Manual 11 for MD++

Python Language as MD++ Input

Mikael Jorda and Wei Cai

January 14, 2015

MD++ will interpret the input file as a Python script if it has `.py` as the extension name and is the executable file was created with the `PY=yes` option. If the executable file was made with the option `PY=no` (default), and if the input file has a `.script` or `.tcl` extension, it will be interpreted respectively with the MD++ or the tcl syntax. For example, to make the `sw` executable file, the command is

```
$ make sw build=R SYS=gpp PY=yes
```

In a Python script, there must be the package `mdpp` imported in order to call MD++ in the script. Then, the command `mdpp.cmd` calls MD++ in Python. The argument must be a string. The string will be interpreted as the MD++ command. We can pass a single MD++ line or several ones with one call to `mdpp.cmd` using the triple string marks (’’’ ’’’). For example, the `si.script` file discussed in Manual 02 can be translated to the following python file, `si.py`, and MD++ will perform identical operations, i.e. creating a perfect crystal and visualizing it, when receiving both files as inputs.

```
-*-Python-script-*-
import mdpp

mdpp.cmd('setnolog')
mdpp.cmd('setoverwrite')
mdpp.cmd('dirname = runs/si-example)
#------------------------------------------------------------
# Create Perfect Lattice Configuration
#
mdpp.cmd('element0 = Si')
mdpp.cmd('crystalstructure = diamond-cubic')
mdpp.cmd('latticeconst = 5.4309529817532409') # (A) for Si
mdpp.cmd('element0 = Silicon')
mdpp.cmd('''
    latticesize = [ 1 0 0 2
                    0 1 0 2
                    0 0 1 3 ]
                    ''')
mdpp.cmd('makecrystal writecn')
#------------------------------------------------------------
# Plot Configuration
#
mdpp.cmd('atomradius = 0.67 bondradius = 0.3 bondlength = 2.8285')
mdpp.cmd('atomcolor = orange highlightcolor = purple')
mdpp.cmd('bondcolor = red backgroundcolor = white')
mdpp.cmd('plotfreq = 10 rotateangles = [ 0 0 0 1.25 ]')
mdpp.cmd('openwin alloccolors rotate saverot eval plot')
mdpp.cmd('sleep quit')
```

We can send parameters to MD++ using the format command of Python.

```
mdpp.cmd('''
    latticesize = [ 1 0 0 {0}
                    0 1 0 {1}
                    0 0 1 {2} ]
                    '''.format(2,2,3))
```

1 Grouping

Python allows us to define our own functions (procedures or subroutines). We can use this to better organize our input file. The syntax for defining a procedure is

```
def proc_name(args) :
      body of the procedure
```

Consider the following Python input file `si2.py`

```
-*-Python-script-*-

import mdpp

#*****************************************
# Definition of procedures
#*****************************************
def initmd(n) :
     mdpp.cmd('setnolog')
     mdpp.cmd('setoverwrite')
     mdpp.cmd('dirname = runs/si-example-{0}'.format(n))
     mdpp.cmd('zipfiles = 1')       # zip output files
     mdpp.cmd('NIC = 200 NNM = 200')
     #-------------------------------------------------------------
     # Create Perfect Lattice Configuration
     #
     mdpp.cmd('element0 = Si')
     mdpp.cmd('crystalstructure = diamond-cubic')
     mdpp.cmd('latticeconst = 5.4309529817532409') \# (A) for Si
     mdpp.cmd('element0 = Silicon')
     mdpp.cmd('''
          latticesize = [ 1 0 0 2
                          0 1 0 2
                          0 0 1 3 ]
                          ''')

#-------------------------------------------------------------
def openwindow() :
     # Plot Configuration
     #
     mdpp.cmd('''
     atomradius = 0.67 bondradius = 0.3 bondlength = 2.8285  # for Si
     atomcolor = orange highlightcolor = purple
     bondcolor = red backgroundcolor = grey70
     plotfreq = 10 rotateangles = [ 0 0 0 1.25 ]
     openwin alloccolors rotate saverot eval plot
     ''')

#*****************************************
# Main program starts here
#*****************************************
initmd(1)
mdpp.cmd('makecrystal writecn')
openwindow()
mdpp.cmd('sleep quit')
```

This input file also asks MD++ to create a perfect Si crystal and visualize it. The main program is contained in the last 4 lines of this file. The first line will lead to the creation of the directory `runs/si-example-1` because the argument for command `initmd` is 1. Using procedures allows us to give more structure to the input file and make it easier to read and manage.

In python, we can also use the modularity to further simplify and automate certain tasks. For example, we can regroup all the visualization functions in one module and the import it in the other input files so we don't have to rewrite all over again functions such as openwindow().

```
##### python module for visualization #####

# visualization module visual.py

import mdpp

def openwindow() :
     # Plot Configuration
     #
     mdpp.cmd('''
     atomradius = 0.67 bondradius = 0.3 bondlength = 2.8285  # for Si
     atomcolor = orange highlightcolor = purple
     bondcolor = red backgroundcolor = grey70
     plotfreq = 10 rotateangles = [ 0 0 0 1.25 ]
     openwin alloccolors rotate saverot eval plot
     ''')
-----------------------------------------------------

##### input file #####

import mdpp
import visual

#*****************************************
# Definition of procedures
#*****************************************
def initmd(n) :
     mdpp.cmd('setnolog')
     mdpp.cmd('setoverwrite')
     mdpp.cmd('dirname = runs/si-example-{0}'.format(n))
     mdpp.cmd('zipfiles = 1')       # zip output files
     mdpp.cmd('NIC = 200 NNM = 200')
     #-------------------------------------------------------------
     # Create Perfect Lattice Configuration
     #
     mdpp.cmd('element0 = Si')
     mdpp.cmd('crystalstructure = diamond-cubic')
     mdpp.cmd('latticeconst = 5.4309529817532409') \# (A) for Si
     mdpp.cmd('element0 = Silicon')
     mdpp.cmd('''
          latticesize = [ 1 0 0 2
                          0 1 0 2
                          0 0 1 3 ]
                          ''')

#*****************************************
# Main program starts here
#*****************************************

initmd(1)
mdpp.cmd('makecrystal writecn')
visual.openwindow()
mdpp.cmd('sleep quit')
```

2 get and set a MD++ variable

`mdpp.get` retrieves the value of an MD++ variable and pass it to Python. For example,

```
print('totalsteps = {0}'.format(mdpp.get(totalsteps))
```

prints the value of MD++ variable `totalsteps` on the screen (which is not possible to do within the original script file). To assign value to the variable `totalsteps`, we can write, e.g.

```
mdpp.cmd('totalsteps = 1000')
```

Every MD++ variable that was bound to a string (by bindvar in md.cpp) can be fetched by `mdpp.get`. Some widely used variables are: number of atoms `NP`, simulation box volume `OMEGA`, potential energy `EPOT`, kinetic energy `KATOM`, instantaneous temperature `Tinst`, pressure `PRESSURE`, current iteration step `curstep`, total internal stress (Virial + kinetic term) `TSTRESS_xx`, `TSTRESS_yy`, `TSTRESS_zz`, `TSTRESS_xy`, `TSTRESS_xz`, `TSTRESS_yz`, etc..

3 get and set an array

`mdpp.get_array` retrieves the value of an MD++ array and pass it to a Python array
`mdpp.set_array` allows Python to assign the value of an MD++ array from a Python array. For example,

```
sx = mdpp.get_array('SR',0,30,3)
```

extracts the 'SR' array to Python list variable `sx`. In MD++, SR contains the information for the scaled coordinates of atoms, in the order of sx0, sy0, sz0, sx1, sy1, sz1, ... Therefore, every 1 out of 3 values in SR corresponds to the x-coordinates. The three indices in the arguments of get_array() are: istart, iend, iskip.
On the other hand, in

```
mdpp.set_array(sx, 'SR', 0, 3)
```

only two indices (istart, iskip) are specified, because the information about the length of the array is contained in sx already. Here is an example in which we create a lattice configuration and modify the positions of certains atoms

```
# Python script

#*****************************************
# Definition of procedures
#*****************************************
import mdpp

def initmd(T) :
    mdpp.cmd('setnolog setoverwrite')
    mdpp.cmd('dirname = runs/si-example-{0}'.format(T))

def makecrystal(nx,ny,nz) :
    # create perfect lattice configuration
    mdpp.cmd('crystalstructure = diamond-cubic')
    mdpp.cmd('latticeconst = 5.4309529817532409') # (A) for Si
    mdpp.cmd('element0 = Silicon')
    mdpp.cmd('latticesize = [ 1 0 0 {0} 0 1 0 {1} 0 0 1 {2} ]'.format(nx,ny,nz))
    mdpp.cmd('makecrystal  finalcnfile = perf.cn writecn')

#*****************************************
# Main program starts here
#*****************************************
initmd(0)
makecrystal(4,4,4)

sx = mdpp.get_array('SR',0,30,3)
sy = mdpp.get_array('SR',1,31,3)
sz = mdpp.get_array('SR',2,32,3)
```

```
print 'sx = ', sx
print 'sy = ', sy
print 'sz = ', sz

print 'modifying position of atom 3 and 5'

sx[3] = sx[3] + 0.220
sy[3] = sy[3] - 0.029
sz[3] = sz[3] + 0.014

sx[5] = sx[5] + 0.091
sy[5] = sy[5] + 0.382
sz[5] = sz[5] + 0.064

mdpp.set_array(sx, 'SR', 0, 3)
mdpp.set_array(sy, 'SR', 1, 3)
mdpp.set_array(sz, 'SR', 2, 3)

sxnew = mdpp.get_array('SR',0,30,3)
synew = mdpp.get_array('SR',1,31,3)
sznew = mdpp.get_array('SR',2,32,3)

print 'sxnew = ', sxnew
print 'synew = ', synew
print 'sznew = ', sznew
```

we get the output

```
sx =  [-0.5, -0.375, -0.5, -0.375, -0.4375, -0.3125, -0.4375, -0.3125, -0.5, -0.375]
sy =  [-0.5, -0.375, -0.375, -0.5, -0.4375, -0.3125, -0.3125, -0.4375, -0.5, -0.375]
sz =  [-0.5, -0.5, -0.375, -0.375, -0.4375, -0.4375, -0.3125, -0.3125, -0.25, -0.25]
modifying position of atom 3 and 5
sxnew =  [-0.5, -0.375, -0.5, -0.155, -0.4375, -0.2215, -0.4375, -0.3125, -0.5, -0.375]
synew =  [-0.5, -0.375, -0.375, -0.529, -0.4375, 0.0695, -0.3125, -0.4375, -0.5, -0.375]
sznew =  [-0.5, -0.5, -0.375, -0.361, -0.4375, -0.3735, -0.3125, -0.3125, -0.25, -0.25]
```

These functions work with usual Python arrays as well as Numpy arrays.

4 Writting, reading data file and plotting data

Instead of printing the data to the screen, it is often more convenient to write it on a file. In Python, data can be written directly to a datafile using

```
f = open('datafile','w')
f.write('data to write')
f.close  # do not forget to close the file
```

However, MD++ can write an output file containing the data by itself which can be read by python to plot some data. For example, the following script runs a NVE simulation on a Silicon crystal and plots the temperature in function of time (script `si-temp.py`)

```
# -*- python script-*-
# MD code of Stinger-Weber Silicon

#*****************************************
# Definition of procedures
#*****************************************

import mdpp

def initmd(T) :
    mdpp.cmd('''
    setnolog
    setoverwrite
    ''')
    mdpp.cmd('dirname = runs/si-example-{0}'.format(T))

def makecrystal(nx,ny,nz) :
    # create perfect lattice configuration
    mdpp.cmd('crystalstructure = diamond-cubic')
    mdpp.cmd('latticeconst = 5.4309529817532409') # (A) for Si
    mdpp.cmd('element0 = Silicon')
    mdpp.cmd('''
    latticesize = [ 1 0 0 {0}
                    0 1 0 {1}
                    0 0 1 {2} ]
    '''.format(nx,ny,nz))
    mdpp.cmd('makecrystal  finalcnfile = perf.cn  writecn')

def setup_window() :
    # plot settings
    mdpp.cmd('''
     atomradius = 0.67 bondradius = 0.3 bondlength = 2.8285 #for Si
     atomcolor = orange highlightcolor = purple
     bondcolor = red backgroundcolor = gray70
    #plot_color_bar = [ 1 -4.85 -4.50 ]  highlightcolor = red
     plotfreq = 10  rotateangles = [ 0 0 0 1.25 ]
    ''')

def openwindow() :
    #Configure and open a plot
    setup_window()

    #### setup_window
    mdpp.cmd('openwin  alloccolors rotate saverot eval plot')
```

```
def relax_fixbox() :
    #Conjugate-Gradient relaxation

    mdpp.cmd('''
    conj_ftol = 1e-7 conj_itmax = 1000 conj_fevalmax = 10000
    conj_fixbox = 1 #conj_monitor = 1 conj_summary = 1
    relax
    ''')

def setup_md() :
    # MD settings (without running MD)

    mdpp.cmd('''
    equilsteps = 0  totalsteps = 100 timestep = 0.0001 # (ps)
    atommass = 28.0855 # (g/mol)
    DOUBLE_T = 0
    randseed = 12345 srand48
    initvelocity totalsteps = 10000 saveprop = 0
    saveprop = 1 savepropfreq = 10 openpropfile
    vt2=1e28
    ensemble_type = NVE integrator_type = VVerlet
    implementation_type = 0

    totalsteps = 5000
    output_fmt = "curstep EPOT KATOM Tinst HELM HELMP TSTRESS_xx TSTRESS_yy TSTRESS_zz"
    plotfreq = 1
    ''')




#*****************************************
# Main program starts here
#*****************************************

#create crystal, relax, run MD/NVE simulation
T_OBJ = 300
initmd(T_OBJ)

makecrystal(4,4,4)

openwindow()

relax_fixbox()
mdpp.cmd('finalcnfile = relaxed.cn writecn')
mdpp.cmd('finalcnfile = relaxed.lammps writeLAMMPS')
mdpp.cmd('finalcnfile = relaxed.cfg writeatomeyecfg')
setup_md()
mdpp.cmd('T_OBJ = {0}'.format(T_OBJ))
mdpp.cmd('initvelocity')
mdpp.cmd('run')

mdpp.cmd('finalcnfile = si100.cn writecn')
mdpp.cmd('finalcnfile = si100.cfg writeatomeyecfg')
```

```
#*****************************************
# Plot simulation data
#*****************************************
from pylab import *

print "Plot simulation data using pylab"

prop = loadtxt('prop.out')
print 'test'
plot( prop[:,0], prop[:,3] )
xlabel("steps")
ylabel("T (K)")
print 'test2'
ion()
show()

#*****************************************
# Sleep
#*****************************************
import time
sleep_seconds = 60
print "Python is going to sleep for " + str(sleep_seconds) + " seconds."
time.sleep(sleep_seconds)
```

In `setup_md` we define all the properties of the simulation. We also define the output format of the file `prop.out` which will be written by MD++ and contain all the parameters we want with the MD++ command output_fmt. We can read the in python using

```
prop = loadtxt('prop.out')
```

and get the values of the properties we are interested in. Here, we want to plot the temperature in function of the timestep. So we get the first and the fourth column of the file. and we plot them using pylab.

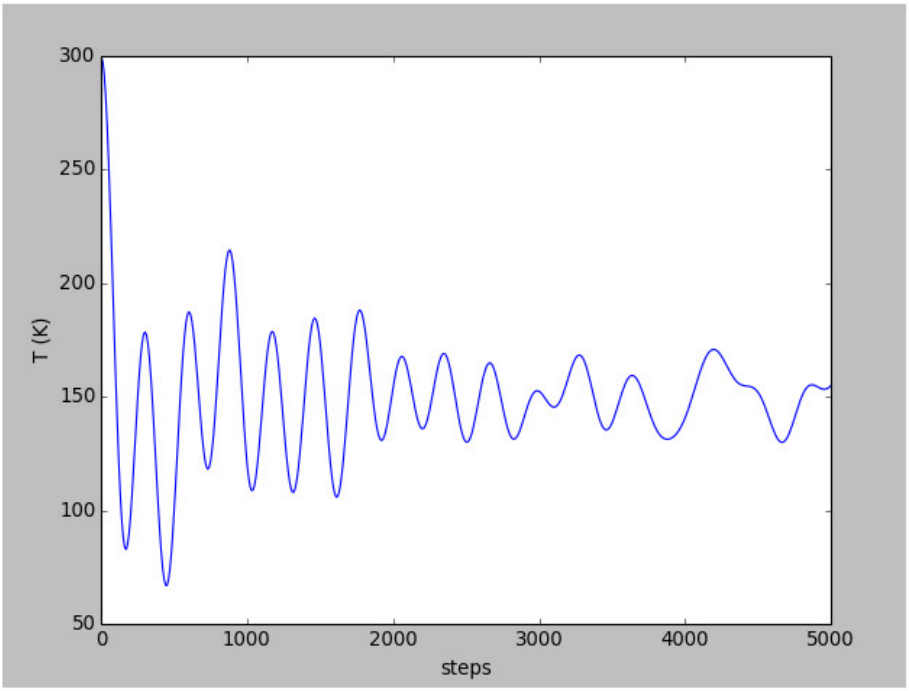

Figure 1: Figure obtained with the script si_temp

5 Command line arguments

With Python we can pass extra arguments from the command line when we run MD++. In this way, one input file can perform a set of different but related tasks, depending on the command line arguments. In the following example (`si-arg`) we can choose whether to run the simulation or to read the output file (only to see the result if the simulation has already been run) with the first argument, and the initial temperature with the second argument.

```
# -*- python script-*-
# MD code of Stinger-Weber Silicon

#*****************************************
# Definition of procedures
#*****************************************

import mdpp

...   same as before   ...

#*****************************************
# Main program starts here
#*****************************************

import sys

if len(sys.argv) == 1 :
    status = 0
elif len(sys.argv) > 1 :
    status = int(sys.argv[1])
print(status)

if len(sys.argv) <= 2 :
    T_OBJ = 300.0
elif len(sys.argv) > 2 :
    T_OBJ = float(sys.argv[2])
print(T_OBJ)

initmd(T_OBJ)

if status == 0 :
    #create crystal, relax, run MD/NVE simulation

    makecrystal(4,4,4)

    openwindow()

    relax_fixbox()
    mdpp.cmd('finalcnfile = relaxed.cn writecn')
    mdpp.cmd('finalcnfile = relaxed.lammps writeLAMMPS')
    mdpp.cmd('finalcnfile = relaxed.cfg writeatomeyecfg')
    setup_md()
    mdpp.cmd('T_OBJ = {0}'.format(T_OBJ))
    mdpp.cmd('initvelocity')
    mdpp.cmd('run')

    mdpp.cmd('finalcnfile = si100.cn writecn')
    mdpp.cmd('finalcnfile = si100.cfg writeatomeyecfg')
```

```
elif status == 1 :
    #read in relaxed, NVE

    mdpp.cmd('incnfile = si100.cn readcn')
    openwindow()

else :
    print('unknown argument. {0}'.format(status))
    mdpp.cmd('quit')

#*****************************************
# Plot simulation data
#*****************************************
print "Plot simulation data using pylab"
from pylab import *
prop = loadtxt('prop.out')
plot( prop[:,0], prop[:,3] )
xlabel("steps")
ylabel("T (K)")
ion()
show()

#*****************************************
# Sleep
#*****************************************
import time
sleep_seconds = 60
print "Python is going to sleep for " + str(sleep_seconds) + " seconds."
time.sleep(sleep_seconds)
```

if we run this by typing

```
$ make sw build=R SYS=gpp PY=yes
$ /bin/sw_gpp scripts/Examples/Python/si-arg.py 0 300.0
```

then we get the same results as before (and as if we put no arguments)
However, if we run

```
$ /bin/sw_gpp scripts/Examples/Python/si-arg.py 1 300.0
```

Then we only read the previous results and plot the temperature.