

Workshop #4: PyRosetta Folding

In this workshop you will write your own Monte Carlo protein folding algorithm from scratch, and we will explore a couple of the tricks used by Simons *et al.* (1997,1999) to speed up the folding search.

Suggested Readings

1. K. T. Simons *et al.*, “Assembly of Protein Structures from Fragments,” *J. Mol. Biol.* **268**, 209-225 (1997).
2. K. T. Simons *et al.*, “Improved recognition of protein structures,” *Proteins* **34**, 82-95 (1999).
3. Chapter 4 (Monte Carlo methods) of M. P. Allen & D. J. Tildesley, *Computer Simulation of Liquids*, Oxford University Press, 1989.

A simple *de novo* folding algorithm

First, we would like to create a simple folding algorithm. Begin with a new pose, and then create a starting structure using:

```
pose=Pose()  
make_pose_from_sequence(pose, "AAAAAAAAAA", "fa_standard")
```

Dump these coordinates and examine briefly in PyMol. You should see ideal bond lengths and angles, although the set of ϕ/ψ angles will not be useful.

Write a program which implements a Monte Carlo algorithm to optimize the protein conformation. In the main program, create a loop with 100 iterations. Each iteration should call a subroutine to make a random trial move, and then score the protein, and then accept or reject the new conformation based on the Metropolis criteria. Use $kT = 1$.

For the random trial move, write a *subroutine* to choose one residue at random and then randomly perturb either the ϕ or ψ angles by a random number chosen from a Gaussian distribution with a standard deviation of 25° . Use the Python built-in `random.gauss()` from the `random` library.

For the energy function, use the standard full-atom scoring approach with *only* the van der Waals and hydrogen bonding terms. With this scoring function, what kind of structures do you expect to be most stable?

At each iteration of the search, output the current pose energy and the lowest energy ever observed. The final output of this program should be the lowest energy conformation that is achieved at any point during the simulation. Be sure to use `low_pose.assign(pose)` rather than `low_pose = pose`, since the latter will only copy a pointer to the original pose.

1. Output the last pose and the lowest-scoring pose observed, and view them in PyMol. Plot the energy and lowest-energy observed vs. cycle number. What are the energies of the initial, last, and lowest-scoring pose? Is your program working? Has it converged to a good solution?
2. Using the program you wrote for workshop 2, force the A₁₀ sequence into an α -helix. Does this structure have a lower score than that produced by your algorithm? What does this mean about your sampling or discrimination?
3. Since your program is a stochastic search algorithm, it may not produce an ideal structure consistently, so try running the simulation multiple times or with a different number of cycles (if necessary). Using a kT of 1, your program may need to make up to 500,000 iterations.

Low-resolution (centroid) scoring

Following the treatment of Simons *et al.* (1999), Rosetta can score a protein conformation using a low-resolution representation. This will make the energy calculation faster.

4. Load a protein with which you are familiar (e.g. ras or Cetuximab). Calculate the full-atom energy and note the coordinates of residue 5 using `print pose.residue(5)`.
5. Convert the pose to the centroid form by using the `SwitchResidueTypeSetMover`:

```
switch = SwitchResidueTypeSetMover('centroid')
switch.apply(pose)
print pose.residue(5)
```

How many atoms are now in residue 5? How is this different than before?

6. Score the new, centroid-based pose using the standard score function `score3`. What is the new total score? What scoring terms are included in `score3`? Do these match Simons?

7. Convert the pose back to all-atom form by using another switch mover (`SwitchResidueTypeSetMover('fa_standard')`). Confirm that you have all the atoms back. Are the atoms in the same position as before?
8. Adjust your folding algorithm to use centroid residue types. How much faster does your program run?

Protein fragments

9. Create a 3-mer fragment file from the fragment library (<http://robeta.bakerlab.org/fragmentsubmit.jsp>) for a given test sequence. This file contains 3-mer fragments for the test sequence we are trying to fold. You should see sets of three-lines describing each fragment. For the first fragment, which PDB file does it come from? Is this fragment helical, sheet, or loop, or a combination? What are the ϕ , ψ , and ω angles of the middle residue?
10. How many 3-residue windows are there in your 11-residue protein? How many fragments does the data file have per window? You might check your answer using the shell command `wc`, which can tell you how many total lines are in the fragment file.
11. Create a new subroutine in your folding code for an alternate random move based upon a fragment insertion. Prior to calling the subroutine, load the set of fragments from the fragment file:

```
fragset = ConstantLengthFragSet(3)
fragset.read_fragment_file("aatestA03_05.200_v1_3")
```

Next, create a fragment mover using this fragment set and a “MoveMap.” A MoveMap specifies which degrees of freedom are allowed to change in the pose when the mover is applied (in this case, all backbone torsion angles):

```
movemap = MoveMap()
movemap.set_bb(True)
mover_3mer = ClassicFragmentMover(fragset, movemap)
```

Each time this mover is applied, it will select a random window and insert a random fragment:

```
mover_3mer.apply(pose)
```

When you change your random move to a fragment insertion, how much faster is your folding code? Does it converge to a protein-like conformation more quickly?

Programming exercises

1. Fold a 10-mer poly-alanine using 100 independent trajectories (use any variant of the folding algorithm that you like). Create a Ramachandran plot using the lowest-scoring conformations from all 100 independent trajectories. Repeat this for an 10-mer poly-glycine. How do the plots differ? Compare with the plots in Richardson's article.
2. Test your folding program's ability to predict a real fold from scratch. Choose a small protein to keep the computation time down, such as Hox-B1 homeobox protein (1b72) or RecA (2reb). How many iterations and how many independent trajectories do you need to run to find a good structure?
3. Modify your folding program to include a simulated annealing temperature schedule, decaying exponentially from $kT = 100$ to $kT = 0.1$ over the course of the search. Again, fold a test protein. Does this approach work better?
4. Modify your folding program to remove the Metropolis criterion and instead accept trial moves *only* when the energy decreases. Plot energy vs. iteration, and examine the final output structures from multiple runs. How is the convergence and performance affected? Why?

Thought questions

1. [advanced] How might you design an intermediate-resolution representation of side chains that has more detail than the centroid approach yet is faster than the full atom approach? Which types of residues would most benefit from this type of representation?
2. [introductory] What are the limitations of these types of folding algorithms?