Assignment 3: Serial Ab Initio Protein Folding Using PyRosetta

---

1. Background and Objectives

Proteins perform a myriad of biological functions, and knowing their three-dimensional structures is crucial. When a homologous template is not available, **ab initio folding** methods are used to predict protein structures from first principles. In this assignment, you will build a serial (non-parallel) pipeline for ab initio folding inspired by Rosetta's AbinitioRelax protocol.

You will work with the well-known 35-residue villin headpiece sequence:

> **Villin Headpiece Sequence:**
> `MLSDEDFKAFGMTRSAFANLPLWKQQNLKKEKLLF`
>
> *This sequence has been widely used as a benchmark in ab initio folding studies (see, e.g., PDB ID 1VII or the work of McKnight and co-workers).*

Your pipeline will perform the following tasks:

1. **Generate a Starting Pose:**
   Create an idealized fullatom pose from the villin headpiece sequence using PyRosetta's `pose_from_sequence()` function.

2. **Linearize and Convert to Centroid Mode:**
   Linearize the pose by setting backbone torsion angles to nearly extended values, then convert the pose into centroid mode to simplify sidechain representation and speed up sampling.

3. **Setup MoveMap and Fragment Movers:**
   Define a MoveMap that allows all backbone motions. Load provided fragment library files (a 9-mer and a 3-mer fragment file) to generate fragment movers using the PyRosetta functions `ConstantLengthFragSet()` and `ClassicFragmentMover()`.
   Input files required:

   - Long fragment file: e.g. `aat000_09.frag`

- Short fragment file: e.g. `aat000_03.frag`
  (These files are typically generated from the Robetta server.)

4. **Combine Moves with Monte Carlo Sampling:**
   Chain the fragment insertion moves with a SequenceMover and wrap them in a TrialMover controlled by a Monte Carlo object (using a centroid score function like "score3"). The Monte Carlo object will use the Metropolis criterion to accept or reject moves based on:
   $p(\text{accept}) = \min(1, \exp(-\Delta E / kT))$
   where ΔE is the energy change and kT is the temperature parameter (set, for example, to 3.0).

5. **Recover and Convert the Best Decoy:**
   After running a fixed number of cycles (e.g., 300 cycles), recover the lowest-energy decoy using the Monte Carlo object's recovery function, convert it back to fullatom mode, and write the output to a PDB file.

6. **(Optional) Analysis and Visualization:**
   Using BioPython and py3Dmol, align the predicted decoy structure with a provided native structure (e.g., `native.pdb`) by extracting Cα atoms and using the `Superimposer` class to compute the RMSD. Visualize the aligned structures using py3Dmol with distinct coloring.

---

2. Detailed Description of PyRosetta Functions and Components

Below is a step-by-step explanation of each coding component and the PyRosetta functions involved:

A. Initialization and Pose Creation

- **`pyrosetta.init(extra_options="-in::file::fullatom -mute all")`**
  Initializes the PyRosetta environment. The flag `-in::file::fullatom` starts the system in fullatom mode, and `-mute all` reduces verbosity.

- **`pose_from_sequence(sequence, "fa_standard")`**
  Creates an idealized fullatom pose from the input sequence (here, the villin headpiece). This function builds the protein's backbone and sidechains using standard geometry.

- **Linearization of the Pose:**
  A custom function loops over each residue in the pose (using `pose.total_residue()`) and sets:
  - φ (phi) to –150°
  - ψ (psi) to 150°
  - ω (omega) to 180°
    This "linearizes" the structure to an extended conformation, ensuring the starting point is unbiased.

B. Centroid Conversion and MoveMap Setup

- **`SwitchResidueTypeSetMover("centroid")`**
  Converts the fullatom pose to a centroid representation where sidechains are reduced to single atoms. This simplifies the energy landscape.

- **`rosetta.core.kinematics.MoveMap()`**
  Creates a MoveMap object to control which degrees of freedom can change. In this assignment, we set the MoveMap to allow all backbone (bb) movements using `movemap.set_bb(True)`.

C. Fragment Library Loading and Movers

- **`ConstantLengthFragSet(fragment_length, fragment_file)`**
  Reads the provided fragment library file (either for 9-mer or 3-mer fragments) and creates a set of fragments of a given constant length.

- **`ClassicFragmentMover(fragset, movemap)`**
  Applies fragment insertion moves to the pose using the fragments loaded from the library and the defined MoveMap. This simulates local conformational changes.

- **`RepeatMover(mover, repeat_count)`**
  Wraps a mover (e.g., a ClassicFragmentMover) so that it is applied multiple times within each cycle.

- **`SequenceMover()`**
  Combines several movers sequentially. Here, it is used to chain long fragment moves (wrapped in a RepeatMover) and short fragment moves.

D. Monte Carlo Sampling and Trial Moves

- `create_score_function("score3")`
  Creates a centroid-mode score function (here “score3”) that estimates the energy of the pose. It is a simplified energy function suitable for low-resolution sampling.

- `MonteCarlo(pose, scorefxn, kT)`
  Initializes a Monte Carlo object with the current pose, a score function, and a temperature parameter kTkTkT. The object tracks energy changes and implements the Metropolis criterion.

- `TrialMover(seq_mover, mc)`
  Wraps the SequenceMover so that after each set of fragment insertions, the Monte Carlo object decides whether to accept or reject the move.

- `RepeatMover(trial_mover, cycles)`
  Applies the entire trial move repeatedly for a fixed number of cycles, enabling thorough sampling of conformational space.

E. Recovery and Conversion

- `mc.recover_low(pose)`
  Recovers the lowest-energy (best) decoy recorded by the Monte Carlo object during the simulation.

- `SwitchResidueTypeSetMover("fa_standard")`
  Converts the decoy from centroid back to fullatom mode for detailed analysis and visualization.

- `pose.dump_pdb(filename)`
  Writes the final fullatom pose to a PDB file.

F. Analysis and Visualization (Optional)

- **BioPython’s `PDBParser` and `Superimposer`:**
  - `PDBParser()` loads PDB files of the native and decoy structures.
  - `Superimposer()` aligns two sets of Cα atoms and calculates the RMSD.

- **py3Dmol:**
  Used for interactive 3D visualization of the aligned structures, allowing you to color-code the native and predicted models (for example, native in green and decoy in magenta).

---

3. Input Files and Their Sources

Students will be required to provide the following files:

1. **Protein Sequence:**
   - **Villin Headpiece Sequence:**
     `MLSDEDFKAFGMTRSAFANLPLWKQQNLKKEKLLF`
     Reference: Commonly used in ab initio folding studies; see e.g., literature related to PDB ID 1VII.
2. **Fragment Library Files:**
   - **Long Fragment File:** e.g., `aat000_09.frag` (9-mer fragments)
   - **Short Fragment File:** e.g., `aat000_03.frag` (3-mer fragments)
     These files can be generated using the Robetta server's fragment library service.
3. **Native Structure PDB File (Optional, for Analysis):**
   - A PDB file (e.g., `native.pdb`) representing the experimentally determined structure of the villin headpiece for alignment and RMSD calculation.
4. *(For Template-Based Modeling Option)* **Template Files:**
   - A homologous template PDB file and an alignment file in PIR format (if students choose the template-based approach). These files must be placed in a directory called `templates`.

---

4. Assignment Instructions

Assignment Tasks

Part I – Structure Prediction (50 marks)

Choose one of the following approaches:

Option A: Template-Based Modeling (if a homolog is available)

- Generate an automated alignment using Biopython and build an ensemble of models with Modeller.
- Evaluate the models using a DOPE score (or similar scoring function) and select the best model.
- Save the selected model as a PDB file.

Option B: Ab Initio Folding (if no homolog is available)

- **Pose Creation:**
  Generate a starting pose from the provided villin headpiece sequence using `pose_from_sequence()`.
- **Linearization:**
  Linearize the pose by setting backbone torsion angles.
- **Centroid Conversion:**
  Convert the pose to centroid mode using `SwitchResidueTypeSetMover("centroid")`.
- **MoveMap Setup:**
  Create a MoveMap with full backbone flexibility.
- **Fragment Insertion:**
  Load provided fragment libraries (long and short) using `ConstantLengthFragSet()` and create fragment movers with `ClassicFragmentMover()`.
  Wrap these with `RepeatMover` and combine them with a `SequenceMover`.
- **Monte Carlo Sampling:**
  Set up a Monte Carlo object with `MonteCarlo()`, wrap your sequence moves in a `TrialMover`, and repeat the process with a `RepeatMover` for a fixed number of cycles.

- **Recovery and Conversion:**
  Recover the lowest-energy decoy using `mc.recover_low()`, convert it back to fullatom mode with `SwitchResidueTypeSetMover("fa_standard")`, and output the final structure as a PDB file.

Part II – Analysis and Visualization (20 marks)

- **Structural Alignment:**
  Use BioPython's `PDBParser` and `Superimposer` to align the predicted structure (decoy) with the provided native structure and compute the RMSD.

- **Visualization:**
  Visualize both structures using py3Dmol with distinct coloring.

Part III – Reporting and Discussion (10 marks)

- **Documentation:**
  Provide clear, well-commented code and a report (max 5 pages) describing:
    - Your design choices and parameter selection.
    - The functioning of each PyRosetta function used.
    - A discussion comparing template-based and ab initio methods.
    - An interpretation of your RMSD and energy results.

Submission Requirements

1. **Code:**
   A complete, self-contained Python script (or scripts) implementing your pipeline.

2. **Output Files:**

   - Final predicted structure (PDB file).
   - Energy convergence plots (if applicable).
   - RMSD calculation output.

3. **Report:**
   A PDF document detailing your approach, analysis, and discussion.

4. **References:**
   Cite the villin headpiece sequence source (e.g., literature or PDB ID 1VII) and any key Rosetta or PyRosetta documentation used.