Marathon Manual

Tim van Boxtel

April 1, 2013

Contents

Introduction

Marathon is a Python script designed to iteratively rotate molecules read from Protein Data Bank files. All rotations along a given lattice are performed for each branch in a molecule and the results are printed to output files. Plotting of the iterations is also provided. Currently lattices along the 90 degree and 45 degree unit are supported.

Usage

The script is intended to be run from the command line.

Requirements

Marathon requires Python v2.7, as well as the python packages numpy and (Optional for plotting) matplotlib. To install the requirements it is advised to use the python `easy_install` or the more recent `pip`, ie:

```
pip install numpy
pip install matplotlib
```

Mac OSX

A good article detailing how to best install numpy and matplotlib on Mac OS-X is here. If interactive plotting is needed, use the QtAgg backend in your matplotlibrc file.

Program Execution

This guide assumes that python and the script requirements are installed and available on the system path.
Furthermore, it assumes that the `marathon.py` script is also available in the path.

To iterate cubic rotations on a single file `mymol.pdb` and output to a local directory `output_dir`, one can simply execute::

```
python marathon.py -o output_dir mymol.pdb
```

This will save .pdb files in the `output_dir/mymol/` directory corresponding to each rotation iteration

By default, the iterations are simply numbered incrementally from 1. If more detailed iteration names are required, the `-d` or `--detailed` flags are available which will save the filenames in a more detailed manner stating the exact rotation matrix on each branch that was applied for that iteration, this takes a form [B?R?], where '?' are the branch number and roation number, accordingly.

The script can optionally calculate the root mean square distance for each iteration. This is done by calling the `-r` or `--rmsd` flags during program execution. This will create a file 'rmsd.txt' in the output directory and append the iteration name and rmsd for that iteration. The script `min_rmsd.py` can read this file to output the iterations with the minimum values.

For a full description of all options available from the command line, type::

```
python marathon.py --help
```

Plotting

Marathon also has simple plotting functionality, capable of plotting a molecule in 3D space. This is accomplished using the python matplotlib package. The plots are signaled using the `-p` or `--plot` flag upon program execution::

```
python marathon.py -o output_dir -p mymol.pdb
```

Optionally, the `-i` or `--interactive` flags can be given and the program will display each plot using matplotlib's interactive plot viewer. This is useful for getting immediate feedback as to what rotations are being performed. The interactive flag renders the plot flag unneccesary.

If using interactive mode, ensure that an appropriate matplotlib backend is available and configured in your home directory or in a matplotlibrc file located in the directory calling the script. For more info on matplotlib backends, see the matplotlib documentation.

Methodology

This section describes the algorithm that marathon uses to rotate a given molecule. It describes the file format used, definitions of flexible points and subsequent branches, rotation lattices presently supported, branch rotations, and rotation branch overlap detection.

PDB File format

The molecules are read and written using the *.pdb* file format as specified by the World Wide Protein Data Bank. Marathon currently uses a 3rd party module to read and write the molecules using this format. The relevant information parsed from these files for the purpose of the script are the atom ids, positions and element label, and the bond connections between the atoms

Branches and Flexible Points

This script defines a **flexible point** as any nitrogen (N) atom in the molecule sharing bonds with *only* carbon (C) atoms and at least 2 of them. This applies to any atom along a chain that does not occupy and end point. A flexible point may have 2 or more bonds attached to it, the set of flexible point and bonding atom represents a unique **branch** in the molecule. The unique branches and the subsequent chain of atoms along that branch are what are rotated for each rotation iteration. The vector that is made by the flexible point and the bond atom is called a **branch vector**. This branch vector serves as the reference vector for all rotations.

Rotation Lattices

The exact rotations which are used for each branch for each iteration are dependent on the rotation lattice that is being applied. Currently, a cubic (i.e. 90deg) and a triangular (i.e. 45deg) lattice are supported. In general, the rotation matrices are computed from using the Euler-Rodrigues Forumlas.

Cubic

A cubic lattice is the set of 90 degree rotations that a branch can undergo in 3D space. For the unit branch shown in Figure 1, there are 6 unique rotations that can be performed relative to the branch vector.

Triangular

Similar to the cubic case, but the unit branch is rotated in increments of 45 degrees. This set is larger than the cubic case with 26 unique rotations on a single unit branch, as can be seen in Figure 2.

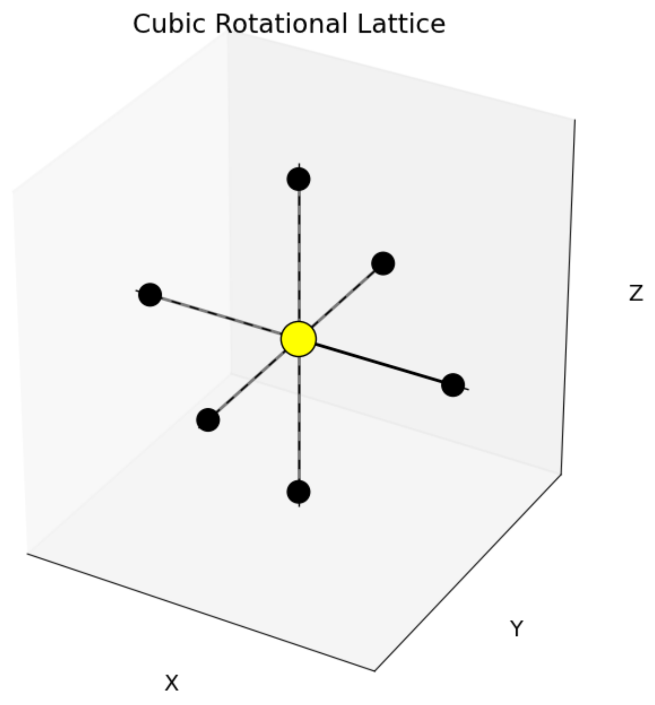

Figure 1: cubic unit rotation lattice. There are 6 rotations per branch

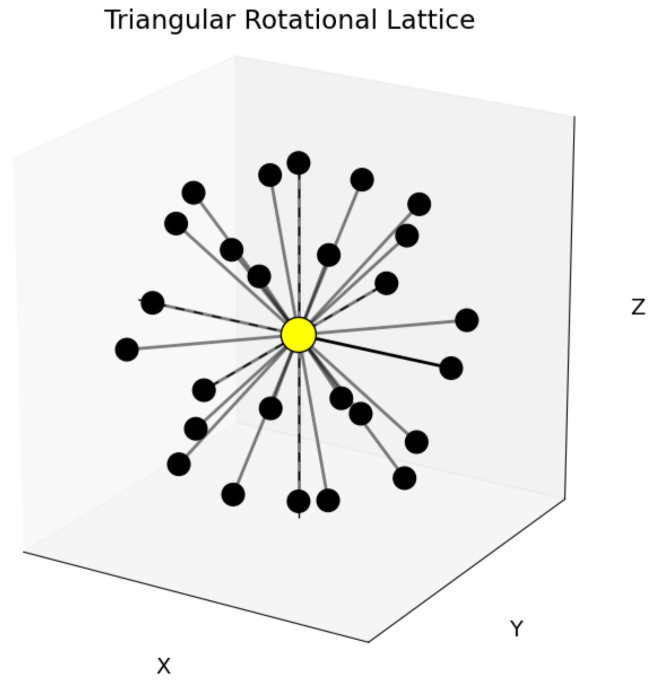

Figure 2: Triangular unit rotation lattice. There are 26 unique rotations per branc

Permutations of Branch Rotations

What was described in the section on rotation lattices applies to the simple case of 1 flexible point and 2 unique branches. While this aids in forming a convincing picture of the unique rotation lattices, these must all be permuted for each branch that is detected. In general there will be an upper limit of R^B permutations, where R is the number of rotations on the unit branch and B is the number of unique branches in the molecule. This, however, does not take into account overlap that will ultimately occur between other bonds in the molecule.

Overlap Detection

Before the entire branch is rotated during an iteration, the branch vector is first rotated and compared with the other branch vectors sharing the flexible point which have already been rotated for that iteration.. If there is overlap between any branch vector and the rotated branch vector, i.e. if they are parallel, then the iteration is discarded as being infeasible.

If the rotation angle divides the unit circle equally, then there will be

$$O_L = \sum_i^N R(b_i - 1)$$

overlapping bond rotations, where b_i is the number of bonds at flexible point i and R is the number of unit rotations. This is intuitive as for the ith flexible point, there will a total of $R * (b_i)$ overlaps.

Expected Number of Iterations

Combining the rotation lattice and branch numbers with the expected branch overlap yields a total iteration count of

$$R^B - \sum_i^N R(b_i - 1)$$

As an example, the unit branch rotated over a cubic lattice results in $R = 6$, $B = 2$, and $O_L = 6$. This gives a total of 30 unique rotations without branch overlap. Consequently, for the same molecule and a triangular rotation lattice, $R = 26$, $B = 2$, $O_L = 26$ giving 650 unique rotations.

Contact

The original author of the script can be contacted at tim@vanboxtel.ca with questions. The source code can be found at http://github.com/SunPowered/marathon.git