

Absolut! Documentation

1- How to install Absolut!



The absolut package allows to handle bindings of CDR3 Amino Acid sequences around virtual antigens in a 3D grid. Antigens can be converted from a real PDB structure to a lattice representation [*option discretize*], lists of CDR3 sequences can be 'folded' around the antigen, giving the list of optimal structural bindings [*option repertoire*], and features can be extracted from these bindings [*option getFeatures and poolFeatures*].

Three variants of the program can be installed / compiled depending on your needs:

- **AbsolutNoLib** is the easiest version to run, as it does not require any library. More than 200 antigens are available in the library [*option listAntigens*], and all data processing (bindings and features) are available.
- **Absolut** is the full version of the program, which additionally allows to discretize antigens and display them in 3D [*option visualize*]. A Qt-based graphical interface assists the discretization, but it is also possible to use qithout graphical interface and compile without the Qt library (see Advanced).
- **AbominationMPI** is the parallellized version specially designed to bind millions of CDR3 sequences around antigens [*option repertoire*] using the MPI library. Note that Delicab and Absolut are already multi threaded, but some cluster require MPI when the program is to be run on different nodes.

Downloading Absolut!: **will be in github soon**

Requirements:

- C++ compiler: g++ and gcc (that should normally include a linker ld). Linux: 'sudo apt-get install build-essential'. On MACs that would be clang 'brew install llvm'. Windows: mingw or visual C++.
- For using Qt, recommends installing qcreator with the qt distribution. Linux: sudo apt-get install qcreator, qt5-default, libqt5
- For discretizing antigens: Wget, for downloading the PDB files and python, for pdb-tool PDB processing.

Notes:

- When using Qt, do not use qmake in the folder where the original Makefile is (that would replace it). Suggest to use qcreator, it will compile in a separate folder.

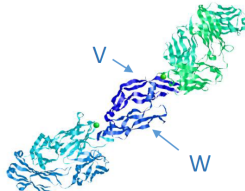
Absolut! Documentation

2- How to use Absolut! In a glance

Option 1: *discretize*: Open a graphical interface to discretize an antigen using LatFit.

```
./Delicab discretize 1CZ8 VW 5.25 FuC
```

Input 1: Antigen structure PDB name
1CZ8



Input 2: Chains in the PDB to discretize
VW One letter per chains, no separation

(optional) Input 3: Resolution of the 3D lattice
Default value: 5.25 Å

(optional) Input 4: Method of discretization
Tells which points from the PDB are transformed:
CA for Carbon Alpha
CoM for Centroid center of the side-chain only,
FuC for fused center of the whole AA [default value]

Output 1: Best representation of the selected chains in the 3D lattice:

1CZ8_VWInLattice.txt

Number of subchains: 2

Structure of each subchain:
133152 SULDDLUURRDDURDSLDUDRDDLLDDRUULUDRRLUSRDUDUULDRURRLUDDLUULRDLRSLDLDLASSDLRSRU
LDRULRLSRU
133160 USRSRUUDDLLUUDLSURUDLSUURRLURLLUDRUULSDRSLSDULRURUJDRSLRDLRLDDLUUDUURSLR
LRLDRRLURR

AA sequence (concatenated, all subchains):
VVYKFMVYQSYCHPIETLVDFQEPDEIYIFKPCVPLMRGCGCCNDEGLECVPTESNITMQIMRIKPHQGQIHIGEMSFLOHKNKCECRPKVVKFMDVYQ
RVSCHPIETLVDFQEPDEIYIFKPCVPLMRGCGCCNDEGLECVPTESNITMQIMRIKPHQGQIHIGEMSFLOHKNKCECRPK

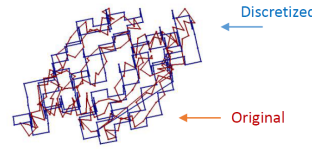
=> List additional inaccessible positions

124833	33 30 30	124898	34 31 30
128929	33 30 31	128999	39 31 31
132900	36 28 32	132903	39 28 32
133028	36 30 32	133031	39 30 32
137122	34 30 33		

C++ code to add this structure in the library:
string agStruct = "SULDDLUUR..."

1CZ8discretized5.25FuC.pdb

PDB containing the original chains in free space and their lattice version [direct output of latfit]



Side Outputs:

1CZ8deIns.txt Original PDB without insertions

1CZ8_VWprepared.pdb Only chains of interest from original PDB [used as input for latfit]

Inputs:

- PDB_ID The 4-character name of the PDB to discretize (will automatically download if not in the folder)
- Chains The names of the chains to be discretized (one char per chain, put together)
- Resolution The resolution of the 3D lattice [default 5.25]
- TypePos The type of positions used for discretization [**CA** for Carbon Alpha, **CoM** for Centroid center of the side-chain only, and **FuC** for fused center of the whole AA – default FuC]

Example:

```
./Absolut 1OB1 C  
./Absolut 1CZ8 VW  
./Absolut 1CZ8 VW 5.25 FuC
```

Outputs:

- Download the original PDB and fasta files from the PDB server, saves it into the running folder
- 1CZ8deIns.pdb new PDB with the same structures but where insertions are removed by shifting the ID of each residue (using pdb-tools). Note that side information of the PDB might be removed in this step, like the position of glycans.
- 1CZ8_VWprepared.pdb new PDB with only the chains of interest (as input for latfit)
- 1CZ8discretized5.25FuC.pdb Latfit output: new PDB file, one chain has the original positions (lines ATOM) and the positions in the 3D lattice (lines HETATOM)
- 1CZ8_VWInLattice.txt Description of the discretized antigen [Each chain is described as a starting position in the lattice (6-digits number) and a list of moves in space (straight S, up U, down D, left L, right R). See 'info_position' to convert lattice positions.

Note:

- A graphical interface will open, allowing to change the discretization parameters and visualize the results.
See the following pages for description of the graphical interface
- Please do not run this option in debug, the call to python pdb-tools scripts using system() usually fails (no idea)
- Please keep this order of arguments

Absolut! Documentation

Graphical interface for discretization:

The screenshot shows the Absolut! graphical interface for discretization. The interface is divided into several sections:

- PDB ID** (1RVZ) and **Filename** (1RVZ.pdb).
- View PDB in 3D** button.
- Chains read from the PDB** (A HEMAGGLUTININ).
- Chains of interest** (ACEGIKBDHFJL).
- FASTA** sequence view.
- Atom** selection (FuC).
- nKeep struct per iteration** (25).
- Allow jumps** checkbox.
- Lattice distance** (5.25).
- Average from PDB** (3.90).
- Iterate** button.
- Discretized PDB** output.
- Discretize into file** button.
- Convert into lattice into file** button.
- View PDB+ discretized in 3D** button.
- View receptor structures** button.
- Randomly picked** checkbox.
- Number to show** (0).
- Size receptors** (7).
- Min interact** (10).
- Multi-chain structure** list.

1: PDB file ID, will be downloaded directly from PDB database

2: Open Rasmol to view the original PDB. Suggestions: Use rasmol terminal to play: 'select :A' to select chain A, 'restrict :A' to only show chain A. Or 'select hetero' to select the glycans or so. Suggestion: colour shapely, show spacefill, and unselect hydrogens.

3: Shows the chains available in the PDB (note: sometimes PDB files have weird formats and chains do not appear)

Selection of chains to be processed:

4: List of chains to be further discretized. The Merge command will create a new PDB file where all those chains are put together in one chain.

5: Fasta information of the PDB. Note: some residues might be missing in the crystal structure, so not all AAs will appear in the structure.

Discretization using the latfit library:

6: What will be discretized. Three options: 'CA': only the Carbon Alpha of each amino acid is used. 'CoM': only the center of mass of the side-chain of the AAs is taken (centroid representation), but the backbone is not included. 'FuC': Fused Center, where the center of mass of both the side-chain and the backbone is taken.

7: Latfit reconstructs a structure from one tail, adding residues one by one in grid space positions. nKeep is the number of best structures so far it stores before adding the next amino acid. Min 1, suggest 25.

8: Some crystal structures have missing residues, or when you use more than one chain, there is a gap between the last AA of one chain to the first AA of the next. These are called jumps and we need to use this option when calling latfit.

9: During the discretization into lattice, we can decide what is the distance between two consecutive AAs. When discretizing the Carbon Alphas, the average distance is 3.9Å between two next ones. However, when discretizing centroids or fused centers, the average distance might bigger. Using higher distance between points generates a more compact structure (with less holes) and with lower distances it makes longer tails. The best is to look at the dRMSD between original and discretized structure for different lattice distances.

10: Average distance between C Alphas in the PDB file.

11: Automatically iterates all possible lattice distances and types of discretization (see 6).

12: Text of the generated PDB file after discretization. It contains the original structure of the points to be discretized in 3D (the C Alphas, or centroids, or fused centers), and the position of discretized points as well.

13: Runs latfit to discretize the selected chains into a lattice.

14: Reads the PDB generated by latfit and transforms into our lattice representation with Straight, Up, Down, Left, Right.

15: Visualization of the discretized lattice in 3D. Holes (defined as external points with 5 contacts or more to the structure, are marked by a green small sphere).

16: Generates all structures around the antigen and shows a small number of them, to see where the antibodies would bind. Options:

17: Normally, it takes an antibody sequence randomly and shows the top structures with higher total energy. However, by clicking random, a random set of structures would be shown instead.

18: Number of structures to show at the same time

19: Size of antibody (receptor) structures. This is number of bounds, so add +1 to have their number of AAs.

20: Minimum number of contact points of the structures.

21: Result of converting the discretized protein into our lattice system: Number of subchains, and for each subchain, the starting position in space (a 6-digit ID that encodes $x + 64y + 4096z$) and the structure. At the end, the list of AAs in the structure is given.

Commands when visualizing structures in the program:

Q/D: Previous/next structure. S: Axis B: Black / White background J: next heatmap (the last one is the merging of all)

A: Mode Amino Acid coloring / random color

H: Mode Heatmap / random color

I/L: smaller/larger AA spheres

Arrows (left/right): moves the structures.

X/Y/Z x/y/z: move the current point +/- along an axis

P: (on/off) prints the position of the current point in count

O: (on/off): Outputs the current display as image, automatic naming

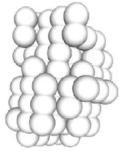
Absolut! Documentation

Option 2: *repertoire*. Calculates the best binding of CDR3 sequences around a discretized antigen from the library.

```
./Delicab repertoire 1FBI ListCDR3s.txt NbThreads
./AbsolutNoLib repertoire 1FBI ListCDR3s.txt NbThreads
mpirun -n NbProcesses ./AbominationMPI repertoire 1FBI ListCDR3s.txt NbThreadsPerProc
```

Input 1: Antigen structure ID (inside the code)

1FBI
1FBI_X



Input 2: Precalculated possible structures (200MB)

The file should be present in the folder of calculation
./AbsolutNoLib info_fileNames 1FBI

Input 3: List of CDR3s to 'structurally annotate'


```
1 CAGPSTTPVYFDYW
2 CARAYYSNDYW
3 CARWDDYDDWFAYW
4 CARESSGYGYW
5 CARYNYGPMYDW
...
```

(optional) **Input 4: Tag** to be put in output filenames

(optional) **Input 5: First to Last** lines to process

Output: Annotated Files

One file per process; Each CDR3 annotated with binding structures / energy
Ex: 1FBI_XFinalBindings_Process_2_Of_2.txt

CDR3			CARDIVTTWPYYAMDYW			
Slides (11 AA)			CARDIVTTWPY			
			ARDIVTTWPY			
			RDIVTTWPYA			
#Antigen 1FBI	ID_slide_Variant	CDR3	Best?	Slide	Energy	Structure
42881_00a		CARDIVTTWPYYAMDYW	false	CARDIVTTWPY	-54.77	129120~DSLRLRDDSD
42881_01a		CARDIVTTWPYYAMDYW	false	ARDIVTTWPY	-61.49	125152~UUUSURUDUD
42881_02a		CARDIVTTWPYYAMDYW	false	RDIVTTWPYA	-54.07	141278~SLSSRRSDSD
42881_03a		CARDIVTTWPYYAMDYW	false	DIVTTWPYAM	-59.4	121119~RUSLLSDDUD
42881_03b		CARDIVTTWPYYAMDYW	false	DIVTTWPYAM	-59.4	116959~USSDSDRLRL
42881_04a		CARDIVTTWPYYAMDYW	false	IVTTWPYAMD	-62.9	121055~USSDSDRLRL
42881_05a		CARDIVTTWPYYAMDYW	false	VTTWPYAMDY	-64.95	137374~URDSRUURDU
42881_06a		CARDIVTTWPYYAMDYW	true	TTWPYAMDYW	-65.09	141405~SSSDRRDLRS
42882_00a		CARDKGAYSNSWYFDVW	false	CARDKGAYSNS	-47.55	137312~RULUUSUSDS
42882_01a		CARDKGAYSNSWYFDVW	true	ARDKGAYSNSW	-48.6	129120~DSLRLSLLD

Temp. Output: Backup files with ongoing calculations, in case of crash (1/thread)

Ex: TemporaryBindingsFor5KN5_Ctest_t1_Part1_of_2.txt

Inputs:

- **Antigen_ID** The ID of the antigen in the library. The full name would contain the chains, for instance 1FBI_X. However, provided there is only one antigen for this PDB ID, **we can use 1FBI** as shorter name.
- **Precomputed structure file.**

Ultimately, the program needs three precomputed files for an antigen. A list of structures, a dictionary of structures to binding codes, and a compact list of binding codes. These (weird) files look like:

- SUDRRec88cb568db9617734142c2e1cb8ae9e-10-11-91fbe8b6e59930b8e5270c6093af8136**Structures.txt**
- SUDRRec88cb568db9617734142c2e1cb8ae9e+NISQH9359bc275d9357a840bfec064d42ea6a-10-11-91fbe8b6e59930b8e5270c6093af8136.txt
- SUDRRec88cb568db9617734142c2e1cb8ae9e+NISQH9359bc275d9357a840bfec064d42ea6a-10-11-91fbe8b6e59930b8e5270c6093af8136**Compact.txt**

The first file (Structures.txt) is smaller (like 100MB) while the two other ones can be huge (1GB). Luckily, the program can recompute the files ii) and iii) quite fast, so **you just need to provide the Structures file.**

Where to find the structures file? philippe-robert.com/Absolut/Structures/

If the structure file is not available?

You need to recompute it, can take up to 5 days [not parallelized]. Use the command **singleBindings** for that.

- **Repertoire file:** List of CDR3 to process, with a unique ID for each of them [ID can be integer or string or whatever] Should be two columns: ID then CDR3 sequence.

Outputs: Annotated raw binding file

Absolut! Documentation

Advanced: Installing the required libraries for Windows

Installing Qt framework <https://www.qt.io/> => download => open source

Recommend to use a Qt distribution that includes a C++ compiler if you don't have:

<https://www.qt.io/offline-installers>

Installing gsl library

<https://sourceforge.net/projects/gnu-scientific-library-windows/>

Make sure git is installed in a folder WITHOUT SPACES, so not in Program Files => needed to reinstall manually outside the program center. Git can be found here: <https://git-scm.com/download/win>

Find mingw32-make.exe and rename it to mingw32-make.exe as just make.exe and make sure it's in the path, so that make can be run directly on a command line.

```
./configure --host=x86_64-w64-mingw32 --prefix=/mingw/local --enable-shared --enable-static
```

```
$ make
```

```
$ make install
```

Then the dll are created in the .libs folders

Same thing for OpenGL libraries. Seems GLUT is not supported anymore => Change to freeglut. Good thing, we can have multiple time a GlutMainLoop window, need to find how to do it later. So, to install freeglut:

Install freeglut

Help can be found there <https://medium.com/@bhargav.chippada19/how-to-setup-opengl-on-mingw-w64-in-windows-10-64-bits-b77f350cea7e>

Needed to install cmake

<https://cmake.org/download/>

and to put it's folder in the path

Then downloaded freeglut

<http://prdownloads.sourceforge.net/freeglut/freeglut-3.0.0.tar.gz?download>

Then from CMD (and not from git bash), inside the unpacked freeglut, do:

```
cmake -G "MinGW Makefiles" -S . -B . -DCMAKE_INSTALL_PREFIX= x86_64-w64-mingw32
```

```
do make all
```