Workshop #3: Scoring

Scoring Poses

A basic function of Rosetta is calculating the energy or *score* of a biomolecule. Rosetta has a standard energy function for all-atom calculations as well as several scoring functions for low-resolution protein representations. In addition, you can tailor an energy function by including scoring terms of your choice with custom weights.

For these exercises, use the protein Ras (PDB ID 6Q21) and load it into a pose called `ras`. Be sure to clean the PDB file and use only one chain.

1. To score a protein, you must first define a scoring function:

   ```
   from pyrosetta.teaching import *
   scorefxn = get_fa_scorefxn()
   ```

   (Note that, beginning in this chapter, we will instruct you to import from the pyrosetta.teaching namespace. This is to save time. Best programming practice, however, would be to only import those specific keywords required by one's script from the appropriate namespaces.)

   The method `get_fa_scorefxn` tells Rosetta to load the default full-atom energy terms. To see these terms, you can print the score function:

   ```
   print scorefxn
   ```

   What terms are in the score function, and what are their relative weights?

   (Appendix A includes a list of Rosetta scoring function terms, which you may wish to reference.)

2. Set up your own custom score function that includes just van der Waals, solvation, and hydrogen bonding terms, all with weights of 1.0. Use the following to start:

   ```
   scorefxn2 = ScoreFunction()
   scorefxn2.set_weight(fa_atr, 1.0)
   scorefxn2.set_weight(fa_rep, 1.0)
   ```

   (The first line declares the object `scorefxn2` as part of the `ScoreFunction` class.) Enter the other weights and then confirm that the weights are set correctly.

3. Evaluate the energy of Ras with the standard score function:

   ```
   print scorefxn(ras)
   ```

   What is the total energy of Ras? __________

   Break the energy down into its individual pieces with the `show` method:

   ```
   scorefxn.show(ras)
   ```

   Which are the three most dominant contributions, and what are their values? Is this what you would have expected? Why?

4. Unweighted, individual component energies of each residue in a structure are stored in the `Pose` object and can be accessed by its `energies()` object. For example, to break the energy down into each residue's contribution use:

   ```
   print ras.energies().show(<n>)
   ```

   where <`n`> is the residue number.

   What are the total van der Waals, solvation, and hydrogen-bonding contributions of residue 24? (Note that the *backbone* hydrogen-bonding terms for each residue are *not* available from the `Energies` object.)

5. The van der Waals and solvation terms are atom–atom pairwise energies calculated from a pre-tabulated lookup table, dependent on the distance between the two atoms and their types. You can access the lookup table (`etable`) directly to check these energy calculations on an atom-by-atom basis:

    ```
    r1 = ras.residue(24)
    r2 = ras.residue(20)
    a1 = r1.atom("N")
    a2 = r2.atom("O")
    etable_atom_pair_energies(a1, a2, scorefxn)
    ```

    (Note that the `etable_atom_pair_energies()` function requires `Atom` objects, not the `AtomID` objects we saw in Workshop #2.)

    What are the attractive, repulsive, and solvation components between the nitrogen of residue 24 and the oxygen of residue 20?

    How does Rosetta separate the "attractive" and "repulsive" van der Waals components? (Hint: it is *not* the r^{-6} and r^{-12} parts of the Lennard-Jones equation.)

6. The hydrogen-bonding score component requires identification of acceptor hybridization state and calculation of geometric parameters including distance, acceptor bond angle, proton bond angle, and a torsion angle. The hydrogen-bonding energies are stored in an `HBondSet` object. You can access the list of hydrogen bonds by creating an `HBondSet` object, filling the set from the pose (after making sure the pose has had its `Energies` object updated based on neighboring residues within the pose), and then using the `HBondSet.show()` command:

    ```
    from rosetta.core.scoring.hbonds import HBondSet
    hbond_set = hbonds.HBondSet()
    ras.update_residue_neighbors()
    hbonds.fill_hbond_set(ras, False, hbond_set)
    hbond_set.show(ras)
    ```

    (Note that some functions and classes, such as `HBondSet`, must be referenced with their proper "namespace", such as the namespace `hbonds` here.)

    The above steps have been combined in the PyRosetta `toolbox` method `get_hbonds()`, so that we can also simply type:

    ```
    from pyrosetta.toolbox import get_hbonds
    hbond_set = get_hbonds(ras)
    hbond_set.show(ras)
    ```

An individual residue's hydrogen bonds can be looked up from the set using its residue number:

```
hbond_set.show(ras, 24)
```

How many hydrogen bonds does residue 24 make? _______

Using the parameters from the show function, sketch a picture of this hydrogen bond. Label all possible atoms, distances, and angles.

7. Analyze the energy between residues Y102 and Q408 in cetuximab (1YY9). (You'll need to load that structure into a new `Pose` object.

   a. As explained in Workshop #2, internally, a `Pose` object has a list of residues, numbered starting from 1. To find the residue numbers of Y102 of chain D and Q408 of chain A, use the residue chain identifier and the PDB residue number to convert to the pose numbering:

   ```
   print pose.pdb_info().pdb2pose('A', 102)
   ```

   D:Y102: _____ A:Q408: _____

   b. Score the pose and determine the Van der Waals energies and solvation energy between these two residues. Use the following commands to isolate contributions from particular pairs of residues, where `rsd1` and `rsd2` are the two residue objects of interest (not the residue number — use `pose.residue(res_num)` to access the objects):

   ```
   emap = EMapVector()
   scorefxn.eval_ci_2b(rsd1, rsd2, pose, emap)
   print emap[fa_atr]
   print emap[fa_rep]
   print emap[fa_sol]
   ```

   c. How do Rosetta's calculations compare to what you might calculate by hand? (See references.)

d. Create a new PDB file containing coordinates for just the two residues Y102 and Q408. Repeat the above calculations. Which energies change? Why?

Energies and the PyMOL Mover

The `PyMOLMover` class contains a method for sending score function information to PyMOL, which will then color the structure based on relative residue energies.

8. Instantiate a `PyMOLMover` and then use `pymol.send_energy(pose)` to send the coloring command to PyMOL after scoring `ras` with `scorefxn`.
   What color is residue Pro34? ______
   What color is residue Ala66? ______
   Which residue has a lower energy? _______

9. `pymol.send_energy(pose, fa_atr)` will have PyMOL color only by the attractive van der Waals energy component. What color is residue 34 if colored by *solvation* energy? ______

If its `update_energy` option is true, the PyMOL mover will update the colors by energy every time the mover is applied to the pose:

```
pymol.update_energy(True)
pymol.energy_type(fa_atr)
pymol.apply(pose)
```

You can also have PyMOL label each Cα with the value of its residue's energy:

```
pymol.label_energy(pose, fa_rep)
```

Finally, if you have scored the pose first, you can have PyMOL display all of the calculated hydrogen bonds for the structure:

```
pymol.send_hbonds(pose)
```

Programming Exercises

1. *Interface energy*. Write a program that can calculate the binding energy of EGFR to cetuximab. You will need to make separate PDB files for the antigen, antibody, and complex. In PyMOL, select one of these peptides, then use File→Save Molecule.

   Use the following formula for binding energy:

$$E_{\text{binding}} = E_{\text{complex}} - E_{\text{antibody}} - E_{\text{antigen}}$$

Submit your script along with its output, which should include values for the total binding energy, along with the *weighted* contributions to the binding energy from Van der Waals, hydrogen bonding, and solvation. What does your result suggest about these two proteins *in vitro*? What are some inaccuracies in the way you've calculated the binding energy?

2. *Statistical energy functions*. Write a program to output a file of data of the C–N bond lengths observed in a set of ten high-resolution X-ray protein structures. (One source of curated structures is the WHATIF sets at http://swift.cmbi.ru.nl/swift/whatif/select/.)

    a. Import the file into a spreadsheet, and plot the data as a histogram of probability versus bond length and as a statistical energy ($E = -kT \ln P$, where P is probability and kT is set to 0.6 kcal/mol.) versus bond length. Try a bin size of 0.01 Å.

    b. Look up the CHARMm parameters for this bond stretch, and plot a curve over the statistical energy vs. bond length figure to show the CHARMm model for this motion.

    c. Do the statistics match what would be produced by a harmonic oscillator under the CHARMm potential? Specifically, are the average bond length and the CHARMm spring constant K correct? If not, what should it be? You may need to fit a parabola to your data to find the average bond length and the spring constant K.

3. Write a program to loop over all pairs of atoms in two residues to confirm that the sum of the individual atom–atom pair energies (calculated using the `Etable` lookup) is the same as the total residue-residue pair energy.

References

1. E. Neria, S. Fischer & M. Karplus, "Simulation of activation free energies in molecular systems," *J. Chem. Phys.* **105**, 1902-1921 (1996).
2. T. Kortemme, A. V. Morozov & D. Baker, "An orientation-dependent hydrogen bonding potential improves prediction of specificity and structure for proteins and protein-protein complexes," *J. Mol. Biol.* **326**, 1239-1259 (2003).
3. D. Eisenberg & A. D. McLachlan, "Solvation energy in protein folding and binding," Nature 319, 199-203 (1986).
4. T. Lazaridis & M. Karplus, "Effective energy function for proteins in solution," Proteins 35, 133-152 (1999).