PyMobCal/CMobCal Developer's Manual 3.0

November 11, 2019

NOTE: I am assuming you have read the quick guide and you have a working version of mobcal

1 What version for devs?

For development I recommend the python version. Testing new algorithms, parameters, etc. is much easier to do in python. No need to compile, and has plenty of tools to do just that. The C version should be used only once the new features have been tested on python.

2 Ctypes, f2py

I will not discuss how to use Ctypes and f2py here. The test folders have plenty of examples. If the need arises, I may discuss this in the future. In brief, Ctypes is used to call C-function in python, whereas f2py does the same thing for Fortran. In practice let's say that you want to test/compare a function written in C or fortran, but not the entire code. Then you use those.

To "activate" Ctype in the C code set TEST_CTYPES to 1. It expects that the random number generator is passed by Python instead of using its own.

3 How to change atom parameters

. In python make changes to the lines:

```
AtomList=(
    atom('Hydrogen',1.008,0.65e-03*ConstList.xe,2.38e-10,2.2e-10,1),
    atom('Carbon',12.01,1.34e-3*ConstList.xe,3.043e-10,2.7e-10,12),
    atom('Nitrogen',14.01,1.34e-3*ConstList.xe,3.043e-10,2.7e-10,14),
    atom('Oxygen',16.00,1.34e-3*ConstList.xe,3.043e-10,2.7e-10,16),
    atom('Sodium',22.99,0.0278e-3*ConstList.xe,(3.97/1.12246)*1e-10,2.853e-10,2
    atom('Silicon',28.09,1.35e-3*ConstList.xe,3.5e-10,2.95e-10,28),
    atom('Sulfur',32.06,1.35e-3*ConstList.xe,3.5e-10,3.5e-10,32),
    atom('Iron',55.85,1.35e-3*ConstList.xe,3.5e-10,3.5e-10,56)
)
```

For each line you have the properties of one atom:

```
class atom:
'''
* Structure containing information about a specific atomtype
*
* Attributes
* ----------
* name : name of the atom
* mass : atomic mass
* epsilonLj,sigmaLJ : Lennard-Jones parameters
* rhs : parameters for hard sphere scattering
* id : atomic number
*
'''
```

In C (with the same meaning as above):

```
const struct atom AtomList[]={
    /*
    * Here is a list with the atom recognized by MobCal.
    * For new atoms, add a line here.
    * The software defaults to carbon (and issues a warning) if the
    * atomtype specified is not found here.
    */
    {"Hydrogen",1.008,0.65e-03*XE,2.38e-10,2.2e-10,1},
    {"Carbon",12.01,1.34e-3*XE,3.043e-10,2.7e-10,12},
    {"Nitrogen",14.01,1.34e-3*XE,3.043e-10,2.7e-10,14},
    {"Oxygen",16.00,1.34e-3*XE,3.043e-10,2.7e-10,16},
    {"Sodium",22.99,0.0278e-3*XE,(3.97/1.12246)*1e-10,2.853e-10,23},
    {"Silicon",28.09,1.35e-3*XE,3.5e-10,2.95e-10,28},
    {"Sulfur",32.06,1.35e-3*XE,3.5e-10,3.5e-10,32},
    {"Iron",55.85,1.35e-3*XE,3.5e-10,3.5e-10,56}
};
```

4 Algorithms

Here, I briefly walk you through the functions of each algorithms. Each function is documented in the source code for input/output parameters. I am not listing ”service” functions that take care of allocating/freeing memory or perform basic operations. Those should be straightforward to use. As usual, if the need arises, they can be added here.

4.1 PA, EHS

Currently, PA and EHS are computed by the same function mobil4(). Mobil4() contains the Monte Carlo routine and calls the function che() to check for collisions as well as to generate the trajectory for EHS.

4.2 TM

The function mobil2() takes care of it. The scattering angle is computed by the function gsang(). gsang() relies on three functions:

- dljpot() for computing the forces and potential
- deriv() that computes the derivatives used for the integration of the equations of motion
- diffeq() that solves the equations of motion.

4.3 Reading input

We have two functions for this:

- fcoord() reads the coordinates the first time. It sets variables that are not supposed to vary such as number of atoms and units used.
- ncoord() reads the 2nd to last structure. It is a light version of fcoord().

Both functions assume that the input is mobcal-like. If you use a different input format, those two must be adapted.

4.4 Output

In the C code the printing verbosity is controlled by the following preprocessors flags:

```
#define PRINT_IT  0
#define PRINT_IP  0
#define PRINT_IGS 0
#define PRINT_IU1 0
#define PRINT_IU2 0
#define PRINT_IU3 0
```

and the following variables:

```
print_im2
print_im4
```

In Python, all those variables are contained in:

```
    class printSwitch:
    '''
    Some switches used to turn on/off some debugging features.
    Unless you are modifying the code, keep defaults

c     print switches ip=1  print scattering angles
c                    it=1  print trajectory
c                    iu1=1 print initial coordinates from FCOORD
c                    iu2=1 print angles and coordinates from ROTATE
c                    iu3=1 print angles from ROTATE
c                    iv=1  print all potentials from POTENT
c                    im2=1 print brief information from MOBIL2
c                    im4=1 print brief information from MOBIL4
```

```
c                      igs=1 print out angles, v, and b from MOBIL4 into
c                            a temporary file called hold
    '''
    def __init__(self):
        self.ip=0
        self.it=0
        self.iu1=0
        self.iu2=0
        self.iu3=0
        self.iv=0
        self.im2=0
        self.im4=0
        self.igs=0
```

4.5 Rotations

The code uses two rotations functions:

- rotate() to rotate a structure
- rantate() to rotate a structure of a random angle.

5 Running tests

The C code has a flag TEST to be set to 1. It is used to make sure that the code runs in a reasonable amount of time for performing multiple tests.