Explanation Molecular Mechanics Algorithm

Written by Filip Aleksic 10754830

This algorithm contains 9 different modules. I will explain them briefly here in the order at which they are used.

1. *Main.f90*: Calls the different functions and subroutines to be used
2. *Types.f90*: Contains all the types and variables used in the algorithm.
3. *Input.f90*: Reads the input xyz file and stores all atoms in the Atom type. Also contains a function that outputs the final result after minimization.
4. *Distance.f90*: Contains 2 functions which allow for distance calculations between 2 points in xyz space. Relative distance avoids usage of the sqrt intrinsic function to increase performance slightly.
5. *Init.f90*: Establishes the bonds of the different atoms. If the distance between 2 atoms is below a certain bond threshold, append it to the “Bond” list of each atom type. This results in carbon atoms having a “Bond” list of size 4, and hydrogen of size 1. The main way I store bonding information is through the use of pointers which point to the main atom array. So for example: Atoms(1)%Bond(1)%p points to the atom that the atom at position 1 of Atoms is bonded to. If this atom was a carbon, the Bond list would have multiple entries so the second bond would be Atoms(1)%Bond(2)%p etc.
6. *Test.f90*: Contains 2 subroutines that can be used for bug fixing. Test\_Bond is called as it is especially useful. It checks if carbons have 4 bonds, and hydrogens have 1. If they have more (or less) something clearly went wrong and needs fixing.
7. *Bondgroups.f90*: Contains several subroutines and functions all used in determining bond chains of 3 and 4 atoms. Initially, every atom in the atom list is iterated and ran through the Find\_Bonding subroutine. This subroutine finds chains of 3 and 4 atoms and stores them in the Bonding type using a simple appending subroutine called Push\_Back\_Bonding. The bonding type is a 2d array, where the different rows are different chains. So Chain\_T(1,:) contains 3 pointers pointing to 3 atoms in a chain of bonds. Chain\_T(2, :) points to another 3 atoms in a chain of bonds, etc. After all chains have been found, the duplicates are deleted using the Remove\_Duplicates subroutine. Using the associated intrinsic function which returns true if 2 pointers point to the same point in memory, we can efficiently check if 2 chains are the same. The associated function is called in the Check\_Association function.
8. *Minimize.f90*: This module has all the functions and subroutines used in the Metropolis minimization. New\_Positions picks a random atom, and moves it by a random amount. The random number between -1 and 1 is generated by the random function. To terminate the metropolis minimization I use a lastChange value. If a new record minimum energy is found, it is stored in EnergyMin. The coordinates are stored in Atoms\_Min. If the algorithm does not find a new minimum for 5000 iterations, we assume this minimum is the absolute minimum and terminate.
9. *Energy.f90*: Contains all the functions to calculate the energy of a given molecule. Straight forward use of the formulas in the assignment, however some things were changed. In the stretch function, to avoid counting both AB and BA stretch I simply divide every stretch by 0.5. This will nullify the effect of double counting. The angles between chains of 3 and 4 atoms are calculated differently as well. I use the Angle function, which creates 2 vectors. In the case of ABC we create vector1 = and vector2 = . For ABCD vector1 becomes and vector2 . The angle in radians is then calculated with the following formula.

This approach is used for both the bending and torsional energy.

Results of my algorithm give me the following average data for methane and n-butane over 5 runs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Molecule | Energy Start (Kcal) | Energy End (Kcal) | Number of Iterations | Runtime (s) |
| Methane | 1.30522 | 1.29293 | 103423 | 3.419 |
| n-Butane | -2.20380 | -2.78844 | 118535 | 0.525 |

The results seem to imply that the minimum last change value for termination is set too high. However, with accuracy in mind I do not mind giving up some performance to guarantee a minimum is reached when runtimes are still extremely short. If larger molecules were to be tested, the minimum termination point should be adjusted as at that point the runtime will become prohibitively slow.