

# Antibody single-state design

**Bold text means that these files and/or this information is provided.**

*Italicized text means that this material will NOT be conducted during the workshop*

fixed width text means you should type the command into your terminal

If you want to try making files that already exist (e.g., input files), write them to a different directory! (mkdir my\_dir)

## Tutorial

This tutorial is an example of an antibody single-state design experiment. The goal of this experiment is to take a previously crystallized antibody-antigen complex and optimize the antibody sequence for increased affinity for its target. Antibody single-state design is also referred to as affinity maturation, sequence optimization, or simply design throughout the protocol - these terms can be used interchangeably. However, it should be noted that this is a separate protocol than de novo antibody design, which is described in a separate tutorial. The difference between these protocols is that this protocol optimizes an existing antibody-antigen complex, whereas de novo antibody design creates a new antibody from scratch for a target antigen. If successful, this protocol will result in an optimized antibody sequence with increased affinity for its target antigen.

In this tutorial we are going to use the co-crystal structure of anti-influenza antibody CH67 against influenza hemagglutinin (HA) H1 SolomonIslands/03/2006.

1. Create a directory in the `single_state_design` directory called `my_files` and switch to that directory. Although many files you need for the tutorial are located in the `input_files` directory, we will work from `my_files` for the rest of the tutorial.

```
mkdir my_files
cd my_files
```

2. Prepare the input complex for design.

1. Download the co-complex from the Protein Databank (PDB). This complex is under the PDB ID `4hxx`. **The 4HKX.pdb file is provided in the input\_files directory.** However the instructions for downloading this PDB file are also provided below.

1. Go to [rcsb.org](http://rcsb.org) and type '4hxx' in the search bar.
2. Click on 'Download Files' on the right side of the page, then 'PDB Format'.
3. Save the PDB file in the `my_files` directory as '4HKX.pdb'.

2. Prepare the PDBs for running through Rosetta. In general before running a PDB through Rosetta you should remove water molecules and all ligands that are non-essential to your protocol. We will use an automated script to do this processing.

1. We want to pull hemagglutinin (chain E) and the antibody chains (chain A+B) from the PDB 4HKX.

```
python2.7 ~/rosetta_workshop/rosetta/tools/protein_tools/scripts/clean_pdb.py 4HKX ABE
```

2. As an extra processing step we will remove any protein atoms that are not involved in the antibody-antigen interface. This will make the protocol run faster without any negative impact on the results. In this case we will delete the constant domain of the antibody on both the heavy and light chains. We will manually edit the PDB file in PyMOL to remove these atoms. Enter the following commands in the PyMOL command prompt.

```
pymol 4HKX_ABE.pdb
as cartoon
```

```
select heavy_constant, resi 339-438 and chain A
select light_constant, resi 537-639 and chain B
```

Look at the 4HKX complex in PyMOL and notice the heavy and light chain constant domains that were selected with the previous command. Notice how these domains are very far from the antibody-antigen interface. These can be removed to make the design protocol run more quickly.

```
remove heavy_constant
remove light_constant
save 4HKX_trim.pdb, 4HKX_ABE
```

3. Next we will rename and reorder the chains in this complex. As a general convention it's good to name antibody chains H and L (heavy and light) and antigen chain A. This script will reorder the chains in our PDB, rename them to H,L,A, and renumber them starting from residue number 1.

```
python2.7 ../../scripts/reorder_pdb_chains.py --new_chain_order A,B,E \
--new_chain_ids H,L,A --norestart 4HKX_trim.pdb 4HKX_renum.pdb
```

3. Prepare a residue file (resfile) for design.

When designing a protein we need some way to tell Rosetta which residues should be designed, which should be repacked with no design, and which should be ignored. The residue file, known as a resfile, serves this purpose. Full documentation of the resfile format can be found at [https://www.rosettacommons.org/docs/latest/rosetta\\_basics/file\\_types/resfiles](https://www.rosettacommons.org/docs/latest/rosetta_basics/file_types/resfiles).

In this tutorial, we will use script `define_interface.py` to define which residues are at the antibody-antigen interface. Interface residues on the antibody will be redesigned, and those on the antigen side will be repacked. This algorithm defines interface residues as those with a heavy atom within 5 Å of a heavy atom on a residue on the opposing side of the interface.

1. Run `define_interface.py` to generate a resfile with designable and repackable residues.

```
python2.7 ../../scripts/define_interface.py --side1 HL --side2 A --design-side 1 \
--repack --output 4HKX 4HKX_renum.pdb
```

4. Repack or relax the template structure.

Rosetta protocols often work better on a structure that has been processed in some way after downloading from the PDB. Protein structures frequently have small clashes between side chains that are easily resolved by letting Rosetta optimize side chain conformations (known as repacking) or minimizing backbone phi-psi angles to relieve such clashes (known as relaxing). These problems are exaggerated when working with a low-resolution structure, when all side chain atoms may not be easy to place from the electron density.

In this tutorial we will relax our input complex while restraining the atoms to their starting positions. This allows Rosetta to relieve clashes while preventing the structure from moving too far from what was experimentally determined. More information on the relax protocol is available at [https://www.rosettacommons.org/docs/latest/application\\_documentation/structure\\_prediction/relax](https://www.rosettacommons.org/docs/latest/application_documentation/structure_prediction/relax). **The relax options file and relax command file are provided in the `input_files` directory.**

1. Copy the options file and command line for running relax from the `input_files` directory.

```
cp ../input_files/relax.options .
cp ../input_files/relax.command .
```

2. Run relax with constraints on the 4hkx complex.

```
~/rosetta_workshop/rosetta/main/source/bin/relax.default.linuxgccrelease \  
@relax.options -s 4HKX_renum.pdb > relax.out &
```

3. The relaxed model will take some time to run (~90 min) - you can move on to the next step using pre-generated models. Copy the best scoring model to 4HKX\_relax.pdb. In this case we only make a single model, but in general it's recommended to make a larger number of models (~10-50). **In this case there are pre-generated relaxed models located in the output\_files directory.**

The lowest energy relaxed structure is 4HKX\_renum\_0009.pdb. Copy this model to your current directory.

```
cp ../output_files/4HKX_renum_0009.pdb 4HKX_relax.pdb
```

5. Design the antibody in our relaxed structure.

At this point all of our input files are ready and we can run design. We will run design through a RosettaScripts XML file - this allows more flexibility in creating a design protocol. In this tutorial the design protocol will use a single round of fixed backbone design.

Generally in protein design it is useful to use iterations of design and backbone motion. *An example XML of design iterated with backrub motions is provided in the input\_files directory, but will not be used in this tutorial.* Backrub motions are small rotations of the backbone designed to mimic protein flexibility in solution. Small perturbations of the antibody backbone can provide more backbone diversity among our models to improve sequence diversity. Backrub motion in design is recommended for production runs.

1. Copy design.xml and design.options from the input\_files directory.

```
cp ../input_files/design.xml .  
cp ../input_files/design.options .  
cp ../input_files/design.command .
```

2. Read through the XML and options files, and familiarize yourself with what different steps of the protocol are doing.
3. Generate ten designed models. These models will finish shortly (~1 minute per design).

```
~/rosetta_workshop/rosetta/main/source/bin/rosetta_scripts.default.linuxgccrelease \  
@design.options -parser:protocol design.xml -out:suffix _design \  
-scorefile design.fasc
```

4. As a control we will repeat the same protocol without designing any residues. This is necessary because in our analysis we will want to compare the score and binding energy of designed models to the native sequence, and this comparison is only valid if our native sequence models are subjected to the same level of optimization as the designed models. Copy the XML, resfile and command line for the design control to the current directory. The XML protocol is identical except for which resfile is being used. 4HKX\_control.resfile is the same as the previously used resfile, except the designed residues are changed from ALLAA (design) to NATAA (repack).

```
cp ../input_files/4HKX_control.resfile .  
cp ../input_files/design_control.xml .  
cp ../input_files/design_control.command .
```

5. Generate ten control models.

```
~/rosetta_workshop/rosetta/main/source/bin/rosetta_scripts.default.linuxgccrelease \  
@design.options -parser:protocol design_control.xml -out:suffix _control \  
-scorefile control.fasc
```

6. While you are waiting for design and control models to finish you can move on to the next step with the pre-generated results in the `output_files` directory. Make a new directory and copy the files `design.fasc` and `control.fasc` from the `output_files` directory into this new directory.

6. Analyze the designed sequences.

To analyze the designed sequences it is useful to look at the score, binding energy, and binding density of the models. In a successful design run these metrics should be significantly lower for the designed models than the control models. We will pull these values from the score file and plot them side by side.

1. Plot the score and binding energy of designed models against control models. The script `compare_design_to_control.py` will take in the score files of both your design and control models and will make a plot of score and binding energy.

```
python2.7 ../../scripts/compare_design_to_control.py control.fasc design.fasc
```

```
gthumb *png &
```

2. In this case the designs have improved stability and binding affinity compared to our native sequence. The next step is to look at what mutations specifically were made that result in this improvement. We will make a sequence logo from our models that shows which mutations were made and how frequent they were. Use the `design_analysis.py` script to make a sequence logo from our designed models.

```
python2.7 ../../scripts/design_analysis.py --prefix design --res 4HKX.resfile \  
--native 4HKX_relax.pdb *design*.pdb
```

```
gthumb design_seq_log.png &
```

3. Open the lowest scoring control and design models in PyMOL and look at the amino acids introduced by design and how they interact with the antigen.

```
sort -nk2 design.fasc | head -1
```

```
sort -nk2 control.fasc | head -1
```

```
pymol 4HKX_relax_control_0001.pdb 4HKX_relax_design_0001.pdb
```

# Antibody multistate design

**Bold text means that these files and/or this information is provided.**

*Italicized text means that this material will NOT be conducted during the workshop*

fixed width text means you should type the command into your terminal

If you want to try making files that already exist (e.g., input files), write them to a different directory! (mkdir my\_dir)

## Tutorial

This tutorial is an example of an antibody multistate design experiment. This builds off of the previous tutorial, antibody single-state design. In that tutorial we designed an antibody (CH67) against a single antigen (influenza hemagglutinin H1 SolomonIslands/03/2006). This type of design is known as single-state design, since we were designing a single antibody-antigen complex. This tutorial will explore multistate design, where we are designing multiple complexes (also known as states) simultaneously. We will design CH67 to retain binding for H1 SolomonIslands/03/2006 while also designing it to bind the pandemic strain H1 California/07/2009, which is not bound by CH67. In this case we will design two states - CH67 complexed with H1 SolomonIslands/03/2006, and CH67 complexed with to H1 California/07/2009.

1. Create a directory in the multistate\_design directory called my\_files and switch to that directory. Although many files you need for the tutorial are located in the input\_files directory, we will work from my\_files for the rest of the tutorial.

```
mkdir my_files
cd my_files
```

2. Prepare the input complexes for design.

1. Download the co-complex from the Protein Databank (PDB). We will copy the 4HKX complex from the previous tutorial.

```
cp ../input_files/4HKX_renum.pdb .
```

We also need to download the structure of the H1 California/07/2009, PDB ID 3ubq. **The 3UBQ.pdb file is provided in the input\_files directory.** However the instructions for downloading this PDB file are also provided below.

1. Go to rcsb.org and type '3ubq' in the search bar.
  2. Click on 'Download Files' on the right side of the page, then 'PDB Format'.
  3. Save the PDB file in the my\_files directory as '3UBQ.pdb'.
2. Prepare the PDBs for running through Rosetta. In general before running a PDB through Rosetta you should remove water molecules and all ligands that are non-essential to your protocol. We will use an automated script to do this processing.

1. We want to pull the HA1 domain of hemagglutinin (chain A) from the PDB 3UBQ.

```
python2.7 ~/rosetta_workshop/rosetta/tools/protein_tools/scripts/clean_pdb.py 3UBQ A
```

2. The HA structure of H1 SolomonIslands/03/2006 in the 4HKX complex is truncated to the globular head domain of HA. Since the structure in H1 California/07/2009 includes the full HA we will truncate it to match the 4HKX complex. Also, because the H1 California/07/2009 strain is not bound by CH67 we have to create a mock complex based on the 4HKX structure. We will manually align these PDB files in PyMOL to create a model of CH67 bound to H1 California/07/2009. Enter the following commands in the PyMOL command prompt to align the two structures, truncate the H1 California/07/2009 antigen and create a mock complex with CH67.

```
pymol 3UBQ_A.pdb 4HKX_renum.pdb
```

Click the 'S' button on the bottom left to show the sequences.

Align 3UBQ to the HA chain in 4HKX. Click the 'A' button next to 4HKX\_renum on the right side of the screen, click 'align', 'to molecule', '3UBQ\_A'.

From the sequence alignment we can determine where the sequence should be truncated. Enter the following command to remove these leading and trailing amino acids.

```
select truncation, 3UBQ_A and (resi 1-47 or resi 261-323)
remove truncation
```

Save the HA from the 3UBQ structure with the antibody from 4HKX - this will make up our mock complex that we will later design.

```
save 3UBQ_Ab.pdb, 3UBQ_A or ( 4HKX_renum and chain H+L )
```

1. Rename and reorder the chains in the CH67-3UBQ complex to the same convention as in the CH67-4HKX complex.

```
python2.7 ../../scripts/reorder_pdb_chains.py --new_chain_order H,L,A \
--new_chain_ids H,L,A --norestart 3UBQ_Ab.pdb 3UBQ_Ab_renum.pdb
```

3. Prepare a residue file (resfile) for multistate design.

Resfiles in multistate design function exactly as in single-state design. The resfile format is described in more detail in the single-state design tutorial. The main difference is that you should create a resfile for each state, since each state can designate different residues for repacking.

In this tutorial, we will use script `define_interface.py` to define which residues are at the antibody-antigen interface. Interface residues on the antibody will be redesigned, and those on the antigen side will be repacked. This algorithm defines interface residues as those with a heavy atom within 5 Å of a heavy atom on a residue on the opposing side of the interface.

**Note: the number of designed residues must be the same for each state. This is very important, and multistate design will not run if a different number of designed residues are specified.** For this reason, after we create the resfiles for 4HKX and 3UBQ, we have to manually edit the resfiles to remove designed residues that are present in one resfile but not the other.

1. Run `define_interface.py` to generate a resfile with designable and repackable residues.

```
python2.7 ../../scripts/define_interface.py --side1 HL --side2 A --design-side 1 \
--repack --output 4HKX 4HKX_renum.pdb
python2.7 ../../scripts/define_interface.py --side1 HL --side2 A --design-side 1 \
--repack --output 3UBQ 3UBQ_Ab_renum.pdb
```

2. Open the two resfiles in `gedit` to make sure both resfiles have the same number of designable residues.

```
gedit 4HKX.resfile 3UBQ.resfile &
```

Notice that residues 30, 32, 50, 54, 57, 59, 102, 103, 108, 110, 111, 207, and 210 are designed in one resfile and not the other. Delete these residues and save the two resfiles.

4. Repack or relax the template structure.

We use the same strategy for relaxation of our input structure with constraints from the previous tutorial - however now we only need to relax the 3UBQ complex.

1. Copy the relaxed 4HKX structure from the single state design tutorial.

```
cp ../input_files/4HKX_relax.pdb .
```

2. Copy the options file and command line for running relax from the input\_files directory.

```
cp ../input_files/relax.options .
cp ../input_files/relax.command .
```

3. Run relax with constraints on the 3ubq complex.

```
~/rosetta_workshop/rosetta/main/source/bin/relax.default.linuxgccrelease \
  @relax.options -s 3UBQ_Ab_renum.pdb > relax.out &
```

4. The relaxed model will take some time to run (~90 min) - you can move on to the next step using pre-generated models. Copy the best scoring model to 3UBQ\_relax.pdb. In this case we only make a single model, but in general it's recommended to make a larger number of models (~10-50). **In this case there are pre-generated relaxed models located in the output\_files directory.** The lowest energy model is 3UBQ\_Ab\_renum\_0004.pdb. Copy this file to the current directory.

```
cp ../output_files/3UBQ_Ab_renum_0004.pdb 3UBQ_relax.pdb
```

5. Run multistate design on our antibody complexed with both antigens.

At this point all of our input files are ready and we can run multistate design. The multistate design protocol involves four rounds of sequence design with constraints applied to encourage sequence convergence between all of the states. Between these design steps we will include backbone flexibility via backrub motions. At the end of the protocol there is a step in which we select the best sequence from all of our states, and use this as the final sequence.

1. Copy multistate\_design.xml and multistate\_design.options from the input\_files directory.

```
cp ../input_files/multistate_design.xml .
cp ../input_files/multistate_design.options .
cp ../input_files/multistate_design.command .
```

2. Read through the XML and options files, and familiarize yourself with what different steps of the protocol are doing.

3. Generate ten designed models. These models will take some time (~5 minute per design).

```
~/rosetta_workshop/rosetta/main/source/bin/rosetta_scripts.default.linuxgccrelease \
  @multistate_design.options -parser:protocol multistate_design.xml \
  -out:suffix _multistate_design -scorefile multistate_design.fasc
```

4. As a control we will repeat the same protocol without designing any residues. We will run this control design for both 4HKX and 3UBQ complexes separately. Copy the same control files from the input\_files directory and use these to run a control experiment for the 3UBQ complex.

```
cp ../input_files/design.options .
cp ../input_files/3UBQ_control.resfile .
cp ../input_files/4HKX_control.resfile .
cp ../input_files/3UBQ_design_control.xml .
cp ../input_files/4HKX_design_control.xml .
cp ../input_files/design_control.command .
```

5. Generate ten control models for the 3UBQ complex.

```
~/rosetta_workshop/rosetta/main/source/bin/rosetta_scripts.default.linuxgccrelease \  
@design.options -parser:protocol 3UBQ_design_control.xml -out:suffix _control \  
-scorefile 3UBQ_control.fasc -s 3UBQ_relax.pdb
```

```
~/rosetta_workshop/rosetta/main/source/bin/rosetta_scripts.default.linuxgccrelease \  
@design.options -parser:protocol 4HKX_design_control.xml -out:suffix _control \  
-scorefile 4HKX_control.fasc -s 4HKX_relax.pdb
```

6. The design and control models will take some time to finish. While you are waiting you can move on to the next step with the pre-generated results in the `output_files` directory. Make a new directory and copy the files `multistate_design.fasc`, `3UBQ_control.fasc`, and `4HKX_control.fasc` from the `output_files` directory into this new directory.

6. Analyze the designed sequences.

Analysis of multistate design results is slightly more complicated than the analysis of single-state designs. The output of design is a set of fifty pairs of models, one of CH67 in complex with the antigen 4HKX and the other in complex with the antigen 3UBQ. Each pair of models will have an identical sequence in the designed residues. For example, the outputs `4HKX_relax_multistate_design_0001.pdb` and `3UBQ_relax_multistate_design_0001.pdb` will have the same CDR sequences in complex with the two different antigens. To analyze these results we will look at the difference in score, binding energy, and binding density for the designed sequences in both the 3UBQ and 4HKX complexes.

1. Plot the score and binding energy of designed models against control models. The script `compare_design_to_control_multistate.py` plots the same metrics as the script used in the single-state design tutorial, but will split the results to show the effect of mutations on both the 3UBQ and 4HKX complexes.

```
python2.7 ../../scripts/compare_design_to_control_multistate.py 3UBQ \  
3UBQ_control.fasc multistate_design.fasc  
python2.7 ../../scripts/compare_design_to_control_multistate.py 4HKX \  
4HKX_control.fasc multistate_design.fasc  
gthumb *png &
```

2. Use the `design_analysis.py` script to look at what mutations specifically were made in the design process. Make a sequence logo from our designed models to illustrate these changes. Since each multistate design run outputs a pair of complex (3UBQ and 4HKX) with an identical sequence, we only need to analyze one complex of the two to see all the mutations that were made.

```
python2.7 ../../scripts/design_analysis.py --prefix multistate_design \  
--res 4HKX.resfile --native 4HKX_relax.pdb 4HKX*design*.pdb  
gthumb multistate_design_seq_log.png &
```

3. In this case Rosetta was able to introduce mutations that result in an antibody with a lower (more favorable) score bound to both antigens. However the binding energy and binding density seem to close to or slightly worse than the values from the native complex. Frequently in multistate design antibody mutations will establish stronger intramolecular contacts, since these contacts are uniform across all states, at the expense of cross-interface contacts, which can vary between target states. Open up the lowest scoring models in PyMOL and analyze what impact the Rosetta-introduced mutations have on interactions with both targets.