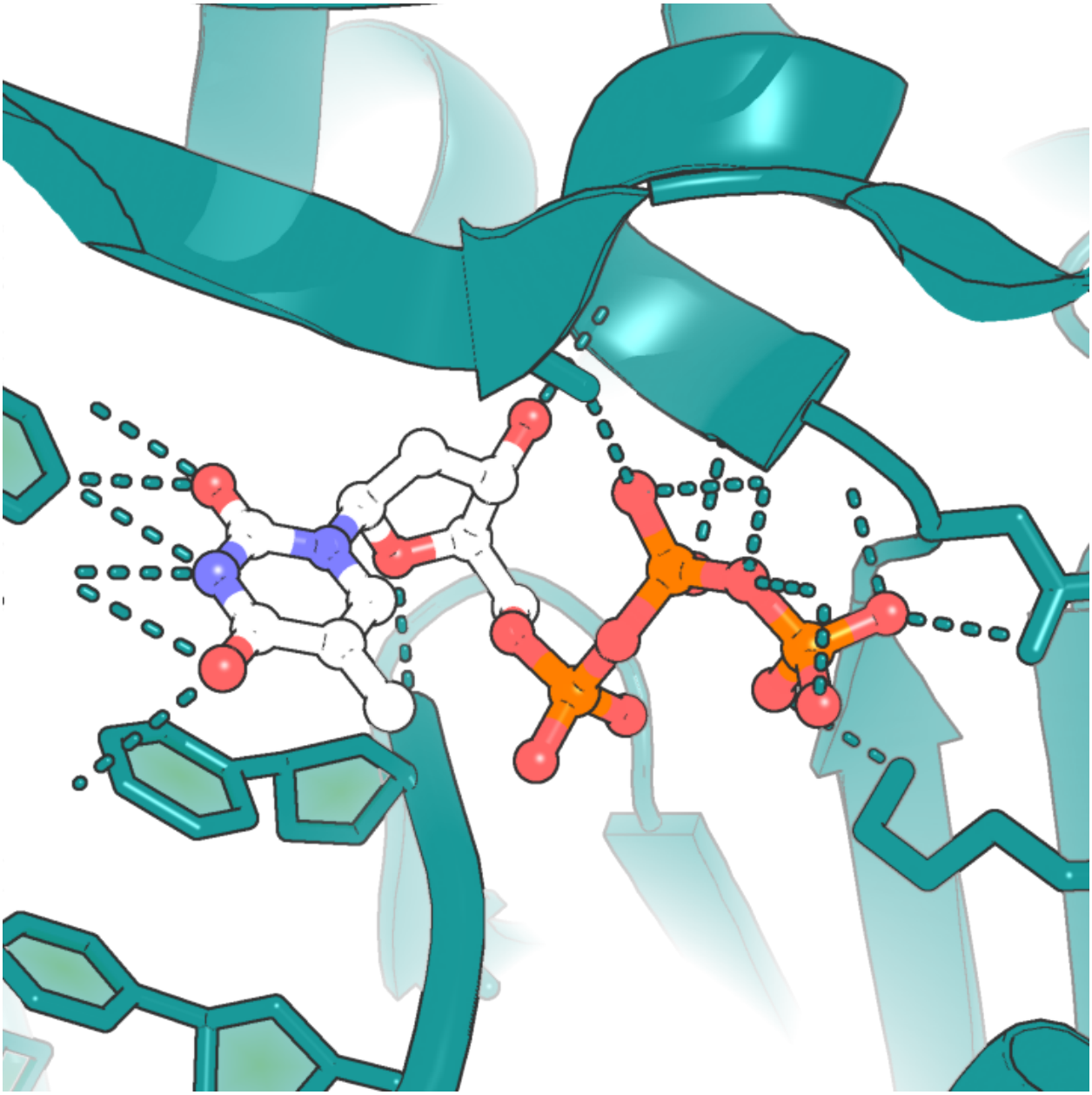
(continued from previous page)

```
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

)";

/** @brief Finds contacts between a ligand molecule and a protein.
 *
 * CATEGORIES: core::data::io::Pdb
 * KEYWORDS: PDB input; contact map; ligand
 * GROUP: Structure calculations;
 * IMG: ap_ligand_contacts.png
 * IMG_ALT: Contacts found between 5EDW protein and its ligand TTP
 */
int main(const int argc, const char* argv[]) {

  if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about␣
↪missing program parameter

  core::data::io::Pdb reader(argv[1]); // --- file name (PDB format, may be gzip-ped)

  const std::string code(argv[2]);    // --- The ligand code is the second parameter␣
↪of the program
  double cutoff = utils::from_string<double>(argv[3]); // The third parameter is the␣
↪contact distance (in Angstroms)

  std::cout << " ---- ligand ---- | --------- partner -------- | distance\n";
  std::cout << "c  res  id atname |  c  res  id   type  atname | in Angstrom\n";

  for (size_t i = 0; i < reader.count_models(); ++i) { // --- Iterate over all models␣
↪in the input file
    core::data::structural::Structure_SP strctr = reader.create_structure(i);

    // --- Here we use a standard <code>find_if</code> algorithm to find the ligand␣
↪residue by its 3-letter code
    auto ligand = std::find_if(strctr->first_residue(), strctr->last_residue(), [&
↪code](core::data::structural::Residue_SP res) {return (res->residue_type().
↪code3==code);});
    if(ligand== strctr->last_residue()) { // --- If no ligand - print a message and␣
↪take next structure
      std::cerr << "Model " << i << " of " << argv[1] << " has no " << argv[2] << "␣
↪residue\n";
      continue;
    }

```

(continues on next page)

(continued from previous page)

```
    if (reader.count_models() > 1) std::cout << "# Model " << i + 1 << "\n";
    for (auto it_resid = strctr->first_residue(); it_resid != strctr->last_residue();
↪++it_resid) {
      if(*it_resid == *ligand) continue;
      double d = (*it_resid)->min_distance(*ligand);
      if (d < cutoff) { // --- if this is close enough,
        for(auto const & ligand_atom : **ligand) {
          for(auto const & other_atom : **it_resid) {
            if(ligand_atom->distance_to(*other_atom) <= cutoff) {
              std::cout << utils::string_format("%4s  %3s %4d %4s     %4s  %3s %4d
↪%6s %4s   %6.3f\n",
                                                (**ligand).owner()->id().c_str(),
                                                (**ligand).residue_type().code3.c_
↪str(), (**ligand).id(),
                                                ligand_atom->atom_name().c_str(),
                                                (**it_resid).owner()->id().c_str(),
                                                (**it_resid).residue_type().code3.c_
↪str(), (**it_resid).id(),
                                                core::chemical::monomer_type_
↪name((**it_resid).residue_type()).c_str(),
                                                other_atom->atom_name().c_str(),
                                                ligand_atom->distance_to(*other_
↪atom));
            }
          }
        }
      }
    }
  }
}
```



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:

```
#include <iostream>
#include <random>

#include <core/index.hh>
#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/calc/structural/transformations/Rototranslation.hh>
#include <core/calc/numeric/Pca3.hh>
#include <utils/exit.hh>
#include <core/calc/structural/angles.hh>

std::string program_info = R"(

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
```

(continues on next page)

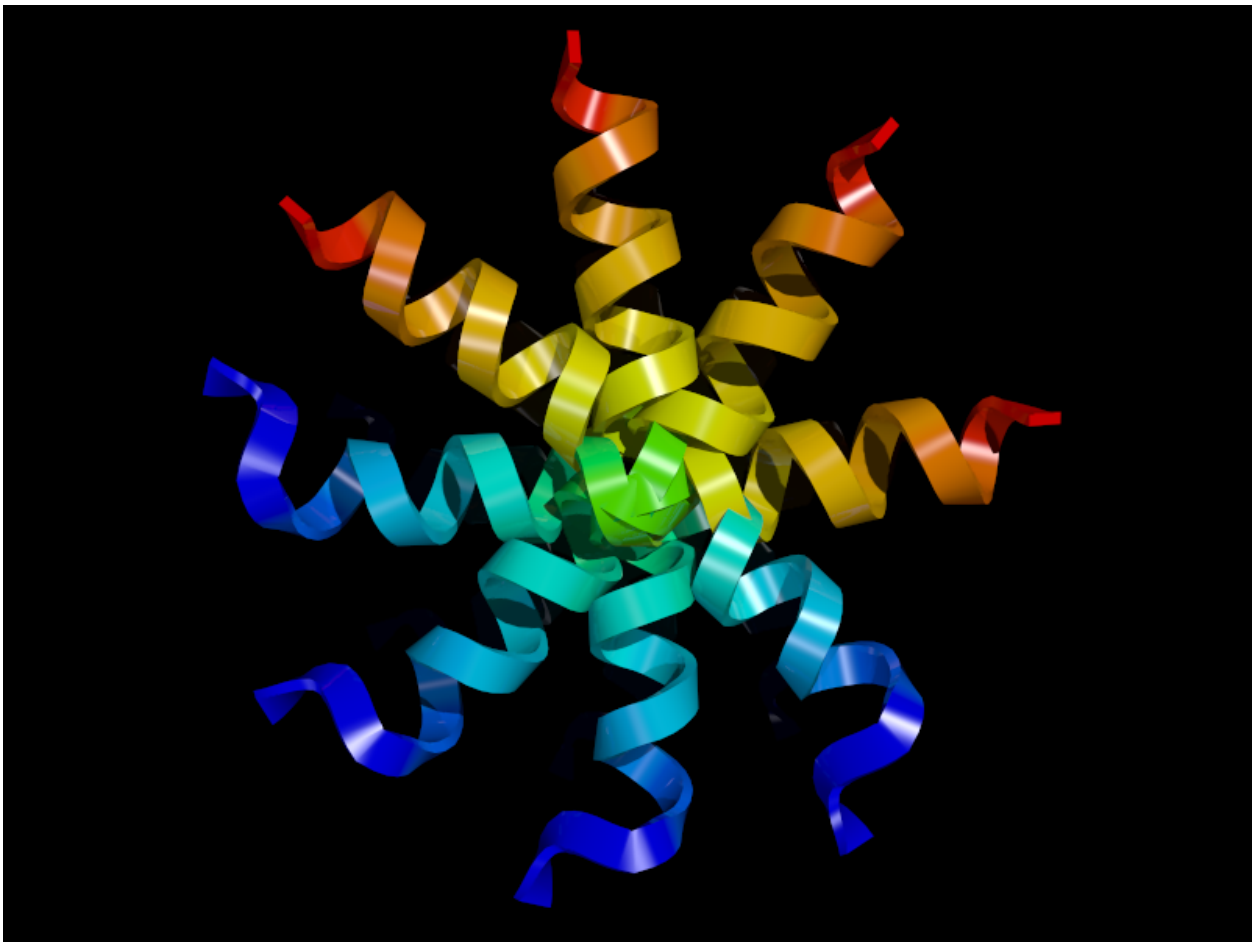
(continued from previous page)

```
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

)";

/** @brief Shows how to rotate a piece of a protein structure
 *
 * CATEGORIES: core/calc/numeric/PCA.hh
 * KEYWORDS: PDB input; structural fragment; structure selectors; PCA; transformations
 * GROUP: Structure calculations;
 * IMG: helices.png
 * IMG_ALT: Alpha helix rotated a few times by a fixed angle
 */
int main(const int argc, const char *argv[]) {

  using namespace core::data::basic; // --- for Vec3 and Array2D

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

  core::data::io::Pdb reader(argv[1]); // --- read the input PDB file
  core::data::structural::Structure_SP structure_sp = reader.create_structure(0); // -
↪-- create a structure corresponding to the first model

  std::vector<core::data::structural::PdbAtom_SP> points3d;

  // --- if a chain-ID and residue range were also given, create a selector to␣
↪extract the relevant part of the input
  if (argc > 4) {
    std::string selection_string = utils::string_format("%c:%d-%d", argv[2][0],␣
↪atoi(argv[3]), atoi(argv[4]));
    core::data::structural::selectors::SelectChainResidues selector(selection_string);
    // --- if selector selects (returns true), copy the atoms
    for (auto atom_it = structure_sp->first_atom(); atom_it != structure_sp->last_
↪atom(); ++atom_it)
      if (selector((*atom_it)->owner())) points3d.push_back(*atom_it);
  } else { // --- If there is no selection, copy all the atoms from the given␣
↪structure
    for (auto atom_it = structure_sp->first_atom(); atom_it != structure_sp->last_
↪atom(); ++atom_it)
      points3d.push_back(*atom_it);
  }
  core::calc::numeric::Pca3 pca3(points3d);
  auto rt = pca3.create_transformation();
  std::cout << "MODEL        1\n";
  for (auto atom : points3d) {
    rt.apply(*atom);
    std::cout << (atom)->to_pdb_line() << "\n";
  }
  std::cout << "ENDMDL\n";
```

(continues on next page)

(continued from previous page)

```
  auto rt2 = core::calc::structural::transformations::Rototranslation::around_axis(
    Vec3(0,0,1),core::calc::structural::to_radians(45.0),Vec3(0,0,0));
  for(int i=2;i<5;i++) {
    std::cout << "MODEL        " << i << "\n";
    for (auto atom : points3d) {
      rt2.apply(*atom);
      std::cout << atom->to_pdb_line() << "\n";
    }
    std::cout << "ENDMDL\n";
  }

}
```



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:

```
#include <iostream>
#include <chrono>

#include <core/data/io/fasta_io.hh>

#include <core/alignment/NWAligner.hh>
#include <core/alignment/scoring/SimilarityMatrix.hh>
#include <core/alignment/scoring/SimilarityMatrixScore.hh>
#include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>
#include <core/calc/statistics/OnlineStatistics.hh>
#include <core/calc/statistics/NormalDistribution.hh>
#include <core/calc/statistics/Random.hh>
#include <core/protocols/PairwiseSequenceIdentityProtocol.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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.
```

(continues on next page)

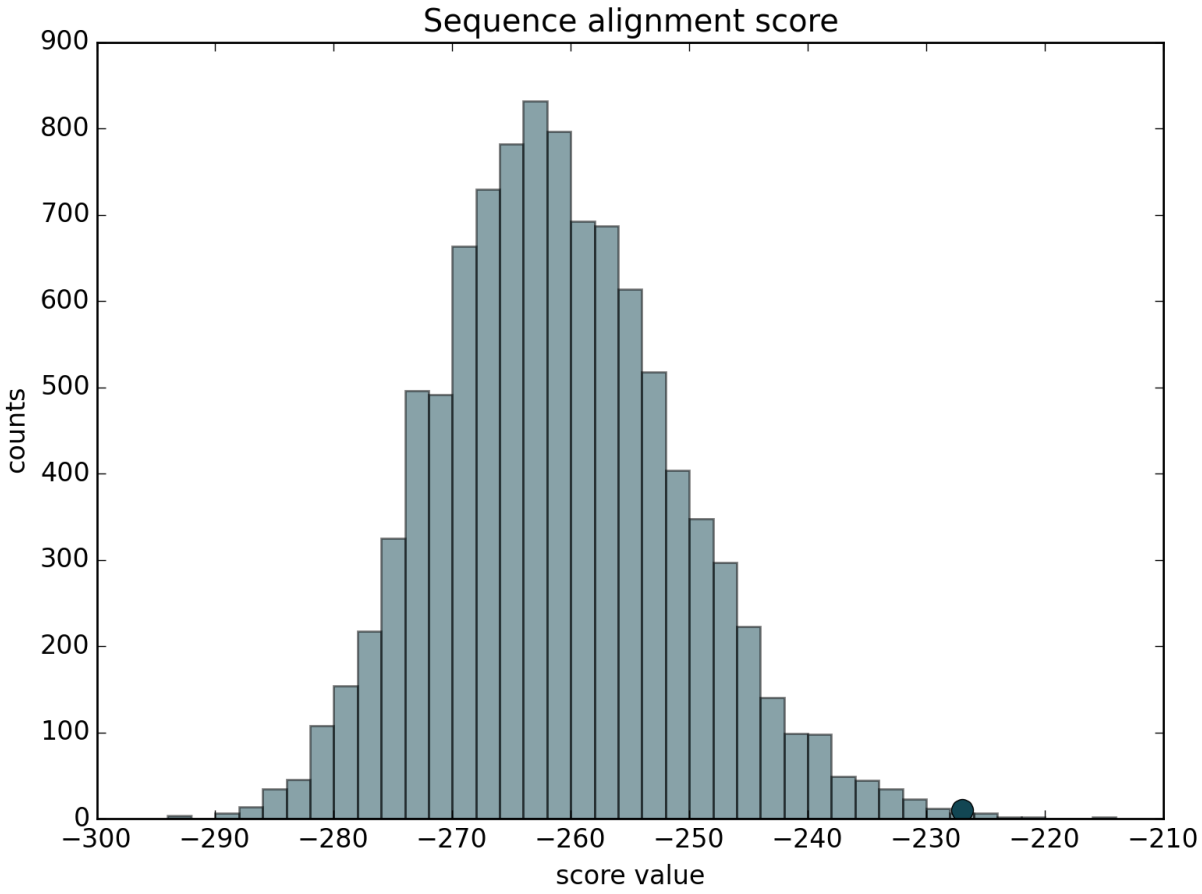
(continued from previous page)

```
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

)";

/** @brief Calculate global sequence alignment scores with one sequence randomly shuffled and estimates alignment p-value
 *
 * CATEGORIES: core::alignment::NWAligner
 * KEYWORDS:   FASTA input; Needleman-Wunsch; sequence alignment; statistics
 * GROUP:      Alignments
 * IMG: ap_shuffled_sequence_alignment.png
 * IMG_ALT: Statistics of random sequence alignment between 1BC6 and SFL95851.1
 */
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;
  using namespace core::alignment::scoring;

  core::index2 n_shuffles = (argc > 3) ? atoi(argv[3]) : 1000; // --- The number of random shuffles

  // --- Read the query sequence
  std::vector<std::shared_ptr<Sequence>> input_sequences;
  read_fasta_file(argv[1], input_sequences);

  // --- find longest sequence to initialize aligner object large enough
  unsigned max_len = std::max(input_sequences[0]->length(), input_sequences[1]->length());

  // --- create aligner object
  core::alignment::NWAligner<short, SimilarityMatrixScore<short>> aligner(max_len);

  // --- read similarity matrix from a file (i.e. BLOSUM62)
  std::string substitution_matrix_name = (argc > 2) ? argv[2] : "BLOSUM62";
  NcbiSimilarityMatrix_SP sim_m = NcbiSimilarityMatrixFactory::get().get_matrix("BLOSUM62");

  // --- go through all db sequences and align them with the given query
  auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!

  std::string j_seq_copy = input_sequences[1]->sequence;
  SimilarityMatrixScore<short> score(input_sequences[0]->sequence, j_seq_copy, *sim_m);
  // --- find score of the alignment; just the score - this is faster than aligning and keeping backtracking info
```

(continues on next page)

(continued from previous page)

```
  short result = aligner.align_for_score(-10, -1, score);

  core::calc::statistics::Random & r = core::calc::statistics::Random::get();
  r.seed(12345);  // --- seed the generator for repeatable results
  core::calc::statistics::OnlineStatistics stats; // --- online (on-the fly) statistics calculator
  for (size_t i = 0; i < n_shuffles; ++i) {
    shuffle(j_seq_copy.begin(), j_seq_copy.end(), r);
    SimilarityMatrixScore<short> score(input_sequences[0]->sequence, j_seq_copy, *sim_m);
    short res = aligner.align_for_score(-10, -1, score);
    stats(res);
    std::cout << res << "\n";
  }
  auto end = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> time_span = std::chrono::duration_cast<std::chrono::duration<double>>(end - start);
  std::cerr << "# " << n_shuffles << " alignment shuffled scores computed within "
            << time_span.count() << " [s]\n";
  std::cout << "# alignment score: " << result << "\n";
  std::cout << "# normal p-value, avg, sdev: "
            << 1 - core::calc::statistics::NormalDistribution::cdf(result, stats.avg(), sqrt(stats.var())) << " "
            << stats.avg()
            << " " << sqrt(stats.var()) << "\n";
  core::protocols::PairwiseSequenceIdentityProtocol protocol;
  protocol.substitution_matrix("BLOSUM62").gap_open(-10).gap_extend(-1);
  protocol.add_input_sequence(input_sequences[0]);
  protocol.add_input_sequence(input_sequences[1]);
  protocol.run();
  std::cout << "# same value calculated by a library function: " << protocol.count_identical(0, 1) << "\n";
}
```



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

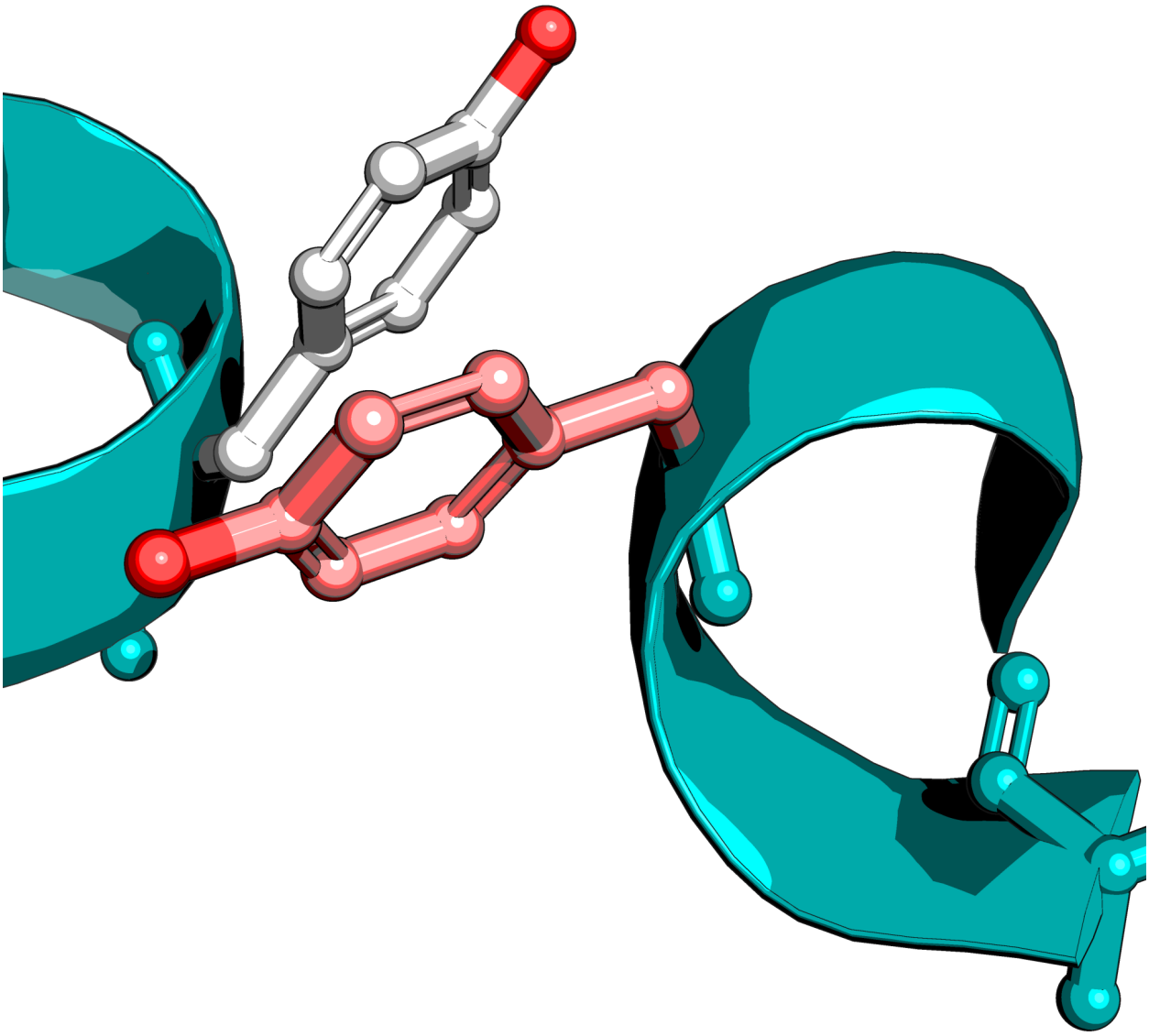
Program source:

```
#include <iostream>
#include <core/data/io/Pdb.hh>
#include <core/calc/structural/interactions/StackingInteraction.hh>
#include <core/calc/structural/interactions/StackingInteractionCollector.hh>
#include <utils/exit.hh>
#include <utils/LogManager.hh>

std::string program_info = R"(

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

)";

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

/** @brief Finds stacking interactions in a given PDB file.
 *
 * CATEGORIES: core::calc::structural::interactions::StackingInteraction
```

(continues on next page)

(continued from previous page)

```
 * KEYWORDS:   PDB input; PDB line filter; stacking interactions
 * GROUP: Structure calculations;
 * IMG: ap_stacking_interactions_sq.png
 * IMG_ALT: Two tyrosine residues in stacking interaction
 */
int main(const int argc, const char *argv[]) {

  if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about␣
↪missing program parameter
  utils::LogManager::INFO();                        // --- INFO is the default logging␣
↪level; set it to FINE to see more

  // ---------- Read a PDB file given as an argument to this program
  Pdb reader(argv[1],                 // --- input PDB file
      all_true(is_not_water, is_not_alternative, is_not_hydrogen,
      invert_filter(is_bb)),                        // --- Inverted backbone selector ␣
↪reads only side chains
      core::data::io::only_ss_from_header, true);                          // --- yes,␣
↪read header

  core::data::structural::Structure_SP s = reader.create_structure(0);

  // ---------- 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]);

  StackingInteractionCollector collector=StackingInteractionCollector();
  std::vector<ResiduePair_SP> sink;
  collector.collect(*s,sink);
  std::cout << StackingInteraction::output_header()<<"\n";

  for (const ResiduePair_SP ri:sink) {
    StackingInteraction_SP bi = std::dynamic_pointer_cast<StackingInteraction>(ri);
    if (bi) std::cout << *bi << "\n";
  }
}
```



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

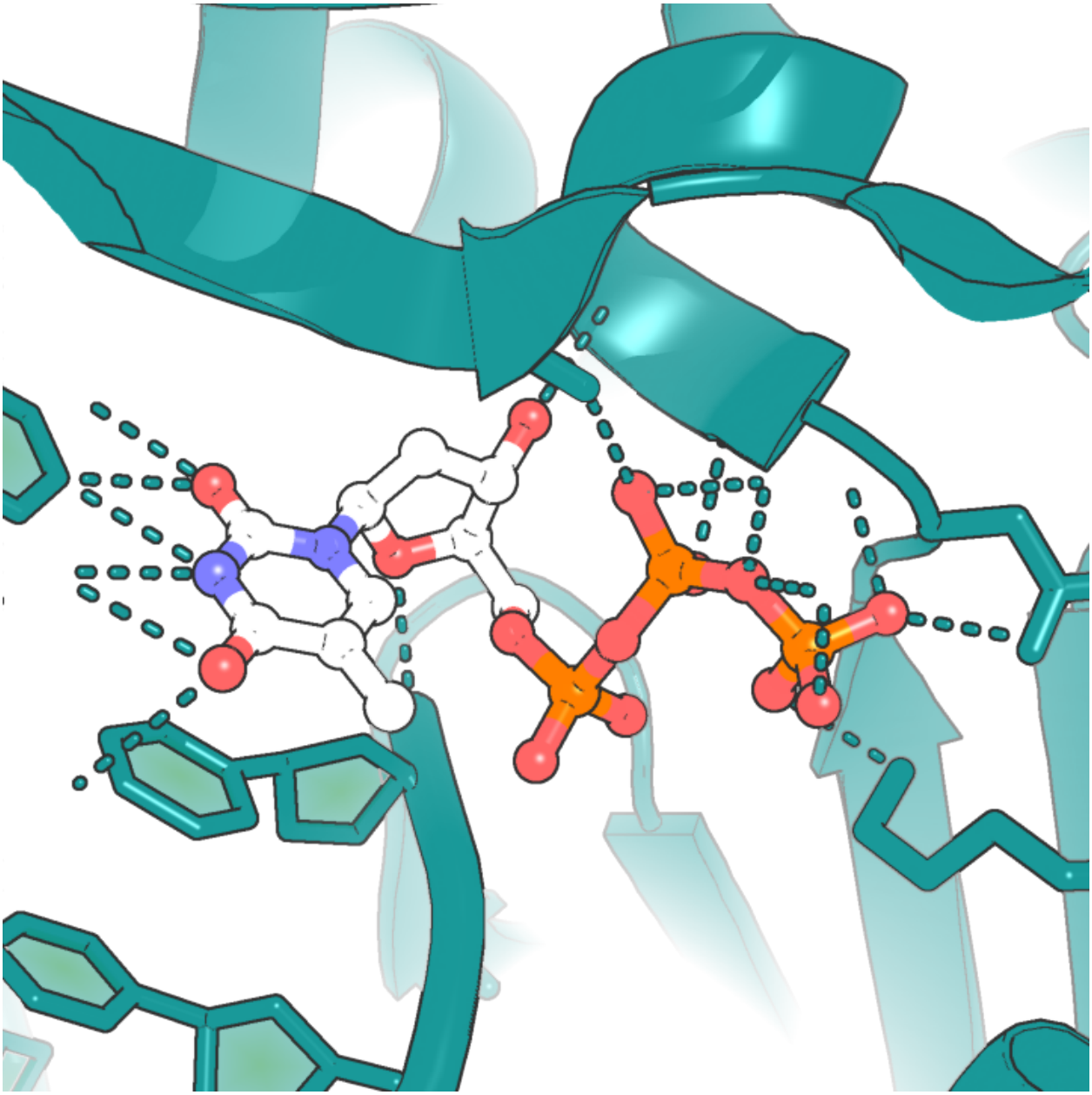
Program source:

```
#include <iostream>
#include <map>

#include <core/data/io/Pdb.hh>
#include <core/calc/structural/interactions/VdWInteraction.hh>
#include <core/calc/structural/interactions/VdWInteractionCollector.hh>
#include <utils/string_utils.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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):


)";

/** @brief Finds all van der Waals interactions in a given protein structure.
 *
 * CATEGORIES: core::calc::structural::interactions::VdWInteraction
 * KEYWORDS: PDB input; interactions
 * GROUP: Structure calculations;
```

(continues on next page)

(continued from previous page)

```
 * IMG: ap_ligand_contacts.png
 * IMG_ALT: Contacts found between 5EDW protein and its ligand TTP
 */
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::calc::structural::interactions;

  VdWInteractionCollector collector;
  for (size_t i_protein = 1; i_protein < argc; ++i_protein) { // --- Iterate over all models in the input file
    core::data::io::Pdb reader(argv[i_protein]); // --- file name (PDB format, may be gzip-ped)

    for (size_t i_model = 0; i_model < reader.count_models(); ++i_model) { // --- Iterate over all models in the input file
      std::vector<ResiduePair_SP> sink;
      core::data::structural::Structure_SP strctr = reader.create_structure(i_model);
      collector.collect(*strctr, sink);

      std::cout << VdWInteraction::output_header()<<"\n";
      for (const ResiduePair_SP ri:sink) {
        VdWInteraction_SP bi = std::dynamic_pointer_cast<VdWInteraction>(ri);
        if (bi) std::cout << *bi << "\n";
      }
    }
  }
}
```



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:

```
#include <iostream>
#include <iomanip>

#include <core/algorithms/predicates.hh>
#include <core/alignment/NWAligner.hh>
#include <core/alignment/scoring/SimilarityMatrix.hh>
#include <core/alignment/scoring/SimilarityMatrixScore.hh>
#include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>
#include <core/chemical/AAHydrophobicity.hh>
#include <core/data/io/Pdb.hh>
#include <core/data/io/clustalw_io.hh>

#include <utils/exit.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
```

(continues on next page)

(continued from previous page)

```

std::string program_info = R"(

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
)";

/** @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
 *
 * CATEGORIES: core/chemical/AAHydrophobicity;
 * KEYWORDS:   PDB input; hydrophobicity; structure selectors; PDB line filter; sequence alignment; MSA input
 * GROUP:      Sequence calculations
 */
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; // is_not_alternative, Pdb and read_clustalw_file() are from this namespace
  using namespace core::data::sequence;
  using namespace core::data::structural; // Structure and Residue come from here

  using namespace core::alignment::scoring;

  Pdb reader(argv[1], all_true(is_not_alternative, is_not_water));
  Structure_SP strctr = reader.create_structure(0); // create a Structure object from the first model found in the input file
  Chain & first_chain = *(*strctr)[0]; // --- We assume the first chain is the one used in MSA
  first_chain.erase(std::remove_if(first_chain.begin(), first_chain.end(),
      core::algorithms::Not<selectors::IsAA>(selectors::IsAA())), first_chain.end());

```

(continues on next page)

(continued from previous page)

```
  std::vector<double> kd_values;
  const core::chemical::AAHydrophobicity &kd_scale = core::chemical::AAHydrophobicity::KyteDoolittle;

  std::ofstream out("out.pdb");
  // ---------- The case when we have both a PDB file and a multiple sewuence alignment (.aln file)
  if (argc ==4) {
    std::vector<Sequence_SP> msa;   // --- placeholder for aligned sequences
    core::data::io::read_clustalw_file(argv[2],msa); // --- read the MSA and store sequences in a vector

    // ---------- Find the reference sequence in the alignment
    std::string ref_sequence_name(argv[3]); // --- the name of the sequence
    auto s = std::find_if(msa.begin(), msa.end(),
      [&ref_sequence_name](Sequence_SP s) { return s->header().find(ref_sequence_name) != std::string::npos; });
    if (s == msa.end())
      utils::exit_OK_with_message(
        "Can't find the reference sequence in the given MSA. Is the name correct: " + ref_sequence_name);
    Sequence_SP ref_sequence = *s;

    // --- Create a sequence object for the first chain of the PDB deposit
    core::data::sequence::SecondaryStructure_SP pdb_seq = first_chain.create_sequence();

    // ---------- we have to align the reference sequence with the sequence found in the given PDB file
    // ---------- as they might differ; we set PDB sequence to be a query and the reference - as a template
    unsigned max_len = std::max(pdb_seq->length(), ref_sequence->length());
    core::alignment::NWAligner<short, SimilarityMatrixScore<short>> aligner(max_len);
    NcbiSimilarityMatrix_SP sim_m = NcbiSimilarityMatrixFactory::get().get_matrix("BLOSUM62");
    SimilarityMatrixScore<short> score(pdb_seq->sequence, ref_sequence->sequence, *sim_m);
    aligner.align(-10, -1, score);
    auto alignment = aligner.backtrace();

    std::cout << "#msa_col aa_col aa res_id : avg_KD n_aa\n";
    // ---------- Iterate over all columns of the MSA
    for (core::index2 i_res = 0; i_res < ref_sequence->length(); ++i_res) {
      if (ref_sequence->get_monomer(i_res).is_gap()) continue;
      int j = alignment->which_query_for_template(i_res); // --- -1 denotes a gap, otherwise the index is non-negative
      if (j < 0) continue;
      double avg_kd = 0;
      double n = 0;
      for (Sequence_SP si:msa) {
        if (!si->get_monomer(i_res).is_gap()) {
          avg_kd += kd_scale.hydrophobicity(si->get_monomer(i_res));
          ++n;
        }
      }
      std::cout
        << utils::string_format("%4d %4d %c %4d : %5.2f %3d\n", i_res, j, first_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.
↪5 and b-factor can't be negative
108       for (const PdbAtom_SP &a : *(first_chain[j])) {
109         a->b_factor(avg_kd);
110         out << a->to_pdb_line() << "\n";
111       }
112     }
113
114     return 0;
115   }
116
117   // ---------- The case when we have only PDB file : iIterate over all residues in␣
↪the structure
118   for (auto res_it = strctr->first_residue(); res_it != strctr->last_residue(); ++res_
↪it) {
119     double val = kd_scale.hydrophobicity((*res_it)->residue_type()) + 5.0;
120
121     for (const PdbAtom_SP &a : **res_it) {
122       a->b_factor(val);
123       out << a->to_pdb_line() << "\n";
124     }
125   }
126   out.close();
127 }
```

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 small500_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:

```
#include <utils/exit.hh>
#include <core/alignment/aligner_factory.hh>
#include <core/data/basic/Array2D.hh>
#include <core/data/sequence/Sequence.hh>
#include <core/protocols/AlignmentPValuesProtocol.hh>
#include <core/data/io/fasta_io.hh>
#include <core/data/basic/SparseMap2D.hh>

std::string program_info = R"(

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)

```

(continues on next page)

(continued from previous page)

```
USAGE:
    ap_AlignmentPValuesProtocol input.fasta

EXAMPLE:
    ap_AlignmentPValuesProtocol small500_95identical.fasta

)";

/** @brief Uses AlignmentPValuesProtocol protocol to calculate all pairwise p-values for a given set of sequences
 *
 * CATEGORIES: core/protocols/AlignmentPValuesProtocol.hh
 * KEYWORDS:   FASTA input; sequence alignment; statistics
 * GROUP:      Alignments
 */
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::sequence;
  using namespace core::protocols;
  using namespace core::alignment;

  core::protocols::AlignmentPValuesProtocol protocol;
  protocol.gap_open(-10).gap_extend(-1).substitution_matrix("BLOSUM62").keep_alignments(true).
    alignment_method(AlignmentType::SEMIGLOBAL_ALIGNMENT).keep_alignments(true).n_threads(4);
  protocol.n_shuffles(30).p_value_cutoff(0.01);

  std::vector<Sequence_SP> input_sequences;
  core::data::io::read_fasta_file(argv[1], input_sequences);
  for(Sequence_SP si: input_sequences) protocol.add_input_sequence(si);

  auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
  protocol.run();
  auto end = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> time_span = std::chrono::duration_cast<std::chrono::duration<double>>(end - start);
  std::cerr << input_sequences.size() * (input_sequences.size() - 1) / 2.0
            << " global alignment sequence similarities calculated within " << time_span.count() << " [s]\n";

  protocol.print_p_values(std::cout);
}
```

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:

```
#include <vector>
#include <string>
#include <fstream>
#include <iostream>

#include <cstring>


#include <core/data/io/Pdb.hh>
#include <utils/io_utils.hh>

#include <utils/options/output_options.hh>
#include <utils/options/input_options.hh>
#include <core/protocols/LigandsOnGridProtocol.hh>
#include <utils/exit.hh>

```

(continues on next page)

(continued from previous page)

```
using namespace core::data::io;           // PDB is from this namespace
using namespace core::data::structural;
using namespace core::data::structural::selectors;
using namespace core::calc::structural;
using namespace utils;
using namespace std;


std::string program_info = R"(

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

USAGE:
    ap_LigandsOnGridProtocol box_grid_width models-list
)";

/** @brief Reads list of pdb files and creates grid with ligands in it.
 *
 * The first model on list (index = 0 ) is the representative one.
 *
 * CATEGORIES: core::protocols::LigandsOnGridProtocol
 * GROUP: Structure calculations;  Docking;
 * KEYWORDS: PDB input;
 */
int main(const int argc, const char *argv[]) {

  if (argc < 3) utils::exit_OK_with_message(program_info);

  std::vector<std::string> pdb_files; //vector of string file names
  utils::read_listfile(argv[2], pdb_files);
  // assumes that ligand is in B chain
  AtomSelector_SP select_ligand = std::static_pointer_cast<AtomSelector>(std::make_shared<ChainSelector>("B"));
  // assumes that receptor is in A chain
  AtomSelector_SP select_receptor = std::static_pointer_cast<AtomSelector>(std::make_shared<ChainSelector>("A"));
  // creating LigandsOnGridProtocol object
  core::protocols::LigandsOnGridProtocol ligpro = core::protocols::LigandsOnGridProtocol(select_ligand, select_receptor);
  // setting box_grid_size
  ligpro.box_grid_size(atof(argv[1]));
  PdbLineFilter filter = core::data::io::is_ca;
  Pdb pdb = Pdb(pdb_files[0], filter);
  //creating structure from first file on a list
  Structure_SP strctr = pdb.create_structure(0);
  // adding structure to LigandsOnGridProtocol object
  ligpro.add_input_structure(strctr);
  // loading and adding rest of structures from cat_list
  for (int i = 1; i < pdb_files.size(); i++) {
    Pdb pdb = Pdb(pdb_files[i], filter);
    pdb.fill_structure(0, *strctr);
//    std::cout << pdb_files[i] << "\n";
    ligpro.add_input_structure(strctr);
  }

  // running the calculation to put ligands into grid
```

(continues on next page)

(continued from previous page)

```
  ligpro.calculate();
  // creating a copy of a vector with hashes from filled grid cells
  std::vector<core::index4> grid_cells = ligpro.grid()->filled_cells();

  core::index4 index = 0; // variable to remember iterator for biggest cell
  int size = 100; //variable to remember SIZE
  while (grid_cells.size() > 0 and size >= 10) { //until vector is not empty and there are cells bigger than 10
    size = 0;
    for (core::index4 i = 0; i < grid_cells.size(); i++) { // iterating over cells
      //std::cout<<grid_cells[i]<<" "<<ligpro.grid()->get_cell(grid_cells[i]).size()<<"\n";
      if (ligpro.grid()->get_cell(grid_cells[i]).size() > size) { //checking if current cell size is bigger then SIZE
        size = ligpro.grid()->get_cell(grid_cells[i]).size(); //if yes, changing size and index values
        index = grid_cells[i];
      }
    }
    if (size >= 10) {
      // std::cout<<index<<" "<<ligpro.grid()->get_cell(index).size()<<"\n";

      std::vector<core::index4> hashes; //vector to store neighbor cells
      ligpro.grid()->get_neighbor_cells(index, hashes); //getting all hashes for neighbors cells
      for (core::index4 ind = 0; ind < hashes.size(); ind++) {
        grid_cells.erase(std::remove(grid_cells.begin(), grid_cells.end(), hashes[ind]),
                         grid_cells.end()); //attepmt to erase biggest cell from the vector
      }

      std::ofstream of(utils::to_string(index) + ".out");
      std::vector<core::data::structural::PdbAtom_SP> sink;
      ligpro.grid()->get_neighbors(index, sink);
      for (core::index4 a = 0; a < sink.size(); a++) {//iterating over Atoms in sink
        of << sink[a]->id() << "\n"; //writing to file
      }
      of.close();
    }
  }
}
```

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:

```
#include <ctime>
#include <iostream>
#include <sstream>

#include <utils/Logger.hh>
#include <utils/io_utils.hh>

#include <utils/options/Option.hh>
#include <utils/options/OptionParser.hh>
#include <utils/options/input_options.hh>

#include <core/data/basic/Vec3.hh>
#include <core/data/io/Pdb.hh>
```

(continues on next page)

(continued from previous page)

```
#include <core/alignment/scoring/LocalStructure7.hh>
#include <core/alignment/scoring/LocalStructure5.hh>
#include <core/alignment/scoring/LocalStructureMatch.hh>
#include <utils/exit.hh>

utils::Logger l("ap_LocalStructureMatch");

std::string program_info = R"(

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

)";

/** @brief Finds contiguous structural segments that are similar between two structures
 *
 * CATEGORIES: core/alignment/scoring/LocalStructureMatch;
 * KEYWORDS:   PDB input; structure match
 * GROUP:      Alignments
 */
int main(const int argc, const char *argv[]) {

  if(argc < 4) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
  int match_size = atoi(argv[1]);

  double max_print_distance = (argc==5) ? atof(argv[4]) : 100000.0;
  using namespace core::alignment::scoring; // --- for LocalStructure7, LocalStructure5 and LocalStructureMatch
  using namespace core::data::basic;  // --- for Coordinates_SP and Vec3
  Coordinates_SP xyz_q = std::make_shared<std::vector<Vec3>>();
  Coordinates_SP xyz_t = std::make_shared<std::vector<Vec3>>();
  core::data::io::Pdb::read_coordinates(argv[2], *xyz_q, true, core::data::io::is_ca);
  if (match_size == 7) {
    LocalStructure7 local_query(xyz_q);
    core::data::io::Pdb::read_coordinates(argv[3], *xyz_t, true, core::data::io::is_ca);
    LocalStructure7 local_tmplt(xyz_t);
    LocalStructureMatch<LocalStructure7, 8> lm(local_query, local_tmplt);
    lm.print(std::cout, max_print_distance);
```

(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_
      ↪ca);
65      LocalStructure5 local_tmplt(xyz_t);
66      LocalStructureMatch<LocalStructure5, 8> lm(local_query, local_tmplt);
67      lm.print(std::cout, max_print_distance);
68    }
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:

```
#include <iostream>
#include <vector>
#include <string>

#include <core/data/basic/Vec3I.hh>
#include <core/BioShellVersion.hh>

#include <utils/string_utils.hh>
#include <utils/options/Option.hh>
#include <utils/options/OptionParser.hh>
#include <utils/options/output_options.hh>
#include <utils/options/sampling_options.hh>

#include <simulations/systems/CartesianChains.hh>
#include <simulations/systems/BuildFluidSystem.hh>
#include <simulations/movers/RotateRigidMolecule.hh>
#include <simulations/movers/TranslateMolecule.hh>
#include <simulations/movers/MoversSetSweep.hh>
#include <simulations/forcefields/mm/Water3PointEnergy.hh>
#include <simulations/forcefields/mm/WaterModelParameters.hh>

#include <simulations/sampling/IsothermalMC.hh>
#include <simulations/observers/ObserveEvaluators.hh>
#include <simulations/observers/cartesian/PdbObserver.hh>
#include <simulations/observers/ObserveMoversAcceptance.hh>
```

(continues on next page)

(continued from previous page)

```
#include <simulations/observers/TriggerEveryN.hh>
#include <simulations/observers/AdjustMoversAcceptance.hh>
#include <simulations/observers/cartesian/ExplicitPdbFormatter.hh>
#include <simulations/evaluators/CallEvaluator.hh>
#include <simulations/systems/SimpleAtomTyping.hh>


using namespace core::data::basic;

utils::Logger logs("ap_MC_water");

std::string program_info = R"(

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

)";

/** @brief Isothermal Monte Carlo simulation of water.
 *
 */
int main(const int argc,const char* argv[]) {

  using core::data::basic::Vec3Cubic;
  using namespace simulations::systems;
  using namespace simulations::movers; // for MoversSet
  using namespace simulations::observers::cartesian; // for all observers
  using simulations::forcefields::WaterModelParameters;

  logs << utils::LogLevel::INFO << "BioShell version:\n" << core::BioShellVersion().to_string() << "\n";

  core::index4 n_outer_cycles = 1000;
  core::index4 n_inner_cycles = 10;
  double temperature = 298;  // in Kelvins
  core::index4 n_molecules = 216; // 216
  core::calc::statistics::Random::seed(1234);

  double water_density = 0.99823;
  double water_mass = 18.01528;
  core::data::structural::Structure_SP water_structure = nullptr;

  if (argc < 5) std::cerr << program_info;
  else {
    if (utils::is_integer(argv[1])) n_molecules = atoi(argv[1]);
    else { // --- read an input file if given
      core::data::io::Pdb reader(argv[1]);
      water_structure = reader.create_structure(0);
      n_molecules = water_structure->count_residues();
    }
    temperature = atof(argv[2]);
    n_inner_cycles = atoi(argv[3]);
    n_outer_cycles = atoi(argv[4]);
```

(continues on next page)

(continued from previous page)

```
  }
  double water_volume =  n_molecules * 10 * water_mass/6.02214;
  double box_len = pow(water_volume / water_density, 0.33333333333333);

  // --- Initialize periodic boundary conditions
  core::data::basic::Vec3I::set_box_len(box_len);
  logs << utils::LogLevel::INFO << "box width for " << int(n_molecules) << " molecules : " << box_len << "\n";

  WaterModelParameters::load_models();
  WaterModelParameters tip3p = WaterModelParameters::get_model("TIP3P");

  // --- Create water structure if not loaded from PDB
  if (water_structure == nullptr) {
    core::data::structural::Residue_SP hoh = tip3p.create_residue();
    core::data::structural::Residue_SP water_molecule = std::make_shared<core::data::structural::Residue>(1,"HOH");
    PointGridGenerator_SP grid = std::make_shared<SimpleCubicGrid>(box_len, n_molecules);
    water_structure = BuildFluidSystem::build_structure(*hoh, grid);
  }

//  SimpleAtomTyping tip3p_typing({"HOH"}, {"O", "H"}, {" O  ", " H  "});
  // --- Create the system to be sampled
  std::vector<std::string> res_types {"HOH"};
  std::vector<std::string> atom_types {"O", "H"};
  std::vector<std::string> pdb_types {" O  ", " H  "};
  AtomTypingInterface_SP tip3p_typing = std::make_shared<SimpleAtomTyping>(res_types, atom_types, pdb_types);
  CartesianChains system(tip3p_typing, *water_structure);
  CartesianChains backup(system);

  // --- Create energy function - TIP3P potential
  simulations::forcefields::mm::Water3PointEnergy en(tip3p);

  // --- Create a mover, which is a random perturbation of an atom in this case, and place it in a movers' set
  std::shared_ptr<RotateRigidMolecule> rot = std::make_shared<RotateRigidMolecule>(system, backup, en, 0);
  std::shared_ptr<TranslateMolecule> trs = std::make_shared<TranslateMolecule>(system, backup, en);
  MoversSet_SP movers = std::make_shared<simulations::movers::MoversSetSweep>();
  movers->add_mover(rot, n_molecules);
  movers->add_mover(trs, n_molecules);

  // --- 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>("");
  // ---------- Create an observer which calls energy calculation and prints it to a file
//  std::shared_ptr<simulations::observers::ObserveEvaluators> obs = std::make_shared<simulations::observers::ObserveEvaluators>("energy.dat");
  std::function<double(void)> recent_energy = [&en, &system]() { return en.energy(system) / (system.count_residues() * 1000); };
```

(continues on next page)

(continued from previous page)

```
  obs->add_evaluator(
    std::make_shared<simulations::evaluators::CallEvaluator<std::function
↪<double(void)>>>(recent_energy, "energy", 8));

  std::shared_ptr<simulations::observers::AdjustMoversAcceptance> observe_moves
    = std::make_shared<simulations::observers::AdjustMoversAcceptance>(*movers,
↪"movers.dat", 0.4);
  observe_moves->observe_header();

  std::shared_ptr<AbstractPdbFormatter> fmt = std::make_shared<ExplicitPdbFormatter>
↪(*water_structure);
  auto observe_trajectory = std::make_shared
↪<simulations::observers::cartesian::PdbObserver>(system, fmt, "water_tra.pdb");
//  observe_trajectory->trigger(std::make_shared
↪<simulations::observers::TriggerEveryN>(10));
  mc.outer_cycle_observer(observe_trajectory); // --- commented out to save disk space
  mc.outer_cycle_observer(observe_moves);
  mc.outer_cycle_observer(obs);
  mc.cycles(n_inner_cycles,n_outer_cycles,1);

  mc.run();

  simulations::observers::cartesian::PdbObserver final(system, fmt, "final.pdb");
  final.observe();
//  logs << utils::LogLevel::INFO << "Final energy " << lj_energy.calculate() << "\n";
}
```

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:

```
#include <iostream>

#include <core/alignment/MSAColumnConservation.hh>
#include <core/data/io/clustalw_io.hh>
#include <utils/exit.hh>
#include <utils/io_utils.hh>
#include <core/data/io/fasta_io.hh>

std::string program_info = R"(

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

(continues on next page)

(continued from previous page)

```
  else
    std::cout << "#pos   gaps   Shanon Relative Variation SumOfPairs JensenShannon\n";

  for (size_t ipos = 0; ipos < msa[0]->length(); ++ipos)
    std::cout << utils::string_format("%4d %c %7.3f %7.3f %7.3f %7.3f %7.3f %7.3f\n", ipos, seq_str[ipos],
      consrv.evaluate(core::alignment::ColumnConservationScores::GapPercent, ipos),
      consrv.evaluate(core::alignment::ColumnConservationScores::ShannonEntropy, ipos),
      consrv.evaluate(core::alignment::ColumnConservationScores::RelativeEntropy, ipos),
      consrv.evaluate(core::alignment::ColumnConservationScores::Variation, ipos),
      consrv.evaluate(core::alignment::ColumnConservationScores::SumOfPairs, ipos),
      consrv.evaluate(core::alignment::ColumnConservationScores::JensenShannonDivergence, ipos));
}
```

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

Keywords:

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

Categories:

- core/alignment/NWAligner

Input files:

- ferrodoxins.fasta
- input2.fasta

Output files:

- stdout.out

Program source:

```
#include <iostream>
#include <chrono>
#include <algorithm>

#include <core/data/io/fasta_io.hh>

#include <core/alignment/NWAligner.hh>
#include <core/alignment/scoring/SimilarityMatrix.hh>
#include <core/alignment/scoring/SimilarityMatrixScore.hh>
#include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>
#include <core/alignment/on_alignment_computations.hh>
```

(continues on next page)

(continued from previous page)

```
#include <core/alignment/PairwiseSequenceAlignment.hh>
#include <core/data/io/alignment_io.hh>
#include <core/data/sequence/Sequence.hh>

#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Calculate all pairwise sequence alignments between sequences read from two FASTA files : query and database
 *
 * CATEGORIES: core/alignment/NWAligner
 * KEYWORDS:   FASTA input; Needleman-Wunsch; sequence alignment
 * GROUP:      Alignments
 */
int main(const int argc, const char* argv[]) {

  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  using namespace core::data::io;
  using namespace core::data::sequence;
  using namespace core::alignment::scoring;

  // --- the query sequence
  std::vector<std::shared_ptr<Sequence>> query_sequences;
  read_fasta_file(argv[1], query_sequences);
  // --- container for the sequence database
  std::vector<std::shared_ptr<Sequence>> db_sequences;
  read_fasta_file(argv[2], db_sequences);

  // --- find longest sequence to initialize aligner object large enough
  unsigned max_len = 0;
  std::for_each(query_sequences.begin(), query_sequences.end(),
    [&max_len](const Sequence_SP s) { max_len = std::max(max_len, unsigned(s->length())); });
```

(continues on next page)

(continued from previous page)

```
  std::for_each(db_sequences.begin(), db_sequences.end(),
    [&max_len](const Sequence_SP s) { max_len = std::max(max_len, unsigned(s->length())); });

  // --- create aligner object
  core::alignment::NWAligner<short, SimilarityMatrixScore<short>> aligner(max_len);
  // --- read similarity matrix from a file (e.g. BLOSUM62)
  NcbiSimilarityMatrix_SP sim_m = NcbiSimilarityMatrixFactory::get().get_matrix((argc > 3) ? argv[3] : "BLOSUM62");
  // --- go through all db sequences and align them with the given query
  auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
  for (size_t i = 0; i < query_sequences.size(); ++i) {
    for (size_t j = 0; j < db_sequences.size(); ++j) {
      // --- Here we create a sequence similarity object that will score a match
      // --- between individual positions from the two sequences being aligned
      SimilarityMatrixScore<short> score(query_sequences[i]->sequence, db_sequences[j]->sequence, *sim_m);

      // ---------- calculate local alignment
      aligner.align(-14, -2, score);

      // ---------- Convert the abstract alignment to a pairwise sequence alignment object
      const core::alignment::PairwiseAlignment_SP  ali = aligner.backtrace();
      core::alignment::PairwiseSequenceAlignment seq_ali(ali, query_sequences[i], db_sequences[j]);

      // ---------- check basics statistics of the alignment
      core::index2 identical = core::alignment::sum_identical(seq_ali);
      core::index2 n_aligned = seq_ali.alignment->n_aligned();
      std::cout <<utils::string_format("# %s %s id: %6.3f  cov: %6.3f\n",
          utils::split(query_sequences[i]->header())[0].c_str(), utils::split(db_sequences[j]->header())[0].c_str(),
          identical / double(query_sequences[i]->length()), n_aligned / double(query_sequences[i]->length()) );
      // ---------- Print the alignment in Edinburgh format
      core::data::io::write_edinburgh(seq_ali, std::cout, 80);

      // --- Alternatively one can find only the score of the alignment;
      // --- just the score - this is faster than aligning and keeping backtracking info
      short result = aligner.align_for_score(-10, -1, score);
    }
  }
  auto end = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> time_span = std::chrono::duration_cast<std::chrono::duration<double>>(end - start);
  std::cerr << db_sequences.size() * query_sequences.size() << " global alignment scores computed within "
            << time_span.count() << " [s]\n";
}
```

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:

```
#include <iostream>

#include <core/index.hh>
#include <core/calc/statistics/Random.hh>
#include <core/calc/statistics/OnlineStatistics.hh>

std::string program_info = R"(

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.

```

(continues on next page)

(continued from previous page)

```
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
)";

/** @brief Reads a file with real values and calculates simple statistics: min, mean, stdev, max.
 * If no input file is provided, the program calculates the statistics from a random sample
 *
 * CATEGORIES: core::calc::statistics::OnlineStatistics; core::calc::statistics::Random
 * KEYWORDS:   random numbers; statistics
 * GROUP: Statistics;
 */
int main(const int argc, const char *argv[]) {

  core::calc::statistics::OnlineStatistics stats;
  if(argc < 2) {
    // --- complain about missing program parameter
    std::cerr << program_info;
    // ---------- Use the random engine if no data is provided
    core::calc::statistics::Random r = core::calc::statistics::Random::get();
    r.seed(12345);  // --- seed the generator for repeatable results
    core::calc::statistics::UniformRealRandomDistribution<double> uniform_random;
    for (core::index4 n = 0; n < 100000; ++n) stats(uniform_random(r));
  } else {
    std::ifstream in(argv[1]);
    double r;
    while(in) {
      in >> r;
      stats(r);
    }
  }

  std::cout << "#cnt   min        avg      sdev  skewness  kurtosis    max  bimodalitycoefficient\n";
  std::cout << utils::string_format("%d %f %f %f %f %f %f %f\n",stats.cnt(),stats.min(),stats.avg(),
    sqrt(stats.var()),stats.skewness(),stats.kurtosis(),stats.max(), stats.bimodality_coefficient());
}
```

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-model1.pdb 2gb1-model2.pdb
```

Keywords:

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

Categories:

- core::protocols::PairwiseCrmsd

Input files:

- 2gb1-model3.pdb
- 2gb1-model1.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)

```
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-model1.pdb 2gb1-model2.pdb

)";

/** @brief Calculates crmsd value for a set of protein structures (at least two)
 *
 * CATEGORIES: core::protocols::PairwiseCrmsd
 * KEYWORDS:   PDB input; crmsd; structure selectors
 * GROUP: Structure calculations;
 */
int main(const int argc, const char *argv[]) {

  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  using core::data::basic::Vec3;
  using namespace core::data::structural::selectors; // --- for all AtomSelector types
  using namespace core::data::io;
  using namespace core::protocols;

  std::vector<core::data::structural::Structure_SP> structures;
  std::vector<std::string> tags;
  for (int i = 1; i < argc; ++i) {
    core::data::io::Pdb reader(argv[i],all_true(is_not_alternative,is_not_water),keep_all, false); // --- note we read all atoms but skip alternate locators and waters
    structures.push_back(reader.create_structure(0));
    tags.push_back(structures.back()->code());
  }

  // ---------- crmsd on C-alpha : this is the
  std::cout <<"# crmsd on alpha carbons:\n";
  std::shared_ptr<AtomSelector> is_CA = std::make_shared<IsCA>();
  PairwiseCrmsd rmsd_ca(structures, is_CA, tags);
  rmsd_ca.crmsd_cutoff(20.0); // crmsd cutoff large enough to get some output
  rmsd_ca.calculate();

  // ---------- crmsd on backbone
  std::cout <<"# crmsd on heavy backbone atoms:\n";
  std::shared_ptr<AtomSelector> is_bb = std::make_shared<IsBB>();
  std::shared_ptr<AtomSelector> not_h = std::make_shared<NotHydrogen>();
  std::shared_ptr<LogicalANDSelector> heavy_bb = std::make_shared<LogicalANDSelector>();
  heavy_bb->add_selector(is_bb);
  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)

```

std::string program_info = R"(

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.


)";

/** @brief Uses PairwiseSequenceIdentityProtocol protocol to calculate all pairwise sequence identity values for a set of sequences
 *
 * CATEGORIES: core/protocols/PairwiseSequenceIdentityProtocol.hh
 * KEYWORDS:   FASTA input; sequence alignment
 * GROUP:      Alignments
 */
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::sequence;
  using namespace core::protocols;
  using namespace core::alignment;

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

(continues on next page)

(continued from previous page)

```
//    if_use_fasta = true;
//    --my_argc;
//  }
  core::index2 n_threads = (my_argc > 2) ? atoi(argv[2]) : 4;
  float cutoff = (my_argc > 3) ? atof(argv[3]) : 0.25;

  logs << utils::LogLevel::INFO << "number of threads used : " << n_threads << "\n";
  logs << utils::LogLevel::INFO << "seq. similarity cutoff : " << cutoff << "\n";

  core::protocols::PairwiseSequenceIdentityProtocol protocol;
  protocol.printed_seqname_length(20).gap_open(-10).gap_extend(-1).substitution_
↪matrix("BLOSUM62").
    keep_alignments(false).alignment_method(AlignmentType::SEMIGLOBAL_ALIGNMENT).n_
↪threads(n_threads);
  protocol.if_use_fasta_filter(if_use_fasta).seq_identity_cutoff(cutoff).batch_
↪size(10000);
  protocol.printed_seqname_length(10);

  if (my_argc == 5) {
    protocol.select_query(atoi(argv[4]));
    logs << utils::LogLevel::INFO << "Using sequence at index " << atoi(argv[4]) << "␣
↪as a query\n";
  }
  if (my_argc == 6) {
    for (core::index4 i = atoi(argv[4]); i <= atoi(argv[5]); ++i) protocol.add_
↪query(i);
    logs << utils::LogLevel::INFO << "Using " << atoi(argv[5]) - atoi(argv[4]) << "␣
↪query sequences\n";
  }

  std::vector<Sequence_SP> input_sequences;
  core::data::io::read_fasta_file(argv[1], input_sequences);
  for(Sequence_SP si: input_sequences) protocol.add_input_sequence(si);

  auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
  protocol.run();
  auto end = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> time_span = std::chrono::duration_cast
↪<std::chrono::duration<double>>(end - start);
  logs << utils::LogLevel::INFO << (size_t ) protocol.n_jobs_completed()
            << " global alignment sequence identities calculated within " << time_
↪span.count() << " [s]\n";

  protocol.print_header(std::cout);
  protocol.print_sequence_identity(std::cout);
}
```

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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/calc/structural/ProteinArchitecture.hh>
#include <utils/exit.hh>
#include <utils/LogManager.hh>

using namespace core::data::structural;
using namespace core::data::io;
```

(continues on next page)

(continued from previous page)

```
using namespace core::data::basic;

std::string program_info = R"(

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

)";

/** @brief Calculates a map of backbone hydrogen bonds.
 *
 * CATEGORIES: core/calc/structural/ProteinArchitecture;
 * KEYWORDS:   PDB input; Hydrogen bonds; Protein structure features
 * GROUP: Structure calculations;
 */
int main(const int argc, const char* argv[]) {

  using namespace core::calc::structural;

  utils::LogManager::INFO(); // --- Turn it to FINE to see a lot more of messages, e.g about missed h-bonds

  if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
  core::data::io::Pdb reader(argv[1], core::data::io::all_true(is_not_alternative, is_not_water),
      core::data::io::keep_all, true); // --- Read in a PDB file
  Structure_SP strctr = reader.create_structure(0); // --- create a Structure object from the first model

  bool if_dssp = (argc > 2) && (strcmp(argv[2], "DSSP") == 0);
  core::calc::structural::ProteinArchitecture pa(*strctr, if_dssp);

  std::cout <<"# ---------- Secondary structure elements ----------\n";
  for (const auto sse : pa.sse_vector())
    std::cout << *sse << "\n";

  std::cout <<"# ---------- Beta strand connectivity ----------\n";
  auto sse_graph = pa.create_strand_graph();
  sse_graph->print_adjacency_matrix(std::cerr);
```

(continues on next page)

(continued from previous page)

```
  for(auto e_it = sse_graph->cbegin_strand();e_it!=sse_graph->cend_strand();++e_it) {
    std::cout << (*e_it)->info()<<" paired with:\n";
    for(auto partner_it = sse_graph->cbegin_strand(*e_it); partner_it != sse_graph->
↪cend_strand(*e_it); ++partner_it) {
      auto pairing_sp = sse_graph->get_strand_pairing(*e_it, *partner_it);
      std::cout << "\t" << (*partner_it)->name() << " " << Strand::strand_type_
↪name(pairing_sp->pairing_type)
                << " by " << pairing_sp->hydrogen_bonds().size() << " hbonds\n";
    }
  }
}
```

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:

```
#include <iostream>
#include <string>
#include <stdexcept>
#include <stdlib.h>
#include <fstream>
#include <vector>

#include <simulations/evaluators/CallEvaluator.hh>
#include <simulations/forcefields/CalculateEnergyBase.hh>
#include <simulations/evaluators/Evaluator.hh>
#include <simulations/forcefields/TotalEnergy_OBSOLETE.hh>

#include <simulations/observers/ObserveEvaluators.hh>
#include <simulations/observers/ObserveMoversAcceptance.hh>
#include <simulations/observers/TriggerEveryN.hh>
#include <simulations/observers/ObserveReplicaFlow.hh>
#include <simulations/observers/ObserveWLSampling.hh>

#include <simulations/sampling/IsothermalMC.hh>
#include <simulations/sampling/ReplicaExchangeMC.hh>
#include <simulations/systems/ising/RubikCube.hh>

#include <utils/options/sampling_options.hh>
#include <utils/options/sampling_from_cmdline.hh>
#include <simulations/sampling/WangLandauSampler.hh>


using namespace simulations;
using namespace simulations::systems::ising;

utils::Logger logs("ap_Rubik_simulation");

std::string program_info = R"(

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

(continues on next page)

(continued from previous page)

```
)";

/** @brief Turns energy of a system into an energy bin index (integer)
 * @param energy - system's energy
 * @return integer assigned to a bin; may be negative
 */
inline int bfe(double energy) { return (int) energy; }

std::shared_ptr<simulations::sampling::WangLandauSampler> prepare_wl_simulation(const simulations::SimulationSettings &settings) {

  using namespace utils::options;
  int system_size = settings.get<int>("cube_size");
  Rubik_SP system = std::make_shared<Rubik>(system_size);
  logs << "Minimum energy: " << system->calculate()<<"\n";
  system->scramble();    // Set the cube to a random conformation
  logs << "starting energy: " << system->calculate()<<"\n";

  // ---------- Movers definition ----------
  simulations::movers::MoversSet_SP movers = std::make_shared<simulations::movers::MoversSetSweep>();
  movers->add_mover(std::static_pointer_cast<simulations::movers::Mover>(system), system_size * system_size);

  // ---------- Create the sampler ----------
  std::shared_ptr<simulations::sampling::WangLandauSampler> sampler = std::make_shared<simulations::sampling::WangLandauSampler>(
      movers, system->calculate(), bfe, system_size * system_size * 6);
  sampler->reset(settings);

  simulations::forcefields::CalculateEnergyBase_SP energies;

  simulations::observers::ObserveEvaluators_SP observations
      = std::make_shared<simulations::observers::ObserveEvaluators>(utils::string_format("energy-wl.dat"));
  observations->add_evaluator(system);
  sampler->outer_cycle_observer(observations);
  sampler->outer_cycle_observer(std::make_shared<simulations::observers::ObserveWLSampling>(*sampler, "wl.dat"));

  simulations::observers::ObserveMoversAcceptance_SP obs_ms
      = std::make_shared<simulations::observers::ObserveMoversAcceptance>(*movers, utils::string_format("movers-wl.dat"));
  obs_ms->observe_header();
  sampler->outer_cycle_observer(obs_ms);

  return sampler;
}

std::shared_ptr<simulations::sampling::ReplicaExchangeMC> prepare_replica_simulation(const simulations::SimulationSettings& settings) {

  using namespace utils::options;
  std::vector<Rubik_SP> systems;
  std::vector<simulations::sampling::IsothermalMC_SP> replica_samplers;
```

(continues on next page)

(continued from previous page)

```
  std::vector<simulations::forcefields::CalculateEnergyBase_SP> energies;
  std::vector<double> temperatures;
  utils::split(settings.get<std::string>(replicas),temperatures, ',');
  core::index4 n_outer_cycles = settings.get<core::index4>(mc_outer_cycles);
  core::index4 n_inner_cycles = settings.get<core::index4>(mc_inner_cycles);
  core::index4 n_exchanges = settings.get<core::index4>(replica_exchanges);

  int system_size =  settings.get<int>("cube_size");
  for (core::index2 irepl = 0; irepl < temperatures.size(); ++irepl) {

    // ---------- Create the systems to be sampled ----------
    Rubik_SP system = std::make_shared<Rubik>(system_size);
    system->scramble();    // Set the cube to a random conformation
    systems.push_back(system);
    energies.push_back(system);

    // ---------- Movers definition ----------
    simulations::movers::MoversSet_SP movers = std::make_shared
↪<simulations::movers::MoversSetSweep>();
    movers->add_mover(std::static_pointer_cast<simulations::movers::Mover>(system),␣
↪system_size * system_size);

    // ---------- Create the sampler ----------
    auto sampler = std::make_shared<simulations::sampling::IsothermalMC>(movers,␣
↪temperatures[irepl]);
    replica_samplers.push_back(sampler);
    sampler->cycles(n_inner_cycles, n_outer_cycles);

    simulations::observers::ObserveEvaluators_SP observations
        = std::make_shared<simulations::observers::ObserveEvaluators>(utils::string_
↪format("energy-%.3f.dat", temperatures[irepl]));
    observations->add_evaluator(system);
    sampler->outer_cycle_observer(observations);
    simulations::observers::ObserveMoversAcceptance_SP obs_ms
        = std::make_shared<simulations::observers::ObserveMoversAcceptance>(*movers,
                                                                            ␣
↪utils::string_format("movers-%.3f.dat", temperatures[irepl]));
    obs_ms->observe_header();
    sampler->outer_cycle_observer(obs_ms);
  }
  auto remc = std::make_shared<simulations::sampling::ReplicaExchangeMC>(replica_
↪samplers, energies, true);
  auto remc_flow = std::make_shared<simulations::observers::ObserveReplicaFlow>(*remc,
↪ "replica_flow.dat");
  remc->exchange_observer(remc_flow);
  remc->replica_exchanges(n_exchanges);

  return remc;
}

/** @brief The program runs a Replica Exchange Monte Carlo simulation of a Rubik's␣
↪cube system.
 *
 * This example shows how to simulate a system using BioShell library
 *
 * CATEGORIES: simulations/sampling/ReplicaExchangeMC;
 * KEYWORDS:   Monte Carlo; sampling;  observer; simulation
```

(continues on next page)

(continued from previous page)

```
 * IMG_ALT: Example results from a Rubik's cube simulations
 */
int main(const int argc, const char *argv[]) {

  using namespace utils::options; // --- All the options are in this namespace
  static Option cube_size("-c", "-cube_size", "size of the Rubik's cube");
  static Option sampler("-s", "-sampler", "MC sampler: 'remc' or 'wl'");

  utils::options::OptionParser &cmd = utils::options::OptionParser::get();
  cmd.register_option(utils::options::help, verbose, rnd_seed, cube_size(3), sampler);
  cmd.register_option(mc_outer_cycles(10000), mc_inner_cycles(10), mc_cycle_factor(1),
↪ replica_exchanges(10));
  cmd.register_option(begin_temperature(2.0), end_temperature(0.5), temp_steps(0.1),
                      replicas("2,1.75,1.5,1.25,1.0,0.8,0.7,0.6,0.5"));

  if (!cmd.parse_cmdline(argc, argv)) return 1;

  if (rnd_seed.was_used()) {
    auto rnd = option_value<core::calc::statistics::Random::result_type>(rnd_seed);
    core::calc::statistics::Random::seed(rnd);
    logs << utils::LogLevel::SEVERE << "Pseudorandom start: " << rnd << "\n";
  } else {
    core::calc::statistics::Random::get().seed(12345);  // --- seed the generator for␣
↪repeatable results
    logs << utils::LogLevel::SEVERE << "Pseudorandom start with seed: 12345\n";
//    core::calc::statistics::Random::seed(time(0));
//    logs << utils::LogLevel::SEVERE << "Pseudorandom start with time(0) seed: \n";
  }

  simulations::SimulationSettings settings;
  settings.insert_or_assign(cmd, true);

  if ((option_value<std::string>(sampler) == "wl") || (option_value<std::string>
↪(sampler) == "WL")) {
    auto wl = prepare_wl_simulation(settings);
    wl->run();
  } else {
    auto remc = prepare_replica_simulation(settings);
    remc->run();
  }

}
```

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:

```
#include <iostream>
#include <chrono>
#include <algorithm>

#include <core/data/io/fasta_io.hh>

#include <core/data/sequence/Sequence.hh>
#include <core/alignment/SWAligner.hh>
#include <core/alignment/scoring/SimilarityMatrix.hh>
#include <core/alignment/scoring/SimilarityMatrixScore.hh>
#include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>
#include <core/alignment/PairwiseSequenceAlignment.hh>
#include <core/data/io/alignment_io.hh>
```

(continues on next page)

(continued from previous page)

```

#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Calculate all pairwise sequence alignments between sequences read from two FASTA files : query and database
 *
 * CATEGORIES: core/alignment/SWAligner
 * KEYWORDS:   FASTA input; sequence alignment
 * GROUP:      Alignments
 */
int main(const int argc, const char* argv[]) {

  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  using namespace core::data::io;
  using namespace core::data::sequence;
  using namespace core::alignment::scoring;

  // --- the query sequence
  std::vector<std::shared_ptr<Sequence>> query_sequences;
  read_fasta_file(argv[1], query_sequences);
  // --- container for the sequence database
  std::vector<std::shared_ptr<Sequence>> db_sequences;
  read_fasta_file(argv[2], db_sequences);

  // --- find longest sequence to initialize aligner object large enough
  unsigned max_len = 0;
  std::for_each(query_sequences.begin(), query_sequences.end(),
    [&max_len](const Sequence_SP s) { max_len = std::max(max_len, unsigned(s->length())); });
  std::for_each(db_sequences.begin(), db_sequences.end(),
    [&max_len](const Sequence_SP s) { max_len = std::max(max_len, unsigned(s->length())); });

  // ---------- Create aligner object
```

(continues on next page)

(continued from previous page)

```
  core::alignment::SWAligner<short, SimilarityMatrixScore<short>> aligner(max_len);

  // ---------- read similarity matrix from a file (e.g. BLOSUM62)
  NcbiSimilarityMatrix_SP sim_m = NcbiSimilarityMatrixFactory::get().get_matrix((argc > 3) ? argv[3] : "BLOSUM62");

  // ---------- Go through all db sequences and align them with the given query
  auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
  for (size_t i = 0; i < query_sequences.size(); ++i) {
    for (size_t j = 0; j < db_sequences.size(); ++j) {

      // ---------- Here we create a sequence similarity object that will score a match
      // ---------- between individual positions from the two sequences being aligned
      SimilarityMatrixScore<short> score(query_sequences[i]->sequence, db_sequences[j]->sequence, *sim_m);

      // ---------- calculate local alignment
      aligner.align(-10, -1, score);

      // ---------- Convert the abstract alignment to a pairwise sequence alignment object
      const core::alignment::PairwiseAlignment_SP  ali = aligner.backtrace();
      core::alignment::PairwiseSequenceAlignment seq_ali(ali, query_sequences[i], db_sequences[j]);

      // ---------- Print the alignment in Edinburgh format
      core::data::io::write_edinburgh(seq_ali, std::cout, 80);
    }
  }
  auto end = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> time_span = std::chrono::duration_cast<std::chrono::duration<double>>(end - start);
  std::cerr << db_sequences.size() * query_sequences.size() << " local alignment scores computed within "
            << time_span.count() << " [s]\n";
}
```

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 ain 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:

```
#include <iostream>

#include <core/data/io/hssp_io.hh>
#include <core/data/io/fasta_io.hh>
#include <core/data/io/clustalw_io.hh>
#include <core/data/sequence/SequenceProfile.hh>
#include <core/protocols/SequenceWeightingProtocol.hh>
#include <utils/exit.hh>
```

(continues on next page)

(continued from previous page)

```
#include <utils/io_utils.hh>

std::string program_info = R"(

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 ain 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

)";

/** @brief Reads a MSA in ClustalW format  and prints a sequence profile
 *
 * CATEGORIES: core/data/sequence/SequenceProfile; core/protocols/SequenceWeightingProtocol
 * KEYWORDS:   sequence profile; Clustal input; MSA
 * GROUP: Sequence calculations;
 */
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;
  using namespace core::data::sequence;

  // ---------- Load all sequences into a vector
  std::vector<Sequence_SP> msa;   // --- Sequence_SP is just a shorter name for std::shared_ptr<Sequence>
  auto root_extn = utils::root_extension(argv[1]);
  if ((root_extn.second == "aln") || (root_extn.second == "clustalw")) {
    core::data::io::read_clustalw_file(argv[1], msa, true);
  } else if (root_extn.second == "hssp") {
    core::data::io::read_hssp_file(argv[1], msa, true, true);
  } else
    core::data::io::read_fasta_file(argv[1], msa);

  std::vector<double> seq_weights{1,1.0}; // --- just one weight of value 1.0
  // ---------- Set up and run sequence weighting protocol if needed
  if ((argc == 3) && (strcmp(argv[2], "-w") == 0)) {
    core::protocols::HenikoffSequenceWeights protocol;
    protocol.n_threads(4).add_input_sequences(msa);
    protocol.run();
    seq_weights.clear();
```

(continues on next page)

(continued from previous page)

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

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:

```
#include <utils/exit.hh>
#include <core/data/basic/Array2D.hh>
#include <core/data/sequence/Sequence.hh>
#include <core/data/io/fasta_io.hh>
#include <core/data/io/clustalw_io.hh>
#include <core/protocols/SequenceWeightingProtocol.hh>
#include <utils/io_utils.hh>

```

(continues on next page)

(continued from previous page)

```
std::string program_info = R"(

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

)";

/** @brief Shows how to use SequenceWeightingProtocol class
 *
 * CATEGORIES: core/protocols/SequenceWeightingProtocol
 * KEYWORDS:   FASTA input; sequence alignment; sequence identity; sequence weighting
 * GROUP: Sequence calculations;
 */
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::sequence;
  using namespace core::protocols;

  bool if_align = true;
  std::vector<Sequence_SP> input_sequences;
  auto root_extn = utils::root_extension(argv[1]);
  if ((root_extn.second == "aln") || (root_extn.second == "clustalw")) {
    core::data::io::read_clustalw_file(argv[1], input_sequences);
    if_align = false;
  } else
    core::data::io::read_fasta_file(argv[1], input_sequences);

  core::protocols::HenikoffSequenceWeights protocol;
  protocol.n_threads(1);
  protocol.add_input_sequences(input_sequences);
  auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
  protocol.run();
  auto end = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> time_span = std::chrono::duration_cast<std::chrono::duration<double>>(end - start);
  std::cerr << input_sequences.size() * (input_sequences.size() - 1) / 2.0
            << " sequence similarities calculated within " << time_span.count() << " [s]\n";
```

(continues on next page)

(continued from previous page)

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

```
#include <iostream>
#include <cmath>

#include <core/index.hh>
#include <core/calc/statistics/Random.hh>
#include <core/calc/statistics/WeightedOnlineStatistics.hh>

std::string program_info = R"(

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:
```

(continues on next page)

(continued from previous page)

```
    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
)";

/** @brief Reads a file with two columns: real values and their weights, and calculates their mean and stdev.
 *
 * If no input file is provided, the program calculates the statistics from a random sample
 *
 * CATEGORIES: core/calc/statistics/ap_WeightedOnlineStatistics; core/calc/statistics/Random
 * KEYWORDS:   statistics
 * GROUP: Statistics;
 */
int main(const int argc, const char *argv[]) {

  core::calc::statistics::WeightedOnlineStatistics stats;
  if (argc < 2) {
    // --- complain about missing program parameter
    std::cerr << program_info;
    // ---------- Use the random engine if no data is provided
    core::calc::statistics::Random r = core::calc::statistics::Random::get();
    r.seed(12345);  // --- seed the generator for repeatable results
    std::normal_distribution<double> normal_random;
    for (core::index4 n = 0; n < 10000; ++n) {
      double x = normal_random(r);
      if (x <= 2.0) stats(x, 0.1); // --- insert the random point with an arbitrary weight = 0.1
      else for (int i = 0; i < 10; ++i) stats(x, 0.01); // in the tail insert points ten times with weight 1/10
    }
  } else {
    std::ifstream in(argv[1]);
    double x, w;
    while (in) {
      in >> x >> w;
      stats(x, w);
    }
  }

  std::cout << "#count sum_wghts   avg      sdev\n";
  std::cout << utils::string_format("%d  %lf %f %f \n", stats.cnt(), double(stats.sum_of_weights()), stats.avg(), sqrt(stats.var()));
}
```

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:

```
#include <iostream>
#include <chrono>
#include <algorithm>

#include <core/data/io/fasta_io.hh>

#include <core/alignment/NWAlignerAnyGap.hh>
#include <core/alignment/PairwiseSequenceAlignment.hh>
#include <core/alignment/scoring/Picasso3.hh>
#include <core/alignment/scoring/FrequencyScaledGapPenalty.hh>
#include <core/data/io/alignment_io.hh>
```

(continues on next page)

(continued from previous page)

```
#include <core/data/sequence/Sequence.hh>

#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Calculate all pairwise sequence alignments between sequence profiles
 *
 * CATEGORIES: core/alignment/NWAlignerAnyGap
 * KEYWORDS:   FASTA input; Needleman-Wunsch; sequence alignment
 * GROUP:      Alignments
 */
int main(const int argc, const char* argv[]) {

  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  using namespace core::data::io;
  using namespace core::data::sequence;
  using namespace core::alignment::scoring;

  float gap_open = -10;
  float gap_extend = -1;
  if(argc == 5) {
    gap_open = atof(argv[3]);
    gap_extend = atof(argv[4]);
  }

  utils::Logger logs("ap_align_profiles");

  // --- the query profile
  SequenceProfile_SP query = read_profile_table(argv[1]);
  std::vector<float> query_gap_open, query_gap_extend;
  query->get_probabilities(core::chemical::Monomer::GAP,query_gap_open);
  query->get_probabilities(core::chemical::Monomer::GPE,query_gap_extend);

  logs << utils::LogLevel::INFO << "Query sequence is: " << query->sequence<<"\n";

  // --- the template profile
  SequenceProfile_SP tmplt = read_profile_table(argv[2]);
  std::vector<float> tmplt_gap_open, tmplt_gap_extend;
  tmplt->get_probabilities(core::chemical::Monomer::GAP,tmplt_gap_open);
```

(continues on next page)

(continued from previous page)

```
  tmplt->get_probabilities(core::chemical::Monomer::GPE,tmplt_gap_extend);

  logs << utils::LogLevel::INFO << "Template sequence is: " << tmplt->sequence<<"\n";

  // --- scoring system
  const Picasso3 scoring(query,tmplt);
  const FrequencyScaledGapPenalty gaps(gap_open,gap_extend,query_gap_open,query_gap_
↪extend,tmplt_gap_open,tmplt_gap_extend);

  // --- create aligner object
  core::alignment::NWAlignerAnyGap<Picasso3,FrequencyScaledGapPenalty>␣
↪aligner(std::max(query->length(),tmplt->length()));

  auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
  float score = aligner.align(scoring,gaps);
  auto ali = aligner.backtrace();
  core::alignment::PairwiseSequenceAlignment seq_ali(ali, query, tmplt);
  std::cout << seq_ali.get_aligned_query('*') << "\n";
  std::cout << seq_ali.get_aligned_template('*') << "\n";

  auto end = std::chrono::high_resolution_clock::now();
  std::chrono::duration<double> time_span = std::chrono::duration_cast
↪<std::chrono::duration<double>>(end - start);
}
```

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:

```
#include <iostream>
#include <iomanip>

#include <core/data/io/Pdb.hh>
#include <core/calc/statistics/OnlineMultivariateStatistics.hh>
#include <utils/string_utils.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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)

```

(continues on next page)

(continued from previous page)

```
where the covariance between is computed

)";

/** @brief Reads a multimodel PDB trajectory and calculates correlation between atomic coordinates
 *
 * CATEGORIES: core::data::io::Pdb::fill_structure; core::calc::statistics::OnlineMultivariateStatistics
 * KEYWORDS: PDB input; statistics
 * GROUP: Structure calculations;
 */
int main(const int argc, const char *argv[]) {

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

  core::data::io::Pdb reader(argv[1], core::data::io::is_ca); // --- Read PDB file, may be gzip-ped; take only the lines with C-alphas
  std::vector<core::data::basic::Vec3> atoms(reader.count_atoms(0));
  std::vector<double> xyz(atoms.size() * 3);
  core::calc::statistics::OnlineMultivariateStatistics stats(xyz.size());

  // --- Read all models from the deposit, store alpha carbons from each model as a separate vector of double values
  for (size_t i = 0; i < reader.count_models(); ++i) { // --- Iterate over all models in the input file
    reader.fill_structure(i, atoms);
    // --- utilize coordinates of the new pose
    for (size_t j = 0; j < atoms.size(); ++j) {
      xyz[j * 3] = atoms[j].x;
      xyz[j * 3 + 1] = atoms[j].y;
      xyz[j * 3 + 2] = atoms[j].z;
    }
    stats(xyz);
  }

  std::vector<std::string> labels;
  const auto structure = reader.create_structure(0);
  for(auto it = structure->first_const_residue(); it!=structure->last_const_residue(); ++it)
    labels.push_back(utils::string_format("%4d %3s CA", (**it).id(), (**it).residue_type().code3.c_str()));


  std::cout << "# i-resid coord  j-resid coord  i    j  correlation\n";
  std::string xyz_chars = "XYZ";
  std::cout << "#ipos j-pos correlation\n";
  for (size_t i = 0; i < xyz.size(); ++i) {
    for (size_t j = 0; j < xyz.size(); ++j) {
      std::cout << labels[int(i / 3)] << "-" << xyz_chars[i % 3] << " " << labels[int(j / 3)] << "-" << xyz_chars[j % 3];
      std::cout << " " << std::setw(4) << i << " " << std::setw(4) << j << " " << stats.covar(i, j) << "\n";
    }
  }
}
```

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:

```
#include <iostream>

#include <core/data/io/BlastXMLReader.hh>
#include <core/alignment/on_alignment_computations.hh>
#include <core/calc/clustering/DistanceByValues1B.hh>
#include <core/calc/clustering/HierarchicalCluster.hh>
#include <core/calc/clustering/HierarchicalClustering1B.hh>

#include <utils/exit.hh>
#include <utils/LogManager.hh>

std::string program_info = R"(

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
```

(continues on next page)

(continued from previous page)

```
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

)";

/** @brief Reads output from blast search (XML format) and selects a non-redundant subset of sequences
 *
 * CATEGORIES: core::calc::clustering::HierarchicalClustering; core::data::io::BlastXMLReader
 * KEYWORDS:   hierarchical clustering; blast
 * GROUP:      File processing;Data filtering
 */
int main(const int argc, const char *argv[]) {

  utils::LogManager::INFO();

  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  using namespace core::data::io;
  using namespace core::calc::clustering;

  BlastXMLReader blast_reader;
  auto hits = blast_reader.parse(argv[1]);
  std::vector<std::string> sequences; // --- vector containing all sequences from the last iteration of psiblast
  std::vector<std::string> seq_ids;   // --- vector containing identifiers of these sequences; sequences.size() equals to seq_ids.size()
  std::map<std::string,std::string> seq_by_id; // --- maps sequence IDs (keys) to the respective sequences (values)
  for (const auto hsp: hits.back()) {
    seq_ids.push_back(hsp.hit_accession());
    sequences.push_back(std::string(hsp.query_start() - 1, '-') + hsp.sbjct());
    seq_by_id[seq_ids.back()] = sequences.back();
  }

  DistanceByValues1B dist(seq_ids, 254,255);
  for(core::index4 i=1;i<sequences.size();++i)
    for(core::index4 j=0;j<i;++j) {
      double val = core::alignment::sum_identical(sequences[i],sequences[j]);
      val /= std::min(sequences[i].length(), sequences[j].length());
      core::index1 d = core::index1(250 * (1-val));
//        std::cout << i << " " << j << " " << core::alignment::sum_identical(sequences[i], sequences[j])
//                  << " " << val << " " << int(d) << "\n";
      dist.set(i, j, d);
      dist.set(j, i, d);
    }
```

(continues on next page)

(continued from previous page)

```

  HierarchicalClustering1B hac(dist.labels(), "");
  CompleteLink1B merge;
  hac.run_clustering(dist, merge);

  // --- Uncomment the line below to print the clustering tree (may be a lot of output)
  //  hac.write_merging_steps(std::cerr);

  std::vector<std::string> elements; // --- vector used to store elements of each cluster
  core::index1 cutoff = core::index1((1.0 - atof(argv[2])) * 250);
  std::cerr << "# clustering cutoff set to " << int(cutoff) << "\n";
  auto clusters = hac.get_clusters(cutoff, 1);
  std::cerr << "# " << sequences.size() << " hits' set reduced to " << clusters.size() << " representatives\n";
  for (core::index2 i = 0; i < clusters.size(); i++) {
    const auto & c = clusters[i];
    std::string medoid_id = medoid_by_average_distance<core::index1, std::string, DistanceByValues1B >(c, dist).medoid;
    elements.clear();
    collect_leaf_elements(std::static_pointer_cast<BinaryTreeNode<std::string>>(c), elements);
    std::cout << "> " << medoid_id;
    if(elements.size() > 1) {
      std::cout << " represents also:";
      for(const std::string & e: elements)
        if(e!=medoid_id) std::cout << " "<<e;
    }
    core::data::sequence::remove_gaps(seq_by_id[medoid_id]);
    std::cout << "\n" << seq_by_id[medoid_id] << "\n\n";
  }
}
```

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:

```
#include <iostream>
#include <fstream>

#include <stdexcept>
#include <core/data/io/BlastXMLReader.hh>
#include <core/algorithms/trees/algorithms.hh>
#include <core/index.hh>
#include <core/data/io/XML.hh>

#include <core/data/io/XMLElement.hh>
#include <core/data/io/Hsp.hh>
#include <utils/exit.hh>

std::string program_info = R"(

ap_blastxml_to_fasta reads a XML file produced by PsiBlast and extracts sequences of␣
↪all hits.

```

(continues on next page)

(continued from previous page)

```
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"

)";

using namespace core::data::io;

struct BlastXMLVisitor {

  void operator()(std::shared_ptr<core::algorithms::trees::TreeNode<XMLElementData>> n) {

    if (n->element.name() == "Hsp") {
      auto xmlel = std::static_pointer_cast<XMLElement>(n);
      auto xmlel_root = std::static_pointer_cast<XMLElement>(n->get_root()->get_root());

      const std::string &sequence = xmlel->find_value("Hsp_hseq");
      const std::string &seq_name = xmlel_root->find_value("Hit_accession");
      std::cout << "> " << seq_name << "\n" << sequence << "\n";
    }
  }
};

/** @brief Reads XML produced by psiblast and creates FASTA file containing all hits
 *
 * CATEGORIES: core::data::io::XML; core::algorithms::trees::TreeNode
 * KEYWORDS:   XML; data structures
 * GROUP:      File processing;Format conversion
*/
int main(int argc, char *argv[]) {

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

  XML xxx;
  std::shared_ptr<XMLElement> root = xxx.load_data(argv[1]);

  auto it = root->begin();
  while ((*it)->element.name() != "BlastOutput_iterations") ++it; // --- Visit branches until you find BlastOutput_iterations

  core::index2 iteration_counter = 0;
  for (const auto &v : **it) {
    if (v->element.name() == "Iteration") {
      std::cout << "\n# ------ iteration " << ++iteration_counter << " --------\n";
      core::algorithms::trees::depth_first_preorder((*it)->get_right(), BlastXMLVisitor());
    }
  }

```

(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 --ELPDMYGWTKENVQVFGKWTGIEVTYQGNGSHVTAQSSDTGTALKKLKKLTITLGE
[        UniRef50_A0A111B192]   151   221    0 (  0%)   48 ( 82%)   4.26e-22    1  ␣
↪94 AEEVPDMYGWTKETAETLAKWLNIELEFQGSGSTVQKQDVRANTAIKDIKKITLTLGD
[            UniRef50_P59676]   750   229    0 (  0%)   48 ( 82%)   2.43e-21    1 ␣
↪693 AEEVPDMYGWTKETAETLAKWLNIELEFQGSGSTVQKQDVRANTAIKDIKKITLTLGD
[            UniRef50_T0UT66]   466   227    0 (  0%)   29 ( 50%)   3.87e-21    2 ␣
↪410 -DAVPDMYGWTKKNADIFGEWTGIEITYKGSGKKVTKQSVKMNTSLNKTKKITLTLGD
[        UniRef50_A0A0T8ADZ4]   322   223    0 (  0%)   58 (100%)   5.32e-21    1 ␣
↪265 VEEIPDMYGWKKETAETFAKWLDIELEFEGSGSVVQKQDVRTNTAIKNIKKIKLTLGD
[        UniRef50_A0A139PMG7]   412   222    0 (  0%)   48 ( 82%)   1.37e-20    1 ␣
↪355 AEEVPDMYGWTKATAETLAKWLNIELEFEGSGSTVQKQDVRANTAIKDIKKITLTLGD
[        UniRef50_A0A0E9EQ17]   236   212    0 (  0%)   29 ( 51%)   5.33e-20    3 ␣
↪181 --EMPDMYGWTKKNVETFGEWLGIKVHVKSKGSKVVAQSVKTNASLKKIKEITITLGD
```

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

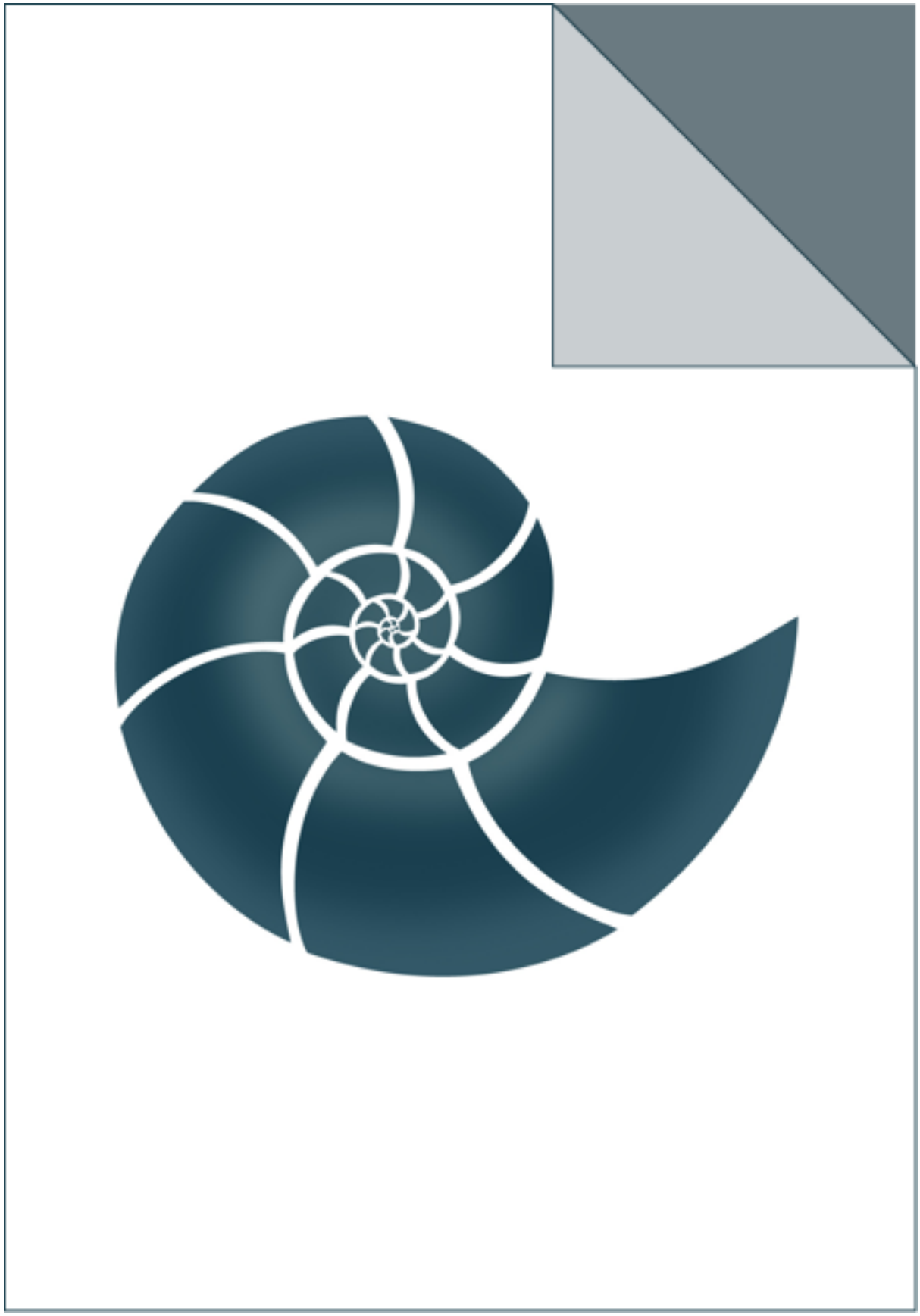
Program source:

```
#include <iostream>
#include <fstream>

#include <stdexcept>
#include <core/data/io/BlastXMLReader.hh>
#include <core/algorithms/trees/algorithms.hh>
#include <core/index.hh>
#include <core/data/io/XML.hh>

#include <core/data/io/XMLElement.hh>
#include <core/data/io/Hsp.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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 --ELPDMYGWTKENVQVFGKWTGIEVTYQGNGSHVTAQSSDTGTALKKLKKLTITLGE
[        UniRef50_A0A111B192]   151   221    0 (  0%)   48 ( 82%)   4.26e-22    1  ␣
↪94 AEEVPDMYGWTKETAETLAKWLNIELEFQGSGSTVQKQDVRANTAIKDIKKITLTLGD
[            UniRef50_P59676]   750   229    0 (  0%)   48 ( 82%)   2.43e-21    1 ␣
↪693 AEEVPDMYGWTKETAETLAKWLNIELEFQGSGSTVQKQDVRANTAIKDIKKITLTLGD
[            UniRef50_T0UT66]   466   227    0 (  0%)   29 ( 50%)   3.87e-21    2 ␣
↪410 -DAVPDMYGWTKKNADIFGEWTGIEITYKGSGKKVTKQSVKMNTSLNKTKKITLTLGD
[        UniRef50_A0A0T8ADZ4]   322   223    0 (  0%)   58 (100%)   5.32e-21    1 ␣
↪265 VEEIPDMYGWKKETAETFAKWLDIELEFEGSGSVVQKQDVRTNTAIKNIKKIKLTLGD
[        UniRef50_A0A139PMG7]   412   222    0 (  0%)   48 ( 82%)   1.37e-20    1 ␣
↪355 AEEVPDMYGWTKATAETLAKWLNIELEFEGSGSTVQKQDVRANTAIKDIKKITLTLGD
[        UniRef50_A0A0E9EQ17]   236   212    0 (  0%)   29 ( 51%)   5.33e-20    3 ␣
↪181 --EMPDMYGWTKKNVETFGEWLGIKVHVKSKGSKVVAQSVKTNASLKKIKEITITLGD

)";

```

(continues on next page)

(continued from previous page)

```
using namespace core::data::io;
/** @brief Reads XML produced by psiblast and creates High Scoring FASTA Pair
 *
 * CATEGORIES: core::data::io::XML; core::algorithms::trees::TreeNode
 * KEYWORDS:   XML; data structures; HSP
 * GROUP:      File processing;Format conversion
*/

int main(int argc, char *argv[]) {

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

  // ---------- creates BlastXMLReader object
  BlastXMLReader p;
  // ---------- parse data from XML file and returns it to variable
  auto iterations = p.parse(argv[1]);
  std::cout << Hsp::output_header << "\n";
  // print Hsp line for every hit for every iteration
  for (core::index2 i = 0; i < iterations.size(); i++) {
    for (core::index2 j = 0; j < iterations[i].size(); j++) std::cout << iterations[i][j] << "\n";
  }

  return 0;
}
```



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-folt 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:

```
#include <iostream>

#include <core/index.hh>
#include <core/calc/statistics/Random.hh>
#include <core/calc/statistics/OnlineStatistics.hh>
#include <core/calc/statistics/simple_statistics.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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-folt 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)
```

(continues on next page)

(continued from previous page)

```
USAGE:
    ap_bootstrap_quantile quantile_value infile
    ap_bootstrap_quantile quantile_value


)";

/** @brief Reads a file with real values and calculates statistics for a given quantile
 *
 * If no input file is provided, the program calculates the statistics from a random sample
 *
 * CATEGORIES: core::calc::statistics::simple_statistics.hh; core::calc::statistics::Random
 * KEYWORDS:   random numbers; statistics
 * GROUP: Statistics;
 */
int main(const int argc, const char *argv[]) {

  if(argc ==1)  utils::exit_OK_with_message(program_info);

  double quantile_level = atof(argv[1]);

  core::calc::statistics::OnlineStatistics stats;
  if(argc < 3) {
    // --- complain about missing program parameter
    //std::cerr << program_info;
    // ---------- Use the random engine if no data is provided - for testing purposes
    size_t n_data = 10000;
    std::vector<double> data(n_data);
    core::calc::statistics::Random r = core::calc::statistics::Random::get();
    r.seed(12345);  // --- seed the generator for repeatable results
    core::calc::statistics::NormalRandomDistribution<double> normal_random;
    for (core::index4 n = 0; n < n_data; ++n) data[n] = normal_random(r);
    const auto out = core::calc::statistics::bootstrap_quantile(data, quantile_level, 100);
    std::cout << "q-level value stdev\n" << quantile_level << " " << out.first << " " << out.second << "\n";
  } else {
    std::vector<double> data;
    for(size_t i_file=2;i_file<argc;++i_file) {
      data.clear();
      std::ifstream in(argv[i_file]);
      double r;
      while(in) {
        in >> r;
        data.push_back(r);
      }
      in.close();
      const auto out = core::calc::statistics::bootstrap_quantile(data, quantile_level, 100);
      std::cout << "fname q-level value stdev\n" << argv[i_file] << " " << quantile_level << " " << out.first << " "
                << out.second << "\n";
    }
  }
```

(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:

```
#include <iostream>
#include <core/data/io/Pdb.hh>
#include <utils/exit.hh>
#include <iomanip>

std::string program_info = R"(

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

USAGE:
    ap_create_crystal 5edw.pdb

)";

/** @brief ap_create_crystal reads a given PDB file and prints all atoms in a unit
↪cell.
 *
 * CATEGORIES: core/data/io/Pdb;
```

(continues on next page)

(continued from previous page)

```
 * KEYWORDS:   PDB input; PDB line filter; Structure
 * GROUP: Structure calculations;
 */
int main(const int argc, const char *argv[]) {

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

  core::data::io::Pdb reader(argv[1], // file name (PDB format, may be gzip-ped)
    core::data::io::keep_all,         // a predicate - now read all atoms
      core::data::io::keep_all, true);                              // parse PDB header

  std::shared_ptr<core::data::io::Remark290> r290 = reader.symmetry_operators();
  core::data::structural::Structure_SP s = reader.create_structure(0);
  std::cout << "# Symmetry operators found: " << r290->count_operators() << "\n";
  core::data::basic::Vec3 tmp;
  core::index2 im = 0;
  for (const auto &rt: *r290) {
    std::cout << "MODEL  " << std::setw(6) << ++im << "\n";
    for (auto a_it = s->first_atom(); a_it != s->last_atom(); ++a_it) {
      tmp.set((**a_it));
      rt.apply(**a_it);
      std::cout << (*a_it)->to_pdb_line() << "\n";
      (*a_it)->set(tmp);
    }
    std::cout << "ENDMDL\n";
  }
}
```

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:

```
#include <iostream>

#include <core/index.hh>
#include <core/data/io/Pdb.hh>
#include <core/data/basic/Vec3I.hh>
#include <core/calc/statistics/Histogram.hh>

#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Calculates  Radial Distribution Function
 *
 * CATEGORIES: core::data::basic::Vec3
 * KEYWORDS: PDB input; simulation
 * GROUP: Structure calculations;
 */
int main(const int argc, const char *argv[]) {
```

(continues on next page)

(continued from previous page)

```

  if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  double L = utils::from_string<double>(argv[4]); // The third parameter is the box width (in Angstroms)
  core::data::basic::Vec3I::set_box_len(L);
  core::data::io::Pdb reader(argv[1]); // --- file name (PDB format, may be gzip-ped)

  core::index4 n_atoms = reader.count_atoms(0);
  core::index4 n_frmes = reader.count_models();

  std::vector<core::data::basic::Vec3I> frame_i(n_atoms);

  core::calc::statistics::Histogram<double, core::index4> h(0.01, 0, L/4.0);
  // ---------- Load coordinates to memory and accumulate RDF ----------
  for (int i_start = 0; i_start < n_frmes; ++i_start) {
    reader.fill_structure(i_start, frame_i);
    for(core::index4 i=1;i<n_atoms;++i) {
      for(core::index4 j=0;j<i;++j) {
        double d = frame_i[i].distance_to(frame_i[j]);
        h.insert(d);
      }
    }
  }

  double norm = 4 * M_PI * n_atoms / pow(L, 3.0) * n_frmes;
  for(core::index4 i=0;i<h.count_bins();++i) {
    std::cout << h.bin_middle_val(i) << " " << h.get_bin(i) / (norm * h.bin_middle_val(i) * h.bin_middle_val(i)) << " "
              << h.get_bin(i) << "\n";
  }
  // ---------- Calculate displacement ----------
}
```

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:

```
#include <vector>
#include <string>
#include <fstream>
#include <iostream>
#include <cstring>

#include <core/data/io/Pdb.hh>
```

(continues on next page)

(continued from previous page)

```
#include <utils/options/output_options.hh>
#include <utils/options/input_options.hh>
#include <utils/exit.hh>

using namespace core::data::io; // PDB is from this namespace
using namespace core::data::structural;
using namespace core::calc::structural;
using namespace utils;

std::string program_info = R"(

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
)";

/** @brief Reads cat_list of pdb files and creates multimodel pdb with CA only
 *
 * CATEGORIES: core::data::io::is_ca
 * KEYWORDS:   PDB input; CA only; structure selectors;PDB output; PDB line filter
 * GROUP:      File processing;Format conversion
 */
int main(const int argc, const char *argv[]) {

  if (argc < 2) utils::exit_OK_with_message(program_info);

  PdbLineFilter filter = core::data::io::is_ca;
  std::ofstream out;
  out.open("all.pdb");
  std::ifstream pdb_list(argv[1]);
  std::string pdb_file;
  std::getline(pdb_list, pdb_file);
  Pdb pdb = Pdb(pdb_file, filter);
  Structure_SP strctr = pdb.create_structure(0);
  out << "MODEL     1\n";
  for (auto it = strctr->first_atom(); it != strctr->last_atom(); it++) out << (*it)->to_pdb_line() << "\n";
  out << "ENDMDL\n";
  core::index4 i = 2;
  while (std::getline(pdb_list, pdb_file)) {
    Pdb pdb = Pdb(pdb_file, filter);
    pdb.fill_structure(0, *strctr);
    out << utils::string_format("MODEL     %6d\n", i);
    for (auto it = strctr->first_atom(); it != strctr->last_atom(); it++)
      out << (*it)->to_pdb_line() << "\n";
    out << "ENDMDL\n";
    i++;
```

(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:

```
#include <vector>
#include <algorithm>

#include <core/data/io/Pdb.hh>
#include <core/calc/structural/interactions/ContactMap.hh>

#include <utils/exit.hh>

```

(continues on next page)

(continued from previous page)

```
std::string program_info = R"(

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
)";

/** @brief Calculates overlap between contact maps calculated for two (or more) structures
 *
 * CATEGORIES: core::calc::structural::ContactMap
 * KEYWORDS: PDB input; contact map
 * GROUP: Structure calculations;
 */
int main(const int argc, const char *argv[]) {

  if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  using namespace core::data::io; // PDB is from this namespace
  using namespace core::data::structural;
  using namespace core::data::structural::selectors;    // --- for all structural selectors
  using namespace core::calc::structural::interactions;

  AtomSelector_SP selector = std::make_shared<IsSC>();
  core::data::io::PdbLineFilter filter = core::data::io::is_not_water;
  if (std::strcmp(argv[1],"CA")==0 ) {
    selector = std::make_shared<IsNamedAtom>(" CA ");
    core::data::io::PdbLineFilter filter = core::data::io::is_ca;
  }
  if (std::strcmp(argv[1],"CB")==0) {
    core::data::io::PdbLineFilter filter = core::data::io::is_cb;
    selector = std::make_shared<IsNamedAtom>(" CB ");
  }
  double cutoff = utils::from_string<double>(argv[(argc==5) ? 4 : 3]); // The third/fourth parameter is the contact distance (in Angstroms)

```

(continues on next page)

(continued from previous page)

```
  // --- This is the case when user gave a reference structure (e.g. the native one)
  if (argc == 5) {
    Pdb pdb_native = Pdb(argv[2], filter);
    core::data::structural::Structure_SP reference_structure = pdb_native.create_
↪structure(0);
    ContactMap_SP reference_map = std::make_shared<ContactMap>(*reference_structure,␣
↪cutoff, selector);

    Pdb models_pdb = Pdb(argv[3], filter);
    ContactMap cmap(*reference_structure, cutoff, selector);
    std::vector<std::pair<core::index2, core::index2>> contacts;
    for (core::index4 i = 0; i < models_pdb.count_models(); ++i) {
      models_pdb.fill_structure(i,*reference_structure);
      ContactMap cmap(*reference_structure, cutoff, selector);
      std::cout << i << " " << reference_map->jaccard_overlap_coefficient(cmap) << "\n
↪";
    }
  } else {
    Pdb models_pdb = Pdb(argv[2], filter);
    core::data::structural::Structure_SP structure = models_pdb.create_structure(0);
    std::vector<std::vector<std::pair<core::index2, core::index2>>> models(models_pdb.
↪count_models());
    for (int i = 0; i < models_pdb.count_models(); ++i) {
      models_pdb.fill_structure(i,*structure);
      ContactMap cmap(*models_pdb.create_structure(i), cutoff, selector);
      cmap.nonempty_indexes(models[i]);
    }

    for (core::index4 i = 1; i < models.size(); ++i) {
      for (core::index4 j = 0; j < i; ++j)
        std::cout << i << " " << j << " "
              << core::calc::structural::interactions::jaccard_overlap_
↪coefficient(models[i], models[j]) << "\n";
    }
  }
}
```

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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/calc/structural/transformations/Crmsd.hh>
#include <utils/options/Option.hh>
#include <utils/options/OptionParser.hh>
#include <utils/options/input_options.hh>
#include <utils/options/structures_from_cmdline.hh>
#include <utils/options/select_options.hh>
#include <utils/options/selector_from_cmdline.hh>
#include <utils/exit.hh>
#include <core/algorithms/basic_algorithms.hh>
#include <utils/options/output_options.hh>
#include <core/data/structural/selectors/SelectChainBreaks.hh>

std::string program_info = R"(

Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on common subset of␣
↪atoms
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
    using namespace core::data::io;
    using namespace utils::options;
    using namespace core::data::basic;
    using namespace core::data::structural;
    using namespace core::data::structural::selectors;
    using namespace core::calc::structural;
    using namespace core::calc::structural::transformations;

    Crmsd<std::vector<Vec3>, std::vector<Vec3>> rms;

    utils::options::OptionParser &cmd = OptionParser::get("ap_crmsd_on_common_subset");
    // ---------- input PDB structures
    cmd.register_option(input_pdb, input_pdb_native, input_pdb_list, input_pdb_path, input_pdb_header);
    // ---------- selecting options
    cmd.register_option(select_ca, select_bb, select_bb_cb, select_cb, select_atoms_by_name, select_aa, select_chains, all_models);
    // ---------- output PDB structures
    cmd.register_option(output_pdb,output_name_prefix);
    cmd.register_option(verbose, db_path);

  if (!cmd.parse_cmdline(argc, argv)) return 1;
    Structure_SP native = native_from_cmdline();
    std::vector<core::data::structural::Structure_SP> structures;
    std::vector<std::string> structure_ids;
    utils::options::structures_from_cmdline(structure_ids, structures);

    // ---------- Load atoms from the native structure
    std::vector<std::string> native_names;
    std::map<std::string, PdbAtom_SP> native_name_to_atom;
    extract_atom_by_name(native, native_names, native_name_to_atom, select_aa.was_used());
    std::sort(native_names.begin(), native_names.end());

    std::vector<Vec3> q, t;
    std::cout << "native nres, atoms " << native->count_residues() << " " << native->count_atoms() << "\n";
    std::cout << "model nres, atoms " << structures[0]->count_residues() << " " << structures[0]->count_atoms() << "\n";

  for (auto i = 0; i < structures.size(); ++i) {
    // ---------- Load atoms from a query structure
    std::vector<std::string> q_names;
    std::map<std::string, PdbAtom_SP> q_name_to_atom;
    extract_atom_by_name(structures[i], q_names, q_name_to_atom, select_aa.was_used());
    std::sort(q_names.begin(), q_names.end());

    // ---------- find the common subset
    std::vector<std::string> common_atom_names;
    core::algorithms::intersect_sorted(native_names.begin(), native_names.end(), q_names.begin(), q_names.end(),
        common_atom_names);

    // ---------- get the two subset of atoms
    q.clear();
```

(continues on next page)

(continued from previous page)

```
    t.clear();
    std::shared_ptr<std::vector<double>> errors = std::make_shared<std::vector<double>
↪>();
    for (const std::string &name: common_atom_names) {
      t.push_back(*q_name_to_atom[name]);
      q.push_back(*native_name_to_atom[name]);
    }

    double rms_val = rms.crmsd(q, t, q.size(), true);
    rms.calculate_crmsd_value(q, t, q.size(), errors);
    // ---------- This is the moment when we can dump the transformed structure into␣
↪a PDB file
    if(output_pdb.was_used()) {
        int iatm = -1;
        std::string fname;
        if(input_pdb_list.was_used())
        fname = option_value<std::string>(output_name_prefix)+utils::split(structure_
↪ids[i],{'/'}).back() + ".pdb";
        else  fname = option_value<std::string>(output_pdb);
        std::ofstream out(fname);
      for (const std::string &name: common_atom_names) {
          q_name_to_atom[name]->b_factor((*errors)[++iatm]);
          out << q_name_to_atom[name]->to_pdb_line() << "\n";
//        std::cout << name << " " << (*errors)[++iatm] << "\n";
      }

    }

    std::cout << native->code() << " " << i << " crmsd: " << rms_val << "\n";
  }
}
```

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:

```
#include <iostream>
#include <map>

#include <core/data/io/Pdb.hh>
#include <core/protocols/PairwiseCrmsd.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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.

)";

using namespace core::data::structural;

/** @brief ap_docking_crmsd calculates crmsd between two ligand positions after docking to a receptor.
```

(continues on next page)

(continued from previous page)

```
 *
 * CATEGORIES: core::protocols::PairwiseCrmsd
 * KEYWORDS: PDB input; crmsd; docking; structure selectors
 * GROUP: Structure calculations;  Docking;
 */
int main(const int argc, const char* argv[]) {

  using namespace core::data::structural::selectors;

  if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  const std::string code(argv[2]);    // --- The ligand code is the second parameter of the program
  AtomSelector_SP select_ligand = nullptr; // --- Ligand selector object
  if (code.size() == 3) // --- If the code is 3 characters long, its a residue code
    select_ligand = std::static_pointer_cast<AtomSelector>(std::make_shared<SelectResidueByName>(code));
  else {
    AtomSelector_SP select_chain = std::static_pointer_cast<AtomSelector>(std::make_shared<ChainSelector>(code[0]));
    std::shared_ptr<LogicalANDSelector> select_chain_ca = std::make_shared<LogicalANDSelector>();
    select_chain_ca->add_selector( std::make_shared<IsCA>() );
    select_chain_ca->add_selector(select_chain);
    select_ligand = select_chain_ca;
  }

  std::shared_ptr<LogicalANDSelector> select_receptor = std::make_shared<LogicalANDSelector>(); // --- Receptor selector object
  AtomSelector_SP select_not_ligand = std::static_pointer_cast<AtomSelector>(std::make_shared<InverseAtomSelector>(*select_ligand));
  select_receptor->add_selector(std::make_shared<IsCA>());
  select_receptor->add_selector(select_not_ligand);

  core::protocols::PairwiseCrmsd crmsd_protocol(select_receptor, select_ligand);

  for(core::index2 i=3;i<argc;++i) {
      core::data::io::Pdb reader(argv[i], core::data::io::is_not_hydrogen);
      if (reader.count_models()>1) {
        for (core::index2 j = 0; j < reader.count_models(); ++j)
          crmsd_protocol.add_input_structure(reader.create_structure(j), utils::string_format("%s:%4d",argv[i], j));
      } else
          crmsd_protocol.add_input_structure(reader.create_structure(0), argv[i]);
  }

  crmsd_protocol.crmsd_cutoff(50.0); // crmsd cutoff large enough to get some output
  crmsd_protocol.output_stream( std::shared_ptr<std::ostream>(&std::cout, [](void*) {}) );
  if(std::string(argv[1]) != "-") {
    core::data::io::Pdb reader(argv[1], core::data::io::is_not_hydrogen);
    Structure_SP native = reader.create_structure(0);
    crmsd_protocol.calculate(native);
  } else crmsd_protocol.calculate();

}
```

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:

```
#include <iostream>

#include <utils/exit.hh>
#include <utils/io_utils.hh>

std::string program_info = R"(

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

)";

/** @brief Simple app downloads a pdb file from RCSB website
 *
 * CATEGORIES: utils/read_properties_file
 * KEYWORDS:   PDB file; download
 * GROUP:      File processing
 */
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␣
↪missing program parameter
25
26   std::ofstream out(std::string(argv[1])+".pdb");
27   out << utils::download_pdb(argv[1]);
28   out.close();
29 }
```

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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/calc/structural/ProteinArchitecture.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

(continues on next page)

(continued from previous page)

```
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

)";
/** @brief DSSP implementation
 *
 * CATEGORIES: core::calc::structural::ProteinArchitecture;
 * KEYWORDS:   PDB input; Hydrogen bonds; secondary structure; DSSP; Protein structure features
 * GROUP: Structure calculations;
 */
int main(const int argc, const char* argv[]) {

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

  core::data::io::Pdb reader(argv[1]); // file name (PDB format, may be gzip-ped)
  core::data::structural::Structure_SP strctr = reader.create_structure(0);

  core::calc::structural::ProteinArchitecture pa(*strctr);
  std::cout << pa.hec_string() << "\n";
}
```

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:

```
#include <iostream>
#include <core/data/io/ss2_io.hh>
#include <core/data/io/DsspData.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Reads a DSSP file and prints the secondary structure of each chain in SS2 format.
 *
 * @see ap_DsspData.cc converts DSSP to FASTA format
 * CATEGORIES:  core::data::io::DsspData
 * KEYWORDS:   DSSP; Structure; secondary structure; Format conversion
```

(continues on next page)

(continued from previous page)

```
 * GROUP:      File processing; Format conversion
 */
int main(const int argc, const char* argv[]) {

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

  // --- read a DSSP file - the first command line argument of the program
  core::data::io::DsspData dssp(argv[1], true);
  for (const auto & ss2 : dssp.sequences())                // --- for each protein sequence found in the DSSP data ...
    core::data::io::write_ss2(*ss2,std::cout);          // --- print it as SS2!
}
```

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:

```
#include <iostream>

#include <core/algorithms/UnionFind.hh>
#include <core/data/io/fasta_io.hh>
#include <utils/exit.hh>
#include <utils/io_utils.hh>

std::string program_info = R"(

ap_find_in_fasta reads a file in FASTA format and prints only these sequences which␣
↪satisfy the following filters:
```

(continues on next page)

(continued from previous page)

```
  - 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

)";

/** @brief This program reads a file with sequences in FASTA format and sorts them by length.
 * DNA sequences, sequences that are shorter than 15 residues and those having more than 10 Xs are removed
 *
 * CATEGORIES: core/data/io/fasta_io;
 * KEYWORDS:   FASTA input; FASTA output; sequence; FASTA; pre-processing; sequence filters
 * GROUP:      File processing;Data filtering
 */
int main(const int argc, const char* argv[]) {

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

  // --- Sequence_SP is just a std::shared_ptr to core::data::sequence::Sequence type
  using core::data::sequence::Sequence_SP;

  // --- Create a container where the sequences will be stored
  std::vector<Sequence_SP> sequences;

  // --- Read a file (or files) with FASTA sequences; sequences are appended to the given vector
  for (int i = 1; i < argc; ++i) core::data::io::read_fasta_file(argv[i], sequences);

  // --- Remove sequences that do not come from proteins
  sequences.erase(std::remove_if(sequences.begin(),sequences.end(),
      [](const Sequence_SP s){ return !s->is_protein_sequence;}),sequences.end());

  // --- Remove sequences that are too short
  sequences.erase(std::remove_if(sequences.begin(),sequences.end(),
      [](const Sequence_SP s){ return s->length()<20;}),sequences.end());

  // --- Remove sequences that contain 10 or more 'X' characters (i.e. unknown amino acids)
  sequences.erase(std::remove_if(sequences.begin(),sequences.end(),
      [](const Sequence_SP s){ return std::count(s->sequence.begin(), s->sequence.end(), 'X')>10;}),sequences.end());

  // --- Now, sort the sequences by length
  std::sort(sequences.begin(),sequences.end(),
      [](const Sequence_SP si,const Sequence_SP sj){ return si->length()<sj->length();});

  // --- Remove duplicates
```

(continues on next page)

(continued from previous page)

```
  core::algorithms::UnionFind<Sequence_SP, core::index4> uf;
  uf.add_element(sequences[0]);
  for (size_t i = 1; i < sequences.size(); ++i) {
    uf.add_element(sequences[i]);
    for (int j = i - 1; j >= 0; --j) {
      if (sequences[i]->length() - sequences[j]->length() > 20) break;
      if (sequences[i]->sequence.find(sequences[j]->sequence) != std::string::npos)
↪uf.union_set(i, j);
    }
  }
  for (size_t i = 0; i < sequences.size(); ++i) {
    core::index4 set_id = uf.find_set(i);
    if (set_id != i) {
      std::string new_header = sequences[set_id]->header() + " " + sequences[i]->
↪header();
      sequences[set_id]->header(new_header);
    }
  }

  // --- Print sequences to stdout
  for (size_t i = 0; i < sequences.size(); ++i) {
    if (uf.find_set(i) == i) std::cout << core::data::io::create_fasta_
↪string(*sequences[i]) << "\n";
  }
}
```

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 HGPs 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:

```
#include <iostream>

#include <core/alignment/FilterByHighlyGappedColumns.hh>
#include <core/data/io/clustalw_io.hh>
#include <utils/exit.hh>
#include <utils/io_utils.hh>
#include <core/data/io/fasta_io.hh>


std::string program_info = R"(
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

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

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:

```
#include <iostream>
#include <core/data/io/Pdb.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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
```

(continues on next page)

(continued from previous page)

```
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

)";

/** @brief Filters a PDB file by a given filter
 *
 * CATEGORIES: core::data::io::Pdb;
 * KEYWORDS:   PDB input; PDB line filter
 */
int main(const int argc, const char* argv[]) {

  if(argc < 3) {
    program_info += "\nKnown filters:\n";
    for (const std::string &name: core::data::io::Pdb::pdb_filter_names)
      program_info += "\t" + name;
    utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
  }
  std::string filter_names(argv[2]);
  for (int i = 3; i < argc; ++i) filter_names += " " + std::string(argv[i]);
  core::data::io::Pdb reader(argv[1], // file name (PDB format, may be gzip-ped)
                             filter_names, // filter names combined into a single string
                             false); // don't parse header to achieve highest speed

  for (int im = 0; im < reader.count_models(); ++im)
    for (const core::data::io::Atom &pdb_line : (*reader.atoms[im]))
      std::cout << pdb_line.to_pdb_line() << "\n";

}
```

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:

```
#include <iostream>

#include <core/data/io/fasta_io.hh>
#include <utils/exit.hh>
#include <utils/io_utils.hh>

std::string program_info = R"(

Program reads a sequence database in FASTA format and a text file with sequence␣
↪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:

```
#include <iostream>
#include <core/data/io/alignment_io.hh>
#include <core/data/io/fasta_io.hh>
#include <core/data/io/pir_io.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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
```

(continues on next page)

(continued from previous page)

```
 -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
)";

/** @brief Extract alignments from HHPred output
 *
 * CATEGORIES: core::data::io::read_hhpred;
 * KEYWORDS:   sequence alignment; FASTA; PIR; Edinburgh
 * GROUP:      File processing; Format conversion
 */
int main(const int argc, const char* argv[]) {

  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about
↪missing program parameter

  std::vector<core::alignment::PairwiseSequenceAlignment_SP> alignments;
  core::data::io::read_hhpred(argv[1], alignments);

  char flag = argv[2][1]; // --- E, e, F, f, P, p
  switch(flag) {
    case 'E' :
    case 'e' :
      for(const auto & seq_ali : alignments)
        core::data::io::write_edinburgh(*seq_ali, std::cout, 80);
      break;
    case 'F' :
    case 'f' :
      for(const auto & seq_ali : alignments)
        std::cout << core::data::io::create_fasta_string(*seq_ali, 80) << "\n";
      break;
    case 'P' :
    case 'p' :
      for(const auto & seq_ali : alignments)
        std::cout << core::data::io::create_pir_string(*seq_ali, 80) << "\n";
      break;
    default: std::cerr << "Incorrect output format requested!\n";
  }

}
```

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:

```
#include <iostream>
#include <numeric>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/data/io/Pdb.hh>
#include <core/calc/structural/interactions/StackingInteraction.hh>
#include <core/calc/structural/interactions/StackingInteractionCollector.hh>
#include <core/calc/structural/interactions/VdWInteractionCollector.hh>
#include <core/calc/structural/interactions/VdWInteraction.hh>
#include <core/calc/structural/interactions/HydrogenBondInteraction.hh>
#include <core/calc/structural/interactions/HydrogenBondCollector.hh>
#include <utils/LogManager.hh>

#include <utils/exit.hh>

std::string program_info = R"(

Finds ligand - protein interactions in a given PDB file. Ligand code must also be␣
↪provided
```

(continues on next page)

(continued from previous page)

```

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

)";

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


/** @brief Finds stacking, hbonds and van der Waals interactions for ligand in a given PDB file.
 *
 * CATEGORIES: core::data::io::Pdb; core::calc::structural::interactions
 * KEYWORDS:   PDB input; PDB line filter; interactions
 * GROUP: Structure calculations;
 */
int main(const int argc, const char *argv[]) {

    if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
    utils::LogManager::FINER(); // --- INFO is the default logging level; set it to FINE to see more

    // --- Read a PDB file given as an argument to this program
    Pdb reader(argv[1], // --- input PDB file
               all_true(is_not_water, is_not_alternative, is_not_hydrogen,
                        invert_filter(is_bb)), // --- Inverted backbone selector reads only side chains
                keep_all, true); // --- yes, read header

    core::data::structural::Structure_SP s = reader.create_structure(0);
    std::string code_3 = argv[2];

    VdWInteractionCollector vdw_collector = VdWInteractionCollector();
    HydrogenBondCollector hb_collector = HydrogenBondCollector();
    StackingInteractionCollector stack_collector = StackingInteractionCollector();

    std::vector<ResiduePair_SP> v_sink;
    std::vector<ResiduePair_SP> hb_sink;
    std::vector<ResiduePair_SP> s_sink;

    hb_collector.collect(*s, hb_sink);
    vdw_collector.collect(*s, v_sink);
    stack_collector.collect(*s, s_sink);

    std::cout << VdWInteraction::output_header() << "\n";
    for (const ResiduePair_SP ri:v_sink) {
        VdWInteraction_SP bi = std::dynamic_pointer_cast<VdWInteraction>(ri);
        if (bi && bi->first_residue()->residue_type().code3.c_str() == code_3) std::cout << *bi << "\n";
```

(continues on next page)

(continued from previous page)

```
    }

    std::cout << HydrogenBondInteraction::output_header() << "\n";
    for (const ResiduePair_SP ri:hb_sink) {
        HydrogenBondInteraction_SP bi = std::dynamic_pointer_cast
↪<HydrogenBondInteraction>(ri);
        if (bi && bi->first_residue()->residue_type().code3.c_str() == code_3)␣
↪std::cout << *bi << "\n";
    }

    std::cout << StackingInteraction::output_header() << "\n";
    for (const ResiduePair_SP ri:s_sink) {
        StackingInteraction_SP bi = std::dynamic_pointer_cast<StackingInteraction>
↪(ri);
        if (bi && bi->first_residue()->residue_type().code3.c_str() == code_3)␣
↪std::cout << *bi << "\n";
    }
}
```

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:

```
#include <iostream>
#include <map>

#include <core/data/io/Pdb.hh>
#include <utils/string_utils.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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
```

(continues on next page)

(continued from previous page)

```
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

)";

/** @brief Finds contacts between a ligand molecule and a multimodel-protein.
 *
 * CATEGORIES: core::data::io::Pdb
 * KEYWORDS: PDB input; contact map; ligand
 * GROUP: Structure calculations;
 */
int main(const int argc, const char* argv[]) {

  if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  core::data::io::Pdb reader(argv[1]); // --- file name (PDB format, may be gzip-ped)

  const std::string code(argv[2]);    // --- The ligand code is the second parameter of the program
  double cutoff = utils::from_string<double>(argv[3]); // The third parameter is the contact distance (in Angstroms)

  std::map<std::string, int> results; // --- Map used to store results
  for (size_t i = 0; i < reader.count_models(); ++i) { // --- Iterate over all models in the input file
    core::data::structural::Structure_SP strctr = reader.create_structure(i);

    // --- Here we use a standard <code>find_if</code> algorithm to find the ligand residue by its 3-letter code
    auto ligand = std::find_if(strctr->first_residue(), strctr->last_residue(), [&code](core::data::structural::Residue_SP res) {return (res->residue_type().code3==code);});
    if(ligand== strctr->last_residue()) { // --- If no ligand - print a message and take next structure
      std::cerr << "Model " << i << " has no " << argv[2] << " residue\n";
      continue;
    }
    for (auto it_resid = strctr->first_residue(); it_resid != strctr->last_residue();++it_resid) {
      double d = (*it_resid)->min_distance(*ligand);
      if (d < cutoff) { // --- if this is close enough,
        std::string key = utils::string_format("%3s  %4s  %4d",(*it_resid)->residue_type().code3.c_str(),
          (*it_resid)->owner()->id().c_str(), (*it_resid)->id());
        if (results.find(key) == results.end()) results[key] = 1;
        else results[key]++;
      }
    }
  }

```

(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:

```
#include <iostream>

#include <core/index.hh>
#include <core/data/io/Pdb.hh>
#include <core/data/basic/Vec3I.hh>
#include <core/calc/statistics/OnlineStatistics.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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
```

(continues on next page)

(continued from previous page)

```
will be evaluated

)";

/** @brief Calculates  average displacement of a small molecule as a function of time over a trajectory
 *
 * CATEGORIES: core::data::basic::Vec3Cubic
 * KEYWORDS: PDB input; simulation
 * GROUP: Structure calculations;
 */
int main(const int argc, const char *argv[]) {

  if (argc < 4) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  double L = utils::from_string<double>(argv[3]); // The third parameter is the box width (in Angstroms)
  core::data::basic::Vec3I::set_box_len(L);
  core::data::io::Pdb reader(argv[1]); // --- file name (PDB format, may be gzip-ped)

  core::index4 n_atoms = reader.count_atoms(0);
  core::index4 n_frmes = reader.count_models();
  core::index4 t_max = n_frmes / 5;
  std::vector<std::vector<core::data::basic::Vec3I>> v;

  // ---------- Load coordinates to memory ----------
  for (int i_start = 0; i_start < n_frmes; ++i_start) {
    std::vector<core::data::basic::Vec3I> vi(n_atoms);
    reader.fill_structure(i_start, vi);
    v.push_back(vi);
  }

  // ---------- Calculate displacement ----------
  std::vector<core::calc::statistics::OnlineStatistics> avg(t_max);
  for (size_t i_start = 0; i_start < n_frmes - t_max - 1; ++i_start) {
    const std::vector<core::data::basic::Vec3I> &v0 = v[i_start];
    for (size_t i_t = 1; i_t <= t_max; ++i_t) {
      const std::vector<core::data::basic::Vec3I> &vi = v[i_start + i_t];
      for (size_t i_atom = 0; i_atom < n_atoms; ++i_atom)
        avg[i_t - 1](sqrt(v0[i_atom].distance_square_to(vi[i_atom])));
    }
  }

  for (size_t i_t = 1; i_t <= t_max; ++i_t) {
    std::cout << utils::string_format("%5d %f %f\n", i_t, avg[i_t - 1].avg(), sqrt(avg[i_t - 1].var()));
  }
}
```

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 CEEEEEECCCCCCEEEEEECCHHHHHHHHHHHHHHHCCCCCEEEEECCC-CEEEEEC

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:

```cpp
#include <iostream>

#include <core/algorithms/predicates.hh>
#include <core/data/io/Pdb.hh>
#include <core/data/io/fasta_io.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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
CEEEEEECCCCCCEEEEEECCHHHHHHHHHHHHHHHHCCCCCEEEEECCCCEEEEEC

)";

/** @brief 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.
 * User can select a chain by providing its code as the second argument of the program
 * USAGE:
 *     ap_pdb_to_fasta_ss 5edw.pdb A
 *
 * CATEGORIES: core::data::io::Pdb; core::algorithms::Not;␣
↪core::data::sequence::SecondaryStructure
 * KEYWORDS:   PDB input; FASTA output; secondary structure; predicates
 * GROUP:      File processing; Format conversion
 */
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 and create_fasta_string lives there
  using namespace core::data::structural; // Chain and

  Pdb reader(argv[1],is_not_alternative,core::data::io::keep_all,true);
  Structure_SP strctr = reader.create_structure(0);
```

(continues on next page)

(continued from previous page)

```

  // Iterate over all chains
  for (auto it_chain = strctr->begin(); it_chain!=strctr->end(); ++it_chain) {
    Chain & c = **it_chain; // --- dereference iterator for easier access
    if ((argc > 2) && ((*it_chain)->id() != argv[2])) continue;

    // --- The line below uses STL algorithm with BioShell predicate to remove all the residues lacking c-alpha
    c.erase(std::remove_if(c.begin(), c.end(), core::algorithms::Not<selectors::ResidueHasCA>(selectors::ResidueHasCA())), c.end());

    if(c.size()>0) {
      // --- Create a sequence object (including secondary structure information)
      core::data::sequence::SecondaryStructure_SP s = (*it_chain)->create_sequence();
      // --- Write sequence as FASTA
      std::cout << create_fasta_string(*s) << "\n";
      // --- Write secondary structure as FASTA
      std::cout << create_fasta_secondary_string(*s) << "\n";
    }
  }
}
```

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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/data/io/pir_io.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Reads a PDB file and writes  protein sequence(s) in PIR format
USAGE:
    ap_pdb_to_pir 5edw.pdb

)";

/** @brief Reads a PDB file and writes  protein sequence(s) in PIR format
 *
 * CATEGORIES: core::data::io::PirEntry
 * KEYWORDS:   PDB input; PIR
 * GROUP:      File processing;Format conversion
 */
int main(const int argc, const char* argv[]) {

```

(continues on next page)

(continued from previous page)

```
  if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  using namespace core::data::io;
  using namespace core::data::sequence;

  core::data::io::Pdb reader(argv[1],is_not_alternative, only_ss_from_header, true);
  core::data::structural::Structure_SP strctr = reader.create_structure(0);

  // Iterate over all chains
  for (auto it_chain = strctr->begin(); it_chain!=strctr->end(); ++it_chain) {
    PirEntry e("",(*it_chain)->create_sequence()->sequence);
    e.type(PirEntryType::STRUCTURE_X);
    e.code(strctr->code());
    e.first_residue_id((*it_chain)->front()->id());
    e.first_chain_id((*it_chain)->char_id());
    e.last_residue_id((*it_chain)->back()->id());
    e.last_chain_id((*it_chain)->char_id());

    std::cout << e<<"\n";
  }
}
```

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

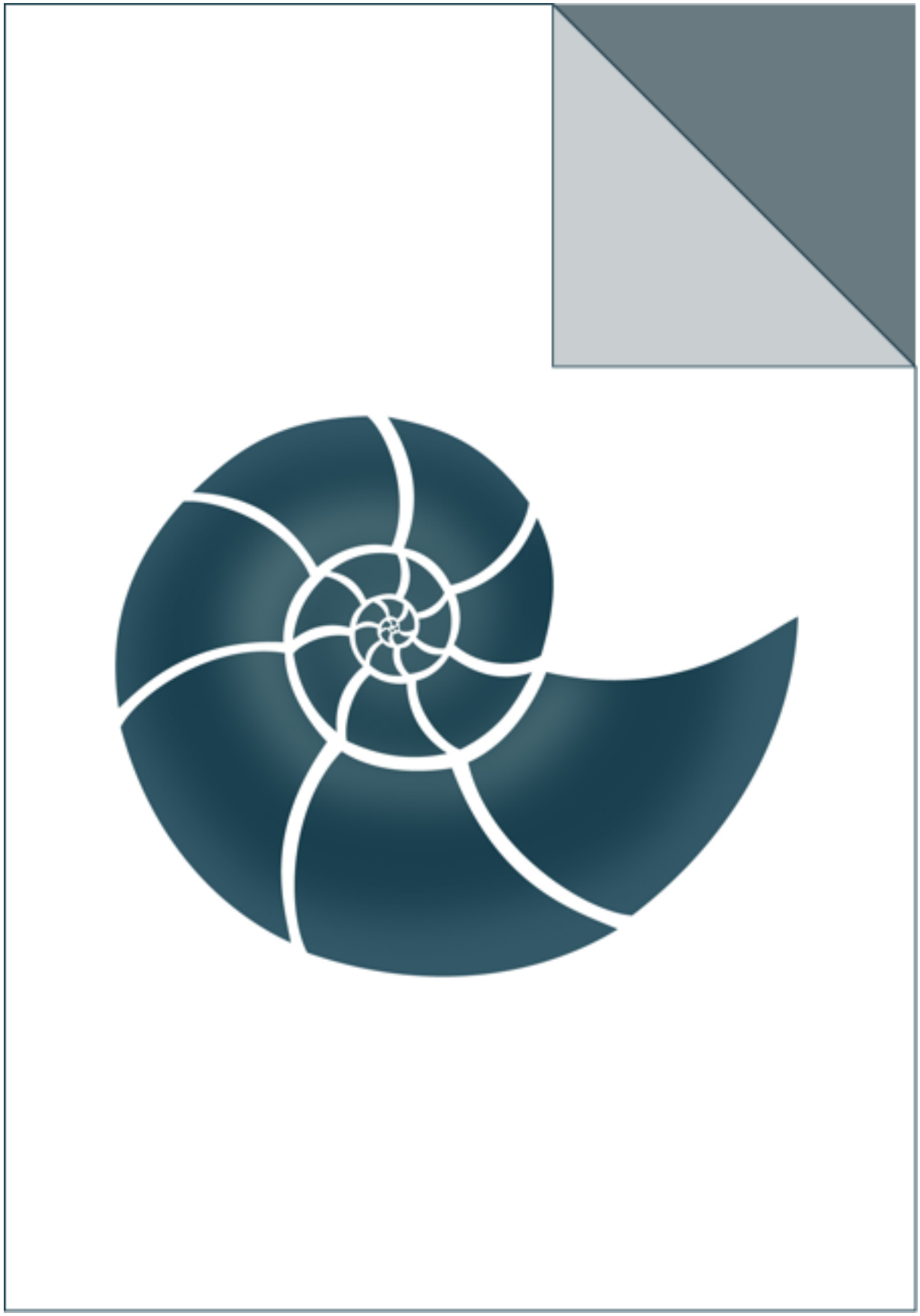
Program source:

```
#include <iostream>

#include <core/data/io/pir_io.hh>
#include <core/data/io/fasta_io.hh>

#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief  Reads a file with sequences in PIR format and converts them to FASTA.
 *
```

(continues on next page)

(continued from previous page)

```
 * CATEGORIES: core/data/io/pir_io;
 * KEYWORDS:   PIR; FASTA output
 * GROUP:      File processing;Format conversion
 */
int main(const int argc, const char* argv[]) {

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

  using core::data::sequence::Sequence_SP; // --- Sequence_SP is just a std::shared_ptr to core::data::sequence::Sequence type
  using namespace core::data::io;          // --- This is required so PirEntry can be printed with << operator

  // --- Create a container where the sequences will be stored
  std::vector<Sequence_SP> sequences;

  // --- Read a file with PIR sequences
  core::data::io::read_pir_file(argv[1], sequences);

  // --- Write them in FASTA
  for (const Sequence_SP s : sequences)
    std::cout << core::data::io::create_fasta_string(*s) << "\n";

  // --- The sequence data is actually in FASTA format; just upper-casted to Sequence_SP
  // --- Here we down-cast it back to the derived type
  std::cout << "The source PIR data was:\n";
  for (const Sequence_SP s : sequences)
    std::cout << *std::dynamic_pointer_cast<core::data::sequence::PirEntry>(s) << "\n";
}
```



ap_reorder_profile_columns

Reads a sequence profile (ASN.1 file format) and shuffles profile's columns as requested. Resulting profile is writen 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_reorder_profile_columns input.asn1 [column-order]
```

EXAMPLE:

```
./ap_reorder_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:

```
#include <iostream>

#include <core/chemical/Monomer.hh>
#include <core/data/sequence/SequenceProfile.hh>
#include <utils/exit.hh>
#include <utils/LogManager.hh>
#include <utils/io_utils.hh>

std::string program_info = R"(

Reads a sequence profile (ASN.1 file format) and shuffles profile's columns as␣
↪requested.
Resulting profile is writen 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
```

(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 d1or4A_.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␣
   ↪missing program parameter
34
35   using namespace core::data::io;
36   using namespace core::data::sequence;
37   utils::Logger logs("ap_reorder_profile_columns");
38   std::string order_string = (argc>2) ? argv[2] : nice_order;
39   logs << utils::LogLevel::INFO << "new aa order is: " << order_string << "\n";
40   SequenceProfile_SP profile_in = core::data::sequence::read_ASN1_checkpoint(argv[1]);
41   SequenceProfile_SP profile_out = profile_in->create_reordered(order_string);
42   profile_out->write_table(std::cout);
43 }
```

ap_rescore_alignment

Reads sequence alignment(s) in the FASTA format and recalculates scores. The input file may contained 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:

```
#include <iostream>
#include <utils/exit.hh>

#include <core/data/io/fasta_io.hh>
#include <core/alignment/PairwiseSequenceAlignment.hh>
#include <core/alignment/on_alignment_computations.hh>
#include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>

using namespace core::data::io;
using namespace core::alignment::scoring;
using namespace core::data::sequence;

std::string program_info = R"(

Reads sequence alignment(s) in the FASTA format and recalculates scores. The input␣
↪file
```

(continues on next page)

(continued from previous page)

```
may contained 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

)";

/** @brief Estimates pairwise sequence similarity for a set of sequences given in a FASTA format
 *
 * CATEGORIES: core/alignment/on_alignment_computations.cc;
 * KEYWORDS:   sequence alignment; FASTA input; sequence alignment score
 * GROUP:      Alignments
 */
int main(const int argc, const char *argv[]) {

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

  // --- read input database fasta file
  std::vector<Sequence_SP> ali_fasta;  // --- stores sequences (should be already aligned)
  core::data::io::read_fasta_file(argv[1], ali_fasta);

  std::string matrix_name = (argc>2) ? argv[2] : "BLOSUM62";
  short gap_open = (argc > 3) ? atoi(argv[3]) : -10;
  short gap_extend = (argc > 4) ? atoi(argv[4]) : -1;
  NcbiSimilarityMatrix_SP sim_m = NcbiSimilarityMatrixFactory::get().get_matrix(matrix_name);
  // --- prints both fasta sequences names and their recalculated score
  for (size_t i = 1; i < ali_fasta.size(); ++i)
    for (size_t j = 0; j < i; ++j)
      std::cout << std::setw(10) << ali_fasta[i]->header().substr(0, 10) << " "
                << std::setw(10) << ali_fasta[j]->header().substr(0, 10)
                << " " << core::alignment::calculate_score(*ali_fasta[i], *ali_fasta[j], *sim_m, gap_open, gap_extend)
                << "\n";
}
```

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:

```
#include <iostream>
#include <algorithm>
#include <set>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/calc/structural/sasa.hh>
```

(continues on next page)

(continued from previous page)

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

```
#include <iostream>

#include <core/data/io/scorefile_io.hh>
#include <utils/exit.hh>
#include <utils/Logger.hh>

std::string program_info = R"(

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

)";

/** @brief Reads a score-file or a silent file (produced by Rosetta) and extracts requested columns of scores
 *
 * CATEGORIES: core::data::io::scorefile_io
 * KEYWORDS:   Rosetta scorefile;
```

(continues on next page)

(continued from previous page)

```
 * GROUP:      File processing;Data filtering
 */
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;

  utils::Logger logs("ap_scorefile_columns");

  std::shared_ptr<NamedDataTable> fsc = read_scorefile(argv[1]);
  std::vector<core::index1> columns;
  if (argc > 2) {
    for (core::index1 i = 2; i < argc; ++i)
      if (fsc->has_column(argv[i]))
        columns.push_back(fsc->column_index(argv[i]));
      else
        logs << utils::LogLevel::WARNING << "Unknown column ID: " << argv[i] << "\n";
  } else {
    columns.push_back(fsc->column_index("score"));
    columns.push_back(fsc->column_index("rms"));
  }
  std::vector<std::string> tags;
  std::cout << "#";
  for (core::index1 icol : columns) std::cout << " "<< fsc->column_name(icol) ;
  std::cout << "\n";
  for (const auto &row: *fsc) {
    for (core::index1 icol : columns) std::cout << row[icol] << " ";
    std::cout << "\n";
  }
}
```

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:

```
#include <iostream>
#include <map>

#include <core/data/io/Pdb.hh>
#include <core/calc/structural/transformations/Crmsd.hh>
#include <utils/exit.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/protocols/PairwiseLigandCrmsd.hh>

std::string program_info = R"(

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
```

(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␣
   ↪for which crmsd will be evaluated
44 and 00199.pdb and the two other files are conformation after docking. In the second␣
   ↪and third examples,
45 B is the ID of the chain containing a ligand peptide.
46
47 )";
48
49 using namespace core::data::structural;
50
51 /** @brief ap_peptide_docking_crmsd calculates crmsd of a peptide that is bound to a␣
   ↪receptor
52  *
53  * CATEGORIES: core::protocols::PairwiseLigandCrmsd
54  * KEYWORDS: PDB input; crmsd; docking; structure selectors
55  * GROUP: Structure calculations; Docking;
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, [](void*) {}
      ) );
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_superipose_pdb_by_ligand 4rm4A.pdb  model.pdb
```

Keywords:

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

Categories:

- core/calc/structural/transformations/PairwiseCrmsd

Program source:

```
#include <memory>
#include <iostream>
#include <vector>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/protocols/PairwiseCrmsd.hh>
#include <utils/LogManager.hh>

using namespace core::data::structural;

std::string program_info = R"(

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:
```

(continues on next page)

(continued from previous page)

```
    ./ap_superipose_pdb_by_ligand 4rm4A.pdb  model.pdb
)";

/** @brief Superimposes protein structures by matching ligand molecules.
 * *
 * CATEGORIES: core/calc/structural/transformations/PairwiseCrmsd
 * KEYWORDS:   PDB input; rototranslation; superimposition; crmsd; docking
 * GROUP: Structure calculations;
 */
int main(const int argc, const char *argv[]) {

  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  utils::LogManager::get().FINE();
  selectors::AtomSelector_SP selector = std::make_shared<selectors::IsCA>(); // --- select a ligand residue by its 3-letter code

  core::data::io::Pdb read_native(argv[1], core::data::io::keep_all); // --- Read the native (reference) structure, keep all atoms
  Structure_SP native = read_native.create_structure(0);
  std::vector<Structure_SP> models; // --- Container for targets to be superimposed
  for (int i = 2; i < argc; ++i) {
    core::data::io::Pdb reader(argv[i], core::data::io::keep_all);
    for (int j = 0; j < reader.count_models(); ++j) // --- Read all models from each target PDB file
      models.push_back(reader.create_structure(j));
  }

  selectors::AtomSelector_SP select_all = std::make_shared<selectors::AtomSelector>();
  core::protocols::PairwiseCrmsd rms_calc(models, selector);
  std::shared_ptr<std::ostream> out = std::make_shared<std::ofstream>("out.pdb");
  rms_calc.calculate(native, out);
}
```

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_superipose_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)

```
using namespace core::data::structural;

std::string program_info = R"(

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_superipose_pdb_by_ligand 4rm4A.pdb HEM 5ofqA.pdb
)";

/** @brief Superimposes protein structures by matching ligand molecules.
 * *
 * CATEGORIES: core/calc/structural/transformations/PairwiseCrmsd
 * KEYWORDS:   PDB input; rototranslation; superimposition; crmsd; docking
 * GROUP: Structure calculations;
 */
int main(const int argc, const char *argv[]) {

  if(argc < 4) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  utils::LogManager::get().FINE();
  selectors::AtomSelector_SP selector = std::make_shared<selectors::SelectResidueByName>(argv[2]); // --- select a ligand residue by its 3-letter code

  core::data::io::Pdb read_native(argv[1], core::data::io::keep_all); // --- Read the native (reference) structure, keep all atoms
  Structure_SP native = read_native.create_structure(0);
  std::vector<Structure_SP> models; // --- Container for targets to be superimposed
  for (int i = 3; i < argc; ++i) {
    core::data::io::Pdb reader(argv[i], core::data::io::keep_all);
    for (int j = 0; j < reader.count_models(); ++j) // --- Read all models from each target PDB file
      models.push_back(reader.create_structure(j));
  }

  selectors::AtomSelector_SP select_all = std::make_shared<selectors::AtomSelector>();
  core::protocols::PairwiseCrmsd rms_calc(models, selector);
  std::shared_ptr<std::ostream> out = std::make_shared<std::ofstream>("out.pdb");
  rms_calc.calculate(native, out);
}
```

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:

```
#include <iostream>
#include <core/data/io/Pdb.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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:
```

(continues on next page)

(continued from previous page)

```
    ex_Remark290 5edw.pdb

)";

/** @brief ex_Remark290 demo shows how to access symmetry operators stored in a PDB file header.
 *
 * CATEGORIES: core/data/io/Pdb;
 * KEYWORDS:   PDB input; PDB line filter; Structure
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
  for (int i = 1; i < argc; ++i) {
    core::data::io::Pdb reader(argv[i], // file name (PDB format, may be gzip-ped)
      core::data::io::is_bb,            // a predicate to read only the ATOM lines corresponding to backbone atoms
      core::data::io::keep_all,         // keep all header lines
      true);                            // parse PDB header

    std::shared_ptr<core::data::io::Remark290> r290 = reader.symmetry_operators();
      std::shared_ptr<core::data::io::Remark350> r350 = reader.biomolecule_symmetry();

    std::cout << "# Symmetry operators found: " << r290->count_operators() << "\n";
    for (const auto &rt: *r290) {
      std::cout << rt << "\n";
    }
    std::cout << "# Biological symmetry biomolecules found: " << r350->count_biomolecules() << "\n";
      for (const auto &sym: *r350) {
          std::cout << "For chains: ";
          for (auto c: sym.first) std::cout<< c<<" ";
          std::cout << "with size "<< sym.second.size()<<"\n";
          for (auto rt: sym.second) std::cout<< rt<<" ";
          std::cout <<"\n";
      }

  }

}
```

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

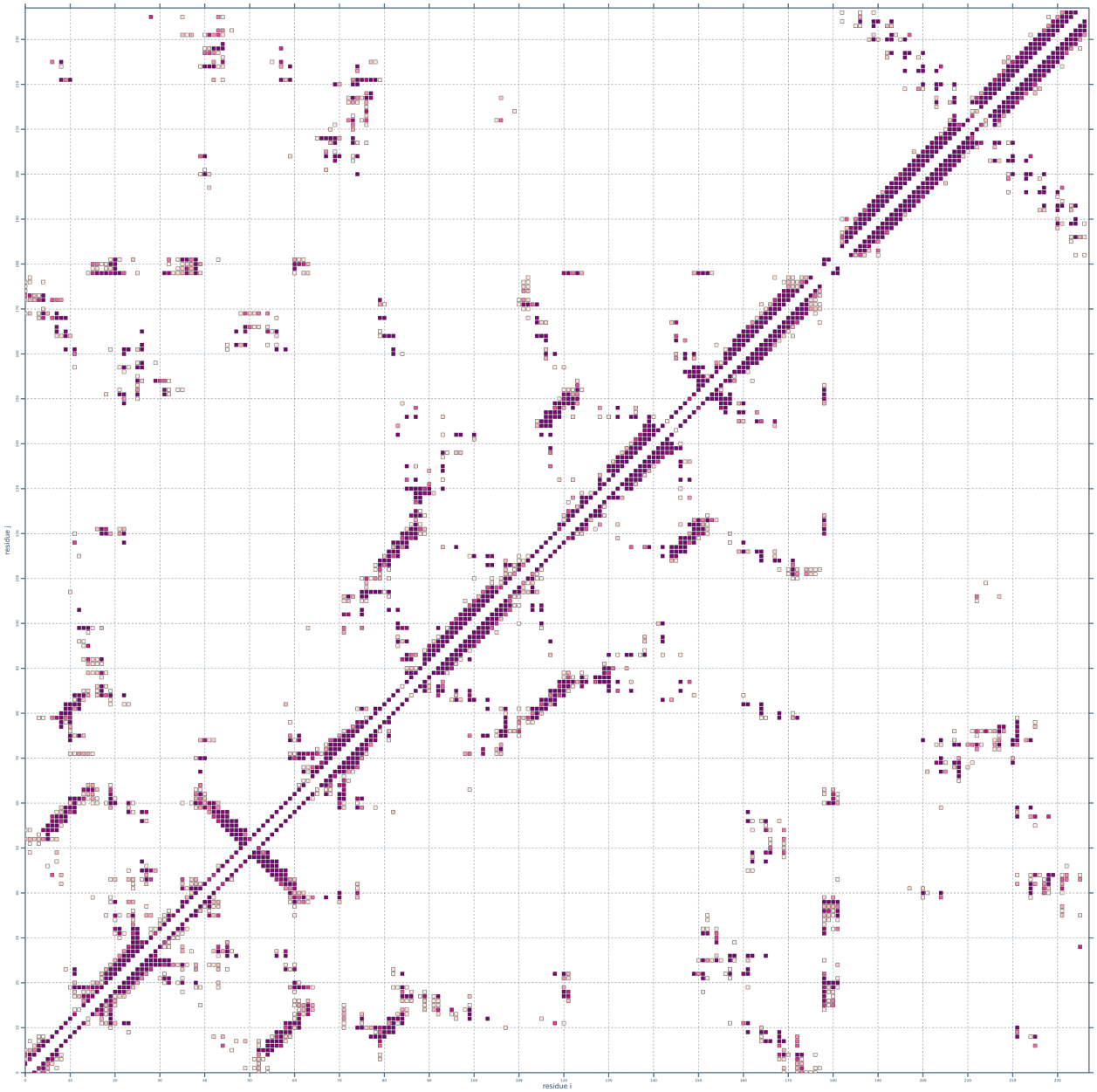
Program source:

```
import sys

from pybioshell.core.data.io import Pdb
from pybioshell.core.calc.structural import evaluate_phi, evaluate_psi, ContactMap
import time

if len(sys.argv) < 3 :
  print("""

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


```

(continues on next page)

(continued from previous page)

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


EXAMPLE:
    python3 contact_map.py 2kwi.pdb 4.5


CATEGORIES: core/calc/structural/ContactMap
KEYWORDS:   PDB input; contact map
GROUP: Structure calculations;
IMG: ap_contact_map.png

  """)
  sys.exit()

pdb = Pdb(sys.argv[1],"")
cutoff = float(sys.argv[2])
contact_map = ContactMap(pdb.create_structure(0), cutoff)
for i_model in range(1,pdb.count_models()) :
  strctr = pdb.create_structure(i_model)
  contact_map.add(strctr)
  print("model",i_model,"added", file=sys.stderr)

res_max = contact_map.max_row_index()
print("  i  ci res_i    j  cj resj n_cont")
for i in range(res_max) :
# --- Get residue index for residue indexed by i; residue index is a structure␣
↪holding chain ID, residue ID and an insertion code
  ri = contact_map.residue_index(i)
  for j in range(i-1) :
    rj = contact_map.residue_index(j)
    if contact_map.has_element(i,j) :
      print("%4d  %c %4d%c  %4d  %c %4d%c  %4d" %(i,ri.chain_id, ri.residue_id, ri.i_
↪code,
        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:

```
import sys, math

from pybioshell.core.data.io import Pdb
from pybioshell.core.data.basic import Vec3
from pybioshell.std import vector_core_data_basic_Vec3

from pybioshell.core.calc.structural.transformations import *
from pybioshell.utils import LogManager

LogManager.INFO()

if len(sys.argv) < 2:
    print("""

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

```

(continues on next page)

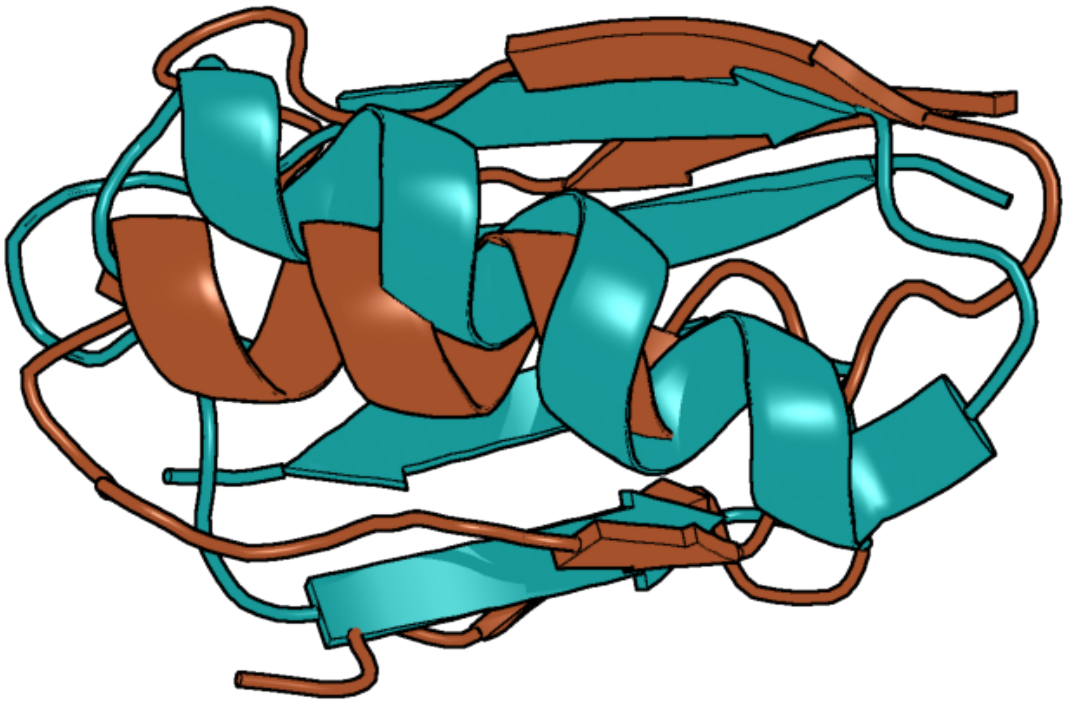
(continued from previous page)

```
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␣
    ↪a single PDB file
40
41      pdb = Pdb(sys.argv[1], "is_ca", False)
42      n_atoms = pdb.count_atoms(0)
43
44      structure = pdb.create_structure(0)
45      models = []
46
47      for i_model in range(0, pdb.count_models()):
48          xyz = vector_core_data_basic_Vec3()
49          for i in range(n_atoms): xyz.append(Vec3())
50          models.append(xyz)
51
52      for i_model in range(0, pdb.count_models()):
53          pdb.fill_structure(i_model, models[i_model])
54          for j in range(i_model):
55              try:
56                  print("%2d %2d %6.3f" % (i_model, j, rms.crmsd(models[i_model],␣
    ↪models[j], len(models[j]))))
57              except:
58                  sys.stderr.write(str(sys.exc_info()[0]) + " " + str(sys.exc_
    ↪info()[1]))
59
60  else:  # --- The case when two or more PDB files are given, calculates first vs. the␣
    ↪rest
61
62      pdb = Pdb(sys.argv[1], "is_ca", False)
63      n_atoms = pdb.count_atoms(0)
64      structure = pdb.create_structure(0)
65
66      xyz = vector_core_data_basic_Vec3()
67      for i in range(n_atoms): xyz.append(Vec3())
68      pdb.fill_structure(0, xyz)
69
70      for pdb_fname in sys.argv[2:]:
71          other_pdb = Pdb(pdb_fname, "is_ca", False)
72          other_structure = other_pdb.create_structure(0)
73          if n_atoms != other_pdb.count_atoms(0):
74              print("The two structures have different number of CA atoms!\n")
75          other_xyz = vector_core_data_basic_Vec3()
76          for i in range(n_atoms): other_xyz.append(Vec3())
77          other_pdb.fill_structure(0, other_xyz)
78          try:
79              print("%s: %6.3f" % (pdb_fname.split("/")[-1].split(".")[0], rms.
    ↪crmsd(xyz, other_xyz, n_atoms)))
```

(continues on next page)

(continued from previous page)

```
        except:
            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 python3 hist_from_scorefile.py -h 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:

```
import sys,argparse

from pybioshell.core.data.io import find_pdb, read_scorefile

from os.path import expanduser, join
home_dir = expanduser("~")
# It is assumed  VisuaLife library is installed in your $HOME/src.git/visualife/src/
↪directory
sys.path.append(join(home_dir, "src.git/visualife/src/"))

from core.Plot import Plot
from core.SvgViewport import SvgViewport
from core.styles import ColorRGB, color_by_name

from pybioshell.core.calc.statistics import HistogramDD

if len(sys.argv) < 2 :
    print("""

```

(continues on next page)

(continued from previous page)

```
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

Call python3 hist_from_scorefile.py -h for full help


CATEGORIES: core/calc/statistics/HistogramDD; core/data/io/read_scorefile
KEYWORDS: Rosetta scorefile; histogram; energy plot
GROUP: Statistics;
IMG: rms_hist.svg

  """)
    sys.exit()

# ----------argument parsing
parser = argparse.ArgumentParser(description='Reads Rosetta scorefiles and plot histogram of given column name (default is rms) for all files in one plot')

parser.add_argument('-f', '--file', help="input .fsc file", nargs='+', required=True)
parser.add_argument('-c', '--column', help="column name for histogram", nargs=1, required=False, default=["rms"])
parser.add_argument('-min', '--min', help="minimum data value", nargs=1, required=False)
parser.add_argument('-max', '--max', help="maximum data value", nargs=1, required=False)
parser.add_argument('-b', '--bin_width', help="bin width", nargs=1, required=False, default=[1.0])

parser = parser.parse_args()

xmin = float(parser.min[0]) if parser.min else min(alldata)
xmax = float(parser.max[0]) if parser.max else max(alldata)
step = float(parser.bin_width[0])

data = []
alldata = []

# ----------filling lists with data from file
for i in range(len(parser.file)):
  p = read_scorefile(parser.file[i])
  indx = p.column_index(parser.column[0])
  print("#Making histogram for ",parser.column[0]," column at index ",indx)
  lhist = []
  for row in p:
    lhist.append(float(row[indx]))
  data.append(lhist)
  alldata.extend(lhist)

# -----------ploting-----------
drawing = SvgViewport("%s_hist.svg"%(parser.column[0]), 0, 0, 800, 650,color="white")
pl = Plot(drawing,100,700,100,550,xmin, xmax,0,0.5,axes_definition="UBLR")

stroke_color = color_by_name("SteelBlue").create_darker(0.3)
```

(continues on next page)

(continued from previous page)

```
pl.axes["B"].label = parser.column[0]

for key,ax in pl.axes.items() :
  ax.fill, ax.stroke, ax.stroke_width = stroke_color, stroke_color, 2.0
  ax.tics(0,5)
pl.axes["U"].tics(5,0)
pl.axes["R"].tics(5,0)
pl.draw_axes()
pl.plot_label = "%s histogram" %(parser.column[0])
pl.draw_plot_label()


box_width = 0.9 * step if len(data) == 1 else step/(len(data)+1.0)
for i in range(len(data)):
  # ------------------ here we actually make a histogram -----------------
  h = HistogramDD(step,xmin,xmax)
  for j in data[i]: h.insert(j)
  h.normalize()
  x_data = []
  y_data = []
  for b in range(h.count_bins()):
    x_data.append(h.bin_min_val(b)+box_width*i)
    y_data.append(h.get_bin(b))

  print(h)

  pl.bars(x_data, y_data, width=step, title="hist%s"%(i))

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

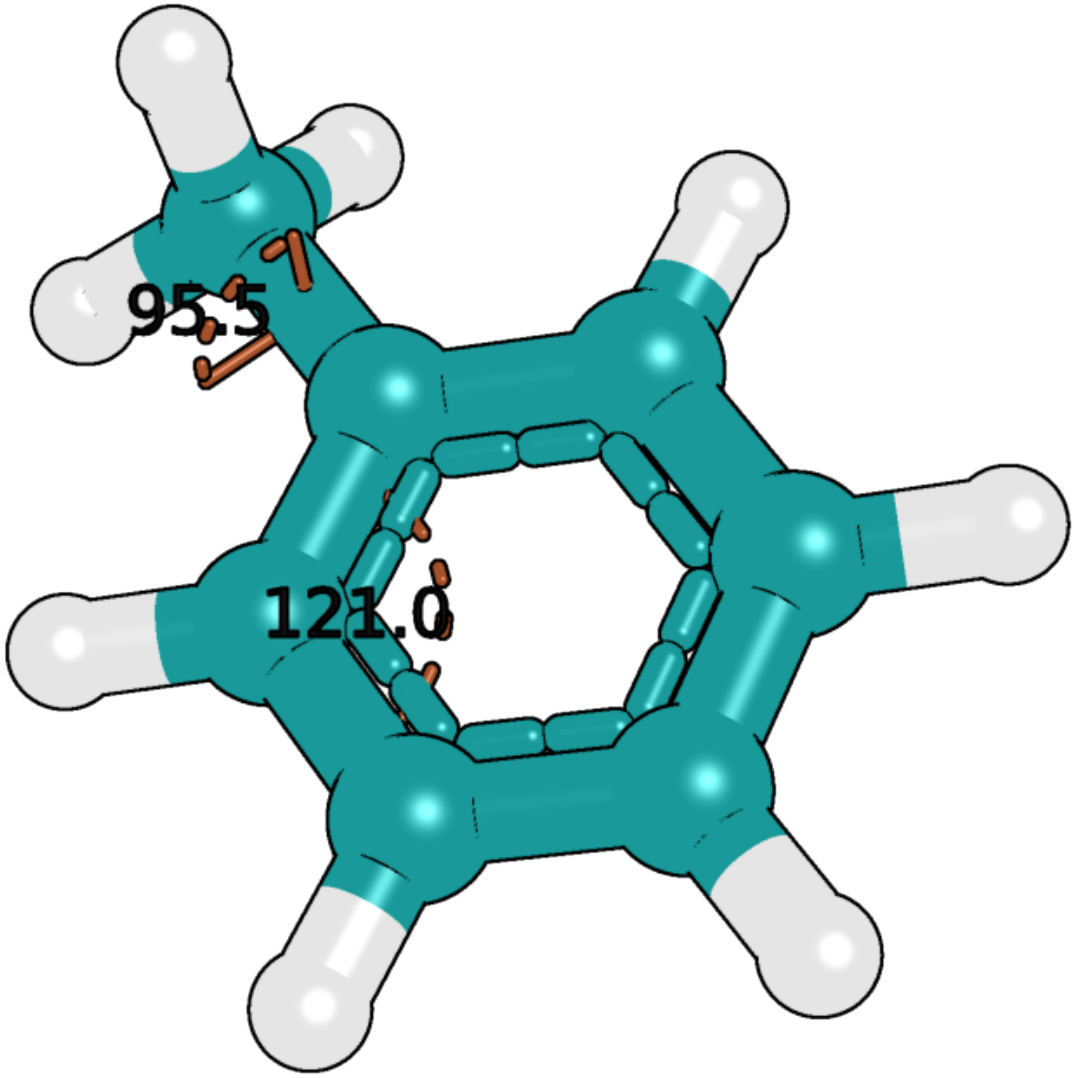
Program source:

```
import sys

from pybioshell.core.data.io import Pdb
from pybioshell.core.calc.structural import evaluate_phi, evaluate_psi

if len(sys.argv) < 2 :
  print("""

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


CATEGORIES: core/calc/structural/evaluate_phi; core/calc/structural/evaluate_psi
KEYWORDS:   PDB input; structural properties
GROUP: Structure calculations
IMG: Toluen_dihedral_flat_angle.png

  """)
  sys.exit()

factor = 180.0/3.14159
structure = Pdb(sys.argv[1],"",False).create_structure(0)
for code in structure.chain_codes() :
  chain = structure.get_chain(code)
  n_res = chain.count_aa_residues()
  for i_res in range(1,n_res-1) :
    try :
      r = chain[i_res]
      r_prev = chain[i_res-1]
      r_next = chain[i_res+1]
      phi = evaluate_phi(r_prev,r)
      psi = evaluate_psi(r,r_next)
      print("%d %s %c %7.2f %7.2f" % (r.id(), r.residue_type().code3, r.owner().id(),
↪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:

```
import sys,argparse

from pybioshell.core.data.io import find_pdb, read_scorefile

from os.path import expanduser, join
home_dir = expanduser("~")
# It is assumed  VisuaLife library is installed in your $HOME/src.git/visualife/src/␣
↪directory
sys.path.append(join(home_dir, "src.git/visualife/src/"))

from core.Plot import Plot
from core.SvgViewport import SvgViewport
from core.styles import ColorRGB, color_by_name

from pybioshell.core.calc.statistics import HistogramD4

if len(sys.argv) < 2 :
    print("""

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]
```

(continues on next page)

(continued from previous page)

```
24
25
26  EXAMPLE:
27      python3 score_rms_plot.py -f 1pgx-abinitio.fsc 1pgx-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␣
     ↪prepares score to rms plot for all given files as a series in one picture')
39
40  parser.add_argument('-f', '--file', help="input .fsc file", nargs='+', required=True)
41  parser.add_argument('-x', '--x_range', help="range for X axis of a plot (two values:␣
     ↪min max)", nargs=2, required=False)
42  parser.add_argument('-y', '--y_range', help="range for X axis of a plot (two values:␣
     ↪min max)", nargs=2, required=False)
43
44  parser = parser.parse_args()
45
46
47  energy = []
48  rms = []
49  allenergy = []
50  allrms = []
51  s=""
52
53  #----------filling lists with data from file
54  print("Plotting score vs. rms for:",end="")
55  for i in range(len(parser.file)):
56    s+=" "+parser.file[i]
57
58    p = read_scorefile(parser.file[i])
59
60    en = p.column_index("score")
61    xrms = p.column_index("rms")
62
63    e = []
64    r = []
65
66    for row in p:
67      e.append(float(row[en]))
68      r.append(float(row[xrms]))
69
70    energy.append(e)
71    rms.append(r)
72    allenergy.extend(e)
73    allrms.extend(r)
74
75  #----------plotting energy plot
76
77  xfrom = float(parser.x_range[0]) if parser.x_range else min(allrms)-1
```

(continues on next page)

(continued from previous page)

```
xto = float(parser.x_range[1]) if parser.x_range else max(allrms)+1
yfrom = float(parser.y_range[0]) if parser.y_range else min(allenergy)-10
yto = float(parser.y_range[1]) if parser.y_range else max(allenergy)+10


drawing = SvgViewport("outputs_from_test/score_to_rms.svg", 0, 0, 800, 650,color=
↪"white")
pl = Plot(drawing,100,700,100,600,xfrom,xto,yfrom,yto,axes_definition="UBLR")

stroke_color = color_by_name("SteelBlue").create_darker(0.3)
pl.axes["B"].label = "rmsd"
pl.axes["L"].label = "score"

for key,ax in pl.axes.items() :
  ax.fill, ax.stroke, ax.stroke_width = stroke_color, stroke_color, 2.0
  ax.tics(0,5)
pl.axes["U"].tics(5,0)
pl.axes["R"].tics(5,0)
pl.draw_axes()
pl.plot_label = "score_to_rms"
pl.draw_plot_label()
print(s)
for i in range(len(energy)):
  pl.scatter(rms[i],energy[i], markersize=2, markerstyle='c', title="serie-%s"%(i))

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

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

Keywords:

- *FASTA input*
- *sequence alignment*

Categories:

- core/protocols/PairwiseSequenceIdentityProtocol

Input files:

- cyped.CYP109.fasta

Output files:

- stdout.out

Program source:

```
import sys

from pybioshell.std import vector_std_shared_ptr_core_data_sequence_Sequence_t
from pybioshell.core.data.io import read_fasta_file
from pybioshell.core.alignment import AlignmentType
from pybioshell.core.protocols import PairwiseSequenceIdentityProtocol


if len(sys.argv) < 3 :
  print("""

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

    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


CATEGORIES: core/protocols/PairwiseSequenceIdentityProtocol
KEYWORDS:   FASTA input; sequence alignment
GROUP:      Alignments
```

(continues on next page)

(continued from previous page)

```
IMG:     ap_aligned-1k6m-1bif.png
  """)

  sys.exit()

cutoff = float(sys.argv[2])
sequences = vector_std_shared_ptr_core_data_sequence_Sequence_t()
read_fasta_file(sys.argv[1], sequences, True)
align_protocol = PairwiseSequenceIdentityProtocol()
n_seq = align_protocol.add_input_sequences(sequences)
align_protocol.n_threads(4).alignment_method(AlignmentType.SEMIGLOBAL_ALIGNMENT)
align_protocol.run()

for i in range(1,n_seq) :
  for j in range(i) :
    seq_id = align_protocol.get_sequence_identity(i,j)
    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:

```
import sys, math,copy

from pybioshell.core.data.io import find_pdb
from pybioshell.core.data.basic import Vec3Cubic

from pybioshell.std import vector_core_data_basic_Vec3Cubic

if len(sys.argv) < 3 :
    print("""

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


CATEGORIES: core/data/basic/Vec3Cubic
KEYWORDS:   PDB input; PBC; SURPASS; Vec3Cubic
IMG:        unwrapped.gif

  """)
    sys.exit()


```

(continues on next page)

(continued from previous page)

```
pdb = find_pdb(sys.argv[1], "./")
n_atoms = pdb.count_atoms(0)
cutoff = float(sys.argv[2])

Vec3Cubic.set_box_len(cutoff)
xyz = vector_core_data_basic_Vec3Cubic()
for i in range(n_atoms): xyz.append(Vec3Cubic())

for i_model in range(0, pdb.count_models()) :
  xyz = vector_core_data_basic_Vec3Cubic()
  for i in range(n_atoms): xyz.append(Vec3Cubic())
  pdb.fill_structure(i_model, xyz)
  structure = pdb.create_structure(i_model)
  n_res = 0
  print("MODEL          ",i_model+1 )
  for ia in range(structure.count_chains()):
    chain = structure[ia]
    #wrapping first atom of every chain to the first box
    if xyz[n_res ].x > cutoff: xyz[n_res ].x -= cutoff
    if xyz[n_res ].x < 0: xyz[n_res ].x += cutoff
    if xyz[n_res ].y > cutoff: xyz[n_res ].y -= cutoff
    if xyz[n_res ].y < 0: xyz[n_res ].y += cutoff
    if xyz[n_res ].z > cutoff: xyz[n_res ].z -= cutoff
    if xyz[n_res ].z < 0: xyz[n_res ].z += cutoff
    chain[0][0].set(xyz[n_res])
    #calculating unwraped coordinates
    for ir in range(chain.count_residues()-1):
      ax = xyz[n_res+ir+1].closest_delta_x(xyz[n_res+ir])
      ay = xyz[n_res+ir+1].closest_delta_y(xyz[n_res+ir])
      az = xyz[n_res+ir+1].closest_delta_z(xyz[n_res+ir])
      xyz[n_res+ir+1].x = xyz[n_res+ir].x + ax
      xyz[n_res+ir+1].y = xyz[n_res+ir].y + ay
      xyz[n_res+ir+1].z = xyz[n_res+ir].z + az

      chain[ir+1][0].set(xyz[n_res+ir+1])
    n_res+=ir+2
    #writing to PDB file
    for ir in range(chain.count_residues()) :
      resid = chain[ir]
      print(resid[0].to_pdb_line())
  print("ENDMDL")

```

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:

```
import sys

from pybioshell.core.alignment import SequenceNWAligner, SequenceSWAligner
from pybioshell.core.data.io import read_fasta_file

if len(sys.argv) < 3 :
  print("""

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


CATEGORIES: core/alignment/SequenceNWAligner
KEYWORDS:   alignment
GROUP: Sequence calculations

  """)
  sys.exit()
q = read_fasta_file(sys.argv[1])[0]
t = read_fasta_file(sys.argv[2])[0]
# ---------- calculate a global alignment
aligner = SequenceNWAligner(max(q.length(), t.length()))
```

(continues on next page)

(continued from previous page)

```
# ---------- calculate a local alignment
#aligner = SequenceSWAligner(max(q.length(), t.length()))
if len(sys.argv) < 4 :
    score = aligner.align(q, t, -10, -1, "BLOSUM62")
else:
    score = aligner.align(q, t, int(sys.argv[3]), int(sys.argv[4]), "BLOSUM62")

alignment = aligner.backtrace_sequence_alignment()
print("# score:", score)
print("> query\n" + alignment.get_aligned_query())
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:

```
import sys

from pybioshell.core.data.sequence import read_ASN1_checkpoint

if len(sys.argv) < 2 :
  print("""

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


CATEGORIES: core/data/sequence/read_ASN1_checkpoint
```

(continues on next page)

(continued from previous page)

```
KEYWORDS:   sequence profile; Format conversion
GROUP:      File processing; Format conversion

  """)
  sys.exit()

profile = read_ASN1_checkpoint(sys.argv[1])
profile.write_table_header()
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:

```
import sys

from pybioshell.core.data.io import Pdb
from pybioshell.core.calc.structural import ProteinArchitecture


if len(sys.argv) < 2 :
    print("""

Shows how to use BetaStructuresGraph class from Python


USAGE:
    python3 betastructures_graph.py input.pdb


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

(continues on next page)

(continued from previous page)

```

CATEGORIES: core/calc/structural/ProteinArchitecture
KEYWORDS:   PDB input; graphs

  """)
    sys.exit()

pdb_fname = sys.argv[1]
structure = Pdb(pdb_fname, "").create_structure(0)
architecture = ProteinArchitecture(structure)
beta_graph = architecture.create_strand_graph()
strands = beta_graph.get_strands_copy()

print("    ",end ="")
for i in range(len(strands)) :  print(" S%2d " % (i),end ="")
print()

for i in range(len(strands)) :
  pairings = []
  print("S%2d: " % (i),end ="")
  for j in range(len(strands)) :
    are_paired = False
    try :
      p = beta_graph.get_strand_pairing(strands[i],strands[j])
      are_paired = True
    except:  pass
    if are_paired :
      print("  X  ",end ="")
      pairings.append(j)
    else : print("     ",end ="")
  print("| %-45s paired with %d" % (str(strands[i]),pairings[0]),end ="")
  for pi in range(1,len(pairings)) :
    print(", %d" % (pairings[pi]),end ="")
  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:

```
import sys

from pybioshell.core.data.io import Pdb, write_pdb

if len(sys.argv) < 2 :
    print("""

```

(continues on next page)

(continued from previous page)

```
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


CATEGORIES: core/data/io/Pdb
KEYWORDS:   PDB input; structure selectors; PDB output
GROUP: File processing

  """)
    sys.exit()

input_fnames = open(sys.argv[1])
reader = Pdb(input_fnames.readline().strip(),"is_ca",False)
structure = reader.create_structure(0)
out_fname = "out.pdb" if len(sys.argv) == 2 else sys.argv[2]
write_pdb(structure, out_fname, 1)
i_model = 2
print("Reading PDB files: ",end="")
for pdb_fname in input_fnames :
    print(pdb_fname.strip().split("/")[-1],end=" ")
    reader = Pdb(pdb_fname.strip(),"is_ca",False)
    reader.fill_structure(0,structure)
    write_pdb(structure, out_fname, i_model)
    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:

```
import sys

from pybioshell.core.data.io import find_pdb

if len(sys.argv) < 2 :
  print("""

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

    CATEGORIES: core/data/io/find_pdb
    KEYWORDS: PDB input; center protein; internal coordinates
```

(continues on next page)

(continued from previous page)

```
    GROUP: Structure calculations;

  """)
  sys.exit()

which_model = 0 if len(sys.argv) == 2 else (int(sys.argv[2])-1)

pdb_reader = find_pdb(sys.argv[1], "./")
structure = pdb_reader.create_structure(which_model)
cx, cy, cz, n = 0, 0, 0, 0
for ic in range(structure.count_chains()) :
    chain = structure[ic]
    for ir in range(chain.count_residues()) :
      resid = chain[ir]
      for ai in range(resid.count_atoms()) :
        cx += resid[ai].x
        cy += resid[ai].y
        cz += resid[ai].z
        n+=1.0

cx /= n
cy /= n
cz /= n
print("# Center was:",cx,cy,cz)

for ic in range(structure.count_chains()) :
    chain = structure[ic]
    for ir in range(chain.count_residues()) :
      resid = chain[ir]
      for ai in range(resid.count_atoms()) :
        resid[ai].x -= cx
        resid[ai].y -= cy
        resid[ai].z -= cz
        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:

```
import sys

sys.path.append("/Users/dgront/src.git/bioshell/bin")
from pybioshell.core.data.io import Pdb

if len(sys.argv) < 2 :
    print("""

Checks if a given structure has chain breaks


USAGE:
    python3 check_structure.py input.pdb


EXAMPLE:
    python3 check_structure.py 2gb1.pdb

```

(continues on next page)

(continued from previous page)

```

CATEGORIES: core/calc/structural/
KEYWORDS:   PDB input; structural properties
GROUP: Structure calculations

  """)
    sys.exit()

N_gaps = 0
structure = Pdb(sys.argv[1],"",False).create_structure(0)
for code in structure.chain_codes() :
    chain = structure.get_chain(code)
    r_prev, r = chain[0], chain[0]
    prev_ca = r.find_atom(" CA ")
    the_ca = prev_ca
    for ires in range(1,chain.size()):
        r_prev = r
        prev_ca = the_ca
        if not prev_ca:
            continue
        r = chain[ires]
        the_ca = r.find_atom(" CA ")
        if not the_ca:
            continue

        d = the_ca.distance_to(prev_ca)
        if d > 4.0:
            N_gaps += 1
            print("chain %c: too long distance between CA of  %s%d and %s%d residue:
↪%6.3f" %
                  (r.owner().id(), r_prev.residue_type().code3, r_prev.id(), r.
↪residue_type().code3, r.id(), d))

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:

```
import sys

from pybioshell.core.data.io import Cif, write_mol2
from pybioshell.core.chemical import MonomerStructure


if len(sys.argv) < 2 :
    print("""

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]
```

(continues on next page)

(continued from previous page)

```

EXAMPLE:
    python3 cif_to_mol2.py HEM.cif [molecule_name]
    python3 cif_to_mol2.py HEM.cif HAEM

CATEGORIES: core/data/io/write_mol2
KEYWORDS:   CIF input; MOL2 output; Format conversion
GROUP:      File processing; Format conversion
  """)
    sys.exit()

mm = MonomerStructure.from_cif(sys.argv[1])
if len(sys.argv) > 2:                   # --- set molecule name if provided from␣
↪command line
    mm.molecule_name = sys.argv[2]
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:

```
import sys

from pybioshell.std import vector_std_shared_ptr_core_data_sequence_Sequence_t
from pybioshell.core.data.io import read_hssp_file, write_clustalo_file, read_fasta_
↪file, read_clustalw_file

from pybioshell.utils import LogManager

LogManager.FINEST()

if len(sys.argv) < 3 :
    print("""

Converts multiple sequence alignment (MSA) data from one format to another.
Known input formats:  FASTA (.fasta), HSSP (.hssp) and ClustalW/ClustalO (.aln)
```

(continues on next page)

(continued from previous page)

```
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


CATEGORIES: core/data/io/read_hssp_file
KEYWORDS:   MSA;  Format conversion
GROUP: File processing;

  """)
    sys.exit()

msa = vector_std_shared_ptr_core_data_sequence_Sequence_t() # --- vector to hold␣
↪sequences obtained from an input file
extension_in = sys.argv[1].split('.')[-1]                  # --- detect the input␣
↪format
if extension_in == 'aln':
    read_clustalw_file(sys.argv[1], msa)
elif extension_in == 'hssp':
    read_hssp_file(sys.argv[1], msa)
elif extension_in == 'fasta':
    read_fasta_file(sys.argv[1], msa)

f = open(sys.argv[2],"w")
for seq in msa:
    print(">", seq.header(), file=f)
    print(seq.sequence, file=f)
f.close()

```

crmsd_on_ligands.py

Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on all atoms of a ligand conformations. This scripts 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:

```
import sys, math

from pybioshell.core.data.io import Pdb
from pybioshell.std import vector_core_data_basic_Vec3
from pybioshell.core.calc.structural.transformations import *
from pybioshell.utils import LogManager
LogManager.INFO()

if len(sys.argv) < 3 :
    print("""

Calculates cRMSD (coordinate Root-Mean-Square Deviation) value on all atoms of a ligand conformations.

This scripts reads one or more PDB files, extracts all ligands that matches a given
```

(continues on next page)

(continued from previous page)

```
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


CATEGORIES: core/calc/structural/transformations/CrmsdOnVec3
KEYWORDS:   PDB input; ligand; crmsd
GROUP: Structure calculations

  """)
    sys.exit()

rms = CrmsdOnVec3()

pdb_codes = []          # --- PDB code for every input structure: for informative output
structures = []         # --- contains all input structures
residues = []           # --- ligand residue objects (to keep information about residue number and chain)
atoms_by_ligand = []    # --- a list of atoms for every ligand
for pdb_code in sys.argv[2:] :
  pdb = Pdb(pdb_code,"",False)
  structure = pdb.create_structure(0)
  structures.append(structure)
  for ic in range(structure.count_chains()) :
    chain = structure[ic]
    for ir in range(chain.terminal_residue_index() + 1,chain.size()) :
        resid = chain[ir]
        code3 = resid.residue_type().code3
        if code3 != sys.argv[1] : continue # Skip other ligands
        residues.append(resid)
        pdb_codes.append(pdb_code)
        atoms = vector_core_data_basic_Vec3()
        for ia in range(resid.count_atoms()):
            atoms.append(resid[ia])
        atoms_by_ligand.append(atoms)

for i_ligand in range(0, len(atoms_by_ligand)) :
      ir = residues[i_ligand]
      for j_ligand in range(i_ligand):
          jr = residues[j_ligand]
          crmsd_val = rms.crmsd(atoms_by_ligand[i_ligand], atoms_by_ligand[j_ligand], len(atoms_by_ligand[j_ligand]))
          print("%s %4d %3s %c - %s %4d %3s %c : %7.3f" %
            (pdb_codes[i_ligand], ir.id(), ir.residue_type().code3, ir.owner().id(),
            pdb_codes[j_ligand], jr.id(), jr.residue_type().code3, jr.owner().id(), 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:

```
import sys
from random import random, seed

from pybioshell.core.data.io import read_fasta_file, create_fasta_string


if len(sys.argv) < 3 :
  print("""

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
```

(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:

```
import sys, argparse

from pybioshell.core.data.io import read_scorefile

if len(sys.argv) < 2 :
  print("""

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

CATEGORIES: core/data/io/read_scorefile
KEYWORDS: Rosetta scorefile;
GROUP: Statistics;

  """)
  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␣
    ↪it's 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()
46
```

find_rings.py

Reads in a mall 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:

```
import sys

from pybioshell.core.chemical import PdbMolecule
from pybioshell.core.chemical import find_rings

if len(sys.argv) < 2 :
  print("""

Reads in a mall 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:
```

(continues on next page)

(continued from previous page)

```
    python3 find_rings.py 9ZB_ideal.pdb

    CATEGORIES: core/chemical/
    KEYWORDS: PDB input; small molecules
    GROUP: small molecules;

  """)
  sys.exit()

mol = PdbMolecule.from_pdb(sys.argv[1])
rings = find_rings(mol)
for ring in rings:
  print("# ------------")
  for atom in ring:
    print(mol.get_atom(atom).to_pdb_line())

```

hhpred_to_modeller.py

Reads an output file produced by HHPred, that contains alignments between a query protein and template protein structues. 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:

```
import sys

from pybioshell.core.data.io import read_hhpred, create_pir_string


if len(sys.argv) < 2 :
  print("""

    Reads an output file produced by HHPred, that contains alignments between a query protein and template protein structues.
    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)


```

(continues on next page)

(continued from previous page)

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


EXAMPLE:
    python3 hhpred_to_modeller.py CYP51F.hhpred 1 2


CATEGORIES: core/data/io/read_hhpred
KEYWORDS:   HHPred; comparative modelling
GROUP: File processing; Format conversion

  """)
  sys.exit()

alignments = read_hhpred(sys.argv[1])
which_ali  = sys.argv[2:] if len(sys.argv) > 2 else [1]
print(len(alignments),"alignments found in",sys.argv[1])
for i in which_ali :
  i = int(i)
  print("retriving alignment:",i,"as %d.pir" % (i))
  f = open("%d.pir" % (i), "w")
  f.write(create_pir_string(alignments[i-1], 80))
  f.close()
```

ligand_contacts.py

Finds contacts between a ligand molecule and a protein. This scripts 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 detectplausible 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)

```
if len(sys.argv) < 4:
    print("""

Finds contacts between a ligand molecule and a protein.

This scripts 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 detectplausible 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


CATEGORIES: core/data/io/Pdb
KEYWORDS:   PDB input; ligand; structural properties
GROUP: Structure calculations

  """)
    sys.exit()

cutoff = float(sys.argv[2])

first_pdb = 3
extra_acceptors, extra_donors = {}, {}
if sys.argv[3] == '-':
    first_pdb = 5
elif sys.argv[3].endswith(".json"):
    extra_donors = json.loads(open(sys.argv[3]).read())
    first_pdb = 5

if sys.argv[4].endswith(".json"):
    extra_acceptors = json.loads(open(sys.argv[4]).read())

hb_filter = HydrogenBondFilter()
for code3 in extra_acceptors.keys():
    for atom_name in extra_acceptors[code3]:
        hb_filter.add_acceptor_definition(code3, atom_name)
for code3 in extra_donors.keys():
    for atom_name in extra_donors[code3]:
        hb_filter.add_donor_definition(code3, atom_name)

for pdb_code in sys.argv[first_pdb:]:  # --- Iterate over PDB input files
```

(continues on next page)

(continued from previous page)

```
    if len(sys.argv[first_pdb:]) > 1: print("# Pdb file %s" % (pdb_code.split("/")[-1].split(".")[0]))
    pdb = Pdb(pdb_code, "", False)
    print(" ---- ligand ---- | --------- partner -------- | distance")
    print("c  res  id atname |  c  res  id   type  atname | in Angstrom")

    for m in range(pdb.count_models()):  # --- Iterate over all models in the input file
        if pdb.count_models() > 1: print("# Model %d" % (i + 1))
        structure = pdb.create_structure(m)

        for ic in range(structure.count_chains()):
            lig_chain = structure[ic]
            for ir in range(lig_chain.count_residues()):
                ligand = lig_chain[ir]
                code3 = ligand.residue_type().code3

                if code3 != sys.argv[1]: continue  # Skip other ligands
                for iic in range(structure.count_chains()):
                    other_chain = structure[iic]
                    for r in range(other_chain.count_residues()):  # ----Iterate over residues
                        res = other_chain[r]
                        if res == ligand: continue
                        d = res.min_distance(ligand)  # ---- If residue is close enough to ligand
                        if d < cutoff:
                            for ilig in range(ligand.count_atoms()):
                                for ioth in range(res.count_atoms()):
                                    ligand_atom = ligand[ilig]
                                    other_atom = res[ioth]
                                    if ligand_atom.distance_to(other_atom) <= cutoff:
                                        extras = ""
                                        if hb_filter(ligand_atom, other_atom,ligand_atom.distance_to(other_atom)):
                                            extras = "HYDROGEN_BOND"
                                        print("%s  %3s %4d %4s     %s  %3s %4d %6s %4s   %6.3f %s" % (ligand.owner().id(),
                                                                                                     ligand.residue_type().code3, ligand.id(), ligand_atom.atom_name(),
                                                                                                     res.owner().id(), res.residue_type().code3, res.id(),
                                                                                                     monomer_type_name(res.residue_type()), other_atom.atom_name(),
                                                                                                     ligand_atom.distance_to(other_atom), 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 scripts 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 transfrom coordinates a molecule of a drug and to compute crsmd 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:

```
import sys, math

sys.path.append("../../../../../bin/")

from pybioshell.core.data.basic import Vec3
from pybioshell.core.data.io import Pdb
from pybioshell.std import vector_core_data_basic_Vec3
from pybioshell.core.calc.structural.transformations import *
from pybioshell.utils import LogManager

LogManager.INFO()

if len(sys.argv) < 3:
    print("""

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 scripts 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 transfrom coordinates a molecule of a drug and to compute crsmd 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


CATEGORIES: core/calc/structural/transformations/CrmsdOnVec3
KEYWORDS:   PDB input; ligand; crmsd
GROUP: Structure calculations

  """)
    sys.exit()

rms = CrmsdOnVec3()


class Entry:

    def __init__(self, code, structure):

        self.pdb_code = code  # --- PDB code for every input structure: for informative output
        self.structure = structure  # --- contains all input structures
        self.ligand = None
        self.cofactor = None
        self.atoms_superimposed = vector_core_data_basic_Vec3()  # --- a list of atoms to define rototranslation
```

(continues on next page)

(continued from previous page)

```
        self.atoms_crmsd = vector_core_data_basic_Vec3()  # --- a list of atoms to compute crmsd

    def is_OK(self):
        return self.ligand and self.cofactor

    def add_ligand(self,ligand):
        for ia in range(ligand.count_atoms()):
            e.atoms_crmsd.append(ligand[ia])
        self.ligand = ligand

    def add_cofactor(self,cofactor):
        for ia in range(cofactor.count_atoms()):
            e.atoms_superimposed.append(cofactor[ia])
        self.cofactor = cofactor

rototranslation_code3 = sys.argv[1]
crmsd_code3 = sys.argv[2]

entries = []
for pdb_code in sys.argv[3:]:
    pdb = Pdb(pdb_code, "is_not_alternative is_not_water", False)
    structure = pdb.create_structure(0)
    for ic in range(structure.count_chains()):
        chain = structure[ic]
        # start from chain.terminal_residue_index() + 1 if you are sure the PDB file has TER lines
        e = Entry(pdb_code.split("/")[-1], structure)
        for ir in range(0, chain.size()):
            code3 = chain[ir].residue_type().code3
            if code3 == rototranslation_code3: e.add_cofactor(chain[ir])
            elif code3 == crmsd_code3: e.add_ligand(chain[ir])
        if e.is_OK(): entries.append(e)

tmp_vec = Vec3()
output = open("%s-by-%s.pdb" % (crmsd_code3, rototranslation_code3), "w")
n_superimposed = len(entries[0].atoms_superimposed)
n_crmsd = len(entries[0].atoms_crmsd)
for ei in entries:
    for ej in entries:
        if ei == ej: continue
        try :
            if len(ei.atoms_superimposed) != len(ej.atoms_superimposed):
                print("superimposed sets differ in size")
                continue
            crmsd_val_1 = rms.crmsd(ei.atoms_superimposed, ej.atoms_superimposed, n_superimposed, True)
            if len(ei.atoms_crmsd) != len(ej.atoms_crmsd):
                print("crsmd sets differ in size")
                continue
            crmsd_val_2 = rms.calculate_crmsd_value(ei.atoms_crmsd, ej.atoms_crmsd, n_crmsd)
            print("%s %4d %3s %c - %s %4d %3s %c : %7.3f  %7.3f" %
                  (ei.pdb_code, ei.ligand.id(), ei.ligand.residue_type().code3, ei.ligand.owner().id(),
                   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)

```
            if ej == entries[0]:
                output.write("MODEL     %1\n")
                for ai in range(ei.ligand.count_atoms()):
                    rms.apply(ei.ligand[ai])
                    output.write(ei.ligand[ai].to_pdb_line() + "\n")
                output.write("ENDMDL\n")
        except:
            pass

output.write("MODEL     %d\n" % (1))
for ai in range(entries[0].ligand.count_atoms()):
    output.write(entries[0].ligand[ai].to_pdb_line() + "\n")
output.write("ENDMDL\n")

output.write("MODEL     %d\n" % (2))
for ai in range(entries[0].cofactor.count_atoms()):
    output.write(entries[0].cofactor[ai].to_pdb_line() + "\n")
output.write("ENDMDL\n")

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)

```
LogManager.INFO()
IF_ROW_OUTPUT = False

if len(sys.argv) < 3:
    print("""

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


CATEGORIES: core/calc/structural/transformations/CrmsdOnVec3
KEYWORDS:   PDB input; ligand; crmsd
GROUP: Structure calculations

  """)
    sys.exit()

rms = CrmsdOnVec3()
select_ligand = SelectResidueByName(sys.argv[1])
ref_strctr = Pdb(sys.argv[2], "").create_structure(0)
ref_atoms = copy_selected_coordinates(ref_strctr, select_ligand)

for f_model in sys.argv[3:]:
    strctr = Pdb(f_model, "").create_structure(0)
    ligand_atoms = copy_selected_coordinates(strctr, select_ligand)
    crsmd_val = rms.crmsd(ligand_atoms, ref_atoms, len(ref_atoms), True)  # superimpose a reference onto a model
    # crsmd_val = rms.crmsd(ref_atoms, ligand_atoms, len(ref_atoms), True)      # superimpose a model onto a reference
    if IF_ROW_OUTPUT:
        print("%7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f %8.3f %8.3f %8.3f %8.3f %8.3f %8.3f" %
          (rms.rot_x().x, rms.rot_x().y, rms.rot_x().z,
           rms.rot_y().x, rms.rot_y().y, rms.rot_y().z,
           rms.rot_z().x, rms.rot_z().y, rms.rot_z().z,
           rms.tr_before().x, rms.tr_before().y, rms.tr_before().z,
           rms.tr_after().x, rms.tr_after().y, rms.tr_after().z))
    else:
        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:

```
import sys

from pybioshell.core.data.io import find_pdb


if len(sys.argv) < 2 :
    print("""

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


```

(continues on next page)

(continued from previous page)

```
EXAMPLE:
   python3 list_pdb_ligands.py 5edw.pdb


CATEGORIES: core/data/io/find_pdb
KEYWORDS:   PDB input; ligand
GROUP:      File processing; Data filtering

  """)
    sys.exit()

for pdb_fname in sys.argv[1:] :
    structure = find_pdb(pdb_fname, "./").create_structure(0)
    for ic in range(structure.count_chains()) :
        chain = structure[ic]
        #print(chain.terminal_residue_index())

        for ir in range(chain.terminal_residue_index() + 1,chain.size()) :
            resid = chain[ir]
            code3 = resid.residue_type().code3
            if resid.residue_type().code3 == "HOH" : continue # Skip water molecules, they are so obvious and abundant
            formula = structure.formula(code3)
            hetname = structure.hetname(code3)
            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:

```
import sys

from pybioshell.std import vector_std_shared_ptr_core_data_sequence_Sequence_t
from pybioshell.core.data.io import read_clustalw_file
from pybioshell.core.data.sequence import SequenceProfile


if len(sys.argv) < 2 :
  print("""

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

```

(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:

```
import sys

from pybioshell.core.data.io import read_fasta_file, Pdb


if len(sys.argv) < 3 :
  print("""

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

```

(continues on next page)

(continued from previous page)

```
USAGE:
    python3 partial_thread.py ali.fasta template.pdb [chain-id]


EXAMPLE:
    python3 partial_thread.py 2azaA_2pcyA-ali.fasta 2aza.pdb A


CATEGORIES: core/alignment
KEYWORDS:   FASTA input; sequence
GROUP:      File processing; Data filtering

  """)
  sys.exit()

def select_aligned_residues(query_seq, template_seq, template_chain):
    j = 0
    out = []
    for i in range(len(query_seq)):
        if template_seq[i] != '-':
            if query_seq[i] != '-':  out.append(template_chain[j])
            j += 1
    return out


def print_atoms(residues):
    for r in residues:
        for i in range(r.count_atoms()):
            print(r[i].to_pdb_line())


fasta = read_fasta_file(sys.argv[1])
seq1 = fasta[0].sequence
seq1_gapless = fasta[0].create_ungapped_sequence().sequence
seq2 = fasta[1].sequence
seq2_gapless = fasta[1].create_ungapped_sequence().sequence

print(fasta[0].sequence,fasta[1].sequence)

strctr = Pdb(sys.argv[2],"").create_structure(0)
if len(sys.argv) > 3:
    chain = strctr.get_chain(sys.argv[3])
else:
    chain = strctr[0]
seq_pdb = chain.create_sequence().sequence
if seq_pdb == seq1_gapless:
    atoms = select_aligned_residues(seq2, seq1, chain)
    print_atoms(atoms)
elif seq_pdb == seq2_gapless:
    atoms = select_aligned_residues(seq1, seq2, chain)
    print_atoms(atoms)
else:
    print("template sequence can't be identified in the given alignment")
```

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:

```
import sys

from pybioshell.core.data.io import Pdb

if len(sys.argv) < 2 :
    print("""

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


CATEGORIES: core/data/io/Pdb
KEYWORDS:   PDB output; clustering
GROUP:      File processing; Structure calculations

  """)
    sys.exit()

chain_ids = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890abcdefghijklmnopqrstuvw"
clusters_file = open(sys.argv[1])
ligand_code = sys.argv[2]
iline = 0
for line in clusters_file:
    iline += 1
```

(continues on next page)

(continued from previous page)

```
    tokens = line.strip().split()
    outp = open("c"+str(iline)+tokens[0]+".pdb","w")
    p = Pdb(tokens[1],"", False)
    s = p.create_structure(0)
    i_chain = 0
    for ic in range(s.count_chains()):
        c = s[ic]
        for ir in range(c.count_residues()):
            r = c[ir]
            for ia in range(r.count_atoms()):
                a = r[ia]
                outp.write(a.to_pdb_line() + "\n")

    for fname in tokens[2:]:
        p = Pdb(fname,"", False)
        s = p.create_structure(0)
        for ic in range(s.count_chains()):
            c = s[ic]
            for ir in range(c.count_residues()):
                r = c[ir]
                if r.residue_type().code3 != ligand_code: continue
                r.owner().id(chain_ids[i_chain])
                for ia in range(r.count_atoms()):
                    a = r[ia]
                    outp.write(a.to_pdb_line() + "\n")
    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_
    ↪residues(),
31            "residues and", s.count_atoms(), "atoms\n")
32      print("deposited : ", s.deposition_date())
33      print("Is XRAY?  : ", (s.is_xray()))
34      print("Is NMR?   : ", (s.is_nmr()))
35      print("Is EM?    : ", (s.is_em()))
36      print("resolution: ", s.resolution())
37      print("R-value   : ", s.r_value())
38      print("R-free    : ", s.r_free())
39      if len(s.keywords()) > 0:
40          print("Keywords  : ", s.keywords()[0], end="")
41          for k in s.keywords()[1:]:
42              print(", ", k, end="")
43          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:

```
import sys

from pybioshell.core.data.io import find_pdb

# change that setting to False to include ligands in the output sequence
IF_EXCLUDE_LIGANDS = True

if len(sys.argv) < 2 :
  print("""

Extracts amino acid (or nucleotide) sequence from a PDB file.

```

(continues on next page)

(continued from previous page)

```
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


CATEGORIES: core/data/io/find_pdb
KEYWORDS:   PDB input; FASTA; Format conversion
GROUP:      File processing; Format conversion

  """)
  sys.exit()

for pdb_fname in sys.argv[1:] :
  structure = find_pdb(pdb_fname, "./").create_structure(0)
  for ic in range(structure.count_chains()) :
    chain = structure[ic]
    print(">",structure.code(), chain.id())
    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

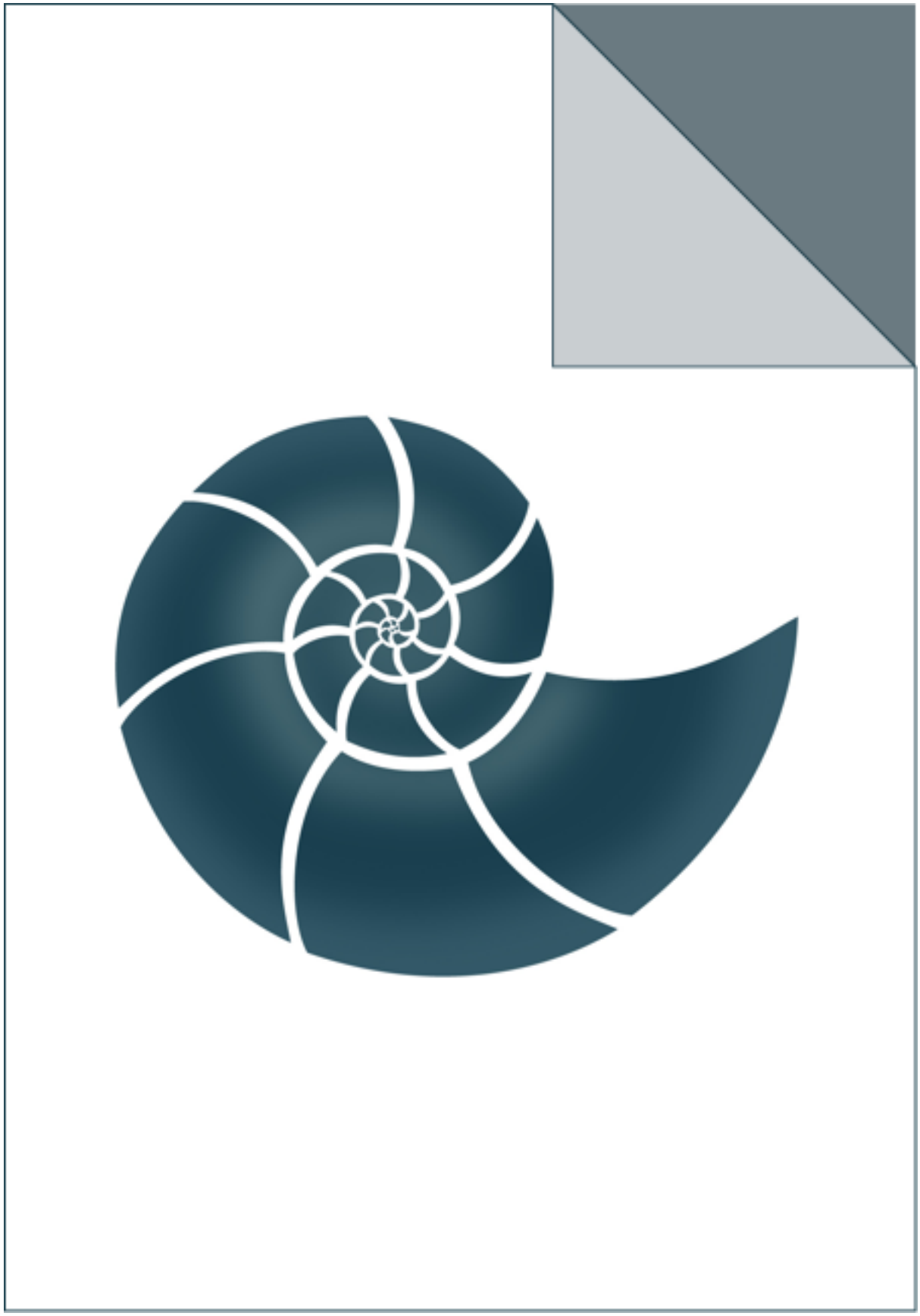
Program source:

```
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
11 USAGE:
```

(continues on next page)

(continued from previous page)

```
    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

CATEGORIES: core/data/io/write_seq
KEYWORDS:   PDB input; secondary structure; Format conversion
GROUP:      File processing; Format conversion
  """)
  sys.exit()

structure = Pdb(sys.argv[1],"").create_structure(0)
outname = sys.argv[2] if len(sys.argv) > 2 else "stdout"
for ic in range(structure.count_chains()) :
    chain = structure[ic]
    ss = chain.create_sequence()
    a = create_seq_string(ss)
    print(a)

```



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
9  Extracts amino acid (or nucleotide) sequence from a PDB file.
10
11
```

(continues on next page)

(continued from previous page)

```
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

CATEGORIES: core/data/io/write_ss2
KEYWORDS:   PDB input; secondary structure; Format conversion
GROUP:      File processing; Format conversion
  """)
  sys.exit()

structure = find_pdb(sys.argv[1], "./").create_structure(0)
outname = sys.argv[2] if len(sys.argv) > 2 else "stdout"
for ic in range(structure.count_chains()) :
    chain = structure[ic]
    ss = chain.create_sequence()
    write_ss2(ss,outname)
    print() # Print empty line to separate chain: note that it works only when␣
↪printed to stdout

```

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:

```
import sys, math

from pybioshell.core.data.io import find_pdb
from pybioshell.core.data.basic import Vec3Cubic

from pybioshell.std import vector_core_data_basic_Vec3Cubic

if len(sys.argv) < 3 :
    print("""

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
```

(continues on next page)

(continued from previous page)

```

CATEGORIES: core/data/basic/Vec3Cubic
KEYWORDS:   PDB input; structural properties
GROUP: Structure calculations;

  """)
    sys.exit()


pdb = find_pdb(sys.argv[1], "./")
n_atoms = pdb.count_atoms(0)
cutoff = float(sys.argv[2])

Vec3Cubic.set_box_len(cutoff)
xyz = vector_core_data_basic_Vec3Cubic()
for i in range(n_atoms) : xyz.append( Vec3Cubic() )
histogram = [0 for i in range(121)]
for i_model in range(0, pdb.count_models()) :
  # print i_model
  pdb.fill_structure(i_model, xyz)
  for i_atom in range(n_atoms) :
      for j_atom in range(i_atom) :
          d = xyz[i_atom].closest_distance_square_to(xyz[j_atom],12*12)
          if d < 144 : histogram[ int(math.sqrt(d)*10) ] += 1

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:

```
import sys
from pybioshell.core.data.io import read_scorefile

if len(sys.argv) < 2 :
  print("""

Simple example that parses a score file (Rosetta output)


USAGE:
    python3 read_scorefile.py score-file

EXAMPLE:
    python3 read_scorefile.py scores.sf

CATEGORIES: core/data/io/read_scorefile
KEYWORDS:   scorefile input;
GROUP:      File processing; Format conversion
```

(continues on next page)

(continued from previous page)

```
  """)
  sys.exit()

sf = read_scorefile(sys.argv[1])

print("Number of rows: %d" % len(sf))
print("Number of columns: %d" % sf[0].size())
print("Known columns:")
for i in range(sf[0].size()) :
  print(sf.column_name(i))

```

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:

```
import sys, math

from pybioshell.core.data.io import find_pdb
from pybioshell.core.data.basic import Vec3
from pybioshell.std import vector_core_data_basic_Vec3

from pybioshell.core.calc.structural import *
from pybioshell.utils import LogManager
LogManager.INFO()



if len(sys.argv) < 2 :
    print("""

Calculates the radius of gyration from given pdb file coordinates.


USAGE:
    python3 rg.py input.pdb
```

(continues on next page)

(continued from previous page)

```

EXAMPLE:
    python3 rg.py 1cey.pdb


CATEGORIES: core/calc/structural/calculate_Rg_square
KEYWORDS:   PDB input; structural properties
GROUP: Structure calculations;

  """)
    sys.exit()

for pdb_fname in sys.argv[1:] :
    pdb=find_pdb(pdb_fname, "./")
    n_atoms = pdb.count_atoms(0)

    structure = pdb.create_structure(0)
    models=[]

    for i_model in range(0, pdb.count_models()) :
      xyz=vector_core_data_basic_Vec3()
      for i in range(n_atoms) : xyz.append( Vec3() )
      models.append(xyz)

    for i_model in range(0, pdb.count_models()) :
      pdb.fill_structure(i_model, models[i_model])
      try:
        print("Rg for %s, model # %5d : %7.3f" % (pdb_fname.split("/")[-1].split(".
↪")[0],i_model,
            math.sqrt(calculate_Rg_square(models[i_model][0], models[i_model][n_atoms-
↪1])))))
      except:
        sys.stderr.write(str(sys.exc_info()[0])+" "+str(sys.exc_info()[1]))


```

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
7  from pybioshell.core.calc.structural.transformations import *
8  from pybioshell.utils import LogManager
9
10 REFERENCE_FROM, REFERENCE_TO = 23, 32 # 25, 390
11
12 LogManager.INFO()
13
14 if len(sys.argv) < 3:
15     print("""
16
17 Superimposes protein structures based on a structural fragment.
18
19
20 This script superimposes all models given at command line (at least one) on the␣
   ↪reference structure.
21 The superimposition is based on C-alpha atoms of residues from %d to %d. If you need␣
   ↪another fragment,
22 change these values in the script!
23
24 USAGE:
25     python3 superimpose_by_fragment.py reference.pdb model1.pdb [model2.pdb...]
26
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 -␣
   ↪only C-alfas
43 structure = pdb.create_structure(0)
44 n_atoms = REFERENCE_TO - REFERENCE_FROM + 1
45 xyz = vector_core_data_basic_Vec3()                                  # --- std::vector␣
   ↪of Vec3 object is required to calculate superimposition
46 for i in range(REFERENCE_FROM, REFERENCE_TO+1):                # --- fill the vector␣
   ↪with the selected reference coordinates
47     r = structure[0][i]                              # --- i-th residue of the first␣
   ↪chain
48     xyz.append(r.find_atom(" CA "))
49
50 #out_fname = "rot.pdb"
51 for pdb_fname in sys.argv[2:]:                                       # --- iterate␣
   ↪over all models
52     out_fname = "rot-" + pdb_fname.split(os.path.sep)[-1]
53     other_pdb = Pdb(pdb_fname, "", False)
54     other_structure = other_pdb.create_structure(0)
55     other_xyz = vector_core_data_basic_Vec3()                # --- container for␣
   ↪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␣
    ↪with the selected  coordinates
59              r = other_structure[0][i]                       # --- i-th residue of the␣
    ↪first chain
60              other_xyz.append(r.find_atom(" CA "))
61          rms_val = rms.crmsd(xyz, other_xyz, n_atoms, True)
62          rms.apply_inverse(other_structure)
63          write_pdb(other_structure, out_fname, 0)
64      except:
65          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 referance 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)

```
    print("""

Calculates TMScore value on two or more structures.
First file is a referance structure and the second can be multimodel 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


CATEGORIES: core/calc/structural/transformations/TMScore
KEYWORDS:   PDB input; TMScore
GROUP: Structure calculations


  """)
    sys.exit()


if len(sys.argv) == 3:

    pdb = Pdb(sys.argv[1], "is_not_water", False)
    n_atoms = pdb.count_atoms(0)
    ref_structure = pdb.create_structure(0)
    ref_xyz = vector_core_data_basic_Vec3()
    for i in range(n_atoms): ref_xyz.append(Vec3())
    pdb.fill_structure(0, ref_xyz)

    pdb = Pdb(sys.argv[2], "is_not_water", False)
    n_atoms = pdb.count_atoms(0)

    structure = pdb.create_structure(0)
    models = []

    for i_model in range(0, pdb.count_models()):
        xyz = vector_core_data_basic_Vec3()
        for i in range(n_atoms): xyz.append(Vec3())
        models.append(xyz)

    for i_model in range(0, pdb.count_models()):
        pdb.fill_structure(i_model, models[i_model])
        tmscore = TMScore(models[i_model],ref_xyz)
        try:
          print("%2d %6.3f" % (i_model,tmscore.tmscore()))
        except:
          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

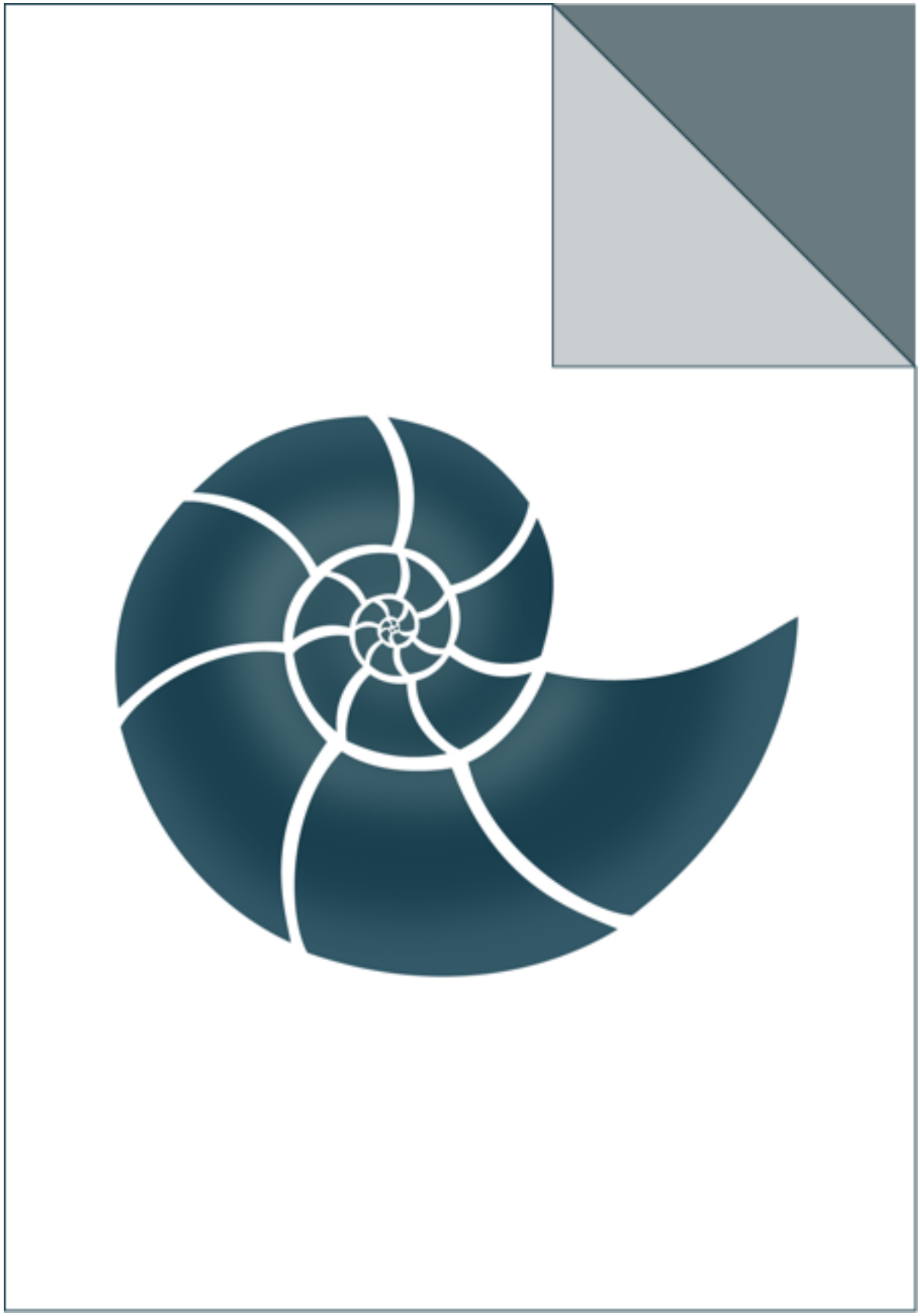
Program source:

```
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)

```

CATEGORIES: core/calc/structural/SaturatedRing6Geometry
KEYWORDS:   PDB input; structural properties
GROUP: Structure calculations

  """)
    sys.exit()


pdb = find_pdb(sys.argv[1], "./")
strctr = pdb.create_structure(0)
for i_chain in range(strctr.count_chains()) :
  chain = strctr[i_chain]
  for i_res in range(chain.count_residues()) :
    if chain[i_res].residue_type().code3 == sys.argv[2] :
      atoms = []
      for at_name in sys.argv[3:] :
        try :
          at_name_fixed = at_name.replace("_"," " )
          atoms.append( chain[i_res].find_atom(at_name_fixed))
          if not atoms[-1] :
            sys.stderr.write("Can't find atom "+at_name_fixed+" in "+sys.argv[2]+"␣
↪residue\n")
        except :
          sys.stderr.write("Can't find atom "+at_name+" in "+sys.argv[2]+" residue\n")
      s = SaturatedRing6Geometry(atoms[0],atoms[1],atoms[2],atoms[3],atoms[4],
↪atoms[5])
      print(s.first_wing_angle(),s.second_wing_angle())

```



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

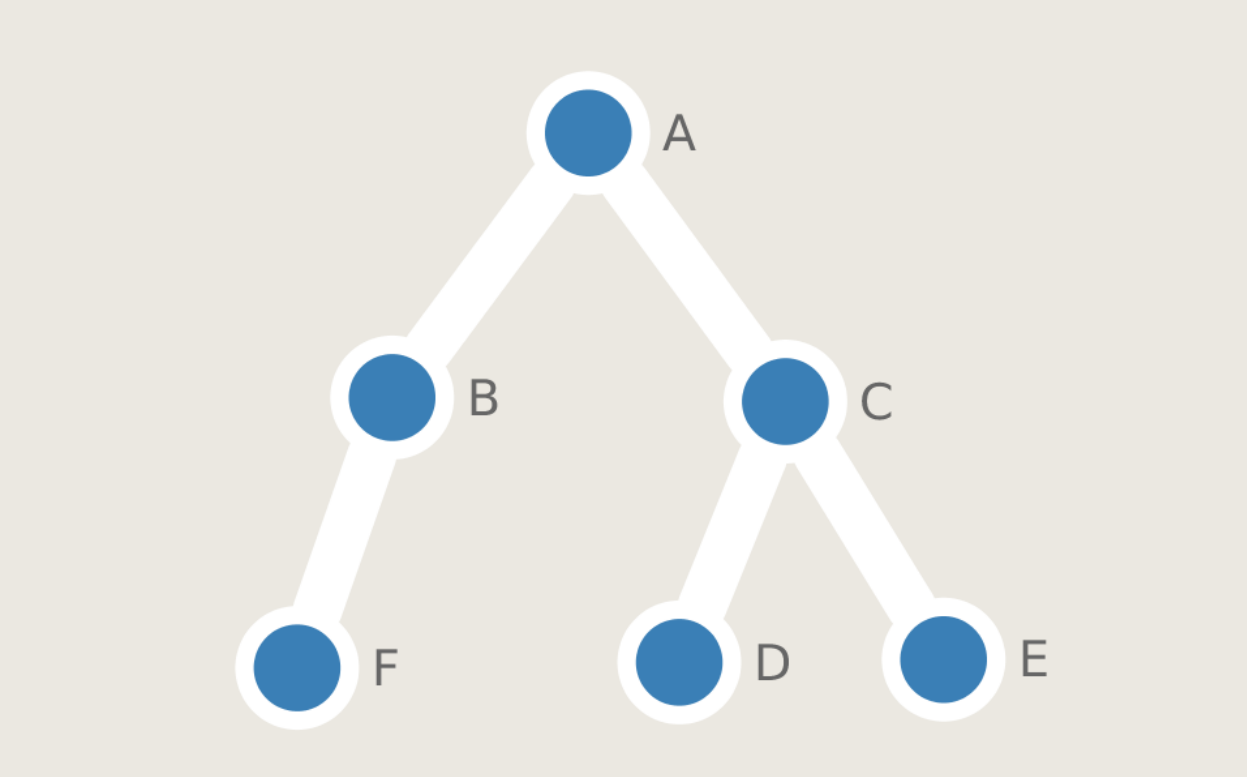
Program source:

```
#include <memory>
#include <iostream>

#include <core/algorithms/trees/TreeNode.hh>
#include <core/algorithms/trees/algorithms.hh>
#include <core/algorithms/trees/trees_io.hh>


/** @brief Simple demo for BinaryTreeNode class
 *
 * This program creates a small tree with 6 nodes and performs various operations on it
 *
 * CATEGORIES: core/algorithms/trees/BinaryTreeNode
 * KEYWORDS:   algorithms; data structures; graphs
 * IMG: ex_BinaryTreeNode_1.png
 * IMG_ALT: Example tree node
 */
int main(const int argc, const char* argv[]) {

  using namespace core::algorithms::trees;

  typedef std::shared_ptr<BinaryTreeNode<char>> Node_SP; // --- Let's make the typename shorter
  Node_SP p1(new BinaryTreeNode<char>(0, 'A'));
  Node_SP p2(new BinaryTreeNode<char>(1, 'B'));
  Node_SP p3(new BinaryTreeNode<char>(2, 'C'));
```

(continues on next page)

(continued from previous page)

```
  Node_SP p4(new BinaryTreeNode<char>(3, 'D'));
  Node_SP p5(new BinaryTreeNode<char>(4, 'E'));
  Node_SP p6(new BinaryTreeNode<char>(5, 'F'));

  p1->set_left_right(p2, p3);
  p3->set_left_right(p4, p5);
  p2->set_left(p6);
  std::cout << "Size of the whole tree and its right branch: " << size(p1) << " " <<␣
↪size(p3)
            << " (should be 6 and 3)\n";

  std::vector<char> elements;
  collect_leaf_elements(p1, elements);
  std::cout << "Leaf-only elements (E D F):\n";
  for (std::vector<char>::const_iterator i = elements.begin(); i != elements.end();␣
↪i++)
    std::cout << *i << ' ';
  std::cout << "\n";

  std::cout << "All elements stored on the tree (A C E D B F):\n";
  elements.clear();
  collect_elements(p1, elements);
  for (std::vector<char>::const_iterator i = elements.begin(); i != elements.end();␣
↪i++)
    std::cout << *i << ' ';
  std::cout << "\n";

  std::cout << "Leaf-only nodes (E D F):\n";
  elements.clear();
  std::vector<Node_SP> nodes;
  collect_leaf_nodes(p1, nodes);
  for (Node_SP & i : nodes)
    std::cout << i->element << ' ';
  std::cout << "\n";

  std::cout << "The tree was:\n";
  XMLFormatters<Node_SP> xml(std::cout);
  write_tree(p1, xml.start, xml.leaf, xml.stop);

  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:

```
#include <iostream>
#include <memory>

#include <core/index.hh>
#include <core/algorithms/graph_algorithms.hh>
#include <core/chemical/Molecule.hh>
```

(continues on next page)

(continued from previous page)

```
#include <core/chemical/molecule_utils.hh>
#include <core/calc/structural/angles.hh>
#include <core/data/structural/PdbAtom.hh>
#include <core/chemical/PdbMolecule.hh>
#include <core/chemical/Bond.hh>

core::chemical::PdbMolecule_SP create_toluene_molecule() {

  using namespace core::chemical;
  using namespace core::data::structural;

  // --- Define atoms that we use to build a molecule
  PdbAtom_SP atoms[] = {std::make_shared<PdbAtom>(1, " C1 ", 0, 0, 0),
                        std::make_shared<PdbAtom>(2, " C2 ", 1.24, 0.72, 0),
                        std::make_shared<PdbAtom>(3, " C3 ", 1.24, 2.16, 0),
                        std::make_shared<PdbAtom>(4, " C4 ", 0, 2.88, 0),
                        std::make_shared<PdbAtom>(5, " C5 ", -1.24, 2.16, 0),
                        std::make_shared<PdbAtom>(6, " C6 ", -1.24, 0.72, 0),
                        std::make_shared<PdbAtom>(7, " C7 ", 0, -1.52, 0)};
  PdbMolecule_SP toluene = std::make_shared<PdbMolecule>();

  // --- Insert atoms into the molecule
  for (PdbAtom_SP ai : atoms) toluene->add_atom(ai);
  // --- Create bonds between them
  toluene->bind_atoms(0, 1, BondType::AROMATIC);
  toluene->bind_atoms(1, 2, BondType::AROMATIC);
  toluene->bind_atoms(2, 3, BondType::AROMATIC);
  toluene->bind_atoms(3, 4, BondType::AROMATIC);
  toluene->bind_atoms(4, 5, BondType::AROMATIC);
  toluene->bind_atoms(0, 5, BondType::AROMATIC);
  toluene->bind_atoms(0, 6, BondType::SINGLE);

  return toluene;
}

/** @brief Demonstrates how to create a Molecule object based on PdbAtom data type␣
↪(as nodes of the graph)
 *
 * This demo is similar to ex_Molecule_vec3, the difference is that here PdbAtom␣
↪instances are used as graph nodes
 * rather than Vec3 instance. It creates a toluene molecule and detects planar and␣
↪dihedral angles.
 *
 * CATEGORIES: core::chemical::Molecule
 * KEYWORDS: molecule
 * IMG: Toluen_dihedral_flat_angle.png
 * IMG_ALT: Planar angles in a toluen molecule
 */
int main(const int argc, const char *argv[]) {

  using namespace core::chemical;
  using namespace core::data::structural;

  PdbMolecule_SP molecule;
  if (argc == 1)
    molecule = create_toluene_molecule();
  else {
```

(continues on next page)

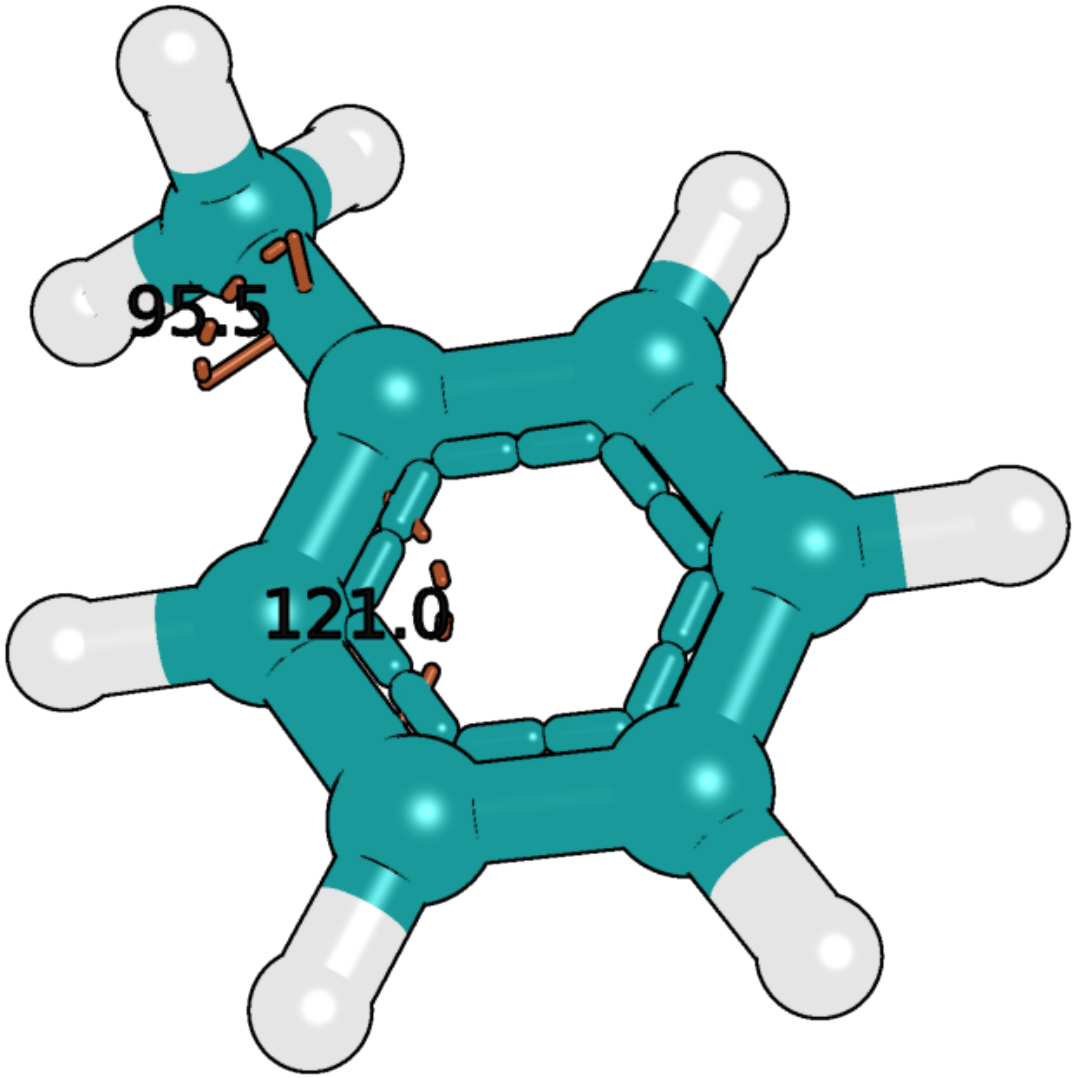
(continued from previous page)

```
    // --- Read structure that we use to build a molecule
    core::data::io::Pdb reader(argv[1]); // file name (PDB format, may be gzip-ped)
    core::data::structural::Structure_SP strctr = reader.create_structure(0);
    // --- Create molecule object
    molecule = structure_to_molecule(*strctr);
    //molecule = create_molecule<Structure::atom_iterator>(strctr->first_atom(), strctr->last_atom(), 0.1);
  }

  // --- Print some info about the molecule
  std::cout << molecule->count_atoms() << " atoms, " << molecule->count_bonds() << " bonds\n";
  for (auto atom_it=molecule->begin_atom();atom_it!=molecule->end_atom();++atom_it) {
    PdbAtom_SP ai = *atom_it;
    std::cout << "atom " << ai->id() << " bonded to " << molecule->count_bonds(ai) << " atoms:";
    for (auto n_it = molecule->begin_atom(ai); n_it != molecule->end_atom(ai); ++n_it)
      std::cout << " " << (*n_it)->id();
    std::cout << "\n";
  }

  // --- Find all planar angles in the molecule
  std::vector<std::tuple<PdbAtom_SP, PdbAtom_SP, PdbAtom_SP>> planars;
  find_planar_angles(*molecule, planars);
  // --- Sort the angles just to make the output stable i.e. every time in the same order so it can be used for benchmarking
  std::sort(planars.begin(), planars.end(), ComparePlanarAngles());

  std::cout << "Detected planar angles:\n"; // --- Evaluate and print all the planars
  for (auto pi : planars) {
    PdbAtom &a1 = *std::get<0>(pi);
    PdbAtom &a2 = *std::get<1>(pi);
    PdbAtom &a3 = *std::get<2>(pi);
    std::cout << a1.id() << " -- " << a2.id() << " -- " << a3.id() << " " <<
    core::calc::structural::evaluate_planar_angle(a1, a2, a3) * 180.0 / 3.14159 << "\n";
  }

  // --- Find all torsion angles in the molecule
  std::vector<std::tuple<PdbAtom_SP, PdbAtom_SP, PdbAtom_SP, PdbAtom_SP>> torsions;
  find_torsion_angles(*molecule, torsions);
  // --- Sort also dihedral angles
  std::sort(torsions.begin(), torsions.end(), CompareDihedralAngles());
  std::cout << "Detected dihedral angles:\n"; // --- Evaluate and print all the planars
  for (auto ti : torsions) {
    PdbAtom &a1 = *std::get<0>(ti);
    PdbAtom &a2 = *std::get<1>(ti);
    PdbAtom &a3 = *std::get<2>(ti);
    PdbAtom &a4 = *std::get<3>(ti);
    std::cout << a1.id() << " -- " << a2.id() << " -- " << a3.id() << " -- " << a4.id() << " " <<
    core::calc::structural::evaluate_dihedral_angle(a1, a2, a3, a4) * 180.0 / 3.14159 << "\n";
  }

  // --- 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_
      ↪name();
115   std::cout << "\n";
116 }
```



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

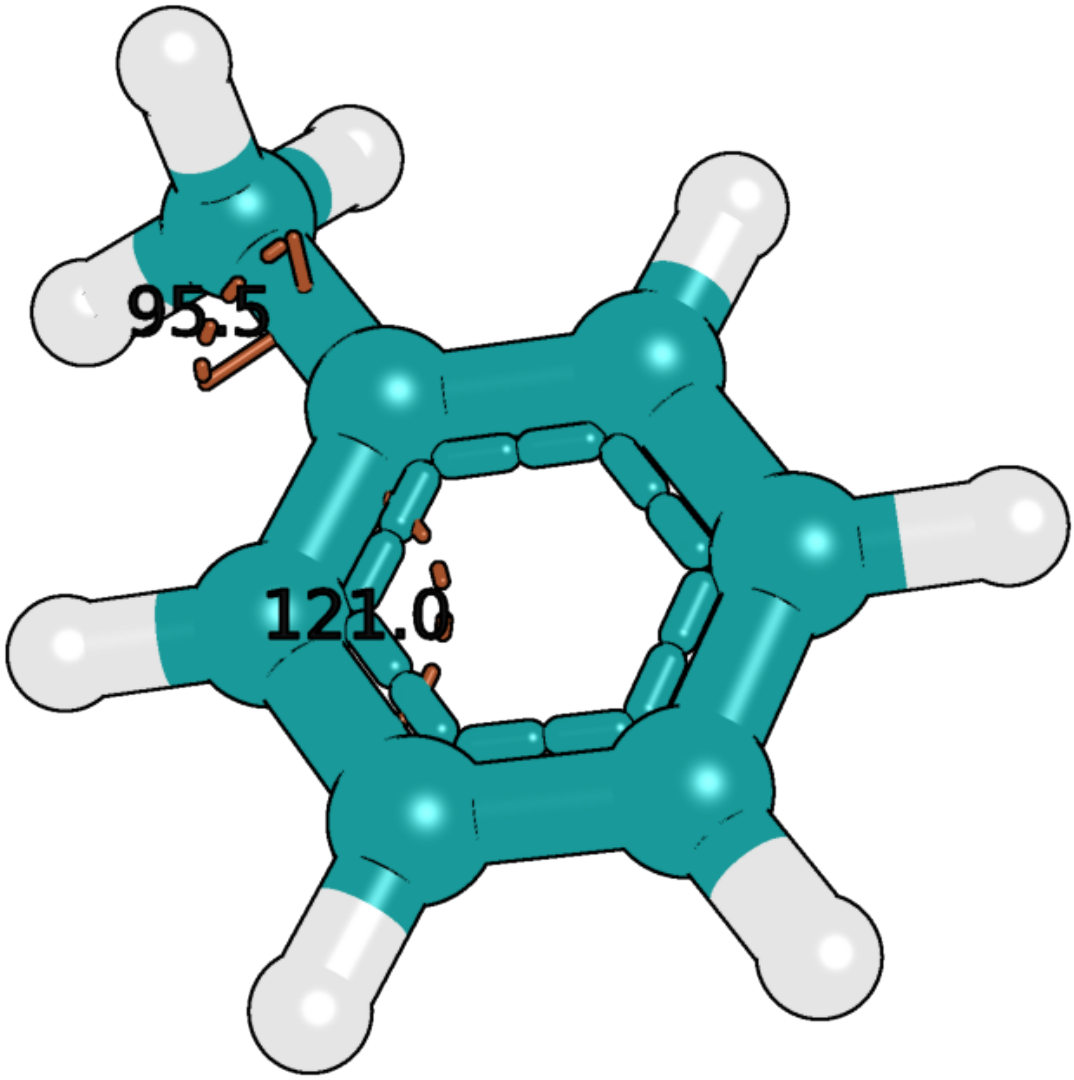
Program source:

```
#include <iostream>
#include <memory>

#include <core/chemical/Molecule.hh>
#include <core/chemical/molecule_utils.hh>
#include <core/calc/structural/angles.hh>
#include <core/data/basic/Vec3.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Demonstrates how to create a Molecule object based on Vec3 data type (Vec3 are nodes of the graph)
 *
 * This demo is similar to ex_Molecule, the difference is that here Vec3 instances are used as graph nodes
 * rather than PdbAtom instance. It creates a toluene molecule and detects planar angles.
 *
 * CATEGORIES: core::chemical::Molecule
 * KEYWORDS: molecule
 * IMG: Toluen_dihedral_flat_angle.png
 * IMG_ALT: Planar angles in a toluen molecule
 */
int main(const int argc, const char *argv[]) {

```

(continues on next page)

(continued from previous page)

```
  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::chemical;
  using namespace core::data::structural;

  Molecule<Vec3> toluene;
  Vec3 atoms[] = {Vec3(0, 0, 0),
                  Vec3(1.24, 0.72, 0),
                  Vec3(1.24, 2.16, 0),
                  Vec3(0, 2.88, 0),
                  Vec3(-1.24, 2.16, 0),
                  Vec3(-1.24, 0.72, 0),
                  Vec3(0, -1.52, 0)};

  // --- Mark atom numbers to check if the molecule is correct
  for (core::index2 i = 0; i < 7; ++i) atoms[i].register_ = i;

  // --- Insert atoms into the molecule
  for (Vec3 & ai : atoms) toluene.add_atom(ai);
  // --- Create bonds between them
  toluene.bind_atoms(0, 1, BondType::AROMATIC);
  toluene.bind_atoms(1, 2, BondType::AROMATIC);
  toluene.bind_atoms(2, 3, BondType::AROMATIC);
  toluene.bind_atoms(3, 4, BondType::AROMATIC);
  toluene.bind_atoms(4, 5, BondType::AROMATIC);
  toluene.bind_atoms(0, 5, BondType::AROMATIC);
  toluene.bind_atoms(0, 6, BondType::SINGLE);

  std::cout << "Connectivity (bonds):\n";
  for(auto atom_it=toluene.cbegin_atom();atom_it!=toluene.cend_atom();++atom_it) {
    std::cout << (*atom_it).register_ << " : ";
    for(auto atom_it2=toluene.cbegin_atom(*atom_it);atom_it2!=toluene.cend_atom(*atom_
↪it);++atom_it2)
      std::cout << " "<<(*atom_it2).register_;
    std::cout << "\n";
  }

  // --- Find all planar angles in the molecule
  std::vector<std::tuple<Vec3, Vec3, Vec3>> planars;
  find_planar_angles(toluene, planars);

  std::vector<double> planar_values;
  std::cout << "Detected planar angles:\n"; // --- Evaluate and print all the planars
  for (auto pi : planars) {
    Vec3 &a1 = std::get<0>(pi);
    Vec3 &a2 = std::get<1>(pi);
    Vec3 &a3 = std::get<2>(pi);
    planar_values.push_back(core::calc::structural::evaluate_planar_angle(a1, a2, a3)␣
↪* 180.0 / 3.14159);
  }

  // --- Sort the values before printing them to make the output stable
  std::sort(planar_values.begin(),planar_values.end());
  for (double value:planar_values) std::cout << value << "\n";
}
```



ex_NcbiSimilarityMatrixFactory

Test for loading substitution matrices available in BioShell. The program eads 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

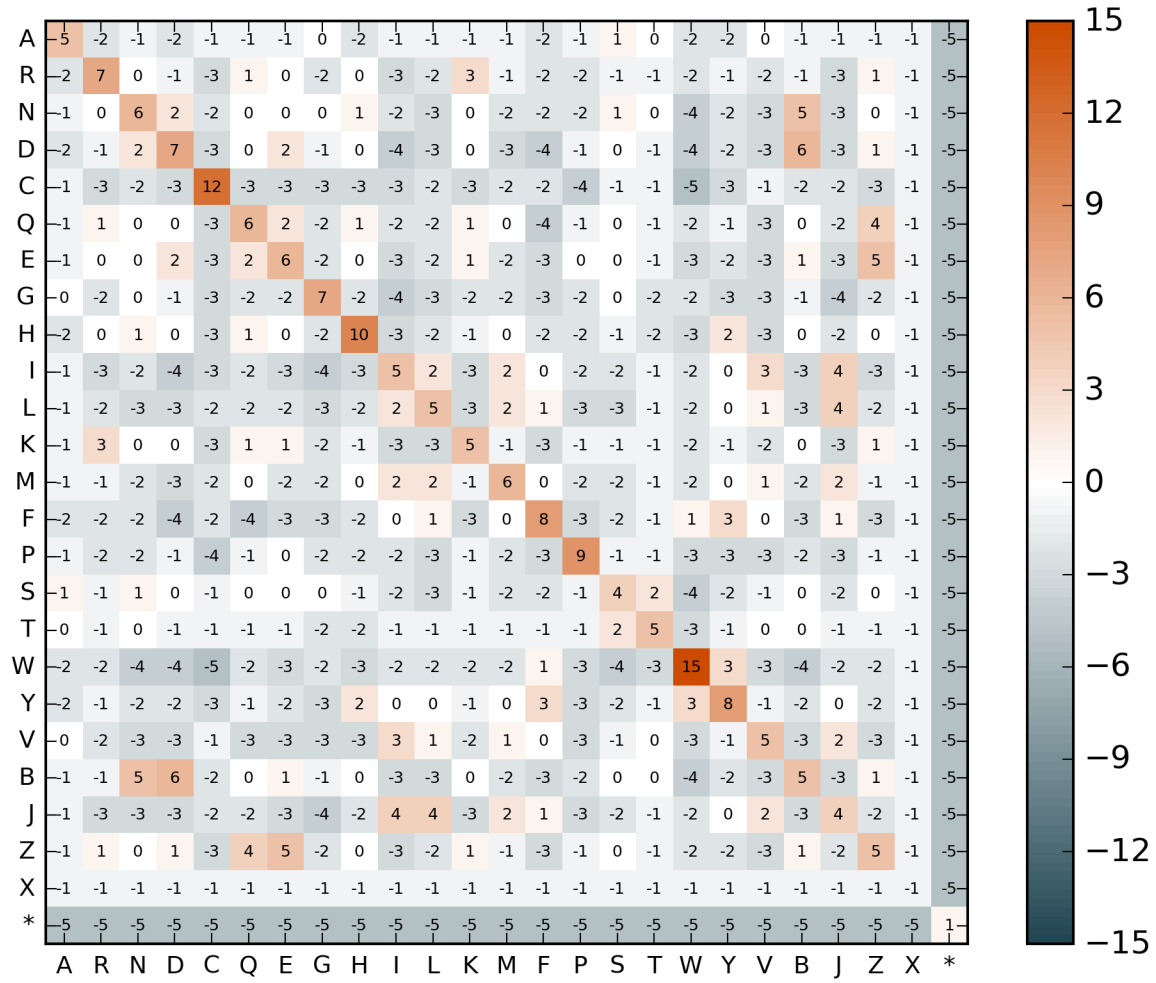
Program source:

```
#include <iostream>

#include <core/alignment/scoring/SimilarityMatrix.hh>
#include <core/alignment/scoring/NcbiSimilarityMatrixFactory.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Test for loading substitution matrices available in BioShell.

The program eads 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

)";


/** @brief Test for loading substitution matrices available in BioShell
 *
 * CATEGORIES: core::alignment::scoring::NcbiSimilarityMatrixFactory
 * KEYWORDS:   sequence alignment; substitution matrix
 * IMG: heatmap_1.png
 * IMG_ALT: BLOSUM62 matrix plotted
 */
```

(continues on next page)

(continued from previous page)

```
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::alignment::scoring;

  NcbiSimilarityMatrixFactory sim_factory = NcbiSimilarityMatrixFactory::get();

  if (argc == 1) {
    std::vector<std::string> names;
    sim_factory.get().matrix_names(names);
    std::cout << "\nMatrices defined in BioShell:\n";
    for (const auto &n : names)
      std::cout << "\t" << n;
    std::cout << "\n";
  } else {
    NcbiSimilarityMatrix_SP m = sim_factory.load_matrix(argv[1]);
    std::stringstream out;
    out << "\n";
    m->print("%4d", 4, out);
    std::cout << out.str() << "\n";
  }
}
```



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:

```
#include <iostream>

#include <simulations/evaluators/CallEvaluator.hh>
#include <simulations/forcefields/CalculateEnergyBase.hh>

#include <simulations/movers/ising/SingleFlip2D.hh>
#include <simulations/movers/ising/WolffMove2D.hh>

#include <simulations/observers/ObserveEvaluators.hh>
```

(continues on next page)

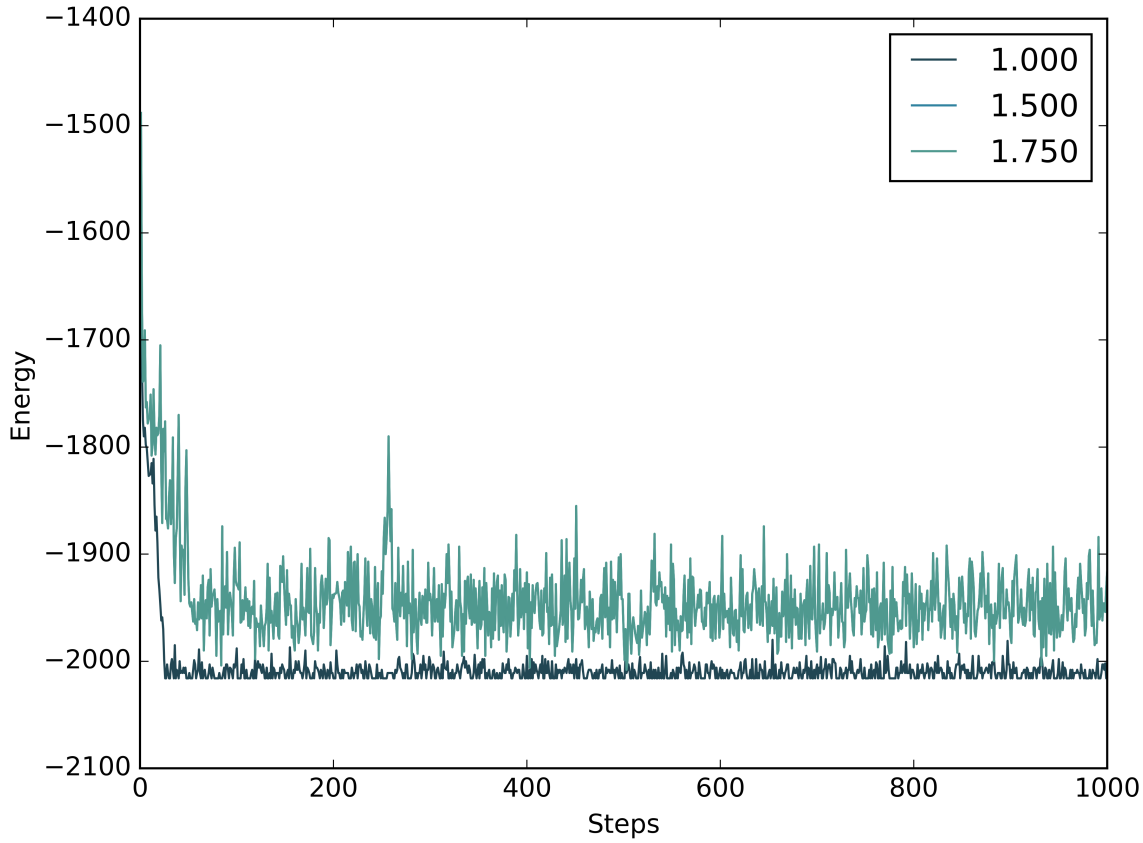
(continued from previous page)

```
#include <simulations/observers/ObserveMoversAcceptance.hh>
#include <simulations/observers/TriggerEveryN.hh>
#include <simulations/observers/ObserveReplicaFlow.hh>

#include <simulations/sampling/IsothermalMC.hh>
#include <simulations/sampling/ReplicaExchangeMC.hh>
#include <simulations/systems/ising/Ising2D.hh>

using namespace core::data::basic;

utils::Logger logs("ex_REMC_Ising");

std::string program_info = R"(

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
)";


/** @brief The program runs a Replica Exchange Monte Carlo simulation of a simple 2D Ising system (spin glass).
 *
 * This example shows how to set up a REMC simulation
 *
 * CATEGORIES: simulations/sampling/ReplicaExchangeMC; simulations/systems/ising/Ising2D
 * KEYWORDS:   REMC; Ising2D; observer; simulation
 * IMG: Energy_plot.png
 * IMG_ALT: Energy over time in Ising model
 */
int main(const int argc,const char* argv[]) {

  using namespace simulations::systems::ising;
  using namespace simulations::movers::ising;

  core::index4 n_outer_cycles = 10;
  core::index4 n_inner_cycles = 10;
  core::index4 n_exchanges = 10;
  std::vector<double> temperatures = {100.0, 7.5, 5, 4, 3, 2.5, 2.25, 2, 1.75, 1.5, 1};

  core::index2 system_size = 32;
  if (argc < 2) std::cerr << program_info;
  else {
    system_size = atoi(argv[1]);
    n_inner_cycles = atoi(argv[2]);
    n_outer_cycles = atoi(argv[3]);
```

(continues on next page)

(continued from previous page)

```
    n_exchanges = atoi(argv[4]);
  }

  core::calc::statistics::Random::get().seed(12345);  // --- seed the generator for repeatable results

  std::vector<std::shared_ptr<Ising2D<core::index1,core::index2>>> systems;
  std::vector<simulations::sampling::IsothermalMC_SP> replica_samplers;
  std::vector<simulations::forcefields::TotalEnergy_SP> energies;

  for (core::index2 irepl = 0; irepl < temperatures.size(); ++irepl) {

    // ---------- Create the systems to be sampled ----------
    std::shared_ptr<Ising2D<core::index1,core::index2>> system
        = std::make_shared<Ising2D<core::index1,core::index2>>(system_size, system_size);
    system->initialize();    // Populate system with random spins
    systems.push_back(system);
    energies.push_back(system);

    // ---------- Movers definition ----------
    simulations::movers::MoversSet_SP movers = std::make_shared<simulations::movers::MoversSetSweep>();
    movers->add_mover( std::make_shared<SingleFlip2D<core::index1,core::index2>>(*system),system->count_spins());
    movers->add_mover( std::make_shared<WolffMove2D<core::index1,core::index2>>(*system),system->count_spins()*0.2);

    // ---------- Create the sampler ----------
    auto sampler = std::make_shared<simulations::sampling::IsothermalMC>(movers,temperatures[irepl]);
    replica_samplers.push_back(sampler);
    sampler->cycles(n_inner_cycles,n_outer_cycles);

    simulations::observers::ObserveEvaluators_SP observations
      = std::make_shared<simulations::observers::ObserveEvaluators>(utils::string_format("energy-%.3f.dat",temperatures[irepl]));
    observations->add_evaluator(system);
    sampler->outer_cycle_observer(observations);
    simulations::observers::ObserveMoversAcceptance_SP obs_ms
      = std::make_shared<simulations::observers::ObserveMoversAcceptance>(*movers,utils::string_format("movers-%.3f.dat",temperatures[irepl]));
    obs_ms->observe_header();
    sampler->outer_cycle_observer(obs_ms);
  }

  bool replica_isothermal_observation_mode = true;
  auto remc = std::make_shared<simulations::sampling::ReplicaExchangeMC>(replica_samplers, energies, replica_isothermal_observation_mode);
  auto remc_flow = std::make_shared<simulations::observers::ObserveReplicaFlow>(*remc,"replica_flow.dat");
  remc->exchange_observer(remc_flow);
  remc->replica_exchanges(n_exchanges);
  remc->run();
}
```



ex_SelectChainResidueAtom

Extracts a fragment of a PDB file by applying a SelectChainResidueAtom selector. The selection string constists 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

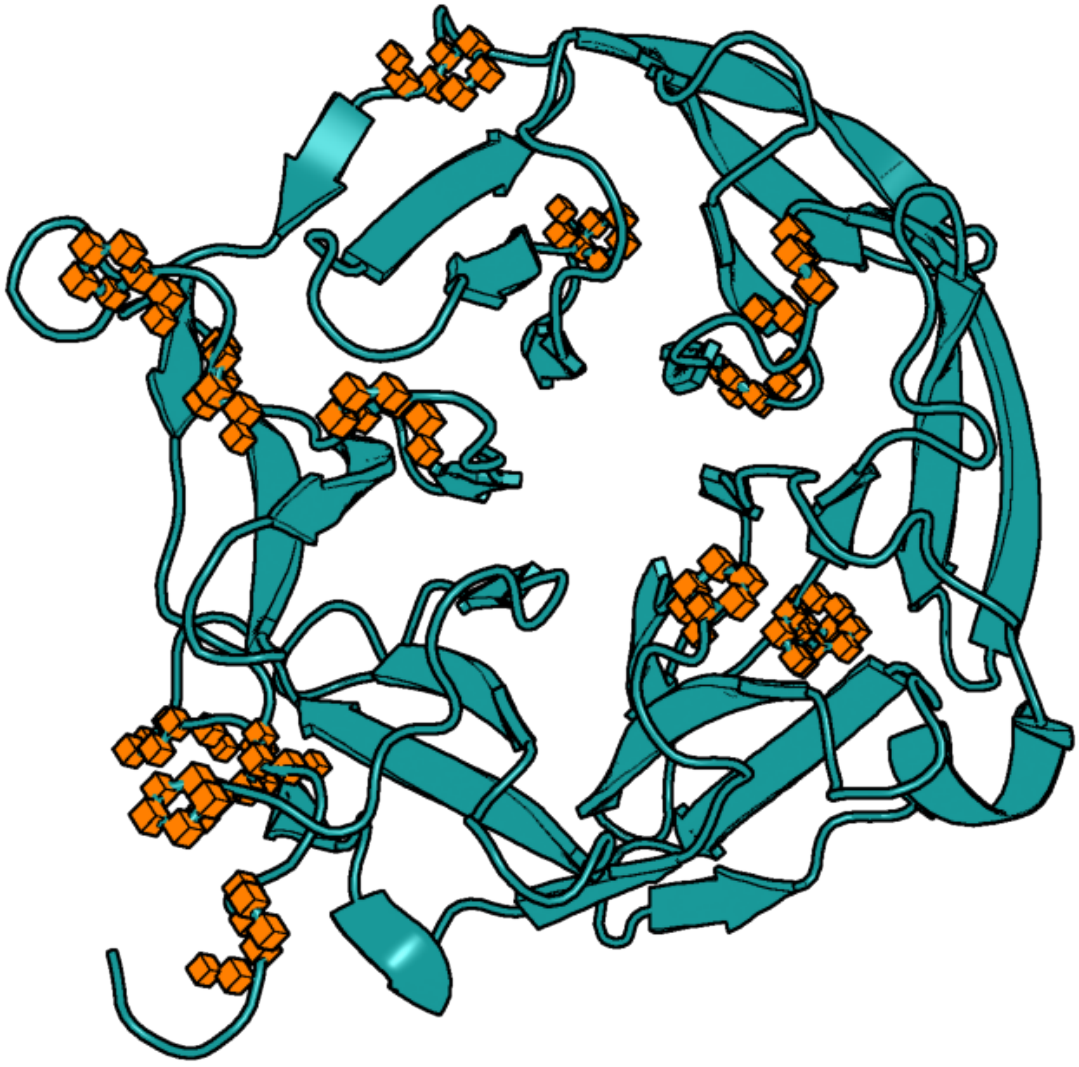
Program source:

```
#include <iostream>
#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Extracts a fragment of a PDB file by applying a SelectChainResidueAtom selector.

The selection string constists 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_

)";

/** @brief Extracts a fragment of a PDB file.
 *
 * CATEGORIES: core::data::structural::StructureSelector
 * KEYWORDS:   structure selectors; PDB input; PDB output
 * IMG: ex_SelectChainResidueAtoms_1.png
 * IMG_ALT: Proline residue selected from 1OFZ deposit
 */
int main(const int argc, const char* argv[]) {

  using namespace core::data::structural;

  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

```

(continues on next page)

(continued from previous page)

```
  // ---------- Read a PDB file and create a Structure object
  core::data::io::Pdb reader(argv[1],    // --- data file
      core::data::io::keep_all);         // --- a predicate to read ALL the ATOM lines␣
↪(by default hydrogens are excluded)
  Structure_SP strctr = reader.create_structure(0);

  // --- Create a selector object from a selector string
  selectors::SelectChainResidueAtom sel(argv[2]);
  Structure_SP full_copy = strctr->clone(sel); // --- cloning with this selector␣
↪makes a deep copy of everything
  for(auto atom_it=full_copy->first_atom();atom_it!=full_copy->last_atom();++atom_it)
    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::SelectResidueByName; core::data::structural::SelectPlanarCAGeometry

Input files:

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

Output files:

- stdout.out

Program source:

```
#include <core/data/io/Pdb.hh>
#include <core/calc/structural/angles.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/data/structural/selectors/SelectPlanarCAGeometry.hh>

#include <utils/exit.hh>
#include <utils/io_utils.hh>

std::string program_info = R"(
```

(continues on next page)

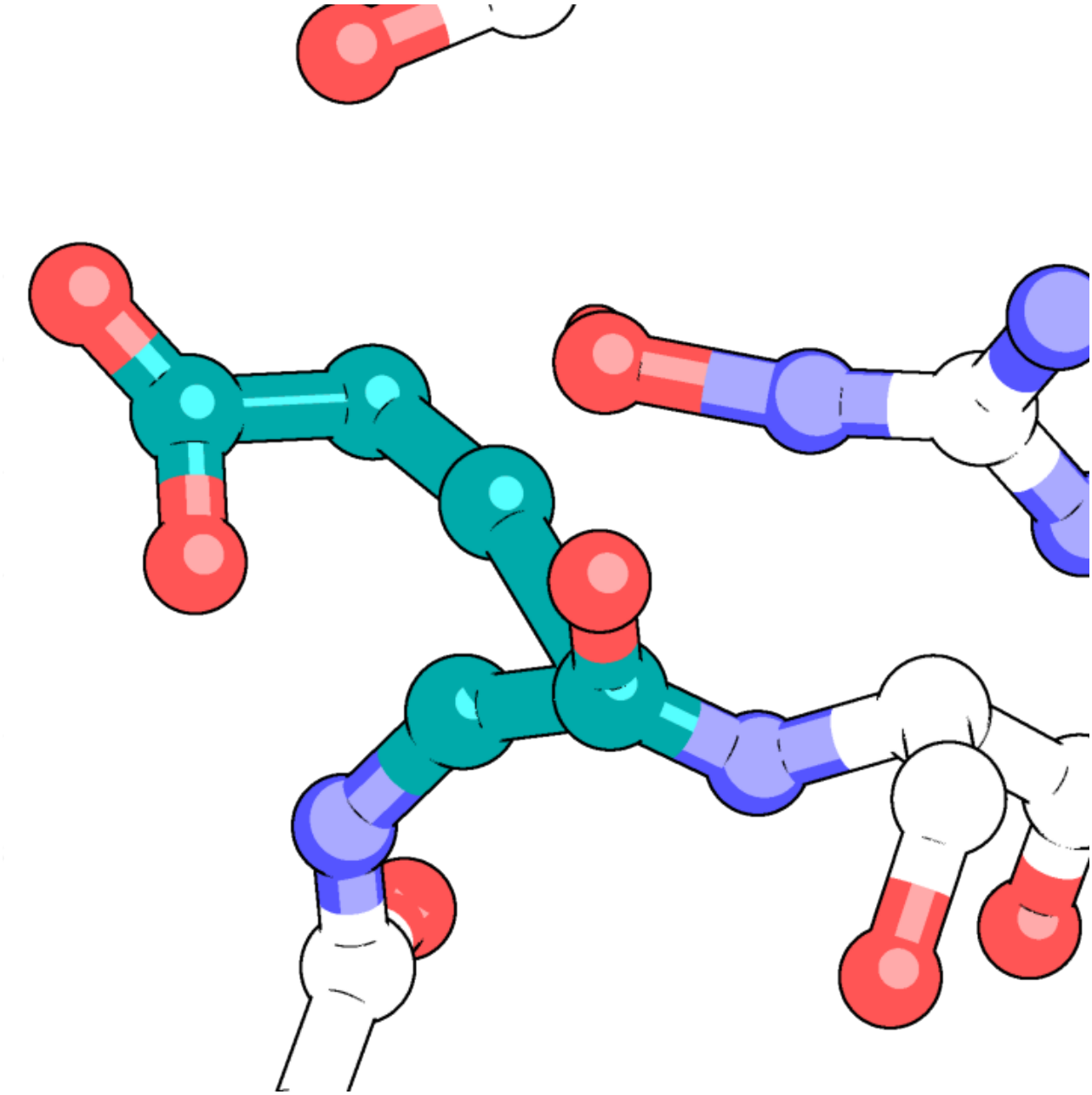
(continued from previous page)

```
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␣
    ↪should.
31   *
32   * CATEGORIES: core::data::structural::ResidueHasBBCB;␣
    ↪core::data::structural::SelectResidueByName;␣
    ↪core::data::structural::SelectPlanarCAGeometry
33   * KEYWORDS:   residue geometry; structure selectors; PDB input; structure validation
34   * IMG: 1NXB_A_56_Glu_pymol_5.png
35   * IMG_ALT: GLU56 of 1NXB deposit has planar geometry of alpha-carbon (witch␣
    ↪contradicts basic chemical knowledge)
36   */
37  int main(const int argc, const char *argv[]) {
38
39    if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about␣
    ↪missing program parameter
40    using namespace core::data::io;
41    using core::calc::structural::to_degrees;
42
43    core::data::io::Pdb reader(argv[1], is_not_alternative); // file name (PDB format,␣
    ↪may be gzip-ped)
44    core::data::structural::Structure_SP strctr = reader.create_structure(0);
45
46    // --- Selector that returns true if a residue has beta-carbon
47    core::data::structural::selectors::ResidueHasBBCB has_bb_cb;
48    // --- Selector that test the geometry on alpha carbon
49    core::data::structural::selectors::SelectPlanarCAGeometry tester;
50    // --- Selector that selects GLY residues
51    core::data::structural::selectors::SelectResidueByName is_gly("GLY");
52    for (auto res_it = strctr->first_residue(); res_it != strctr->last_residue(); ++res_
    ↪it) {
53      // --- If a residue is not GLY and has C-beta ...
54      if ((has_bb_cb(**res_it)) && (!is_gly(**res_it)))
55        std::cout << utils::string_format("%4d %3s %4s %s %7.2f %s\n", (*res_it)->id(),
56          (*res_it)->residue_type().code3.c_str(), (*res_it)->owner()->id().c_str(),␣
    ↪(tester(**res_it)) ? "WRONG" : " OK  ",
57          to_degrees(tester.evaluate_angle(**res_it)), utils::basename(strctr->code()).
    ↪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:

```
ex_VonMisesDistribution N
ex_VonMisesDistribution mu kappa
```

EXAMPLES:

```
ex_VonMisesDistribution 10000
ex_VonMisesDistribution 1.5708 100.0
```

Keywords:

- *statistics*

Categories:

- core/calc/statistics/VonMisesDistribution

Output files:

- stdout.out

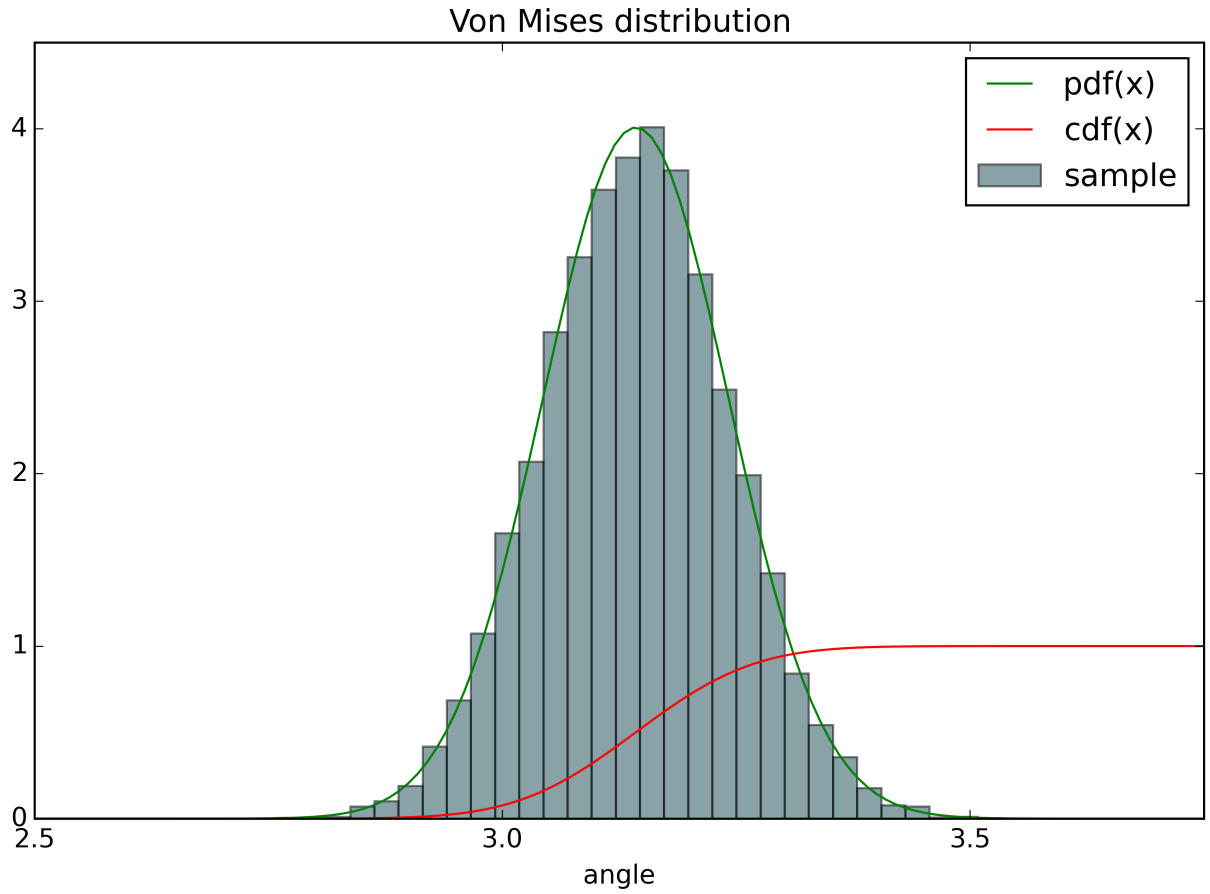
Program source:

```
#include <iostream>
#include <random>

#include <core/calc/statistics/VonMisesDistribution.hh>
#include <core/calc/statistics/Random.hh>

std::string program_info = R"(

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:
    ex_VonMisesDistribution N
    ex_VonMisesDistribution mu kappa

EXAMPLES:
    ex_VonMisesDistribution 10000
    ex_VonMisesDistribution 1.5708 100.0

)";

/** @brief Example which estimates parameters of von Mises distribution and tabulates its values
 * CATEGORIES: core/calc/statistics/VonMisesDistribution
 * KEYWORDS: statistics
 * IMG: von_mises.png
```

(continues on next page)

(continued from previous page)

```
 * IMG_ALT: Example von Mises distribution: histogram of a sample, pdf(x) and cdf(x)
 */
int main(const int argc, const char *argv[]) {

  using namespace core::calc::statistics;

  core::calc::statistics::VonMisesDistribution f(std::vector<double>{0.0, 1.0}); // --
↪- initial distribution
  if (argc == 3) {
    double mu = atof(argv[1]);
    double kappa = atof(argv[2]);
    f.copy_parameters_from(std::vector<double>{mu, kappa});
    std::cout << "# tabulating VonMisesDistribution: " << f << "\n";
    for (double x = -M_PI; x <= M_PI; x += M_PI / 25.0)
      std::cout
          << utils::string_format("%6.3f %9f %5.3f\n", x, f.evaluate(x),
                                  VonMisesDistribution::cdf(x, f.mu(), f.kappa()));
    return 0;
  }

  core::index4 N = 1000;
  if (argc < 2) std::cerr << program_info;
  else N = atoi(argv[1]);
  std::vector<std::vector<double>> input_data;
  Random r = Random::get();
  r.seed(9876543);
  std::normal_distribution<double> dist(M_PI / 2.0, 0.1);

  // ---------- prepare data that will be use to estimate the distribution
  for (core::index4 i = 0; i < N; ++i) {
    std::vector<double> v({dist(r)});
    input_data.push_back(v);
  }
  f.estimate(input_data); // --- run the estimation
  std::cout << f << "\n";
  // ---------- now prepare weighted data: just copy some points ten times and insert␣
↪them with weight 0.1
  input_data.clear();
  std::vector<double> weights;
  for (core::index4 i = 0; i < N; ++i) {
    double x = dist(r);
    std::vector<double> v({x});
    if (x < 2) {
      input_data.push_back(v);
      weights.push_back(1.0);
    } else {
      for (int j = 0; j < 10; ++j) {
        input_data.push_back(v);
        weights.push_back(0.1);
      }
    }
  }
  f.estimate(input_data, weights); // --- run the estimation based on weighted␣
↪observations
  std::cout << f << "\n";
}
```



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

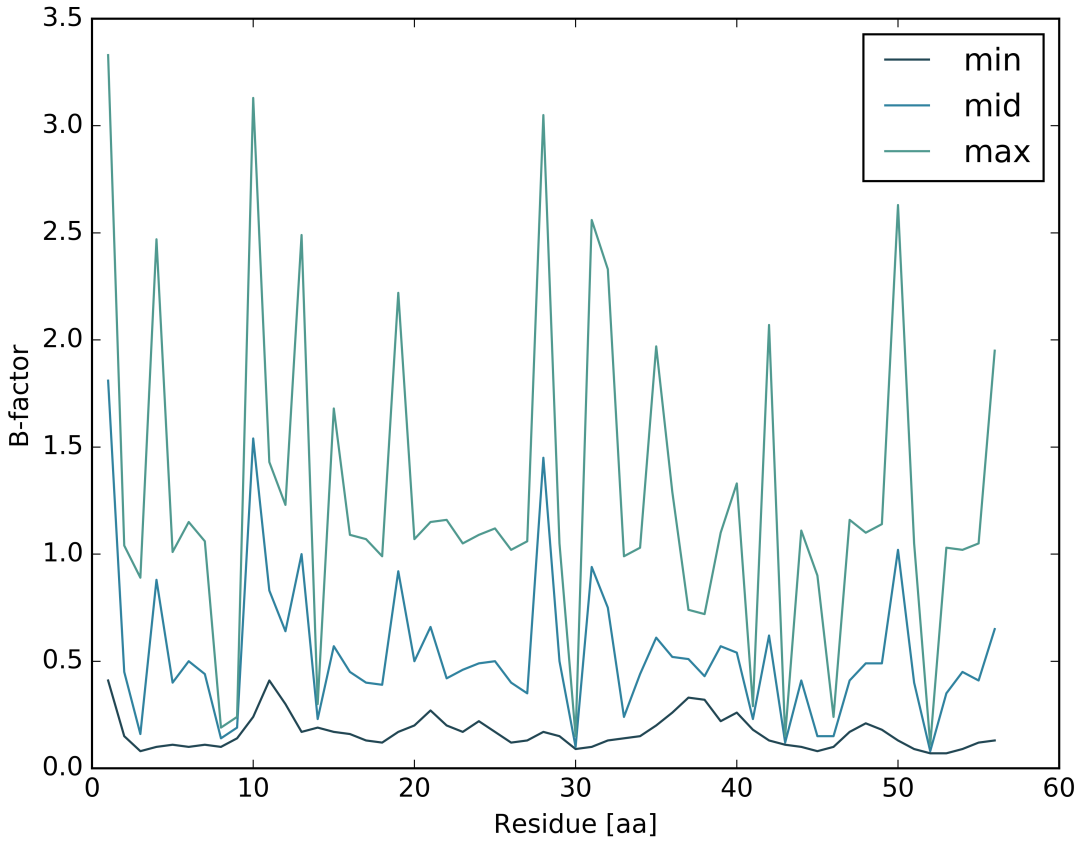
Program source:

```
#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <utils/string_utils.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";


/** @brief Reads a PDB file and per-residue statistics of B-factors
 *
 * CATEGORIES: core::data::io::Pdb;
 * KEYWORDS:   PDB input; B-factors; structure selectors
 * IMG: Bfactor_plot.png
 * IMG_ALT: B-factors of 2GB1 PDB deposit
 */
int main(const int argc, const char *argv[]) {

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

  core::data::io::Pdb reader(argv[1]); // file name (PDB format, may be gzip-ped)
  core::data::structural::Structure_SP strctr = reader.create_structure(0);
  core::data::structural::selectors::IsBB is_bb;
  core::data::structural::selectors::IsAA is_aa;
  for (auto res_it = strctr->first_residue(); res_it != strctr->last_residue(); ++res_it) {
    if (!is_aa(**res_it)) continue;
    double min = 9999.0, max = -999.0, avg = 0.0, n = 0.0;
```

(continues on next page)

(continued from previous page)

```
    for (auto atom : **res_it) {
//      if (is_bb(*atom)) continue; // --- uncommment that line to compute statistics␣
↪for side chain only
      double bf = atom->b_factor();
      if (min > bf) min = bf;
      if (max < bf) max = bf;
      avg += bf;
      n += 1.0;
    }
    std::cout << (*res_it)->residue_type().code1 << " " <<
    utils::string_format("%4d %5.2f %5.2f %5.2f\n", (*res_it)->id(), min, avg / n,␣
↪max);
  }
}
```



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:

```
#include <iostream>
#include <iomanip>
#include <core/data/io/Pdb.hh>

#include <core/calc/structural/protein_angles.hh>
#include <core/chemical/ChiAnglesDefinition.hh>
#include <utils/exit.hh>
#include <core/calc/structural/angles.hh>

std::string program_info = R"(

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:
```

(continues on next page)

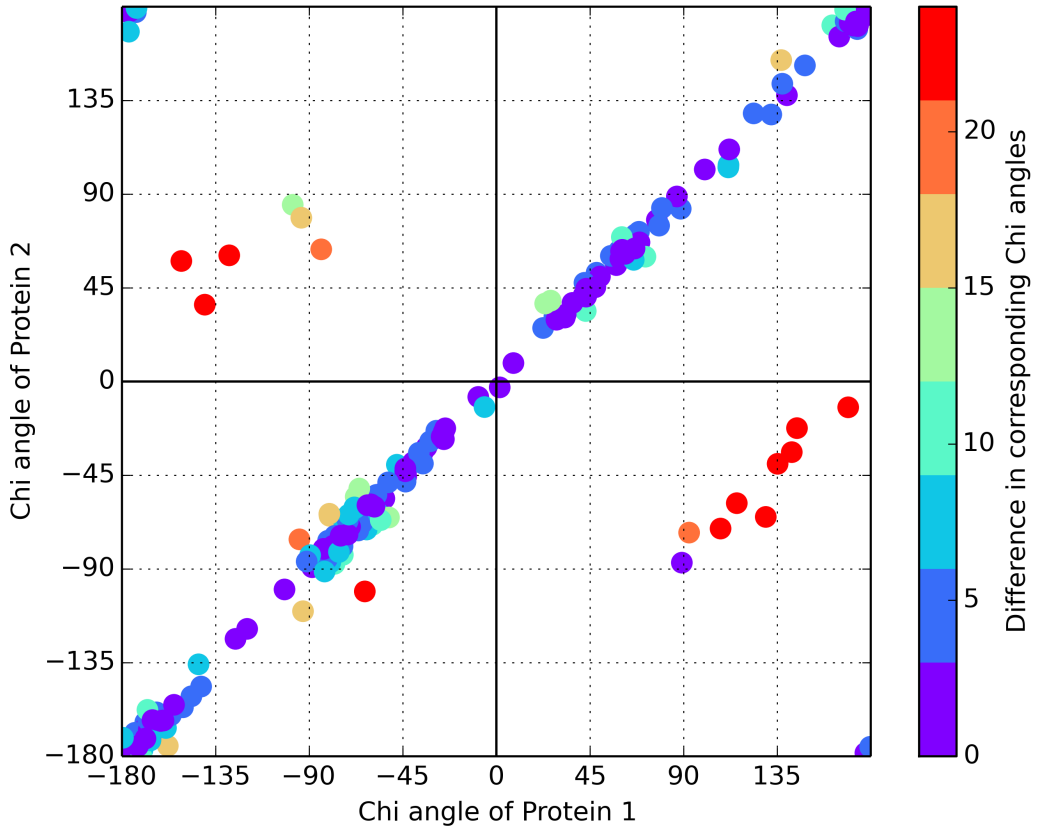
(continued from previous page)

```
    ex_chi_correlation 1bgx_aligned.pdb 1xo1_aligned.pdb

)";

/** @brief Calculates correlation between Chi dihedral angles measured in two different protein structures.
 *
 * CATEGORIES: core/calc/structural/evaluate_chi;
 * KEYWORDS:   PDB input; structural properties; rotamers
 * IMG: ex_chi_correlation_plot.png
 * IMG_ALT: Example correlation of chi angles between two homologus structures
 */
int main(const int argc, const char* argv[]) {

  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  using namespace core::data::structural;
  core::data::io::Pdb readerA(argv[1]); // file name (PDB format, may be gzip-ped)
  Structure_SP proteinA = readerA.create_structure(0);
  core::data::io::Pdb readerB(argv[2]);
  Structure_SP proteinB = readerB.create_structure(0);

  std::vector<std::vector<double>> chiA, chiB;
  std::vector<std::string> labels; // for nice output on a screen
  std::vector<std::string> mpt_annotationsA; // MPT string - one per residue only (for other lines of a given residue empty strings are inserted)
  std::vector<std::string> mpt_annotationsB;

  if (proteinA->count_residues() != proteinB->count_residues()) {
    std::cerr << "The two input PDB files should contain only the aligned parts of input proteins and be equal in length!\n";
    return 0;
  }
  auto a_res_it = proteinA->first_const_residue();
  auto b_res_it = proteinB->first_const_residue();
  while(a_res_it!=proteinA->last_const_residue()) {

    if ((*a_res_it)->residue_type() == (*b_res_it)->residue_type()) { // check chi angles

      std::vector<double> ca, cb;
      for (core::index2 k = 1; k <= core::chemical::ChiAnglesDefinition::count_chi_angles((*a_res_it)->residue_type()); ++k) {
        try {
          double aA = core::calc::structural::evaluate_chi((**a_res_it), k);
          double aB = core::calc::structural::evaluate_chi((**b_res_it), k);
          ca.push_back(aA);
          cb.push_back(aB);
          labels.push_back(utils::string_format("%4d%c %4d%c %c %1d", (*a_res_it)->id(), (*a_res_it)->icode(),
              (*b_res_it)->id(), (*b_res_it)->icode(), (*a_res_it)->residue_type().code1, k));
        } catch (utils::exceptions::AtomNotFound e) {
          std::cerr << e.what() << "\n";
          std::cerr << "Can't define chi angle for residue " << (**a_res_it) << "\n";
        }
```

(continues on next page)

(continued from previous page)

```
      }
      if (ca.size() == cb.size() && ca.size() > 0) {
        chiA.push_back(ca);
        chiB.push_back(cb);
        mpt_annotationsA.push_back(core::calc::structural::define_rotamer(ca));
        mpt_annotationsB.push_back(core::calc::structural::define_rotamer(cb));
      }
    }
    ++a_res_it;
    ++b_res_it;
  } // end of while loop over residues

  std::cout << "#ires jres aa k  ichi_k   jchi_k delta(chi) irot jrot\n";
  double err = 0.0;
  core::index2 n_reoriented = 0;

  size_t ilabel=0;
  for(size_t ires=0;ires<chiA.size();++ires) {
    for (size_t i = 0; i < chiA[ires].size(); ++i) {
      double e = fabs(chiA[ires][i] - chiB[ires][i]);
      if (e > M_PI) e = 2.0 * M_PI - e;
      if (e > 0.523) ++n_reoriented; // --- i.e. 30 degrees of error
      std::cout << labels[ilabel] << utils::string_format("%8.2f %8.2f %8.2f",
        core::calc::structural::to_degrees(chiA[ires][i]), core::calc::structural::to_
↪degrees(chiB[ires][i]),
        core::calc::structural::to_degrees(e));
      if (i == 0)
        std::cout << std::setw(5) << mpt_annotationsA[ires] << std::setw(5) << mpt_
↪annotationsB[ires] << "\n";
      else std::cout << "\n";
      err += e;
      ++ilabel;
    }
  }
  std::cout <<"# avg_err, n_diff, n_res: " << err / double(chiA.size()) << " "<<n_
↪reoriented<<" "<<chiA.size()<<"\n";
}
```



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

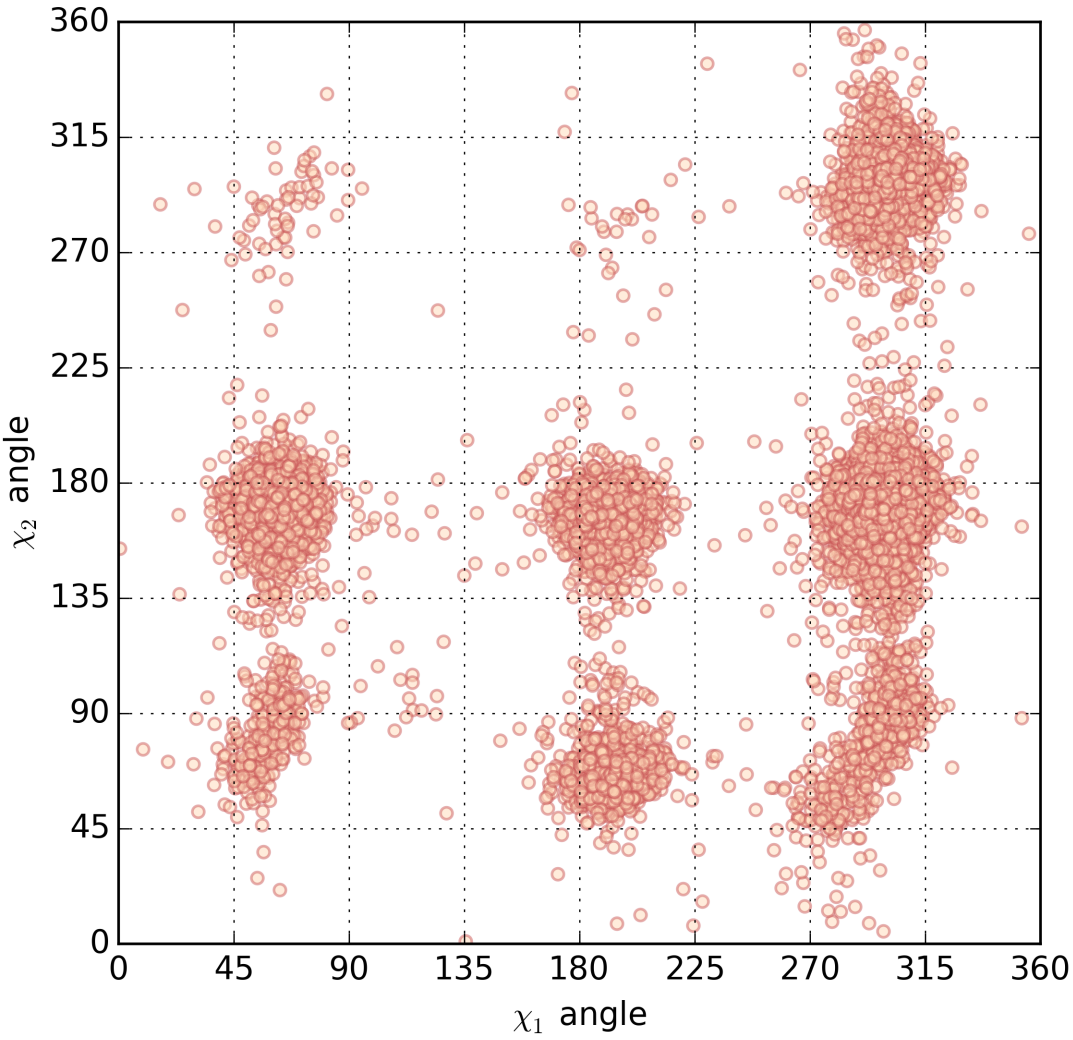
Program source:

```
#include <iostream>

#include <core/data/io/Pdb.hh>

#include <core/chemical/ChiAnglesDefinition.hh>
#include <core/calc/structural/protein_angles.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Calculates all side chain Chi dihedral angles for the input protein structure
USAGE:
    ex_evaluate_chi input.pdb
EXAMPLE:
    ex_evaluate_chi 2kwi.pdb

)";

/** @brief Calculates all side chain Chi dihedral angles for the input protein␣
↪structure
 *
 * CATEGORIES: core::chemical::ChiAnglesDefinition; core::calc::structural::evaluate_
↪chi()
 * KEYWORDS:   PDB input; structural properties; structure validation
 * IMG: ex_evaluate_chi.png
 * IMG_ALT: Chi1-Chi2 statistics for ILE residue
 */
int main(const int argc, const char *argv[]) {

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

  core::data::io::Pdb reader(argv[1]);  // Create a PDB reader for a given file
  core::data::structural::Structure_SP str = reader.create_structure(
    0); // Create a structure object from the first model

  for (auto ires = str->first_residue(); ires != str->last_residue(); ++ires) { //␣
↪iterate over all residues
    std::string line = utils::string_format("%4d %3s %4s", (*ires)->id(), (*ires)->
↪residue_type().code3.c_str(), (*ires)->owner()->id().c_str());

    try {
      for (unsigned short i = 1; i <= core::chemical::ChiAnglesDefinition::count_chi_
↪angles((*ires)->residue_type()); ++i)
        line += utils::string_format(" %6.1f", core::calc::structural::evaluate_
↪chi(**ires, i) * 180.0 / 3.1415);
```

(continues on next page)

(continued from previous page)

```
      std::cout << line << "\n";
    } catch(const std::exception& e) {
          std::cerr << "Skipping incomplete residue: "<<line<<"\n";
    }
  }
}
```



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

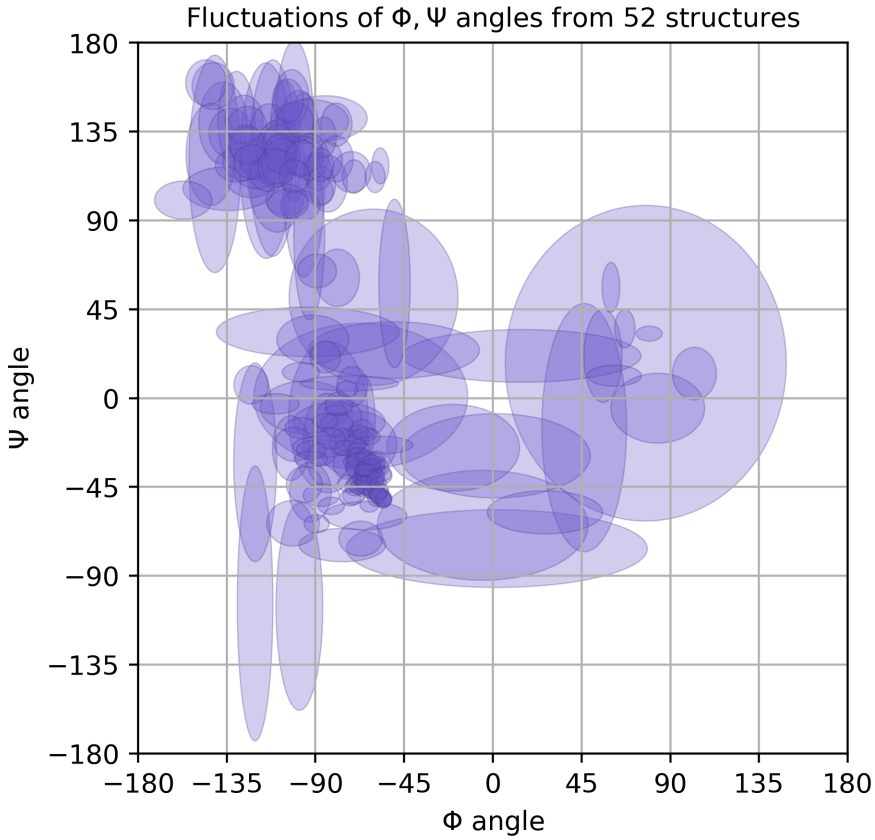
Program source:

```
#include <iostream>

#include <iostream>
#include <core/data/io/Pdb.hh>
#include <core/calc/structural/local_backbone_geometry.hh>
#include <utils/exit.hh>
#include <core/data/structural/ResidueSegmentProvider.hh>
#include <core/data/structural/selectors/ResidueSegmentSelector.hh>
#include <core/data/structural/selectors/SelectPlanarCAGeometry.hh>
#include <core/protocols/selection_protocols.hh>

std::string program_info = R"(

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

)";

/** @brief Calculates Phi,Psi angles (Ramachandran map) for the input protein structure
 *
 * CATEGORIES: core::calc::structural::LocalBackboneProperties
 * KEYWORDS:   PDB input; structural properties; structure validation
 * IMG: phi_psi_scatterplot.png
 * IMG_ALT: Phi,Psi values plotted for every residue of 2KWI PDB deposit; radius of a circle denotes standard deviation calculated from the NMR ensemble
 */
```

(continues on next page)

(continued from previous page)

```
int main(const int argc, const char *argv[]) {

  using namespace core::calc::structural;
  using namespace core::data::structural;
  using namespace core::data::io;

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

  core::data::io::Pdb reader(argv[1], keep_all, keep_all, true);  // Create a PDB reader for a given file
  selectors::HasProperlyConnectedCA is_connected;
  core::data::structural::selectors::ResidueHasAllHeavyAtoms check_atoms;
  core::data::structural::selectors::SelectPlanarCAGeometry if_flat;
  Phi phi(1);
  Psi psi(1);
  for(int i=0;i<reader.count_models();++i) {
    core::data::structural::Structure_SP str = reader.create_structure(i);
    if (argc==3) {
      core::data::structural::selectors::ChainSelector pick_chain(argv[2]);
      core::protocols::keep_selected_chains(pick_chain, *str);
    }
    ResidueSegmentProvider rsp(str, 3);
    while (rsp.has_next()) {
      const ResidueSegment_SP seg = rsp.next();
      if (is_connected(*seg)) {
        for(int i=0;i<3;++i) {
          if (if_flat((*seg)[i])) break;
          if (!check_atoms((*seg)[i])) break;
        }
        const auto &res = *(*seg)[1];
        std::cout << utils::string_format("%4s %s %4d %3s  ", str->code().c_str(), res.owner()->id().c_str(), res.id(), res.residue_type().code3.c_str());
        std::cout << seg->sequence()->sequence << " " << seg->sequence()->str()<< " ";
        std::cout << utils::string_format(phi.format(), phi(*seg)) << " ";
        std::cout << utils::string_format(psi.format(), psi(*seg)) << "\n";
      }
    }
  }
}
```



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:

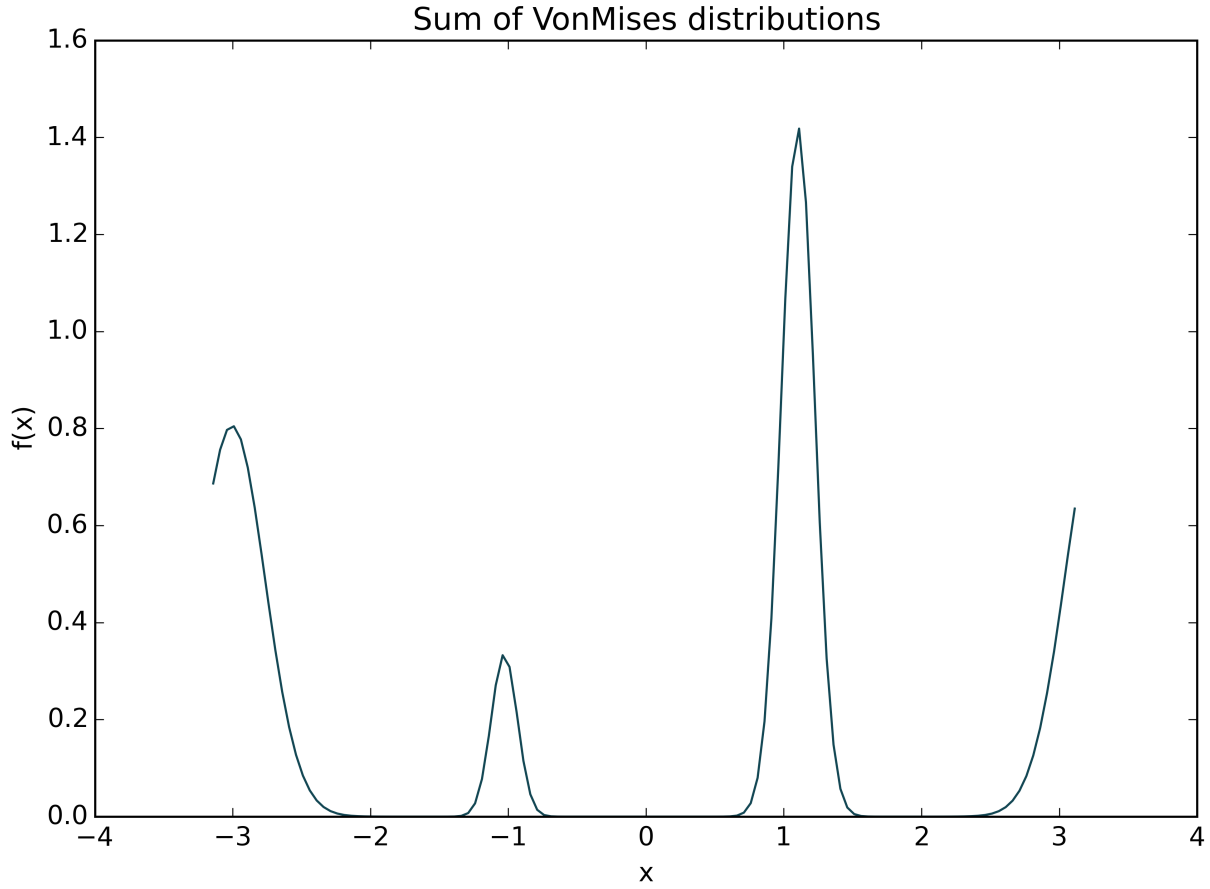
- core/calc/statistics/VonMisesDistribution

Output files:

- stdout.out

Program source:

```
#include <iostream>
#include <random>
#include <cstdlib>

#include <core/calc/statistics/VonMisesDistribution.hh>
#include <core/calc/statistics/Random.hh>

std::string program_info = R"(

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
)";

/** @brief Example which evaluates a mixture of Von Mises distribution so it can be plotted nicely
 * CATEGORIES: core/calc/statistics/VonMisesDistribution
 * KEYWORDS: statistics
 * IMG: ex_plot_VonMises_mixture.png
 * IMG_ALT: Mixture of von Mises functions plotted
 */
int main(const int argc, const char *argv[]) {

  using namespace core::calc::statistics;

  std::vector<double> factors;
  std::vector<VonMisesDistribution> components;
  for (size_t i = 1; i < argc; i += 3) {
    factors.push_back(atof(argv[i]));
    components.push_back(VonMisesDistribution{atof(argv[i + 1]), atof(argv[i + 2])});
  }

  for (double x = -M_PI; x <= M_PI; x += M_PI / 62.8) {
    double val = 0;
    for (size_t i = 0; i < components.size(); ++i) val += factors[i] * components[i].evaluate(x);
    std::cout << utils::string_format("%6.3f %9f\n", x, val);
  }
}
```



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:

```
#include <iostream>
#include <core/data/basic/Array2DSymmetric.hh>
#include <core/calc/statistics/Random.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Simple test for Array2DSymmetric class.
 *
 * The test fills a mtrix with random data and prints it on the screen.
 *
 * CATEGORIES: core::data::basic::Array2DSymmetric
 * KEYWORDS:   data structures; random numbers
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  core::calc::statistics::Random r = core::calc::statistics::Random::get();
  std::uniform_int_distribution<core::index1> uniform_bytes;
  r.seed(12345);
  core::data::basic::Array2DSymmetric<core::index1> m(10);
  for (core::index4 n = 0; n < 1000; ++n) {
    core::index1 i = uniform_bytes(r) % 10;
    core::index1 j = uniform_bytes(r) % 10;
    m.set(i, j, uniform_bytes(r));
  }
  m.print("%4d",std::cout);
}
```

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:

```
#include <iostream>
#include <iomanip>        // for std::setw()
#include <ios>               // for std::boolalpha
#include <sstream>
#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Simple example showing how to use atom selectors.
```

(continues on next page)

(continued from previous page)

```

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
)";

/** @brief Demonstrates how to use atom selectors.
 *
 * CATEGORIES: core/data/structural/structure_selectors.hh
 * KEYWORDS:   PDB input; structure selectors
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

    std::stringstream in(thr);                // Create an input stream that will provide data from a string
    core::data::io::Pdb reader(in,      // data stream
        core::data::io::keep_all);      // a predicate to read ALL the ATOM lines (hydrogens are excluded by default)
    core::data::structural::Structure_SP strctr = reader.create_structure(0);

    core::data::structural::selectors::IsCA ca_test;
    core::data::structural::selectors::IsCB cb_test;
    core::data::structural::selectors::IsBB bb_test;
    core::data::structural::selectors::IsElement is_H("H");
    core::data::structural::selectors::IsBBCB bb_cb_test;
```

(continues on next page)

(continued from previous page)

```
    core::data::structural::selectors::InverseAtomSelector not_bb_cb(bb_cb_test);
    core::data::structural::selectors::IsNamedAtom is_og1(" OG1"); // note the␣
↪padding for four characters!
    std::cout <<"atom  is_CA     is_CB     is_BB     is_H     is_OG1  is_bb_CB !is_bb_
↪CB\n";
    for(auto ai = strctr->first_atom(); ai != strctr->last_atom(); ++ai)
      std::cout << (*ai)->atom_name()<<"  "
                << std::setw(5) << std::boolalpha << ca_test(**ai)<<"  "
                << std::setw(5) << std::boolalpha << cb_test(**ai)<<"  "
                << std::setw(5) << std::boolalpha << bb_test(**ai)<<"  "
                << std::setw(5) << std::boolalpha << is_H(**ai)<<"  "
                << std::setw(5) << std::boolalpha << is_og1(**ai)<<"  "
                << std::setw(5) << std::boolalpha << bb_cb_test(**ai)<<" "
                << std::setw(5) << std::boolalpha << not_bb_cb(**ai)<<"\n";
}
```

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:

```
#include <iostream>
#include <iomanip>

#include <core/chemical/AtomicElement.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Unit test which shows how to use AtomicElement class. It prints all the element names

USAGE:
./ex_AtomicElement

)";

/** @brief Example showing how to use AtomicElement class
 *
 * CATEGORIES: core::chemical::AtomicElement
 * KEYWORDS: chemical elements
 */
int main(int argc, char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::chemical;

  if (argc < 2)
    for (const std::pair<std::string, AtomicElement> &e : AtomicElement::elements_by_
↪symbol)
```

(continues on next page)

(continued from previous page)

```
      std::cout << e.second << std::endl;
  else
    for (int i = 1; i < argc; i++) std::cout << AtomicElement::by_symbol(argv[i]) <<
↪"\n";
}
```

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:

```
#include <core/data/io/Pdb.hh>
#include <utils/exit.hh>
#include <core/algorithms/graph_algorithms.hh>
#include <core/data/structural/BetaStructuresGraph.hh>
#include <core/calc/structural/ProteinArchitecture.hh>

std::string program_info = R"(

Reads a PDB file, creates a BetaStructuresGraph for it and finds all strands as␣
↪connected components of that graph
USAGE:
    ex_BetaStructuresGraph 5edw.pdb

)";

/** @brief Creates a BetaStructuresGraph and finds all strands
 *
 * CATEGORIES: core::data::structural::BetaStructuresGraph
 * KEYWORDS:   PDB input
 */
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::structural;
```

(continues on next page)

(continued from previous page)

```
  using namespace core::data::io;

  core::data::io::Pdb reader(argv[1],is_not_alternative,only_ss_from_header, true);
  core::data::structural::Structure_SP strctr = reader.create_structure(0);

  core::calc::structural::ProteinArchitecture a(*strctr);
  BetaStructuresGraph_SP g = a.create_strand_graph();
  auto sheets = core::algorithms::connected_components<BetaStructuresGraph, Strand_SP,
↪ StrandPairing_SP>(*g);
  int cnt = 0;
  for(const auto & sheet: sheets) {
    std::cout << utils::string_format("-------------- Sheet %d -------------------\n",
↪++cnt);
    for(auto it=sheet->cbegin_strand();it!=sheet->cend_strand();++it)
      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:

```
#include <iostream>
#include <core/BioShellVersion.hh>
#include <utils/exit.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

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

)";

/** @brief Test for BioShellVersion class prints the BioShell version info
 *
 * CATEGORIES: core/BioShellVersion
 * KEYWORDS:   bioshell
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  std::cout << "BioShell boilerplate:\n";
  std::cout << core::BioShellVersion() << "\n";
}
```

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:

```
#include <math.h>

#include <iostream>
#include <random>

#include <core/data/io/DataTable.hh>
#include <core/calc/statistics/Random.hh>
#include <core/calc/statistics/BivariateNormal.hh>
#include <core/calc/statistics/RobustDistributionDecorator.hh>

std::string program_info = R"(

Estimates parameters of a two-dimensional Gaussian distribution
```

(continues on next page)

(continued from previous page)

```

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.

)";

/** @brief Estimates parameters of a two-dimensional Gaussian distribution
 *
 * CATEGORIES: core::calc::statistics::BivariateNormal; core::calc::statistics::Random
 * KEYWORDS:   statistics; random numbers; estimation
 */
int main(const int argc, const char* argv[]) {

  using namespace core::calc::statistics;

  std::vector<std::vector<double> > data_2D;
  std::vector<double> row(2);
  if (argc == 1) { // --- No input file? Generate random data for the test

    std::cerr << program_info;

    Random rd = core::calc::statistics::Random::get();
    rd.seed(12345); // --- seed the generator for repeatable results
    unsigned N = 100000; //--- the number of random points to use in tests
    core::calc::statistics::NormalRandomDistribution<double> nX(1.0, 2.5);
    core::calc::statistics::NormalRandomDistribution<double> nY(2.0, 0.7);

    for (unsigned i = 0; i < N; ++i) { // --- get a random sample in 2D
      double x = nX(rd);
      double y = nY(rd);
      row[0] = x - y; // --- make X variable correlated with Y
      row[1] = x + y;
      data_2D.push_back(row);
    }
  } else {
    core::data::io::DataTable in_data(argv[1]);
    int column_x_id = 0, column_y_id = 1;
    if (argc > 3) {
      column_x_id = utils::from_string<int>(argv[2]);
      column_y_id = utils::from_string<int>(argv[3]);
    }
    for (const auto &data_row : in_data) {
      row[0] = data_row.get<double>(column_x_id);
      row[1] = data_row.get<double>(column_y_id);
```

(continues on next page)

(continued from previous page)

```
      data_2D.push_back(row);
    }
  }

  std::vector<double> initial_parameters{0.0, 0.0, 1.0, 1.0, 1.0};
  // --- Here we declare a 2D normal distribution ...
  core::calc::statistics::BivariateNormal n(initial_parameters);
  // ... and estimate its parameters
  const std::vector<double> & params = n.estimate(data_2D);
  // show the estimated parameters of the distribution
  std::cout << "          estimated parameters: " << params[0] << " " << params[1] <<
↪" " << params[2] << " " << params[3] << " " << params[4] << "\n";
  // ... and estimate its parameters
  core::calc::statistics::RobustDistributionDecorator<BivariateNormal> rn(initial_
↪parameters, 0.05);
  const std::vector<double> & params_r = rn.estimate(data_2D);
  // show the estimated parameters of the distribution
  std::cout << " estimated parameters (robust): " << params_r[0] << " " << params_
↪r[1] << " " << params_r[2] << " " << params_r[3] << " " << params_r[4] << "\n";
}
```

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:

```
#include <iostream>
#include <random>

#include <core/index.hh>
#include <core/calc/statistics/Random.hh>
#include <core/data/basic/BoundedPriorityQueue.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

using namespace core::data::basic;

/** @brief Simple demo for BoundedPriorityQueue class
```

(continues on next page)

(continued from previous page)

```
 *
 * This program creates a BoundedPriorityQueue and fills it with random numbers.
 * When printed, they should be ordered descending
 *
 * CATEGORIES: core::data::basic::BoundedPriorityQueue
 * KEYWORDS:   algorithms; data structures; BoundedPriorityQueue
 */
int main(int cnt, char *argv[]) {

  if ((cnt > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  // ---------- test on real values
  typedef std::function<bool(const double, const double)> ComparatorType;
  core::data::basic::BoundedPriorityQueue<double, ComparatorType, ComparatorType> q(
    [&](double x, double y) { return x > y; },
    [&](double x, double y) { return x == y; }, 10, 20, -std::numeric_limits<double>
↪::max());
  core::calc::statistics::Random r = core::calc::statistics::Random::get();
  std::uniform_real_distribution<float> flat_f(0, 10.0);
  for (core::index4 i = 0; i < 30; ++i)
    q.push(flat_f(r));
  for (core::index4 i = 1; i < q.size(); ++i) {
    std::cout << q[i] << " ";
    if (q[i - 1] < q[i])
      std::cerr
        << utils::string_format("Incorrect ordering in a bounded priority queue, %f␣
↪before %f\n", q[i - 1], q[i]);
  }

  std::cout << "\n";

  // ---------- test on integers
  typedef std::function<bool(int, int)> ComparatorTypeI;
  core::data::basic::BoundedPriorityQueue<int, ComparatorTypeI, ComparatorTypeI> q_i(
    [&](int x, int y) { return x > y; },
    [&](int x, int y) { return x == y; }, 10, 20, -std::numeric_limits<int>::max());
  std::uniform_int_distribution<int> flat(0, 20);
  for (core::index4 i = 0; i < 30; ++i)
    q_i.push(flat(r));
  for (core::index4 i = 1; i < q_i.size(); ++i) {
    std::cout << q_i[i] << " ";
    if (q_i[i - 1] < q_i[i])
      std::cerr
        << utils::string_format("Incorrect ordering in a bounded priority queue, %f␣
↪before %f\n", q_i[i - 1], q_i[i]);
  }
}
```

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:

```
#include <iostream>
#include <core/data/basic/Vec3Cubic.hh>
#include <simulations/systems/BuildPolymerChain.hh>
#include <simulations/systems/SingleAtomType.hh>
#include <simulations/systems/CartesianChains.hh>
#include <simulations/observers/cartesian/PdbObserver.hh>
#include <simulations/observers/cartesian/ExplicitPdbFormatter.hh>


#include <utils/exit.hh>

using namespace simulations::systems;
using namespace core::data::structural;
using core::data::basic::Vec3Cubic;


std::string program_info = R"(

Creates a mixture of simple polymer chains in a periodic box

USAGE:
    ./ex_BuildPolymerChain box_width n_chains n_atoms_in_chain

)";

/* @brief Creates a mixture of simple polymer chains in a periodic box
```

(continues on next page)

(continued from previous page)

```
 *
 * CATEGORIES: simulations::systems::BuildPolymerChain
 * KEYWORDS:   Chain; ResidueChain; Polymer; Vec3Cubic
 */
int main(const int argc, const char *argv[]) {

  if (argc < 4) utils::exit_OK_with_message(program_info);

  double box = atof(argv[1]);
  core::index2 n_chains = atoi(argv[2]);
  core::index2 n_res_each = atoi(argv[3]);

  // --- here we create a Structure object of n_chains polyalanine chains
  Structure_SP starting_structure = std::make_shared<Structure>("");
  std::string sequence(n_res_each,'A'); // --- We create a polyalanine chain, the sequence is made by many 'A's
  for (core::index2 i = 0; i < n_chains; ++i)
    starting_structure->push_back( Chain::create_ca_chain(sequence, std::string{utils::letters[i]} )); // --- A + 1 makes the chain code

  // --- we have to renumber atoms so the indexes are consistent in the whole structure
  core::index4 i_atom = 0;
  for(Chain_SP m : *starting_structure)
    std::for_each(m->first_atom(), m->last_atom(), [&](PdbAtom_SP e) {(e)->id(++i_atom);});
 // Vec3Cubic::set_box_len(box); // --- set periodic box width
  std::shared_ptr<AtomTypingInterface> atom_typing = std::make_shared<SingleAtomType>(); // --- simplest atom typing possible
  CartesianChains chains(atom_typing,*starting_structure);
  BuildPolymerChain chain_builder(chains);
  chain_builder.generate(3.8,5.5);
  core::index4 ai = 0;
    std::shared_ptr<simulations::observers::cartesian::AbstractPdbFormatter> fmt = std::make_shared<simulations::observers::cartesian::ExplicitPdbFormatter>(*starting_structure);
    simulations::observers::cartesian::PdbObserver start(chains, fmt, "");
    start.observe();
}
```

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:

```
#include <iostream>
#include <vector>
#include <map>

#include <core/calc/statistics/Cart.hh>
#include <core/algorithms/basic_algorithms.hh>

using namespace core::calc::statistics;
using namespace core::data::io;
using namespace core::data::basic;

utils::Logger logs("ex_Cart");

/** @brief Shows how to use CART classification model
 *
 * CATEGORIES: core::calc::statistics::Cart
 * KEYWORDS:   CART; observer
 */
int main(const int argc, const char *argv[]) {

  DataTable dt;
  dt.load(argv[1]);

  std::vector<LabelledObservationVector_SP> observations;
  std::vector<std::string> class_names;
  std::vector<core::index2> class_ids;
  std::map<std::string, core::index2> class_to_id;
```

(continues on next page)

(continued from previous page)

```

  // --- First find distinct labels
  core::index2 i_class = 0;
  for (const TableRow &tr : dt) {
    if (class_to_id.find(tr.back()) == class_to_id.end()) {
      class_ids.push_back(i_class);
      class_to_id[tr.back()] = i_class;
      ++i_class;
    }
  }

  for (const TableRow &tr : dt)
    observations.push_back(std::make_shared<LabelledObservationVector>(tr, class_to_
↪id[tr.back()], 0, tr.size() - 2));

  // --- print some debug info : known classes etc.
  logs << utils::LogLevel::INFO << "classification into " << class_ids.size() << "␣
↪classes\n";
  if (logs.is_logable(utils::LogLevel::INFO)) {
    logs << utils::LogLevel::INFO << "Known classes:\n";
    core::index1 icol = 0;
    for (auto c:class_to_id) {
      logs << c.first << " ";
      ++icol;
      if (icol % 10 == 0) logs << "\n";
    }
    logs << "\n";
  }

  // --- create the CART classifier and train it
  Cart cart(class_ids);
  cart.train(observations);
  std::cout << cart;

  // --- test the classifier for the training data set
  core::index4 n_ok = 0;
  for(const auto o : observations) {
    n_ok += (cart.classify(o) == o->label());
  }
  std::cout << utils::string_format("# classification test:\n# success rate: %d of %d␣
↪(%6.2f%%)\n", n_ok, observations.size(),
    100.0*n_ok / float(observations.size()));
}
```

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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/PdbAtom.hh>
#include <core/calc/structural/transformations/Rototranslation.hh>
#include <core/calc/structural/transformations/transformation_utils.hh>
#include <core/calc/structural/transformations/CartesianToSpherical.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

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
```

(continues on next page)

(continued from previous page)

```
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)";

/** @brief Calculates spherical coordinates using BioShell and 'by hand' to check of it works
 *
 * CATEGORIES: core/calc/structural/CartesianToSpherical;
 * KEYWORDS:   internal coordinates
 */
int main(const int argc, const char *argv[]) {

  using namespace core::data::structural;
  using namespace core::calc::structural::transformations;

  std::stringstream in_stream(input_pdb); // --- Create an input stream from the text
  core::data::io::Pdb reader(in_stream, core::data::io::keep_all); // --- read from this stream
  core::data::structural::Structure_SP strctr = reader.create_structure(0); // --- create a structure object

  auto residue = *(strctr->first_residue()); // --- get the residue ...
  PdbAtom_SP n = residue->find_atom(" N  "); // --- and extract three atoms
  PdbAtom_SP ca = residue->find_atom(" CA ");// ---  to form a local coordinate system (LCS)
  PdbAtom_SP c = residue->find_atom(" C  ");
  PdbAtom_SP cb = residue->find_atom(" CB "); // --- CB will be transformed

  Rototranslation_SP rt = local_coordinates_three_atoms(*n, *ca, *c);
  CartesianToSpherical to_spherical;

  Vec3 cb_local, cb_spherical;
  rt->apply(*cb, cb_local);
  to_spherical.apply(cb_local, cb_spherical);

  double r = cb_local.length();
  double theta = acos(cb_local.z / r);
  double phi = atan2(cb_local.y, cb_local.x);

  std::cout << "local computed by BioShell: " << cb_local << "\n";
  std::cout << "spherical by BioShell:      " << cb_spherical << "\n";
  std::cout << "spherical computed here:    " << utils::string_format("%8.3f %8.3f %8.3f\n", r, theta, phi);
  double x = r * sin(theta) * cos(phi);
  double y = r * sin(theta) * sin(phi);
  double z = r * cos(theta);
  std::cout << "local from inversion:       " << utils::string_format("%8.3f %8.3f %8.3f\n", x, y, z);
  to_spherical.apply_inverse(cb_spherical);
  std::cout << "local by BioShell   :       " << cb_spherical << "\n";
}
```

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:

```
#include <iostream>

#include <core/chemical/ChiAnglesDefinition.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Shows how to look up information on Chi angle definitions
 *
 * This example prints how many Chi angles are defined for ARG and wish atoms define Chi2 of TRP
 *
 * CATEGORIES: core::chemical::ChiAnglesDefinition
 * KEYWORDS:   structural properties
 */
int main(const int argc, const char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

```

(continues on next page)

(continued from previous page)

```
  // List atoms that define the second Chi angle in TRP residue
  std::cout << "Chi_2 in TRP:";
  for (const std::string &a : core::chemical::ChiAnglesDefinition::chi_angle_atoms("TRP", 2)) // "TRP" defines a residue, "2" stands for Chi_2
    std::cout << a << " ";
  std::cout << "\n";
  const core::chemical::Monomer &m = core::chemical::Monomer::ARG; // Create a local reference to ARG monomer (just to make the following lines shorter)
  std::cout << "\nAll Chi angles for in " << m.code3 << " :\n";
  for (unsigned short i = 1; i <= core::chemical::ChiAnglesDefinition::count_chi_angles(m); ++i) {  // Count how many Chi angles ARG has
    std::cout << "Chi" << i << " ";
    for (const std::string &a : core::chemical::ChiAnglesDefinition::chi_angle_atoms(m, i)) // List atoms for each of them
      std::cout << a << " ";
    std::cout << "\n";
  }
}
```

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:

```
#include <core/data/io/Cif.hh>
#include <utils/Logger.hh>
#include <utils/LogManager.hh>

#include <utils/exit.hh>

std::string program_info = R"(

Unit test which shows how to read CIF files.

USAGE:
    ex_Cif file.cif
EXAMPLE:
    ex_Cif AA3.cif

)";

/** @brief ex_Cif tests reading CIF files
 *
 * CATEGORIES: core/data/io/Cif
 * KEYWORDS:   CIF input
 */
```

(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␣
   ↪missing program parameter
26
27   utils::LogManager::FINEST(); // --- INFO is the default logging level; set it to␣
   ↪FINE to see more
28   core::data::io::Cif reader(argv[1]);
29   std::cout << reader;
30 }
```

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:

```
#include <memory>
#include <iostream>
#include <random>
#include <core/algorithms/Combinations.hh>

#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Unit test for BioShell's combination generator prints all possible tripeptides by␣
↪taking all 3-element
combinations of 20-elements set.

USAGE:
./ex_Combinations

)";


/** @brief A simple example shows how to generate Combination
 *
 * The program generates all possible tripeptides as ${20}\choose {3}$ combinations
 *
 * CATEGORIES: core::algorithms::Combination;
 * KEYWORDS:  algorithms; random
 */
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
      ","GLY","HIS","ILE","LEU","LYS","MET","PRO","SER",
33    "THR","TYR","TRP","VAL"};
34
35    std::vector<std::string> a_combination(3);
36    core::algorithms::Combinations<std::string> generator(3,amino_acids);
37    int cnt = 0;
38    while (generator.next(a_combination)) {
39      std::cout << a_combination[0] << " " << a_combination[1] << " " << a_
      combination[2] << "\n";
40      ++cnt;
41    }
42    std::cout << "# " << cnt << " combinations generated\n";
43  }
```

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:

```
#include <iostream>
#include <core/data/io/fasta_io.hh>
#include <core/data/io/DsspData.hh>
#include <utils/exit.hh>

std::string program_info = R"(

ex_DsspData reads a DSSP file and writes secondary structure in FASTA format

USAGE:
    ex_DsspData input.dssp
EXAMPLE:
    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␣
   ↪each chain in FASTA format.
18  *
19  * @see ex_dssp_to_ss2.cc converts DSSP to SS2 format
20  *
21  * CATEGORIES: core/data/io/DsspData
22  * KEYWORDS:   DSSP; FASTA output; Structure; secondary structure; Format conversion
23  */
24 int main(const int argc, const char* argv[]) {
25
26   if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about␣
   ↪missing program parameter
27
28   core::data::io::DsspData dssp(argv[1], true);  // --- read a DSSP file - the first␣
   ↪command line argument of the program
29   for (const auto & ss2 : dssp.sequences()) // --- for each protein sequence found in␣
   ↪the DSSP data ...
30     std::cout << core::data::io::create_fasta_string(*ss2, 80) << "\n" // --- print␣
   ↪the sequence as FASTA
31         << core::data::io::create_fasta_secondary_string(*ss2, 80) << "\n";  // ---␣
   ↪print the secondary structure as FASTA
32 }
```

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 small500_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:

```
#include <core/index.hh>
#include <utils/exit.hh>
#include <core/data/io/fasta_io.hh>
#include <core/protocols/PairwiseSequenceIdentityProtocol.hh>
#include <core/protocols/FastaMatchProtocol.hh>

std::string program_info = R"(

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.
```

(continues on next page)

(continued from previous page)

```
13
14  USAGE:
15      ex_FastaMatchProtocol input.fasta [n_threads]
16
17  EXAMPLES:
18      ex_FastaMatchProtocol small500_95identical.fasta 4
19
20  REFERENCE:
21    Smith, Temple F., and Michael S. Waterman. "Identification of common molecular␣
    ↪subsequences."
22    PNAS 85 (1988): 2444-8 doi:10.1073/pnas.85.8.2444
23
24  )";
25
26  /** @brief Uses FastaMatchProtocol protocol to find similar substrings between two␣
    ↪amino acid sequences
27   *
28   * CATEGORIES: core/protocols/PairwiseSequenceIdentityProtocol.hh
29   * KEYWORDS:   FASTA input; sequence alignment; statistics
30   */
31  int main(const int argc, const char* argv[]) {
32
33    if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about␣
    ↪missing program parameter
34
35    using namespace core::data::sequence;
36    using namespace core::protocols;
37    using namespace core::alignment;
38
39    utils::Logger logs("ex_FastaMatchProtocol");
40    core::index2 n_threads = (argc > 2) ? atoi(argv[2]) : 4;
41
42    bool if_store_diagonals = false;
43    logs << utils::LogLevel::INFO << "number of threads used : " << n_threads << "\n";
44
45    core::protocols::FastaMatchProtocol protocol;
46    protocol.minimum_diagonal_coverage(0.9).shortest_match_recorded(20).minimum_
    ↪identity(0.9).longest_gap(8);
47    protocol.batch_size(10000).n_threads(n_threads).keep_alignments(if_store_diagonals).
    ↪printed_seqname_length(5);
48
49    std::vector<Sequence_SP> input_sequences;
50    core::data::io::read_fasta_file(argv[1], input_sequences);
51    for(Sequence_SP si: input_sequences) protocol.add_input_sequence(si);
52
53    auto start = std::chrono::high_resolution_clock::now(); // --- timer starts!
54    protocol.run();
55    auto end = std::chrono::high_resolution_clock::now();
56    std::chrono::duration<double> time_span = std::chrono::duration_cast
    ↪<std::chrono::duration<double>>(end - start);
57    logs << utils::LogLevel::INFO << (size_t ) protocol.n_jobs_completed()
58         << " FASTA matches calculated within " << time_span.count() << " [s]\n";
59
60    if(if_store_diagonals) {
61      std::cout << "Diagonals:\n";
62      protocol.print_header(std::cout);
63      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:

```
#include <memory>
#include <iostream>
#include <iomanip>

#include <core/algorithms/GraphWithData.hh>
#include <core/algorithms/SimpleGraph.hh>

/** @brief A unit test for SimpleGraph and GraphWithData classes
 *
 * This program creates small graph data structures and test their methods
 *
 * CATEGORIES: core/algorithms/GraphWithData; core/algorithms/SimpleGraph
 * KEYWORDS:   algorithms; data structures
 * IMG_ALT: Example tree node
 */
int main(const int argc, const char* argv[]) {

  using namespace core::algorithms;

  GraphWithData<SimpleGraph,int,std::string> g;
  g.add_vertex(0);
  g.add_vertex(1);
  g.add_vertex(2);
  g.add_vertex(3);
  g.add_edge(0,2,"0-2");
  g.add_edge(1,2,"1-2");
  g.add_edge(3,2,"3-2");
  g.add_edge(core::index4(3),core::index4(0),"0-3");

  std::cout << "# adjacency matrix\n";
  g.print_adjacency_matrix(std::cout);
  std::cout << "# are 0 and 3 connected?\n" << std::boolalpha << g.are_connected(0,3)<
↪<"\n";

  g.remove_edge(0,3);
  std::cout << "# are 0 and 3 still connected?\n" << std::boolalpha << g.are_
↪connected(0,3)<<"\n";
  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:

```
#include <iostream>
#include <sstream>
#include <vector>
#include <numeric> // for std::accumulate

#include <core/algorithms/trees/algorithms.hh>
#include <core/calc/clustering/DistanceByValues.hh>
#include <core/calc/clustering/HierarchicalCluster.hh>
#include <core/calc/clustering/HierarchicalClustering.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

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

)";

using namespace core::calc::clustering;

static utils::Logger l("ex_HierarchicalClustering");

/// Data points to be clustered
std::vector<std::string> points = {"A", "B", "C", "E", "G", "L", "M", "Q", "R", "T",
↪"X", "Y", "Z"};
```

(continues on next page)

(continued from previous page)

```

/// Distance function defined for the data above; here the alphabetic distance is used
DistanceByValues<float> calc_distance_matrix(std::vector<std::string> points) {

  DistanceByValues<float> d(points, 99.0, 99.0);
  for (size_t i = 1; i < d.n_data(); i++)
    for (size_t j = 0; j < i; j++) {
      float v = std::sqrt((points[j][0] - points[i][0]) * (points[j][0] - points[i][0]));
      d.set(i, j, v);
      d.set(j, i, v);
    }

  return d;
}

/** @brief Example showing how to use hierarchical clustering method.
 *
 * CATEGORIES: core::calc::clustering::HierarchicalClustering
 * KEYWORDS: clustering; hierarchical clustering
 */
int main(int cnt, char *argv[]) {

  if ((cnt > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  DistanceByValues<float> d = calc_distance_matrix(points);

  HierarchicalClustering<float, std::string> hac(d.labels(), "");
  hac.run_clustering(d, "");
  std::vector<std::string> elements;
  for (size_t i = 0; i < hac.count_steps(); i++) {
    elements.clear();
    std::shared_ptr<BinaryTreeNode<std::string> > c_node(std::static_pointer_cast<BinaryTreeNode<std::string>>(hac.clustering_step(i)));
    core::algorithms::trees::collect_leaf_elements(c_node, elements);
    std::string a = std::accumulate(elements.begin(), elements.end(), std::string(""));
    a.erase(std::remove(a.begin(), a.end(), ' '), a.end());
    std::sort(a.begin(), a.end());
    std::cout <<  "Clustering step: "<<i<<" : ";
    std::cout << a << "\n";
  }

  // --- write the clustering steps to a stream
  std::ostringstream sso;
  hac.write_merging_steps(sso);
  std::cout <<sso.str();

  // --- get medoid element for of the clusters created at distance d = 1.0
  auto clusters = hac.get_clusters(1.0, 2);
  for (core::index2 i = 0; i < 4; i++)
    std::cout << medoid_by_average_distance<float, std::string, DistanceByValues<float> >(clusters[i], d).medoid << "\n";
}
```

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:

```
#include <iostream>
#include <sstream>
#include <vector>
#include <numeric> // for std::accumulate

#include <core/algorithms/trees/algorithms.hh>
#include <core/algorithms/UnionFind.hh>
#include <core/calc/clustering/DistanceByValues1B.hh>
#include <core/calc/clustering/HierarchicalCluster.hh>
#include <core/calc/clustering/HierarchicalClustering1B.hh>
#include <core/BioShellEnvironment.hh>
#include <utils/LogManager.hh>
```

(continues on next page)

(continued from previous page)

```

using namespace core::calc::clustering;

static utils::Logger l("ex_HierarchicalClustering1B");

std::string program_info = R"(

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

)";

const int MAX = 10;             // --- longest sequence id string

DistanceByValues1B read_distance_matrix(const std::string &distance_file, std::set
↪<std::string> & labels) {

  core::index1 val;
  std::string line;
  std::ifstream infile(distance_file);
  char name_i[MAX], name_j[MAX];
  while (std::getline(infile, line)) {
    if(scanf_row(line, name_i, name_j, val)==3) {
      labels.insert(name_i);
      labels.insert(name_j);
    }
  }
  infile.close();
  infile.open(distance_file);

  std::vector<std::string> v( labels.begin(), labels.end() );
  core::algorithms::UnionFindSI4 uf(labels.size());
  for (const std::string &s:v) uf.add_element(s);
  DistanceByValues1B d(v);
  while (std::getline(infile, line)) {
    if (scanf_row(line, name_i, name_j, val) == 3) {
```

(continues on next page)

(continued from previous page)

```
      size_t i_index = d.at(name_i);
      size_t j_index = d.at(name_j);
      d.set(i_index, j_index, val);
      d.set(j_index, i_index, val);
      if(val==255)
        uf.union_set(i_index, j_index);
    }
  }

  std::cout << "# UnionFind groups larger than 2 (representative PDB-ID, group size):\n";
  const auto sets = uf.retrieve_sets();
  for (const auto &set: sets) {
    if (set.second.size() > 2)
      std::cout << uf.element(set.first) << " : " << set.second.size() << "\n";
  }
  std::cout << sets.size() << "\n";

  return d;
}

/** @brief Example showing how to use hierarchical clustering method - 1 byte version.
 *
 * CATEGORIES: core::calc::clustering::HierarchicalClustering
 * KEYWORDS: clustering; hierarchical clustering
 */
int main(int cnt, char *argv[]) {


  std::set<std::string> labels_set;
  DistanceByValues1B d = read_distance_matrix(argv[1], labels_set);
  std::cout << labels_set.size() << " items for clustering\n";

  utils::LogManager::INFO();

  HierarchicalClustering1B hac(d.labels(), "");
  CompleteLink1B merge;

  hac.run_clustering(d, merge);

  // --- write the clustering steps to a stream
  hac.write_merging_steps(std::cerr);
}
```

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:

```
#include <cmath>
#include <iostream>
#include <vector>

#include <core/calc/numeric/interpolators.hh>
#include <core/calc/numeric/Interpolate1D.hh>
#include <core/data/io/DataTable.hh>
#include <utils/exit.hh>

using namespace core::calc::numeric;

std::string program_info = R"(

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

)";
```

(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,
   ↪vy, cri);
41   double step = (vx.back() - vx.front())/ atof(argv[2]);
42   for (double x = vx[0]; x <= vx.back(); x += step) {
43     std::cout << x << " " << ip(x) << "\n";
44   }
45 }
```

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:

```
#include <cmath>
#include <iostream>
#include <vector>

#include <core/calc/numeric/interpolators.hh>
#include <core/calc/numeric/InterpolatePeriodic1D.hh>
#include <core/calc/structural/angles.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

using namespace core::calc::numeric;

/** @brief Simple test for interpolation of a periodic 1D function
 *
 * CATEGORIES: core::calc::numeric::InterpolatePeriodic1D;
 * KEYWORDS:   interpolation
 */
int main(int cnt, char *argv[]) {

```

(continues on next page)

(continued from previous page)

```
  if ((cnt > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  std::vector<float> x, y; // --- vectors for points used for interpolation
  double step = M_PI / 50.0;  // --- interpolation data step
  for (double ix = 0.0; ix < 2 * M_PI; ix += step) { // --- generate data for␣
↪interpolation knots
    x.push_back(ix);
    y.push_back(sin(ix));
  }

  CatmullRomInterpolator<float> cri; // --- Interpolating engine
  InterpolatePeriodic1D<std::vector<float>, float, CatmullRomInterpolator<float> >␣
↪i1d2(x, y, cri, 2*M_PI);
  double max_error = 0.0; // --- used to keep track of the maximum interpolation error
  double worst_x = 0.0;
  // --- The function has been defined in the range \f$[0,\pi]\f$
  // --- interpolation goes in the range \f$[-20\pi,40\pi]\f$
  for (float x = -20 * M_PI; x <= 40 * M_PI; x += 0.1 * step) {
    double error = fabs(sin(x) - i1d2(x));
    if (error > max_error) {
      max_error = std::max(error, max_error);
      worst_x = x;
    }
  }
  std::cout << "Maximum interpolation error:" << max_error << " for x= " << worst_x <
↪< "\n";
}
```

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:

```
#include <cmath>
#include <iostream>
#include <memory>
#include <sstream>
#include <vector>

#include <core/calc/numeric/interpolators.hh>
#include <core/calc/numeric/InterpolatePeriodic2D.hh>

#include <core/calc/structural/angles.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

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

)";

using namespace core::calc::numeric;

/** @brief Simple test for interpolation of a periodic 2D function
 *
 * CATEGORIES: core::calc::numeric::InterpolatePeriodic2D;
 * KEYWORDS:   interpolation
```

(continues on next page)

(continued from previous page)

```
 */
int main(int cnt, char* argv[]) {

  if ((cnt > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  double inter_step = core::calc::structural::to_radians(5.0); // --- interpolation step : 5 degrees
  const double EPS = 0.0001;
  std::vector<double> vx; // --- vector of function arguments: X axis
  std::vector<double> vy; // --- vector of function arguments: Y axis
  for (double x = -M_PI; x < M_PI-EPS; x += inter_step) {
    vx.push_back(x);
    vy.push_back(x); // --- we use the same values both for X and Y but in general they may differ
  }
  core::index2 nx = vx.size(); // the number of grid points

  // --- Prepare data to be interpolated
  std::shared_ptr<core::data::basic::Array2D<double>> data_periodic = std::make_shared<core::data::basic::Array2D<double>>(nx,nx);
  for (size_t ix = 0; ix < vx.size(); ++ix)
    for (size_t iy = 0; iy < vy.size(); ++iy) data_periodic->set(ix, iy, sin(vx[ix]) * cos(vy[iy]));

  // --- Create the actual interpolator object
  CatmullRomInterpolator<double> cri;
  InterpolatePeriodic2D<double, CatmullRomInterpolator<double> > ip(-M_PI,inter_step,nx,-M_PI,inter_step,nx, data_periodic, cri);
  double max_error = 0.0;
  for (double x = -5; x <= 4.0; x += inter_step / 3.0) {
    for (double y = -5.0; y <= 4.0; y += inter_step / 3.0) {
      double v = sin(x) * cos(y); // --- calculate the true value of the interpolated function ...
      double vi = ip(x, y);     // --- also the interpolated value
      double err = fabs(v - vi);
      max_error = std::max(err, max_error);
    }
  }

  std::cout << "Maximum interpolation error: " << max_error << "\n";
}
```

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:

```
#include <iostream>
#include <fstream>
#include <simulations/systems/ising/Ising2D.hh>
#include <simulations/movers/ising/SingleFlip2D.hh>
#include <simulations/movers/ising/WolffMove2D.hh>

#include <simulations/observers/ObserveEvaluators.hh>
#include <simulations/movers/MoversSetSweep.hh>
#include <simulations/sampling/SimulatedAnnealing.hh>


/* @brief Simple but fully functional Ising simulating program.
 *
 * Runs simulated annealing simulations for 100x100
 *
 * CATEGORIES: simulations::systems::ising::Ising2D
 * KEYWORDS:   Mover;Simulated Annealing;Ising2D
 */
using namespace simulations::systems::ising;
using namespace simulations::movers::ising;

int main(const int argc, const char *argv[]) {
  /* Simulation controlling variables */
  int n_cols = 10, n_rows = 10;                    // size of system

  /* Other settings necessary for the simulation */
  int seed = 12345;                              // seed for rng
  core::calc::statistics::Random::seed(seed);

  /* Initializing the system */
  std::shared_ptr<Ising2D<core::index1,core::index2>> system = std::make_shared
↪<Ising2D<core::index1,core::index2>>(n_rows, n_cols);
```

(continues on next page)

(continued from previous page)

```
  system->initialize();     // Populate system with random spins

  simulations::movers::MoversSet_SP movers = std::make_shared
↪<simulations::movers::MoversSetSweep>();
  movers->add_mover( std::make_shared<SingleFlip2D<core::index1,core::index2>>
↪(*system),system->count_spins());
  movers->add_mover( std::make_shared<WolffMove2D<core::index1,core::index2>>
↪(*system),system->count_spins()*0.2);

  std::vector<float> temperatures = {5,4,3,2.5,2.25,2,1.75,1.5,1};
  simulations::sampling::SimulatedAnnealing sa(movers,temperatures);
  sa.cycles(100,100);

  simulations::observers::ObserveEvaluators_SP observations = std::make_shared
↪<simulations::observers::ObserveEvaluators>("");
  observations->add_evaluator(system);
  sa.outer_cycle_observer(observations);

  sa.run();
  observations->finalize();
}
```

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

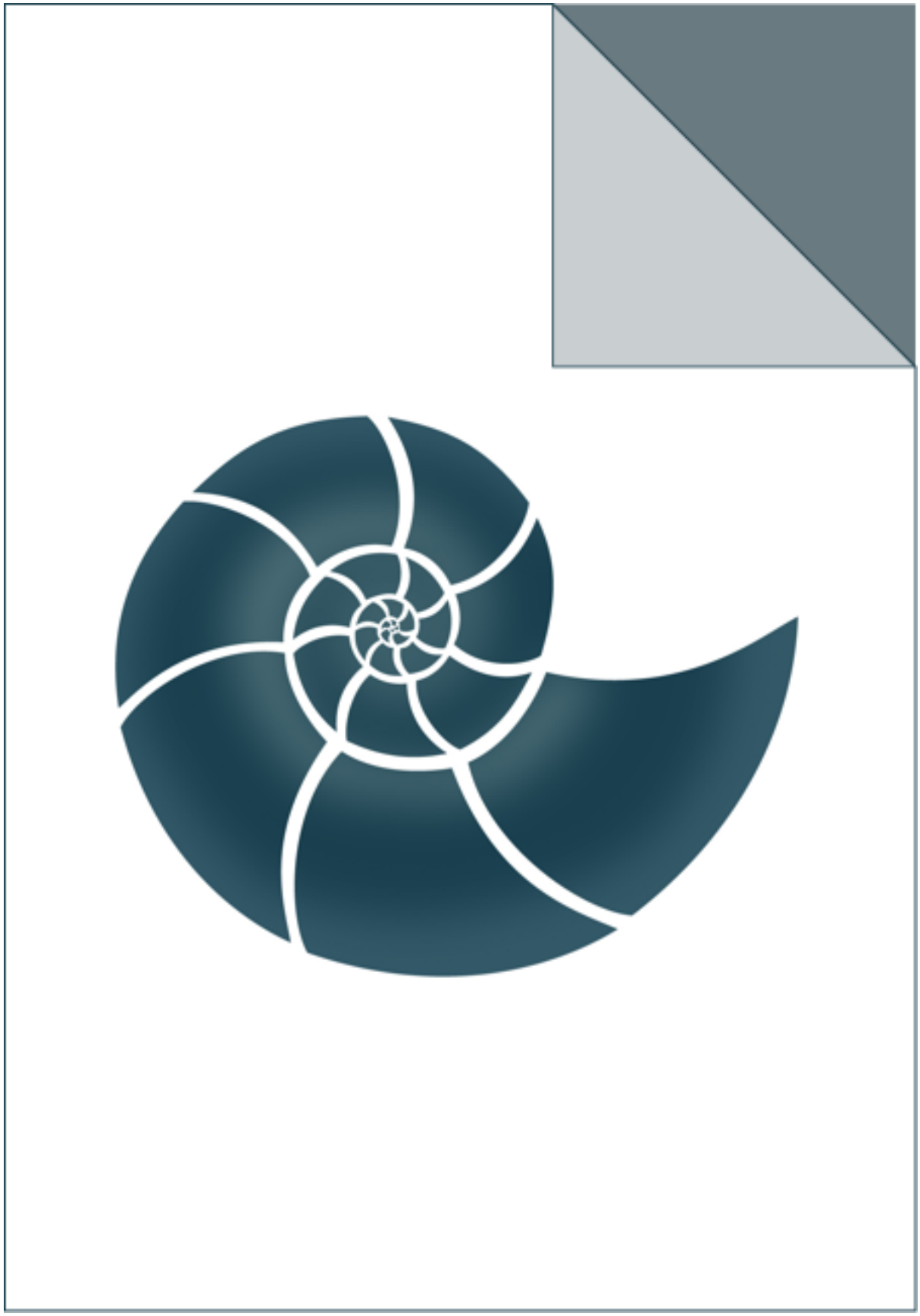
Program source:

```
#include <iostream>

#include <core/algorithms/IterateIJ.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief A simple example shows how to use IterateIJ
 *
 *
 * CATEGORIES: core::algorithms::IterateIJ
 * KEYWORDS:  data structures; algorithms
 */
int main(const int argc, const char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  // ---------- Here we declare an iterator that generates indexes for a square 5x5 2D array
```

(continues on next page)

(continued from previous page)

```
  core::algorithms::IterateIJ ij1(5, false);
  for (auto ij:ij1) std::cout << ij.first << " " << ij.second << "\n";
  std::cout <<"\n";

  // ---------- Here we declare an iterator that generates indexes for the UR triangle of a 5x5 2D array
  core::algorithms::IterateIJ ij2(5, true);
  for (auto ij:ij2) std::cout << ij.first << " " << ij.second << "\n";
  std::cout <<"\n";

  // ---------- Now only selected rows of a square 5x5 2D array
  core::algorithms::IterateIJ ij3(5, false);
  ij3.add_selected_row(1).add_selected_row(2);
  for (auto ij:ij3) std::cout << ij.first << " " << ij.second << "\n";
  std::cout <<"\n";

  // ---------- Now only selected rows and only  UR triangle of a 5x5 2D array
  core::algorithms::IterateIJ ij4(5, true);
  ij4.add_selected_row(2).add_selected_row(3);
  for (auto ij:ij4) std::cout << ij.first << " " << ij.second << "\n";
}
```



ex_JsonNode

Demonstrates how to handle JSON data.

USAGE:

```
./ex_JsonNode
```

Keywords:

- JSON

Categories:

- core::data::io::JsonNode

Output files:

- stdout.out

Program source:

```
#include <iostream>
#include <core/data/io/json_io.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Demonstrates how to handle JSON data.

USAGE:
    ./ex_JsonNode

)";

/** @brief Demo for handling JSON data
 *
 * The example tests whether a JSON data is parsed and printed correctly.
 *
 * CATEGORIES: core::data::io::JsonNode
 * KEYWORDS:   JSON
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::data::io;

  // Create JsonNode object using constructors
  JsonNode_SP n0 = std::make_shared<JsonNode>(std::make_shared<JsonValue>("options"),
↪1);
```

(continues on next page)

(continued from previous page)

```
  n0->add_branch(std::make_shared<JsonNode>(std::make_shared<JsonValue>("size","56"),2));
  n0->add_branch(std::make_shared<JsonNode>(std::make_shared<JsonValue>("color","red"),3));
  std::cout << n0; // This is how to print a full JSON tree

  std::string json = R"({"options" : {"size" : 56 , "color" : {"fg" : "red", "feature" : "none", "bg" : {"r":25,"g":124,"b":19} }, "do" : "all"})"
  ;
  JsonNode_SP root = read_json(json); // Here json string is parsed
  std::cout << root; // and here send back to a stream

  // Here a JsonArray instance is created; an empty array at first ...
  std::shared_ptr<JsonArray> jv1 = std::make_shared<JsonArray>("res-1");
  // and now that empty array is fileld with data; <code>"011"</code> string means␣
↪that the first token (i.e. 37) does not
  // require quotes (hence logical 0) and the two latter tokens do (logical 1)
  jv1->values("011",37,"ALA",'A');
  JsonNode_SP jv2 = create_json_node("res-2","011",38,"PHE",'A');
  std::cout << (*jv1) << " " << (jv2) << "\n";

  // Create JsonNode object using helper methods, provided by <code>json_io.hh</code>
  JsonNode_SP another_root = create_json_node();
  another_root->add_branch( create_json_node(jv1) );
  another_root->add_branch( jv2 );
  std::cout << another_root;
}
```

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:

```
#include <iostream>
#include <iomanip>
#include <core/data/io/Pdb.hh>

#include <core/data/io/DataTable.hh>
#include <core/calc/statistics/KDE_1D.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Reads one column of observations and calculates Kernel Density Estimator (KDE) with␣
↪given bandwidth value for the data.
```

(continues on next page)

(continued from previous page)

```
 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.

)";

/** @brief Reads one column of observations and calculates Kernel Density Estimator (KDE) for the data
 *
 * CATEGORIES: core/calc/statistics/KDE_1D
 * KEYWORDS:   statistics; estimation; data table
 */
int main(const int argc, const char* argv[]) {

  if(argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  core::data::io::DataTable input_data(argv[1]); // --- Here we read a text file with data values in columns
  std::vector<double> col1; // --- empty vector to hold the data from a file

  bool is_periodic = (argc > 2 && argv[argc-1][0] == 'p');
  // --- Here we read the input file and create a KDE estimator for the data in the first column
  core::calc::statistics::KDE_Kernel kernel_type =  core::calc::statistics::normal_kernel;
  core::calc::statistics::KDE_1D kde(input_data.column(0, col1), atof(argv[2]), kernel_type, is_periodic);

  // --- find max and min value in the file; tabulate the data in 100 steps
  double min = (argc < 5) ? *std::min_element(col1.begin(), col1.end()) : atof(argv[3]);
  double max = (argc < 5) ? *std::max_element(col1.begin(), col1.end()) : atof(argv[4]);
  double step = (max - min) / 300.0;
  for (double x = min; x <= max; x += step) std::cout << utils::string_format("%8.3f %8.3f\n", x, kde(x, is_periodic));
}
```

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:

```
#include <iostream>

#include <core/index.hh>
#include <core/calc/numeric/LBFGS.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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.

)";

using namespace core::calc::numeric;

class TestFunction : public DerivableFunction<double> {
public:

  double a, b;

  TestFunction() : a(1.0), b(30.0) {}

```

(continues on next page)

(continued from previous page)

```
  virtual double operator()(const std::vector<double> &x) {
    return (a - x[0]) * (a - x[0]) + b * (x[1] - x[0] * x[0]) * (x[1] - x[0] * x[0]);
  }

  virtual double operator()(const std::vector<double> &x, std::vector<double> &
↪gradient) {

    float t1 = a - x[0];
    float t2 = b * (x[1] - x[0] * x[0]);
    gradient[1] = 2 * b * t2;
    gradient[0] = -2.0 * (x[0] * gradient[1] + t1);

    return t1 * t1 + t2 * t2;
  }

  core::index2 dim() const { return 2; }
};

/** @brief Example shows how to use BFGS function minimizer
 *
 *
 * CATEGORIES: core/calc/numeric/Bfgs
 * KEYWORDS:   numerical methods
 */
int main(const int argc, const char *argv[]) {

  TestFunction f;
  LBFGS<double> minimizer(2);

  std::vector<double> x{0.0,0.0};
  double val;
  core::index2 nit = minimizer.minimize(f, x, val);

  std::cout << "Minimum value " << val << " found at [";
  for (const double v:x) std::cout << v << ' ';
  std::cout << "] after " << nit << " iterations\n";
  return 0;
}
```

ex_MC_Ar

The program runs an isothermal MC simulation of argon gas. By default it stars 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:

```
#include <cstdio>
#include <ctime>
#include <iostream>
#include <thread>

#include <core/data/basic/Vec3I.hh>
#include <core/BioShellVersion.hh>


#include <utils/string_utils.hh>
#include <utils/options/Option.hh>
#include <utils/options/OptionParser.hh>
#include <utils/options/output_options.hh>
#include <utils/options/sampling_options.hh>

#include <simulations/systems/CartesianAtoms.hh>
#include <simulations/systems/BuildFluidSystem.hh>
#include <simulations/systems/SingleAtomType.hh>
#include <simulations/movers/TranslateAtom.hh>
#include <simulations/forcefields/cartesian/LJEnergySWHomogenic.hh>
#include <simulations/sampling/IsothermalMC.hh>
#include <simulations/observers/ObserveEvaluators.hh>
#include <simulations/observers/cartesian/PdbObserver.hh>
#include <simulations/evaluators/CallEvaluator.hh>
#include <simulations/observers/ObserveMoversAcceptance.hh>
```

(continues on next page)

(continued from previous page)

```
#include <simulations/observers/TriggerEveryN.hh>
#include <simulations/observers/AdjustMoversAcceptance.hh>
#include <simulations/observers/cartesian/SimplePdbFormatter.hh>

using namespace core::data::basic;

utils::Logger logs("ex_MC_Ar");

std::string program_info = R"(

The program runs an isothermal MC simulation of argon gas. By default it stars 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]

)";

const double EPSILON = 1.654E-21;         // [J] per molecule
const double EPSILON_BY_K = EPSILON / 1.381E-23;          // = 119.6 in Kelvins
const double SIGMA = 3.4;                 // in Angstroms

/** @brief Isothermal Monte Carlo simulation of argon gas.
 *
 */
int main(const int argc,const char* argv[]) {

  using core::data::basic::Vec3I;
  using namespace simulations::systems;
  using namespace simulations::movers; // for MoversSet
  using namespace simulations::observers::cartesian; // for all observers

  logs << utils::LogLevel::INFO << "BioShell version:\n"<<core::BioShellVersion().to_string() << "\n";
  core::index4 n_outer_cycles = 1000;
  core::index4 n_inner_cycles = 100;
  double density = 0.5;      // density of the system controls how many atoms will be contained in the box
  double temperature = 97;  // in Kelvins
  core::index4 n_atoms = 64;
  double max_jump = 0.5;                // Random move range (in Angstroms)

  core::data::structural::Structure_SP argon_structure = nullptr;
  core::data::structural::PdbAtom_SP ar_atom = std::make_shared<core::data::structural::PdbAtom>(1," AR");
  if (argc < 6) std::cerr << program_info;
  else {
    if (utils::is_integer(argv[1])) n_atoms = atoi(argv[1]);
    else { // --- read an input file if given
      core::data::io::Pdb reader(argv[1]);
      argon_structure = reader.create_structure(0);
      n_atoms = argon_structure->count_atoms();
    }
    density = atof(argv[2]);
    temperature = atof(argv[3]);
    n_inner_cycles = atoi(argv[4]);
```

(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, 0.33333333333333);

  // --- 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)

```
//  std::shared_ptr<simulations::observers::ObserveMoversAcceptance> observe_moves
//    = std::make_shared<simulations::observers::ObserveMoversAcceptance>(*movers,
↪"movers.dat");

  std::shared_ptr<simulations::observers::AdjustMoversAcceptance> observe_moves
    = std::make_shared<simulations::observers::AdjustMoversAcceptance>(*movers,
↪"movers.dat", 0.4);
  observe_moves->observe_header();

  std::shared_ptr<AbstractPdbFormatter> fmt = std::make_shared<SimplePdbFormatter>("␣
↪AR ", "AR ", "AR");
  auto observe_trajectory = std::make_shared
↪<simulations::observers::cartesian::PdbObserver>(ar, fmt, "ar_tra.pdb");
  observe_trajectory->trigger(std::make_shared<simulations::observers::TriggerEveryN>
↪(10));
  mc.outer_cycle_observer(observe_trajectory); // --- commented out to save disk space
  mc.outer_cycle_observer(observe_moves);
  mc.outer_cycle_observer(obs);
  mc.cycles(n_inner_cycles,n_outer_cycles,1);

  mc.run();

  simulations::observers::cartesian::PdbObserver final(ar, fmt, "final.pdb");
  final.observe();
  logs << utils::LogLevel::INFO << "Final energy " << lj_energy.energy(ar) << "\n";
}
```

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:

```
#include <iostream>
#include <vector>

#include <core/data/io/Pdb.hh>
#include <core/BioShellEnvironment.hh>
#include <simulations/forcefields/mm/MMForceField.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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:
```

(continues on next page)

(continued from previous page)

```
    ./ex_MMAtomTyping [param-file] input.pdb
EXAMPLE:
    ./ex_MMAtomTyping 2gb1.pdb
    ./ex_MMAtomTyping amber03_atoms.par 2gb1.pdb

)";

/** @brief Assigns MM atom typing for every atom of the given protein according to the given force field parametrisation file
 *
 * CATEGORIES: simulations/forcefields/mm/MMAtomTyping
 * KEYWORDS:   PDB input; atom typing; force field
 */
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 simulations::forcefields::mm;
  MMForceField mmff("AMBER03");
  const MMAtomTyping & atom_typing = *(mmff.mm_atom_typing());
  core::data::io::Pdb pdb((argc == 2) ? argv[1] : argv[2]);
  core::data::structural::Structure_SP s = pdb.create_structure(0);
  for (auto chain: *s) {
    for (core::index2 ires = 0; ires < chain->size(); ++ires) {
      for (auto atom : *(*chain)[ires]) {
        core::index2 t = atom_typing.atom_type(*atom);
        std::cout << *(*atom).owner() << " " << (*atom).atom_name() << " : " << t << " " << atom_typing.atom_internal_name(t) << "\n";
      }
    }
  }
}
```

ex_MMBondEnergy

Keywords:

- no_keywords

Categories:

- no_categories

Input files:

- 2gb1.pdb
- amber03_ffbonded.itp
- amber03_atoms.par

Output files:

- stdout.out

Program source:

```
#include <iostream>

#include <core/data/basic/Vec3.hh>
#include <core/data/structural/Structure.hh>
#include <utils/LogManager.hh>

#include <simulations/forcefields/mm/MMBondEnergy.hh>
#include <simulations/forcefields/mm/MMBondedParameters.hh>
#include <simulations/forcefields/mm/MMForceField.hh>

int main(const int argc, const char *argv[]) {

  using namespace simulations::forcefields::mm; // for all MM - related force field␣
↪classes
  using namespace core::data::structural; // for Structure, Residue, PdbAtom
  using namespace core::data::basic; // for Vec3

  utils::LogManager::get().set_level("FINE");

  if (argc < 2) {
    std::cerr << "USAGE:\n\t./ex_MMBondEnergy atom_typing ff_bonded.par input.pdb\n\n
↪";
    exit(0);
  }

  // --- Create molecular mechanic bond energy object
    MMForceField mmff("AMBER03");

    MMBondedParameters bond_manager(argv[2],mmff.mm_atom_typing());
```

(continues on next page)

(continued from previous page)

```
28
29    // --- Read structure for which calculate bond energy
30    core::data::io::Pdb reader(argv[3]); // file name (PDB format, may be gzip-ped)
31    core::data::structural::Structure_SP strctr = reader.create_structure(0);
32    auto rc = CartesianChains(mmff.mm_atom_typing(), *strctr);
33
34    // --- Create molecular mechanic bond energy object
35    simulations::forcefields::mm::MMBondEnergy bond_energy(rc,bond_manager);
36    const auto & bonds = bond_energy.get_bonds();
37      std::cout << "#    i      j      E(d0)      d0     d\n";
38
39    // --- Calculate energy for each bond in the structure
40    double total = 0.0;
41    for (auto it = bonds.cbegin(); it != bonds.cend(); ++it) {
42      double en = bond_energy.calculate(*it);
43      total += en;
44      double dreal = (rc)[it->i].distance_to((rc)[it->j]);
45      std::cout << utils::string_format("%5d %5d %10.3f    %4.2f %4.2f\n", (*it).i,␣
↪(*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:

```
#include <string>
#include <chrono>

#include <core/data/basic/Vec3.hh>

#include <utils/string_utils.hh>
#include <utils/LogManager.hh>

#include <simulations/forcefields/mm/MMNonBonded.hh>
#include <simulations/forcefields/mm/MMBondType.hh>
#include <simulations/forcefields/mm/MMBondEnergy.hh>
#include <simulations/forcefields/mm/MMPlanarEnergy.hh>
#include <simulations/forcefields/mm/MMDihedralEnergy.hh>
#include <simulations/forcefields/mm/MMBondedParameters.hh>
#include <simulations/forcefields/mm/MMForceField.hh>

int main(const int argc, const char *argv[]) {

  using namespace core::data::structural; // for Structure, Residue, PdbAtom
  using namespace core::data::basic; // for Vec3
  using namespace simulations::forcefields::mm; // for all MM - related force field␣
↪classes
  using namespace simulations::systems; // for ResidueChain

  utils::LogManager::get().set_level("INFO");

  if (argc < 2) {
    std::cerr << "USAGE:\n\t./ex_MMEnergy atoms.par ff_bonded.par 2gb1.pdb\n\n";
```

(continues on next page)

(continued from previous page)

```
    exit(0);
  }

  // --- Read atom typing and bond parameters
    MMForceField mmff("AMBER03");
    simulations::forcefields::mm::MMBondedParameters ff(argv[2],mmff.mm_atom_typing());

  // --- Read structure for which calculate bond energy
  core::data::io::Pdb reader(argv[3]); // file name (PDB format, may be gzip-ped)
  Structure_SP strctr = reader.create_structure(0);
  auto rc = CartesianChains(mmff.mm_atom_typing(), *strctr);

  size_t i = 0;
//  for (auto atom_it = strctr->first_const_atom(); atom_it != strctr->last_const_atom(); ++atom_it) {
//    auto at = atom_typing->atom_type(**atom_it);
//    (*rc)[i].register_ = (1.0 + at.charge()) * 10000.0;
//    ++i;
//  }

  // --- Create molecular mechanic bond energy object
  simulations::forcefields::mm::MMBondEnergy bond_energy(rc, ff);
  // --- Create molecular mechanic planar energy objects
  simulations::forcefields::mm::MMPlanarEnergy planar_energy(rc, ff, bond_energy);
  // --- Create molecular mechanic dihedral energy objects
  simulations::forcefields::mm::MMDihedralEnergy dihedral_energy(rc, ff, bond_energy);

  // --- Create non-bonded energy; pass the reference to bond energy object so non-bonded energy can exclude bonds
  MMNonBonded nb_energy(rc, bond_energy);
  //  nb_energy.get_excluded_pairs().print(std::cout);

  auto start = std::chrono::high_resolution_clock::now();
   std::cout<<"#bond_energy planar_energy dihedral_energy  nb_energy\n";
  std::cout << utils::string_format("  %10.3f    %10.3f      %10.3f %10.3f\n",
    bond_energy.energy(rc), planar_energy.energy(rc), dihedral_energy.energy(rc), nb_energy.energy(rc));
  auto end = std::chrono::high_resolution_clock::now();
  std::cerr << "# computed in " << std::chrono::duration<double>(end - start).count() << " [s]\n";
}
```

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:

```
#include <iostream>
#include <string>
#include <vector>
#include <tuple>

#include <utils/string_utils.hh>
#include <utils/LogManager.hh>
#include <core/data/structural/Structure.hh>
#include <simulations/forcefields/mm/MMBondType.hh>
#include <simulations/forcefields/mm/MMBondEnergy.hh>
#include <simulations/forcefields/mm/MMNonBondedSW.hh>
#include <simulations/forcefields/mm/MMNonBonded.hh>
#include <simulations/forcefields/mm/MMForceField.hh>

/** \todo_code Fix this demo one MM topology files parser is ready
 *
 */
int main(const int argc, const char *argv[]) {

  using namespace simulations::forcefields::mm; // for all MM - related force field␣
↪classes
  using namespace core::data::structural; // for Structure, Residue, PdbAtom
  using namespace core::data::basic; // for Vec3
  using namespace simulations::systems; // for ResidueChain

  utils::LogManager::get().set_level("FINE");

  if (argc < 2) {
```

(continues on next page)

(continued from previous page)

```
    std::cerr << "USAGE:\n\t./ex_MMNonBondEnergy data_atom_typing data_bond_typing␣
↪input.pdb\n\n";
    exit(0);
  }

  // --- Read atom typing and bond parameters
    MMForceField mmff("AMBER03");
  MMBondedParameters bonded_manager(argv[2],mmff.mm_atom_typing());


  // --- Read structure for which calculate bond energy
  core::data::io::Pdb reader(argv[3]); // file name (PDB format, may be gzip-ped)
  Structure_SP strctr = reader.create_structure(0);
  auto rc = CartesianChains(mmff.mm_atom_typing(), *strctr);

  // --- Create molecular mechanic bond energy object
  MMBondEnergy bond_energy(rc, bonded_manager);

  // --- Create non-bonded energy; pass the reference to bond energy object so non-
↪bonded energy can exclude bonds
  MMNonBonded nb_energy(rc,bond_energy);
  std::cout<<"# list of atoms pair excluded from pairwise calculation information\n";
 // nb_energy.get_neighbor_list().print(std::cout);

  // --- Calculate total non-bonded energy of the structure
  double energy_by_str = nb_energy.energy(rc);
  std::cout << utils::string_format("%10.3f\n", energy_by_str);
}
```

ex_MeanFieldDistributions

Prints values for a given Mean Field potential so it can be plotted nicely. The parameters of the program are: MF potental 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:

```
#include <string>

#include <simulations/forcefields/mf/MeanFieldDistributions.hh>

#include <utils/exit.hh>

std::string program_info = R"(

Prints values for a given Mean Field potential so it can be plotted nicely. The␣
↪parameters of the program are:
MF potental 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)
)";

/** @brief Prints values for a given Mean Field potential so it can be plotted nicely.
 *
 * The program works for any potential stored in the Bioshell row-wise format; both␣
↪CABS and SURPASS potentials may be
 * plotted with this utility. Program usage:
 *
 * ex_MeanFieldDistributions  "forcefield/cabs/R13_cabs.dat"  0.01 2.0 15.0 AD.HH
 *
 * where the parameters of the program are:
 *   MF potental file name, pseudocounts, min_x, max_x, potential name [other␣
↪potential names ..]
```

(continues on next page)

(continued from previous page)

```
 *
 * CATEGORIES: simulations/forcefields/mf/MeanFieldDistributions
 * KEYWORDS:   force field
 */
int main(const int argc, const char *argv[]) {

  using namespace simulations::forcefields::mf;

  if(argc < 5) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  double pseudocounts = utils::from_string<double>(argv[2]);
  double min_x = utils::from_string<double>(argv[3]);
  double max_x = utils::from_string<double>(argv[4]);

  std::shared_ptr<MeanFieldDistributions> mf = load_1D_distributions(argv[1], pseudocounts);
  std::vector<EnergyComponent_SP> terms;
  if (argc > 5)
    for (int i = 5; i < argc; ++i) {
      std::string label(argv[i]);
      if(mf->contains_distribution(label)) terms.push_back(mf->at(label));
      else std::cerr <<"Key "<<label<<" not found!\n";
    }
  else
    for (const std::string label:mf->known_distributions())
        terms.push_back(mf->at(label));

  for (double d = min_x; d <= max_x; d += 0.0125) {
    std::cout << utils::string_format("%7.3f",d);
    for (const auto e : terms) std::cout << utils::string_format(" %8.3f", (*e)(d));
    std::cout << "\n";
  }
}
```

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:

```
#include <iostream>
#include <algorithm> // for std::count_if
#include <iterator>  // for std::distance

#include <utils/string_utils.hh>
#include <core/chemical/Monomer.hh>
#include <core/chemical/monomer_io.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Unit test which demonstrates  functionality of core::chemical::Monomer data type.

USAGE:
./ex_Monomer

)";

// First we declare a function object used to count how many monomers have type = 'P' i.e. "protein"
bool IsAA(const core::chemical::Monomer &m) { return (m.type == 'P') || (core::chemical::Monomer::get(m.parent_id).type == 'P'); }

/** @brief Example demonstrates functionality of core::chemical::Monomer data type.
 *
 * CATEGORIES: core::chemical::Monomer
 * KEYWORDS:   monomers
 */
int main(const int argc, const char *argv[]) {

```

(continues on next page)

(continued from previous page)

```
  using namespace core::chemical;
  // First we iterate over all monomers and count, how many of them are actually amino acids
  // See how bioshell's iterators work together with std library
  int n_aa = std::count_if(Monomer::cbegin(), Monomer::cend(), IsAA);

  std::cout << std::distance(Monomer::cbegin(), Monomer::cend()) << " standard monomers found, including "
            << n_aa << " peptide-forming.\n";

  std::cout << "The order of standard amino acid residues is:\n";
  for (core::index2 i = 0; i < n_aa; ++i) {
    const Monomer &m = Monomer::get(i);
    std::cout << utils::string_format("%2d %c %3s %c\n", i, m.code1, m.code3.c_str(), m.type);
  }

  load_monomers_from_db(); // --- load the database of all known monomers
  // Count amino acid monomers again
  n_aa = std::count_if(Monomer::cbegin(), Monomer::cend(), IsAA);
  std::cout << "Monomer database loaded; " << std::distance(Monomer::cbegin(), Monomer::cend())
            << " monomers found, including " << n_aa << " peptide-forming.\n";

  // Now let's count how many non-standard residues are derived from ALA
  // Simply a parent_id of a monomer must be equal to ALA.id
  // This time we use a lambda expression rather than a functor
  n_aa = std::count_if(Monomer::cbegin(), Monomer::cend(), [](const Monomer &m) { return m.parent_id == Monomer::ALA.id; });
  std::cout << "There are " << n_aa << " derived from alanine\n";
}
```

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:

```
#include <utils/Logger.hh>
#include <utils/LogManager.hh>
#include <core/chemical/MonomerStructure.hh>
#include <core/chemical/MonomerStructureFactory.hh>


#include <utils/exit.hh>

std::string program_info = R"(

Unit test which shows how to read CIF files.

USAGE:
    ex_MonomerStructure file.cif
EXAMPLE:
    ex_MonomerStructure AA3.cif

)";

/** @brief ex_MonomerStructure tests reading CIF files
 *
 * CATEGORIES: core/chemical/MonomerStructure
```

(continues on next page)

(continued from previous page)

```
 * KEYWORDS:   CIF input
 */
int main(const int argc, const char *argv[]) {

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

  utils::LogManager::FINE(); // --- INFO is the default logging level; set it to FINE to see more
  core::chemical::MonomerStructure_SP str = core::chemical::MonomerStructure::from_cif(argv[1]);
  std::cout<<"\nPOLAR H: ";
  for (auto i: str->polar_hydrogens()) std::cout<<i->atom_name()<<" ";
  std::cout<<"\nNONPOLAR H: ";

  for (auto i: str->nonpolar_hydrogens()) std::cout<<i->atom_name()<<" ";
  std::cout<<"\nNONPOLAR: ";

  for (auto i: str->nonpolar_heavy()) std::cout<<i->atom_name()<<" ";
  std::cout<<"\nDONORS: ";

  for (auto i: str->hydrogen_donors()) std::cout<<i->atom_name()<<" ";
  std::cout<<"\nACCEPTORS: ";

  for (auto i: str->hydrogen_acceptors()) std::cout<<i->atom_name()<<" ";
          std::cout<<"\n";

  core::chemical::MonomerStructureFactory m = core::chemical::MonomerStructureFactory::get_instance();
  core::chemical::MonomerStructure_SP mstr = m.get("PRO");
  std::cout<<mstr->code3<<"\n";
  std::cout<<"\nPOLAR H: ";
  for (auto i: mstr->polar_hydrogens()) std::cout<<i->atom_name()<<" ";
}
```

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:

```
#include <iostream>
#include <algorithm>
#include <set>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/calc/structural/NeighborGrid3D.hh>

#include <utils/exit.hh>
#include <utils/LogManager.hh>

std::string program_info = R"(

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

```

(continues on next page)

(continued from previous page)

```
)";


/** @brief Finds possible structural neighbors of a given atom
 *  *
 * CATEGORIES: core::calc::structural::NeighborGrid3D
 * KEYWORDS: PDB input; structure selectors
 */
int main(const int argc, const char* argv[]) {

  if (argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
  utils::LogManager::FINE();

  using namespace core::data::io;
  Pdb reader(argv[1], all_true(is_not_water,is_not_alternative)); // --- file name (PDB format, may be gzip-ped)

  utils::Logger logs("ex_NeighborGrid3D");
  using namespace core::data::structural;
  Structure_SP strctr = reader.create_structure(0);
  selectors::SelectChainResidues select_query(argv[3]);
  // --- Here we find the query residue, selected by a selector string
  Residue_SP query_resid = nullptr;
  for(auto it=strctr->first_residue();it!=strctr->last_residue();++it) {
    if (select_query(*it)) {
      query_resid = *it;
      logs << utils::LogLevel::INFO << "selecting spatial neighbors of " <<
           " " << query_resid->residue_type().code3 << query_resid->residue_id() << "\n";
      break;
    }
  }

  // --- Create a 3D grid object
  double grid_mesh = atof(argv[2]);
  core::calc::structural::NeighborGrid3D grid(*strctr,grid_mesh);

  // --- Print content of the grid
  std::cout << "# Atoms as they are located on the grid\n";
  std::cout << "# Grid center is: " << grid.cx() << " " << grid.cy() << " " << grid.cz() << "\n";
  core::index2 ix, iy, iz;
  for(const auto & hash : grid.filled_cells()) {
    grid.xyz_from_hash(hash, ix, iy, iz);
    for(const PdbAtom_SP & at : grid.get_cell(hash)) {
      std::cout << "# " << hash << " " << at->owner()->residue_type().code3 << at->owner()->id() << " " << *at
                << " ix: " << ix << " iy: " << iy << " iz: " << iz << "\n";
    }
  }

  // --- Print neighbor cells of the selected residue
  core::index4 hash_ca = grid.hash(*query_resid->find_atom_safe(" CA "));
  std::vector<core::index4> neighb_hash;
  grid.get_neighbor_cells(hash_ca, neighb_hash);
  std::cout << "# neighbors of a cell " << hash_ca << "\n# ";
```

(continues on next page)

(continued from previous page)

```
  for (core::index4 n:neighb_hash) std::cout << " " << n;
  std::cout << "\n";

  // --- Mark the selection on a PDB file by setting B-factor to 10.0 (all other
↪atoms to 0.0)
  float max_distance = 0;
  std::vector< PdbAtom_SP> result;
  for(auto it=strctr->first_atom();it!=strctr->last_atom();++it) (**it).b_factor(0.0);
  for(PdbAtom_SP a : *query_resid) {
    result.clear();
    grid.get_neighbors(*a,result);
    for(auto atom_sp:result) {
      atom_sp->b_factor(10.0);
      max_distance = std::max(max_distance,atom_sp->distance_to(*a));
    }
  }

  for(auto it=strctr->first_atom();it!=strctr->last_atom();++it)
    std::cout << (**it).to_pdb_line() << "\n";
  std::cout << "# Max distance: " <<max_distance << "\n";
}
```

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:

```
#include <iostream>
#include <random>
#include <math.h>

#include <core/calc/statistics/NormalDistribution.hh>
#include <core/calc/statistics/Random.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

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

)";

/** @brief Unit test for NormalDistribution class.
 *
 * CATEGORIES: core/calc/statistics/NormalDistribution; core/calc/statistics/Random
 * KEYWORDS:   NormalDistribution; random numbers
 */
int main(const int argc, const char *argv[]) {

```

(continues on next page)

(continued from previous page)

```
  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::calc::statistics;

  Random rd = core::calc::statistics::Random::get();
  rd.seed(12345); // --- seed the generator for repeatable results
  std::mt19937 gen(rd());
  std::normal_distribution<> d(10.5, 2.0); // --- The original distribution we take a␣
↪sample from
  // --- Note that the container for samples is two-dimensional! Each sample is␣
↪placed in a separate row
  std::vector<std::vector<double> > data;
  std::vector<double> row(1);
  for (unsigned short i = 0; i < 1000; ++i) {
    row[0] = (d(gen));
    data.push_back(row); // --- This works only because C++ makes an implicit copy of␣
↪the vector we place into the outer vector
  }

  // --- Here we estimate the distribution parameters
  core::calc::statistics::NormalDistribution n(0.0, 1.0);
  std::vector<double> E = n.estimate(data);
  std::cout << "True values:      average = 10.5; stdev = 2.0\n";
  std::cout << "Estimated values: average = "
            << E[0] << " sdev = " << E[1] << "\n";            // Calculate average␣
↪and stdev of values in the vector
}
```

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:

```
#include <iostream>

#include <utils/options/Option.hh>
#include <utils/options/OptionParser.hh>
#include <utils/LogManager.hh>

using namespace utils::options;

/** @brief Shows how to use BioShell command line parser in your own program
 *
 * To test this program, run:
 * ./ex_OptionParser -n=4 -nn=1,2,3,4
 *
 * CATEGORIES: utils::options::OptionParser
 * KEYWORDS:   option parsing
 */
int main(const int cnt, const char *argv[]) {

  // --- Limit the stdout on stderr (logging)
  utils::LogManager::WARNING();

  // --- First get th parser instance (it's a singleton)
  utils::options::OptionParser &cmd = OptionParser::get("ex_OptionParser");

  // --- This is how to register an option that has already been declared in BioShell library
  cmd.register_option(utils::options::verbose, help);

  // --- User can also declare  non-standard options
  Option number("-n", "-number", "returns an integer");
  Option numbers("-nn", "-numbers", "returns a vector of integers");
  Option value("-x", "-value_x", "returns a real value of X");
  // --- Options that have beed declared, must also be registered
  // --- (Declaration doesn't mean automatic registration)
  cmd.register_option(number, numbers, value);
```

(continues on next page)

(continued from previous page)

```
35
36    // --- after all the relevant options were registered, we parse a program command line
37    cmd.parse_cmdline(cnt, argv);
38
39    // ---- User should not check for -help and -verbose flags : this is automatically done by OptionParser
40
41    // --- Once command line has been parsed, we may check for a program parameter and retrieve its value:
42    if (numbers.was_used()) std::cout << "A number given: " << option_value<int>(number) << "\n";
43
44    // --- Options may be also accessed by their long name (but not by the abbreviated name, so cmd.was_used("-x") won't work
45    if (cmd.was_used("-value_x")) std::cout << "A number given: " << option_value<double>("-value_x") << "\n";
46
47    // --- This shows how to read a vector of values
48    if(cmd.was_used(numbers)) {
49      std::vector<int> v = option_value<std::vector<int>, int>(numbers);
50      std::cout << "Given set of values: ";
51      for (auto vi : v) std::cout << vi << " ";
52      std::cout << "\n";
53      // --- This is how the raw string given at the command line may be accessed:
54      std::cout << "The raw string associated with -value_x option was: " << cmd.value_string(numbers) << "\n";
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:

```
#include <iostream>

#include <core/index.hh>
#include <core/calc/statistics/Random.hh>
#include <core/calc/statistics/P2QuantileEstimation.hh>

std::string program_info = R"(

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:
```

(continues on next page)

(continued from previous page)

```
    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

)";

/** @brief Reads a file with real values and calculates simple statistics: min, mean,
↪stdev, max.
 *
 * If no input file is provided, the program calculates the statistics from a random
↪sample
 *
 * CATEGORIES: core/calc/statistics/OnlineStatistics; core/calc/statistics/Random
 * KEYWORDS:   statistics
 */
int main(const int argc, const char *argv[]) {

  if(argc < 3) {
    // --- complain about missing program parameter
    std::cerr << program_info;
    // ---------- Use the random engine if no data is provided
    core::calc::statistics::Random r = core::calc::statistics::Random::get();
    r.seed(12345);  // --- seed the generator for repeatable results
    std::normal_distribution<double> normal_random;
    core::calc::statistics::P2QuantileEstimation quartile1(0.25),quartile2(0.5),
↪quartile3(0.75);
    for (core::index4 n = 0; n < 10000; ++n) {
      double rr = normal_random(r);
      quartile1(rr);
      quartile2(rr);
      quartile3(rr);
    }
    std::cout << "Quantile 0.25 :"<<quartile1.p_value() << "\n"; // Should be -0.675
    std::cout << "Quantile 0.50 :"<<quartile2.p_value() << "\n"; // Should be  0.0
    std::cout << "Quantile 0.75 :"<<quartile3.p_value() << "\n"; // Should be  0.675

  } else {
    std::ifstream in(argv[1]);
    core::calc::statistics::P2QuantileEstimation stats(atof(argv[2]));
    double r;
    core::index4 cnt = 0;
    while (in >> r) {
      ++cnt;
      stats(r);
    }
    std::cout << "Quantile " << atof(argv[2]) << " " << stats.p_value() << " based on
↪" << cnt << " observations\n";
  }
}
```

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:

```
#include <iostream>
#include <iterator>

#include <core/alignment/PairwiseAlignment.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Simple example showing how to retrieve arbitrary data according to a␣
↪sequence alignment object
 *
 * CATEGORIES: core::alignment::PairwiseAlignment;
 * KEYWORDS:   sequence alignment
 */
int main(const int argc, const char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  // ---------- The two sequences that are already globally aligned, therefore␣
↪indexes in both sequences start from 0
```

(continues on next page)

(continued from previous page)

```
  core::alignment::PairwiseAlignment ali("FTFTALILL-AVAV", 0, "--FTAL-LLAAV--", 0);

  // ---------- query "objects" : that may be C-alpha atoms, residues, etc; here just chars
  std::vector<char> query_chars = {'F', 'T', 'F', 'T', 'A', 'L', 'I', 'L', 'L', 'A', 'V', 'A', 'V'}; // --- all the characters of the query sequence
  std::vector<char> tmplt_chars = {'F', 'T', 'A', 'L', 'L', 'L', 'A', 'A', 'V'};    //  // --- all the characters of the template sequence

  // ---------- container for the expected result
  std::vector<char> query_chars_aligned;
  std::vector<char> tmplt_chars_aligned;

  // ---------- set up query "objects" in the order as they appear in the alignment; print result on the screen
  ali.get_aligned_query(query_chars, '-', query_chars_aligned);

  // ---------- show results (it should be identical as the original alignment
  std::copy(query_chars_aligned.begin(), query_chars_aligned.end(), std::ostream_iterator<char>(std::cout, ""));
  std::cout << "\n"; // ---------- Should print FTFTALILL-AVAV

  // ---------- Now we extract both query and template objects; only the mutually aligned positions (no gaps)
  query_chars_aligned.clear();
  ali.get_aligned_query_template(query_chars, tmplt_chars, query_chars_aligned, tmplt_chars_aligned);

  // ---------- show results
  std::copy(query_chars_aligned.begin(), query_chars_aligned.end(), std::ostream_iterator<char>(std::cout, ""));
  std::cout << "\n"; // ---------- Should print FTALLLAV
  std::copy(tmplt_chars_aligned.begin(), tmplt_chars_aligned.end(), std::ostream_iterator<char>(std::cout, ""));
  std::cout << "\n"; // ---------- Should also print FTALLLAV

  // ---------- ... and finally print the alignment as a path
  std::cout << ali.to_path() << "\n";
}
```

ex_PairwiseSequenceAlignment

Simple example showing how to work with PairwiseSequenceAlignment data structure, e.g. how to create such an object and hot 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:

```
#include <iostream>

#include <core/BioShellEnvironment.hh>

#include <core/alignment/on_alignment_computations.hh>
#include <core/alignment/PairwiseAlignment.hh>
#include <core/alignment/PairwiseSequenceAlignment.hh>
#include <core/data/io/alignment_io.hh>
#include <core/data/sequence/Sequence.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

Simple example showing how to work with PairwiseSequenceAlignment data structure, e.g. how to
create such an object and hot to print it in different formats.

USAGE:
./ex_PairwiseSequenceAlignment

)";

std::string Q52825_1 = "IDVLLGADDGSLAFVPSEFSISPGEKIVFKNNAGFPHNIVFDEDSIPSGVDASKISMSEEDLLNAKGETFEVALSNKGEYSFYCSPHQGAGMVGKVTV";
std::string P80401_2 = "VQMLNKGTDGAMVFEPGFLKIAPGDTVTFIPTDKS-HNVETFKGLIPDGV---------PDFKSKPNEQYQVKFDIPGAYVLKCTPHVGMGMVALIQV";

/** @brief Simple example showing how to work with PairwiseSequenceAlignment data structure.
```

(continues on next page)

(continued from previous page)

```
 *
 * CATEGORIES: core::alignment::PairwiseAlignment; core::alignment::PairwiseSequenceAlignment;
 * KEYWORDS: sequence alignment
 */
int main(const int argc, const char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::alignment;
  using namespace core::alignment::scoring;
  using namespace core::data::sequence; // for core::data::sequence::Sequence

  // ---------- Test for global alignment ----------
  // --- Alignment defined as path : '-' and '-' mean a gap in a template and in a query sequence, respectively; '*' is a match
  PairwiseAlignment_SP ali = std::make_shared<PairwiseAlignment>(0, 0, 0, "--*****-**|**--");

  // ---------- The two sequences that will be aligned
  Sequence_SP query = std::make_shared<Sequence>("query", "ITFTALILLAVAV", 1);
  Sequence_SP tmplt = std::make_shared<Sequence>("tmplt", "FTALLLAAV", 1);

  PairwiseSequenceAlignment seq_ali(ali, query, tmplt);

  // --- Show alignment as a path
  std::cout << "Alignment path:\n" << ali->to_path() << "\n\n";

  // ---------- Print the alignment in Edinburgh format
  core::index2 identity = sum_identical(seq_ali);
  core::index2 n_gaps = seq_ali.alignment->length() - seq_ali.alignment->n_aligned();
  std::cout << "# score: " << seq_ali.alignment_score() << " length: " << seq_ali.alignment->length()
            << " n_identical: " << identity << " n_gaps: " << n_gaps << "\n";
  core::data::io::write_edinburgh(seq_ali, std::cout, 80);

  // ---------- Test for local alignment ----------
  // --- Alignment defined as path : '-' and '-' mean a gap in a template and in a query sequence, respectively; '*' is a match
  PairwiseAlignment_SP loc_ali = std::make_shared<PairwiseAlignment>(2, 0, 0.0, "****-**|**");
  PairwiseSequenceAlignment loc_seq_ali(loc_ali, query, tmplt);

  // ---------- Print the alignment in Edinburgh format
  identity = sum_identical(loc_seq_ali);
  n_gaps = loc_seq_ali.alignment->length() - loc_seq_ali.alignment->n_aligned();
  std::cout << "# score: " << loc_seq_ali.alignment_score() << " length: " << loc_seq_ali.alignment->length()
            << " n_identical: " << identity << " n_gaps: " << n_gaps << "\n";
  core::data::io::write_edinburgh(loc_seq_ali, std::cout, 80);


  PairwiseSequenceAlignment loc_seq_ali2("Q52825_1", Q52825_1, 0, "P80401_2", P80401_2, 0, 0.0);
  loc_seq_ali2.template_sequence->first_pos(28);
  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:

```
#include <iostream>
#include <random>

#include <core/index.hh>
#include <core/calc/numeric/basic_algebra.hh>
#include <core/calc/numeric/Pca3.hh>
#include <utils/exit.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

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.

)";

/** @brief Orients 3D points along the axes using PCA algorithm
 *
 * CATEGORIES: core/calc/numeric/basic_algebra.hh
 * KEYWORDS: PCA; transformations
```

(continues on next page)

(continued from previous page)

```
 */
int main(const int argc, const char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::data::basic; // --- for Vec3 and Array2D

  std::mt19937 gen;
  std::uniform_real_distribution<double> r(0, 1);
  std::vector<Vec3> points3d;
  for (core::index2 i = 0; i < 100; ++i) {
    double x = r(gen), y = r(gen), z = r(gen);
    points3d.emplace_back(x + 0.3 * y + 0.6 * z, 0.4 * x + 1.9 * y + 0.7 * z, 0.3 * x␣
↪+ 0.1 * y + 0.7 * z);
  }
  core::calc::numeric::Pca3 pca3(points3d);
  auto rt = pca3.create_transformation();
  for (auto &p:points3d) {
    std::cout << p << " ";
    rt.apply(p);
    std::cout << p << "\n";
  }
}
```

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:

```
#include <iostream>
#include <core/data/io/Pdb.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Reads a PDB file and creates a Structure object.
```

(continues on next page)

(continued from previous page)

```
 *
 * Input PDB data is filtered so only protein backbone atoms are loaded
 *
 * CATEGORIES: core::data::io::Pdb;
 * KEYWORDS:   PDB input; PDB line filter; Structure
 */
int main(const int argc, const char* argv[]) {

  if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
  for (int i = 1; i < argc; ++i) {
    core::data::io::Pdb reader(argv[i], // file name (PDB format, may be gzip-ped)
      core::data::io::is_bb,            // a predicate to read only the ATOM lines corresponding to backbone atoms
        core::data::io::only_ss_from_header, true);                              // parse PDB header
    core::data::structural::Structure_SP backbone = reader.create_structure(0);
    std::cout << "protein " << reader.pdb_code() << " has " << backbone->count_chains()
              << " chain(s), " << backbone->count_residues()
              << " residues and " << backbone->count_atoms() << " backbone atoms\n";
    std::cout << "title          : " << backbone->title() << "\n";
    std::cout << "compound       : " << backbone->compound() << "\n";
    std::cout << "classification : " << backbone->classification() << "\n";
    std::cout << "deposited      : " << backbone->deposition_date() << "\n";
    std::cout << "Is XRAY?       : " << ((backbone->is_xray()) ? "YES\n" : "No\n");
    std::cout << "Is NMR?        : " << ((backbone->is_nmr()) ? "YES\n" : "No\n");
    std::cout << "Is EM?         : " << ((backbone->is_em()) ? "YES\n" : "No\n");
    std::cout << "resolution     : " << backbone->resolution() << "\n";
    std::cout << "R-value        : " << backbone->r_value() << "\n";
    std::cout << "R-free         : " << backbone->r_free() << "\n";
    if(backbone->keywords().size()>0)
      std::cout << "keywords  : " << backbone->keywords()[0];
    for (auto it = ++backbone->keywords().cbegin(); it != backbone->keywords().cend(); ++it)
      std::cout << ", "<< *it;
    std::cout << "\n";
  }
}
```

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:

```
#include <iostream>

#include <core/calc/numeric/Quaternion.hh>
#include <core/data/basic/Vec3.hh>
#include <core/data/io/Pdb.hh>
#include <core/calc/statistics/Random.hh>

/** @brief ex_Quaternion illustrates how to use Quaternion class
 *
 * CATEGORIES: core::calc::numeric::Quaternion
 * KEYWORDS: algebra
 */
int main(const int argc, const char* argv[]) {

  using namespace core::calc::numeric;
  using namespace core::data::basic; // --- for Vec3

  Quaternion p, rot;
  core::calc::statistics::Random::seed(0);
  rot.random();       // --- Random rotation axis
  //rot = Quaternion::create_from_axis_angle(1,0,0,M_PI/3.0);

  // ---------- Read a structure to be rotated
  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)

```
  const auto strctr_sp = reader.create_structure(0);

  // ---------- Iterate over atoms and rotate them one by one
  for(auto a_it = strctr_sp->first_atom(); a_it != strctr_sp->last_atom(); ++a_it) {
    // --- use quaternion rotation method
    p.i = (**a_it).x;
    p.j = (**a_it).y;
    p.k = (**a_it).z;
    p.rotate_by(rot);
    std::cout << p.i<<" "<<p.j<<" "<<p.k <<"\n";

    // --- and now the same, but with apply() method - output should be numerically␣
↪the same
    rot.apply(**a_it);
    std::cout << (**a_it).to_pdb_line() << "\n";
  }

  return 0;
}
```

ex_REMC_Ar

The program runs an Replica Exchange MC simulation of argon gas. By default it stars 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)

```
#include <utils/options/output_options.hh>
#include <utils/options/sampling_options.hh>

#include <simulations/systems/CartesianAtomsSimple.hh>
#include <simulations/systems/BuildFluidSystem.hh>
#include <simulations/systems/SingleAtomType.hh>
#include <simulations/movers/TranslateAtom.hh>
#include <simulations/forcefields/cartesian/LJEnergySWHomogenic.hh>
#include <simulations/sampling/IsothermalMC.hh>
#include <simulations/observers/ObserveMoversAcceptance.hh>
#include <simulations/observers/TriggerEveryN.hh>
#include <simulations/observers/ObserveReplicaFlow.hh>
#include <simulations/observers/ObserveEnergyComponents.hh>
#include <simulations/observers/UpdateSystemTags.hh>
#include <simulations/observers/AdjustMoversAcceptance.hh>
#include <simulations/observers/cartesian/SimplePdbFormatter.hh>

#include <utils/exit.hh>

using namespace core::data::basic;

utils::Logger logs("ex_REMC_Ar");

std::string program_info = R"(

The program runs an Replica Exchange MC simulation of argon gas. By default it stars 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 ...]

)";

const double EPSILON = 1.654E-21;        // [J] per molecule
const double EPSILON_BY_K = EPSILON / 1.381E-23;         // = 119.6 in Kelvins
const double SIGMA = 3.4;                // in Angstroms

/** @brief A helper function that creates a dummy structure of 830 argon atoms.
 *
 * The structure is necessary to be able to save the system state in the PDB format. It will not be used
 * in the simulation - just for output formatting. The atoms may be stored in multiple chains, because PDB file
 * format allows a single chain to have at most 9999 atoms.
 *
 * CATEGORIES: simulations::sampling::ReplicaExchangeMC
 * KEYWORDS:   Mover;Replica Exchange; Monte Carlo; sampling
 */
core::data::structural::Structure_SP create_argon_structure(const core::index4 n_ar) {

  using namespace core::data::structural;
  Structure_SP s = std::make_shared<Structure>("");
  core::index2 n_chains = n_ar / 9999;
  core::index4 n_atoms = 0;
```

(continues on next page)

(continued from previous page)

```
  for (core::index2 ic = 0; ic < n_chains; ++ic) {
    Chain_SP chain = std::make_shared<Chain>(std::string{utils::letters[ic]});
    for (core::index4 i = 0; i < 9999; ++i) {
      Residue_SP res = std::make_shared<Residue>(i + 1, " AR");
      res->push_back(std::make_shared<PdbAtom>(i + 1, " AR ",␣
↪core::chemical::AtomicElement::ARGON.z));
      chain->push_back(res);
      ++n_atoms;
    }
    s->push_back(chain);
  }
  if (n_atoms < n_ar) {
    Chain_SP chain = std::make_shared<Chain>(std::string{utils::letters[n_chains]});
    for (core::index4 i = n_atoms; i < n_ar; ++i) {
      Residue_SP res = std::make_shared<Residue>(i + 1 - n_atoms, " AR");
      res->push_back(std::make_shared<PdbAtom>(i + 1 - n_atoms, " AR ",␣
↪core::chemical::AtomicElement::ARGON.z));
      chain->push_back(res);
    }
    s->push_back(chain);
  }

  return s;
}

/** @brief Isothermal Monte Carlo simulation of argon gas.
 *
 */
int main(const int argc,const char* argv[]) {

  using core::data::basic::Vec3Cubic;
  using namespace simulations::systems;
  using namespace simulations::observers::cartesian;
  using namespace simulations::forcefields; // for CalculateEnergyBase, NeighborList
  using namespace simulations::movers; // for MoversSet

  // ---------- Define some new types so the program is easier to read and lines get␣
↪shorter
  typedef typename cartesian::LJEnergySWHomogenic LjEnergy;
  typedef typename simulations::observers::ObserveEnergyComponents<TotalEnergy>␣
↪EnergyObserverType;
  typedef std::shared_ptr<EnergyObserverType> EnergyObserverType_SP;

  core::index4 n_atoms;

  if (argc < 8) utils::exit_OK_with_message(program_info);

  std::vector<core::data::structural::Structure_SP> argon_structures;
  bool input_from_file = false;
  if (utils::is_integer(argv[1])) {
    n_atoms = atoi(argv[1]);
    argon_structures.push_back(create_argon_structure(n_atoms));
  }
  else { // --- read an input file if given
    core::data::io::Pdb reader(argv[1]);
    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)

```
    n_atoms = argon_structures[0]->count_atoms();
    input_from_file = true;
  }

  double density = atof(argv[2]);
  core::index4 n_inner_cycles = atoi(argv[3]);
  core::index4 n_outer_cycles = atoi(argv[4]);
  core::index4 n_exchanges = atoi(argv[5]);
  double ar_volume = 4.0 / 3.0 * M_PI * SIGMA * SIGMA * SIGMA * n_atoms;
  double box_len = pow(ar_volume / density, 0.33333333333333);
  core::calc::statistics::Random::seed(1234);

  // --- Initialize periodic boundary conditions for the box length
  core::data::basic::Vec3Cubic::set_box_len(box_len);
  logs << utils::LogLevel::INFO << "box width for "<<int(n_atoms)<<" atoms : " << box_
↪len << "\n";

  std::vector<double> temperatures;
  for (int i = 6; i < argc; ++i) temperatures.push_back(atof(argv[i]));

  AtomTypingInterface_SP ar_type = std::make_shared<SingleAtomType>(" AR");
  std::vector<std::shared_ptr<CartesianAtomsSimple<Vec3Cubic>>> systems;
  core::data::structural::PdbAtom_SP ar_atom = std::make_shared
↪<core::data::structural::PdbAtom>(1," AR");

  std::shared_ptr<AbstractPdbFormatter> fmt = std::make_shared<SimplePdbFormatter>("␣
↪AR ", "AR ", "AR");

  std::vector<simulations::sampling::IsothermalMC_SP> replica_samplers;
  std::vector<CalculateEnergyBase_SP> energies;
  std::vector<simulations::observers::ObserverInterface_SP> mover_adjusters;
  for (core::index2 irepl = 0; irepl < temperatures.size(); ++irepl) {

    // ---------- Create the systems to be sampled ----------
    CartesianAtoms ar(ar_type, n_atoms);
    systems.push_back(ar);

    // ---------- Distribute atoms in the box or use coordinates from provided PDB␣
↪file
    if(input_from_file) {
      const auto strctr = argon_structures[irepl % argon_structures.size()];
      core::index2 ia=0;
      for(auto a_it = strctr->first_const_atom();a_it !=strctr->last_const_atom();++a_
↪it) {
        (*ar)[ia].set(**a_it);
        ++ia;
      }
    }
    else
      BuildFluidSystem<Vec3Cubic>::generate(*ar, *ar_atom, n_atoms);

    // ---------- Create energy function - just LJ potential
    std::shared_ptr<NeighborList_OBSOLETE<Vec3Cubic>> nbl = std::make_shared
↪<NeighborList_OBSOLETE<Vec3Cubic>>(*ar, 9.0, 3.0);
    std::shared_ptr<LjEnergy> lj_energy_term = std::make_shared<LjEnergy>(*ar, *nbl,␣
↪SIGMA, EPSILON_BY_K);
    std::shared_ptr<TotalEnergy_OBSOLETE<ByAtomEnergy_OBSOLETE>> lj_energy =␣
↪std::make_shared<TotalEnergy_OBSOLETE<ByAtomEnergy_OBSOLETE>>();
```

(continues on next page)

(continued from previous page)

```
    lj_energy->add_component(lj_energy_term,1.0);
    energies.push_back(std::static_pointer_cast<class CalculateEnergyBase>(lj_
↪energy));

    // --- 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<Vec3Cubic>> translate = std::make_shared
↪<TranslateAtom<Vec3Cubic>>(*ar, *lj_energy_term);
    MoversSet_SP movers = std::make_shared<simulations::movers::MoversSetSweep>();
    movers->add_mover(translate, n_atoms);
    translate->max_move_range(0.5); // --- set the maximum distance a single atom can␣
↪be moved by a single MC perturbation

    // --- Create an observer to record and adjust movers acceptance rate
    simulations::observers::AdjustMoversAcceptance_SP adj = std::make_shared
↪<simulations::observers::AdjustMoversAcceptance>(
      *movers, utils::string_format("movers-%d.dat", irepl), 0.4);
    mover_adjusters.push_back(adj);
    adj->observe_header();

    // ---------- create an isothermal Monte Carlo sampler
    auto  mc = std::make_shared<simulations::sampling::IsothermalMC>(movers,
↪temperatures[irepl]);

    // ---------- Create an observer for energy components
    EnergyObserverType_SP obs_en
      = std::make_shared<EnergyObserverType>(*lj_energy,utils::string_format("energy-
↪%d.dat",irepl));
    obs_en->observe_header();

    // ---------- Create an observer for trajectory in PDB format
    auto observe_trajectory = std::make_shared
↪<simulations::observers::cartesian::PdbObserver_OBSOLETE<Vec3Cubic>>(
      *ar, fmt, utils::string_format("ar_tra-%d.pdb", irepl));

    observe_trajectory->trigger(std::make_shared
↪<simulations::observers::TriggerEveryN>(10));
    mc->outer_cycle_observer(observe_trajectory);
    mc->outer_cycle_observer(obs_en);
    mc->cycles(n_inner_cycles,n_outer_cycles,1);
    replica_samplers.push_back(mc);
  }

  bool replica_isothermal_observation_mode = true;
  simulations::sampling::ReplicaExchangeMC remc(replica_samplers, energies, replica_
↪isothermal_observation_mode);
  auto remc_flow = std::make_shared<simulations::observers::ObserveReplicaFlow>(remc,
↪"replica_flow.dat");
  remc.exchange_observer(remc_flow);
  auto tag_updater = std::make_shared<simulations::observers::UpdateSystemTags
↪<CartesianAtomsSimple<Vec3Cubic>>>(systems,remc);
  tag_updater->observe();
  remc.exchange_observer(tag_updater);

  remc.replica_exchanges(n_exchanges);
  for (auto o : mover_adjusters) remc.exchange_observer(o);
  remc.run();
```

(continues on next page)

(continued from previous page)

```
210
211   simulations::observers::cartesian::PdbObserver_OBSOLETE<Vec3Cubic>␣
↪final(*systems[0], fmt, "final.pdb");
212   final.observe();
213   for (core::index2 i_system = 1; i_system < systems.size(); ++i_system) {
214     for (core::index4 i_atom = 0; i_atom < systems[0]->n_atoms; ++i_atom)
215       systems[0]->operator[](i_atom) = systems[i_system]->operator[](i_atom);
216     final.observe();
217   }
218   final.finalize();
219 }
```

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:

```
#include <iostream>
#include <iomanip>

#include <core/data/sequence/ReduceSequenceAlphabet.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief ex_ReduceSequenceAlphabet lists all reduced amino acid alphabets registered in BioShell library
```

(continues on next page)

(continued from previous page)

```
 *
 * CATEGORIES: core::data::sequence::ReduceSequenceAlphabet
 * KEYWORDS:   reduced alphabet
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using core::data::sequence::ReduceSequenceAlphabet;

  int i = 1;
  std::cout << "Known alphabets:\n";
  for (auto it = ReduceSequenceAlphabet::cbegin(); it !=␣
↪ReduceSequenceAlphabet::cend(); ++it) {
    std::cout << std::setw(8) << it->first << ((i % 10 == 0) ? "\n" : " ");
    ++i;
  }
  std::cout << "\n";
  for (int i = 1; i < argc; ++i) {
    std::cout << "Listing alphabet " << argv[i] << ":\n";
    core::data::sequence::ReduceSequenceAlphabet_SP alph =␣
↪ReduceSequenceAlphabet::get_alphabet(argv[i]);
    std::cout << (*alph) << "\n";
  }
}
```

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:

```
#include <iostream>
#include <iomanip>
#include <core/data/io/Pdb.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Simple example reads a PDB file and checks if all amino acid residues have complete backbone.
EXAMPLE:
    ex_Residue 5edw.pdb

)";

/** @brief Reads a PDB file and checks if all amino acid residues have complete backbone
 *
 * CATEGORIES: core::data::structural::Structure; core::data::structural::Residue
 * KEYWORDS:   PDB input; pre-processing
 */
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();␣
    ↪++it_resid) {
29        const core::chemical::Monomer & m = (*it_resid)->residue_type();
30        core::data::structural::PdbAtom_SP atom_sp;
31        if(m.type=='P') {
32          atom_sp = (*it_resid)->find_atom(" N  ");
33          if(atom_sp==nullptr) { std::cout << "Missing backbone atom N \n"; is_OK =␣
    ↪false; }
34          atom_sp = (*it_resid)->find_atom(" CA ");
35          if(atom_sp==nullptr) { std::cout << "Missing backbone atom CA \n"; is_OK =␣
    ↪false; }
36          atom_sp = (*it_resid)->find_atom(" C  ");
37          if(atom_sp==nullptr) { std::cout << "Missing backbone atom C \n"; is_OK =␣
    ↪false; }
38          atom_sp = (*it_resid)->find_atom(" O  ");
39          if(atom_sp==nullptr) { std::cout << "Missing backbone atom O \n"; is_OK =␣
    ↪false; }
40        }
41      }
42      if(is_OK) std::cout << "Backbone complete!\n";
43  }
```

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

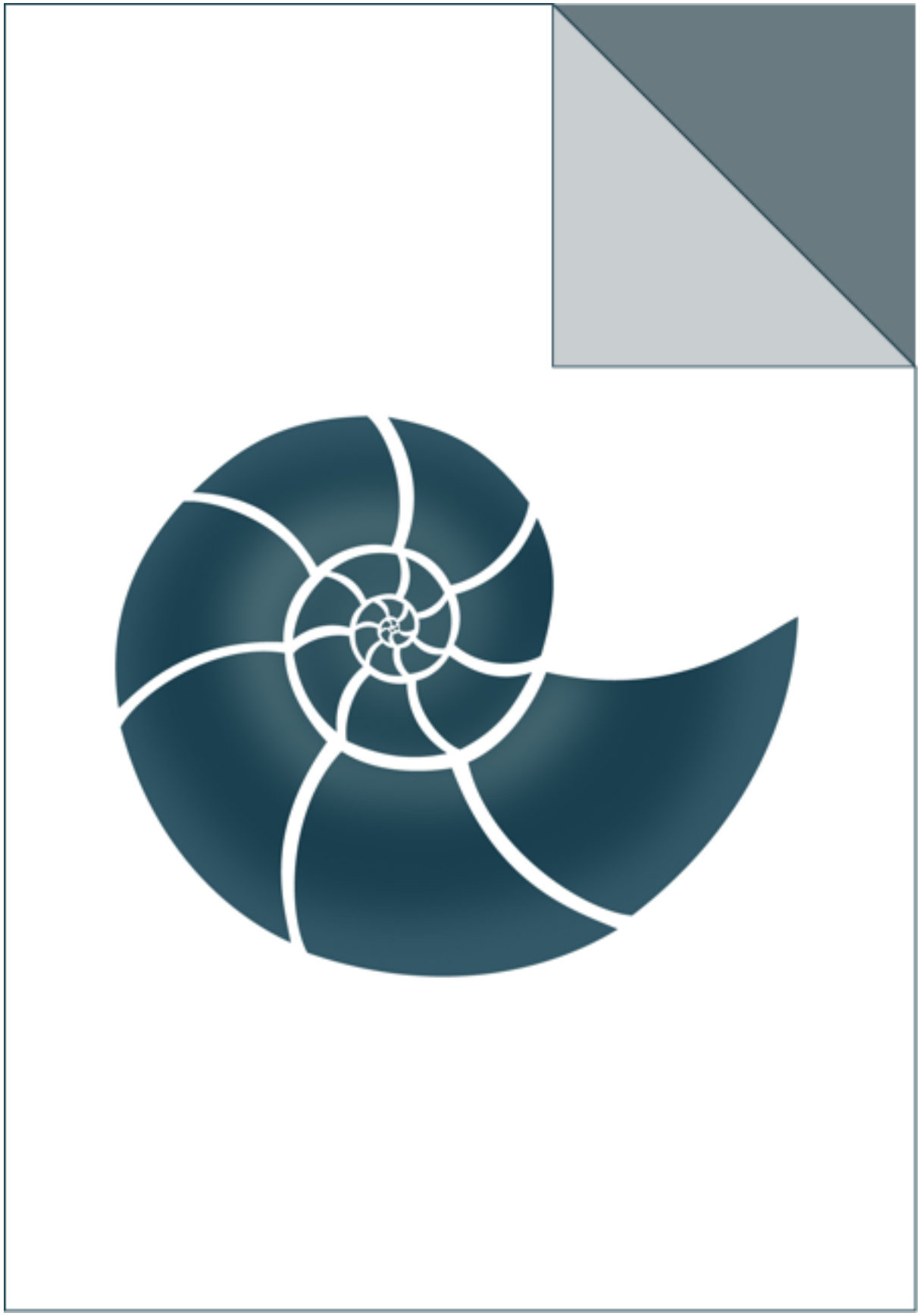
Program source:

```
#include <math.h>

#include <iostream>
#include <random>
#include <vector>

#include <core/calc/statistics/NormalDistribution.hh>
#include <core/calc/statistics/RobustDistributionDecorator.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

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
)";
```

(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;␣
    ↪core::calc::statistics::RobustDistributionDecorator
29   * KEYWORDS: estimation
30   */
31  int main(const int argc, const char* argv[]) {
32
33    if ((argc > 1) && utils::options::call_for_help(argv[1]))
34      utils::exit_OK_with_message(program_info);
35
36    using namespace core::calc::statistics;
37
38    unsigned int rd = 9876543;
39    std::mt19937 gen(rd);
40    core::index4 N = 10000; //--- the number of random points to use in tests
41    core::index4 Nnoise = 10; //--- the number of random points from the noise␣
    ↪distribution
42
43    // ---------- The two distributions used in this test
44    std::normal_distribution<> base(2.0, 0.5); // --- the "base" distribution
45    std::normal_distribution<> noise(2.0, 50); // --- the "noise" distribution
46
47    // ----------
48    std::vector<std::vector<double> > random_points;
49
50    if (argc==1) { // ---------- Generate random sample
51      for (core::index4 i = 0; i < N; ++i)
52        random_points.emplace_back( std::initializer_list<double>{base(gen)} );
53      for (core::index4 i = 0; i < Nnoise; ++i)
54        random_points.emplace_back( std::initializer_list<double>{noise(gen)} );
55    } else {    // ---------- Read data from a file
56      std::fstream infile(argv[1], std::ios_base::in);
57      double a;
58      while (infile >> a) {
59        random_points.emplace_back(std::initializer_list<double>{a});
60      }
61    }
62
63    std::vector<double> init_params{1.0,0.0}; // --- initial parameters of the␣
    ↪estimated distribution
64    NormalDistribution n(init_params);
65    n.estimate(random_points);
66    std::cout << "Estimated:           " << n << "\n";
67    RobustDistributionDecorator<NormalDistribution> rn(init_params, 0.05);
68    rn.estimate(random_points);
69    std::cout << "Estimated (robust): "<< rn << "\n";
70    if (argc==1)  std::cout << "True distribution:  2.0   0.5\n";
71  }
```



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:

```
#include <iostream>
#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/SelectChainBreaks.hh>
#include <utils/exit.hh>
#include <utils/LogManager.hh>


std::string program_info = R"(

Reads a PDB file and prints list of chain breaks found in every chain

EXAMPLE:
    ex_SelectChainBreaks 4mcb.pdb

)";

/** @brief Reads a PDB file and prints a list of chain breaks found in every chain
 *
 * CATEGORIES: core::data::structural::SelectChainBreaks
```

(continues on next page)

(continued from previous page)

```
 * KEYWORDS:   structure selectors; PDB input
 */
int main(const int argc, const char *argv[]) {

  using namespace core::data::structural::selectors;
    utils::LogManager::INFO();


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

  // ---------- Read a PDB file and create a Structure object
    core::data::io::PdbLineFilter filt1 = core::data::io::all_true(core::data::io::is_ca, core::data::io::is_standard_atom,core::data::io::is_not_alternative);
    core::data::io::PdbLineFilter filt2 = core::data::io::all_true(core::data::io::is_ca, core::data::io::is_hetero_atom,core::data::io::is_not_alternative);
    core::data::io::PdbLineFilter filt3 = core::data::io::one_true(filt1, filt2);


    core::data::io::Pdb reader(argv[1],filt3);
  auto strctr = reader.create_structure(0);

  SelectChainBreaks sel;
  bool chain_is_OK = true;
  for (const auto &chain : *strctr) {
    if (std::distance(chain->begin(),chain->terminal_residue()) < 3) {
      std::cerr << "Chain " << chain->id() << " of " << strctr->code() << " is too short\n";
      continue;
    }
    if (chain->count_aa_residues() < chain->size() * 0.8) {
      std::cerr << "Chain " << chain->id() << " of " << strctr->code() << " is not a protein\n";
      continue;
    }
    std::cerr << "# Processing " << strctr->code() << " chain " << chain->id() << "\n";
    size_t first_aa = 0;
    while ((*chain)[first_aa]->residue_type().type != 'P') ++first_aa;
    const auto last_it = chain->terminal_residue() - 1;
    for (auto res_it = chain->begin() + first_aa + 1; res_it != last_it; ++res_it) {
      auto next = (**res_it).next();
      if(next== nullptr) {
        next = *(++std::find(chain->begin(),chain->end(),*res_it));
        std::cerr << "Residue following "<<(**res_it)<<" is not an amino acid!\n";
      }
      auto prev = (**res_it).previous();
      if(prev == nullptr) {
        prev = *(--std::find(chain->begin(),chain->end(),*res_it));
        std::cerr << "Residue preceding "<<(**res_it)<<" is not an amino acid!\n";
      }

      if (sel(*res_it)) {
        chain_is_OK = false;
        if (sel.last_chainbreak_type == RIGHT) {
          std::cout << utils::string_format("%s %4s %4d%c %4d%c %6.2f\n", 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.
↪right_side_distance);
71            ++res_it;
72            if (res_it == last_it) break;
73          }
74          if (sel.last_chainbreak_type == LEFT) {
75            std::cout << utils::string_format("%s %4s %4d%c %4d %c %6.2f\n", strctr->
↪code().c_str(), chain->id().c_str(),
76                prev->id(), prev->icode(), (*res_it)->id(), (*res_it)->icode(), sel.
↪left_side_distance);
77          }
78          if (sel.last_chainbreak_type == BOTH) {
79            std::cout << utils::string_format("%s %4s %4d%c %4d %c %6.2f %6.2f\n",␣
↪strctr->code().c_str(), chain->id().c_str(),
80                prev->id(), prev->icode(), (*res_it)->id(), (*res_it)->icode(), sel.
↪left_side_distance);
81            std::cout << utils::string_format("%s %4s %4d%c %4d %c\n", strctr->code().c_
↪str(), chain->id().c_str(),
82                (*res_it)->id(), (*res_it)->icode(), next->id(), next->icode(), sel.
↪right_side_distance);
83            ++res_it;
84            if (res_it == last_it) break;
85          }
86        }
87      }
88      if(chain_is_OK) std::cout << utils::string_format("%s %4s OK\n", strctr->code().c_
↪str(), chain->id().c_str());
89    }
90  }
```

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:

```
#include <iostream>
#include <sstream>
#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Simple example showing how to select a structural fragment based on residue IDs

USAGE:
./ex_SelectResidueRange

)";

/** @brief Shows how to select a structural fragment based on residue IDs
 *
 * CATEGORIES:  core::data::structural::SelectResidueRange
 * KEYWORDS:   structure selectors; PDB input; STL; algorithms
 */
// --- Only C-alpha atoms are listed here to keep this example short and simple
std::string fragment =
    R"(ATOM    312  CA  ALA A  -1     -10.035   4.811   1.920  1.00  0.24           C␣
↪
ATOM    322  CA  VAL A   0     -13.437   5.248   0.258  1.00  0.33           C
```

(continues on next page)

(continued from previous page)

```
ATOM    338  CA  ASP A   1     -12.201   3.975  -3.121  1.00  0.24           C
ATOM    350  CA  ALA A   1A     -9.237   2.226  -4.777  1.00  0.18           C
ATOM    360  CA  ALA A   1B     -7.956   5.461  -6.338  1.00  0.24           C
ATOM    370  CA  THR A   2      -7.460   7.449  -3.135  1.00  0.21           C
ATOM    384  CA  ALA A   3      -6.080   4.229  -1.648  1.00  0.12           C
)"
;

int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::data::structural;

  std::stringstream in(fragment);                    // Create an input stream that will provide data from a string
  core::data::io::Pdb reader(in);
  Structure_SP strctr = reader.create_structure(0);

  std::cout << "IDs of the residues available for selection:";
  std::for_each(strctr->first_residue(), strctr->last_residue(), [](Residue_SP r){std::cout << r->residue_id()<<" ";});
  std::cout << "\n";

  selectors::SelectResidueRange range0("-1-1");
  std::cout << "selector " << range0.selector_string() << " selects: "
      << std::count_if(strctr->first_residue(), strctr->last_residue(), range0) << " residues\n";

  selectors::SelectResidueRange range1("-1-1A");
  std::cout << "selector " << range1.selector_string() << " selects: "
      << std::count_if(strctr->first_residue(), strctr->last_residue(), range1) << " residues\n";

  selectors::SelectResidueRange range2("*");
  std::cout << "selector " << range2.selector_string() << " selects: "
      << std::count_if(strctr->first_residue(), strctr->last_residue(), range2) << " residues\n";
}
```

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:

```
#include <iostream>
#include <chrono>
#include <algorithm>

#include <core/data/io/fasta_io.hh>

#include <core/data/sequence/Sequence.hh>
#include <core/alignment/SemiglobalAligner.hh>
#include <core/alignment/PairwiseAlignment.hh>
#include <core/alignment/PairwiseSequenceAlignment.hh>
#include <core/alignment/scoring/SimilarityMatrix.hh>
#include <core/alignment/scoring/SimilarityMatrixScore.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

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

)";

using namespace core::data::sequence;
using namespace core::alignment::scoring;

```

(continues on next page)

(continued from previous page)

```
/// An example score function used by BioShell pairwise sequence alignment methods.
/** Such a scoring object must provide three components:
 *   - a scoring operator, whose arguments are positions in the scored sequences (query and template, respectively)
 *   - query_length() method, and
 *   - tmplt_length() method
 */
struct IdentityScore {
  IdentityScore(const Sequence & query,const Sequence & tmplt) : q(query.sequence), t(tmplt.sequence) {}

  /// Alignment score is 1 when the two compared letters are identical and 0 otherwise
  short operator()(const core::index2 i,const core::index2 j) const { return q[i]==t[j]; }
  /// Returns the length of a template sequence
  core::index2 tmplt_length() const { return t.length(); }
  /// Returns the length of a query sequence
  core::index2 query_length() const { return q.length(); }

  const std::string & q;
  const std::string & t;
};

/** @brief  Calculate a pairwise sequence alignment between two sequences with identity scoring method.
 *
 * The program 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
 *
 * CATEGORIES: core::alignment::SemiglobalAligner; core::alignment::PairwiseSequenceAlignment
 * KEYWORDS:   sequence alignment
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  Sequence_SP query = std::make_shared<Sequence>("query","CATACGTCGACGGCT",1);
  Sequence_SP tmplt = std::make_shared<Sequence>("tmplt","ACGACGT",1);

  // --- create aligner object
  core::index2 max_len = std::max(query->length(),tmplt->length());
  core::alignment::SemiglobalAligner<short, IdentityScore> aligner(max_len);

  // --- find score of the alignment; just the score - this is faster than aligning and keeping backtracking info
  IdentityScore s(*query, *tmplt);
  short result1 = aligner.align_for_score(-10, -1, s);

  short result2 = aligner.align(-10, -1, s);
  core::alignment::PairwiseAlignment_SP ali = aligner.backtrace();
  std::cerr << ali->query_length() << " " << query->sequence << " " << ali->template_length() << " " << tmplt->sequence
       << "\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 << " " <<␣
      ↪result2 << " " << result3 << "\n"
81         << seq_ali << "\n";
82  }
```

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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/data/io/fasta_io.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Reads a PDB file and extracts its SEQRES sequence(s)
 * CATEGORIES: core::data::io::Pdb
 * KEYWORDS:   PDB input; Structure; sequence
```

(continues on next page)

(continued from previous page)

```
 */
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(argv[1], // file name (PDB format, may be gzip-ped)
      is_ca,          // read only CA atoms
      keep_all, true);          //  parse PDB header !

  std::shared_ptr<Seqres> seq_res = std::static_pointer_cast<Seqres>(reader.header.find("SEQRES")->second);
  for(const auto & chain_seq : seq_res->sequences) {
    const std::string header = reader.pdb_code()+" : "+chain_seq.first;
    core::data::sequence::Sequence_SP s = seq_res->create_sequence(chain_seq.first, header);
    if((s->length()>20) && (s->get_monomer(2).type=='P'))
      std::cout << core::data::io::create_fasta_string(*s,-1)<<"\n";
  }
}
```

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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/data/io/fasta_io.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Reads a PDB file and prints a fragment of its sequence.
 *
 * CATEGORIES: core/data/io/Sequence
 * KEYWORDS:   PDB input; sequence
```

(continues on next page)

(continued from previous page)

```
 */
int main(const int argc, const char* argv[]) {

  if(argc < 4) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  using namespace core::data::io;

  // --- Try this test program with 3wn7 as the input structure !
  Pdb reader(argv[1], // file name (PDB format, may be gzip-ped)
    all_true(is_not_hydrogen,is_not_water),          // don't read hydrogens, skip water molecules
      core::data::io::keep_all, true);          //  parse PDB header !

  core::index2 from = atoi(argv[3]);
  core::index2 to = atoi(argv[4]);
  auto structure = reader.create_structure(0);
  auto chain = structure->get_chain(argv[2][0]);
  auto sequence = chain->create_sequence();

  // --- Should be 324 (324 is the ID of the very first residue of 3wn7 chain A)
  std::cout << "Id of the first residue: " << sequence->first_pos() << "\n";
  Sequence fragment(*sequence, from, to); // Cut from residue whose ID is 366
  std::cout << fragment.first_pos() << " " << fragment.sequence << "\n";
}
```

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:

```
#include <memory>
#include <iostream>
#include <random>

#include <core/calc/statistics/Random.hh>

#include <simulations/movers/Mover.hh>
#include <simulations/movers/MoversSetSweep.hh>
#include <simulations/evaluators/EchoEvaluator.hh>
#include <simulations/evaluators/CallEvaluator.hh>
#include <simulations/sampling/SimulatedAnnealing.hh>

#include <simulations/observers/ObserveEvaluators.hh>

/** @brief  Models a point particle in 1D space with harmonic energy.
 * This simple class implements the Mover interface. One has to implement just two
 * essential methods: <code>move()</code> and <code>undo()</code>. For simplicity, in this example energy calculation
 * is implemented within the <code>move()</code> method.
 */
class HarmonicSystemMover : public simulations::movers::Mover {
public:
  const double x0;         ///< Initial position of the particle, where energy is 0.0
  double x;                ///< Actual position of the particle at the end of the spring
  double recent_energy;    ///< Actual energy of the spring

  HarmonicSystemMover() : simulations::movers::Mover("HarmonicSystemMover"), x0(0.0) {}
```

(continues on next page)

(continued from previous page)

```

  /** @brief  Moves the particle randomly in either direction.
   * This method is declared abstract in Mover class and must be implemented here
   */
  virtual bool move(simulations::sampling::AbstractAcceptanceCriterion &mc_scheme)  {

    double old_en = (x - x0) * (x - x0);
    double delta_x = 0.1 - 0.2 * rand_coordinate(generator);
    x += delta_x;
    double new_en = (x - x0) * (x - x0);
    inc_move_counter();

    if (!mc_scheme.test(old_en, new_en)) {
      undo();
      recent_energy = old_en;
      return false;
    } else {
      recent_energy = new_en;
      return true;
    }
  }

  /** @brief Back up the most recent move.
   * This method is declared abstract in Mover class and must be implemented here
   */
  inline void undo() {
    dec_move_counter();
    x -= delta_x;
  }

  /// Yet another method inherited from the base class
  virtual const std::string &name() const { return name_; }

  /// Does nothing, but must be implemented since it's been declared as virtual in␣
↪the base class
  void max_move_range(const double max_range) {}

  /// Reads the maximum range for a move
  virtual double max_move_range() const { return 0.1; }

private:
  double delta_x;
  std::uniform_real_distribution<double> rand_coordinate;
  core::calc::statistics::Random &generator = core::calc::statistics::Random::get();
  static std::string name_;
};

std::string HarmonicSystemMover::name_ = "HarmonicSystemMover";

/** @brief A simple example shows how to use Monte Carlo simulated annealing.
 *
 * This demo also shows how to implement a simple mover and how to hook it up to␣
↪sampling protocol.
 * CATEGORIES: simulations/generic/movers/Mover; simulations/generic/sampling/
↪SimulatedAnnealing; simulations/generic/evaluators/EchoEvaluator;
 * CATEGORIES: simulations/generic/evaluators/EchoEvaluator; simulations/generic/
↪evaluators/CallEvaluator
```

(continues on next page)

(continued from previous page)

```
 * KEYWORDS:   Monte Carlo; Mover; sampling; simulated annealing; evaluators
 */
int main(const int argc, const char *argv[]) {

  using namespace simulations::evaluators;

  // ---------- Here we create a system to me modelled
  std::shared_ptr<HarmonicSystemMover> harmonic_ptr = std::make_shared
↪<HarmonicSystemMover>();
  // --- You need a mover set to use SimulatedAnnealing protocol, even if the set␣
↪contains just one mover
  simulations::movers::MoversSet_SP moves = std::make_shared
↪<simulations::movers::MoversSetSweep>();
  moves->add_mover(harmonic_ptr,1);

  simulations::sampling::SimulatedAnnealing sa(moves,{2.0,1.5,1.0});

  // ---------- Create an observer which calls evaluators and writes the observations␣
↪on the screen
  std::shared_ptr<simulations::observers::ObserveEvaluators> obs = std::make_shared
↪<simulations::observers::ObserveEvaluators>("");
  // --- Create an evaluator which just return the X position of the particle
  // --- Add the evaluator to the observer. Obviously there might be may evaluators␣
↪added to this observer;
  // --- then a nice table will be printed with a column corresponding to an␣
↪evaluator.
  obs->add_evaluator(std::make_shared<EchoEvaluator<double>>(harmonic_ptr->x,
↪"position X"));
  obs->add_evaluator(std::make_shared<EchoEvaluator<double>>(harmonic_ptr->recent_
↪energy,"energy"));

  std::function<double(void)> get_temperature = [&sa]() { return sa.temperature(); };
  obs->add_evaluator(
    std::make_shared<CallEvaluator<std::function<double(void)>>>(get_temperature,
↪"temperature"));

  sa.outer_cycle_observer(obs);
  sa.cycles(100,100);
  sa.run();
}
```

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:

```
#include <iostream>
#include <iomanip>
#include <core/algorithms/predicates.hh>
#include <core/data/io/Pdb.hh>
#include <utils/exit.hh>

std::string program_info = R"(

ex_Structure reads a PDB file and prints a list of all atoms grouped by residues they belong to
EXAMPLE:
    ./ex_Structure 5edw.pdb

)";
```

(continues on next page)

(continued from previous page)

```

/** @brief Reads a PDB file and prints a list of all atoms grouped by residues they belong to.
 *
 * CATEGORIES: core::data::structural::Structure
 * KEYWORDS:   PDB input; Structure; Chain; Residue; PdbAtom; STL
 */
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;

    core::data::io::Pdb reader(argv[1],is_not_alternative); // file name (PDB format, may be gzip-ped)
    core::data::structural::Structure_SP strctr = reader.create_structure(0);

    // Iterate over all chains
    for (auto it_chain = strctr->begin(); it_chain!=strctr->end(); ++it_chain) {
      std::cout << "---------- chain " << (*it_chain)->id() << " ----------\n";
      // Iterate over all residues
      for (auto it_res = (*it_chain)->begin(); it_res!=(*it_chain)->end(); ++it_res) {
        std::cout << std::setw(5)<<(*it_res)->id()<<" "<<(*it_res)->residue_type().code3<<" :";
        for (auto it_atom = (*it_res)->begin(); it_atom!=(*it_res)->end(); ++it_atom) {
        if (((*it_atom)->alt_locator() == ' ') || ((*it_atom)->alt_locator() == 'A'))
          std::cout << " " << (*it_atom)->atom_name();
        }
        std::cout <<"\n";
      }
    }
}
```

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:

```
#include <iostream>

#include <utils/ThreadPool.hh>
#include <utils/string_utils.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Unit test which shows how to use a ThreadPool class.

USAGE:
./ex_ThreadPool

)";

/// Operator called by each thread (pretends to run very time consuming calculations)
struct Op { std::string operator()(int i) { return "Hello " + utils::to_string(i); } }
 ↪;

/** @brief Simple test for a ThreadPool class
 *
 * CATEGORIES: utils/ThreadPool
 * KEYWORDS:  concurrency; multi-threading
 */
int main(const int cnt, const char *argv[]) {

  if ((cnt > 1) && utils::options::call_for_help(argv[1]))
    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:

```
#include <iostream>
#include <string>

#include <core/data/basic/ThreadSafeMap.hh>

/** @brief Shows how to use ThreadSafeMap class
 *
 * CATEGORIES: core::data::basic::ThreadSafeMap
 * KEYWORDS:   data container
 */
int main(const int argc, const char* argv[]) {

  core::data::basic::ThreadSafeMap<std::string,int> map;
  int one = 1, two = 2;
  map.insert_or_assign("one",one);
  map.insert_or_assign("two",two);
}
```

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:

```
#include <memory>
#include <iostream>
#include <random>

#include <core/algorithms/trees/kd_tree.hh>
#include <core/algorithms/trees/BinaryTreeNode.hh>
#include <core/algorithms/trees/algorithms.hh>

#include <core/data/basic/Vec3.hh>
#include <core/calc/statistics/Random.hh>

using core::data::basic::Vec3;
using namespace core::algorithms::trees;
using namespace core::calc::statistics;


/// Tree traversal operation prints each node on the screen
struct PrintPoint {
  void operator()(std::shared_ptr<BinaryTreeNode<KDTreeNode<Vec3>> > node) {
    std::cout << node->element.element << " " << node->element.level << "\n";
  }
};


/** @brief A simple example shows how to use BioShell kd-tree routines.
 *
 * The program generates N=500 random points and partites them into KD-tree. Later␣
↪the tree is used to find spatal
 * neighbors in 3D space.
 *
 * CATEGORIES: core::algorithms::trees::BinaryTreeNode; core::algorithms::trees::kd_
↪tree.hh
```

(continues on next page)

(continued from previous page)

```
 * KEYWORDS:   neighborhood detection; data structures; algorithms
 */
int main(const int argc, const char* argv[]) {

  // ---------- Here we generate N random points in 3D space to be partitioned
  const unsigned short N = 5000;
  Random::seed(0);                              // seed random number generator
  Random & gen = Random::get();                 // get rnd generator singleton
  UniformRealRandomDistribution<double> rnd;  // uniform distribution will be used to assign coordinates
  std::vector<Vec3> atoms;                      // container for the points

  // ---------- We use <code>emplace_back()</code> to create Vec3 objects directly in the container
  for(unsigned int i=0;i<N;++i) atoms.emplace_back(rnd(gen),rnd(gen),rnd(gen));

  // ---------- Here the actual kd-tree is constructed
  std::shared_ptr<BinaryTreeNode<KDTreeNode<Vec3>> > root = create_kd_tree<Vec3, std::vector<Vec3>::iterator, CompareAsReferences<Vec3>>(atoms.begin(),atoms.end());

  // ---------- Here we divide all the points into \f$2^2=4\f$ groups
  std::vector<std::shared_ptr<BinaryTreeNode<KDTreeNode<Vec3>>>> node_groups;
  collect_given_level(root, 2, node_groups);
  unsigned short subcluster_id = 0;
  for(const auto node:node_groups) {    // then we mark all nodes in each subtree by a distinct ID number
    depth_first_preorder(node, [subcluster_id](std::shared_ptr<BinaryTreeNode<KDTreeNode<Vec3>>> node) {
      node->element.level = subcluster_id; });
    ++subcluster_id;
  }

  breadth_first_preorder(root, PrintPoint()); // finally, each node is printed

  // ---------- Here is an example how to search the tree
  Vec3 query(0.7,0.7,0.4); // a query point

  // ---------- Here is an example how to find the closes element
  Vec3 best_point;
  double d = search_kd_tree(root, [](const Vec3 &v1, const Vec3 &v2) { return v1.distance_to(v2); }, query, best_point);
  std::cout << "point closest to query: " << best_point << " : " << d << "\n";

  Vec3 q_low(0.68,0.68,0.38);
  Vec3 q_up(0.8,0.8,0.45);
  std::vector<Vec3> hits;
  search_kd_tree(root,  q_low, q_up, 3, hits);
  std::cout << "point within a box bounded by: " << q_low << " and " << q_up << ":\n";
  for (const auto &p:hits) std::cout << p << "\n";
}
```

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:

```
#include <memory>
#include <iostream>
#include <string>

#include <core/algorithms/trees/TreeNode.hh>
#include <core/algorithms/trees/algorithms.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Unit test which shows how to use a TreeNode data structure defined in BioShell

USAGE:
./ex_TreeNode

)";

/** @brief Simple demo for TreeNode class
 *
 * This program creates a small tree with 7 nodes
 *
 * CATEGORIES: core::algorithms::trees::TreeNode
 * KEYWORDS:   algorithms; data structures; graphs
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
```

(continues on next page)

(continued from previous page)

```
    utils::exit_OK_with_message(program_info);

  using namespace core::algorithms::trees;

  std::shared_ptr<TreeNode<std::string>> n1 = std::make_shared<TreeNode<std::string>>("A",1);
  std::shared_ptr<TreeNode<std::string>> n2 = std::make_shared<TreeNode<std::string>>("B",2);
  std::shared_ptr<TreeNode<std::string>> n3 = std::make_shared<TreeNode<std::string>>("C",3);
  std::shared_ptr<TreeNode<std::string>> n4 = std::make_shared<TreeNode<std::string>>("D",4);
  std::shared_ptr<TreeNode<std::string>> n5 = std::make_shared<TreeNode<std::string>>("E",5);
  std::shared_ptr<TreeNode<std::string>> n6 = std::make_shared<TreeNode<std::string>>("F",6);
  std::shared_ptr<TreeNode<std::string>> n7 = std::make_shared<TreeNode<std::string>>("G",7);
  n1->add_branch(n2);
  n1->add_branch(n3);
  n2->add_branch(n4);
  n2->add_branch(n5);
  n2->add_branch(n7);
  n5->add_branch(n6);
  depth_first_preorder(n1,[](std::shared_ptr<TreeNode<std::string>> n){ std::cout << n->id<< "\n";});

  std::cout << "Size of the tree: " << size(n1)<<"\n";

  return 0;
}
```

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:

```
#include <memory>
#include <iostream>

#include <core/algorithms/UnionFind.hh>
#include <core/calc/statistics/Random.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Unit test which shows how to use the Union-Find algorithm.

USAGE:
./ex_UnionFind

REFERENCE:
    https://en.wikipedia.org/wiki/Disjoint-set_data_structure

)";

// ---------- Data type of objects that will be clustered
struct Point2D {
  float x, y;

  Point2D(float nx, float ny) : x(nx), y(ny) {}

```

(continues on next page)

(continued from previous page)

```
  float distance_to(const Point2D &p) { return sqrt((x - p.x) * (x - p.x) + (y - p.y) * (y - p.y)); }
};

// ---------- this operator is necessary because core::algorithms::UnionFind keeps std::map of data points
bool operator<(const Point2D &lhs, const Point2D &rhs) { return lhs.x < rhs.x; }

/** @brief A simple example shows how to use UnionFind algorithm.
 *
 * The program calculates greedy clustering of points in 2D. The number of points can be provided from command line
 *
 * CATEGORIES: core::algorithms::UnionFind;
 * KEYWORDS:   data structures; data structures; algorithms
 */
int main(const int argc, const char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::calc::statistics;

  // ---------- Here we generate N random points in 3D space to be partitioned
  const unsigned short N = (argc > 1) ? atoi(argv[1]) : 500;
  const float cutoff = (argc > 2) ? atof(argv[2]) : 0.05;
  std::vector<Point2D> points;        // container for the points

  Random::seed(0);                              // seed random number generator
  Random &gen = Random::get();                  // get rnd generator singleton
  UniformRealRandomDistribution<double> rnd;  // uniform distribution will be used to assign coordinates

  core::algorithms::UnionFind<Point2D, core::index2> uf;
  for (unsigned int i = 0; i < N; ++i) {
    points.emplace_back(rnd(gen), rnd(gen)); // we use <code>emplace_back()</code> to create point objects directly in the container
    uf.add_element(points.back());
    for (unsigned int j = 0; j < i; ++j) {
      if (points[i].distance_to(points[j]) < cutoff) uf.union_set(i, j);
    }
  }

  std::cout << "# x-coord    y-coord  cluster_assignment\n";
  for (unsigned int i = 0; i < N; ++i)
    std::cout << points[i].x << " " << points[i].y << " " << uf.find_set(i) << "\n";
}
```

ex_WL_Ar

The program runs a Wang-Landau MC simulation of argon gas. By default it stars 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:

```
#include <iostream>
#include <thread>

#include <core/data/basic/Vec3Cubic.hh>

#include <utils/string_utils.hh>
#include <utils/options/OptionParser.hh>

#include <simulations/systems/CartesianAtoms.hh>
#include <simulations/systems/BuildFluidSystem.hh>
#include <simulations/systems/SingleAtomType.hh>

#include <simulations/movers/TranslateAtom.hh>

#include <simulations/forcefields/cartesian/LJEnergySWHomogenic.hh>

#include <simulations/sampling/WangLandauSampler.hh>

#include <simulations/observers/ObserveEvaluators.hh>
#include <simulations/observers/cartesian/PdbObserver.hh>
#include <simulations/observers/ObserveWLSampling.hh>
#include <simulations/observers/AdjustMoversAcceptance.hh>
#include <simulations/observers/TriggerEveryN.hh>
#include <simulations/observers/cartesian/SimplePdbFormatter.hh>

#include <simulations/evaluators/CallEvaluator.hh>

using namespace core::data::basic;

utils::Logger logs("ex_WL_Ar");

std::string program_info = R"(

The program runs a Wang-Landau MC simulation of argon gas. By default it stars from a␣
↪regular lattice conformation
```

(continues on next page)

(continued from previous page)

```
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]

)";

const double EPSILON = 1.654E-21;        // [J] per molecule
const double EPSILON_BY_K = EPSILON / 1.381E-23;         // = 119.6 in Kelvins
const double SIGMA = 3.4;                // in Angstroms

inline int bin_from_energy(double E)
{
    return (int)(E / 100);
}

/** @brief Isothermal Monte Carlo simulation of argon gas.
 *
 */
int main(const int argc,const char* argv[]) {

    using core::data::basic::Vec3Cubic;
    using namespace simulations::systems;
    using namespace simulations::movers; // for MoversSet
    using namespace simulations::observers::cartesian; // for all observers

    core::index4 n_outer_cycles = 1000;
    core::index4 n_inner_cycles = 1000;
    double density = 0.5;     // density of the system controls how many atoms will be contained in the box
    double temperature = 97;  // in Kelvins
    core::index4 n_atoms = 256;
    double max_jump = 0.5;                // Random move range (in Angstroms)

    core::data::structural::Structure_SP argon_structure = nullptr;
    core::data::structural::PdbAtom_SP ar_atom = std::make_shared<core::data::structural::PdbAtom>(1," AR");
    if (argc < 6) std::cerr << program_info;
    else {
        if (utils::is_integer(argv[1])) n_atoms = atoi(argv[1]);
        else { // --- read an input file if given
            core::data::io::Pdb reader(argv[1]);
            argon_structure = reader.create_structure(0);
            n_atoms = argon_structure->count_atoms();
        }
        density = atof(argv[2]);
        temperature = atof(argv[3]);
        n_inner_cycles = atoi(argv[4]);
        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, 0.33333333333333);

    // --- Initialize periodic boundary conditions
    core::data::basic::Vec3Cubic::set_box_len(box_len);
    logs << utils::LogLevel::INFO << "box width for " << int(n_atoms) << " atoms : " << 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,␣
↪EPSILON_BY_K);
107
108   // --- Create a mover, which is a random perturbation of an atom in this case, and␣
↪place it in a movers' set
109   std::shared_ptr<TranslateAtom> translate = std::make_shared<TranslateAtom>(ar, ar_
↪backup, lj_energy);
110   translate->max_move_range_allowed(1.5);
111   MoversSet_SP movers = std::make_shared<simulations::movers::MoversSetSweep>();
112   movers->add_mover(translate, n_atoms);
113   translate->max_move_range(max_jump); // --- set the maximum distance a single atom␣
↪can be moved by a single MC perturbation
114
115     // --- create a Wang-Landau Monte Carlo sampler
116     double initial_energy = lj_energy.energy(ar);
117   logs << utils::LogLevel::INFO << "Initial energy of the system (used to limit WL␣
↪sampling) : " << initial_energy << "\n";
118
119   simulations::sampling::WangLandauSampler sampler(movers, initial_energy, bin_from_
↪energy, initial_energy + 1);
120
121     // ---------- Create an observer which calls energy calculation and prints it on␣
↪the screen
122     std::shared_ptr<simulations::observers::ObserveEvaluators> obs = std::make_shared
↪<simulations::observers::ObserveEvaluators>("");
123     std::function<double(void)> recent_energy = [&lj_energy,&ar]() { return lj_energy.
↪energy(ar); };
124   obs->add_evaluator(
125       std::make_shared<simulations::evaluators::CallEvaluator<std::function
↪<double(void)>>>(recent_energy, "energy", 8));
126
127   std::shared_ptr<AbstractPdbFormatter> fmt = std::make_shared<SimplePdbFormatter>("␣
↪AR ", "AR ", "AR");
128   auto observe_trajectory = std::make_shared
↪<simulations::observers::cartesian::PdbObserver>(ar, fmt, "ar_tra.pdb");
129
130     observe_trajectory->trigger(std::make_shared
↪<simulations::observers::TriggerEveryN>(1));
131     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,
    ↪"movers.dat", 0.4);
135
136     sampler.outer_cycle_observer(observe_moves);
137     sampler.outer_cycle_observer(obs);
138     sampler.cycle_size(1000);
139     sampler.inner_cycles(n_inner_cycles);
140     sampler.outer_cycles(n_outer_cycles);
141     sampler.outer_cycle_observer(std::make_shared
    ↪<simulations::observers::ObserveWLSampling>(sampler, "wl.dat"));
142
143     sampler.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
149 }
```

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:

```
#include <iostream>

#include <simulations/observers/ObserveWLSampling.hh>

#include <simulations/movers/ising/SingleFlip2D.hh>
#include <simulations/movers/ising/WolffMove2D.hh>

#include <simulations/observers/ObserveMoversAcceptance.hh>
#include <simulations/observers/TriggerEveryN.hh>

#include <simulations/sampling/WangLandauSampler.hh>
#include <simulations/systems/ising/Ising2D.hh>

using namespace core::data::basic;

utils::Logger logs("ex_WL_Ising");

std::string program_info = R"(

The program runs a Wang-Landau Monte Carlo simulation of a simple 2D Ising system (spin glass)

)";


/** @brief Turns energy of a system into an energy bin index (integer)
 * @param energy - system's energy
 * @return integer assigned to a bin; may be negative
 */
inline int bfe(double energy) { return (int) energy; }

/** @brief The program runs a Wang-Landau Monte Carlo simulation of a simple 2D Ising system (spin glass).
 *
 * This example shows how to set up a WL simulation
 *
 * CATEGORIES: simulations/sampling/WangLandauSampler; simulations/systems/ising/Ising2D
```

(continues on next page)

(continued from previous page)

```
 * KEYWORDS:   observer; simulation
 */
int main(const int argc, const char *argv[]) {

  using namespace simulations::systems::ising;
  using namespace simulations::movers::ising;
  using namespace simulations::observers;

  core::index4 n_outer_cycles = 1;
  core::index4 n_inner_cycles = 10000;

  core::index2 system_size = 10;
  if (argc < 2) std::cerr << program_info;
  else {
    system_size = atoi(argv[1]);
    if (argc == 4) {
      n_inner_cycles = atoi(argv[2]);
      n_outer_cycles = atoi(argv[3]);
    }
  }

  core::calc::statistics::Random::get().seed(12345);  // --- seed the generator for repeatable results

  // ---------- Create the system to be sampled ----------
  std::shared_ptr<Ising2D<core::index1, core::index2>> system
      = std::make_shared<Ising2D<core::index1, core::index2>>(system_size, system_size);
  system->initialize();    // Populate system with random spins

  // ---------- Movers definition ----------
  simulations::movers::MoversSet_SP movers = std::make_shared<simulations::movers::MoversSetSweep>();
  movers->add_mover(std::make_shared<SingleFlip2D<core::index1, core::index2>>(*system), system->count_spins());
  movers->add_mover(std::make_shared<WolffMove2D<core::index1, core::index2>>(*system), system->count_spins() * 0.2);

  // ---------- Create the sampler ----------
  const double initial_energy = system->calculate();
  simulations::sampling::WangLandauSampler sampler(movers, initial_energy, bfe, 13);
  sampler.inner_cycles(n_inner_cycles);
  sampler.outer_cycles(n_outer_cycles);
  sampler.inner_cycle_observer(std::make_shared<ObserveWLSampling>(sampler, "wl.dat"));

  sampler.run();
}
```

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:

```
#include <iostream>

#include <core/data/io/XML.hh>
#include <core/data/io/XMLElement.hh>
#include <utils/LogManager.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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>
```

(continues on next page)

(continued from previous page)

```
     <description>Example text.</description>
     <options type="color">
          <item value="Purple" shade="bright" />
          <item>Green</item>
          <item>Orange</item>
     </options>
</product>
)";

/** @brief Simple for XML I/O utils.
 *
 * CATEGORIES: core/data/io/XML
 * KEYWORDS:   XML
 */
int main(int argc, char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  utils::LogManager::FINE();
  XML xxx;
  if (argc == 1) {
    std::istringstream input(xml_data);
    xxx.load_data(input);
  } else xxx.load_data(argv[1]);
  std::cout << xxx.document_root();

  return 0;
}
```

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:

```
#include <iostream>

#include <core/data/io/alignment_io.hh>
#include <core/BioShellEnvironment.hh>
#include <core/alignment/NWAligner.hh>
#include <core/alignment/on_alignment_computations.hh>
#include <core/alignment/PairwiseAlignment.hh>
#include <core/alignment/PairwiseSequenceAlignment.hh>
#include <core/alignment/scoring/SimilarityMatrix.hh>
#include <core/alignment/scoring/SimilarityMatrixScore.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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]
```

(continues on next page)

(continued from previous page)

```

EXAMPLE:
  ./ex_alignment_io example.edinb

)";

/** @brief Read alignment in Edinburgh format or calculate a new one from given sequences; write Edinburgh.
 *
 * CATEGORIES: core/data/io/alignment_io.hh
 * KEYWORDS:   sequence alignment
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::alignment;
  using namespace core::alignment::scoring;

  if (argc > 1) { // If there was an input alignment file given, read it!
    std::vector<PairwiseSequenceAlignment_SP> alignments;
    auto ali = core::data::io::read_edinburgh(argv[1], alignments);
    for (const PairwiseSequenceAlignment_SP & ali : alignments)
      core::data::io::write_edinburgh(*ali, std::cout, 80);
  } else { // otherwise, align the two sequences defined below
    // ---------- The two sequences that will be aligned
    std::string query =
"MKGWLFLVIAIVGEVIATSALKSSEGFTKLAPSAVVIIGYGIAFYFLSLVLKSIPVGVAYAVWSGLGVVIITAIAWLLHGQKLDAWGFVGMGLI
";
    std::string tmplt =
"MIYLYLLCAIFAEVVATSLLKSTEGFTRLWPTVGCLVGYGIAFALLALSISHGMQTDVAYALWSAIGTAAIVLVAVLFLGSPISVMKVVGVGLI
";
    // ---------- Gap penalties
    short int open = -10;
    short int extend = -2;
    // ---------- load BLOSUM matrix from bioshell's library; the directory must be defined as a shell variable
    const std::shared_ptr<SimilarityMatrix<short int>> b62_matrix = SimilarityMatrix<short int>::from_ncbi_file("alignments/BLOSUM62");
    const SimilarityMatrixScore<short int> b62_score(query, tmplt, *b62_matrix);
    NWAligner<short int, SimilarityMatrixScore<short int>> global(std::max(query.length(), tmplt.length()));
    // ---------- Compute and backtrace the alignment
    global.align(open, extend, b62_score);
    const PairwiseAlignment_SP ali = global.backtrace();
    // ---------- Convert the abstract alignment to a pairwise sequence alignment object
   core::data::sequence::Sequence query_seq("Q7B1Y7_SALEN",query);
   core::data::sequence::Sequence tmplt_seq("MMR_MYCTU",tmplt);
   core::data::io::write_edinburgh(query_seq, *ali, tmplt_seq, std::cout, 80);
  }
}
```

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:

```
#include <iostream>

#include <core/data/basic/Vec3.hh>
#include <core/calc/numeric/basic_algebra.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

Unit test that calculates eigenvalues and eigenvectors for a 3x3 matrix

USAGE:
./ex_basic_algebra

)";

/** @brief ex_basic_algebra illustrates how to calculate eigenvalues and eigenvectors for a 3x3 matrix
 *
 * CATEGORIES: core/calc/numeric/basic_algebra.hh
 * KEYWORDS: numerical methods
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::calc::numeric;
  using namespace core::data::basic; // --- for Array2D

  Array2D<double> m3x3(3,3,{2, -1, 0, -1, 2, -1, 0, -1, 2}); // --- input matrix to be solved
```

(continues on next page)

(continued from previous page)

```
30
31    std::cout << "\nOriginal matrix:\n";
32    m3x3.print("%8.3f", std::cout);
33
34    std::vector<double> eigenval;
35    core::calc::numeric::eigenvalues3(m3x3, eigenval);
36    std::cout << "\nEigenvalues: " << eigenval[0] << " " << eigenval[1] << " " <<␣
  ↪eigenval[2] << "\n\n";
37
38    auto eigenv = eigenvectors3(m3x3, eigenval);
39    std::cout << "\nEigenvectors:\n" << eigenv[0] << "\n" << eigenv[1] << "\n" <<␣
  ↪eigenv[2] << "\n\n";
40  }
```

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:

```
#include <iostream>
#include <iomanip>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <simulations/representations/cabs/cabs_utils.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

```

(continues on next page)

(continued from previous page)

```
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

)";

using namespace core::data::structural;
using namespace core::data::io;

/** @brief Reads an all-atom structure from a PDB file and produces a structure in CABS representation.
 *
 * CATEGORIES: simulations::representations::cabs::cabs_utils
 * KEYWORDS:   PDB input; CABS; representation
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  // --- Read the input PDB and create a structure object
  core::data::io::Pdb reader(argv[1], is_not_alternative, only_ss_from_header, true);
  core::data::structural::Structure_SP strctr = reader.create_structure(0);

  // --- Check whether loaded structure is in the CABS representation
  if (simulations::representations::is_cabs_model(*strctr))
    std::cerr<<"Loaded structure of "<<argv[1]<<" has CABS representation! Load all-atom model.\n";
  else if (simulations::representations::is_cabsbb_model(*strctr)) {
    std::cerr<<"Loaded structure of "<<argv[1]<<" has CABSBB representation! Load fullatom model.\n";
  }
  else {
  // --- Convert the Structure into CABS representation and write the result in the PDB format
    Structure_SP structure_sp = simulations::representations::cabs_representation(*strctr);
    for (auto atom_sp = structure_sp->first_atom(); atom_sp != structure_sp->last_atom(); ++atom_sp)
      std::cout << (*atom_sp)->to_pdb_line() << "\n";

    // --- Here we generate CONNECT lines so the PDB file displays nicely in PyMOL
    auto prev_ca_sp = *structure_sp->first_atom(); // --- the very first CA in the structure
```

(continues on next page)

(continued from previous page)

```
    for (auto res_sp_it = (++(structure_sp->first_residue())); res_sp_it != structure_
↪sp->last_residue(); ++res_sp_it) {
      auto ca = *((*res_sp_it)->cbegin()); // --- iterator to the CA of this residue
      core::data::io::Conect cn(prev_ca_sp->id(), ca->id());
      std::cout << cn.to_pdb_line();
      if ((*res_sp_it)->count_atoms() < 2) { // --- it has also CB
        auto cb = *((*res_sp_it)->cbegin() + 2); // --- CB is always the third one
        core::data::io::Conect cn(ca->id(), cb->id());
        std::cout << cn.to_pdb_line();
        if ((*res_sp_it)->count_atoms() < 2) { // --- it has also CB
          auto sc = *((*res_sp_it)->cbegin() + 3); // --- SC is always the fourth one
          core::data::io::Conect cn(cb->id(), sc->id());
          std::cout << cn.to_pdb_line();
        }
      }
      prev_ca_sp = ca;
    }
  }
}
```

ex_cabs_rotamers

Keywords:

- no_keywords

Categories:

- no_categories

Input files:

- 2gb1.pdb

Output files:

- stdout.out

Program source:

```
#include <iostream>

#include <core/chemical/ChiAnglesDefinition.hh>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/data/structural/selectors/SelectChainBreaks.hh>
#include <core/data/structural/selectors/SelectClashingResidues.hh>
#include <core/data/structural/selectors/SelectContiguousResidues.hh>

#include <core/calc/structural/angles.hh>
#include <core/calc/structural/protein_angles.hh>
#include <core/calc/structural/transformations/CartesianToSpherical.hh>
#include <core/calc/structural/transformations/transformation_utils.hh>
#include <core/calc/structural/transformations/Rototranslation.hh>

#include <utils/io_utils.hh>
#include <utils/LogManager.hh>

utils::Logger logger("ex_cabs_rotamers");

int main(const int argc, const char* argv[]) {

  using namespace core::chemical;
  using namespace core::data::structural;
  using namespace core::calc::structural;
  using namespace core::calc::structural::transformations;

  utils::LogManager::get().set_level("FINE");

  if(argc==1) {
    return 0;
  }
```

(continues on next page)

(continued from previous page)

```
  core::data::io::Pdb reader(argv[1],
      core::data::io::all_true(core::data::io::is_not_alternative,core::data::io::is_
↪not_hydrogen),
      core::data::io::keep_all, false);  // Create a PDB reader for a given file
  core::data::structural::Structure_SP str = reader.create_structure(0); // Create a␣
↪structure object from the first model

  // ---------- Selector we use to be sure that the residue is correct
  selectors::IsAA is_aa;
  selectors::IsBBCB atom_is_bb_cb;
  selectors::ResidueHasBBCB has_bb_cb;
  selectors::ResidueHasAllHeavyAtoms all_atoms;
  selectors::SelectChainBreaks breaks;
  selectors::SelectClashingResidues clash_test(str,2.3);
  selectors::SelectContiguousResidues check_resids;

  std::cout << "#   r     alpha   theta   alpha   theta      x      y      z  ss res_
↪id aa chain prot rotamer chi angles\n";
  CartesianToSpherical acs;
  std::vector<std::string> lcs_atom_names = {" N  ", " CA ", " C  "};
  auto ires = str->first_residue(); ++ires;
  auto next_res = str->first_residue(); ++(++next_res);
  for (; next_res != str->last_residue(); ++ires,++next_res) { // iterate over all␣
↪residues staring from the second one

    if (!is_aa(**ires)) continue;
    if (((**ires).residue_type() == Monomer::GLY) || ((**ires).residue_type() ==␣
↪Monomer::ALA)) continue;

    if(!has_bb_cb(**ires)) {
      logger << utils::LogLevel::WARNING << "Residue "<<(**ires) << " has incomplete␣
↪backbone or lacks its CB atom\n";
      continue;
    }

    if(!all_atoms(**ires)) {
      logger << utils::LogLevel::WARNING << "Residue side chain "<<(**ires) << " is␣
↪incomplete\n";
      continue;
    };

    if(!check_resids(**ires)) {
      logger << utils::LogLevel::WARNING << "Residue "<<(**ires) << " is broken\n";
      continue;
    };

    if(breaks(**ires)) {
      logger << utils::LogLevel::WARNING << "Residue "<<(**ires) << " is at chain␣
↪break\n";
      continue;
    };

    if(clash_test(**ires)) {
      logger << utils::LogLevel::WARNING << "Residue "<<(**ires) << " is in a steric␣
↪clash\n";
      continue;
    };
```

(continues on next page)

(continued from previous page)

```
82
83      PdbAtom_SP CA = (*ires)->find_atom_safe(" CA ");
84      PdbAtom_SP N = (*ires)->find_atom_safe(" N  ");
85      PdbAtom_SP C = (*ires)->find_atom_safe(" C  ");
86      PdbAtom_SP CB = (*ires)->find_atom_safe(" CB ");
87      double t = core::calc::structural::evaluate_dihedral_angle(*N, *C, *CB, *CA) *␣
↪180.0 / 3.14159;
88      if ((t < -50) || (t > -20)) {
89        logger << utils::LogLevel::WARNING << "Residue " << (**ires) << " has incorrect␣
↪geometry at CA. Dihedral angle: " << t << "\n";
90        continue;
91      };
92
93      core::data::basic::Vec3 cm;
94      double n = 0;
95      std::for_each((*ires)->cbegin(),(*ires)->cend(),[&](const PdbAtom_SP a){ if(!atom_
↪is_bb_cb(*a)) { cm+=(*a); ++n;} });
96
97      if (n == 0) continue;
98
99      cm /= n;
100     Rototranslation_SP lcs = local_coordinates_three_atoms(**ires, lcs_atom_names);
101     Vec3 sph;
102     lcs->apply(cm);
103     acs.apply(cm, sph);
104     std::cout << utils::string_format("%7.4f %7.2f %7.2f %7.4f %7.4f   %6.3f %6.3f %6.
↪3f ",
105       sph.x, to_degrees(sph.y), to_degrees(sph.z), sph.y, sph.z, cm.x, cm.y, cm.z);
106
107     std::cout << utils::string_format("%c %6d %3s %s %s %4s",
108       (*ires)->ss(), (*ires)->id(), (*ires)->residue_type().code3.c_str(), (*ires)->
↪owner()->id().c_str(),
109       utils::basename(str->code()).c_str(),
110       core::calc::structural::define_rotamer(**ires).c_str());
111     for (unsigned short i = 1; i <= core::chemical::ChiAnglesDefinition::count_chi_
↪angles((*ires)->residue_type()); ++i)
112       std::cout << utils::string_format(" %6.1f", core::calc::structural::evaluate_
↪chi(**ires, i) * 180.0 / 3.1415);
113     std::cout << "\n";
114   }
115 }
```

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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <utils/exit.hh>

#include <simulations/representations/cabs/cabs_utils.hh>

using namespace core::data::structural;
using namespace core::data::io;
```

(continues on next page)

(continued from previous page)

```

/** @brief Reads an all-atom structure from a PDB file and produces a structure in␣
↪CABS-BB representation.
 */
std::string program_info = R"(

Converts all-atom protein structure to CABS-bb representation
USAGE:
    ex_cabsbb_representation input.pdb
USAGE:
    ex_cabsbb_representation 2gb1.pdb

)";

/** @brief Converts all-atom protein structure to CABS-bb representation
 *
 *
 * CATEGORIES: simulations/representations/cabs/cabs_utils;
 * KEYWORDS:   PDB input; CABS-bb
 */int main(const int argc, const char* argv[]) {

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

  // --- Read the input PDB and create a structure object
  core::data::io::Pdb reader(argv[1], all_true(is_not_hydrogen,is_not_water,is_not_␣
↪alternative), keep_all, true);
  core::data::structural::Structure_SP strctr = reader.create_structure(0);

  // --- Check whether loaded structure is in the CABSBB representation
  if (simulations::representations::is_cabsbb_model(*strctr))
    std::cerr << "Loaded structure of " << argv[1] << " has CABSBB representation!␣
↪Load fullatom model.\n";
  else if (simulations::representations::is_cabs_model(*strctr))
    std::cerr << "Loaded structure of " << argv[1] << " has CABS representation! Load␣
↪fullatom model.\n";

  else {
    // --- Convert the Structure into CABSBB representation and write the result in␣
↪the PDB format
    core::data::structural::Structure_SP structure_sp =␣
↪simulations::representations::cabsbb_representation(*strctr);
    for (auto atom_sp = structure_sp->first_atom(); atom_sp != structure_sp->last_
↪atom(); ++atom_sp)
      std::cout << (*atom_sp)->to_pdb_line() << "\n";

    // --- Here we generate CONNECT lines so the PDB file displays nicely in PyMOL
    for (auto it = structure_sp->first_const_residue(); it != structure_sp->last_
↪const_residue(); ++it) {
      if ((*it)->count_atoms() == 6) { // --- if this CABSBB residue has 6 atoms, it␣
↪must have the SC atom
        auto cb = *((*it)->cbegin() + 4); // --- CB is always the fifth one
        auto sc = *((*it)->cbegin() + 5); // --- SC is always the sixth one
        core::data::io::Conect cn(cb->id(),sc->id());
        std::cout << cn.to_pdb_line();
      }
    }
```

(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:

```
#include <iostream>

#include <core/data/basic/Array2D.hh>
#include <core/calc/statistics/simple_statistics.hh>

std::string program_info = R"(

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
```

(continues on next page)

(continued from previous page)

```
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.

)";

/** @brief Chi-square test for independence.
 *
 * CATEGORIES: core::calc::statistics::chi2_independence_test;
 * KEYWORDS:   statistics; data table
 */
int main(const int argc, const char* argv[]) {

  using core::data::basic::Array2D;

  if(argc==3) { // --- calculate chi-square test for given chi-square statistics value and the number of DOFs
    int k = utils::from_string<int>(argv[1]);
    double crit = utils::from_string<double>(argv[2]);
    std::cout << "# DOFs:       " << k << "\n";
    std::cout << "# chi2 value: " << crit << "\n";
    std::cout << "# p-value:    " << core::calc::numeric::chi_square_pvalue(k,crit) << "\n";
    return 0;
  }

  if(argc==2) { // --- calculate chi-square test for given data (test the independence of the two first rows)
    Array2D<core::index4> m = Array2D<core::index4>::from_file(argv[1]);
    int k = (m.count_rows() - 1) * (m.count_columns() - 1);
    double crit = core::calc::statistics::chi2_independence_test(m);
    std::cout << "# DOFs:       " << k << "\n";
    std::cout << "# chi2 value: " << crit << "\n";
    std::cout << "# p-value:    " << core::calc::numeric::chi_square_pvalue(k, crit) << "\n";

    return 0;
  }

  std::cerr << program_info;

  std::vector<core::index4> data = {71,154,398,4992,2808,2737};
  Array2D<core::index4> m(2,3, data);

  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) <<
     ↪"\n";
66 }
```

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:

```
#include <vector>
#include <iostream>
#include <iterator>

#include <core/algorithms/basic_algorithms.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Unit test which shows how to find islands of consecutive elements in a container.

USAGE:
./ex_consecutive_find

)";

struct AreConsecutive {
  bool operator()(int i, int n) { return (n - i) == 1; }
};


struct SSranges {
  bool operator()(char ci, char cn) { return (ci==cn); }
};

/** @brief Shows how to find islands of consecutive elements in a container
 *
 * CATEGORIES: core/algorithms/basic_algorithms.hh
```

(continues on next page)

(continued from previous page)

```
 * KEYWORDS:   algorithms; data structures
 */
int main(const int argc, const char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  std::vector<int> v{-3, 1, 2, 4, 5, 7, 8, 9, 10, 12, 12, 13, 16, 18, 20, 21, 22, 23};
  std::vector<std::pair<int, int>> islands;

  int n_islands = core::algorithms::consecutive_find(v.begin(), v.end(),␣
↪AreConsecutive(), islands);

  for (const auto &is : islands) {
    for (int i = is.first; i <= is.second; ++i)
      std::cout << v[i] << " " ;
    std::cout << "\n";
  }

  std::string ss("CEEEEEECCCCCCEEEEEECCHHHHHHHHHHHHHHHHCCCCCEEEEECCCCEEEEEC");
  std::vector<char> v_ss(ss.begin(), ss.end());
  islands.clear();
  n_islands = core::algorithms::consecutive_find(v_ss.begin(), v_ss.end(), SSranges(),
↪ islands);
  for (const auto &is : islands)
    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:

```
#include <iostream>

#include <core/data/io/clustalw_io.hh>
#include <core/data/sequence/sequence_utils.hh>
#include <utils/exit.hh>
#include <utils/io_utils.hh>

std::string program_info = R"(

Reads a Multiple Sequence Alignment (MSA) in ClustalW format and counts residues by its type.

EXAMPLE:
    ./ex_count_residues_by_type cyped.CYP109.aln

)";

/** @brief Reads a MSA in ClustalW format  and prints by-residue counts
 *
 * CATEGORIES: core::data::sequence::sequence_utils
 * KEYWORDS:   clustal input; MSA
 */
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␣
↪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:

```
#include <iostream>
#include <iomanip>
#include <core/data/io/Pdb.hh>
#include <core/calc/structural/protein_angles.hh>
#include <core/chemical/ChiAnglesDefinition.hh>
#include <utils/exit.hh>
#include <core/data/structural/selectors/structure_selectors.hh>

std::string program_info = R"(

Prints rotamer type (M-P-T code) for each amino acid residue in the input PDB␣
↪structure
USAGE:
    ex_define_rotamer input.pdb
```

(continues on next page)

(continued from previous page)

```

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

)";

/** @brief Prints rotamer type (M-P-T code) for each amino acid residue in the input PDB structure
 *
 * CATEGORIES: core::chemical::ChiAnglesDefinition; core::data::structural::ResidueHasAllHeavyAtoms
 * KEYWORDS:   PDB input; structural properties; rotamers; STL
 */
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;
  using namespace core::data::structural;

  Pdb reader(argv[1]); // file name (PDB format, may be gzip-ped)
  Structure_SP strctr = reader.create_structure(0);
  selectors::ResidueHasAllHeavyAtoms has_full_sc;
  selectors::IsAA is_aa;
  // Iterate over all residues
  for (auto ires = strctr->first_residue(); ires != strctr->last_residue(); ++ires) {
    core::data::structural::Residue &res_sp = (**ires);
    if (!is_aa(res_sp)) continue;
    std::cout << std::setw(4) << res_sp.id() << " " << res_sp.residue_type().code3 << " "
              << core::chemical::ChiAnglesDefinition::count_chi_angles(res_sp.residue_type());
    if (has_full_sc(res_sp)) std::cout << std::setw(5) << core::calc::structural::define_rotamer(res_sp);
    else std::cout << " incomplete";
    std::cout << "\n";
  }
}
```

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:

```
#include <math.h>

#include <iostream>
#include <random>
#include <vector>

#include <core/calc/statistics/NormalDistribution.hh>
#include <core/calc/statistics/BivariateNormal.hh>
#include <core/calc/statistics/Combined_1D_2D_Normal.hh>
#include <core/calc/statistics/TrivariateNormal.hh>
#include <core/calc/statistics/expectation_maximization.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

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."
```

(continues on next page)

(continued from previous page)

```
Journal of the Royal Statistical Society: Series B 39 (1977): 1-22. doi: 10.1111/j.2517-6161.1977.tb01600.x

)";

/** @brief Example showing how to use expectation-maximization method
 *
 * CATEGORIES: core::calc::statistics::NormalDistribution;core::calc::statistics::BivariateNormal;core/calc/statistics/expectation_maximization.hh
 * KEYWORDS: estimation; expectation-maximization
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::calc::statistics;

  double rd = 9876543;
  std::mt19937 gen(rd);
  core::index4 N = 10000; //--- the number of random points to use in tests

  // ---------- a few distributions to play with
  double ave1 = 2.0, ave2 = 4.0, ave3 = 6.0;
  double std1 = 0.2, std2 = 0.3, std3 = 0.5;
  std::normal_distribution<> m(ave1, std1);
  std::normal_distribution<> p(ave2, std2);
  std::normal_distribution<> t(ave3, std3);

  // ---------- First let's solve a 1D problem : mixture of 3 Gaussians
  std::vector<std::vector<double> > random_points;
  std::vector<double> r1(1), r2(1), r3(1);
  for (core::index4 i = 0; i < N; ++i) {
    r1[0] = (m(gen));
    r2[0] = (p(gen));
    r3[0] = (t(gen));
    random_points.push_back(r1);
    random_points.push_back(r2);
    random_points.push_back(r3);
  }

  std::vector<core::calc::statistics::NormalDistribution> distributions_1D;
  std::vector<core::index1> index_1D; // --- Distribution assignment computed by EM will be stored here
  core::calc::statistics::NormalDistribution d1(ave1, std1), d2(ave2, std2), d3(ave3, std1);
  distributions_1D.push_back(d1);
  distributions_1D.push_back(d2);
  distributions_1D.push_back(d3);

  std::cout << "1D distributions: starting params\n";                //print parameters_ of all distributions
  for (const auto & d : distributions_1D) std::cout << d << "\n";
  double score = expectation_maximization(random_points, distributions_1D, index_1D, true);
  std::cout << "1D distributions: resulting params\n";                //print parameters_ of all distributions
```

(continues on next page)

(continued from previous page)

```
  for (const auto & d : distributions_1D) std::cout << d << "\n";
  std::cout << "\n";

  // ---------- now a mixture of 2 Gaussians in 2D
  std::vector<core::calc::statistics::BivariateNormal> distributions_2D;
  std::vector<core::index1> index_2D;
  core::calc::statistics::BivariateNormal d4(ave2, ave3, std2, std3, 0.1), d5(ave3,
↪ave2, std3, std2, 0.1);
  distributions_2D.push_back(d4);
  distributions_2D.push_back(d5);
  std::vector<std::vector<double> > data_2D;
  std::vector<double> rr1(2), rr2(2);
  for (core::index4 i = 0; i < N; ++i) {
    rr1[0] = (p(gen));
    rr1[1] = (t(gen));
    rr2[0] = (t(gen));
    rr2[1] = (p(gen));
    data_2D.push_back(rr1);
    data_2D.push_back(rr2);
  }

  std::cout << "2D distributions: starting params\n";                //print
↪parameters_ of all distributions
  for (const auto & d : distributions_2D) std::cout << d << "\n";
  expectation_maximization(data_2D, distributions_2D, index_2D, true);
  std::cout << "2D distributions: resulting params\n";                //print
↪parameters_ of all distributions
  for (const auto & d : distributions_2D) std::cout << d << "\n";
}
```

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:

```
#include <iostream>
#include <iomanip>
#include <core/data/io/Pdb.hh>
#include <core/chemical/Molecule.hh>
#include <core/chemical/molecule_utils.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief A simple example shows how to select a chemical group of a molecule using␣
↪find_side_group() method.
```

(continues on next page)

(continued from previous page)

```
 *
 * This example prints atoms for each side chain in a protein
 * CATEGORIES: core/chemical/find_side_group;
 * KEYWORDS:   data_structures;graphs ; residue side chains
 */
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::structural;
  core::data::io::Pdb reader(argv[1],core::data::io::is_not_alternative); // file name (PDB format, may be gzip-ped)
  Structure_SP strctr = reader.create_structure(0); // create a Structure object from the first deposit found in the input file

  // --- Here we create a molecule object; 0.1 is the tolerance for bond lengths (used to detect bonds)
  auto molecule_sp = core::chemical::create_molecule(strctr->first_atom(),strctr->last_atom(),0.1);

  // --- Iterate over all residues in the structure
  for(auto res_it = strctr->first_residue();res_it!=strctr->last_residue();++res_it) {
    auto ca = (*res_it)->find_atom(" CA "); // alpha carbon is the preceding atom
    auto cb = (*res_it)->find_atom(" CB "); // beta carbon is the atom where a side chain is attached
    if((ca== nullptr)||(cb== nullptr)) continue;
    std::vector<PdbAtom_SP> sc;
    core::chemical::find_side_group<PdbAtom_SP>(ca,cb,*molecule_sp,sc);
    std::cout << utils::string_format("%4d %s :",(*res_it)->id(),(*res_it)->residue_type().code3.c_str());
    for(const PdbAtom_SP & a : sc)
      std::cout << " " << a->atom_name();
    std::cout << "\n";
  }

}
```

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:

```
#include <iostream>

#include <core/data/basic/Array2D.hh>
#include <core/calc/statistics/simple_statistics.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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:
```

(continues on next page)

(continued from previous page)

```
    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.

)";

/** @brief Calculates Goodman and Kruskal's gamma parameters
 *
 * CATEGORIES: core::calc::statistics::goodman_kruskal_rank_correlation;
 * KEYWORDS:   statistics; data table
 */
int main(const int argc, const char* argv[]) {

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

  using core::data::basic::Array2D;

  // --- Read an input file - data table format
  Array2D<core::index4> m = Array2D<core::index4>::from_file(argv[1]);
  std::cout << core::calc::statistics::goodman_kruskal_rank_correlation(m)<<"\n";
}
```

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:

```
#include <vector>
#include <iostream>
#include <random>

#include <core/calc/clustering/greedy_clustering.hh>

/// A distance operator calculates the distance between two points indexed by <code>i
↪</code> and <code>j</code>
struct PointDistance {
  std::vector<double> & points;

  /// Constructor just copies the reference of a data vector
  PointDistance(std::vector<double> & pts) : points(pts) {}
  /// Call-operator computes the distance
  double operator()(const size_t i,const size_t j) const { return fabs(points[i]-
↪points[j]); }
};

/** @brief Example showing how to use greedy clustering method.
 *
 * CATEGORIES: core::calc::clustering::greedy_clustering()
 * KEYWORDS: clustering
 */
int main(const int argc, const char* argv[]) {

  // --- Prepare random number generators
  std::mt19937 gen(1234567);
  std::normal_distribution<> d1(10.5, 2.0);
  std::normal_distribution<> d2(-0.5, 2.0);
  std::vector<double> data;

  // --- Generate 20 random values
  for (unsigned short i = 0; i < 10; ++i) {
    data.push_back(d1(gen));
    data.push_back(d2(gen));
```

(continues on next page)

(continued from previous page)

```
  }

  std::vector<size_t> clusters; // --- Clusters will be stored here
  std::vector<size_t> cluster_members; // --- vector for members assigned to clusters
  PointDistance distance(data); // --- instance of the distance operator
  core::calc::clustering::greedy_clustering(data,distance,5.0,clusters,cluster_
↪members);

  // --- Show results
  std::cout << "n_clusters: " << clusters.size() << "\n";
  std::cout << "cluster assignment: ";
  for (const unsigned short i : cluster_members) std::cout << i << " ";
  std::cout << "\n";
}
```

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:

```
#include <iostream>

#include <core/data/io/hssp_io.hh>
#include <core/data/io/fasta_io.hh>
#include <utils/exit.hh>
#include <utils/options/OptionParser.hh>

std::string program_info = R"(

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:
```

(continues on next page)

(continued from previous page)

```
    ./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
)";

/** @brief Reads an MSA in HSSP format and writes a FASTA file.
 *
 *
 * USAGE:
 *     ex_hssp_to_fasta 1crn.pdb
 *
 * CATEGORIES: core::data::io::hssp_io;
 * KEYWORDS:   sequence alignment; FASTA; HSSP
 * GROUP:      File processing; Format conversion
 */
int main(const int argc, const char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::data::io;
  std::vector<std::shared_ptr<core::data::sequence::Sequence>>  sink;
  core::data::io::read_hssp_file(argv[1], sink);
  for(const auto & seq:sink) {
    std::cout << create_fasta_string(seq->header(), seq->sequence)<<"\n";
  }
}
```

ex_intersect_sorted

Unit test shows how to 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:

```
#include <vector>
#include <iostream>
#include <iterator>
#include <core/algorithms/basic_algorithms.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Unit test shows how to how to find an intersection of two sorted vectors of data

USAGE:
./ex_intersect_sorted

)";


/** @brief Shows how to find an intersection of two sorted vectors of data
 *
 * CATEGORIES: core/algorithms/basic_algorithms.hh
 * KEYWORDS:   algorithms; data structures
 */
int main(const int argc, const char* argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  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>range1</code> and <code>range2</code> are already sorted!
31   core::algorithms::intersect_sorted(range1.begin(), range1.end(), range2.begin(),␣
     ↪range2.end(), repeated);
32
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   std::cout << "\n";
36 }
```

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

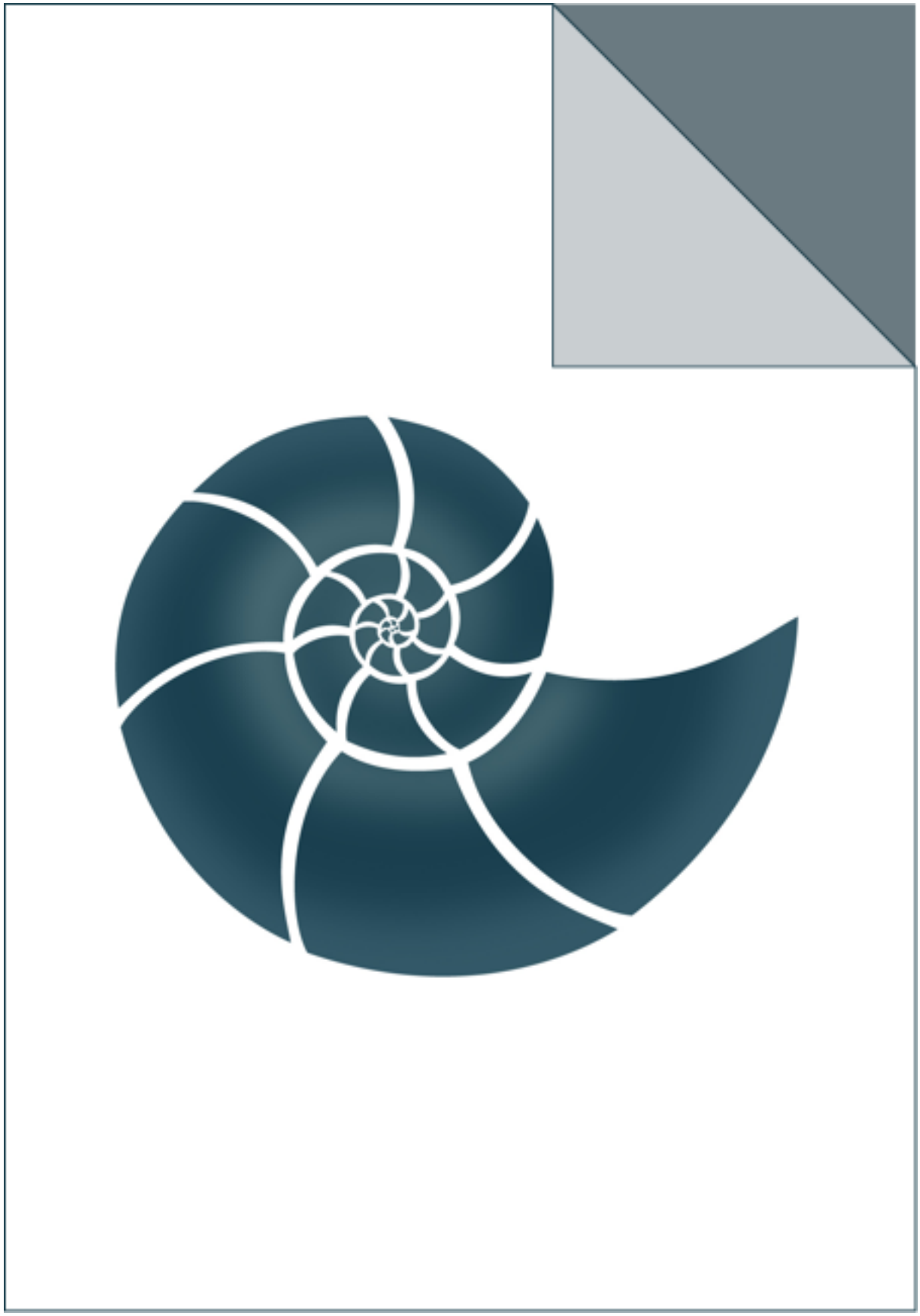
Program source:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/calc/structural/transformations/Rototranslation.hh>
#include <core/calc/structural/transformations/transformation_utils.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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.

```

(continues on next page)

(continued from previous page)

```
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

)";

/** @brief Reads a PDB file and prints local coordinates for side chain atoms
 *
 * CATEGORIES: core::calc::structural::transformations::local_BBQ_coordinates
 * KEYWORDS:   PDB input; local coordinates
 */
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::structural;
  using namespace core::calc::structural::transformations;

  core::data::io::Pdb reader(argv[1]);
  Structure_SP strctr = reader.create_structure(0);

  Rototranslation_SP rt = nullptr;
  for(auto a_chain : *strctr) {
    for (core::index2 i_residue = 1; i_residue < a_chain->count_residues() - 1; ++i_residue) {
      const Residue & the_residue = *(*a_chain)[i_residue];
      const Residue & prev_residue = *(*a_chain)[i_residue - 1];
      const Residue & next_residue = *(*a_chain)[i_residue + 1];
      try {
        rt = local_BBQ_coordinates(*prev_residue.find_atom_safe(" CA "),
          *the_residue.find_atom_safe(" CA "), *next_residue.find_atom_safe(" CA "));
      } catch (utils::exceptions::AtomNotFound ex) {
        i_residue+=2;
        continue;
      }
      Vec3 tmp_atom;
      for (auto i_atom : the_residue) {
        tmp_atom = *i_atom;
        rt->apply(*i_atom);
        std::cout << i_atom->to_pdb_line() << "\n";
        // --- Here we test if the inverse transformation really moves an atom to its original location
        rt->apply_inverse(*i_atom);
        if(tmp_atom.distance_to(*i_atom)>0.001)
          throw std::runtime_error("Incorrect position after transformation!");
      }
    }
  }
}
```



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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/calc/structural/transformations/Rototranslation.hh>
#include <core/calc/structural/transformations/transformation_utils.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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.

```

(continues on next page)

(continued from previous page)

```
USAGE:
    ./ex_local_coordinates_three_atoms input.pdb
EXAMPLE:
    ./ex_local_coordinates_three_atoms 5edw.pdb

)";

/** @brief Reads a PDB file and prints local coordinates for sidechain atoms
 *
 * CATEGORIES: core::calc::structural::transformations::local_coordinates_three_atoms
 * KEYWORDS:   PDB input; local coordinates; rototranslation
 */
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::structural;

  core::data::io::Pdb reader(argv[1]);
  core::data::structural::Structure_SP strctr = reader.create_structure(0);

  for (auto it_resid = strctr->first_residue(); it_resid != strctr->last_residue(); ++it_resid) {

    PdbAtom_SP n = (*it_resid)->find_atom(" N  ");
    PdbAtom_SP ca = (*it_resid)->find_atom(" CA ");
    PdbAtom_SP c = (*it_resid)->find_atom(" C  ");

    if ((n == nullptr)||(ca == nullptr)||(c == nullptr)) {
        std::cout << "Missing backbone atom\n";
        continue;
    }

    core::calc::structural::transformations::Rototranslation_SP rt =
      core::calc::structural::transformations::local_coordinates_three_atoms(*n,*ca,*c);
    Vec3 tmp_atom;
    for (auto i_atom : **it_resid) {
      tmp_atom = *i_atom;
      rt->apply(*i_atom);
      std::cout << i_atom->to_pdb_line() << "\n";
      // --- Here we test if the inverse transformation really moves an atom to its original location
      rt->apply_inverse(*i_atom);
      if(tmp_atom.distance_to(*i_atom)>0.001)
        throw std::runtime_error("Incorrect position after transformation!");
    }
  }
}
```

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:

```
#include <core/data/io/Cif.hh>
#include <core/data/io/mmCif.hh>

#include <utils/Logger.hh>
#include <utils/LogManager.hh>

#include <utils/exit.hh>

std::string program_info = R"(

Unit test which shows how to read CIF files.

USAGE:
    ex_Cif file.cif
EXAMPLE:
    ex_Cif AA3.cif

)";

/** @brief ex_Cif tests reading CIF files
 *
 * CATEGORIES: core/data/io/Cif
```

(continues on next page)

(continued from previous page)

```
 * KEYWORDS:   CIF input
 */
int main(const int argc, const char *argv[]) {

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

  utils::LogManager::INFO(); // --- INFO is the default logging level; set it to FINE to see more
  core::data::io::mmCif reader(argv[1]);

  std::cout<<reader.pdb_code()<<"\n";

  core::data::structural::Structure_SP strc = reader.create_structure(0);
    for (auto a=strc->first_atom();a!=strc->last_atom();++a)
        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:

```
#include <fstream>
#include <iostream>

#include <core/chemical/Monomer.hh>
#include <core/chemical/monomer_io.hh>

#include <utils/options/Option.hh>
#include <utils/options/OptionParser.hh>
#include <utils/options/input_options.hh>
#include <utils/options/output_options.hh>
#include <utils/exit.hh>

using namespace core::chemical;


std::string program_info = R"(

```

(continues on next page)

(continued from previous page)

```
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

)";

/** @brief The program converts a monomer structure from CIF format to internal formats used by BioShell.
 *
 * CATEGORIES: core/chemical/Monomer; utils/options/OptionParser; utils/options/Option
 * KEYWORDS:   monomers; option parsing
 */
int main(const int argc, const char* argv[]) {

  using namespace utils::options;
  utils::options::OptionParser & cmd = OptionParser::get();
  cmd.register_option(utils::options::help);
  cmd.register_option(verbose, mute);
  cmd.register_option(db_path);
  cmd.register_option(input_bin_monomers, input_cif_monomers,input_txt_monomers);
  cmd.register_option(output_file);
  cmd.program_info(program_info);

  if (!cmd.parse_cmdline(argc, argv)) return 1;

  if (input_cif_monomers.was_used()) read_monomers_cif(option_value<std::string>(input_cif_monomers));

  if (input_txt_monomers.was_used()) read_monomers_txt(option_value<std::string>(input_txt_monomers));

  if (input_bin_monomers.was_used()) read_monomers_binary(option_value<std::string>(input_bin_monomers));

  write_monomers_txt(option_value<std::string>(output_file,"monomers.txt"));
}
```

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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/Structure.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Reads a PDB file and writes protein sequence(s) in FASTA format.
```

(continues on next page)

(continued from previous page)

```
 *
 * This is a simplified version of ap_pdb_to_fasta_ss.cc application
 * USAGE:
 *     ex_pdb_to_fasta 5edw.pdb
 *
 * CATEGORIES: core::data::io::Pdb
 * KEYWORDS:   PDB input
 * GROUP:      File processing; Format conversion
 */
int main(const int argc, const char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace core::data::io; // Pdb and create_fasta_string lives there

  Pdb reader(argv[1], is_not_alternative, only_ss_from_header, true);
  core::data::structural::Structure_SP strctr = (reader.create_structure(0));

  // Iterate over all chains
  for (int ic = 0; ic < strctr->count_chains(); ++ic)
    std::cout << "> " << strctr->code() << (*strctr)[ic]->id() << "\n" // --- e.g. prints "> 2gb1 A"
      << (*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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/calc/structural/interactions/BackboneHBondCollector.hh>
#include <utils/exit.hh>

using namespace core::data::structural;
using namespace core::data::io;
using namespace core::data::basic;

std::string program_info = R"(
```

(continues on next page)

(continued from previous page)

```

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

)";

/** @brief Reconstructs peptide hydrogen atoms using two methods and compares the error between them.
 * CATEGORIES: core::calc::structural::peptide_hydrogen()
 * KEYWORDS:   PDB input; hydrogen reconstruction
 */
int main(const int argc, const char* argv[]) {
  if(argc < 2) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter
  core::data::io::Pdb reader(argv[1], all_true(is_not_alternative, is_not_water)); // --- Read in a PDB file
  core::data::structural::Structure_SP strctr = reader.create_structure(0); // --- create a Structure object from the first model

  double err = 0, n = 0;
  auto res_it = ++(strctr->first_residue()); // --- The residue being reconstructed
  auto prev_res_it = strctr->first_residue(); // --- preceding residue
  for (; res_it != strctr->last_residue(); ++res_it) {
    std::cerr << "# reconstructing:" << (**prev_res_it) << " and " << (**res_it) << "\n";
    if (((*prev_res_it)->residue_type().parent_id > 20) || ((*res_it)->residue_type().parent_id > 20)) {  // --- its not an amino acid
      ++prev_res_it;
      continue;
    }
    try {
      if((*prev_res_it)->owner() != (*res_it)->owner()) {
        std::cerr << "# Chain break between residues:" << (**prev_res_it) << " and " << (**res_it) << "\n";
        ++prev_res_it;
        continue;
      }
      // --- Rebuild the peptide hydrogen in a residue pointed by res_it iterator.
      // --- This method actually adds the newly created H atom to the residue
      core::calc::structural::interactions::peptide_hydrogen(*prev_res_it, *res_it);
      auto new_H = (*res_it)->find_atom_safe(" H  ");
      // --- Here we reconstruct amide H knowing the relevant atoms, but now according to the DSSP approach. Resulting H is not inserted
```

(continues on next page)

(continued from previous page)

```
      auto prev_O = (*prev_res_it)->find_atom_safe(" O  ");
      auto prev_C = (*prev_res_it)->find_atom_safe(" C  ");
      auto this_N = (*res_it)->find_atom_safe(" N  ");
      PdbAtom other_H;
      core::calc::structural::interactions::peptide_hydrogen_dssp(*prev_C, *prev_O,␣
↪*this_N, other_H);
      err += other_H.distance_to(*new_H);
      n++;
      for (const auto &atom : **res_it)
        std::cout << atom->to_pdb_line() << "\n"; //-- print all atoms in the current␣
↪residue (in PDB format)
      ++prev_res_it; // --- advance one of the iterators by one residue; the other␣
↪iterator is advanced by the loop
    } catch (utils::exceptions::AtomNotFound e) {
      std::cerr << e.what() << "\n";
      ++prev_res_it;
    }
  }
  std::cout << "# difference between two methods: " << err / n << "\n";
}
```

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:

```
#include <iostream>
#include <map>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>

#include <utils/exit.hh>

std::string program_info = R"(

ex_protein_peptide_interface finds  atomic contacts between a receptor and a peptide␣
↪found in an input PDB file.
```

(continues on next page)

(continued from previous page)

```
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.

)";

unsigned int MAX_PEPTIDE_LENGTH = 35;
unsigned int MIN_PROTEIN_LENGTH = 40;

/** @brief Finds contacts atomic contacts between a receptor and a peptide.
 *  *
 * CATEGORIES: core::data::structural::Structure
 * KEYWORDS: PDB input; contact map; peptide; STL
 */
int main(const int argc, const char* argv[]) {

  if (argc < 3) utils::exit_OK_with_message(program_info); // --- complain about missing program parameter

  using namespace core::data::io;
  Pdb reader(argv[1], all_true(is_not_water,is_not_alternative,is_not_hydrogen)); // --- file name (PDB format, may be gzip-ped)

  core::data::structural::Structure_SP strctr = reader.create_structure(0);
  core::data::structural::selectors::IsAA is_aa_tester;
  core::data::structural::Structure_SP sub_strctr = strctr;// = strctr->clone(is_aa_tester);

  double cutoff = utils::from_string<double>(argv[2]); // The second parameter is the contact distance (in Angstroms)

  for (auto protein_chain_sp: *strctr) { // --- protein_chain_sp is a shared pointer to a chain
    if (protein_chain_sp->size() < MIN_PROTEIN_LENGTH) continue;
    for (auto i_residue_sp: *protein_chain_sp) { // --- i_residue_sp is a shared pointer to a residue
      for (auto peptide_chain_sp: *strctr) {
        if (peptide_chain_sp->size() > MAX_PEPTIDE_LENGTH) continue;
        for (auto j_residue_sp: *peptide_chain_sp) {
          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() << "
↪" << (*j_residue_sp) << " "
63                        << (*j_residue_sp).owner()->id() << " " << d << "\n";
64          }
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:

```
#include <memory>
#include <iostream>
#include <random>

#include <core/algorithms/trees/kd_tree.hh>
#include <core/algorithms/trees/BinaryTreeNode.hh>
#include <core/algorithms/trees/algorithms.hh>
#include <core/data/io/DataTable.hh>
#include <utils/exit.hh>

std::string program_info = R"(

ex_ramachandran_kd_tree partitions observations from a Ramachandran map


```

(continues on next page)

(continued from previous page)

```
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

)";

using namespace core::algorithms::trees;

class Point: public std::pair<float, float> {
public:

  Point(float phi, float psi) : std::pair<float, float>(phi, psi) {}

  float operator[](const size_t k) const { if (k == 0) return first; else return second; }
};

/// Operation that computes the distance between points on Ramachandran map
struct PhiPsiDistance {

  /// This operation will be called at every tree node considered during a tree traversal
  float operator() (const Point & n1,const Point & n2) const {
    float d = (n1.first - n2.first);
    float d2 = d * d;
    d = (n1.second - n2.second);
    d2 += d * d;
    return sqrt(d2);
  }
};

/** @brief Tree traversal operation prints a given point along with its node level and the number of neighbors
 */
struct PrintPoint {

  PrintPoint(std::shared_ptr<BinaryTreeNode<KDTreeNode<Point>>> root, float width) : root_(root), w_(width / 2.0) {}

  /// this operator prints a visited node on the screen
  void operator()(std::shared_ptr<BinaryTreeNode<KDTreeNode<Point>>> node) {

    Point q_low(node->element.element[0] - w_, node->element.element[1] - w_);
    Point q_up(node->element.element[0] + w_, node->element.element[1] + w_);
```

(continues on next page)

(continued from previous page)

```
    std::vector<Point> hits;
    search_kd_tree(root_,  q_low, q_up, 2, hits);

    std::cout << utils::string_format("%7.2f %7.2f %3d %4d\n",
                  node->element.element.first, node->element.element.second, node->element.level, hits.size());
  }
private:
  std::shared_ptr<BinaryTreeNode<KDTreeNode<Point>>> root_;
  float w_;
};

/** @brief A simple example shows how to use BioShell kd-tree routines.
 *
 * The program reads a file with Phi, Psi observations and partitions them in a kd-tree.
 *
 * CATEGORIES: core::algorithms::trees::BinaryTreeNode; core::algorithms::trees::kd_tree.hh
 * KEYWORDS:   neighborhood detection; data structures; algorithms
 */
int main(const int argc, const char* argv[]) {

  if (argc < 2) utils::exit_OK_with_message(program_info); // --- complain about missing program parameters

  core::index2 n_level = (argc > 2) ? atoi(argv[2]) : 4;
  core::index2 width = (argc > 3) ? atof(argv[3]) : 5;
  // ---------- First we read a file with Phi, Psi observations
  core::data::io::DataTable dt;
  dt.load(argv[1]);
  std::vector<Point> points;            // container for the points
  for(const auto & row:dt)
    points.emplace_back(row.get<float>(0), row.get<float>(1));

  // ---------- Here the actual kd-tree is constructed
  std::shared_ptr<BinaryTreeNode<KDTreeNode<Point>>> root =
      create_kd_tree<Point, std::vector<Point>::iterator, CompareAsReferences<Point>>(points.begin(), points.end(), 2);


  std::vector<std::shared_ptr<BinaryTreeNode<KDTreeNode<Point>>>> node_group_representatives;
  collect_given_level(root, n_level, node_group_representatives, 0);
  core::index2 group_id = 0;
  for(const auto node:node_group_representatives) {
    depth_first_preorder(node, [group_id](std::shared_ptr<BinaryTreeNode<KDTreeNode<Point>>> node) {
      node->element.level = group_id; });
    ++group_id;
  }

  // ---------- Here we print each node
  PrintPoint pp(root, width);
  breadth_first_preorder(root, pp); // finally, each node is printed
}
```

ex_random_vector_on_sphere

Simple test shows that random_vector_on_sphere() really produces a unifirm distribution

Keywords:

- no_keywords

Categories:

- simulations/movers/random_vector_on_sphere

Output files:

- stdout.out

Program source:

```
#include <iostream>

#include <core/data/basic/Vec3.hh>

#include <simulations/movers/movers_utils.hh>

using namespace core::data::basic;
using namespace simulations;

/** @brief Simple test shows that random_vector_on_sphere() really produces a unifirm distribution
 *
 * CATEGORIES: simulations/movers/random_vector_on_sphere;
 */
int main(int argc, char *argv[]) {

  core::data::basic::Vec3 v;
  core::data::basic::Vec3 s;
  core::index4 N = 100000;
  for (size_t i = 0; i < N; ++i) {
    simulations::movers::random_vector_on_sphere(v);
    s += v;
    std::cout << v << "\n";
  }
  s /= N;
  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:

```
#include <iostream>

#include <utils/io_utils.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Simple test reads .properties file and prints these settings on the screen
 *
 * CATEGORIES: utils/read_properties_file
```

(continues on next page)

(continued from previous page)

```
 * KEYWORDS:   file utils;properties file
 */
int main(const int argc, const char* argv[]) {

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

  for (int i = 1; i < argc; ++i) {
    // In the single line below we read the properties file
    auto mapa = utils::read_properties_file(argv[i]);

    // Here we print the content of the map in the same format (i.e. .properties)
    for (auto it = mapa.cbegin(); it != mapa.cend(); ++it) {
      std::cout << it->first << " : ";
      for (auto it2 = it->second.cbegin(); it2 != it->second.cend(); ++it2)
        std::cout << (*it2) << " ";
      std::cout << "\n";
    }
  }
}
```

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:

```
#include <iostream>
#include <sstream>
#include <core/data/io/Pdb.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <core/protocols/selection_protocols.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Shows how to use selection protocols functions
 *
 * CATEGORIES: core::protocols::keep_selected_atoms()
```

(continues on next page)

(continued from previous page)

```
 * KEYWORDS:   PDB input; Selection protocols; structure selectors
 */
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::structural;
  using namespace core::data::structural::selectors;
  using namespace core::protocols;

  core::data::io::Pdb reader(argv[1]);

  { // --- section which tests selecting nucleotides
    Structure_SP strctr = reader.create_structure(0);

    std::shared_ptr<AtomSelector> select_nt = std::make_shared<IsNT>();

    keep_selected_atoms(*select_nt, *strctr);
    for (auto chain_sp : *strctr)
      std::cout << utils::string_format("\tchain %s has %3d residues satisfying the selector\n", chain_sp->id().c_str(),
        chain_sp->size());
  }
}
```

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:

```
#include <iostream>
#include <core/data/io/fasta_io.hh>
#include <core/data/io/seq_io.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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

)";

/** @brief Example reads SEQ file and prints the data stored there in FASTA format
```

(continues on next page)

(continued from previous page)

```
 *
 * CATEGORIES: core/data/io/read_seq
 * KEYWORDS:   sequence; FASTA output; secondary structure; Format conversion
 */
int main(const int argc, const char* argv[]) {

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

  core::data::sequence::SecondaryStructure_SP ss = core::data::io::read_seq(argv[1],"
↪");
  ss->header(argv[1]);
  std::cout << core::data::io::create_fasta_string(*ss, 80)<<"\n";
  std::cout << core::data::io::create_fasta_secondary_string(*ss, 80)<<"\n";
}
```

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:

```
#include <iostream>
#include <cmath>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/Structure.hh>
#include <core/data/structural/selectors/structure_selectors.hh>

#include <core/calc/structural/protein_angles.hh>
#include <core/calc/structural/transformations/Rototranslation.hh>
#include <utils/exit.hh>

using namespace core::data::structural;
using namespace core::data::io;
using namespace core::data::basic;

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␣
    ↪protein.
19
20  USAGE:
21      ex_set_dihedral res-id file.pdb phi psi omega
22  EXAMPLE:
23      ex_set_dihedral 2gb1.pdb 18 -80.4 90.4 180.0
24
25  where 2gb1.pdb is the protein structure to be modified, 18 is the residue ID and the␣
    ↪three following
26  real values are Phi, Psi and omega dihedrals (in the range [-180.0,180.0]). The␣
    ↪results is printed in PDB format
27  )";
28
29  /** @brief Sets a particular values for Phi, Psi and Omega angles at a certain␣
    ↪residue in a protein.
30   * *
31   * CATEGORIES: core/calc/structural/transformations/Rototranslation
32   * KEYWORDS:   PDB input; rototranslation; structural properties
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    // --- The first parameter of the program is the PDB file name
39    core::data::io::Pdb reader(argv[1], is_not_alternative, keep_all, false); // ---␣
    ↪Read in a PDB file
40    core::data::structural::Structure_SP strctr = reader.create_structure(0);
41    // --- create a Structure object from the first model and extract the first chain␣
    ↪(indexed as 'A') from it
42    core::data::structural::Chain & chain = *(strctr->get_chain('A'));
43    core::index2 res_idx = utils::from_string<core::index2>(argv[2]);
44    core::calc::structural::transformations::Rototranslation rt;
45
46    // --- Phi rotation; the new value of the Phi angle (in degrees) is the fourth␣
    ↪parameter of this program
47    if (argc > 3 && strlen(argv[3]) > 1) {
48      double phi = core::calc::structural::evaluate_phi(*chain[res_idx-1],*chain[res_
    ↪idx]);
49      std::cerr << "Phi angle before change: " << phi * 180.0 / M_PI << " degrees\n";
50      phi = utils::from_string<double>(argv[3]) * M_PI / 180.0 - phi;
51      PdbAtom & N = *chain[res_idx]->find_atom_safe(" N  ");
52      PdbAtom & CA = *chain[res_idx]->find_atom_safe(" CA ");
53      core::calc::structural::transformations::Rototranslation::around_axis(N, CA, phi,␣
    ↪CA, rt);
54      for (core::index2 ires = 0 ; ires < res_idx; ++ires) {
55        for (PdbAtom_SP ai : *chain[ires]) rt.apply(*ai);
56      }
57      if(chain[res_idx]->find_atom(" H  ")!=nullptr)
58        rt.apply(*chain[res_idx]->find_atom(" H  "));
59    }
60
61    // --- Psi rotation; the new value of the Psi angle (in degrees) is the fifth␣
    ↪parameter of this program
62    if (argc > 4 && strlen(argv[4]) > 1) {
```

(continues on next page)

(continued from previous page)

```
    double psi = core::calc::structural::evaluate_psi(*chain[res_idx],*chain[res_idx+1]);
    std::cerr << "Psi angle before change: " << psi * 180.0 / M_PI << " degrees\n";
    psi = utils::from_string<double>(argv[4]) * M_PI / 180.0 - psi;
    PdbAtom & C = *chain[res_idx]->find_atom_safe(" C  ");
    PdbAtom & CA = *chain[res_idx]->find_atom_safe(" CA ");
    core::calc::structural::transformations::Rototranslation::around_axis(CA, C, psi, C, rt);
    rt.apply(*chain[res_idx]->find_atom_safe(" O  "));
    for (core::index2 ires = res_idx + 1; ires < chain.count_residues(); ++ires) {
      for (PdbAtom_SP ai : *chain[ires]) rt.apply(*ai);
    }
  }

  // --- Omega rotation; the new value of the Omega angle (in degrees) is the fifth parameter of this program
  if (argc > 5 && strlen(argv[5]) > 1) {
    double omega = core::calc::structural::evaluate_omega(*chain[res_idx],*chain[res_idx+1]);
    std::cerr << "Omega angle before change: " << omega * 180.0 / M_PI << " degrees\n";
    omega = utils::from_string<double>(argv[5]) * M_PI / 180.0 - omega;
    PdbAtom & C = *chain[res_idx]->find_atom_safe(" C  ");
    PdbAtom & N = *chain[res_idx+1]->find_atom_safe(" N  ");
    core::calc::structural::transformations::Rototranslation::around_axis(C, N, omega, N, rt);
    for (core::index2 ires = res_idx + 1; ires < chain.count_residues(); ++ires) {
      for (PdbAtom_SP ai : *chain[ires]) rt.apply(*ai);
    }
  }

  std::for_each(chain.first_atom(),chain.last_atom(),[](PdbAtom_SP ai){std::cout << ai->to_pdb_line()<<"\n";});
}
```

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:

```
#include <memory>
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/Chain.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

A very basic example showing how to use shared pointers (from standard C++ 11 library) when programming in BioShell.

USAGE:
./ex_shared_pointers

)";

using namespace core::data::structural;

// --- An example function that takes a reference to an object as an argument
void show_chain_code(const Chain & c) { std::cout << c.id() << "\n"; }

/** @brief A very basic example showing how to use shared pointers (from standard C++ 11 library) when programming in BioShell.
 *
 * CATEGORIES: std::make_shared
 * KEYWORDS:   STL
 */
int main(const int argc, const char* argv[]) {
```

(continues on next page)

(continued from previous page)

```

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  // --- This is how we create a shared pointer to a Chain object.
  // --- The object is empty i.e. it doesn't contain any residues
  Chain_SP chain_sp = std::make_shared<Chain>("A");
  // --- This is the same as above, but here we create a Chain object
  Chain chain_object("A");

  // --- This method creates a chain of a given amino acid sequence and returns a␣
↪shared pointer to it
  Chain_SP longer_sp =  Chain::create_ca_chain("AGGACL","A");

  // --- call a method that takes a reference to an object
  show_chain_code(chain_object);
  // --- here we create a reference from a shared pointer
  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:

```
#include <math.h>

#include <iostream>

#include <core/calc/numeric/numerical_integration.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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,
)";

/// First of the two functions integrated in this example
struct Sin {

  double operator()(double x) { return sin(x); }
} sin_func;

/// Second of the two functions integrated in this example
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    std::cout << core::calc::numeric::simpson_integration(x_square_func, 0.0, 1.0,1000)
   ↪<< "\n";
45  }
```

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:

```
#include <iostream>

#include <core/data/io/fasta_io.hh>
#include <utils/string_utils.hh>
#include <utils/exit.hh>

std::string program_info = R"(

ex_split_fasta reads a FASTA file and writes every sequence from it in a separate file
EXAMPLE:
    ./ex_split_fasta 5edw.fasta

)";
```

(continues on next page)

(continued from previous page)

```

/** @brief Reads a file with sequences in FASTA format and writes each sequence to a separate FASTA file.
 *
 * CATEGORIES: core/data/io/fasta_io.hh;
 * KEYWORDS:   FASTA input; FASTA output; sequence; FASTA; pre-processing
 */
int main(const int argc, const char *argv[]) {


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

  using core::data::sequence::Sequence_SP; // --- Sequence_SP is just a std::shared_ptr to core::data::sequence::Sequence type
  using namespace core::data::io;          // --- for FASTA I/O

  // --- Create a container where the sequences will be stored
  std::vector<Sequence_SP> sequences;

  // --- Read a file with FASTA sequences
  core::data::io::read_fasta_file(argv[1], sequences);

  // --- Write them in separate FASTA files
  for (const Sequence_SP s : sequences) {
    std::string header = s->header();
    std::replace(header.begin(), header.end(), '|', ' '); // --- fix ncbi-style header in FASTA files
    auto words = utils::split(header, {' '}); // --- We take the very first word of the FASTA as a file name; hopefully it is sth meaningful, e.g. a gene name
    std::ofstream out(words[0] + ".fasta");
    out << "> " << s->header() << "\n" << s->sequence << "\n";
    out.close();
  }
}
```

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:

```
#include <iostream>
#include <iomanip>
#include <core/data/io/Pdb.hh>
#include <core/chemical/Molecule.hh>
#include <core/chemical/molecule_utils.hh>
#include <core/data/structural/PdbAtom.hh>

#include <utils/exit.hh>

std::string program_info = R"(

Example that shows how to iterate through structural components

```

(continues on next page)

(continued from previous page)

```
USAGE:
    ex_structure_iterators 1dt7.pdb

where 1dt7.pdb id an input file (PDB format)

)";

/** @brief Shows how to iterate through structural components (residues, atoms, etc)
 *
 * CATEGORIES: core/data/structural/Structure
 * KEYWORDS: PDB input; Structure; Chain; Residue; PdbAtom; STL
 */
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::structural;
  core::data::io::Pdb reader(argv[1],core::data::io::is_not_alternative); // file name (PDB format, may be gzip-ped)
  Structure_SP strctr = reader.create_structure(0);

  // ------- Directly iterate over atoms of a structure, jump over chains, residues, etc.
  // ------- atom_it is an iterator, which points to a shared pointer to an atom
  int n_atoms_1 = 0;
  for (auto atom_it = strctr->first_atom(); atom_it != strctr->last_atom(); ++atom_it) ++n_atoms_1;

  // ------- Iterate over chains, residues, atoms
  int n_atoms_2 = 0;
  int n_chains = 0, n_residues = 0;
  for(auto chain_sp: *strctr) { // --- chain_sp is already a shared pointer to a chain
    ++n_chains;
    for(auto residue_sp: *chain_sp) { // --- residue_sp is already a shared pointer to a residue
      ++n_residues;
      for(auto atom_sp: *residue_sp)  // --- atom_sp is already a shared pointer to an atom
        ++n_atoms_2;
    }
  }

  int n_residues_2 = 0;
  // ------- Iterate over residues of a structure, jump over chains
  // ------- iter_res_i is an iterator, which points to a shared pointer to a residue
  for (auto iter_res_i = strctr->first_residue(); iter_res_i != strctr->last_residue(); ++iter_res_i)
    ++n_residues_2;

  std::cout << "These three atom counts should be equal: " << n_atoms_1 << ", " << n_atoms_2 << " and "
            << strctr->count_atoms() << "\n";
  std::cout << "These three residue counts should be equal: " << n_residues << ", " << n_residues_2 << " and "
            << strctr->count_residues() << "\n";
  std::cout << "These two chain counts should be equal: " << n_chains << " and " << strctr->count_chains() << "\n";
```

(continues on next page)

(continued from previous page)

```
62 }
```

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:

```
#include <iostream>
#include <memory>

#include <core/algorithms/graph_algorithms.hh>
#include <core/chemical/Molecule.hh>
#include <core/chemical/molecule_utils.hh>
#include <core/calc/structural/angles.hh>
#include <core/data/structural/PdbAtom.hh>
#include <core/data/structural/selectors/structure_selectors.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(


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.

```

(continues on next page)

(continued from previous page)

```
USAGE:
    ./ex_structure_to_molecule input.pdb ligand-code

EXAMPLE:
    ./ex_structure_to_molecule 4rm4.pdb HEM

)";

/** @brief 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
 *
 * CATEGORIES: core::chemical::structure_to_molecule
 * KEYWORDS: molecule
 */
int main(const int argc, const char *argv[]) {

  if (argc <3) utils::exit_OK_with_message(program_info);

  using namespace core::chemical;
  using namespace core::data::structural;

  PdbMolecule_SP molecule;

  // --- Read structure that we use to build a molecule
  core::data::io::Pdb reader(argv[1]); // file name (PDB format, may be gzip-ped)
  core::data::structural::Structure_SP strctr = reader.create_structure(0);
  // --- Create molecule object
  molecule = structure_to_molecule(*strctr);

  // --- Find the ligand object(s) for a given 3-letter code
  selectors::SelectResidueByName ligand_by_name(argv[2]);
  std::vector<Residue_SP> ligand;
  strctr->find_residues(ligand_by_name, ligand);

  for (const auto &l:ligand) {      // --- iterate over ligands found
    std::cout << "Bonds between " << l->residue_type().code3 << " and the rest of the protein:\n";
    for (const auto &atom : *l) {   // --- iterate over atoms of a ligand, find all its bounded partners
      for (auto it = molecule->cbegin_atom(atom); it != molecule->cend_atom(atom); ++it) {
        if ((**it).owner() != l)    // --- if the two atoms belong to different residues - print the output
          std::cout << (*atom).atom_name() << " - " << (**it).atom_name() << " " << (**it).owner()->residue_type().code3
                    << " " << (**it).owner()->residue_id() << " " << (**it).owner()->owner()->id() << "\n";
      }
    }
  }
}
```

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:

```
#include <sstream>

#include <utils/io_utils.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

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 =
```

(continues on next page)

(continued from previous page)

```
    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
 )";

/** @brief Simple test to gzip and un-gzip  a string data
 *
 * CATEGORIES: utils/io_utils
 * KEYWORDS:   GZIP
 */
int main(int cnt, char* argv[]) {

  if ((cnt > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  std::string zipped,result;
  // --- here we compress ala_cif_data string with ZIP and store the result in␣
↪another string
  utils::zip_string(ala_cif_data,zipped);
  // --- here we un-zip it back and store the ouput in the string "result"
  utils::unzip_string(zipped,result);
  if (result == ala_cif_data) {
    std::cout << "GZIP OK :-)\n";
    std::cout << "compressed from " << ala_cif_data.size() << " to " << zipped.size()
↪<< " bytes\n";
  } else std::cout << "GZIP ERROR !!!\n";

  // --- Here we un-zip directly to a stream
  std::stringstream ss;
  utils::unzip_string(zipped,ss);

  if (ss.str() == ala_cif_data)
    std::cout << "GZIP OK :-)\n";
  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:

```
#include <vector>

#include <core/algorithms/basic_algorithms.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Unit test for uniquify() method which removes redundant objects from a container

USAGE:
./ex_uniquify

)";

/** @brief Tests uniquify() method which removes redundant objects from a container.
 *
 * CATEGORIES: core/algorithms/basic_algorithms.hh
 * KEYWORDS:   data structures; algorithms
 */
int main(int cnt, char* argv[]) {

  if ((cnt > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  std::vector<int> datum = std::vector<int> { 1, 8, 4, 5, 9, 4, 5 };
  // ---------- Below we define >is_equal< operator
  auto eq = [](std::vector<int>::iterator a, std::vector<int>::iterator b) -> bool {␣
↪return *a == *b; };
```

(continues on next page)

(continued from previous page)

```
  // ---------- Below we define >less_then< operator
  auto lt = [](std::vector<int>::iterator a, std::vector<int>::iterator b) -> bool {return *a < *b; };

  // ---------- Below we apply uniquify() operation on a range of integers
  datum.erase(core::algorithms::uniquify(datum.begin(), datum.end(), eq, lt), datum.end());
  for (int c:datum) std::cout << c << " ";
  std::cout << "\n";

  // ---------- Below we apply uniquify() operation on a range of characters
  std::vector<char> chars = std::vector<char> {'a', 'g', 'd', 'r', 'a', 'd'};
  chars.erase(core::algorithms::uniquify(chars.begin(), chars.end()), chars.end());
  for (char c:chars) std::cout << c << " ";
  std::cout << "\n";
}
```

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:

```
#include <iostream>
#include <ui/www/web_client.hh>
#include <utils/exit.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Simple test for web_client methods  downloads 2GB1 protein from rcsb.org website
USAGE:
    ex_web_client

)";

/** @brief Simple test for web_client methods downloads 2GB1 protein from rcsb.org
 *website
 *
 * CATEGORIES: ui/www/web_client
 * KEYWORDS:   WWW
 */
int main(const int argc, const char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using namespace ui::www;

  http_t *request = http_get("http://files.rcsb.org/view/2GB1.pdb", NULL);
  if (!request) {
    std::cerr << "Invalid request.\n";
    return 1;
```

(continues on next page)

(continued from previous page)

```
  }

  http_status_t status = HTTP_STATUS_PENDING;
  int prev_size = -1;
  while (status == HTTP_STATUS_PENDING) {
    status = http_process(request);
    if (prev_size != (int) request->response_size) {
      std::cout << utils::string_format("%d byte(s) received.\n", (int) request->response_size);
      prev_size = (int) request->response_size;
    }
  }

  if (status == HTTP_STATUS_FAILED) {
    std::cerr << utils::string_format("HTTP request failed (%d): %s.\n", request->status_code, request->reason_phrase);
    http_release(request);
    return 1;
  }

  std::cout << "\nContent type: " << request->content_type << "\n\n" << (char const*) request->response_data << "\n";
  http_release(request);

  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:

```
#include <iostream>
#include <core/calc/structural/protein_angles.hh>
#include <core/calc/structural/angles.hh>
#include <core/data/structural/PdbAtom.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

std::string program_info = R"(

Test for z_matrix_to_cartesian() function recovers cartesian coordinates of a␣
↪fluoroethylene
from Z-matrix (internal coordinates)

USAGE:
./ex_z_matrix_to_cartesian

)";

/** @brief Test for z_matrix_to_cartesian() function
 *
 * This test recovers fluoroethylene from Z-matrix (internal coordinates)
 * CATEGORIES: core::calc::structural::z_matrix_to_cartesian
 * KEYWORDS:   internal coordinates
 */
int main(const int argc, const char *argv[]) {

  if ((argc > 1) && utils::options::call_for_help(argv[1]))
    utils::exit_OK_with_message(program_info);

  using core::data::structural::PdbAtom;
```

(continues on next page)

(continued from previous page)

```
  using namespace core::calc::structural;

  PdbAtom F(1, " F  ", -1.0606, 0.1723, 0.0001, core::chemical::AtomicElement::FLUORINE.z);
  PdbAtom C1(2, " C1 ", 0.1319, -0.4627, -0.0005, core::chemical::AtomicElement::CARBON.z);
  PdbAtom C2(3, " C2 ", 1.2458, 0.2325, 0.0001, core::chemical::AtomicElement::CARBON.z);
  PdbAtom H11(2, " H11", 0.1690, -1.5420, 0.0030, core::chemical::AtomicElement::HYDROGEN.z);
  PdbAtom H21(3, " H21", 2.1991, -0.2751, -0.0004, core::chemical::AtomicElement::HYDROGEN.z);
  PdbAtom H22(3, " H22", 1.2087, 1.3119, 0.0010, core::chemical::AtomicElement::HYDROGEN.z);


  PdbAtom H11_(2, " H11", 0,0,0, core::chemical::AtomicElement::HYDROGEN.z);
  PdbAtom H21_(2, " H21", 0,0,0, core::chemical::AtomicElement::HYDROGEN.z);
  PdbAtom H22_(2, " H22", 0,0,0, core::chemical::AtomicElement::HYDROGEN.z);
  core::calc::structural::z_matrix_to_cartesian(F, C2, C1, 1.0, to_radians(120), to_radians(180), H11_);
  core::calc::structural::z_matrix_to_cartesian(F, C1, C2, 1.0, to_radians(120), to_radians(180), H21_);
  core::calc::structural::z_matrix_to_cartesian(H21_, C1, C2, 1.0, to_radians(120), to_radians(180), H22_);
  std::cout << C2.to_pdb_line() << "\n";
  std::cout << C1.to_pdb_line() << "\n";
  std::cout << F.to_pdb_line() << "\n";
  std::cout << H11_.to_pdb_line() << "\n";
  std::cout << H21_.to_pdb_line() << "\n";
  std::cout << H22_.to_pdb_line() << "\n";
  double error = H11_.distance_to(H11);
  error += H21_.distance_to(H21);
  error += H22_.distance_to(H22);
  std::cout << "# Average error on the three hydrogen atoms: "<<error/3.0<<"\n";
}
```

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 output[1]
- 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 processing these files, which includes substructure extraction, format convertion and data filtering.

7.2.2 Alignments

Sequence alignment and multiple sequence alignment calculations inludes Smith & Waterman[2] and Needleman & Wunsh[3], 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 alphabets[4] 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

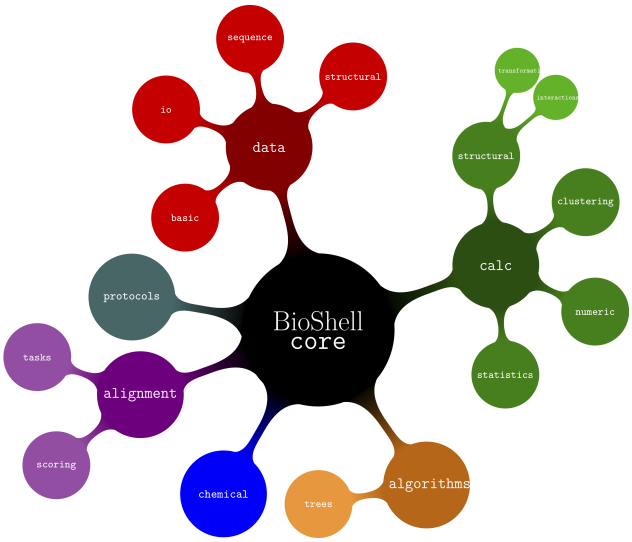
7.3.27 data structures

7.3.28 data table

CHAPTER 8

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 dramaticly 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␣
    ↪contact distance (in Angstroms)
8   core::data::io::Pdb reader(argv[2],filter); // --- file name (PDB format, may be␣
    ↪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 creaded based on the first model defined in the file
- in the **line 4** a `ContactMap` object is creaded 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:

```
#include <iostream>

#include <core/data/io/Pdb.hh>
#include <core/data/structural/Structure.hh>
#include <utils/options/OptionParser.hh>
#include <utils/exit.hh>

 */
int main(const int argc, const char *argv[]) {

  using namespace core::data::io; // Pdb and create_fasta_string lives there

  Pdb reader(argv[1], is_not_alternative, only_ss_from_header, true);
  core::data::structural::Structure_SP strctr = (reader.create_structure(0));

  // Iterate over all chains
  for (int ic = 0; ic < strctr->count_chains(); ++ic)
    std::cout << "> " << strctr->code() << (*strctr)[ic]->id() << "\n" // --- e.g. prints "> 2gb1 A"
      << (*strctr)[ic]->create_sequence()->sequence << "\n";           // --- prints the sequence itself
}
```

The same program written in Pyton 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:

```
import sys

from pybioshell.core.data.io import find_pdb

```

(continues on next page)

(continued from previous page)

```
GROUP:      File processing; Format conversion

  """)
  sys.exit()

for pdb_fname in sys.argv[1:] :
  structure = find_pdb(pdb_fname, "./").create_structure(0)
  for ic in range(structure.count_chains()) :
    chain = structure[ic]
    print(">",structure.code(), chain.id())
    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.

```
structure = pdb.create_structure(0)
for ic in range(structure.count_chains()) :
  chain = structure[ic]
  print(">",structure.code(), chain.id())
  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;

```
for i_model in range(pdb.count_models()) :
  structure = pdb.create_structure(i_model)
  for ic in range(structure.count_chains()):
    chain = structure[ic]
    for ir in range(chain.count_residues()):
      resid = chain[ir]
      for ia in range(resid.count_atoms()):
        if resid[ia].atom_name() == "CA" or resid[ia].atom_name() == "CB":
              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 overwriten.

```
reader = Pdb(pdb_fname,"is_ca",False)
  structure = reader.create_structure(0)
  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, loggging 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++ excpetion 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 system[1].

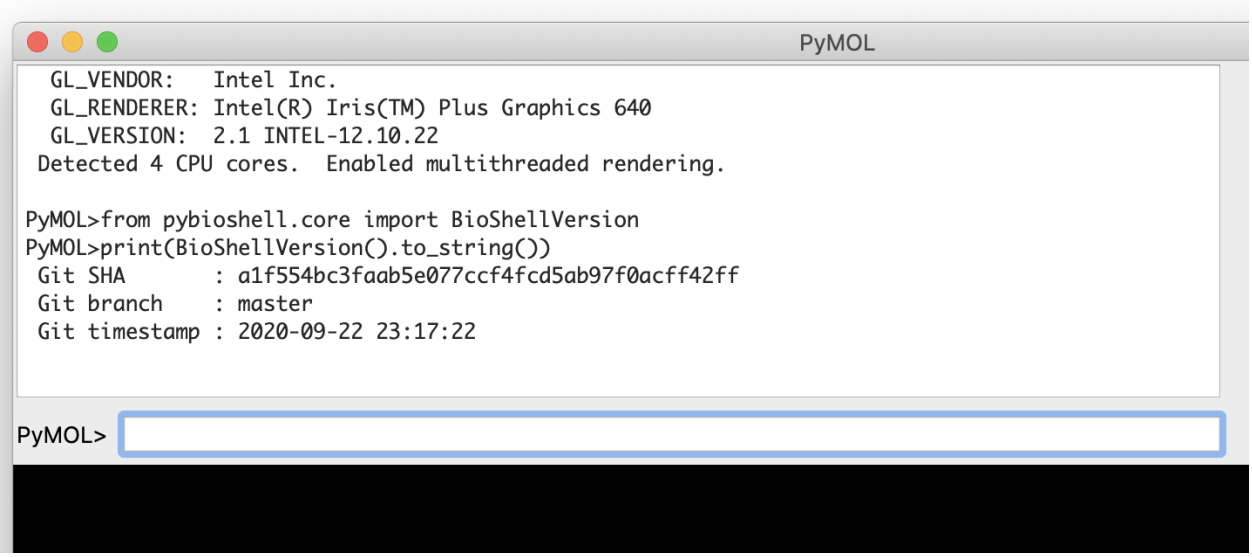
9.3.1 Loading PyBioShell

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*
- **Resutling** *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]

> [equasion 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 stiffnes

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 briliant 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

Index

Symbols

B

C

E