Published in: Rosell M., Rodríguez-Lumbreras L.A., Fernández-Recio J. (2020) Modeling of Protein Complexes and Molecular Assemblies with pyDock. In: Kihara D. (eds) Protein Structure Prediction. Methods in Molecular Biology, vol 2165. Humana, New York, NY. https://doi.org/10.1007/978-1-0716-0708-4_10

Modelling of protein complexes and molecular assemblies with pyDock

Mireia Rosell,[1,2] Luis Angel Rodríguez-Lumbreras,[1,2] Juan Fernández-Recio[1,2,3*]

[1]Barcelona Supercomputing Center (BSC), Barcelona, Spain

[2]Instituto de Ciencias de la Vid y del Vino (ICVV), Consejo Superior de Investigaciones Científicas (CSIC) - Universidad de La Rioja - Gobierno de La Rioja, Logroño, Spain

[3]Institut de Biologia Molecular de Barcelona (IBMB), Consejo Superior de Investigaciones Científicas (CSIC), Barcelona, Spain

Running head:

Modeling of protein complexes with pyDock

Summary

The study of the 3D structural details of protein interactions is essential to understand biomolecular functions at the molecular level. In this context, the limited availability of experimental structures of protein-protein complexes at atomic resolution is propelling the development of computational docking methods that aim to complement current structural coverage of protein interactions. One of these docking approaches is pyDock, which uses van der Waals, electrostatics and desolvation energy to score docking poses generated by a variety of sampling methods, typically FTDock or ZDOCK. The method has shown a consistently good prediction performance in community wide assessment experiments like CAPRI or CASP, and has provided biological insights and insightful interpretation of experiments by modeling many biomolecular interactions of biomedical and biotechnological interest. Here we describe in detail how to perform structural modeling of protein assemblies with pyDock, and the application of its modules to different biomolecular recognition phenomena, such as modeling of binding mode, interface and hot-spot prediction, use of restraints based on experimental data, inclusion of low-resolution structural data, binding affinity estimation, or modeling of homo- and hetero-oligomeric assemblies.

Keywords

Protein-protein interactions, protein structure prediction, computational docking, template-based modeling

1. Introduction

Understanding the structure and energetics of protein interactions has potential applications in diverse fields such as biomedicine, biotechnology or agricultural sciences. However, the experimental determination of structural information on the interactome lags well behind the amount of proteomics and genomics data that are being produced at growing pace. In this context, computational structural prediction is a valuable complement to experimental data and an essential help to interpret genomics information. Indeed, a variety of protein-protein docking tools have been reported to model the atomic details of the interaction between two given proteins [1,2]. One of such docking approaches is pyDock [3], which uses energy-based function to score docking poses generated by a variety of sampling methods. The distributed version for local running is optimized to be automatically used with the sets of rigid-body docking poses obtained by FTDock 2.0 [4] or ZDOCK 2.1 [5] , but it has been applied to score either flexible or rigid docking models from other programs, such as RotBus [6], SwarmDock [7] , ZDOCK 3.0 [8], SDOCK [8], or LightDock [9]. The method is also available as a web server, pyDockWEB, using pyDock version 3 and including a custom parallel FTDock version based on MPI libraries, with grid size optimization for efficient use of FFTW libraries and multi-processor running [10].

The key aspect of pyDock is a unique combination of electrostatics, desolvation and van der Waals terms, with weighting factors originally optimized for a small set of cases to avoid overfitting. Indeed, the method has shown to be robust throughout the years, in regards to its application to new benchmark cases and to different types of interactions. The method has been successfully tested in CAPRI (http://www.ebi.ac.uk/msd-srv/capri/capri.html) and CASP (http://predictioncenter.org/) community wide assessment experiments. Indeed, in the most recent CASP13 edition, the performance of our group in the multimeric targets, based on pyDock models, was ranked within the top 3 groups from a total of over 40 participants, showing the potential of pyDock to model multi-molecular assemblies, including oligomers and multi-domain proteins.

In addition to docking prediction, pyDock provides a variety of additional modules to analyze fundamental problems in biomolecular recognition. The module pyDockNIP, which analyzes the frequency of interface residues in low-energy docking models from pyDock [11], has been reportedly applied to identify interface hot-spot residues [12], that is, residues that contribute the most to the binding affinity, which can be relevant for drug discovery targeting protein-protein interactions with small molecules. The module pyDockSAXS is the first systematically tested approach in using protein docking models to complement low-resolution structural data from Small Angle X-ray Scattering (SAXS) [13-15]. On the other side, interface residue data from bioinformatics predictions, mutational experiments, NMR, cross-linking, etc., can be included as distance restraints with pyDockRST module [16]. Although originally aimed to protein-protein docking, pyDock can be also applied to model protein interactions with other biomolecules, such as protein-RNA [17]. On a more practical side, the pyDock methodology has provided biological insights and helped to interpret experiments in different cases of biomedical and biotechnological interest. One remarkable example is the structural study of host-pathogen complexes, in which pyDock docking together with energetic analysis helped to interpret molecular mechanisms for [18]. Another case of interest is the application of pyDock within a broad structural analysis of members of the family of Hetero Aminoacid Transporters (HAT), such as the integrative modeling of the assembly of transmembrane LAT2 and its ancillary protein 4F2hc, using a combination of modeling, docking, electron microscopy and cross-linking experiments [19].

In this chapter, we will use a protein-protein case of known 3D structure to illustrate the use of the different modules in pyDock for the structural modeling of protein complexes and the characterization of molecular assemblies.

2. Materials

2.1. Input

The pyDock software needs the coordinates of the two interacting proteins, usually as PDB files, but it can also take AMBER coordinate and topology files. When using PDB files, hydrogens are not needed, and if present, they will be removed and rebuilt again by pyDock. In addition, all HETATM coordinates will be removed in the docking calculations.

The pyDock method can use AMBER coordinate files (with extension as `.inpcrd`, `.restrt`, `.rs7`, `.crd`) and topology files (with extension as `.prmtop`, `.parm7`, `.top`) created by the PARM, LeAP, SANDER, or GIBBS programs from AMBER [20]. In this case, the cofactors and other compounds will be included in pyDock calculations.

2.2. Programs

2.2.1. pyDock

The pyDock 3.0 package is available at `https://life.bsc.es/pid/pydock/` (to get pyDock you need to apply for a license by filling in your data; for academic use, you will receive a link to the pyDock distribution file by e-mail; for commercial use, you will be contacted by the authors). Uncompress and untar the pyDock distribution file to extract the `pyDock3` directory.

Next, we need to change permissions of the `pyDock3/data` directory:

```
chmod go+rx data
```

The `pyDock3` directory can be moved to any location of your choice. For instance, let say that it is moved to `/usr/local/software/` directory, then you could call pyDock by:

```
/usr/local/software/pyDock3/pyDock3
```

Moreover, we can define the `PYDOCK` variable in your `.bashrc` file, as follows:

```
export PYDOCK=/usr/local/software/pyDock3/
```

so that the executable of pyDock can be called in a more convenient way:

```
$PYDOCK/pyDock3
```

The pyDock binary has been compiled for Linux 32-bit (see more installation details in Note 1).

2.2.2. FTDock

We need some external programs to generate a set of rigid-body docking poses. In this regard, pyDock is ready to process the output of FTDock 2.0, and we will show here how to install it. The first step is to install the FFTW libraries (see Note 2). Once they are properly installed, we can download the FTDock 2.0 installation file `gnu_licensed_3D_Dock.tar.gz` from `http://www.sbg.bio.ic.ac.uk/docking/download.html` (by clicking in the appropriate link). Uncompressing this file will extract its contents to a new directory called `3D_Dock` (which contains, among others, the folder `progs` with all the needed binaries). Now, within the new `progs` directory, open the *Makefile* file and edit the following lines:

1. FFTW_DIR line: define the full path of the fftw-2.1.5 directory (see Note 2).

   (e.g. `FFTW_DIR= /<your-installation-directory>/fftw-2.1.5`)

2. CC_FLAGS line: remove the `-malign-double` argument.

3. CC_FLAGS line: define `-mcpu=k8` # instead of the default `-mcpu=pentiumpro`

Now within the `progs` directory, compile the program with `./make` (ignore WARNING messages).

2.2.3. SCWRL

In case of incomplete side-chains in the input PDB files, we can use SCWRL (`http://dunbrack.fccc.edu/`) to rebuild them. To use it automatically from pyDock (see Note 3), we need to install SCWRL 3.0. This version is outdated and cannot be directly downloaded from the above web, so you need to obtain the installation file `scwrl3_lin.tar.gz` from their authors. Uncompressing this file will extract its contents to a new directory called `scwrl3_lin`. Within this directory, run:

```
./setup
```

This will create a SCWRL 3.0 binary (`scwrl3`) in such directory.

2.2.4. Other external programs

We can also use ZDOCK (`http://zdock.umassmed.edu/software/`) for the generation of rigid-body docking poses. The pyDock pipeline is ready to process the output of ZDOCK 2.1, so it is advisable to download and install that version. This and the other above programs can be run either manually following each program's instructions, or automatically within pyDock according to instructions herein (see Note 3).

For some functions useful for the analysis of the docking results, as later explained, we will use ICM-Browser (www.molsoft.com).

2.3. Server

The pyDock method is also available as a web service at `https://life.bsc.es/pid/pydockweb`. The web front-end acts as a proxy to the user, removing any complexity aroused from a local installation of the software. Via a user-friendly interface, the user is capable of uploading molecular structural information in PDB format. After finishing, the user will receive the results in a web page, together with all the files generated during the docking project. These files might also be useful for local execution of pyDock, in order to start from a given intermediate step.

3. Methods

pyDock has a highly modular architecture, with a series of modules performing the different functionalities of the program (Figure 1). The general syntax for running pyDock is:

```
$PYDOCK/pyDock3 DOCKNAME modulename
```

Thus, the executable pyDock3 usually needs two arguments: i) *DOCKNAME*, which is the name of the pyDock project and the base for all the files that will be created during the docking pipeline, and ii) *modulename*, which will call for the specific module. The details of the different pyDock modules are described in the running instructions below.

3.1. Parameter file

First, you need to create a text file called `DOCKNAME.ini` , in which *DOCKNAME* will be the name of the pyDock project. This file will contain all the needed information about the interacting proteins (PDB files, chain IDs...).

We will illustrate this with an example, in which we will model the structure of the complex formed between the proteins TolB and Pal (PDB 2HQS) from *E. coli*, involved in maintaining the bacteria outer membrane stability [21]. For this example, we also have the structures of the unbound TolB and Pal proteins: PDB 1c5k (chain ID A) and 1oap (chain ID A), respectively. We will download the coordinate files of these proteins (`1c5k.pdb` and `1oap.pdb`) from `www.pdb.org`. In that case, we will create a text file called `Dock1.ini` (we will use "Dock1" as base name for the docking files) with the following information:

```
[receptor]
pdb        = 1c5k.pdb
mol        = A
newmol     = A

[ligand]
pdb        = 1oap.pdb
mol        = A
newmol     = B
```

The `mol` field is the original chain ID (1 character) of the protein chain/s that will be used for docking, whereas the `newmol` will be the name of the chain ID of that protein in the docking output files (see Note 4). For convention, one of the proteins (usually the largest one) will be the receptor (static position), and the other the ligand (mobile position).

The program can also take a protein structure in AMBER format. To illustrate this with the above example, suppose we generate coordinate files (`1c5k.inpcrd, 1oap.inpcrd`) and topology files (`1c5k.prmtop, 1oap.prmtop`) for our receptor and ligand molecules with LEaP program from AMBER. Then, we should use the following initial file to define the input structures:

```
[receptor]
pdb        = 1c5k.inpcrd,1c5k.prmtop
mol        = -
newmol     = A

[ligand]
pdb        = 1oap.inpcrd,1oap.prmtop
mol        = -
newmol     = B
```

The order of the AMBER files in the `pdb` field is important: it should be first coordinate file, then topology file (with any valid extension as mentioned in section 2.1). We should also note that AMBER coordinate and topology files do not have chain ID. In a case like this, we can define "`mol = -` " or "`mol =` ", and it will take all molecules with no chain ID in the corresponding file.

3.2. Setup the receptor and ligand coordinate files for docking

Before any docking calculation, we need to generate the coordinate files correctly parsed for pyDock, from the receptor and ligand PDB files indicated in the `DOCKNAME.ini` parameter file. For that, run the pyDock setup writing the following line in your console:

```
$PYDOCK/pyDock3 Dock1 setup
```

This command will create the new PDB files for receptor and ligand `Dock1_rec.pdb` and `Dock1_lig.pdb` respectively, which are suitable as input for pyDock.

Some PDBs may have incomplete side-chains, that is, there are missing atoms in the structures. These side-chains can be rebuilt with external programs. One of them is SCWRL, which can be run automatically from pyDock if its location is indicated in the `pydock.conf` file (see Note 3).

3.3. Generating rigid-body docking poses

pyDock can be applied to score rigid-body docking orientations generated by a variety of methods, but it is ready to automatically process the output from ZDOCK 2.1 or FTDock 2.0 docking programs. These programs can be run independently, but we will describe here how to call them automatically from pyDock, for which they should be previously installed (see section 2.2) and their location indicated in `pydock.conf` file (see Note 3).

3.3.1. Running FTDock within pyDock

Before running FTDock within pyDock, the program expects the FTDock preprocessed files for receptor and ligand (in our example: `Dock1_rec.parsed` and `Dock1_lig.parsed`). These files should be created by using the FTDock utility `preprocess-pdb.perl`, but since we already have suitable files from the pyDock setup step (described in section 3.2), we can just create the parsed files by copying the pyDock receptor and ligand files:

```
cp Dock1_rec.pdb Dock1_rec.parsed

cp Dock1_lig.pdb Dock1_lig.parsed
```

Then, we can use the following commands to run FTDock with pyDock:

```
$PYDOCK/pyDock3 Dock1 ftdock
```

This will produce the file `Dock1.ftdock`, where all the docking poses will be stored.

FTDock parameters can be changed for higher precision (see Note 3). In case of multi-core computer architectures, it could be convenient to execute a FTDock MPI version (see Note 5).

3.3.2. Running ZDOCK within pyDock

In our example, we can use the following commands to run ZDOCK (if previously installed) with pyDock:

```
$PYDOCK/pyDock3 Dock1 zdock
```

This will produce the file `Dock1.zdock`, where all the resulting docking poses will be stored.

3.4. Converting the rigid-body docking poses to pyDock format

3.4.1. Converting FTDock output to pyDock format

Now we need to transform the output data from FTDock (`Dock1.ftdock` in our example; in which each solution is represented by the position of the ligand in Cartesian coordinates, and its rotation based on Euler angles) to the rotation and translation matrix that transforms the original ligand coordinates into the different orientations generated by FTDock. This is done by using the following command:

```
$PYDOCK/pyDock3 Dock1 rotftdock
```

This calculation is quite fast and will create a file (`Dock1.rot` in our example) containing the above mentioned transformation matrices for the ligand in all docking poses.

3.4.2. Converting ZDOCK output to pyDock format

We can also transform the output data from ZDOCK (`Dock1.zdock` in our example) to a file suitable for further pyDock scoring, containing the rotation and translation matrix for each docking solution:

```
$PYDOCK/pyDock3 Dock1 rotzdock
```

This calculation is quite fast and will create a file (`Dock1.rot` in our example) containing the above mentioned transformation matrices for all docking poses. Note that the name of the file (`Dock1.rot` in our example) is the same as that from FTDock output, so to avoid overwriting files, it is advisable to rename the original files to keep them with different names (e.g. `Dock1.rot.ftdock`, `Dock1.rot.zdock`).

The docking sets obtained from different docking programs (e.g. FTDock, ZDOCK...) can be scored independently (see section 3.5) by keeping their `DOCKINGNAME.rot` files separated, but they can also be merged into a single docking set for its further processing (see Note 6).

3.5. Scoring the rigid-body docking poses with pyDock

Next step is to use pyDock energy function to score and rank all positions by running *dockser* module with the following command:

```
$PYDOCK/pyDock3 T26 dockser  > dockser.log  &
```

In case of multi-core computer architectures, it could be convenient to execute this scoring step in parallel (see Note 7).

The main output of the pyDock scoring step is a table (`Dock1.ene` in our example) with the detail of the different energy terms for each docking pose. See below the results obtained for this example when using FTDock with the default parameters in `pydock.conf` (`calculate_grid=1.2` and no electrostatics):

```
Conf(1)   Ele(2)      Desolv(3)   VDW(4)     Total(5)     RANK(6)
-------------------------------------------------------------------
3740     -15.199      -8.536      -1.041      -23.839        1
3496     -15.801      -8.436       6.326      -23.604        2
3847     -14.181      -9.301      21.455      -21.336        3
7260      -8.081     -12.723      -1.873      -20.992        4
(...)
```

(1) Conformation number of the docking pose (same as that in the .rot file, last column)
(2) Electrostatic energy term
(3) Desolvation energy term
(4) Van der Waals energy term
(5) Total binding energy (Ele + Desolv + 0.1*VDW)
(6) Rank of the docking pose according to its total binding energy

3.6. Analysis of the docking results

3.6.1. Generating the best-scoring docking models

We can generate the PDB file of a selection of resulting docking poses with the *makePDB* pyDock module, by indicating a range of models as ranked in the docking energy file. In our example, we will build the PDB files for the docking poses ranked 1 to 3 in the `Dock1.ene` file, as follows:

```
$PYDOCK/pyDock3 Dock1 makePDB 1 3
```

This will create three files named `Dock1_3740.pdb`, `Dock1_3496.pdb`, and `Dock1_3847.pdb`, whose names will indicate the conformation numbers (*Conf* column in `Dock1.ene` file) of these top 3 ranked docking models.

3.6.2. Comparing the results with reference complex structure

In some cases, we would like to compare the docking models with a reference complex structure (for test purposes when the complex structure is known; in case of available structure of a complex involving homologous proteins, etc.). While there are different quality measures, one of the most popular is *ligand RMSD*, that is, the RMSD between the positions of the ligand in the reference and modelled complexes, after superimposing the receptor chains of both complexes. This value can be automatically computed by pyDock *dockser* module, if a suitable reference complex is indicated and properly setup in the *`DOCKNAME`*`.ini` file.

In our example, we can use as reference the file 2hqs.pdb (PDB entry 2HQS downloaded from www.pdb.org), and define the following `Dock1.ini` file:

```
[receptor]
pdb        = 1c5k.pdb
mol        = A
newmol     = A

[ligand]
pdb        = 1oap.pdb
mol        = A
newmol     = B

[reference]
pdb        = 2hqs.pdb
recmol     = A
ligmol     = H
newrecmol  = A
```

```
        newligmol  = B
    ______________________________
```

The `newrecmol` field must indicate the same chain ID as `newmol` for the receptor, and the `newligmol` must be the same as `newmol` for the ligand. In this way, when running the *setup* module, the reference file `Dock1_ref.pdb` will be created. And, when running the *dockser* module, the ligand RMSD values for all docking poses in the `Dock1.rot` file will be calculated and shown in the `Dock1.ene` file, in the *RMSD* column. If we have an energy table without RMSD values from a previous execution, and we do not want to execute *dockser* again because of computational cost, we can run pyDock *rmsd* module to calculate and save RMSD values in a separate file `Dock1.RMSD` (which has to be manually added to `Dock1.ene`).

For testing purposes, we usually consider a docking pose as acceptable (a.k.a. near-native) when ligand RMSD < 10 Å, following CAPRI criteria [22]. In our example, the best-ranked near-native model by pyDock (using FTDock within pyDock with the default parameters in `pydock.conf`) is ranked 61 (Figure 2).

3.6.3. Interface and hot-spot prediction from docking

Based on the docking results, it is possible to identify binding regions [11] and to estimate the most likely hot-spot residues [12], by using the pyDockNIP module, which can be called with pyDock *patch* argument. In our example:

```
$PYDOCK/pyDock3 Dock1 patch
```

The resulting `Dock1.ligNIP` and `Dock1.recNIP` files contain the list of ligand and receptor residues with their corresponding *NIP* (Normalized Interface Propensity) values. The `Dock1_rec.pdb.nip` and `Dock1_lig.pdb.nip` files are PDB files in which the B-factor column is replaced by the *NIP* values, so that the results can be easily visualized with most molecular graphic programs (see Note 8).

3.7. Additional pyDock features

3.7.1. Improving docking with distance restraints

In some cases, there is available information on residues of the interacting proteins that might be potentially located at the interface or directly involved in the interaction. Such data can be derived from a variety of sources: mutational experiments, NMR data, bioinformatics analysis, biophysical studies, etc. In those cases, we can use this information to define distance restraints between such potential interface residues in one of the proteins and any residue of the partner one. Then, we can evaluate to what extent such distance restraints are fulfilled in a given docking pose, and use this as a complementary score to the pyDock energy.

In our example, we know from mutagenesis experiments [23] that residues H246, A249 and T292 from TolB, and A88 and E102 from Pal, seem to be directly involved in the interaction between these proteins. We can indicate these residues in the `restr` field of `Dock1.ini` file (see Note 9 for more details):

```
__________________________________________

[receptor]
pdb        = 1c5k.pdb
mol        = A
newmol     = A
restr      = A.Hid.246,A.Ala.249,A.Thr.292


[ligand]
pdb        = 1oap.pdb
mol        = A
newmol     = B
restr      = B.Ala.88,B.Glu.102
__________________________________________
```

Then, we can evaluate the fulfillment of the distance restraints of every docking model in `Dock1.rot` as follows:

```
$PYDOCK/pyDock3 Dock1 dockrst > dockrst.log   &
```

The resulting file `Dock1.rst` contains the scoring of each docking model based on the distance restraints. This file will be automatically combined with the existing `Dock1.ene` file to produce the `Dock1.eneRST` file, in which the docking models will be ranked according to the total scoring (*Total* column), obtained from a combination of pyDock energy (*Total* column in `Dock1.ene`) and restraint-based scoring (*relRST* column). In our example, the near-native docking pose ranked 61 with pyDock, became ranked 6th after including distance restraints in the scoring, and there is another docking pose of even better quality that was ranked 3275 by pyDock alone and became rank 35 after including distance restraints (Figure 2).

3.7.2. Improving docking with SAXS data

Small-angle X-ray scattering (SAXS) technique can provide low-resolution structural information to help characterizing biomolecules and macromolecular assemblies. If SAXS data is available for a given protein-protein complex, it can be used in combination with pyDock scoring to improve the identification of the correct docking models. This approach, called pyDockSAXS [13] has been implemented as a web server (`https://life.bsc.es/pid/pydocksaxs`) [14,15]. We need as input the PDB files with the atomic coordinates of the interacting proteins (either structures or models), and a file containing SAXS experimental data compatible with CRYSOL software version 2.8 [24].

3.7.3. Computing the docking energy score for a single complex structure

Using the pyDock *bindEy* module, we can compute the pyDock docking energy for a given complex structure (either experimentally determined or modelled). In our example, if we want to compute the pyDock energy between receptor and ligand in the reference complex structure (PDB 2HQS), we will define a new Dock2.ini file, indicating the corresponding chains for receptor and ligand in such complex structure:

```
[receptor]
pdb         = 2hqs.pdb
mol         = A
newmol      = A

[ligand]
pdb         = 2hqs.pdb
mol         = H
newmol      = B
```

Now we just need to run the *bindEy* module as follows:

```
$PYDOCK/pyDock3 Dock2 bindEy
```

This will create a `Dock2.ene` table, with the energy terms (individual and total) for only one row, corresponding to the complex structure.

3.7.4. Structural modeling of multi-domain proteins by docking

One of the pyDock applications is to model multi-domain proteins by applying a distance restraint between aminoacids. This can be done with *pyDockTET* method [25], which can be called by *docktet* module.

To illustrate this, we can use our example, given that TolB protein is composed of two clear structural domains as defined by CATH:

```
http://www.cathdb.info/version/latest/domain/1c5kA02
```

Thus, we can split PDB 1C5K in two files, representing the two domains: one called `1c5k_D1.pdb` with the first domain (D1) that can be defined by residues 1-158, and the other called `1c5k_D2.pdb` with the second domain (D2) defined by residues 166-389. Now we will try to rebuild the entire 2-domain protein by docking the individual domains with *pyDockTET*, using distance restraints derived from the length of the linker between residues 158-166 (7 residues). The parameter file `Dock3.ini` will be defined as follows:

```
[receptor]
pdb         = 1c5k_D1.pdb
mol         = A
newmol      = A

[ligand]
pdb         = 1c5k_D2.pdb
mol         = A
newmol      = B

[tether]
receptor    = A.Thr.158
ligand      = B.Thr.166
length      = 7
```

Then, the following steps will be followed to complete the *pyDockTET* procedure, i.e. docking with linker-based distance restraints.

```
$PYDOCK/pyDock3 Dock3 setup
$PYDOCK/pyDock3 Dock3 ftdock
$PYDOCK/pyDock3 Dock3 rotftdock
$PYDOCK/pyDock3 Dock3 dockser
$PYDOCK/pyDock3 Dock3 docktet
```

The output will be a `Dock3.eneTET` table, with the combined pyDock and linker-based restraint energies for each docking model.

4. Case Studies

4.1. Integrative modelling with docking, cross-linking and EM data

The pyDock method has been applied to model many complexes of biological interest, which usually requires a multidisciplinary effort and the integration of a variety of additional computational and experimental data. One interesting case was the first reported model for the assembly of the two subunits of a heteromeric amino acid transporter (HAT) [19]. In this case, an integrative modeling approach was applied to model the interaction between LAT2 with is ancillary protein 4F2hc. First, the transmembrane LAT2 protein was computationally modelled with Modeller 8v1 [26], based on a template from the same HAT family (AdiC, PDB 3OB6). Modelling was challenging due to the low sequence identity (SI) with the template (around 20%), but multiple sequence alignments suggested that the transmembrane domains were highly conserved among the members of the family. Indeed, the models generated for LAT2 indicated high conservation in the topology of the transmembrane helices, but large flexibility in the extracellular loops. The 20 models with best DOPE score from Modeller were docked with the x-ray structure of 4F2hc extracellular domain (PDB 2DH2) by using FTDock 2.0 and ZDOCK 2.1. The resulting 240,000 docking poses were filtered based on distance restraints from a known disulfide bond between subunits, by using pyDockTET (with restraint distance 14 Å). Then, docking poses that would clash with the expected position of the membrane were also removed, which left 3,145 docking models. Finally, the docking model with the best pyDock energy (here we did not include van der Waals term due to the uncertainty in the transmembrane protein models) was found to be in line with the overall shape provided by transmission electron microscopy (TEM) data, and fully consistent with available cross-linking experiments. This model (Figure 3) was further confirmed with additional cross-linking experiments. Finally, by applying pyDock *patch* module, the most important LAT2 residues for the interaction with 4F2hc were identified. Overall, this integrative model generated by a variety of experimental and computational methods provided the first structural insights on the interactions between the two subunits in HAT proteins, helping to understand the stabilizing role of the light subunit by 4F2hc, and its implications for other transporters.

4.2. Docking a homo-dimer with rotational symmetry

The pyDock methodology can be also applied to model protein homo-oligomers, in which two important aspects should be considered. First, when modeling homo-oligomers usually the monomer needs to be modelled (either template-based or *ab initio*), since its unbound structure is rarely available. Second, for practical purposes, we should assume symmetric oligomerization, e.g. rotational symmetry C_2, C_3..., in order to filter the resulting docking models. Based on these considerations, pyDock has been successfully applied in blind conditions to the modeling of a variety of homo-oligomers in the 13th community wide experiment on the Critical Assessment of Techniques for Protein Structure Prediction (CASP), as part of the common CASP13-CAPRI Assembly prediction challenge (`http://predictioncenter.org/casp13/zscores_multimer.cgi`).

One interesting case study in this CASP13 edition is target T1009 (CAPRI code T154), consisting in the homodimeric assembly of α-xylosidase A (from *Aspergillus niger*), based on its monomer sequence formed by 718 residues. We obtained the monomer coordinates from the models already available at the CASP-hosted servers. More specifically, we took the rank #1 predictions from ZHANG-SERVER, BAKER-ROSETTA, and QUARK CASP-hosted servers, which can be extracted from the following file:

```
http://predictioncenter.org/download_area/CASP13/server_predictions/T1009.3D.srv.tar.gz
```

We ran three different docking executions with FTDock 2.0 (with each one of the three monomer models against itself), and another three ones with ZDOCK 2.1, to obtain a total of 36,000 docking poses. Each docking pose was checked for possible C_2 symmetry, by applying the following ICM commands:

```
icm> Rmsd(a_a a_b)
```

This command produces the variable `R_out`, which is the rotation matrix that should be applied to move one of the molecules to the position of the other one. Now we can run:

```
icm> Axis(R_out)
```

After executing this command, we obtain two useful variables: `r_out` representing the rotation angle (°) between the two molecules around the rotation axis; and `r_2out` representing the translation (Å) along the rotation axis (values significantly different from 0 might indicate screw axis symmetry). Thus, in a general case, a homodimeric docking pose with `r_out` ~ 360/n and `r_2out` ~ 0 will be compatible with *n*-mer homo-oligomerization following C_n rotational symmetry. Such *n*-mer homo-oligomer can be easily built from the dimeric docking model following symmetry rules.

In particular, we can identify docking poses with C_2 symmetry as those that satisfy 175° < `|r_out|` < 180°, and `r_2out` < 5 Å. In our CASP13 participation, we only used `r_out` value to select symmetric poses due to technical problems. These poses were further selected and evaluated by pyDock scoring, and the best eight models were submitted to CASP13-CAPRI (two additional models based on an available template were included in the set, for the sake of diversity). The official assessment of results showed that pyDock docking yielded two acceptable models (see Figure 4), which were actually the only successful predictions for this target among all participants.

4.3. Template-based docking

Another interesting application of pyDock is the structural modeling of a protein-protein complex based on available templates. If a suitable template is found, the complex can be directly modelled based on the template, or alternatively, the structures (or models) of the unbound proteins can be directly superimposed onto the available template. In case of closely homologous templates, both approaches can provide reasonable docking models. However, when no clearly homologous templates are found, or when the unbound proteins cannot be easily modelled, the challenge is to build suitable models among the possible docking orientations that can be derived from a variety of remote template structures. The use of pyDock scoring can help to identify the correct models.

This strategy has been successfully applied in blind conditions to model target H0974 (CAPRI code T142) in the CASP13-CAPRI experiment. This target consisted in the heterodimeric assembly of two proteins from part of the lysogeny switch of *Lactococcus phage TP901-1*: the repressor CI (72 residues) and the anti-repressor MOR (95 residues). The coordinates of the individual proteins were taken from the models available in the CASP-hosted servers. More specifically, we took the rank #1 predictions from ZHANG-SERVER, BAKER-ROSETTA, and QUARK CASP-hosted servers, which can be extracted from the following files:

```
http://predictioncenter.org/download_area/CASP13/server_predictions/T0974s1.3D.srv.tar.gz
http://predictioncenter.org/download_area/CASP13/server_predictions/T0974s2.3D.srv.tar.gz
```

To build the homodimer from these monomers, we found a total of 9 possible templates from the five CASP-hosted servers used (ZHANG, ROSETTA, QUARK, MULTICOM-CONSTRUCT and RAPTOR Deep Modeller). These templates were actually homo-dimers, but were found to be suitable to build the hetero-dimer, given the structural similarity between the two interacting proteins. Among them, only the three most structurally conserved templates were used for modeling (PDB codes 1Y7Y, 1UTX, and 2B5A). All modelled monomers (3 for each protein) were structurally superimposed on the 3 different templates, in all possible combinations, thus building a total of 27 hetero-dimer models. Models with more than 300 interatomic clashes (i.e. pairs of non-hydrogen atoms from both proteins within 3 Å distance) were removed. The remaining ones were further minimized with AMBER, and then scored with pyDock.

The best two template-based models according to pyDock scoring were submitted to CASP13-CAPRI (the remaining 8 models were built by pyDock; the proportion of template-based and docking models were based on the low reliability of the identified templates). One of these two template-based submitted models, actually the model #1, showed medium accuracy (see Figure 5). This was a challenging target, in which only 12 out of the 29 CAPRI participants were successful (incidentally, in this target, the submitted models built by pyDock alone were not successful).

5. Notes

1. In order to run pyDock in 64-bit Linux systems, we need to update the system regarding compatibility with 32-bit software. For this, in recent Debian-Like distribution type the following commands:

   ```
   sudo dpkg --add-architecture i386
   sudo apt-get update
   sudo apt-get install zlib1g:i386
   ```

2. The FFTW libraries installation file `fftw-2.1.5.tar.gz` can be downloaded from `http://www.fftw.org/download.html` (by clicking in the appropriate link). Uncompressing this file will extract its contents to a new directory called `fftw-2.1.5`. Within this new directory, compile the libraries by:

   ```
   ./configure --enable-float
   make
   ```

   The argument `--enable-float` will make libraries to use single float precision, which implies faster execution. For double precision (slower), remove this argument. This `fftw-2.1.5` directory can be moved to any other location, as long as its location is indicated when installing FTDock (section 2.2.2).

3. For automatic use of FTDock, ZDOCK and SCWRL programs within pyDock, after installing them locally (see Materials section), indicate the full path of the FTDock and ZDOCK directories, and that of the SCWRL binary, by modifying the corresponding lines in the `$PYDOCK/pyDock3/etc/pydock.conf` file, as follows:

   ```
   (...)
   ZDOCK=/<your-installation-directory>/zdock2.1_linux_64bit/
   FTDOCK=/<your-installation-directory>/3D_Dock/
   SCWRL=/<your-installation-directory>/scwrl3_lin/scwrl3
   (...)
   ```

In the `pydock.conf` file, there are also configuration parameters for FTDock. Some default parameters might need to be changed to yield the optimal results (for best pyDock results, it is advisable to use `elec=1` and `calculate_grid=0.7`).

4. The PDB names in the *`DOCKNAME`*`.ini` file must correspond to the exact names of the PDB files you are using (1C5K.pdb, 1c5k.pdb, pdb1c5k.ent.Z, etc...). If the chain ID in a PDB file is empty, use "-" or " " in the `mol` field to select that chain. If a PDB file contains several copies of the same protein, select only the desired chain by indicating its ID in the `mol` field. If a protein to dock contains several chains (for example L and H chains for antibodies) that are relevant for docking, you may indicate their chain IDs in the `mol` field, separated by comma. The `newmol` field can be used to rename a protein chain in the docking output file, assigning it a name different from `mol`, but it can be also left unchanged. The `newmol` chain IDs must be different for the receptor and the ligand molecules (so that their chains can be distinguished in the docking models).

5. In case of using a computer architecture with multi-core CPUs or a cluster, it is advisable to run FTDock in parallel, especially for large-size proteins. For this, we need first to compile the FFTW libraries to enable MPI. Before anything, install the MPI compilers if needed:

```
sudo apt-get install mpi-default-dev
```

Then, download the FFTW libraries installation file `fftw-2.1.5.tar.gz` from `http://www.fftw.org/download.html` (see Note 2). Uncompressing this file will extract its contents to a new directory called `fftw-2.1.5`. Within this new directory, compile the libraries by:

```
./configure --enable-type-prefix --enable-mpi --prefix=/<fullpath>/fftw-2.1.5
make install
```

Now you can download the `ftdock-mpi-master.zip` file with the optimized FTDock distribution for parallel running [10] from the GitHub repository (`https://github.com/brianjimenez/ftdock-mpi`). Unpack this zipped file, and within the new `ftdock-mpi-master` directory, edit the *Makefile* file to set the FFTW_DIR variable to the full path of the above described `fftw-2.1.5` directory. Now, within the `ftdock-mpi-master` directory, type:

```
./make
```

This will create the program binaries, such as `ftdock`. You can find more information in the `README.md` file.

To execute FTDock in parallel within a given pyDock project, you can download the `parallel-master.zip` file with useful scripts from the GitHub repository (`https://github.com/pyDock/parallel`). Unpacking this zipped file within the $PYDOCK directory will create the `parallel-master` folder (alternatively, you can unzip the file from any location and copy the `parallel-master` folder to the $PYDOCK directory). In our example, assuming that we are using a 8-core computer, we can launch FTDOCK in parallel as follows:

```
$PYDOCK/parallel-master/run_parallel_ftdock.sh Dock1 6 noelec
```

The `run_parallel_ftdock.sh` script has three arguments, the name of the pyDock project, the number of CPU threads (in case of multi-core processor, it is advisable to define a number slightly smaller than the total number of cores), and an optional *noelec* parameter to deactivate the electrostatic evaluation during the model generation.

6. Let suppose we have two different `.rot` files from FTDock and ZDOCK (e.g. Dock1.rot_ftdock and Dock1.rot_zdock). We can join both files in one and renumber the conformation numbers as follows:

```
cat Dock1.rot_ftdock Dock1.rot_zdock > tmp.rot

awk '{$13=NR;print $0}' tmp.rot | column -t > Dock1.rot
```

This new `Dock1.rot` file can be scored by pyDock, effectively including docking poses from FTDock and ZDOCK in a single set.

7. In case of using multi-core computer architectures, the scoring module of pyDock can be run in parallel for faster execution times (especially with large-size proteins). For this, as described in Note 5, you will need to download the `parallel-master.zip` file with useful scripts from the GitHub repository (`https://github.com/pyDock/parallel`). Unpacking this zipped file within the $PYDOCK directory will create the `parallel-master` folder.

In our example, assuming that we are using a 8-core computer, we can launch the pyDock *dockser* module in parallel as follows:

```
$PYDOCK/parallel-master/run_dockser_parallel.sh Dock1 6
```

8. The *NIP* (Normalized Interface Propensity) value obtained from the docking represents the frequency of a given residue to be located at the interface among the 100 lowest-energy solutions of docking.

   If *NIP* = 0, the corresponding residue appears at the interface within the top 100 docking poses as expected by a random distribution.

   If *NIP* < 0, the corresponding residue appears at the interface within the top 100 docking poses less than expected by random.

   If *NIP* > 0.2, the corresponding residue is predicted to be at the interface as it appears significantly more often than expected by random.

9. To define a restraint residue in the `restr` field of *`DOCKNAME`*`.ini`, we need to indicate its chain ID, its 3 letter amino-acid code (first letter in uppercase), and its number, as found in the molecule file used in docking. Note that some residues might be named by its AMBER code in the file used in docking (e.g. Hid, Hie, Hip, Cyx), in which case, such notation should be used. When more than one restraint residues are used, they must be separated by comas with no space.

   A distance restraint defined from a potential interface residue is considered fulfilled when the center of coordinates of its side-chain lies within a distance of 6 Å from any non-hydrogen atom of the partner molecule. For each docking solution, the percentage of satisfied restraints is converted to pseudo-energy (just by multiplying by -1.0) and added to the final scoring function in the *`DOCKNAME`*`.eneRST` file.

Acknowledgment

This work was supported by the Spanish Ministry of Science (grant BIO2016-79930-R).

References

1. Huang S-Y (2014) Search strategies and evaluation in protein-protein docking: principles, advances and challenges. Drug Discov Today 19 (8):1081-1096. doi:10.1016/j.drudis.2014.02.005

2. David WR (2008) Recent Progress and Future Directions in Protein-Protein Docking. Current Protein & Peptide Science 9 (1):1-15. doi:10.2174/138920308783565741

3. Cheng TM-K, Blundell TL, Fernandez-Recio J (2007) pyDock: Electrostatics and desolvation for effective scoring of rigid-body protein–protein docking. Proteins: Structure, Function, and Bioinformatics 68 (2):503-515. doi:10.1002/prot.21419

4. Gabb HA, Jackson RM, Sternberg MJE (1997) Modelling protein docking using shape complementarity, electrostatics and biochemical information. Journal of Molecular Biology 272 (1):106-120. doi:10.1006/jmbi.1997.1203

5. Chen R, Weng Z (2003) A novel shape complementarity scoring function for protein-protein docking. Proteins: Structure, Function, and Bioinformatics 51 (3):397-408. doi:10.1002/prot.10334

6. Solernou A, Fernandez-Recio J (2010) Protein docking by Rotation-Based Uniform Sampling (RotBUS) with fast computing of intermolecular contact distance and residue desolvation. BMC bioinformatics 11:352-352. doi:10.1186/1471-2105-11-352

7. Moal IH, Torchala M, Bates PA, Fernández-Recio J (2013) The scoring of poses in protein-protein docking: current capabilities and future directions. BMC Bioinformatics 14 (1):286. doi:10.1186/1471-2105-14-286

8. Barradas-Bautista D, Moal IH, Fernández-Recio J (2017) A systematic analysis of scoring functions in rigid-body protein docking: The delicate balance between the predictive rate improvement and the risk of overtraining. Proteins: Structure, Function, and Bioinformatics 85 (7):1287-1297. doi:10.1002/prot.25289

9. Jiménez-García B, Roel-Touris J, Romero-Durana M, Vidal M, Jiménez-González D, Fernández-Recio J (2017) LightDock: a new multi-scale approach to protein–protein docking. Bioinformatics 34 (1):49-55. doi:10.1093/bioinformatics/btx555

10. Jiménez-García B, Pons C, Fernández-Recio J (2013) pyDockWEB: a web server for rigid-body protein–protein docking using electrostatics and desolvation scoring. Bioinformatics 29 (13):1698-1699. doi:10.1093/bioinformatics/btt262

11. Fernández-Recio J, Totrov M, Abagyan R (2004) Identification of Protein–Protein Interaction Sites from Docking Energy Landscapes. Journal of Molecular Biology 335 (3):843-865. doi:10.1016/j.jmb.2003.10.069

12. Grosdidier S, Fernández-Recio J (2008) Identification of hot-spot residues in protein-protein interactions by computational docking. BMC bioinformatics 9:447-447. doi:10.1186/1471-2105-9-447

13. Pons C, D'Abramo M, Svergun DI, Orozco M, Bernadó P, Fernández-Recio J (2010) Structural Characterization of Protein–Protein Complexes by Integrating Computational Docking with

Small-angle Scattering Data. Journal of Molecular Biology 403 (2):217-230. doi:10.1016/j.jmb.2010.08.029

14. Jiménez-García B, Fernández-Recio J, Pons C, Svergun DI, Bernadó P (2015) pyDockSAXS: protein–protein complex structure by SAXS and computational docking. Nucleic Acids Research 43 (W1):W356-W361. doi:10.1093/nar/gkv368

15. Jiménez-García B BP, Fernández-Recio J (2019) Structural characterization of protein-protein interactions with pyDockSAXS. . Methods Mol Biol (in press)

16. Chelliah V, Blundell TL, Fernández-Recio J (2006) Efficient Restraints for Protein–Protein Docking by Comparison of Observed Amino Acid Substitution Patterns with those Predicted from Local Environment. Journal of Molecular Biology 357 (5):1669-1682. doi:10.1016/j.jmb.2006.01.001

17. Pérez-Cano L, Romero-Durana M, Fernández-Recio J (2017) Structural and energy determinants in protein-RNA docking. Methods 118-119:163-170. doi:10.1016/j.ymeth.2016.11.001

18. Lucas M, Gaspar AH, Pallara C, Rojas AL, Fernández-Recio J, Machner MP, Hierro A (2014) Structural basis for the recruitment and activation of the &lt;em&gt;Legionella&lt;/em&gt; phospholipase VipD by the host GTPase Rab5. Proceedings of the National Academy of Sciences 111 (34):E3514. doi:10.1073/pnas.1405391111

19. Rosell A, Meury M, Álvarez-Marimon E, Costa M, Pérez-Cano L, Zorzano A, Fernández-Recio J, Palacín M, Fotiadis D (2014) Structural bases for the interaction and stabilization of the human amino acid transporter LAT2 with its ancillary protein 4F2hc. Proceedings of the National Academy of Sciences of the United States of America 111 (8):2966-2971. doi:10.1073/pnas.1323779111

20. Case DA, Cheatham III TE, Darden T, Gohlke H, Luo R, Merz Jr. KM, Onufriev A, Simmerling C, Wang B, Woods RJ (2005) The Amber biomolecular simulation programs. Journal of Computational Chemistry 26 (16):1668-1688. doi:10.1002/jcc.20290

21. Bonsor DA, Grishkovskaya I, Dodson EJ, Kleanthous C (2007) Molecular Mimicry Enables Competitive Recruitment by a Natively Disordered Protein. Journal of the American Chemical Society 129 (15):4800-4807. doi:10.1021/ja070153n

22. Méndez R, Leplae R, De Maria L, Wodak SJ (2003) Assessment of blind predictions of protein–protein interactions: Current status of docking methods. Proteins: Structure, Function, and Bioinformatics 52 (1):51-67. doi:10.1002/prot.10393

23. Ray MC, Germon P, Vianney A, Portalier R, Lazzaroni JC (2000) Identification by genetic suppression of Escherichia coli TolB residues important for TolB-Pal interaction. Journal of bacteriology 182 (3):821-824. doi:10.1128/JB.182.3.821-824.2000

24. Svergun DI BC, Koch MHJ (1995) CRYSOL– a Program to Evaluate X-ray Solution Scattering of BiologicalMacromolecules from Atomic Coordinates. Journal of Applied Crystallography 28:768–773

25. Cheng TMK, Blundell TL, Fernandez-Recio J (2008) Structural assembly of two-domain proteins by rigid-body docking. BMC Bioinformatics 9 (1):441. doi:10.1186/1471-2105-9-441

26. Eswar N. ED, Webb B., Shen MY., Sali A. Protein Structure Modeling with MODELLER. Structural Proteomics Methods in Molecular Biology 426: 145-159. doi:10.1007/978-1-60327-058-8_8

Figure legends

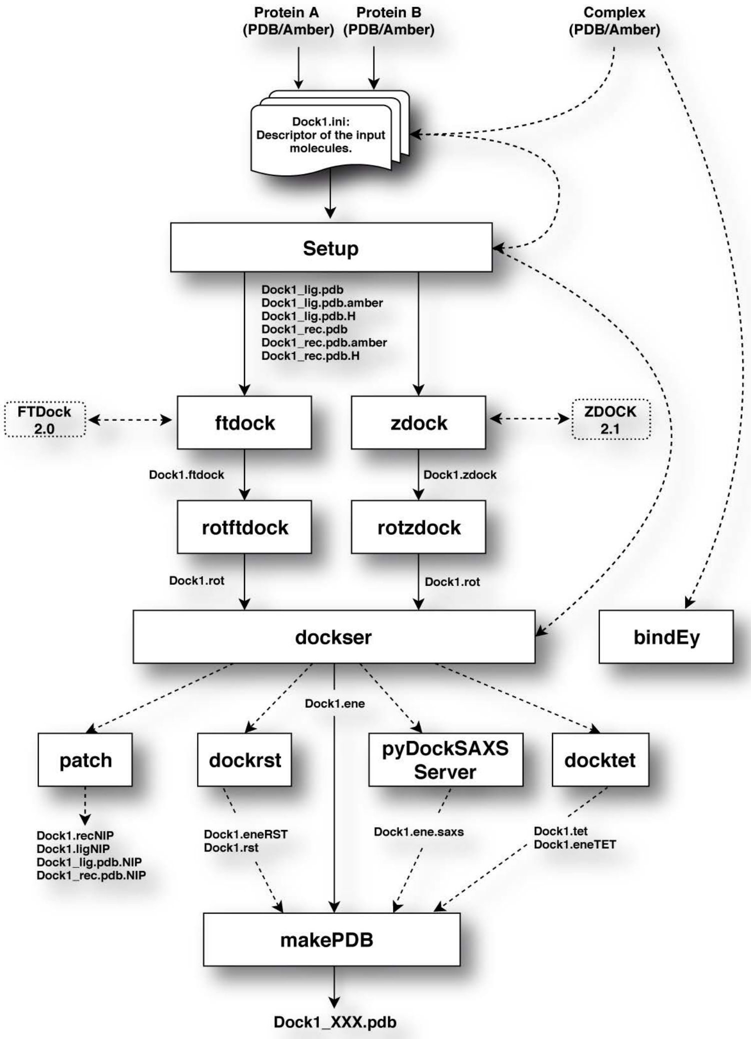
Figure 1. Scheme of pyDock pipeline. The pyDock pipeline with the different pyDock modules are shown.

Figure 2. Docking results for the example case (TolB / Pal interaction). The position of the ligand molecule (in ribbon) in some near-native docking models is shown, after superimposing their receptor molecules (white surface). The best ranked acceptable docking pose (Conf #403; in orange) shows a ligand RMSD 9.3 Å from the x-ray complex structure (PDB 2HQS, in green). This docking model is ranked 61 when scored by pyDock energy alone (*dockser* module), and ranked 6 after including distance restraints with *dockrst* module (restraint residues in cpk). The docking model shown in red (Conf #6201; ligand RMSD 2.9 Å) is ranked 3275 by pyDock alone, and becomes rank 35 after including distance restraints.

Figure 3. Docking model of the 4F2hc and LAT2 complex. Structural model obtained by docking of homology-based models of LAT2 (grey) and 4F2hc x-ray structure (red). The model shown is the best-energy docking pose, after filtering by disulfide bond restraints, and was further confirmed by EM and cross-linking experiments. Residues showing positive cross-linking in wet lab experiments are shown as blue spheres, while residues not showing cross-linking are shown as grey spheres. The distances between these residues in the model are fully consistent with the cross-linking data. The membrane is shown only for visualization purposes.

Figure 4. Models submitted by pyDock for CASP13-CAPRI target T154. The two acceptable models submitted to CASP13 are shown, after superimposing their receptors (white ribbon). The ligand in model #7 is shown in red (8.0 Å ligand RMSD from complex structure), and that of model #9 in orange (12.7 Å ligand RMSD). For comparison purposes, the complex structure (PDB 6DRU) is shown, after superimposing its receptor onto that of the models, with ligand in green ribbon.

Figure 5. Successful model submitted by pyDock for CASP13-CAPRI target T142. Receptor is shown in white, and ligand in red. According to CASP evaluation, the model shows 3.12 Å LocalRMSD, 3.12 Å GlobalRMSD, and 3.20 Å InterfaceRMSD (complex structure not yet released).


Protein A (PDB/Amber)
Protein B (PDB/Amber)
Complex (PDB/Amber)
Dock1.ini: Descriptor of the input molecules.
Setup
Dock1_lig.pdb
Dock1_lig.pdb.amber
Dock1_lig.pdb.H
Dock1_rec.pdb
Dock1_rec.pdb.amber
Dock1_rec.pdb.H
FTDock 2.0
ftdock
zdock
ZDOCK 2.1
Dock1.ftdock
Dock1.zdock
rotftdock
rotzdock
Dock1.rot
Dock1.rot
dockser
bindEy
Dock1.ene
patch
dockrst
pyDockSAXS Server
docktet
Dock1.recNIP
Dock1.ligNIP
Dock1_lig.pdb.NIP
Dock1_rec.pdb.NIP
Dock1.eneRST
Dock1.rst
Dock1.ene.saxs
Dock1.tet
Dock1.eneTET
makePDB
Dock1_XXX.pdb