Genetic Algorithm

Molecular Statistic

2013

1 Introduction

This is the exam project in the course molecular statistics which involves the genetic algorithm for energy minimization. The purpose of this exam project is to do a energy minimization of long alkane molecules using an algorithm called the genetic algorithm which is a flavor of the general Monte-Carlo approach with a force field package called MMTK.

1.1 Force Field and Dihedral angle

For this project we will be using force field theory to do the energy minimization, which means the energy of a curtain molecule configuration is calculated by eq. 2.5 from Jensen;

$$E_{molecule} = \sum_{i}^{bonds} k_i(r_i - r_{i,e})^2 + \sum_{i}^{angles} k_i(\sigma_i - \sigma_{i,e})^2 + \sum_{i}^{dihedrals} V_i[1 \pm \cos(n_i\omega_i)] + \sum_{i>j}^{atompairs} \left(-\frac{A_iA_j}{r_{ij}^6} + \frac{B_iB_j}{r^12_{ij}} + \frac{q_iq_j}{r_{ij}} \right) \tag{1.1}$$

The force field calculation will be done by the Amber 94 forcefield using a python package called MMTK.

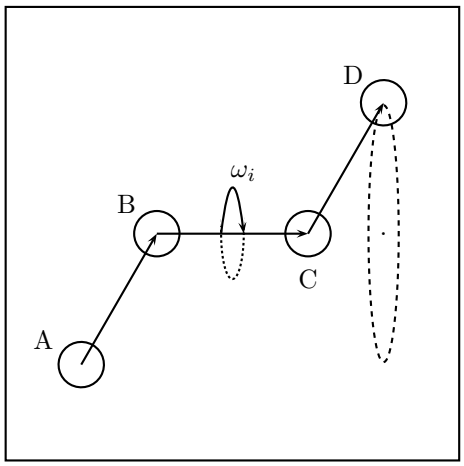

Figure 1.1: Illustrating the dihedral angle ω_i between 3 bonds and 4 carbon centers, A, B, C and D.

In geometry, a dihedral or torsion angle is the angle between two planes, and in our alkane case it is the angle between two CH_2 groups, as seen in figure ??. We want to change the angles in such a way that it finds the minimal energy from a dihedral configuration Ω. The configuration Ω is then defined as a vector of the systems dihedral angles ω_i beween each of the successive CH_2 groups, $\Omega_\beta = \{\omega_1, \omega_2, ..., \omega_N\}$. Each ω_i is assigned an initial random value taken from a uniform distribution from 0 to 360 degrees. These dihedral angles can be seen as the genetic code for the alkanes and by modifying these angles we can minimize the molecules geometry with respect to the total molecule energy..

1.2 Genetic Algorithm

The genetic algorithm is an optimization algorithm that is based on genetic inheritance, hence the name. The genetic algorithm is an efficient algorithm to search through conformational space and locate minima. Here we employ the genetic algorithm to try and locate the lowest energy conformation for an alkane. As described before we define the geometry of an alkane chain through its dihedral angles ω_i between each of the successive CH_2 groups, and so by modifying these angles we change the structure and thereby the energy.

We start out by defining a state vector Ω_β as $\Omega_\beta = \{\omega_1, \omega_2, ..., \omega_N\}$ where we have N dihedral genetic angels. Each ω_i is assigned an initial random value taken from a uniform distribution from 0 to 360 degrees. This can easily be done using numpy, for N dihedral angles;

```
state_vector = np.random.uniform(0.0, 360.0, N)
```

We then create a set of K state vectors. For each of these state vectors, we define a corresponding energy of the molecule $E_\beta(\Omega_\beta)$. The energy is obtained after a short *Conjugate Gradient minimization*[1]. From now on, these K state vectors are referred to as *parents*.

The parents are then changed as a result from the following algorithm steps;

1. **Mating.**
   Two parents Ω_β and Ω_α of length N can combined to give birth to two children with the "genome" of the parents represented by the its state vector. The state vector Ω_β is then being split from the first M elements into the first child, and $(M - N)$ into the next child. Same procedure for parent Ω_α. The cut index M is to be determined randomly.

2. **Mutation.**
   After the children has been born, sometimes, a random mutation is inserted in their 'genome'. This means randomly select one of the N angles ω_i and assign it a new random value, to a probablity of a mutation rate, `mutatation_rate`.

3. **Evolution.**
   Survival of the fittest. The energy of the formed child is evaluated like the parents, minimize the geometry of its state vector. If the energy if the child is lower than one of the parents, then the child is kept and the parent with the larger energy is removed, where the child takes its parents place. If the energy of the child is larger than a parent, there is still a probability P that it survives the evolution step. This is where the Monte Carlo enters the algorithm. P is proportional to the Boltzmann factor, i.e.

   $$P = e^{-\Delta E/(k_B T)} \tag{1.2}$$

   where ΔE is the energy difference between the child and the parent, and T is the temperature of the system. In the following we set $k_B = 1$, so all temperatures are in units of k_B.

   When *one* child has replaced a parent, we consider this to be the end of the generation.[2] This means that no more mating of parents should be carried out and we start over.

[1] This will be explained in the setup section
[2] This is where the `break` command comes in handy in a for-loop

2 Assignment

The main task of this assignment is to implement the genetic algorithm, and minimize the structure of three alcane chains, $C_{10}H_{22}$, $C_{20}H_{42}$ and $C_{40}H_{82}$,

2.1 Setup

To successfully do this assignment you need to install some software dependencies on your computer. This means downloading `install_mmtk.sh` from the project folder (along with rest of the files). Use `cd` to find the file and run the installation script like this;

```
sudo sh install_mmtk.sh
```

in the terminal.

Remember to download the rest of the files from the absalon page.

2.2 Examples

Make sure you have `minimizers.py`, `molecule.py`, `dbutane.py`, `ddecane.py`, `sdicosane.py` and `dtetracontane.py` in the same folder as the python file you working in. To use the functionality provided by these packages you import them in the beginning of your python file:

```
from molecule import Molecule
from minimizers import conjugateGradient
from sdbutane import setDihedral
```

Where `sdbutane` is imported when working on the butane alkane chain.

To do a energy calculation of a random dihedral state of Butane, you use the modules in the following way.

```
m = Molecule('butane')
no_dihedral = 1
dihedral_list = np.random.uniform(0.0, 360.0, no_dihedral)

setDihedral(m, dihedral_list)
conjugateGradient(m)
energy =  m.getEnergy()
```

The above code might seem strange as it is using 'object orientated' coding which we did not cover in this course. On line 1 the molecule `m` is defined, and on line 5 the dihedral angles of the molecule is set which energy is calculated on line 6 and 7.

The function `conjugateGradient` is where the forcefield minimization happens. This function finds the nearest local minimal a stops, where after it is possible to get the energy from the molecule. You need to run the functions in that order for it to work.

The above code is the only code you need to solve this assignment. The rest can be implemented by you.

If you want to see how a molecule looks, use code:

```
m.saveXYZ("my_molecule.xyz")
```

You can open the my_molecule.xyz file in a program called Avogadro by typing "avogadro my_molecule.xyz" in the command line. See the example in file example_savebutane.py.

2.3 Simulation Setup

Before you finish the actual simulation you need to create the functions that simulates the system. To make it easy to start we have split the setup into small manageable steps.

1. Use the code from `example_butane.py` and familiarise yourself with it.
2. Create a list of angles from 0.0 to 180.0 degrees and calculate the energy of Butane for each angle. Plot the result.
3. Implement the following optimization algorithm, called *Greedy optimization*:
   (a) Create a random list of dihedral angles
   (b) Create a integer `no_generations` which represents how long the algorithm will go on.
   (c) Create a for-loop and loop over the no of generations. For each loop generate a random dihedral state and calculate the energy.
   (d) If the energy is lower than the previous, save the new energy and the new state, otherwise discard new state the away and continue the search.
4. Create a function that takes `m` and `dihedral_list` as parameters and returns the energy of the configuration.
5. Create a function that takes the parameters `parent_alpha`, `parent_beta` and `mutation_rate`. Have this function mate the two parents and create two children based on the genetic algorithm step 1 and 2. Use numpy's `ranint` generate a random cut index M

```
m = np.random.ranint(0, N)
```

6. Create two lists, one for the containing the state vectors and one containing the energy related to the state vector. Fill them up with `no_parents` of random dihedral states, and calculate the energy for each.
7. Finish the algorithm by creating a for-loop and loop over number of generations defined, mating each parent pair, selecting what parent and what child survives based on the Genetic algorithm.

2.4 Simulations

Now that you have a working simulation, it is time to do the actual simulations:

- **Simulation 1**
  Minimize $C_{10}H_{22}$, $C_{20}H_{42}$ and $C_{40}H_{82}$ and plot the mean energy for each generation.
- **Simulation 2**
  Check for correlation between temperature T and the energy found after G generation for each molecule. Check for temperatures between 0.5 and 10.0.
- **Simulation 3**
  Check for correlation between mutation rate in the interval [0:0.5] and the energy found after G generation for each molecule.
- **Simulation 4**
  Compare the Greedy and Genetic algorithm by plotting the energy vs generation.
- **Simulation 5**
  Check the population size K of parents. Plot the number of generations it took to converge as a function of the number of K parents.
- **Simulation 6**
  Make the Genetic Algorithm better. You have to come up with one changes to the mating and mutation routine. Change the algorithm to make your own personal minimization algorithm. Document the results.