Hands-on workshop and Humboldt-Kolleg: Density-Functional Theory and Beyond - Basic Principles and Modern Insights

Isfahan, Iran, May 2 – May 13, 2016

Tutorial IV: Sampling conformational spaces
Manuscript for Exercise Problems

Prepared by Adriana Supady and Carsten Baldauf
Isfahan University of Technology
Isfahan, May 6, 2016
Introduction

This tutorial aims to familiarize you with the basic concepts of searches in molecular structure space.

The practice session consists of three parts:

Part A: Genetic algorithm search
- Problem I: Starting the first run
- Problem II: Analysis of the results

Part B: Pool of structures
- Problem III: Looking for duplicates
- Problem IV: Descriptive coordinates
- Problem V: Looking for duplicates - internal coordinates

Part C: And beyond
- Problem VI: Parameters of the GA search
- Problem VII: Alternative search techniques

Please start with Problem I and launch your genetic algorithm (GA) based search for alanine dipeptide. While waiting for the results, please proceed with Part B. Once your GA run is completed, you can work on Problem II. Part C is optional.

In the directory $HandsOn/tutorial_4/$, you can find all the files necessary for this tutorial. Please copy the contents of the skel/ folder into your own working directory. Dedicated folders have been prepared in the skel/ directory for each problem.

Practical notes

- There are more than 100 chemical file formats and some examples are presented in the Appendix in Figure 9. Openbabel\(^1\) is a program that can read and convert files of different formats. E.g. if you want to convert a geometry.in file to a XYZ-file, use:

  obabel -i fhiaims geometry.in -o xyz -O geometry.xyz

  where

  -i <format> specifies the format of the input
  -o <format> specifies the format of the output
  -O <filename> specifies the name of the output file

- You can use Jmol for visualizing 3D structures. Jmol recognizes a number of different chemical formats. You can measure bond lengths, bond angles and dihedral angles with Jmol: just double-click on the first atom and click the remaining ones.

- The atom ordering in a file matters. Most of the scripts/programs rely on the fact that the atoms are ordered in a consistent matter, especially when comparing structures!

- The database Berlin ab-initio amino acid DB\(^2\) provides structural data of 20 amino acids, bare and in cation-complexes [1]. You can use the data stored there for benchmarking!

\(^1\) [https://github.com/openbabel/openbabel](https://github.com/openbabel/openbabel)

\(^2\) [http://aminoaciddb.rz-berlin.mpg.de/](http://aminoaciddb.rz-berlin.mpg.de/)
Investigating molecular structures

Representing a chemical compound

Small organic molecules are often highly flexible and may adapt different 3D structures that differ in properties and energy. For methods like protein-ligand docking or catalyst design it is important to know the low-energy conformers of the molecule that build its molecular ensemble. To this end, the molecular conformational space needs to be sampled efficiently so that all relevant low-energy conformers are found.

Figure 1 depicts popular chemical representations of alanine dipeptide. The chemical formula stores only the composition of the compound. The simplified molecular-input line-entry system (SMILES) \[^2\] string is a convenient representation as it allows for encoding the connectivity, the bond order and the stereochemical information in a one-line notation. It should be noted, that a number of valid SMILES codes can be constructed for the same compound. The great advantage of the SMILES strings is the fact that they are intuitive and can be easily read and written.

<table>
<thead>
<tr>
<th>1D</th>
<th>Chemical formula</th>
<th>C₇H₁₁N₂O₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>SMILES</td>
<td>CC(=O)N<a href="C(=O)NC">C@H</a>C</td>
</tr>
<tr>
<td>3D</td>
<td>InChI</td>
<td>1S/C₇H₁₁N₂O₂/c1-4(6(10)7-3)8-5(2)9/h4H,1-3H3,(H,7,10)(H,8,9)/t4-/m0/s1</td>
</tr>
</tbody>
</table>

**Figure 1:** Alternative chemical representations on the example of alanine dipeptide.

SMILES codes can be used to generate a schematic, 2D representation of a molecule. Finally, the last missing piece of information, namely the spatial arrangement of atoms, is revealed in a 3D representation of a molecule. Two types of coordinates are commonly employed to represent a molecular 3D structure: Cartesian and internal coordinates. In Cartesian coordinates, each atom is represented as a point in 3D space. Cartesian coordinates are universal, intuitive and always relative to the origin of the coordinate system. The simplest internal coordinates are based on the 'Z-matrix coordinates' i.e. include bond lengths, bond angles as well as dihedral angles (torsions) (Figure 1). The main advantage of the internal coordinates is that they are orientation- and location-invariant, i.e. they remain unchanged upon translation and rotation in 3D space. In contrast, the dihedral angles are in most cases the only relevant degrees of freedom (DOF). Bond lengths and bond angles have usually only one minimum, i.e. the energy will increase rapidly if these parameters adopt non-optimal values. On the contrary, there is no single minimum for the value of a dihedral angle and in most cases, diverse values can be adopted. The adopted values depend on the neighboring atoms/functional groups and on the steric interactions within the conformation.

For the purpose of global structure search, only single, non-ring bonds between non-terminal atoms are considered as fully rotatable bonds after excluding bonds that are attached to methyl groups that carry three identical substituents. Further, the \textit{cis}/\textit{trans} nomenclature can be utilized to describe the relative orientation of functional groups within a molecule. In cases in which the functional groups are oriented in the same direction we refer to it as \textit{cis}, whereas, when the groups are oriented in opposite directions, we refer to it as \textit{trans}.

The full representation of a 3D structure is the list of its Cartesian coordinates. An alternative way to store 3D structures is to use a reduced representation that contains the SMILES and DOFs with the corresponding values. The difference between these two alternative representations is illustrated in Figure 2. The substantial advantage of the reduced representation is the fact that for a specified chemical compound, the only stored data are simultaneously the DOFs for the optimization. This is
Figure 2: The comparison of a full and reduced representation of 3D structure of the alanine dipeptide. The full representation contains all atomic Cartesian coordinates. The reduced representation consists of the SMILES string and dictionary of the rotatable bonds together with the corresponding values.

extremely convenient, especially for larger systems. Nevertheless, one should keep in mind that the reduced representation stores no information about the bond lengths and bond angles assuming no substantial changes of these coordinates.

Geometrical similarity of structures

The quantification of the molecular similarity is a common problem that needs to be solved, e.g. in order to remove duplicates from a pool of 3D structures. The most popular approach to quantify the similarity is the root-mean-square deviation RMSD, calculated for two sets of Cartesian coordinates.

Root-mean-square deviation (RMSD)  Given two 3D geometries of a compound with $N$ atoms, the formula for the RMSD is defined as follows:

$$\text{RMSD} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} d_i^2}$$

where $d_i$ is the distance between the corresponding atoms. Although fast to calculate, the RMSD value describes the similarity of two molecular conformations only after the best superposition of the geometries is identified. The most popular algorithm for finding the best alignment of two sets of coordinates is the Kabsch algorithm [3]. After translating the centroids of the two sets of coordinates to the center of the coordinate system, the Kabsch algorithm computes the optimal rotation matrix that minimizes the RMSD. Often only heavy atoms are considered in RMSD calculations. There are multiple advantages of using the Cartesian RMSD, e.g. it is a well-recognized metric, it is easy to calculate and reproduce and is available as a basic functionality in most of the modeling packages.

Torsional RMSD (tRMSD)  Instead of using Cartesian coordinates, the values of the significant torsional degrees of freedom, i.e. rotatable bonds, can be used. Analogically to the Cartesian RMSD, given two 3D geometries with $m$ rotatable bonds, the formula for $tRMSD$ reads:

$$\text{tRMSD} = \sqrt{\frac{1}{m} \sum_{i=1}^{m} \theta_i^2}$$

where $\theta_i$ is the angular difference between values of the corresponding dihedral angles. The calculation of the tRMSD is cheaper compared to the Cartesian RMSD. The value of the tRMSD is also easier to interpret. The major drawback of the tRMSD is the necessity to always provide a list of considered torsions in order to ensure reproducibility.
Characterizing the Potential Energy Surface

The potential energy surface (PES) describes the relation between the geometry and potential energy of a molecule [4] (Figure 3).

Figure 3: Schematic representation of a model PES representing the energy as a function of two coordinates.

Usually, a number of local minima with a global minimum among them exist on the PES of flexible, organic compounds. If the system is in a minimum, any small displacement will increase the potential energy. Each of PES minimum corresponds to a different 3D geometry. As a consequence, a variety of low-energy conformers can be adopted by flexible molecules. Another type of stationary point located on the PES is a first-order saddle point that corresponds to a transition state (TS) structure. A first-order saddle point is a maximum in exactly one direction and a minimum in all other directions.

In the following, we focus on methods for the global optimization, i.e., methods that can be used to find the global minimum on the PES.

Global optimization

The exploration of the high-dimensional PES is a complex task. The solution space is vast and thus it is generally infeasible to tackle the search problem in a deterministic way. A number of stochastic methods have been developed in order to efficiently sample the PES and to generate low-energy conformers. Some of the most popular methods are summarized in Table 1.

Table 1: Popular sampling approaches. Names of freely available programs are highlighted in boldface. Reprinted from [5].

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Implemented, e.g., in</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid-based</td>
<td>based on grids of selected Cartesian or internal coordinates (e.g., grids of different torsion</td>
<td>CAESAR [6], Open Babel [7], MOE [10]</td>
</tr>
<tr>
<td>rule/knowledge-based</td>
<td>use known (e.g., from experiments) structural preferences of compounds</td>
<td>ALFA [11], CONFECT [12], COS-MOS [15, 16], OMEGA [17]</td>
</tr>
<tr>
<td>population-based</td>
<td>improve candidate solutions in a guided search</td>
<td>Balloon [18], Cyndi [19]</td>
</tr>
<tr>
<td>metaheuristic</td>
<td>distance geometry based on a matrix with permitted distances between pairs of atoms</td>
<td>RDKit [20]</td>
</tr>
<tr>
<td>basin-hopping/minima</td>
<td>based on moves across the PES combined with local relaxation</td>
<td>ASE [23], GMIN [24], TINKER SCAN [25]</td>
</tr>
</tbody>
</table>

In this tutorial, we selected a genetic algorithm-based search for sampling the PES. The genetic algorithm (GA) [26, 27, 28] is a metaheuristic optimization method and belongs to family of evolutionary computing techniques. The concept of GA is to mimic evolution and follow the ‘survival of the fittest’ concept. The algorithm starts with a pool of random candidates for solutions. The best of the solutions are allowed to evolve while those unfavorable are removed from the pool. With this, the algorithm uses
the available information in order to explore promising candidates. GAs found numerous applications in the field of 3D structure prediction, e.g. (i) conformational searches for molecules, e.g. unbranched alkanes [29] or polypeptide folding [30]; (ii) molecular design [31]; (iii) protein-ligand docking [32, 33] (iv) cluster optimization [34, 35, 36, 37]; (v) predictions of crystal structures [38, 39]. In most of the applications, the fitness is a function of the total energy. In addition to that, an example of a GA, where the experimental information is included in the search process was suggested by Neiss and Schoos [40].

Fafoom - Flexible algorithm for optimization of molecules

Fafoom is a Python package designed for sampling the conformational space of organic molecules. It is implemented using Python 2.7 and employs the RDKit library [20]. RDKit is an open source collection of cheminformatics and machine-learning software distributed under the BSD license. Fafoom utilizes especially the 2D and 3D molecular operations implemented in RDKit.

Fafoom is distributed under the GNU Lesser GENERAL Public License [41] and is available from:

GitHub: https://github.com/adrianasupady/fafoom

Fafoom performs the global search based on user-curated selection of degrees of freedom and conducts the local optimization on Cartesian coordinates with an external software. The details about the structure of the package can be found in the Fafoom manual.

The following lists some important facts about Fafoom:

- Fafoom measures energy in eV and distances in Å.
- Fafoom comes with an internal verification of the generated 3D structures. A sensible 3D structure needs to fulfill two requirements: (i) the shortest distance between a pair of non-bonded atoms must be longer than a defined threshold (\textit{distance\_cutoff\_1}) and (ii) the longest distance between a pair of bonded atoms needs to be shorter that a chosen threshold (\textit{distance\_cutoff\_2}).
- There are two options in Fafoom that can be utilized to decide if 3D structures are similar or not:
  - Cartesian RMSD - if the RMSD exceeds a certain cutoff, the structures are considered to be different.
  - Degrees of freedom deviation (DOFd) - measure of the variation of the values of the DOFs. DOFd is a list with one value (\textit{True} or \textit{False}) per one type of degree of freedom. E.g. for the type ‘rotatable bond’ the list will store ‘True’ if the tRMSD value for the evaluated 3D structures does not exceed a certain cutoff. If the list contains at least one ‘False’, the two structures are considered to be different. In other words, a pair of similar structures is similar only if the values of all considered degrees of freedom are similar.
- The 3D structures are internally encoded as strings in structure-data format (SDF). The SDF is a combination of the MDL Mol format that stores the information about the atoms, bonds, connectivity and 3D coordinates with any associated data.
- Fafoom keeps track of the generated conformers by maintaining a blacklist. The \textit{blacklist} stores all structures that: (i) were subject to the local relaxation and (ii) resulted from converged local relaxations. The blacklist is consulted in order to evaluate the uniqueness of the newly generated structures.

Example of use: genetic algorithm based search

The main aim of the Fafoom package is to allow for performing genetic algorithm based searches for sampling the conformational space. The operating principle of the algorithm is given by the following
pseudocode (Algorithm 1):

```python
# initialization
while i < popsize:
    x = random_sensible_geometry
    if x is not in the blacklist:
        blacklist.append(x)
        x = local_relaxation(x)
        blacklist.append(x)
        population.append(x)
    i+=1

# iteration
while j < iterations:
    population.sort(index=energy)
    (parent1, parent2) = population.select_candidates(2)
    (child1, child2) = sensible_crossover(parent1, parent2)
    repeat
        (child1, child2) = mutation(child1, child2)
    until child1 and child2 are sensible and are not in the blacklist
    blacklist.append(child1, child2)
    (child1, child2) = local_relaxation(child1, child2)
    blacklist.append(child1, child2)
    population.append(child1, child2)
    population.sort(index=energy)
    population.delete_high_energy_candidates(2)
    if convergence criteria met:
        break
    else:
        j+=1

Algorithm 1: Genetic algorithm for sampling the conformational space of molecules.
```

**Initialization** The algorithm starts by generating a random and sensible 3D structure directly from the SMILES code. The distance geometry method implemented in RDKit is used to obtain the initial coordinates. This initial structure acts as a template for the following structures. During the initialization step, the DOFs are identified and located. In other words, number and location of rotatable bonds, cis/trans bonds and pyranose rings are determined. For each identified DOF type, a list of random values is generated. The following values are utilized: integers in the range $-179^\circ$ to $180^\circ$ (rotatable bonds), $0^\circ$ or $180^\circ$ (cis/trans bonds) and integers from the range 0 to 37 (pyranose rings), each corresponding to one of the 38 sugar puckers. The collection of the 38 puckers contains: two chairs, six boats, six skew-boats, 12 half-chairs and 12 envelopes [42, 43]. If the resulting 3D geometry is sensible and unique, local relaxation is performed.

A structure is unique if it is different from all structures stored in the blacklist. Once the local relaxation is completed, the values of the DOFs are updated and the structure is added to the population and to the blacklist. The procedure (i.e. generating and optimizing a random structure) is repeated until the intended population size $N$ is reached.

**Iteration** The objective function (energy) is optimized as the population evolves over subsequent generations. An iteration begins with assigning fitness, $F$, values to the population members. For each individual, $i$, holds:

$$F_i = \frac{E_{\text{max}} - E_i}{E_{\text{max}} - E_{\text{min}}}$$  \hspace{1cm} (3)

where:

- $E_{\text{min(max)}}$ the lowest (highest) energy among the energies of the structures in the current population
- $E_i$ energy of structure $i$

Consequently: $F = 1$ for the structure with the lowest energy (‘best’, i.e. most stable) and $F = 0$ for the structure with the highest energy (‘worst’, i.e. least stable and unlikely). In the case of populations
with little variation in the energies values, i.e. if $E_{max} - E_{min} < 0.001$ eV, all structures are assigned a fitness value $F = 1$. Once the fitness values are assigned, the genetic operations follow.

**Selection.** Three different mechanisms assigning different selection probabilities, $p$, to the structures are implemented:

- roulette wheel: $p_i = \frac{F_i}{\sum_{n=1}^{N} F_n}$
- reversed roulette wheel: $p_i = \frac{F_{N+1-i}}{\sum_{n=1}^{N} F_n}$
- uniform: $p_i = \frac{1}{N}$

Based on the probability values, two distinct structures are selected and are referred to as ‘parents’.

**Crossing-over.** The goal of the crossing-over is to exchange the information encoded by the parents. The crossing-over procedure is two-step: (i) the lists of values of the DOFs are combined (Figure 4) and (ii) the newly created lists of values are used to generate new structures (‘children’).

The structures generated by the crossing-over step must be sensible. Otherwise this step is repeated until sensible geometries are generated or a maximum number of attempts is reached. If no sensible geometries can be generated, the children are exact copies of the parents.

**Mutation.** A mutation is an operation that alters selected DOF values and results in changes in the 3D structure. The mutations are performed independently for each DOF type. Both children undergo mutation. The probability for a mutation and the maximum number of alterations can be controlled with parameters. The location and actual number of alterations is decided randomly. Mutation is only successful if the 3D structure created according to the alterations is sensible and unique. Otherwise the mutation is repeated until a sensible and unique structure is generated or a maximum number of attempts is exceeded.

**Local optimization and update.** The structures, that are created by the aforementioned genetic operations, are passed on to the external software for local optimization. Once the optimization is completed, the values of the DOFs are updated and the structures are added to the blacklist. After adding the newly optimized structures to the population, the two individuals with the highest energy are eliminated.

**Termination** After a minimum number of iterations, the convergence of the algorithm is evaluated. The algorithm terminates if at least one of the following criteria is met:

- the lowest energy has not changed by more than a defined threshold during a defined number of iterations
- the lowest energy reached a defined value
- maximum number of iteration is exceeded

**Parameters** The Fafoom parameters can be assigned to three groups: (i) molecular settings, (ii) run settings and (iii) GA settings. The list of all parameters (together with descriptions and defaults) can be found in the Fafoom manual.
Part A: Genetic algorithm search

In the first part of this tutorial you will use Fafoom and run a genetic algorithm based search for alanine dipeptide. The structure of the molecule is depicted in Figure 5:

![Alanine dipeptide](image)

**Figure 5: Alanine dipeptide**

Problem I: Starting the first run

In the directory `skel/problem_1/` you will find files needed to run your first GA search. The `ga.py` script is an implementation of Algorithm 1. The `parameters.txt` file lists all needed parameters. Figure 6 shows its contents and highlights some important parameters:

![Parameters.txt content](image)

**Figure 6: Contents of parameters.txt**
The energy calculations will be run externally with FHI-aims. Make sure that the `aims_call` in the `parameters.txt` is correct. The directory `adds` should contain the `control.in` file that will be used for the FHI-aims relaxations. You will find a template for `control.in` file there.

Modify the provided template for the `control.in` file:

- Add the missing xc-type option: `pbe`
- Add the tag: `vdw_correction_hirshfeld`
- Append the 'light' species defaults for the following atoms: C, H, N and O.

The GA and run settings in the `parameters.txt` are selected in such way that a minimal GA run can be conducted: `popsize=3` and `max_iter=3`. Given these settings, the search will start with an initial population of 3 random structures and 3 generations will follow. In each generation, two new candidate structures are built. Within this minimal GA run, a total of $3 + 2 \cdot 3 = 9$ DFT relaxations will be conducted one after each other.

You can start your GA run with:

```
python ga.py parameters.txt
```

The `output.txt` file will be generated immediately and completed during the run. You can check the progress of the GA run with typing:

```
tail -f output.txt
```

In the `output.txt` you will find information about the molecule and the identified degrees of freedom that will be optimized during the run.

**Attention!!**

- Fafoom counts the atoms starting from 0.
- Fafoom selects one possible torsion representation (see Figure 7) and doesn’t change it.

![Figure 7: A dihedral angle can be usually represented in multiple ways. It’s common to select heavy atoms for the representation.](image)

The GA search will last for 1-2 hours. Please proceed to and complete Part B.

**Problem II: Analysis of the results**

Once your GA run is finished, or a least few generations are completed you can take look at the results. As you already saw in Part B, alanine dipeptide can adopt a number of stable conformers. In order to
find most of them, a number of independent GA runs (at least 5) with bigger population size (5-10) and number of iterations (20-30) would be needed.

However, the short GA run that you’ve conducted should have found at least few alanine dipeptide conformers. Take a look at your results:

- each of the folders initial\_* and generation\_* contains files of a single FHI-aims run: control.in, geometry.in, result.out and geometry.out that stores the optimized geometry in FHI-AIMS format.
- mol.sdf - that’s the template 3D structure, that was generated directly from SMILES code by RDKit and used to construct all subsequent geometries.
- backup_* files can be used to restart your calculation.
- output.txt is a log-file of the GA run.

The data that you worked with in Part B are our ’reference data for the alanine dipeptide’.

1. Did your GA run find the reference global minimum? (Don’t worry if it didn’t: with such a short GA the chance is below 20%).

2. How do the energies of the conformers evolve during the GA? Plot the energies of the conformers (relative to the reference global minimum) as a function of the GA stage.

(Instead of analyzing your own results, you can analyze the results of either a short or a long GA, both provided in the solutions/problem_1.)

One way to find the answer to question 1. is to: (i) find out which of the structures of your GA has the lowest energy and (ii) visually inspect it (measure the $\phi$ and $\psi$ angles!). You might remember, that the global minimum of alanine dipeptide is the $C7_{eq}$ conformer (see Figure 8).

In the directory utilities you will find the script get_ga_energy.py. You can run it with:

```
python get_ga_energy.py ga_run_directory
```

The script will plot the energy of the structures optimized in course of the analyzed GA run. The energy is relative to the energy of the global minimum known from the reference data.

Apart from finding the global minimum, a GA run should find a number of local minima. In order to find out, how many uniques structures did your GA runs find:

Write a Python script that:

- Converts the relaxed geometries from FHI-AIMS format to XYZ format.
- Assigns indexes to the resulting structures and stores them in dedicated folder.
- Creates an energy.txt file: in form 'index energy'.

In the directory utilities you will find the script get_ga_structures.py. You can run it with:

```
python get_ga_structures.py ga_run_directory
```

As a result, a new directory all_structures and a energy.txt file will be created. You already know how to remove the duplicates from a pool of structures. Are there duplicates among the structures your GA run found?
Part B: Pool of structures

Problem III: Looking for duplicates

A common problem in analyzing the results of conformational searches is that the resulting pool of structures contain duplicates. Removing the duplicates significantly simplifies the analysis of the data. It is advisable to use a similarity criterion to remove the duplicates from the structure set. Following steps are needed for the duplicate removal:

1. Decide for similarity descriptor: Cartesian RMSD or torsional RMSD.
2. Decide on the geometrical similarity threshold and energy threshold.
3. Start with the structure with the lowest energy.
4. If a geometrically similar structure is find, check the energy difference. Remove the structure from the pool if the energy difference is below a chosen energy threshold.
5. Continue with the structure with the next lowest energy.

In the directory skel/problem_3/ you will find a tar archive all_structures.tar containing 1226 structures of alanine dipeptide and a list of energy values energy.txt. The structures were obtained from 20 independent GA runs. The energy was calculated at the \( PBE + \text{vdW} \) level utilizing light species defaults. You can unpack the tar archive with:

\[
tar -xf all_structures.tar
\]

In the utilities directory you will find a Python script plot_hierarchy.py. This script works with any energy list in the format: index (column 1) and energy value (column 2). In order to plot the energy hierarchy of the pool of structures type:

\[
\text{python plot_hierarchy.py energy.txt}
\]

How does the obtained energy hierarchy look like?

In the utilities directory you will find a Python script cartesian_rmsd.py that calculates the Cartesian RMSD for a pair of structures. In order to calculate the Cartesian RMSD between geometry_1.xyz and geometry_2.xyz, type the following:

\[
\text{python cartesian_rmsd.py geometry_1.xyz geometry_2.xyz True True}
\]

Apart from the files names, there are two additional parameters that can only be True or False. The first parameter, removeHs, gives the possibility to remove the hydrogens before calculating the Cartesian RMSD, i.e. only heavy atoms will be considered. This is often enough to describe the similarity and generally allows for saving time. Nevertheless, in systems, where hydrogen bonding plays an important role, it is advisable to keep the hydrogens upon Cartesian RMSD calculation.

The second parameter, chiral, tells the script whether the molecule should be treated as chiral or not.

Chirality is a molecular property.

- A chiral molecule and its mirror image are not superimposable and need to be considered as different. The molecule and its mirror image are enantiomers. One them is 'D' and the other 'L'. All amino acids, apart from glycine are chiral molecules and are usually 'L'.
- Achiral molecules are identical with corresponding mirror images.

If the molecule is not chiral ('False'), the calculation of the RMSD is performed twice: for the target and the reference and for the target and the mirror of the reference. The lower of the resulting values is then considered.

Now that you know how to calculate the Cartesian RMSD for a pair of structures, you can start to work on removing the duplicates from the pool of structures.
Write a Python script that:

- Reads in the energy values and the corresponding indexes from the `energy.txt` file.
- Sorts the indexes accordingly to the energies values.
- Creates a directory for the unique conformers.
- Starting from the structure with the lowest energy checks, if a similar structure is already present in a directory with unique conformers. If the structure is new, it is copied to the directory with unique conformers. If the structure is already known, it will be ignored.
- Continues until all structures have been evaluated.

In the directory `utilities` you will find a Python script `remove_duplicates.py` that implements the listed steps. You can perform the duplicate removal with typing:

```
python remove_duplicates.py all_structures 0.15 0.015
```

`all_structures` is the directory with the structures before the duplicate removal. 0.15 is the geometrical similarity threshold, i.e. if the Cartesian RMSD of a pair of structures is higher than 0.15, the structures are considered to be different. Further, 0.015 is the energy threshold, i.e. if two structures are different in energy by more than 0.015 eV, they need to be treated as different.

After ca. 1 minute, a new directory `duplicates_removed` containing 45 structures and a new file `energy_final.txt` will be created. The number of the structures left after the duplicate removal depends on the utilized thresholds. It is therefore important to carefully chose the geometrical similarity threshold, so that: (i) it is big enough to allow minor displacements in the atomic positions and (ii) it is low enough to correctly capture structural differences describing two different conformers. It is usually advisable to calculate the RMSD for a number of structure pairs and visually inspect the corresponding structural differences beforehand.

Note that a further verification would be needed to check, if all of the obtained structures are true local minima of the underlying model PES (See Tutorial I, Problem V).

Problem IV: Descriptive coordinates

Please copy the directory with structures left after the duplicate removal and the corresponding energy file to the directory `skel/problem_4`.

In the directory `utilities` you will find a script `measure_dihedral.py` that measures a selected dihedral angle. Try the following to measure the dihedral angle C-C-N-C defined with the atom indexes 1, 2, 4 and 5 in `geometry_1.xyz`:

```
python measure_dihedral.py geometry_1.xyz 1 2 4 5
```

The dihedral angle C-C-N-C (1-2-4-5) is one of the two peptide bonds in alanine dipeptide (Figure 8). Do you think that the peptide bond shows preference for certain values?

Write a Python script that:

- Measures the values of the two peptide bonds for all structures left after the duplicate removal.
- Plot the results as histograms.

In the directory `utilities` you will find the script `get_omegas.py`. You can run it with:

```
python get_omegas.py duplicates_removed/
```

Apart from the peptide bond (commonly denoted as \( \omega \)), peptides feature two further important backbone dihedral angles: \( \phi \) and \( \psi \) (see Figure 8).
Figure 8: The global minimum of alanine dipeptide: $C7_{eq}$ conformer. The atoms are annotated with the corresponding indexes.

Write a Python script that:
- Measures the values of the $\phi$ and $\psi$ angles.
- Makes a $\phi$-$\psi$ plot.

In the directory utilities you will find the script get_psiphi.py. You can run it with:

```python
get_psiphi.py duplicates_removed/energy_final.txt
```

The script will produce a scatter plot $\phi$-$\psi$ and color the circle accordingly (dark blue - low energy, brown - high energy). Are the circles randomly placed?

**Problem V: Looking for duplicates - internal coordinates**

In the previous problem we investigated the preferences of the dihedral angles of alanine dipeptide. Do you think it is enough to use the values of the dihedral angles instead of fitting Cartesian coordinates to describe similarity of two structures?

In Problem III we used the Cartesian RMSD to compare the structures and remove duplicates. Here, we will use the tRMSD to compare the structures and remove duplicates. We will start with selecting the dihedral angles and storing the corresponding indexes in a text file.

First, prepare a `dihedrals.txt` file that lists the indexes of the relevant dihedral angles and save it in the `problem_5` directory. It should look as follows:

```
1 2 4 5
5 6 8 9
2 4 5 6
4 5 6 8
```

These are the indexes of the two peptide bonds and the $\phi$ and $\psi$ angles that you studied in Problem IV.

Write a Python script that, for a pair of structures:
- Measures the values of the selected dihedral angles.
- Returns the tRMSD.

In the directory utilities you will find the script trmsd.py. You can run it with:

```python
trmsd.py geometry_1.xyz geometry_2.xyz dihedrals.txt True
```
The last parameter accounts, similarly as in the case of `cartesian_rmsd.py`, for the chirality. The tRMSD is implemented in such a way, that it adopts values from the $[0, 1]$ range and in units of $\pi$. tRMSD equals 0 for a pair of identical structures. A value equal to 1 $\pi$ corresponds to ‘maximally’ different structures. In the studied cases, it would mean that each dihedral angle differs by $180^\circ$ within a structure pair.

To give an example, a tRMSD value equal $0.1\pi$ corresponds to: (i) a change of $18^\circ$ per each of the 4 dihedral angles or (ii) a $36^\circ$ change of a single dihedral angle.

We can now use the new similarity descriptor to repeat the duplicate removal.

```
Adjust the `remove_duplicates.py` to use the tRMSD instead of Cartesian RMSD.
```

In the directory `utilities` you will find the script `remove_duplicates_tmrspd.py`. You can run it with:
```
python remove_duplicates_tmrspd.py all_structures 0.1 0.015
```

How different is the results from the duplicate removal that you’ve performed in Problem III?

**Part C: And beyond**

**Problem VI: Parameters of the GA search**

The `parameters.txt` file contains a number of diverse parameters. What would be the consequences of:

- setting `rmsd_cutoff_uniq=1.0`, or
- setting `rmsd_cutoff_uniq=0.0`, or
- setting `prob_for_mut_torsion=0.0` and `prob_for_mut_cistrans=0.0`

Check the enclosed Fafoom manual for more information.

**Problem VII: Alternative search techniques**

In the first part of the GA search a number of random structures is generated. Which parameters should be changed in order to run a entirely random search instead of a GA search?

**Problem VIII: GA searches for other dipeptides**

If you want to run a GA search for another amino acid dipeptide, you need to: (i) adjust the `smiles` keyword in the `parameters.txt` file and (ii) if needed, append the missing species defaults to the `control.in` file in the `adds` directory.

You can obtain SMILES codes from chemical databases or by converting a file with a 3D structure with Openbabel:
```
obabel -i xyz geometry.xyz -o smi -O geometry.smi
```

**Problem IX: Clustering of structures**

In Problem III and Problem V you’ve performed a duplicate removal from a pool of structures. Depending on the total number of structures and the character of the task, removing duplicates might be not enough. Often, it is desirable to group structures that are geometrically similar to clusters. Can you reuse `duplicate_removal_trmsd.py` and create `cluster_trmsd.py` in such way that divides the pool of structures into four groups depending on the character of the peptide bond? (group1: (trans, trans), group2: (trans, cis), group3: (cis, trans), group4: (cis, cis))

**Acknowledgments**

We would like to thank the testers of this tutorial for their time and feedback.
Appendix

Chemical formats

**FHI-AIMS**

<table>
<thead>
<tr>
<th>Atom</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Charge</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>0.5646</td>
<td>0.1757</td>
<td>1.4286</td>
<td>O</td>
</tr>
<tr>
<td>H</td>
<td>1.5326</td>
<td>0.2062</td>
<td>1.3936</td>
<td>H</td>
</tr>
<tr>
<td>H</td>
<td>0.2858</td>
<td>0.7863</td>
<td>0.7297</td>
<td>H</td>
</tr>
</tbody>
</table>

**XYZ**

```
3
O  0.5646  0.1757  1.4286
H  1.5326  0.2062  1.3936
H  0.2858  0.7863  0.7297
```

**SDF**

```
water
OpenBabel04141614133D
3  2  0  0  0  0  0  0  0  0  0999 V2000
0.5647  0.1757  1.4286 O   0  0  0  0  0  0  0  0  0  0  0  0
1.5326  0.2062  1.3936 H   0  0  0  0  0  0  0  0  0  0  0  0
0.2858  0.7863  0.7297 H   0  0  0  0  0  0  0  0  0  0  0  0
2  1  1  0  0  0
3  1  1  0  0  0
M END
$$$$
```

Figure 9: Different chemical file formats of water.

References


