NETCDF implementation for molecular dynamic

An example of interfacing NETCDF data with the interacting plotting program xmgrace

Storing dynamic molecular data

- **Purpose : building post-traitement diagnostic tools**
  - Needs
    - Store by time
      - Epot, Ekin, Stress Tensor
    - Store by time and by atoms
      - Position and celerity
    - volume cell and primitive vector.

      We limited ourself in the case of unchanging cell
  - Library use : lib netcdf
    - Avantage : Portable
      : Selfdescriptive

Storing dynamic molecular data

- **Structure of a NetCdf file**
  - - header
    - variable definition, attribute definition of variable
    - global attribute
  - - variable value
- **Netcdf file is a binary file**
- **utilities are provided with libNetCdf**
  - ncgen : ASCII --> NetCdf file
  - ncdump : NetCdf --> ASCII

Storing dynamic molecular data

- **Purpose : building a post-traitement diagnostic tools**
  - modify program : moldyn.f

    loop on itime
    - compute Epot, Ekin, Stress Tensor
    - compute Position and celerity by atom
    - if (itime == 0) write_moldyn_netcdf_header
    - Stocking data every userid time
    - if (mod(itime, userid) == 0)write_moldyn_netcdf_value

    End loop

  - Programs add :
    - write_moldyn_netcdf_header.f
    - write_moldyn_netcdf_value.f
    - handle_err_netcdf.f

Storing dynamic molecular data

```
subroutine write_moldyn_netcdf_header(dtfil, dtset, natom, ncoord, nelt_strten)
 ...../...
     ! Creating file netcdf
      status = nf90_create(ficname, NF90_CLOBBER , ncid)
      if ( status /= nf90_NoErr) call handle_err_netcdf(status)

     ! Defining dimension
     ! Dimension time for netcdf (time dim is unlimited)
      status = nf90_def_dim(ncid, "time", nf90_unlimited, timeDimid)
      if ( status /= nf90_NoErr) call handle_err_netcdf(status)
 .../...
     ! Atoms Dimensions
      status = nf90_def_dim(ncid, "NbAtoms", natom, NbAtomsid)
     if ( status /= nf90_NoErr) call handle_err_netcdf(status)

     ! Defining variables
     ! E_pot
      status = nf90_def_var(ncid, "E_pot", nf90_double , &
     &                                timeDimid, E_potDimid)
     if ( status /= nf90_NoErr) call handle_err_netcdf(status)
```

Storing dynamic molecular data

```
 subroutine write_moldyn_netcdf_header(dtfil, dtset, natom, ncoord, nelt_strten)
...../....
 status = nf90_put_att(ncid, E_potDimid, "units", "hartree")
 if ( status /= nf90_NoErr) call handle_err_netcdf(status)

! Celerity
status = nf90_def_var(ncid, "Celerity", nf90_double , &
&              (/ NbAtomsid,  DimCoordid, timeDimid /), Celid)
 if ( status /= nf90_NoErr) call handle_err_netcdf(status)

status = nf90_put_att(ncid, Celid, "units", "bohr/(atomic time unit)")
 if ( status /= nf90_NoErr) call handle_err_netcdf(status)

! Leaving define mode
 status = nf90_enddef(ncid)
 if ( status /= nf90_NoErr) call handle_err_netcdf(status)

 status = nf90_close(ncid)
 if ( status /= nf90_NoErr) call handle_err_netcdf(status)
end subroutine write_moldyn_netcdf_header
```

Storing dynamic molecular data

```
 subroutine write_moldyn_netcdf_value(itime1, dtfil, dtset, Epot, Ekin,  nbat, &
 &                nbdir, nb1, pos, cel, stress, rprimd, ucvol)
       …/..
       ! Opening file netcdf
        status = nf90_open(ficname, nf90_write, ncid)
       if ( status /= nf90_NoErr) call handle_err_netcdf(status)

       ! Ekin
        status = nf90_inq_varid(ncid, "E_kin", E_kinId)
        if ( status /= nf90_NoErr) call handle_err_netcdf(status)
        status = nf90_put_var(ncid, E_kinId, (/ Ekin /), start = (/ itime1 /), count = (/ 1 /) )
        if ( status /= nf90_NoErr) call handle_err_netcdf(status)
…/…
       status = nf90_close(ncid)
        if ( status /= nf90_NoErr) call handle_err_netcdf(status)

end subroutine write_moldyn_netcdf_value
```

NetCdf File

```
ncdump -h t21.outA_moldyn1.nc
netcdf t21.outA_moldyn1 {
dimensions:
        time = UNLIMITED ; // (101 currently)
        DimTensor = 6 ;
        DimCoord = 3 ;
        NbAtoms = 64 ;
        DimVector = 3 ;
        DimScalar = 1 ;
variables:
        double E_pot(time) ;
                E_pot:units = "hartree" ;
        double E_kin(time) ;
                E_kin:units = "hartree" ;
        double Stress(time, DimTensor) ;
                Stress:units = "hartree/bohr^3" ;
        double Position(time, DimCoord, NbAtoms) ;
                Position:units = "bohr" ;
        double Celerity(time, DimCoord, NbAtoms) ;
                Celerity:units = "bohr/(atomic time unit)" ;
        double PrimitiveVector1(DimVector) ;
        double PrimitiveVector2(DimVector) ;
        double PrimitiveVector3(DimVector) ;
        double Cell_Volume(DimScalar) ;
                Cell_Volume:units = "bohr^3" ;
}
```

Post-traitement visualising tools

- **Post-traitement visualising tools**

- **Purpose :**
  - visualising in one window all case caracteristics
  - taking benefit of all fonctionality of freeware viewer

  - case name
  - graphs : Ptot , Etot , T , Epot
  - textual information : Etot, Temper, Ptot average value
  - other textual information : date, author

- **Viewer : freeware Xmgrace**

- **Developpement Langage : python (freeware, language oriented objet, scripting langage)**

Scientific.IO.NetCDF package

- **Interface Netcdf/Python**
- **Package Scientific.IO.NetCDF**
  - **Class NetCDFFile** :
    - standart Attributes : dimensions and variables (dictionaries)
    - Constructor : **NetCDFFile(filename, mode)** : Create/Open a netcdf file
    - **createVariable(varName, datatype, dimensions)**
      - varName : name of the variable
      - datatype : type of the variable
        - ‘ f ’ : float
        - ‘ d ’ : double precision float
        - ‘ i ’ or ‘ l ’ : int or long
        - ‘ c ’ : character
        - ‘ b ’ : byte
      - dimensions : python tuple

      …/…
  - **Class NetCDFVariable**
    - NetCDFVariable objects behave much like array defined in module Numeric
    - standart Attributes : shape , dimensions .../…
    - Methods : getValue(), assignValue()

DiagAbinit sources

```
def ConvNetCdf_Ascii(file) :
    #read netcdf file , compute Epot, Etot, T, P and store it in ascii file.
    ncfile = NetCDFFile(file, 'r')
    var = ncfile.variables['E_pot']
    E_pot =  var.getValue()
    for dimsize in E_pot.shape:
        Nbtimes = dimsize

    # 27.2113834 : conversion factor 1 Hartree = 27,2113834 eV
    Ha_eV = 27.2113834

    #Energie en eV
    #Conversion E_pot (hartree en eV)
    E_pot = Ha_eV * E_pot
    EcrireFichierTemporaire('EPOT', [Nbtimes] , E_pot)
```

DiagAbInit sources

```
#-------------------------------------------------------------|
#---------------   Program main    ----------------------|

AFFI = 'XMGR'
AFFICHEUR = '/usr/local/freeware/bin/xmgr'
localDir=os.getcwd()

#reading netcdf abinit output file
#writing Etot, P, Epot, Temperature in ascii file
FDR = lireNetcdf()

#compute and write caracteristics values in file label
FXMGR = open("label", 'w')
moy('ETOT', FXMGR)
moy('TEMPER', FXMGR)
moy('PRESS', FXMGR)
.../..
FXMGR.close()
```

DiagAbinit sources

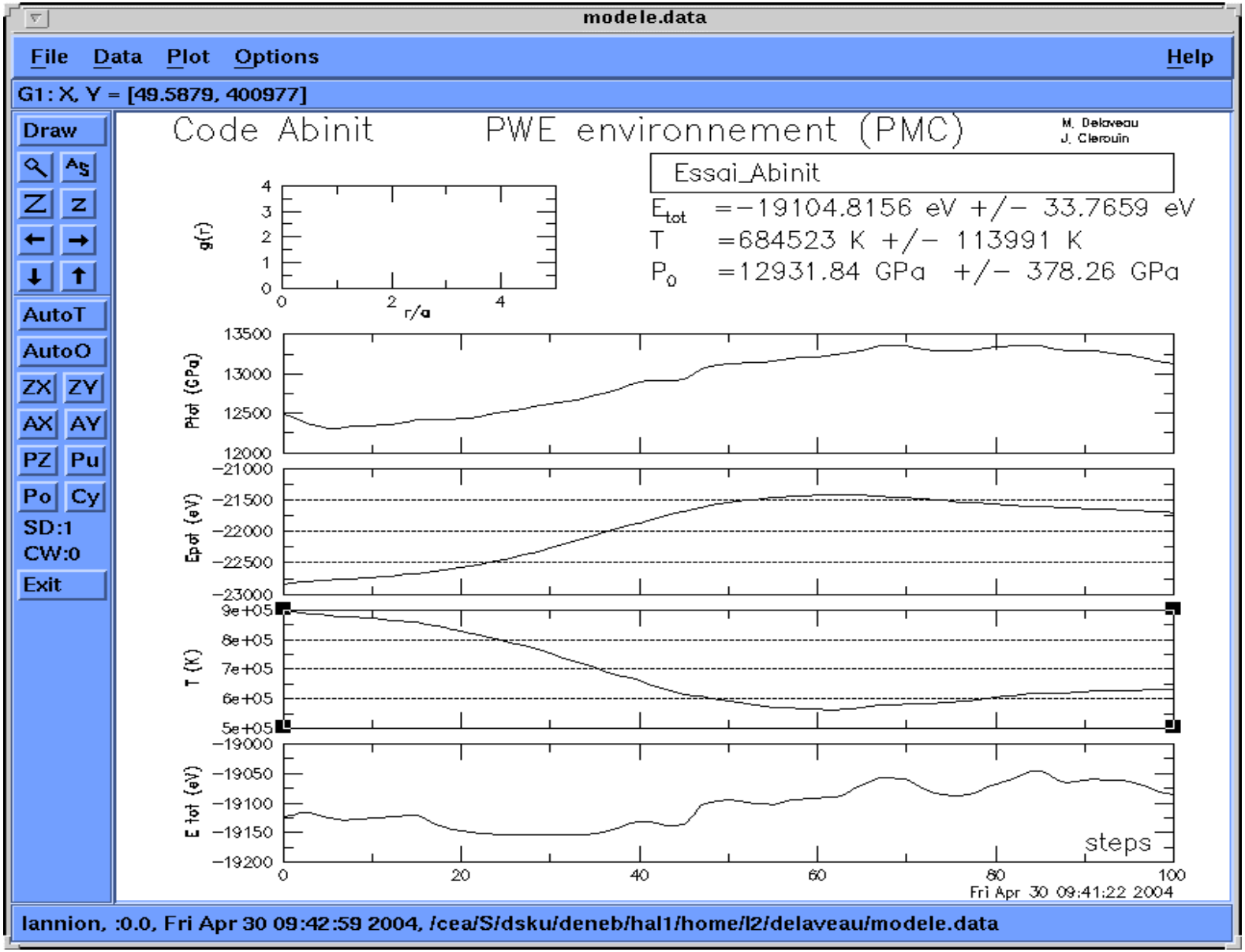
```
fichpara = '/cea/S/home/pwe/pwe/Sources/diag/bilanAbinit' + AFFI + '.para'

#getting model xmgr  file description window to copy it locally
os.system('cp ' + fichpara + ' bilan.para')

#concatening text specific case with xmgr model description file
os.system('cat label >> bilan.para')

#Launching viewer
os.system(AFFICHEUR + '' -param bilan.para -graph 0 -autoscale xy ETOT -graph 1 -
    autoscale xy TEMPER -graph 2 -autoscale xy EPOT -graph 3 -autoscale xy PRESS -
    graph 4 -world 0. 0. 5. 4. G2R '')
```

DiagAbinit


modele.data
File Data Plot Options Help
G1 : X, Y = [49.5879, 400977]
Draw
AutoT
AutoO
ZX ZY
AX AY
PZ Pu
Po Cy
SD:1
CW:0
Exit
Code Abinit PWE environnement (PMC)
M. Delaveau
J. Clerouin
Essai_Abinit
E_tot =−19104.8156 eV +/− 33.7659 eV
T =684523 K +/− 113991 K
P_0 =12931.84 GPa +/− 378.26 GPa
g(r)
r/a
Ptot (GPa)
Epot (eV)
T (K)
E tot (eV)
steps
Fri Apr 30 09:41:22 2004
lannion, :0.0, Fri Apr 30 09:42:59 2004, /cea/S/dsku/deneb/hal1/home/l2/delaveau/modele.data