Dock Design and Enzyme Design Tutorials

Tutorial on Rosetta dock and design compiled by Benjamin Brown (Vanderbilt University, Meiler Lab) for RosettaWorkshop 2018. It is based on (1) the tutorial written by Darwin Fu for the 2013 RosettaWorkshop, and (2) the dock and design protocol capture by Brittany Allison.

Original paper on designing protein-ligand interfaces with RosettaLigand: Allison B, Combs S, DeLuca S, Lemmon G, Mizoue L, Meiler J (2014) Computational Design of Protein-Small Molecule Interfaces. Journal of Structural Biology 185(2):193-202. doi: 10.1016/j.jsb.2013.08.003.

Tutorial on Rosetta enzyme design compiled by Benjamin Brown (Vanderbilt University, Meiler Lab) for RosettaWorkshop 2018. Adapted for RosettaScripts XML format based heavily on (1) the original tutorial written at RosettaCon2011 by Florian Richter (floric at uw dot edu), with help from Patrick Conway, Amanda Loshbaugh, Neil King, and Gert Kiss; (2) the RosettaScripts enzdes protocol capture (available /path/to/rosetta_2017.36.59679/demos/protocol_capture/rosetta_scripts/enzdes) and (3) the RosettaWorkshop 2011 tutorial written by Steven Combs and Dietmar Birzer.

Original tutorial for a complete *de novo* enzyme design run, using the TIM reaction as an example, published in Richter F, Leaver-Fay A, Khare SD, Bjelic S, Baker D (2011) De Novo Enzyme Design Using Rosetta3. PLoS ONE 6(5): e19230. doi:10.1371/journal.pone.0019230

Note - each command line is denoted "~$" prior to the actual command. Do not type the "~$" characters.

This tutorial should be done in the small_molecule_interface_design/ directory. To change into this directory:

```
~$ cd ~/rosetta_workshop/tutorials/small_molecule_interface_design
```

The contents of the demo directory should be:

```
dock_design/
-- parent directory for all files pertaining to the tutorial on Rosetta enzyme design.
Inside of this directory are each of the files and directories indicated below.

       docking/
              input_files/
              -- directory containing apo- and holo-protein .pdb files, ligand .sdf
file, docking flags, and RosettaScripts XML file

              scripts/
              -- python scripts, perl scripts, shell scripts used in the workflow
              -- awk, grep, sed, etc. command lines

              out/
              -- contains additional .pdb and .sc files for practice analysis

              answers/
              -- pre-generated obtained my completing the steps in this tutorial

              design/
              input_files/
              -- directory containing apo- and holo-protein .pdb files, ligand .sdf
file, dock_design flags, resfile, and RosettaScripts XML file

              scripts/
              -- python scripts, perl scripts, shell scripts used in the workflow
              -- awk, grep, sed, etc. command lines
              answers/
              -- pre-generated obtained my completing the steps in this tutorial
 enzdes/
-- parent directory for all files pertaining to the tutorial on Rosetta enzyme design.
Inside of this directory are each of the files and directories indicated below.

       input_files/
       -- directory containing protein scaffold PDB files, ligand PDB file, ligand
parameters file, enzyme design flags, enzyme design constraints files, and
RosettaScripts XML file

       scripts/
       -- python scripts, perl scripts, shell scripts used in the workflow
       -- awk, grep, sed, etc. command lines

       answers/
       -- pre-generated obtained my completing the steps in this tutorial
```

Relevant documentation

1. The above cited PLoS One paper
   http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0019230
2. Documentation for the enzyme design app
   https://www.rosettacommons.org/docs/latest/application_documentation/design/enzyme-design
3. Documentation for the match app
   https://www.rosettacommons.org/docs/latest/application_documentation/design/match
4. Documentation about the enzdes cstfile format used for both matching and enforcing catalytic geometries during design
   https://www.rosettacommons.org/docs/latest/rosetta_basics/file_types/match-cstfile-format
5. Familiarize yourself with how to generate a .params file and rotamer library for your ligand of interest, as described in the ligand docking app documentation.
   https://www.rosettacommons.org/demos/latest/tutorials/prepare_ligand/prepare_ligand_tutorial

Brief Overview

These tutorials cover protein-ligand interface design and enzyme design. These two tasks are related, but distinct in chemically significant ways. The fundamental goal of a protein-ligand interface design experiment is to stabilize binding of a ligand. The goal of an enzyme design experiment is to stabilize the substrate-to-product transition state in a catalytic reaction. In the first tutorial, we work through a typical ligand docking experiment with RosettaLigand, and then we show that interface design can be achieved by allowing protein binding pocket residues to be exchanged during docking. In the second tutorial, we work through the steps necessary to optimize a protein which contains the functional groups necessary to stabilize a theoretical transition state. For both tutorials our example system will consist of an ester ligand and a haloacid dehalogenase-like hydrolase (HAD). At the end, we will compare the changes made to the protein through each design process.

Good luck and have fun!

Ligand Docking Example Problem #1: standard docking study

To effectively perform high-resolution flexible docking with RosettaLigand it is important to have a reasonably good idea of the location of the protein ligand-binding pocket at the start of the experiment (if you are unsure of the location of the binding pocket, software packages such as CASTp can identify good candidate pockets: Joe Dundas et al., Nucleic Acids Res. 2006). The ligand should be manually placed in/near the binding pocket. For this tutorial we have already positioned the ligand in the binding pocket.

1. Inputs required:
    - A .params file for all residues including the ligand and its possible conformers. The Rosetta3 database has .params files for all amino acids, but they will have to be generated for the ligand. To generate the conformers, we will use the Biology and Chemistry Library (BCL). The BCL is available freely with an academic license. It contains a wide variety of tools for cheminformatics, machine learning, and protein structure prediction. After we have generated conformers, we will make a .params file. There are also web servers available for generating ligand conformers.
    - A .pdb file for the apo-protein and separate .pdb files for the primary ligand and its conformers, respectively.
    - A .xml file specifying the ligand docking protocol
    - A .txt file with additional docking options
2. Outputs generated:
    - Two .pdb files corresponding to the ligand and its conformers, respectively, and a .params file for the ligand
    - A Rosetta silent file (Rosetta's version of compressed .pdb structures)
    - A score file
3. Performing docking:
   We have already positioned the ligand in the binding pocket. By default RosettaLigand will take the immediate coordinates of the ligand as a starting position. But, what if you want to dock a handful of ligands? Or a dozen? Or a thousand? You see the point. We do not want to manually place each ligand. Instead, we will tell RosettaLigand to use the centroid coordinates of the ligand we placed in the binding pocket as our starting point for any ligand we want to dock. Open the protein-ligand complex in PyMol, compute the centroid of the ligand, and enter the coordinates in the input_files/ligand_dock.xml file:

Change into the folder for doing standard docking

```
~$ cd dock_design/docking
```

From the terminal type the following:

```
~$ pymol input_files/dock_test_Est_CHba.pdb
```

From the PyMol toolbar type the following:

```
select organic
set_name sele, ligand
centerofmass ligand
quit
```

Edit the input_files/ligand_dock.xml file. Here I have used vi, but you may also use gedit or emacs.

```
~$ gedit input_files/ligand_dock.xml
```

Navigate to the MOVERS section of the .xml file and fill in the centroid coordinates here:

```
<StartFrom name="start_from_X" chain="X">
   <Coordinates x="" y="" z=""/>
</StartFrom>
```

Exit the .xml file and return to the docking directory. Now we will generate conformers and a .params file for our ligand.

```
~$ scripts/bcl-apps-static.exe molecule:Split -input_filenames input_files/D2N.sdf \
    -add_h -implementation @./input_files/sample_conf_dih -output D2N_confs.sdf
~$ cat input_files/D2N.sdf D2N_confs.sdf > D2N_confs_plus_primary.sdf
~$ ~/rosetta_workshop/rosetta/main/source/scripts/python/public/molfile_to_params.py \
    -n D2N -p D2N --conformers-in-one-file D2N_confs_plus_primary.sdf
```

Run the following command to perform the docking study (this should take approximately three minutes):

```
~$ ./scripts/commandline input_files/ligand_dock.xml
```

where the commandline file contains

```
~$ ~/rosetta_workshop/rosetta/main/source/bin/rosetta_scripts.linuxgccrelease \
    @input_files/dock_options.txt -parser:protocol $1
```

The score file contains a complete breakdown of the Rosetta scorefunction for each model. For a quick analysis you can view two score terms in particular:

1. total_score: the total score is reflective of the entire protein-ligand complex and is good as an overall model assessment
2. interface_delta_X: the interface score is the difference between the bound protein-ligand complex and the unbound protein-ligand. Interface score is useful for analyzing ligand effects and for comparing different complexes.

In the out/ directory you will find 100 models pre-generated using the above protocol. In a typical docking experiment you might generate between 500 and 2,000 models per protein/ligand pair depending on the size and complexity of your system. In these cases, you might want to filter the structures first by total_score and subsequently by interface_delta_X. Run the following command to identify the top five best-scoring models. This command identifies the top 20 models by total score and then selects the 5 best models by interface score.

```
~$ cd out
~$ grep -H 'total_score' *.pdb | sort -nk2 | head -n20 | cut -d ':' -f1 | xargs grep
-H interface_delta_X | sort -nk2 | head -n5
```

Now you may visualize the best structures in PyMol.

```
~$ grep -H 'total_score' *.pdb | sort -nk2 | head -n20 | cut -d ':' -f1 | \
    xargs grep -H interface_delta_X | sort -nk2 | head -n5 | \
    awk -F: '{print $1}' > best_models.txt
~$ pymol `cat best_models.txt`
```

Use PyMol to visually compare your highest-scoring model and lowest-scoring model. The "All->Action->preset->ligand sites->cartoon" setting in PyMol is ideal for visualizing interfaces.

Ligand Docking Example Problem #2: docking with interface design

Instead of optimizing the existing binding interface between D2N and the HAD, in this problem we would like Rosetta to predict which binding site mutations would lead to a stronger binding affinity. To do this, Rosetta combines spatial and conformational transformation of the ligand with sequence optimization of the protein. The TaskOperations section of input_files/ligand_dock_design.xml specifies a DetectProteinLigandInterface task. This lets Rosetta know to design residues in proximity to the ligand. It takes as an argument a resfile. In our resfile we allow the selected interface residues to mutate into all other residues except cysteine (ALLAAxc). Other than these minor additions and some additional options, the protocol is essentially equivalent.

All of the files necessary for the dock and design experiment are present. Perform the dock and design experiment by running the following command from the design/ directory (this will take about 5 minutes):

```
~$ cd ~/rosetta_workshop/small_molecule_interface_design/dock_design/design
~$ ./scripts/commandline input_files/ligand_dock_design.xml
```

where the commandline file contains

```
~$ ~/rosetta_workshop/rosetta/main/source/bin/rosetta_scripts.linuxgccrelease \
    @input_files/dock_design_options.txt -parser:protocol $1
```

Analyze your results as discussed in the ligand docking example. Use PyMol to visually inspect differences in binding mode of our ligand as the binding pocket is re-designed.

As before, an out/ directory is provided with 100 structures designed using the above protocol. We can use the same analysis command as in the ligand docking example to evaluate our results.

```
~$ cd out
~$ grep -H 'total_score' *.pdb | sort -nk2 | head -n20 | cut -d ':' -f1 | \
    xargs grep -H interface_delta_X | sort -nk2 | head -n5
```

Now you may visualize the best structures in PyMol.

```
~$ grep -H 'total_score' *.pdb | sort -nk2 | head -n20 | cut -d ':' -f1 | \
    xargs grep -H interface_delta_X | sort -nk2 | head -n5 | \
    awk -F: '{print $1}' > best_models.txt
~$ pymol `cat best_models.txt`
```

A useful way to compare the sequences of the designed models with the wild-type protein is with a multiple sequence alignment. In the out/ directory, you will find a file called "top_5_vs_native.fasta", which contains the sequences of the five models identified in the previous step along with the wild-type sequence. Align them using ClustalW2:

```
~$ clustalw2 -align -infile=top_5_vs_native.fasta -outfile=top_5_vs_native.aln
```

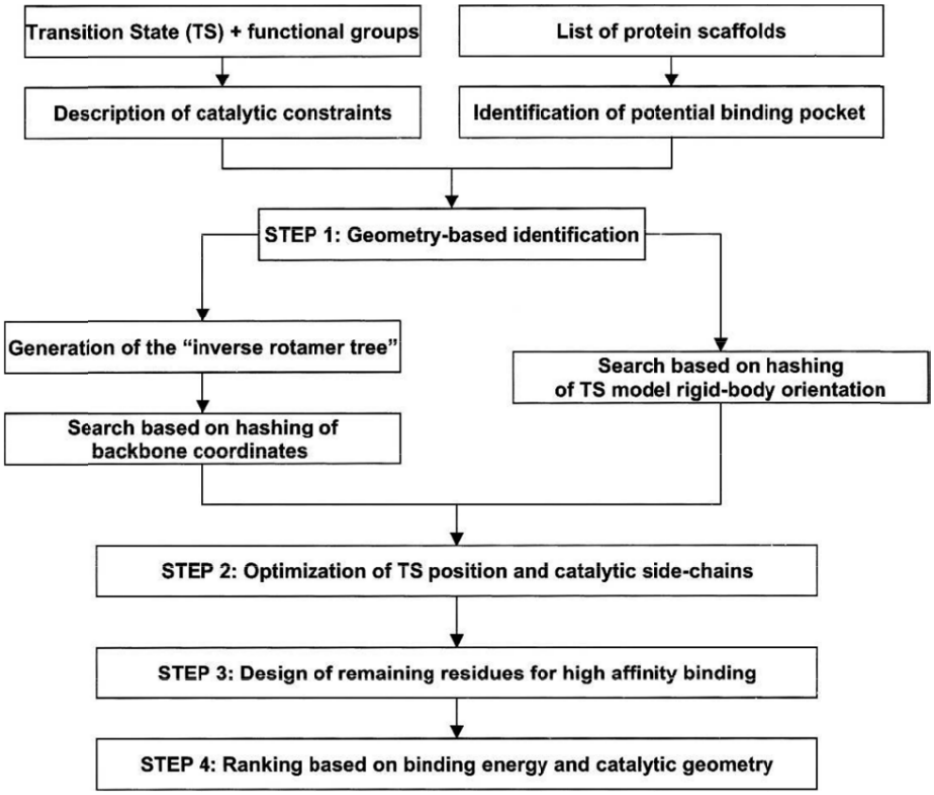
To visualize the most common sequences graphically you can visit http://weblogo.berkeley.edu/logo.cgi and customize the sequence design logo.

Congratulations, you have completed the RosettaLigand docking with design study!

Good job. Take a breather. Now on to enzyme design!

RosettaEnzyme Algorithm

The RosettaEnzyme algorithm is discussed in great detail in Zanghellini et al. **Protein Science** (2006), 15:2785--2794.



In addition to understanding the algorithm, you must have the following:

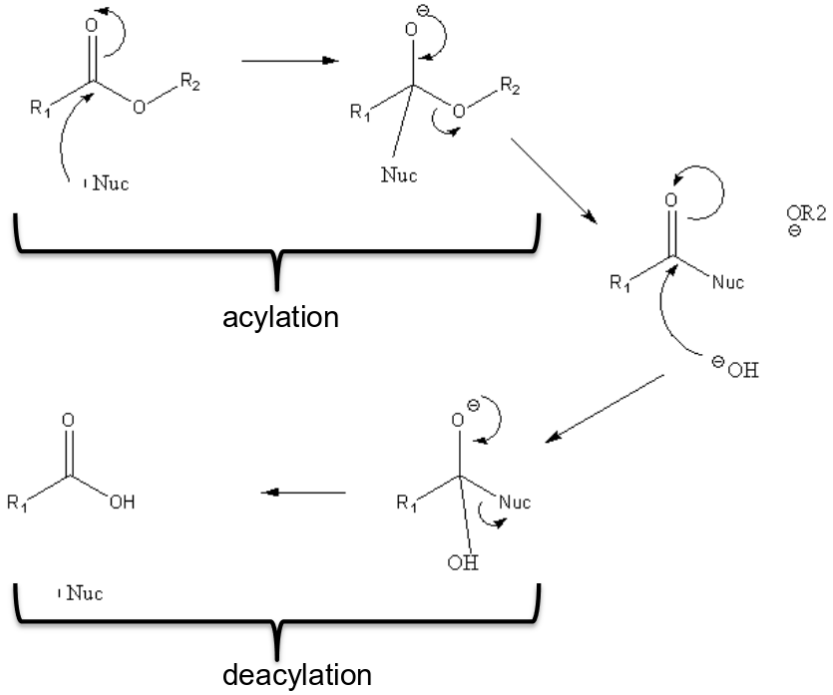
1. An enzyme reaction
2. Knowledge of the mechanism for the enzyme reaction
3. Known intermediates for the enzyme reaction
4. A protein scaffold (RosettaMatch can find one for you, but this is beyond the scope of the present tutorial)

These will be covered in the following sections.

Theozymes: Theoretical enzymes used for computational enzyme designs

Theozymes are theoretical enzymes that have an ideal arrangement of the transition state structure surrounded by catalytic functional groups. In general, a theozyme is created computationally via quantum mechanical studies. In this process, a transition state (TS) is modeled and geometrically optimized. Once the transition state is geometrically optimized, catalytic functional groups are added to stabilize the TS. The actual amino acid side chain is not used, instead, ammonium is used for lysine, acetamide for aspargine, and so on. The placement of the functional groups are done through approximation of ideal distances/angles. Then, quantum mechanical studies are used to find the optimal geometric constraints. A review of such techniques can be found in the paper: **Tantillo Current Opinion in Chemical Biology 1998, 2:743750**. A set of theozymes was created in this paper: **Zhang ACS 2008**. Both are from Kent Houk's group. Quantum mechanical studies done in the paper are done by Gaussian. This tutorial will not cover how to use Gaussian.

The Baker group uses idealized bonds/angles for their theozymes. For our purposes, we will follow the Baker group's strategy. In particular, this example will use the ester hydrolase reaction previously made by Florian Richter in the Baker lab and presented at RosettaCon12 (http://www.rosettadesigngroup.com/workshops/RCW2012/presentations/roscon12_talk_richter.pdf). After defining a theozyme, we will re-design a protein scaffold to optimally accommodate the theozyme. First, we need a reaction mechanism for the theozyme.



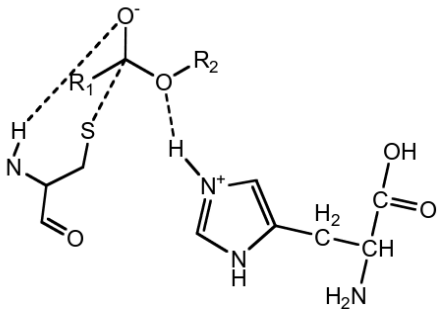
There are several key features of this reaction which we will need to consider in designing the theozyme.

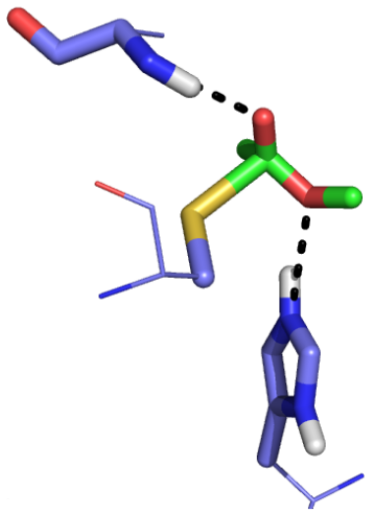
1. A nucleophilic attack occurs at the ester-carbon. Natural hydrolases often use Ser or Cys as the nucleophile. Here, we will use a Cys nucleophile because of its higher intrinsic nucleophilicity.

2. The nucleophilic attack will lead to a tetrahedral intermediate ('acylenzyme intermediate'). We will need to stabilize the negative charge accumulation of the oxyanion formed at the ester-oxygen. We will use a backbone amide to accomplish this stabilization.

3. The leaving group needs to be protonated, and water will need to be deprotonated before we have a hydroxy anion to replace the nucleophile. We will use His as our proton shuttle.

Our theozyme will have therefore have the following approximate structure:





Tutorial

Step 1: Defining a theozyme in Rosetta format

1. Inputs required:
    - A .params file for all residues including the ligand and its possible conformers. The Rosetta3 database has .params files for all

amino acids, but they will have to be generated for the ligand/reaction substrate. We already generated conformers and a .params file for our ligand in the previous tutorial. You will find conformers and a .params file for D2N in the input_files directory.

2. Outputs generated:
    - A .cst (constraints) file. This file is written by the user and defines the geometry of the theozyme in Rosetta format (see enzdes cstfile documentation linked above) for use in subsequent steps of the design process. For this tutorial two unique constraint files have been provided for two unique enzyme design simulations.
3. Defining a theozyme:
    - A theozyme is not defined using Rosetta, but usually in one of the following ways: (1) through quantum mechanical calculations (see references of Ken Houk's work in the PLoS ONE paper), (2) from chemical intuition, or (3) by stealing or using as inspiration a naturally occurring enzyme's active site.

Now that we have chosen catalytic residues for our enzyme, we need to convert this information into constraints that Rosetta understands (e.g. part 2 - generate a constraints file). Ester hydrolysis is one of the most well-studied chemical reactions in biochemistry, and we have a good idea of what may be needed to stabilize the TS. To translate this knowledge to Rosetta, we use constraints to fix theozyme residues in a specific location so they create a TS-stabilizing geometry. The constraint file which will be used for our first example is below.

```
# cst constraint descriptor for esterase active site featuring a
# a cys/his catalytic diad and one backbone oxyanion hole, plus aromatic binding
# F.Richter, Baker lab, UW; floric@u.washington.edu

#block 1 for cysteine interacting with substrate
CST::BEGIN
TEMPLATE:: ATOM_MAP: 1 atom_name: C6 O4 O2
TEMPLATE:: ATOM_MAP: 1 residue3: D2N

TEMPLATE:: ATOM_MAP: 2 atom_type: SH1 ,
TEMPLATE:: ATOM_MAP: 2 residue1: C

CONSTRAINT:: distanceAB: 2.00 0.30 180.00 1
CONSTRAINT:: angle_A: 105.10 4.00 100.00 360.00
CONSTRAINT:: angle_B: 112.00 7.00 100.00 360.00
CONSTRAINT:: torsion_A: 105.00 10.00 50.00 360.00
CONSTRAINT:: torsion_B: 180.00 10.00 0.00 360.00
CONSTRAINT:: torsion_AB: 50.00 40.00 5.00 360.00
CST::END

#block 2 for histidine interacting with substrate
CST::BEGIN
TEMPLATE:: ATOM_MAP: 1 atom_name: O2 C6 O4
TEMPLATE:: ATOM_MAP: 1 residue3: D2N

TEMPLATE:: ATOM_MAP: 2 atom_type: Nhis,
TEMPLATE:: ATOM_MAP: 2 residue1: H

CONSTRAINT:: distanceAB: 3.10 0.20 100.00 0
CONSTRAINT:: angle_A: 120.00 5.00 30.00 360.00
CONSTRAINT:: angle_B: 125.90 10.00 20.00 360.00
CONSTRAINT:: torsion_A: -5.00 15.00 0.00 360.00
CONSTRAINT:: torsion_B: -155.0 15.00 25.00 360.00
CONSTRAINT:: torsion_AB: 0.00 0.00 0.00 180.00
CST::END

#block 3 for bb-H oxyanion hole 1
CST::BEGIN
TEMPLATE:: ATOM_MAP: 1 atom_name: O4 C6 O2
TEMPLATE:: ATOM_MAP: 1 residue3: D2N

TEMPLATE:: ATOM_MAP: 2 atom_type: Nbb ,
TEMPLATE:: ATOM_MAP: 2 is_backbone
TEMPLATE:: ATOM_MAP: 2 residue1: AGSITN
```

```
CONSTRAINT:: distanceAB: 3.00 0.20 20.00 0
CONSTRAINT:: angle_A: 119.10 20.00 5.00 360.00
CONSTRAINT:: angle_B: 120.10 15.00 20.00 360.00
CONSTRAINT:: torsion_AB: 76.50 180.00 0.00 360.00
CST::END

#block 4: accessory for enforcing the cys-his interaction
CST::BEGIN
TEMPLATE:: ATOM_MAP: 1 atom_type: SH1 ,
TEMPLATE:: ATOM_MAP: 1 identical: 1 2
TEMPLATE:: ATOM_MAP: 1 residue3: CYS

TEMPLATE:: ATOM_MAP: 2 atom_type: Nhis,
TEMPLATE:: ATOM_MAP: 2 identical: 2 2
TEMPLATE:: ATOM_MAP: 2 residue1: H

CONSTRAINT:: distanceAB: 3.30 0.40 20.00 0
CONSTRAINT:: angle_A: 130.00 35.00 0.00 360.00
CONSTRAINT:: angle_B: 130.10 35.00 15.00 360.00
CONSTRAINT:: torsion_B: 145.50 15.00 15.00 360.00
CST::END

#block 5 for aromatic binding
CST::BEGIN
TEMPLATE:: ATOM_MAP: 1 atom_name: X1 C10 C12
TEMPLATE:: ATOM_MAP: 1 residue3: D2N

TEMPLATE:: ATOM_MAP: 2 atom_type: aroC,
TEMPLATE:: ATOM_MAP: 2 residue1: WFY

CONSTRAINT:: distanceAB: 3.50 0.20 50.00 0
CONSTRAINT:: angle_A: 90.00 5.00 50.00 360.00
CONSTRAINT:: angle_B: 90.00 5.00 50.00 360.00
CONSTRAINT:: torsion_A: 90.00 5.00 50.00 180.00
CONSTRAINT:: torsion_B: 180.00 15.00 0.00 120.00
```

```
CONSTRAINT:: torsion_AB: 0.00 0.00 0.00 180.00
CST::END
```

Overall organization of the constraints
Comments are seen in the constraints file via the # mark. This format is broken up into five blocks to form a cys/his catalytic diad, one backbone oxyanion hole, and aromatic binding.These blocks are defined by the CST::BEGIN and CST::END and everything that is contained within the notations that are indented. The residues that are being constrained are followed by TEMPLATE designation. Below the TEMPLATE designation is the CONSTRAINT notation. Everything labeled CONSTRAINT are constraints that are placed upon your residues defined in TEMPLATE. Lets delve a little bit more into how to define residues in TEMPLATE.

TEMPLATE and ATOM_MAP
The template tag defines the residue to constrain, either protein side or ligand, and what atoms are used to reference the constraint. ATOM_MAP identifies a residue with either 1 or 2. Remember, you are constraining two residues with respect to one another. Residue one is identified by the ATOM_MAP: 1 flag and residue two is identified by the ATOM_MAP: 2 flag. The type of residue to be constrained is identified by the residue3 or residue1 flag. The simplest case to use is the residue1 flag. residue1 is followed by the one letter amino acid code. If the identity of the residue in this constraint is ambiguous/irrelevant, you can indicate multiple amino acids with consecutive one letter codes. For example, in block 5 residue 2 we don't care about the identity of the amino acid in this position so long as it can make a pi-stacking interaction. Therefore, we use the residue1: WFY notation to denote tryptophan, phenylalanine, and tyrosine. Note that if you use this notation you will have to use the atom_type flag to identify the residues. This is discussed further below. The residue3 notation will always be used to identify the ligand as the ligand will always have a three letter identifier to it. In block 1 residue 1 we use the residue3: D2N to denote the ligand.

The type of atoms to be constrained can be identified by either the atom_name or atom_type flag. It is important to understand that both flags require 3 atoms to be identified. This is because we want to add angle constraints, which require three atoms. Specifically in block 1 for residue1, if you use the atom_name flag, the first atom name is C6, the second O4, and the third O2. Alternatively, you can use the atom_type flag which allows a more flexible definition of the constrained atoms.

The atom_type flag has to be followed by the Rosetta atom type of the first constrained atom of the residue. In case this tag is used, Rosetta will set the 2nd constrained atom as the base atom of the first constrained atom and the third constrained atom as the base atom of the 2ndconstrained atom. (Note: the base atoms for each atom are defined in the ICOOR records of the .params file for that residue type). There are two advantages to using the 'atom_type' tag: First, it allows constraining different residue types with the same file. For example, if a catalytic hydrogen bond is to be constrained, but the user doesn't care if it's mediated by a SEROH or a THROH. Second, if a catalytic residue contains more than one atom of the same type (as in the case of ASP or GLU), but it doesn't matter which of these atoms mediates the constrained interaction, using this tag will cause Rosetta to evaluate the constraint for all of these atoms separately and pick the one with lowest score, i.e.the ambiguity of the constraint will be automatically resolved.

CONSTRAINT
The constraint records identify the distance and angles of the residues' atoms to be used. This is the actual value and strength of the constraint applied between the two residues. A quick glance shows that the line begins with CONSTRAINT:: followed by a designation of A or B whether it is a distance or an angle. The designation of A or B refers to residue 1 for A and residue 2 for B.

Types of constraints
distanceAB:
The distanceAB constraint refers to a distance constraint between Res1:Atom1 and Res2:Atom2. After the distanceAB comes four numbers. For example, in the first block distanceAB is followed by four numbers: 2.00 0.30 180.00 1. The first number 2.00 specifies the optimum distance between residue1, atom1 and residue 2, atom 1. Tolerance for the given distance (our example 2.00) is defined in the second column. The number 0.30 identifies the "distance" that we can move from our 2.00 goal. During the design process that means we can move 1.70 or 2.30 away from the optimal distance. The third number, 180.00 is the "force" of our constraint. This is a measure of how often you allow breaking of your constraint. In Rosetta, this translates to the magnitude of the energy penalty associated with a model in which this constraint is broken. This behavior is modeled with a piecewise function. If x is the actual distance (sampled during enzdes run), x0 the optimal distance (defined in constraints) and k = value in 3rd column (force constant, parameter for strength of the constraint) then the score penalty applied will be: 0 if |x x0| <xtol and k * (|x x0| xtol)^2 else. This is, of course, an optimization problem.You will probably have to play around with this number to get it to work. The last number can either be 1 or 0, where 1 means covalent interaction and 0 means noncovalent interaction for the interacting atoms. Note that in this example the last column is 1. In block 2 the distanceAB constraint has a third column (force constant) value of 100.00 and a fourth column value of 0. This indicates that models which break the covalent bond from block 1 will suffer a greater energy penalty than models which break the non-covalent interaction in block 2.

angle_A & angle_B:
The second constraint to be defined is the set of angle constraints. The angle constraints are defined by the angle_A or angle_B constraint. angle_A refers to the angle between Res1:Atom2 Res1:Atom1 Res2:Atom1. angle_b refers to the angleRes1:Atom1 Res2:Atom1 Res2:Atom2. Similar to distanceAB, there are four numbers after the angle_A, angle_B flag of block 1: 105.10 4.00 100.00 360.00 (angle_A numbers shown for simplicity). The first number is the optimal angle, which is 105.10 in this case.The second number is allowed deviation (4.00 degrees). The third number is the force constant as defined above.The fourth numbers adds periodicity to your degrees. For example: If x0 (optimal distance) is 120 and periodicity is 360, the constraint function will have a single minimum at 120 degrees. If x0 is 120 and periodicity is 180, the constraint function will have two minima, one at 120 degrees and one at 300 degrees. If x0 is 120 and per is 120, the constraint function will have 3 minima at 120, 240, and 360 degrees.

torsion_A, torsion_AB, torsion_B:
The final set of constraint terms refer to a set of dihedrals. torsion_A is the dihedral Res1:Atom3 - Res1:Atom2 - Res1:Atom1 - Res2:Atom1. torsion_AB is the dihedral Res1:Atom2 - Res1:Atom1 - Res2:Atom1 - Res2:Atom2. torsion_B is the dihedral Res1:Atom1 - Res2:Atom1 - Res2:Atom2 - Res2:Atom3. The four columns have equivalent meanings as those in the angle constraints.

Scaffold PDB file preparation
There is an additional step that is needed to finish everything for the constraints file. For each constraint a REMARK needs to be included in the

.pdb file of the scaffold protein to be used by your constraints file. We edited the one in the tutorial for you. The input .pdb file is edited to contain the following line (at the top):

```
REMARK 0 BONE TEMPLATE X D2N 900 MATCH MOTIF A CYS 10 1
```

From the above line,the REMARK line indicates that this is the start of the constraint information. Your pdb must be cleaned prior to this process with all REMARK lines removed except for this line (clean the .pdb file with $ROSETTA/tools/protein_tools/scripts/clean_pdb.py). The TEMPLATE X D2N 900 words are used to describe the ligand. The word TEMPLATE tells Rosetta to start taking information. The X D2N 900 has to match the chain id, ligand name,and residue number of the ligand. The A CYS 10 1 refers to the catalytic residue. The A CYS 10 is the chain id, residue name, and residue number in the .pdb file for the catalytic residue. The last column 1 refers to the first block of catalytic constraints. Remember, in our example above we had five blocks of catalytic constraints. You can have multiple REMARK lines that refer to all the blocks in your constraints file. Declaring the constraints and the exact residues to constrain in two different allows you to use the same .cstfile for any number of matches that have the same type of active site but are in different scaffolds or at different attachment points in the same scaffold.

Step 2: Matching (i.e. identifying suitable sites for the active site theozyme in a library of scaffold proteins)

This step is beyond the scope of this workshop - we will provide a suitable scaffold protein

1. Inputs required:
    - The .params file from Step 1.
    - The .cst file from Step 1.
    - A library of scaffold protein structures in .pdb format. The scaffold library should be as big as possible. Refer to the matcher documentation linked above for instruction on how to prepare a scaffold for matching (which is mainly deciding where the binding site is and what positions will be considered for theozyme residue attachment).A scaffold position file for each scaffold that defines which residues in the scaffold structure will be considered for theozyme residue attachment. The format of scaffold position files, and instructions on how to generate them, can be found in the matcher documentation.
2. Outputs generated:
    - A .pdb file for each “match”. Each match file contains the three-dimensional coordinates of both the scaffold protein and the theozyme, including the ligand, and will be used as an input in step. The number of matches found in the scaffold depends on the complexity of the theozyme, and be anywhere between 0 and hundreds.
3. Performing matching:
    - Example command line

```
~$ ~/rosetta_workshop/rosetta/main/source/bin/match.linuxgccrelease \
    @rosetta_inputs/general_match.flags @rosetta_inputs/1tml_sys.flags
```

Step 3: Design

1. Inputs required
    - The .params file from step 1.
    - The .cst file from step 1.
    - Protein scaffold in .pdb format.
2. Outputs generated
    - Designed 'enzymes' in .pdb format.
    - A score file that contains scoring information for each designed enzyme, which will be used in step 4 to evaluate and rank the designs.
3. Performing design
    - Usually, every match from step 2 is designed several times (~100, depending on computational resources). For a detailed explanation of what the enzyme design application does (and additional user-options), refer to the documentation/paper linked above. Briefly, all residues that are within a sphere of a defined radius (usually 8A) of the ligand (except theozyme residues) are identified and considered mutatable in design. These residues are first mutated to alanine, and the resulting structure is minimized with respect to the theozyme geometry as specified in the .cst file. Then, 2-3 cycles of constrained sequence design and minimization are carried out, and the final sequence is subjected to an unconstrained repack and minimization.

In this tutorial, we will run this stage for one of the matches (input_files/enzdes_test_Est_CHba.pdb) via RosettaScripts with the following command line:

```
~$ ~/rosetta_workshop/rosetta/main/source/bin/rosetta_scripts.linuxgccrelease \
    @input_files/enzdes_flags -nstruct 10 -out::overwrite \
    -parser:protocol input_files/enzdes.xml \
    -in:file:s input_files/enzdes_test_Est_CHba.pdb
```

This command generates (1) 10 designed proteins in .pdb format, and (2) a score file that has one line of score term values and other metrics per protein model. This should take about 3 minutes

You may notice that in your input_files directory there is another protein scaffold and constraint file. In the enzdes_flags file there are also lines commented out which correspond to the alternate scaffold. Investigate the second constraint file and protein scaffold. How are they different from the original? If you would like, switch the # symbols in the enzdes_flags file from one .pdb/.cst to the other, and run the enzdes application again.

Step 4: Evaluating and ranking designs

The purpose of this stage is to reduce the number of candidate designs to a number tractable for visual inspection, and to get rid of designs which have obvious defects. In short, one is looking for designs which have (1) good catalytic geometry, (2) a good predicted protein-ligand binding affinity, (3) a preformed/preorganized active site, and (4) a well-behaved protein (expressible, soluble, stably folded, etc). For each of these 4 criteria, there are terms in the scorefile (refer to documentation linked above for details). Generally, these considerations are accounted for in the following ways: For (1) and (2), the constraint score and protein-ligand interface score are taken as metrics, respectively. For (3), the designed site is repacked without the ligand during the design calculation. Finally, for (4), the designed protein is compared to its derived scaffold at the end of the design calculation. The script DesignSelect.pl, which is part of the Rosetta3 distribution, can read the output scorefile, as well as a file that specifies required values for certain columns, and will then output only those designs in the scorefile that satisfy all required values.

In the dock and design tutorial we performed a sequence alignment and comparison between all the designed sequences. Using the code below, perform a sequence alignment of the sequences produced during enzyme design with ClustalW2:

Obtain the unique identifier strings of each designed PDB:

```
~$ ls enzdes_test_Est_CHba_00*.pdb | awk -F_ '{print $5}' | \
    sed 's/....$//g' > output_file_tags
```

Use the clean_pdb.py script to generate fasta files for each protein:

```
~$ cat output_file_tags | xargs -n1 -P1 -I% \
    bash -c "~/rosetta_workshop/rosetta/tools/protein_tools/scripts/clean_pdb.py \
    --nopdbout enzdes_test_Est_CHba_%.pdb A"
```

Combine the fasta files:

```
~$ cat *.fasta > designed_sequences.fasta
```

Perform the sequence alignment

```
~$ clustalw2 -align -infile=designed_sequences.fasta -outfile=designed_sequences.aln
~$ cat designed_sequences.aln
```

Pick your favorite models from dock/design and enzyme design and open them both up in PyMol. Compare the sequence alignments between the dock/design protocol and the enzyme design protocol for enzdes_test_Est_CHba.pdb. How do the designed sequences differ? Did the dock and design protocol find another purpose for the residues to which we applied constraints in enzyme design (e.g. is the cysteine still a cysteine, or is the histidine still a histidine)? Did the constrained catalytic site geometry influence residue selection?

Step 5: You are done

Congratulations! You have completed the enzyme design tutorial for Rosetta.