Corso di Laurea Magistrale in Scienze Chimiche, Curriculum "Struttura, dinamica e reattività chimica"

Struttura e Dinamica Molecolare di Sistemi Biologici - 2010/2011

Laboratorio 6 - (17 gennaio 2011)

Preparazione e esecuzione di una serie di simulazioni MD-REM

1 Verifica dell'efficienza del REM

Il REM è efficiente se ci sono abbastanza scambi tra repliche. Questo si può verificare in due modi:

1. Verificando il fattore di accettazione medio per gli scambi tra repliche contigue. Questo numero è stampato nell'output di orac per ogni coppia di repliche con la frequenza determinato dal parametro `PRINT` del blocco `&REM`. Uno scambio efficiente prevede valori non inferiori a 10-20%

2. Verificando il grado di esplorazione delle varie repliche da parte di una traiettoria. Nell'arco della simulazione ogni traiettoria dovrebbe esplorare almeno una volta tutte le repliche.

   Si può vedere facilmente con il comando

   ```
   $ plot PAR00*/REM_DIAGNOSTIC -1,2
   ```

   Per fare una distribuzione delle repliche esplorate da ogni traiettoria:

   ```
   $ for i in {00..31} ; do awk '{print $2}' PAR00$i/REM_DIAGNOSTIC.000? |histog
   +i > 00$i.r.h; done
   ```

2 Come trattare run MD-REM lunghi in ORAC

I run più lunghi di $5ns$ vanno fatti in tratti successivi per evitare un accumulo di errori che in certi casi può addirittura portare ad un'interruzione della simulazione. Ogni run crea un file di *restart* da cui riparte quello successivo. All'inizio si fa un run brevissimo, solo per creare il primo *restart*, partendo da una struttura di soluto più solvente ottenuta da una simulazione a 300 K.

2.1 Run di preparazione

Si prende un'istantanea di una traiettoria MD normale (dopo fase di termalizzazione) inizialmente partita con molecola lineare e la si pone(soluto e solvente) in un file a sé, ad es. `REM-start.pdb`. Il run REM partirà da questa configurazione (con i relativi parametri di cella, che vanno desunti dal run di MD normale in caso di run NpT).

rem-start.in (prima parte)

```
&REM
   SETUP 1. 0.01 0.5 1
   STEP 100.
   PRINT 1000.
&END

&SETUP
   CRYSTAL    29.76 29.76 29.76
   READ_PDB   ../GPM12-REM.start.pdb
&END

&SOLUTE
   SCALE_CHARGES 1 1
&END

&SOLVENT
   ADD_UNITS 843
```

```
&END
...
```

Si fa girare per pochi step, quindi si salva un file di *restart* con il nome fisso `loop-old.rst`. Dato che un run di restart utilizza i file di topologia e di parametri in formato binario, questi vanno salvati adesso in formato binario:

rem-start.in (seconda parte)

```
...
&PARAMETERS
   READ_TPG_ASCII ../../lib/amber03.tpg
   READ_PRM_ASCII ../../lib/amber03.prm
   WRITE_TPGPRM_BIN amber03.tpgprm
   JOIN SOLUTE
     gly-h tyr asp asp ala thr lys thr phe gly-o
   END
   JOIN SOLVENT
     spce
   END
&END

&POTENTIAL
...
&END

&INTEGRATOR
...
&END

&SIMULATION
...
&END

&RUN
   CONTROL                    0
   REJECT                   0.0
   TIME                  1000.0
   PRINT                  100.0
   PROPERTY              1000.0
&END

&INOUT
  ASCII   1000.0  OPEN  GPM12-REM.pdb
  RESTART
    write 1000.0 OPEN loop-old.rst
  END
&END
```

2.2 Run di produzione

2.2.1 modello dell'input per un segmento di 1 ns

Nel run di produzione si riparte dal restart appena creato e si crea un nuovo restart (`loop-new.rst`) e un file di traiettoria (`loop-new.xyz`). Il resto dell'input rimane uguale.

Notare che per risparmiare spazio si salva la traiettoria in formato `xyz` (solo coordinate) invece che `PDB`, e escludendo il solvente (cioè definendo un frammento che comprende solo gli atomi del soluto).

loop-0.in

```
&REM
   SETUP 1. 0.01 0.5 0
   STEP 100.
   PRINT 100000.
&END
```

```
&SOLUTE
   SCALE_CHARGES 1 1
&END

&PARAMETERS
   READ_TPGPRM_BIN amber03.tpgprm
&END

&POTENTIAL
...
&END

&INTEGRATOR
...
&END

&SIMULATION
...
&END

&RUN
   CONTROL                    1
   REJECT                   0.0
   TIME                       0
   PRINT                 1000.0
   PROPERTY             10000.0
&END

&INOUT
  print_xyz_only
  PLOT  FRAGMENT 1000.0   OPEN  loop-new.xyz
  RESTART
    read loop-old.rst
    write 10000.0 OPEN loop-new.rst
  END
&END

&PROPERTIES
  DEF_FRAGMENT 1 142
&END
```

2.2.2 ciclo su segmenti di traiettoria

L'input `loop-0.in` così come è stato scritto gira per 0 step. In realtà esso è solo un modello per una semplice procedura (`loop.sh`) che per ogni segmento di traiettoria

- salva i file del segmento precedente, rinominandoli con un numero d'ordine
- rinomina il restart da `new` a `old`
- crea un nuovo input aggiungendo una quantità fissa al `TIME` (tempo di durata del run)

Per eseguire un ciclo del programma che si lancerebbe con `‘‘mpiexec -n 32 orac-p’’`, il comando che si dà è del tipo:

> `loop.sh -f 0 -t 9 -p "mpiexec -n 32 orac-p"`

che lancia il run per 10 volte, con indici che vanno da 0 a 9. In questo modo in ogni sottodirectory `PARxxxx` si ottengono una serie di file `REM_DIAGNOSTIC.0000 ...` e `0000.out ...` `0000.pdb ...`

Per lanciare il ciclo in *background* evitando che alla chiusura della sessione di terminale si interrompa il programma, si usa `nohup`:

> `rm nohup.log; nohup loop.sh -f 0 -t 9 -p "mpiexec -n 32 orac-p" &`

2.2.3 dettagli sulle procedure di loop

Il testo della procedura shell è il seguente:

loop.sh

```
#! /bin/bash
FROM=0;
TO=1;
PROG=orac;
while getopts "f:p:t:" Option do
  case $Option in
    f     )     FROM=$OPTARG ;;
    p     )     PROG="$OPTARG" ;;
    t     )     TO=$OPTARG ;;
esac
done
echo "Running program ´$PROG´, loop will run from $FROM to $TO"
for ((i=$FROM; i<=$TO; i++)); do
    printf -v l "%04d" $i
    echo $l;
    loop-pre-run $i $l;
    $PROG < loop-new.in > loop-new.out
    loop-post-run $i $l;
    for f in loop-new.*; do
          cp $f $f/loop-new/$l
          cp $f $f/loop-new/loop-old
    done
done
```

In sostanza, ad ogni ciclo: si prepara un nuovo input con la procedura `loop-pre-run`; si lancia orac; si manipolano i file creati con `loop-post-run`:

loop-pre-run

```
#! /bin/bash
i=$1;
l=$2;
awk -v I=$i -f loop-pre-run.awk < loop-new.in > a.in
mv a.in loop-new.in
```

loop-pre-run.awk

```
#! /bin/awk -f
/   TIME   /  time=$2+1000000.; print "   TIME                  ",time;next
print
```

loop-post-run

```
#! /bin/bash
l=$2;
for d in PAR00*;do
    cd $d;
    for f in loop-new.*; do
        cp $f $f/loop-new/$l
        cp $f $f/loop-new/loop-old
    done
    cp REM_DIAGNOSTIC REM_DIAGNOSTIC.$l
    cp out* $l.out
    cd ..;
done
```