Protocol capture

Introduction

The following is a protocol capture of designing cyclic peptides based on antibody CDRH3 loops. It will review the design of CDRH3 loop from anti-influenza antibody C05.

All Rosetta commands for this publication were run with version 3e41de71be009712db5ba0f3b0cd1080a1603181, from March 2016.

All materials from this protocol capture can be downloaded from https://github.com/sevya/cyclic_peptide_protocol_capture

Dependencies:

Several scripts require the use of Python 2.7 as well as the Biopython package (https://github.com/biopython/biopython.github.io/). We recommend installing all necessary packages before beginning this protocol. Note that all analysis scripts will only function properly if they are in the correct directory as provided.

Structure preparation

First, the C05 Fab structure (PDB ID 4fnl) was downloaded from the Protein DataBank (PDB; www.rcsb.org) and manually processed in PyMol. All waters were removed from the structure and non-protein residues were also removed. All residues except for one copy of the CDRH3 from chain H (residues 93-102, sequence AKHMSMQQVVSAGWERADLVGDAFDV) were deleted. The loop was then saved to a PDB file with the following command:

```
save C05_H3.pdb
```

The CDRH3 peptide was then renumbered using a python script to convert the numbering to start from 1 and ignore insertion codes. The renumbering script was run with the following command:

```
/path/to/Rosetta/tools/protein_tools/scripts/pdb_renumber.py
C05_H3.pdb C05_H3_renum.pdb
```

Next the PeptideStubMover functionality in Rosetta was used to add a cysteine to the N- and C-termini of the CDRH3 peptide. The following command will add these cysteines:

```
/path/to/Rosetta/main/source/bin/rosetta_scripts.default.linuxgc
crelease -parser:protocol add_disulfide.xml -s C05_H3_renum.pdb -
out:prefix disulfide_ -out:no_nstruct_label -extra_res_fa
CYD.params
```

GeneralizedKIC Loop Closure

Now the peptide is ready for loop modeling simulations. We used Generalized Kinematic Closure (GeneralizedKIC) to close the loop and perturb the φ and ψ angles. Full documentation of the GeneralizedKIC protocol can be found at https://www.rosettacommons.org/docs/latest/scripting_documentation/RosettaScripts/composite_protocols/generalized_kic/GeneralizedKIC. Briefly, we first declare a bond between the cysteines at residues 1 and 28, and set the degrees of freedom to be used in loop modeling. We set residue 13 at the tip of the CDRH3 loop to be the anchor point of loop modeling, and the remaining residue to be mobile degrees of freedom. We next add a perturber that will modify the closed loop by perturbing the φ and ψ angles of all residues in the loop by a value drawn from Gaussian distribution centered at 15 degrees. Finally we add a single round of ROSETTA relax to add side chains and refine the structure before evaluating the energy. The GeneralizedKIC protocol will generate 20 solutions after loop closure, perturbation, and relaxation, and the lowest energy solution is reported as the final decoy. This entire protocol is repeated to generate 1,000 final output decoys. To create the models output directory and run the GeneralizedKIC protocol use the following command:

```
mkdir models

/path/to/Rosetta/main/source/bin/rosetta_scripts.default.linuxgc
crelease @close_relax.flags -s disulfide_C05_H3_renum.pdb
```

Analysis

We will use a python script to analyze the folded peptide models by calculating ROSETTA score and Cα RMSD for each of the models. To do so we will run the following command:

```
python
/path/to/Rosetta/tools/protein_tools/scripts/score_vs_rmsd.py --
table native_sc_vs_rmsd.tsv --native disulfide_C05_H3_renum.pdb
--CA --term total models/disulfide*pdb

python plot_score_vs_rmsd.py native_sc_vs_rmsd.tsv
```

This will make a plot of score vs RMSD for all models. In addition it will give a funnel discrimination score that is used to assess how well models converge on the native conformation. This score is derived from Conway et al.[1]. Overall a lower score (more negative) indicates that the structures are converging well.

Peptide sequence redesign

Next we want to see if we can redesign the peptide for greater stability and convergence on the active conformation. As an example we will take the lowest RMSD structure and run fixed backbone ROSETTADESIGN to optimize the sequence. In a production run we

[1] Patrick Conway et al., "Relaxation of Backbone Bond Geometry Improves Protein Energy Landscape Modeling.," *Protein Science* 23, no. 1 (January 2014): 47–55, doi:10.1002/pro.2389.

recommend to design more than one structure – in the manuscript we redesigned all peptides under 2 Å.

```
mkdir redesign/
sort -nk3 native_sc_vs_rmsd.tsv | head -2
```

Next copy the lowest energy model into the redesign folder. We provide an example structure for the purpose of this protocol capture. We will run 10 iterations of fixed backbone design and use the lowest scoring design. The resfile we use to guide design will allow all residues to be mutated to anything except for cysteine (ALLAAxC) and will disallow design on the N- and C-termini cysteines.

```
cp models/disulfide_C05_H3_renum_close_relax_0050.pdb redesign/

/path/to/Rosetta/main/source/bin/fixbb.default.linuxgccrelease -
s disulfide_C05_H3_renum_close_relax_0050.pdb -nstruct 10 -
out:prefix redesign_ -ex1 -use_input_sc -resfile
redesign.resfile

redesign.resfile:
NATAA
start
2 - 27 A ALLAAxC
```

After making the peptide designs we will analyze the lowest scoring design to see if it improves the folding funnel. We will first use python scripts to convert the sequence from the PDB into a fasta file, then create a resfile from the fasta file to mutate our folding template.

```
/path/to/Rosetta/tools/protein_tools/scripts/get_fasta_from_pdb.
py redesign_disulfide_C05_H3_renum_close_relax_0050_0009.pdb A
redesign.fasta

python fasta_to_resfile.py redesign.fasta
```

This will create a resfile called `redesign_disulfide_C05_H3_renum_close_relax_0050_0009.resfile` that we will use to mutate our peptide template. Navigate back to the starting directory, create the template for folding and run the folding simulations:

```
cd ..

/path/to/Rosetta/main/source/bin/fixbb.default.linuxgccrelease -
s disulfide_C05_H3_renum.pdb -out:prefix d1_ -resfile
redesign/redesign_disulfide_C05_H3_renum_close_relax_0050_0009.r
esfile -out:no_nstruct_label -use_input_sc

/path/to/Rosetta/main/source/bin/rosetta_scripts.default.linuxgc
crelease @close_relax.flags -s d1_disulfide_C05_H3_renum.pdb
```

After the design folding decoys are finished we can analyze the score and RMSD and compare to the native sequence:

```
python
/path/to/Rosetta/tools/protein_tools/scripts/score_vs_rmsd.py --
table d1_sc_vs_rmsd.tsv --native disulfide_C05_H3_renum.pdb --CA
--term total models/d1*pdb
```

```
python plot_score_vs_rmsd.py native_sc_vs_rmsd.tsv
d1_sc_vs_rmsd.tsv
```

Binding affinity measurement

In the design process to this point we haven't accounted for the effect that a mutation may have on binding to the antigen. After we have identified a candidate peptide we next want to make sure that the mutations that stabilize the peptide do not interfere with an interaction hotspot. To do this we will model the redesigned sequence in the context of the antibody-antigen interface. We will thread the redesigned sequence over the co-crystal structure, perform a subtle refinement using ROSETTA relax with backbone constraints, and measure the binding energy.

First make a new subdirectory called `ddg_measure` to place all the new files and navigate to this directory. Next we need to download and prepare the structure of the C05 co-crystal structure (PDB ID 4fp8). Download the structure and use PyMol to delete all chains except for A+H. Remove all waters and non-protein residues from the structure. Then remove all residues on the antibody except for the CDRH3 (residues 93-102, sequence AKHMSMQQVVSAGWERADLVGDAFDV). Save this structure as C05_H3_Ag.pdb

Next we will renumber the structure to make sure the numbering is uniform. Repeat the same command from earlier in the protocol to renumber the structure:

```
/path/to/Rosetta/tools/protein_tools/scripts/pdb_renumber.py
C05_H3_Ag.pdb C05_H3_Ag_renum.pdb
```

Once the structure is prepared we will run the relaxation on both the native sequence and redesigned sequence. As an example the sequence of one of the peptides provided in this manuscript (d1) is provided, along with a native resfile to measure the binding energy of the wild-type loop. Run the constrained relaxation with these two resfiles to create the models:

```
mkdir models
```

```
/path/to/Rosetta/main/source/bin/rosetta_scripts.default.linuxgc
crelease @relax.flags -out:suffix _rlx_d1 -parser:script_vars
resfile=d1
```

This will generate ten models for each of the sequences. To analyze the results use the following command to output the score and binding energy in the bound state:

```
sort -nk2 relax.fasc | grep rlx_native | head -1 | awk '{print
$2"  "$6}'
sort -nk2 relax.fasc | grep rlx_d1 | head -1 | awk '{print $2"
"$6}'
```

Molecular dynamics simulations
This section makes use of several packages:

Amber18/AmberTools18 (available at http://ambermd.org/; note that AmberTools includes the original sander MD code and can be downloaded and installed free of cost, but the GPU-accelerated pmemd.cuda is only available at cost with Amber18). Prior to beginning, make sure your shell environment is setup for Amber18:

```
source /path/to/amber18/amber.sh
```

PyEMMA2 (available for free at http://emma-project.org/latest/INSTALL.html)
After we have finalized our designed peptides from Rosetta, we need to parameterize our structures for MD simulations using Leap. Before we can do that, however, we need to prepare out PDB files.

First, we need to remove hydrogen atoms so that Leap can reassign them:

```
reduce -Trim d4v2.amber.pdb > d4v2.amber.noh.pdb
```

Then, we need to let Amber know that the cysteine residues in our peptides are disulfide bonded. This requires changing the three-letter residue IDs in the PDB files from CYS to CYX and indicating the connection between them. This can be done manually, or by running the following command:

```
pdb4amber -i d4v2.pdb -o d4v2.amber.pdb
```

After these two steps, generate the Leap input file to solvate and neutralize our peptides in TIP4EW water. This can be easily wrapped into a Shell script and parallelized for multiple proteins. Generate the Amber parameter files and input coordinates with the `run_tleap.sh` script:

```
./run_tleap.sh d4v2.amber.pdb
```

Run the script called "`check_prmtop_validity.sh`" to check the `.prmtop` file for errors using ParmEd:

```
./check_prmtop_validity.sh d4v2.amber.prmtop
```

Once we have valid parameter and input coordinate files for our solvated peptide system, we can perform system minimization. We minimize in three stages. First, minimize the buffer with restraints on all protein atoms. Second, minimize the protein with restraints on the buffer atoms. Third, minimize everything without restrains. Use the script titled "`md_minimization.sh`"

```
./md_minimization.sh
```

Our system is now minimized and prepared for heating. Heating is completed in two stages: Stage one will heat from the minimized structure to 100K, and stage two will heat from 100K to 300K. To do this, use the script titled "`md_heat.sh`":

```
./md_heat.sh
```

The final step in the heating may need to be repeated one or two times because of large changes in the periodic boundary conditions. Once system density is equilibrated, we continue on to production in the NPT ensemble. Run the script "`md_production.sh`":

```
./md_production.sh
```

CPPTRAJ Analysis

In this section we will use CPPTRAJ to compute RMSF and per-residue RMSD values of the heavy atoms in our trajectories. In the manuscript, RMSF values were computed this way for the 1x500ns trajectories of each unbound peptide. Note that our trajectories only contain protein atoms. Therefore, we have to use the "dry" version of the topology file generated with Leap. Run the script to compute the RMSFs:

```
./rmsf.sh dry.d4v2.amber.prmtop d4v2.amber_prod.*.nc
```

Separately, per-residue RMSDs were computed for each of the 10 metastable states identified in the estimated Markov models from the 2x4000ns trajectories using the designed peptide structure as a reference. Run the script: "`rmsd_perres.sh`":

```
./rmsd_perres.sh
```

Markov modeling with PyEMMA2

In this section we will use PyEMMA2 to estimate maximum likelihood Markov models from our trajectories. Before passing trajectories to PyEMMA2, you can reimage in CPPTRAJ. Run the script "`combine_trajs.sh`":

```
./combine_trajs.sh dry.d4v2.amber.prmtop
```

PyEMMA2 makes extensive use of the MDTraj package. Therefore, you may instead modify the scripts below to let the MDTraj interface know that you are using periodic boundary conditions. MDTraj will attempt to re-image the system for you. Though, for simulations performed in Amber, I generally prefer to do this in CPPTRAJ.

Next, we create a Python script to featurize the 2x4000ns trajectories for each peptide, cluster them, and estimate Markov models. This script will output stationary distributions for each of 10 metastable states, as well as sample 1000 structures from each metastable state probabilistically.

Run `build_markov_model_pyemma.py` as follows:

```
python build_markov_model_pyemma.py -topfile dry.d4v2.amber.pdb
-trajfiles d4v2.all.nc -backbone_torsions -sidechain_torsions -
stride 1 -tica 0.95 -k_clusters 1000 -k_iter 100 -lag 2 -dt "10
ps" -output_prefix blah -n_metastable_states 10 -
n_samples_each_state 1000
```

The per-residue RMSDs can then be computed with CPPTRAJ for each re-sampled distribution of each metastable state, and the values can be re-weighted with their corresponding stationary distribution probabilities.