12月18日 7th Workshop　　山田，古賀

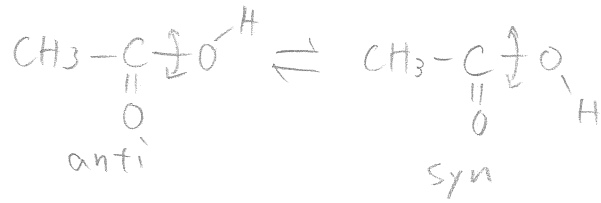
◦ クーロン相互作用による、QM/MM計算のQM領域に対する電荷割当て法

{ A) 適用例
  B) プログラムへの組込み

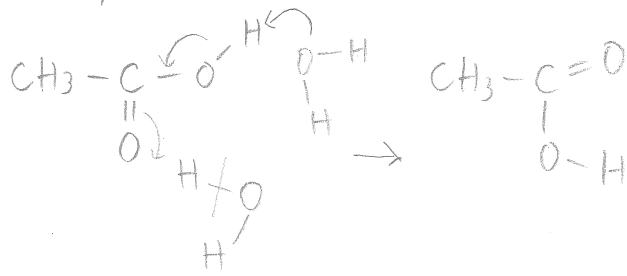
A) 適用例について

• 酢酸の異性化反応（syn ⇄ anti）

1) C-O軸に関する分子内回転



2) 周りの水分子との間におけるプロトン移動



1) について

予想より低温度で回転が起こった

※ きちんとした平衡化は行っていないデータを用いたdynamicsの結果

☺ 何らかの統計量をみることが目的でないので、平衡化の必要はない

◦ QM/MM-MD計算の条件

・NVTアンサンブル　　・周期境界条件(有)

・1000K → 1000K　　・0.5fs × 20000step = 10ps

~~O-H振動は、この回転に寄与するから~~

・QM領域；酢酸1分子　　・MM領域；TIP3P水491分子

・QM計算；RHF/4-31G*

2) について

12/17 に確認した範囲では、上記の反応は起こっていなかった

☆ 平衡化を以下の計算条件で行った

○ 計算条件 / 平衡化計算　2段階で行った

I
- NVT アンサンブル　、周期境界条件(有)
- 500K → 300K　、1fs × 40000 step = 40ps
- QM領域はBellyによって凍らせた　、MM領域：TIP3P 490分子（SHAKE & RATTLE）
- QM領域；酢酸1分子 + 水1分子（カルボニル酸素の近傍に存在）
- QM計算（電子状態計算）は　MP2/6-31+G(d,p)
  [Bellyのときは、HFで充分かもしれない]

II
- ほとんど条件は同じで、Bellyによる固定を解除した
- 300K → 300K　、0.2fs × 10000 step = 2ps
  ①O-H振動を考慮するから

I；MM領域の平衡化

II；系全体の平衡化　← 反応が起こらないように注意が必要

※ I → II のときに、座標だけを受け渡した。
速度は、IIにおいて新しくMaxwell分布に従うように定めた。

※ 上記したように、Iでは HF 計算で充分ではないかと思う。
必要となる時間を考慮すると、特に。
MP2では、2週間近くかかった（ような気がする）

本ページ冒頭で「起こっていなかった」と書いたが、実際には「起こりそうになかった」
→ 300Kで平衡化したが、電荷のダイナミクスをみたいだけなので（とりあえずは）
800KでMD計算を走らせている。

☆ 問題点　長時間(?)走らせてみて分かったこと

PrtGVal.exe (Gaussianデータを吐き出す実行ファイル) においてエラーが出ることが原因で、Roarが異常終了することがしばしばある。

エラー内容は、Gaussianにより生成されるRead-Write Fileに「QM⇒MMの力が書き込まれてない」というもの。

しかし、このエラーが出るまでと座標以外は同じインプットなので、「RWFにその力を残せ」という命令は必ず出ている。

とこで、考えうる原因を列挙してみる(未確認)

- Gaussianは、短時間に何回も実行するように設計されてはいないので、code的に何らかの欠陥がある。
  ⟹ RWFは計算ごとに名前を変えており、この可能性は低いだろう

(~~- Roarの中で、Gaussianインプットファイルの座標以外は、文字列としてメモリ上に保管している。Roar内である条件を満たしたときに、ある手続きがその文字列のメモリ空間を壊してしまう~~)

- このRoar+Gaussianは1つの計算機上で行っているのではなく、NFSマウントされたHDD上にRoarプログラムがあり、Gaussianとのやりとりは、逐一ファイルに書き出して行っているので、データの通信量がバンバン多い。恐らく、RWF → PrtGVal.exeと同時に、若しくは重なる時間があるときに、pcc xx (NFSクライアント) ⇨ NFSサーバで大きな通信があると、RWFのデータが途中で切られてPrtGVal.exeに渡されるのではないか。

B) プログラムへの組込みについて

移植先 3) Roar 2.1 (Amber7) 4) Sander (Amber9)

- 「Link原子の問題」が解決した上で、3)と4)の間の優位性は何か
  - 使用している人の多さ
    → 3)? 4)?
    ⋮
    ⇒ 3)? 4)?

- コード公開の仕方
  - コンパイルした後の実行ファイルだけ公開
  - 追加・変更したソースコード自体を公開
  - ソースコードを変更するバッチファイルを公開

⇒ 3)? 4)?

※ 追加・変更した箇所の統一した仕様が必要ではないか
(共通)
e.g. 変数名, サブルーチン名 と 変更点が一見して分かるように

- 配列のメモリ確保の仕方
  4)では "allocate" 関数による確保
  3)では COMMON文による確保
) 異なる.

"電荷割当て法" ルーチンを組込むとき
  i) それぞれに対して、ルーチンを作ってやる.
  ii) 必要なデータはすべて引数として受け渡し、
    3), 4)に関係なく同じルーチンを作る.