Glycine project 1

1. Description of the files
   There are two files: one just contains the xyz coordinates of 8 minima (8 conformers), and the other larger file contains the xyz coordinates of the data set.

   Both files use the standard xyz format. For each point, the first line is the number of atoms in the molecule. The second line is a comment line so lots of different things could be written there; for us this line usually is the electronic energy (-284.xxx in atomic units for glycine). The following lines are coordinates. The first column are atomic symbols; the next three columns are x, y, and z coordinates; if there are more columns, they could be gradients or other things, but you don't need those.

2. Computing inter-nuclear distances (Heavy atoms only)
   If we just focus on the heavy atoms, conformers {1, 4, 6, 8} look pretty much the same, and so do {2, 3, 5, 7}. So probably you can find whether a configuration is more similar to {1,4,6,8} or {2,3,5,7}, but you cannot tell which particular conformer is the closest to that configuration. I think this serves as your first "homework" project in this group, and you'll gain some experience writing codes, dealing with molecules, etc.

   The first thing to do is to check the order of the two O atoms. In the small file, the O in the CO double bond always comes first, and then the O in the CO single bond. In the larger file, I think this is still true in most cases, but we cannot guarantee. I think the easiest thing to do is to compute the distance between atoms 8 and 10, and also the distance between atoms 9 and 10. If the former is larger, the order is correct. If the former distance is shorter, you'll need to permute atoms 8 and 9 first before calculating the 10 distances.

   After correcting the order of the two O atoms, you can calculate the 10 distances between the 5 heavy atoms. You'll arrange these 10 distances into a vector using a specific order you prefer, for example, $[r_{12}, r_{13}, r_{14}, r_{15}, r_{23}, r_{24}, r_{25}, r_{34}, r_{35}, r_{45}]$.

   For each of the 8 conformers, you'll get such a vector; let's denote these vectors as $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, ... , $\boldsymbol{u}_8$. Then for each point in the large file, you'll get a vector, and let's call these $\boldsymbol{v}_k$ (k=1,2,...,52334). For each $\boldsymbol{v}_k$, you compute $|\boldsymbol{v}_k - \boldsymbol{u}_1|$, $|\boldsymbol{v}_k - \boldsymbol{u}_2|$, …, $|\boldsymbol{v}_k - \boldsymbol{u}_8|$, and rank these 8 values from the smallest to largest. A small $|\boldsymbol{v}_k - \boldsymbol{u}_j|$ means that point *k* in the large data set is closest to conformer *j*, and therefore point *k* is assigned to the cluster of conformer *j*.

   At this stage, it's more like mathematics rather than machine learning. You don't need any machine learning package in Python; all you need probably is Numpy. There are lots of tutorials on this if you Google it, and if you have any questions, just send me an email.