MolDB - Database for Molecular Calculations

Volodymyr P. Sergiievskyi

July 6, 2013

0 Motivation

Typically, each calculation has several stages, e.g:

1. extract pdbs form the sources
2. assign the force-field
3. calculate QM charges
4. start 3DRISM calculations
5. calculate solvation free energy and RDFs

However, relatively often, one wants to try calculations with several different parameters at some stages, which will lead to variety of different results at the next stages. However, often before starting the calculations, one have no idea which exactly parameters are necessary to use. One can come to conclusion only after series of tests had been performed. And the process is continious and endless: the more one investigates the problem, the more different parameters of the algorithms one whant to try. Here is an example:

3DRISM calculations need the following stages:

- extract coordinates from the Pdb files
- assign a Force Field
- calculate Charges
- run 3DRISM

But, there are more than one force-field (CHARM, OPLS, etc). Also, there are more than one way to assign the charges (CHELPG, Muliken etc). As well, one could want to perform 3DRISM calculations with different parameters (Accuracy, Buffer, etc).

So, as a result, to distinguish different methods, one would need to create folders like
`3DRISM_CHARM_CHELPG_Accuracy1e-4_Buffer10A`

Suh a long names do not seem to be good readable. Moreove, you have already a potential source of errors. For example, typically it is very simple to forget the exact order in which you list the parameters. And the resulting folder names are often hardly distinguishible, which is a potential source of errors (Compare

`3DRISM_CHARM_CHELPG_Accuracy1e-4_Buffer10A`
and
`3DRISM_CHELPG_CHARM_Accuracy1e-4_Buffer10A`
)

Another problem with this folder format can appear if one wants to extend the parameters set (which will normally happen to almost any calculation). In that case, one need to use longer names. E.g let's imaginem that at some point one think that it could be perfect to perform the geomertry optimization before the calculations. In that case, the folders will have one additional parameter: geometry optimization on or off, e.g.
`3DRISM_opt_CHARM_CHELPG_Accuracy1e-4_Buffer10A`

At some point, it may turn out, that the method you use by default was not the best. Thus, you should try also another one. In that case, you will need to add something
`3DRISM_opt631p_CHARM_CHELPG_Accuracy1e-4_Buffer10A`

Except of obvious disadvantages (long, hardly readable, hardly distinguisable names), there also appear a problem with compatibility. So, if in previous calculations optimization was not performed, or it was performed with "default" parameters, the explicit name (opt631p) should not be included in the name of folder. But according to the "full" naming convention, it should be included explicitly.

So, one need either to rename all the folders according to the "full" convention, or remember, that some short names in reality correspond to some default parameters (e.g. "opt" means "opt631p", but not anything else). This also reduce readability and comprehansibility of the folder name, and can lead to the errors.

Although such naming conventions can be successfully used in some cases, for small projects with limited number of parameter variations, it is obvious, that in general it would be preferable to have more consistent and universal way to manipulate the calculations.

1 Hashable data storage

Instead of listing all the parameters in the file (or folder) name, it could be better to store them in a separate file. Still, it is reasonable to have one folder per method (because typically one need to store somewhere input/output and supplementary files for each method). However, there are no reason to include all the parameters into the folder name.

Each molecule's files are located in a separate folder. Indide this folder there are subfolders with methods. Inside the methods' subfolders there is a file, which describes the calculation chain. The name of the folder, except of the method name, contains the hash, which uiquely determines the calculation chain and all calculation parameters.

So, the following structure is proposed:

```
-mol1
      -method1_HASH1
             -description.txt
- file1
- file2
-output1
-output2
```

```
       -method1_HASH2
             -description.txt


        -method2_HASH3
             -description.txt



-mol2
-mol3
...
```

To retrieve the data from the database one need a special script (python?), which will go through all the records (subset of records), read description.txt files and build the table like this:

```
molecule method1_param1   method1_param2 ... method1_paramN   method2_param1 methodN_paramN
name1             val11                 val22                 ..
name1             val 21                 val22               ...
.
name2    .
..
```

From such a table it is easy to select the records with given parameters (one can write special python (or even shell) function for this.)

To put data to the database one also should use the script, which will calculate the HASH and insert the data into appropriate folder....

2 Sources, Methods and Prototypes

In each investigation one would probably have the following types of data:

- **Sources.** Files, taken from external sources, which are used as an input for the calculations
- **Methods.** The data structure, described in the previous section, where each method with each set of parameters is located in a separate folder.
- **Prototypes.** The folder, which contain the scripts to peform the calculations and to process the calculation data.

2.1 Sources

In principle, Sources folder can be organized in an arbitrary way. Actually, it depends much on a nature of the source data. In is suggested to include the references to the parpers to the data where possible.

2.2 Prototypes

The prototypes section contain the command sequences to process the data. The main type of such sequences in the *Chain* files, which contain the description of the sequence of calculations.

3 Free Energy Calculations with 3DRISM

We will describe the details of the database organization on example of the solvation free energies calculation with 3DRISM.

For the analyis we choose the set of 504 molecules from the paper *Mobley et al. J. Chem. Theory Comput. 2009, 5, 350-358.* The structures of the molecules and the solvation free energies are provided in the supporting information to the paper. The data is saved in the appropriate structure in the *Source* folder.

To perform the calculations you will need:

- MolDB
- python
- RISM-MOL-3D (download from `http://compchemmpi.wikispaces.com/RISM-MOL-3D`
- octave (for analyzing the data)

3.1 Chain file

The first step is to convert the data to the appropriate formats and save in to the database (forler structure). To do it, in the MolDB are used so-called *Chain* files. Each chain file describes the cain of calculations (actually - workflow). The chain file contains the description of the parameters, the relations between the methods and the scripts which are used for the calculation. Let's look at the *Source.chain* file, available in the *Prototypes* folder:

```
PROJECT CanadaMol

BEGIN CHAIN PROTOTYPE
1 METHOD
RUN METHOD 1



BEGIN METHOD Source
ID 1

BLOCK DEPENDENCIES
0 RECORDS
ENDBLOCK DEPENDENCIES

BLOCK PARAMETERS
2 RECORDS
ref = LIST $MOLDB_PROJECTS/CanadaMol/Sources/JCTC_2009_5_350/JCTC_5_350.bib
```

```
molecule = FILE molecules.lst
ENDBLOCK PARAMETERS

BLOCK SCRIPTS
3 RECORDS

       SCRIPT copyFiles
         ....
       ENDSCRIPT

      SCRIPT parseMol2
         ...
     ENDSCRIPT

  SCRIPT parsePrmtop
     ...
  ENDSCRIPT


ENDBLOCK SCRIPTS

END METHOD ParseSource

END CHAIN PROTOTYPE
```

The first line contain the project name (must be the same as the folder name of the project). Next lines contain the descriptuin of the *Chain*, which starts with the phrase *BEGIN CHAIN PROTOTYPE.* The chain consists of the methods, which can depend on each other (e.g. results of the calculations of one method can be used as an input for other methods). In the next line it is written, how many methods there are in the chain (*1 METHOD*) One of the methods is a *run method*, e.g. the method which will be runned by the scripts. All other methods are used just to provide dependencies/extra parameters. In our case there is only one method, 1 is its identifier (*RUN METHOD 1*)

After that we see the definition of the methods. In our case we have only one method. Each method has both: 1) Name. 2) identifier (ID) The name is used to create the propper folder structure. The identifier is used to describe the dependencies of the methods. The name of the method is given in the first line of the method definition (*BEGIN METHOD Source*), the identifier is given in the next line (*ID 1*)

The method definition contains three main *BLOCKS*. Each block starts with the sentence *BLOCK blockname*, where *blockname* is one of *DEPENDENCIES*, *PARAMETERS* or *SCRIPTS*. The next line of the block defenition describes the number of records in the block, e.g *5 RECORDS*. After that the records are defined.

3.1.1 BLOCK DEPENDENCIES

In this block the dependencies of the methods are described. Each line is one record. In our example we have no dependencies. In general case the syntax is the following:

```
  METHOD methodID AS nickname
```

The parameters of the dependencies are inherited and the folder of the dependencies is available in the scripts under the name *nickname* For example, if some method is dependent on our *Source* method we have:

```
BLOCK DEPENDENCIES
1 RECORD
  METHOD 1 AS SourceNickName
ENDBLOCK DEPENDENCIES
```

3.1.2 BLOCK PARAMETERS

In the parameters block the parameters of the method are described. Each parameter can have several values. The program will iterate over the values and the scripts will be runned for all possible combinations of the values.

Each parameter is given at the separate line. The definition is in the form *name = values* . The parameter name **must be lowercase**.

The values can be either given explicitly or read from file. In the first case one need to use the keyword *LIST* and then space separated values. In the second case the keyword *FILE* is used and then the name of file is given. The values in the file are one-per-line.

Example:

```
BLOCK PARAMETERS
3 RECORDS
   molecule = FILE molecules.lst
   closue = LIST KH HNC
   charges = LIST yes no
ENBLOCK PARAMETERS
```

Also important: Some values of the parameters can have *default* values. In that case these values are not used for HASH generation. This is important, if you already performed some calculations, and then want to extend them using one more degree of freedom (one more parameter). But your calculations were performed with some value of that parameter (which you did not include into the chain file). In that case you can add the value of this parameter to the list followed by the *(default)* keyword.

Example: Let's extend the previous example to the case of different buffer/spacing. Let in the previous calculations we used Buffer=15 and Spacing = 0.5 Angstroem, and now want to try different buffers and different spacings (be we dont want re-do the calculations which are already done).

The parameters block for that case:

```
BLOCK PARAMETERS
5 RECORDS
   molecule = FILE molecules.lst
   closue = LIST KH HNC
   charges = LIST yes no
   buffer = LIST 10 15 (default)  20
   spacing = LIST 0.2 0.3 0.5 (default)
ENBLOCK PARAMETERS
```

The values buffer=15 and spacing =0.5 will not be included to the HASH, and thus the same folders as before will be used.

3.1.3 BLOCK SCRIPTS

In the scripts block the scripts which perform the calculations are defined. Each script starts with the keyword *SCRIPT scriptname*, and ends with *ENDSCRIPT* The scripts are usuall bash scripts. For each script the separate file will be created. They will be runned in the method's folders. All the parameters of the method are available for the scripts as local variables. Also, the folder names of dependencies are available as local variables (the name of these variables are the names which are given in *DEPENDENCIES* section). Also, some special variables are available, namely: *$FOLDER* - the name of the current folder *$PROTOTYPES* - the name of the *Prototypes* folder (where chain files are located).

Also, some global variables related to the MolDB are available (after the installation): *$MOLDB_ROOT*,*$M*
$MOLDB_PROJECTS

The example of the scripts:

```
SCRIPT copyFiles

echo $molecule

Source=$MOLDB_PROJECTS/CanadaMol/Sources/JCTC_2009_5_350

cp $Source/charged_mol2files/$molecule.mol2 .
cp $Source/prmcrd/$molecule.prmtop .
cp $Source/prmcrd/$molecule.crd .

ENDSCRIPT

SCRIPT parseMol2
# parses Mol2 files and gets coors and charges

cat $molecule.mol2 | grep ATOM -A 1000 | grep BOND -B 1000 | head -n-1 | tail -n+2 >mol2.t
cat mol2.tmp | gawk '{print $2}' > atomnames_full.txt
cat atomnames_full.txt | tr -d [0-9] > atomnames.txt
cat mol2.tmp | gawk '{print $3" "$4" "$5}' > $molecule.coors
cat mol2.tmp | gawk '{print $9}' > $molecule.charges

ENDSCRIPT

SCRIPT parsePrmtop
#Uses : makePdb, ambertop2rism.py
# They are located in ~/460GB/Development/utils folder (should be in Paths)
# Note: utils folder should be distributed with the database

echo $FOLDER

makePdb $molecule.coors atomnames.txt > $molecule.pdb
```

```
P=$MOLDB_PROJECTS/CanadaMol/Prototypes

python $P/ambertop2rismmol.py $molecule.prmtop $molecule.pdb $molecule.rism

cat $molecule.rism | gawk '{print $6}' > charges.txt
cat $molecule.rism | gawk '{print $4}' > sigma.txt
cat $molecule.rism | gawk '{print $5}' > epsilon.txt
ENDSCRIPT
```

Other examples one may find in the chain files provided with the MolDB.

3.2 Running the calculations

The calculations are performed in three steps:

1. Creating the folder structure
2. Creating the scripts wich perform all the calculations
3. Running the scripts

3.2.1 moldb_createFolders

So, the first step is to create folder structure in the *Methods* folder. To do this, in the *Prototypes* folder type:

```
moldb_createFolders Source.chain
```

The script will create the folders and copy all the scripts to them. If you want create folders not for all molecules, but for some subset, you can give coma-separated options after the chain file, e.g.

```
moldb_createFolders Source.chain 'molecule=111_trichloroethane'
```

You can use also keywords *LIST* and *FILE*, e.g.:

```
moldb_createFolders Source.chain 'molecule=LIST 111_trichloroethane 1112_tetrachloroethan
```

or

```
moldb_createFolders Source.chain 'molecule=FILE short.lst'
```

In our example there is only one parameters (molecule). In general you can give any number of coma-separated options, e.g:

```
moldb_createFolders Source.chain 'molecule=FILE short.lst,closure=KH'
```

After the folders are created, you would probably want to check that everything is OK with them, before continue the calculations.

To do that you can copy one of the folder names and cd to that folder...

3.2.2 moldb_getRef

Alternatively, you can use *moldb_getRef* command, to get the folder for a molecule/molecules with specific attributes, e.g

```
moldb_getRef Source.chain 'molecule=111_trichloroethane'
```

The syntax of the coma separated list is the same as above.

3.2.3 PARAMETERS,DEPENDENCIES,chain.imp

If you cd to the folder you can see the list of the files. It include all the script files (which were provided in the *Source.chain*, and additionally - three specific files:

- PARAMETERS - this file contains all the parameters of the simuation which will be available to the script:

  ```
  PROJECT=CanadaMol
  FOLDER=$MOLDB_PROJECTS/CanadaMol/Methods/111_trichloroethane/Source/107913302
  PROTOTYPES=$MOLDB_PROJECTS/CanadaMol/Prototypes
  molecule=111_trichloroethane
  ref=$MOLDB_PROJECTS/CanadaMol/Sources/JCTC_2009_5_350/JCTC_5_350.bib
  ```

- DEPENDENCIES - this file contain all the references to the folder which were given in the DEPENDENCIES section.

- chain.imp - The file, analogous to the Source.chain, BUT, instead of lists of variables there are some concreete values, e.g:

  ```
  PROJECT CanadaMol
  BEGIN CHAIN IMPLEMENTATION
  1 METHODS
  RUN METHOD 107913302

  BEGIN METHOD Source
  ID 107913302
  BLOCK DEPENDENCIES
  0 RECORDS
  ENDBLOCK DEPENDENCIES
  BLOCK PARAMETERS
  2 RECORDS
  molecule = 111_trichloroethane
  ref = $MOLDB_PROJECTS/CanadaMol/Sources/JCTC_2009_5_350/JCTC_5_350.bib
  ENDBLOCK PARAMETERS
  BLOCK SCRIPTS
  3 RECORDS
  ....
  ENDBLOCK SCRIPTS
  END METHOD Source
  ```

```
    END CHAIN IMPLEMENTATION
```

Also, you would probably want to run your scripts, to check that there are no errors in them. Just do it like ./copyFiles or ./prepareMol2 while you are in the molecule's folder.

To return from the molecule's folder you can

cat PARAMETERS

copy the PROTOTYPES line and cd there, e.g:

```
cd $MOLDB_PROJECTS/CanadaMol/Prototypes
```

3.2.4 moldb_runScript

After creation the folders you can run the scripts. We have three scripts: copyFiles, parseMol2, parsePrmtop

To run we script one can use *moldb_runScript* command:

`moldb_runScript Source.chain copyFiles`

However, this command does not run the script, it only creates the script file which will go through all the folders and run appropriate script in them. The script name is *ProjectName_ChainName_scriptName.sh*

In our case: *CanadaMol_Source_copyFiles.sh*

If you want to run the script only for specific molecules, you can use the command with options, e.g.

`moldb_runScript Source.chain copyFiles  'molecule=FILE short.lst'`

The scipt like this will be generated:

```
#!/bin/bash
#
#  XPARAM: molecule=FILE short.lst
#
FOLDERS="
$MOLDB_PROJECTS/CanadaMol/Methods/1112_tetrachloroethane/Source/158372845
$MOLDB_PROJECTS/CanadaMol/Methods/111_trichloroethane/Source/107913302
$MOLDB_PROJECTS/CanadaMol/Methods/111_trifluoro_222_trimethoxyethane/Source/652622319
"

N=$(echo $FOLDERS | wc -w )
count=0
P=$(pwd)
for FOLDER in $FOLDERS
do
echo Molecule $count of $N "(" $((count*100/N)) percent done ")"
cd $FOLDER
. ./PARAMETERS
. ./DEPENDENCIES
./copyFiles
count=$((count+1))
done
```

You can run in by just typing:

```
./CanadaMol_Source_copyFiles.sh
```

Also, probably you would like to use more that one thread. In that case use

```
moldb_runScript Source.chain copyFiles -p 2
```

Where 2 is the number of processes. Then several script files will be created named like

```
./CanadaMol_Source_copyFiles_p0.sh
./CanadaMol_Source_copyFiles_p1.sh
```

and the file `./CanadaMol_Source_copyFiles.sh` which run both of them in parallel.

Hint: Don't use parallelization for file copying: it makes no sence because several processes will try acces the HDD simultaniously and efficiency will be low. However, for calculations (3DRISM or so) this option is very useful if you have more than one CPU core (AND enough memory, don't forget!)

So, finally to do everything we need with Source.chain we have to run the following commands:

```
moldb_createFolders Source.chain

moldb_runScript Source.chain copyFiles
./CanadaMol_Source_copyFiles.sh

moldb_runScript Source.chain parseMol2
./CanadaMol_Source_parseMol2.sh

moldb_runScript Source.chain parsePrmtop
./CanadaMol_Source_parsePrmtop.sh
```

Also, additionally: We need to parse the table with experimental/MD values and also store them to the Source folders. This can be done by the separate script which uses *moldb_getRef*.

The script name is *storeTable1*:

```
#!/bin/bash

Table1=../Sources/JCTC_2009_5_350/SI/Table1/Table1.txt

IFS=$'\n'
Cols="molecule"$IFS
Cols=$Cols"DeltaG_elec"$IFS"ErrDeltaG_elec"$IFS
Cols=$Cols"DeltaG_vdw"$IFS"ErrDeltaG_vdw"$IFS
Cols=$Cols"DeltaG_hydr"$IFS"ErrDeltaG_hydr"$IFS
Cols=$Cols"DeltaG_expt"

IFS=$'\n'
for line in $(cat $Table1)
do

molecule=$(echo $line | gawk '{print $1}' )
```

```
folder=$(moldb_getRef Source.chain  molecule=$molecule)

N=0
for col in $Cols
do
N=$((N+1))
val=$(echo $line | gawk "{print \$$N}" | tr  "  -")

echo 'echo '$val ' > '$folder'/'$col'.dat'
done

done
```

This script does not do any actions, it only print the commands to do them (just to give you an opportunity that everything is OK before running).

So, the commands to run are:

```
./storeTable1 > storeTable1_run
chmod 777 storeTable1_run
./storeTable1_run
```

3.3 3DRISM.chain

Now, we have all the Source data prepared, we can run the 3DRISM calculations. We run the 3DRISM for each molecule for such variations: with/without charges, with KH/HNC closure (4 variants in total).

The 3DRISM.chain file is the following:

```
PROJECT CanadaMol

BEGIN CHAIN PROTOTYPE
2 METHODS
RUN METHOD 2

FROM Source.chain IMPORT METHOD 1 AS 1

BEGIN METHOD 3DRISM
ID 2

BLOCK DEPENDENCIES
1 RECORD
METHOD 1 AS Source
ENDBLOCK DEPENDENCIES

BLOCK PARAMETERS
2 RECORDS
closure = LIST HNC KH
```

```
charges = LIST yes (default) no
ENDBLOCK PARAMETERS

BLOCK SCRIPTS
3 RECORDS

SCRIPT prepareInput
...
ENDSCRIPT


SCRIPT run3DRISM
....
ENDSCRIPT

SCRIPT clearCalculations
.....
ENDSCRIPT

ENDBLOCK SCRIPTS

END METHOD 3DRISM

END CHAIN PROTOTYPE
```

We see, that in this file are definded two methods: Source and 3DRISM, and the method 3DRISM depends on Source (see BLOCK DEPENDENCIES)

The method SOurce is *imported* from the file Source.chain: `FROM Source.chain IMPORT METHOD 1 AS 1`

The syntax of IMPORT command:

`FROM file IMPORT METHOD methodIDinFile AS newMethodID`

There are three scripts in the 3DRISM.chain file:

- prepareInput - copies files which are necesary for tehe calculation to the method's folder
- run3DRISM - perform the 3DRISM calculations
- clearCalculations - remove input/output files (necessary for reducing the size of the database)

3.3.1 prepareInput

```
SCRIPT prepareInput

echo $FOLDER

cp $PROTOTYPES/3DRISM/* .
cat parameters.tmpl | sed "s:SED_CLOSURE:$closure:g" > parameters.txt
```

```
cp $Source/$molecule.rism mol.rism

if [ $charges == no ]; then

cat mol.rism | gawk '{print $1" "$2" "$3" "$4" "$5}' > mol.tmp
N=$(cat mol.rism | wc -l)

rm -f zero.txt

for((i=0;i<$N;i++))
do
echo 0 >> zero.txt
done

multicol mol.tmp zero.txt > mol.rism

fi


ENDSCRIPT
```

As we see, the script copies the rism input file from the corresponding Source folder (given by *$Source* variable)

If there are no charges it also generates the file with the last (charge) columns with zeros.

Also, the script changes closure in the parameters file to the correct value.

3.3.2 run3DRISM

```
SCRIPT run3DRISM

echo $FOLDER
echo $molecule $closure

tic=$(date +%s)


multiGridMain parameters.txt 'SoluteStructureFile="mol.rism";' >multiGrid.out &
pid=$!
TimeLimit=30  #% *10 = 300 sec = 5min


for((i=0;i<=$TimeLimit;i++))
do
sleep 10
N=$(ps -A | grep $pid | wc -l)

if [ $N == 0 ]; then
```

```
break;
fi

echo -n .
done

echo $i $pid


if [ $i -ge $TimeLimit ]; then
echo ITERATION DIVERGED
kill -9 $pid
fi

calculateFreeEnergySimple mol_in_water_ > freeEnergy.out

toc=$(date +%s)

cat freeEnergy.out | grep HNC | gawk '{print $2}' > HNC.dat
cat freeEnergy.out | grep KH | gawk '{print $2}' > KH.dat
cat freeEnergy.out | grep GF | gawk '{print $2}' > GF.dat
cat freeEnergy.out | grep VUA | gawk '{print $2}' > PMV.dat

tail -n4 freeEnergy.out
echo Time Elapsed=$((toc-tic))

rm -f *.3d

ENDSCRIPT
```

The script performs the 3DRISM and solvation free energy calculations and stores the results to the separate files (HNC.dat, KH.dat, GF.dat, PMV.dat).

Also it removes the 3d files - 3D distributions of oxygen/hydrogen (to save the disk space - files can be $> 10M$, and you should multiply this values by the number of molecules (504).

3.3.3 clearCalculations

```
SCRIPT clearCalculations

# clears all intermediate files used for the calculation
# prepareInput will be necessary before next start of run3DRISM

rm -f water.slv
rm -f *.grd
rm -f waterRDFs.txt
rm -f parameters.txt
rm -f mol.rism
rm -f *.out

ENDSCRIPT
```

Removes the input/output files (except the results), saves the disc space. Is necessary, if you want to copy your database somewhere (saves space).

3.3.4 run the method

To run the method use the following scripts:

```
moldb_createFolders 3DRISM.chain

moldb_runScript 3DRISM.chain prepareInput
./CanadaMol_3DRISM_prepareInput.sh

moldb_runScript 3DRISM.chain run3DRISM -p 2
./CanadaMol_3DRISM_run3DRISM.sh
```

Here -p 2 means that the calculations will be parllelized to 2 cores. Use another value for another hardware...

4 Visualizing and analyzing Results

4.1 moldb_printResults

To view results of the calculations you can use *moldb_printResults* command.

The syntax is: `moldb_printResults chain rows columns file [ restrictions ]`

The reusults are printed in the table view. Rows and cols are the coma-separated lists of parameters.

The command will read the *file* from all the folders described by *chain*.

Example: 1)

`moldb_printResults 3DRISM.chain molecule charges,closure HNC.dat`

Output:

```
molecule                                charge/HNC charge/KH nocharge/HNC nocharge/KH
1112_tetrachloroethane                    16.148     20.162    18.267        22.088
111_trichloroethane                       14.898      18.44    16.585        19.939
111_trifluoro_222_trimethoxyethane        17.153      22.68    23.571        28.518
111_trifluoropropan_2_ol                  6.5384     10.677    16.742        19.967
111_trimethoxyethane                      12.788     18.214    21.428        26.1
...
```

The script uses the sed commands from the *aliases.txt* to make the headers of the table look better.

2)

`moldb_printResults 3DRISM.chain molecule closure HNC.dat 'charges=yes'`

Output:

```
molecule                                HNC      KH
1112_tetrachloroethane                    16.148   20.162
111_trichloroethane                       14.898    18.44
111_trifluoro_222_trimethoxyethane        17.153    22.68
111_trifluoropropan_2_ol                  6.5384   10.677
111_trimethoxyethane                      12.788   18.214
1122_tetrachloroethane                    14.884   19.005
112_trichloro_122_trifluoroethane         18.765   22.537
112_trichloroethane                       12.128    15.65
11_diacetoxyethane                        11.421   18.174
11_dichloroethane                          12.06   15.201
....
```

If you want to view the results, which has less then 1 parameters (e.g. Source), you can use the following syntax with *dummy* column, e.g:

`moldb_printResults Source.chain molecule dummy DeltaG_vdw.dat 'dummy=MD_nocharge'`

Output:

```
molecule                                MD_nocharge
1112_tetrachloroethane                       1.54
111_trichloroethane                          1.88
111_trifluoro_222_trimethoxyethane           2.31
111_trifluoropropan_2_ol                     2.49
111_trimethoxyethane                         1.66
1122_tetrachloroethane                       1.47
112_trichloro_122_trifluoroethane            1.85
112_trichloroethane                          1.51
11_diacetoxyethane                           1.62
11_dichloroethane                            1.81
....
```

4.2 Visualization using octave (matlab)

To visualize results you can use octave. Below is the example (all the commands - in octave):

-1st step: add the path to your MolDB bin folder to your /.octaverc file, e.g

```
echo "path(path,'"$MOLDB_BIN"');" >> ~/.octaverc
echo "path(path,'"$MOLDB_ROOT/utils"');" >> ~/.octaverc
```

-2nd step - run octave

-3rd step - build the references to the folders from Source.chain and 3DRISM.chain:

```
 Ref3DRISM=moldb_getRef('3DRISM.chain');
 RefSource=moldb_getRef('Source.chain');
```

-4th step - filter the results To select some subset from the references use the *moldb_filter* command:

`I = moldb_filter(Ref3DRISM,{'closure','HNC','charges','yes'});` The command returns the indeces of the specified records in the Ref3DRISM. These indeces can be used for all data related to the Ref3DRISM (e.g. data files loaded for this data)

The parameters to filter are given as a cell array in a key-value pairs `{key1, val1, key2, val2 , ...}`

-5th step load the data files to the cell arrays:

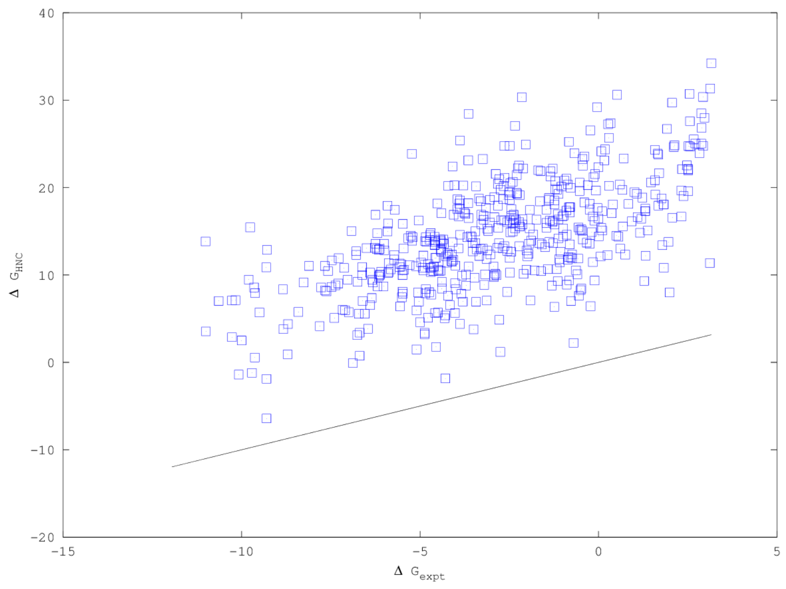
```
Exp_cell = moldb_load(RefSource,'DeltaG_expt.dat');
HNC_cell = moldb_load(Ref3DRISM,'HNC.dat');
PMV_cell = moldb_load(Ref3DRISM,'PMV.dat');
```

Note: the data is returned in the cell array. To convert this arrays to the matrices use `cell_get_values`:

```
Exp = cell_get_values(Exp_cell);
HNC = cell_get_values(HNC_cell);
PMV = cell_get_values(PMV_cell);
```

-6th step: plot the results

```
plot(Exp,HNC(I),'s',Exp,Exp,'k')
xlabel('\Delta G_{expt}')
ylabel('\Delta G_{HNC}')
```

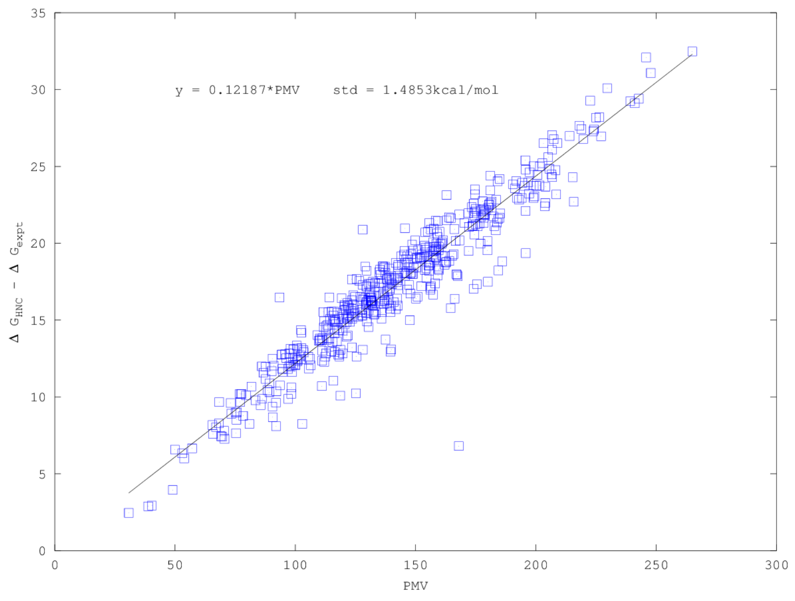


-7th step: fit the results

```
J=isfinite(HNC(I));
a=myregress(HNC(I(J))-Exp(J),PMV(I(J)))
plot(PMV(I),HNC(I)-Exp,'s',PMV(I),a*PMV(I),'k')
xlabel('PMV')
```

```
ylabel('\Delta G_{HNC} - \Delta G_{expt}')
S=nanstd(HNC(I)-a*PMV(I) - Exp)

text(50,30,[ 'y = ' num2str(a) '*PMV    std = ' num2str(S) 'kcal/mol' ])
```


y = 0.12187*PMV   std = 1.4853kcal/mol
Δ G_HNC - Δ G_expt
PMV