ASE 简明介绍

张文帅（wszhang@ustc.edu.cn）

中国科学技术大学超级计算中心

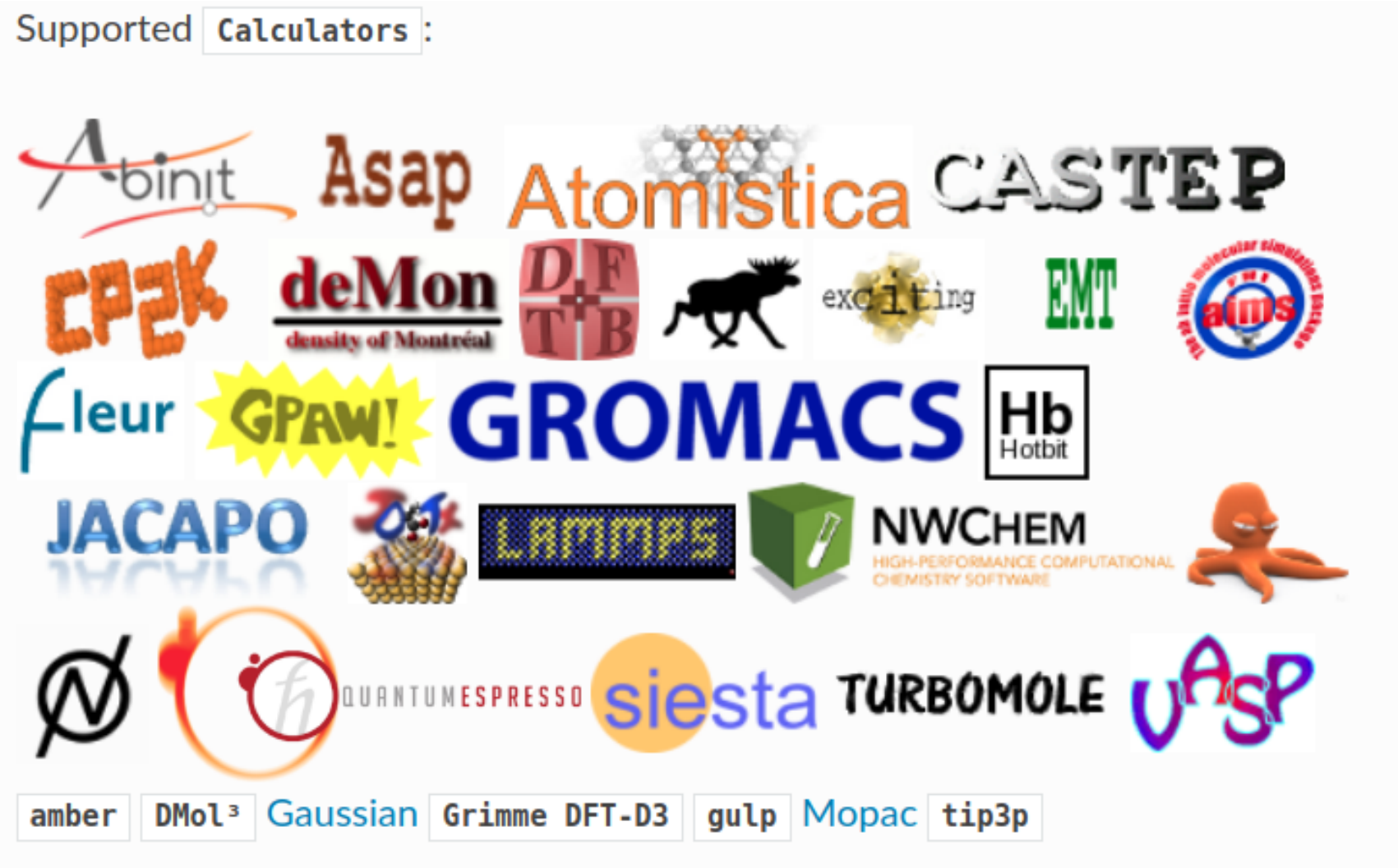
January 4, 2018

目录

简介

ASE is an Atomic Simulation Environment written in the Python programming language with the aim of setting up, manipulating, manipulating, running, visualizing, and analyzing atomistic simulations.

Supported `Calculators`:



设计目标

- Easy to use & Flexible
  - graphical user interface, Command line tool, “for-loop” constructions
- Customizable
  - Python code in ASE is structured in modules
  - ase.calculators for calculating energies, forces and stresses
  - ase.md and ase.optimize modules for controlling the motion of atoms
  - constraints objects and filters for performing nudged-elastic-band calculations etc.
- Pythonic
  - interactively, popular NumPy package, analyse & plot
- Open
  - CAMPOS Atomic Simulation Environment is released under the GNU LGPL 2.1 or any later version.

源码结构

```
[wszhang@node450 test_mpi]$ls /opt/ase/3.13.0/ase/
atom.py       cluster          ga             md                phonons.py          units.py
atom.pyc      collections      geometry       neb.py            quaternions.py      units.pyc
atoms.py      constraints.py   gui            neighborlist.py   run.py              utils
atoms.pyc     data             infrared.py    neighborlist.pyc  spacegroup          vibrations
autoneb.py    db               __init__.py    optimize          structure.py        visualize
build         dft              __init__.pyc   parallel.py       test                xrdebye.py
calculators   dimer.py         io             parallel.pyc      thermochemistry.py
cli           eos.py           lattice        phasediagram.py   transport
[wszhang@node450 test_mpi]$ls /opt/ase/3.13.0/ase/calculators/
abinit.py           dacapo.py    gaussian.py          lj.py             siesta
aims.py             demon        general.py           loggingcalc.py    singlepoint.py
amber.py            dftb.py      gromacs.py           mopac.py          test.py
ase_qmmm_manyqm.py  eam.py       __init__.py          morse.py          tip3p.py
calculator.py       elk.py       __init__.pyc         neighborlist.py   turbomole.py
calculator.pyc      emt.py       interfacechecker.py  nwchem.py         vasp.py
castep.py           exciting.py  interface.py         octopus.py        vdwcorrection.py
checkpoint.py       ff.py        jacapo               onetep.py
cp2k.py             fleur.py     lammpsrun.py         qmmm.py
[wszhang@node450 test_mpi]$ls /opt/ase/3.13.0/ase/optimize/
basin.py                bfgs.py        __init__.py  minimahopping.py  precon
bfgslinesearch.py       fire.py        lbfgs.py     oldqn.py          sciopt.py
bfgslinesearch.py~yuan  fmin_bfgs.py   mdmin.py     optimize.py       test
```

模块与功能

https://wiki.fysik.dtu.dk/ase/ase/ase.html

- The Atoms object
- Units
- File input and output
- Building things
- Equation of state
- Collections
- The data module
- Structure optimization
- Molecular dynamics
- Constraints
- Using the spacegroup subpackage
- Building neighbor-lists
- Geometry tools
- A database for atoms
- Nudged elastic band
- Genetic Algorithm
- ASE's GUI
- General crystal structures and surfaces
- Nanoparticles and clusters
- Visualization
- Calculators
- Density Functional Theory
- Vibration analysis
- Phonon calculations
- Phase diagrams and Pourbaix diagrams
- Thermochemistry
- Utillity functions and classes
- Parallel calculations
- Dimer method
- The Atom object
- Electron transport

Example

```
>>> # Example: structure optimization of hydrogen molecule
>>> from ase import Atoms
>>> from ase.optimize import BFGS
>>> from ase.calculators.nwchem import NWChem
>>> from ase.io import write
>>> h2 = Atoms('H2',
...            positions=[[0, 0, 0], [0, 0, 0.7]])
>>> h2.calc = NWChem(xc='PBE')
>>> opt = BFGS(h2)
>>> opt.run(fmax=0.02)
BFGS:    0  19:10:49     -31.435229      2.2691
BFGS:    1  19:10:50     -31.490773      0.3740
BFGS:    2  19:10:50     -31.492791      0.0630
BFGS:    3  19:10:51     -31.492848      0.0023
>>> write('H2.xyz', h2)
>>> h2.get_potential_energy()
-31.492847800329216
```

安装

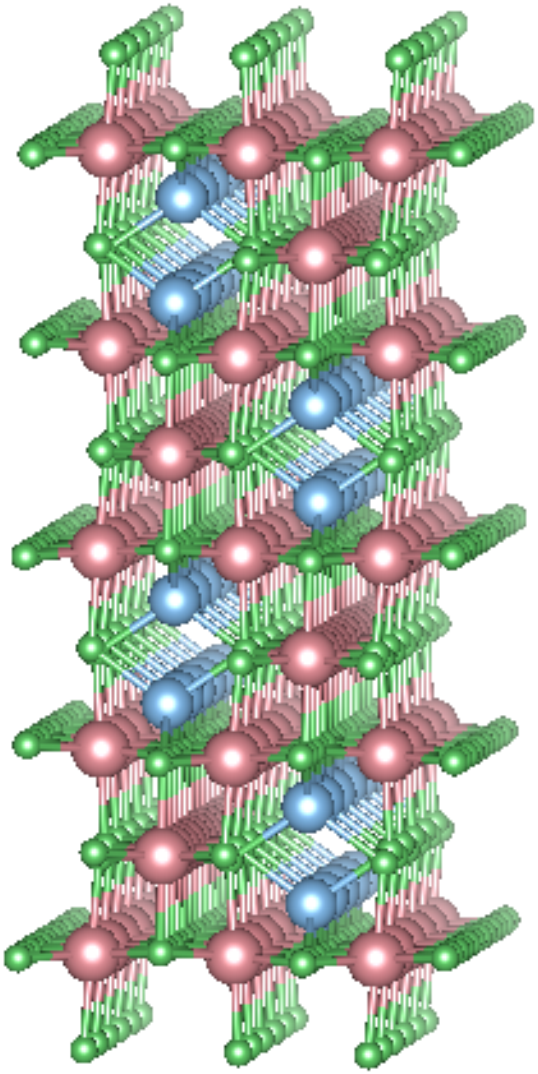
```
Requirements
    Python 2.7, 3.4-3.6
    NumPy (base N-dimensional array package)
Optional, For extra functionality:
    SciPy (library for scientific computing)
    For ase.gui: Matplotlib (2D Plotting)

Installation using pip:
$ pip install --upgrade --user numpy scipy matplotlib
$ pip install --upgrade --user ase

As an alternative to pip, you can also get the source from a tar-file or from Git
$ tar -xf ase-3.15.0.tar.gz
$ ln -s ase-3.15.0 ase
```

Finally: Add ~/ase to your PYTHONPATH environment variable.

Parallel calculation: Abinit

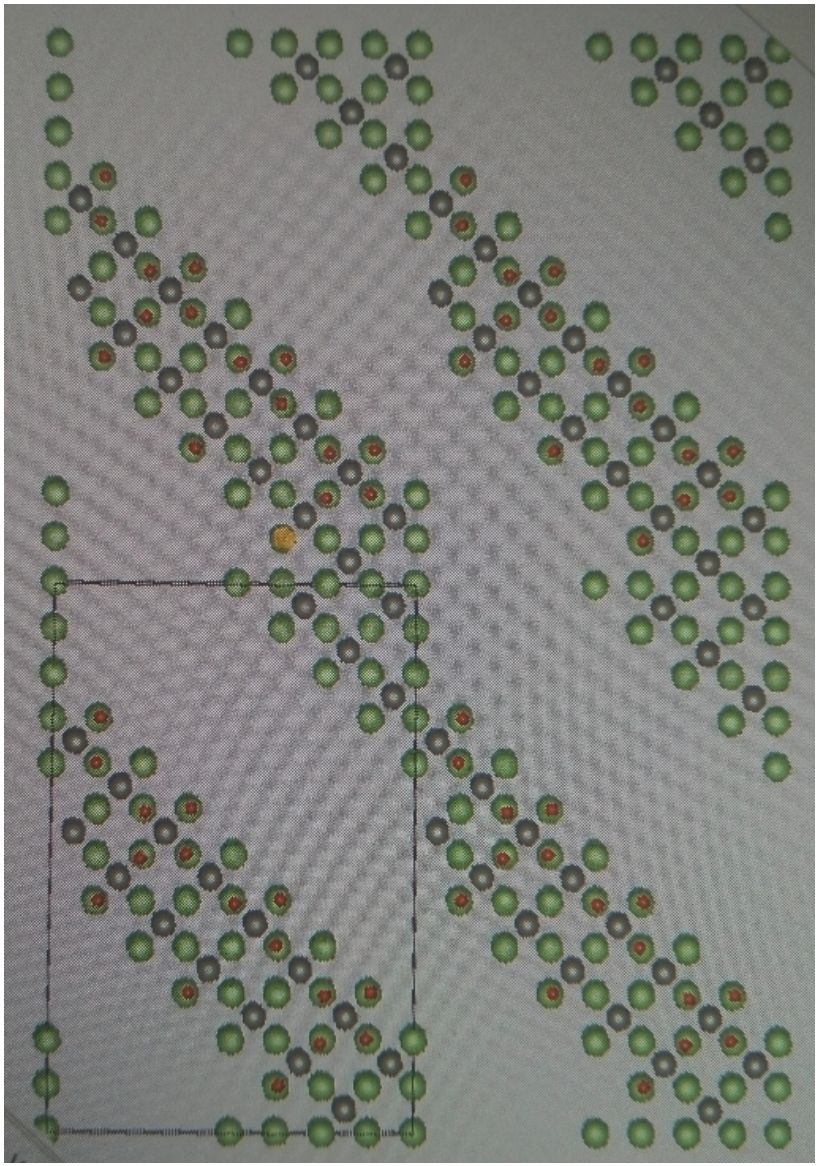
```
from ase import Atoms
from ase.units import Ry
from ase.calculators.abinit import Abinit

a0 = 5.43
bulk = Atoms('Si2', [(0, 0, 0),
                     (0.25, 0.25, 0.25)],
             pbc=True)
b = a0 / 2
bulk.set_cell([(0, b, b),
               (b, 0, b),
               (b, b, 0)], scale_atoms=True)

calc = Abinit(
              command = "mpirun␣-n␣24␣-machinefile␣%s␣abinit␣<␣PREFIX.files␣>␣
                  PREFIX.log"%hostfile, # 关键的设置！！！
              label='Si',
              nbands=8,  # one can specify any abinit keywords
              ecut=10 * Ry,  # warning - used to speedup the test
              kpts=[4, 4, 4],  # warning - used to speedup the test
              chksymbreak=0,
              )

# one can specify abinit keywords also using set
calc.set(toldfe=1.0e-2)  # warning - used to speedup the test
bulk.set_calculator(calc)
e = bulk.get_potential_energy()
```

手工沿着非晶格矢量方向切片很麻烦，容易出错



希望：

错误：



```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
import os
import re
import sys
import time
import subprocess
import string
import copy
import numpy as np
import math
from io import StringIO
from string import ljust
from _elementtree import Element
import ase
import ase.io as aio
import ase.build as abd
import ase.optimize as aopt
import ase.visualize as av
import ase.constraints as ac
from ase import Atoms
from ase.units import Ry

pname = "ZnGaO_5layers_CO2"
#读取原子结构对象:
zngao = aio.read('zngao.cif',format='cif')

```

```
# 创建超胞
# Make SuperCell
#P = np.array([[  3,   0,   0],
#        [  0,   2,   0],
#        [  0,   0,   2]])
#zngaos = abd.make_supercell(zngao, P )


# Cut layers
# 对ZnGaO进行切片，切出5个layer，每个layer面的边矢量为 (2,0,0)，(0,2,-2)，同时容许些许误差：tolerance。
layers = abd.cut(zngao, (2,0,0), (0,2,-2), nlayers=5, origo=(0,0,-0.02), tolerance=0.015, extend=1.0)
# 对切片后的slab进行旋转，晶格矢量也一起旋转：
layers.rotate( (0,1,1), (0,0,1), rotate_cell=True)
layers.set_positions( layers.positions.round(decimals=6,) )
layers.set_cell( layers.cell.round(decimals=6,) )
# 增加第三个晶格矢量z方向的长度，建立真空层：
layers.cell[2][2] = 20.
# 重新对原子结构对象中的原子进行排序，重分配原子序号：
slab = abd.sort(layers)
# 将原子结构平移，置于晶格中央：
slab.center(axis=(0,1,2),)  #axis=(0,1,2),) #vacuum=10.0,about=(1., 1., 1.))


```

切片、扩胞、旋转、替换删除添加、吸附、固定、轨迹 III

```
#设定需要添加的分子结构与旋转取向:
h_CO2 = 2.0
CO2 = abd.molecule('CO2')
CO2.rotate( (0,1,0), (0,0,1), rotate_cell=False)
CO2.rotate( (0,0,1), math.pi/4, rotate_cell=False)

h_CO2H = 2.0
CO2H = abd.molecule('CO2')
CO2H.rotate( (0,1,0), (0,0,1), rotate_cell=False)
CO2H.rotate( (0,0,1), math.pi/4, rotate_cell=False)
CO2H.append(ase.Atom('H') )
CO2H.positions[3] = np.array([-1.2,  1.2,  0.  ])

a = 15.0
CO2.set_cell([ (a, 0, 0),
               (0, a, 0),
               (0, 0, a) ], )#scale_atoms=True)
CO2.set_pbc(pbc=True)
CO2.center(axis=(0,1,2), )#vacuum=10.0,about=(1., 1., 1.))


# 改变特定条件的原子的元素号:
for i in range(len(slab) ) :
    if  slab[i].symbol=='Ga' and abs(slab[i].position[2]-12.947)<0.1 :
        slab[i].symbol = 'Ce'
slab[7].symbol='Ga'
# 删除某序号的原子:
del slab[21]
```

```
# 添加新原子
atom_add = ase.Atom('Ga', [8.3803965000000016, 14.062033500000002,
      15.947150499999999])
print atom_add
slab = slab + atom_add
print slab[-1]


# 添加分子预定吸附位置 add_adsorbate
abd.add_adsorbate(slab, CO2, h_CO2, position=(8.380,14.062) )
stru = slab


# setting PROPERTY
stru.set_pbc((True, True, True))  # a.pbc = (True, True, False)


# 设定结构中的移动受限的原子 Constraints
#constraintlist = []
#constraintlist.append(fix1layer)
#constraintlist.append( ac.FixAtoms(mask=[True for atom in slab ]) )
#constraintlist.append( ac.FixAtoms(indices=[4]) )
#for i in [5,15,16]:
#    constraintlist.append( ac.FixedLine( i, [0,1,0]) )
#
fix1layer = ac.FixAtoms(indices=[atom.index for atom in stru
```

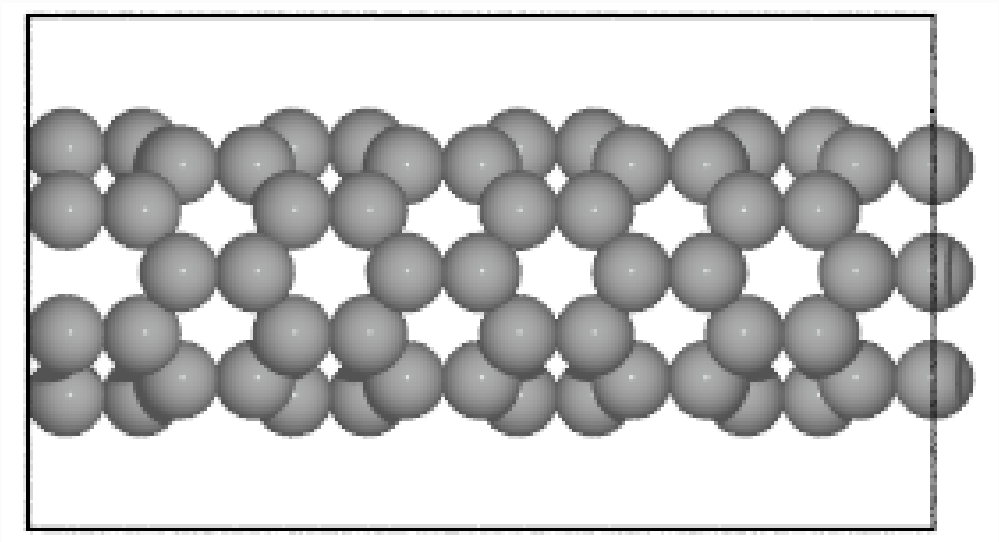
切片、扩胞、旋转、替换删除添加、吸附、固定、轨迹 V

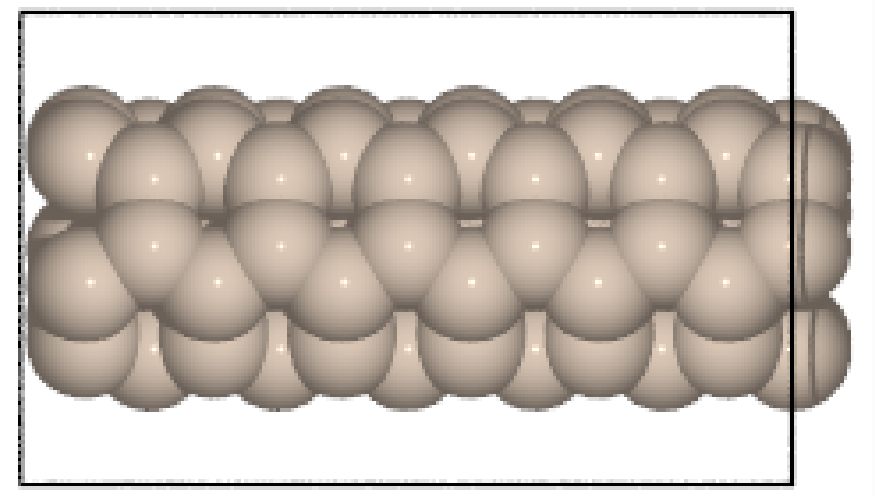
```
            if (atom.position[2] < 7.5 and atom.position[2] > 6.5 and atom.symbol
                == 'Ga' ) ] )
#fix1layer = ac.FixAtoms(indices=[0] )
#constraintlist = [fix1layer]
stru.set_constraint( [fix1layer,] )


# Filters 对晶格进行限制
#from ase.constraints import StrainFilter
#fixed = StrainFilter(stru)
#optobj = ac.UnitCellFilter( stru ) #hydrostatic_strain=True ) #mask=[True,True,
    False,False,False,False] )


# 打开/读取/准备 历史轨迹文件:
# open & read & save traj
#stru_final = aio.read(pname+'.traj',)
#av.view(stru_final)  #:图形化查看原子轨迹，可动画播放
#
traj_hist  = aio.Trajectory(pname+'.traj',) #mode='r')
#stru = traj_hist[-1]
#av.view(traj_hist)
#
traj = aio.Trajectory(pname+'.traj', 'w', stru) #(pname+'.traj', 'a'/'w' , stru)
#traj.write()
```

```
# View & Print & Save Structure
print "\n stru.get_chemical_symbols():\n", stru.get_chemical_symbols()
#print "\n stru.arrays:\n",      stru.arrays  #CO2.positions #CO2.
     get_scaled_positions()
print "\n stru.cell:\n",        stru.cell    #CO2.get_cell() CO2.numbers
print "\n stru.pbc:\n",         stru.pbc
print "\n stru.constraints:\n", stru.constraints
#aio.write(pname+'.ini.pov' , stru, rotation='0x', run_povray=True)
aio.write(pname+'.ini.png' , stru, rotation='-80x')
aio.write(pname+'.ini.vasp', stru, format='vasp')
aio.write(pname+'.ini.xyz',  stru, format='xyz')
aio.write(pname+'.ini.cif',  stru, format='cif')
print "\n saved initial atoms structure "
#av.view(stru)
```

内置常用结构

```
>>> from ase.build import molecule
>>> atoms = molecule('H2O')

>>> from ase.build import nanotube
>>> cnt1 = nanotube(6, 0, length=4)
>>> cnt2 = nanotube(3, 3, length=6, bond=1.4, symbol='Si')
```





生成 kpoints I

ase.dft.kpoints.monkhorst_pack, ase.dft.kpoints.special_points

```
>>> from ase.dft.kpoints import *
>>> monkhorst_pack((4, 1, 1))
array([[-0.375,  0.   ,  0.   ],
       [-0.125,  0.   ,  0.   ],
       [ 0.125,  0.   ,  0.   ],
       [ 0.375,  0.   ,  0.   ]])
>>> get_monkhorst_pack_size_and_offset([[0, 0, 0]])
(array([1, 1, 1]), array([ 0.,  0.,  0.]))

>>> from ase.dft.kpoints(import special_paths, special_points, parse_path_string)
>>> paths = special_paths['bcc']
>>> paths
[['G', 'H', 'N', 'G', 'P', 'H'], ['P', 'N']]
>>> points = special_points['bcc']
>>> points
{'H': [0.5, -0.5, 0.5], 'N': [0, 0, 0.5], 'P': [0.25, 0.25, 0.25],
 'G': [0, 0, 0]}
>>> kpts = [points[k] for k in paths[0]]  # G H N G P H
>>> kpts
[[0, 0, 0], [0.5, -0.5, 0.5], [0, 0, 0.5], [0, 0, 0], [0.25, 0.25, 0.25], [0.5,
     -0.5, 0.5]]
```

生成 kpoints II

ase.dft.kpoints.special_points

Special points from [Setyawana-Curtarolo]:

Cubic	GXMGRX,MR	
FCC	GXWKGLUWLK,UX	
BCC	GHNGPH,PN	
Tetragonal	GXMGZRAZ,XR,MA	../../_images/tetragonal.svg
Orthorhombic	GXSYGZURTZ,YT,UX,SR	
Hexagonal	GMKGALHA,LM,KH	
Monoclinic	GYHCEM1AXH1,MDZ,YD	

Run CP2k Structure Optimization I

https://wiki.fysik.dtu.dk/ase/gallery/gallery.html

```
# 此处省略结构构建
# CP2K
from ase.calculators.cp2k import CP2K
#CP2K.command = "mpijob cp2k_shell.popt"
#os.environ['ASE_CP2K_COMMAND'] = "mpijob cp2k_shell.popt"
#print   "   ASE_CP2K_COMMAND: \n",os.environ['ASE_CP2K_COMMAND'],"\n"
inp_str = '''
&GLOBAL
!  RUN_TYPE CELL_OPT    !GEO_OPT !MD
  PRINT_LEVEL MEDIUM      !MEDIUM LOW
&END GLOBAL
&FORCE_EVAL
   !METHOD Quickstep
   !STRESS_TENSOR ANALYTICAL
   &DFT
      &QS
         EPS_DEFAULT 1.0E-10
      &END QS
      !
      &SCF
         SCF_GUESS ATOMIC  !RESTART !ATOMIC
         EPS_SCF 1.0E-6
         ADDED_MOS 200
         CHOLESKY INVERSE
         &DIAGONALIZATION
            ALGORITHM STANDARD
```

Run CP2k Structure Optimization II

```
27          &END DIAGONALIZATION
28          &MIXING ON
29             METHOD BROYDEN_MIXING   !_NEW   ! BROYDEN_MIXING_NEW BROYDEN_MIXING
                    MULTISECANT_MIXING
30             ALPHA 0.1
31             BETA 1.5
32             !NBROYDEN 8
33          &END MIXING
34          &SMEAR ON
35             !METHOD FERMI_DIRAC    !FERMI_DIRAC !ENERGY_WINDOW
36             !ELECTRONIC_TEMPERATURE [K] 600
37             METHOD ENERGY_WINDOW
38             WINDOW_SIZE [eV] 0.2
39          &END SMEAR
40       &END SCF
41       !
42       &MGRID
43          NGRIDS 4
44          CUTOFF %s
45          REL_CUTOFF %s
46       &END MGRID
47       !
48       !&FORCES ON
49       !&END FORCES
50    &END DFT
51 &END FORCE_EVAL
52 !
53 !&MOTION
```

Run CP2k Structure Optimization III

```
!    &CELL_OPT
!       KEEP_ANGLES
!       !KEEP_SYMMETRY
!       !EXTERNAL_PRESSURE [GPa] 0
!       OPTIMIZER CG
!       &CG
!         &LINE_SEARCH
!           TYPE 2PNT
!         &END LINE_SEARCH
!       &END CG
!       TYPE DIRECT_CELL_OPT  !GEO_OPT
!    &END CELL_OPT
!    !&GEO_OPT
!    !   TYPE MINIMIZATION
!    !   MAX_DR 1.0E-03
!    !   MAX_FORCE 1.0E-03
!    !   RMS_DR 1.0E-03
!    !   RMS_FORCE 1.0E-03
!    !   MAX_ITER 200
!    !   OPTIMIZER BFGS
!       !&BFGS
!       ! MAX_STEEP_STEPS 0
!       ! RESTART_LIMIT 9.0E-01
!       !&END BFGS
!    !&END GEO_OPT
!    !&CONSTRAINT
!    !   &FIXED_ATOMS
!    !     COMPONENTS_TO_FIX XYZ
```

Run CP2k Structure Optimization IV

```
!    !     LIST 2
!    !   &END FIXED_ATOMS
!    !&END CONSTRAINT
!&END MOTION
            '''
calc = CP2K(#
            command = 'mpijob␣cp2k_shell.popt',
            label    = pname,
            #inp = inp_str%cutoff ,
            potential_file    = 'GTH_POTENTIALS' , # POTENTIAL
            pseudo_potential = 'auto' ,
            basis_set_file    = 'BASIS_MOLOPT_addUCL' , # 'BASIS_MOLOPT_UCL', #'
                BASIS_MOLOPT',
            basis_set         = 'DZVP-MOLOPT-SR-GTH' ,
            xc   = 'PBE', # B3LYP, PADE
            #uks = True,
            #charge = 0,
            cutoff   = None, #400 * Ry, #default:400*
            max_scf = 300,
            #debug    =    True,
            #PRINT_LEVEL
            )
cutoff = 1000
rel_cutoff = 60
#for cutoff in [1400]: #600, 800, 1000, 1200]:
     #[200,300,400,500,600,700,800,900,1000]:
#  for rel_cutoff in [40,60,80,100]:
## one can specify abinit keywords also using set :
```

Run CP2k Structure Optimization V

```
##calc.set( toldfe = 1.0e-5 )
calc.set( inp = inp_str%(cutoff,rel_cutoff) )
slab.set_calculator(calc)
#e0_slab = slab.get_potential_energy()
#print "\n cutoff = %s, rel_cutoff = %s, e0_slab = %s "%(cutoff,rel_cutoff,
    e0_slab)
#    os.system( " grep -E -6 \"MULTIGRID INFO\" %s  | tail -n 8 "%(pname+".out")
    )


# Dynamics
#dyn = aopt.BFGSLineSearch(fixed,trajectory=pname+'.traj') #, restart='***.pckl')
dyn = aopt.BFGSLineSearch(slab )#, restart='layers_COPT.traj')
#
#dyn.attach(traj.write)
dyn.attach(traj)
#dyn.replay_trajectory('layers_COPT.traj')
#
#dyn.run(fmax=0.002,steps=150)
dyn.run(fmax=0.02)


e_slab = slab.get_potential_energy()
print "\n␣e_slab␣␣␣␣=␣", e_slab
```

Outputs I

```
   ncpus_tot = 72,  hostdict = {'node109': 24, 'node111': 24, 'node110': 24}

slab.cell:
[[ 16.6716      0.          0.       ]
 [  0.         23.577203    0.       ]
 [ -0.          0.         20.       ]]

slab.pbc:
[ True  True  True]

slab.constraints:
[FixAtoms(indices=[0, 4, 5, 11, 15, 16, 27, 37, 40, 44, 45, 51, 55, 56, 67, 77])]

saved initial atoms structure
                Step[ FC]     Time          Energy          fmax
BFGSLineSearch:    0[  0] 12:08:45  -285534.437492        2.6137
BFGSLineSearch:    1[  2] 12:29:12  -285545.964819       10.2057
BFGSLineSearch:    2[  4] 13:14:37  -285549.406292        1.1386
...
BFGSLineSearch:  206[268] 15:34:38  -285556.418339        0.0255
BFGSLineSearch:  207[269] 15:38:29  -285556.418555        0.0841
BFGSLineSearch:  208[273] 15:59:44  -285556.418613        0.0420
BFGSLineSearch:  209[274] 16:04:11  -285556.418761        0.0751
BFGSLineSearch:  210[275] 16:08:49  -285556.418875        0.0209
BFGSLineSearch:  211[276] 16:13:11  -285556.418995        0.0123

e_slab    =  -285556.418995
```

Structure Optimization Banchmark

C5H12

Geometry optimization of gas-phase molecule.

Calculator used: GPAW (lcao)

Optimizer	Optimizer Steps	Force evaluations	Energy	Time [sec]	Note
BFGS	10	10	-85.26102	32	
LBFGS	10	10	-85.26102	32	
LBFGSLineSearch	11	26	-85.26267	69	
FIRE	38	38	-85.26096	80	
MDMin	21	21	-85.26111	66	
SciPyFminCG	9	18	-85.26218	46	
SciPyFminBFGS	11	21	-85.26165	46	
BFGSLineSearch	8	15	-85.26182	58	
GoodOldQuasiNewton	13	13	-85.26216	36	

nanoparticle

Adsorption of a NH on a Pd nanoparticle.

Calculator used: GPAW (lcao)

Optimizer	Optimizer Steps	Force evaluations	Energy	Time [sec]	Note
BFGS	28	28	-16.23916	109	
LBFGS	57	57	-16.25660	185	
LBFGSLineSearch	16	60	nan	158	An exception occurred
FIRE	73	73	-16.25149	191	
MDMin	47	47	-16.25064	158	
SciPyFminCG	17	48	nan	153	An exception occurred
SciPyFminBFGS	17	18	-16.24096	75	

Molecular dynamics

```
## 此处省略结构构建

dyn = VelocityVerlet(atoms, dt=5.0 * units.fs,
                     trajectory='md.traj', logfile='md.log')

dyn.run(1000)  # take 1000 steps
```

Nudged Elastic Band I

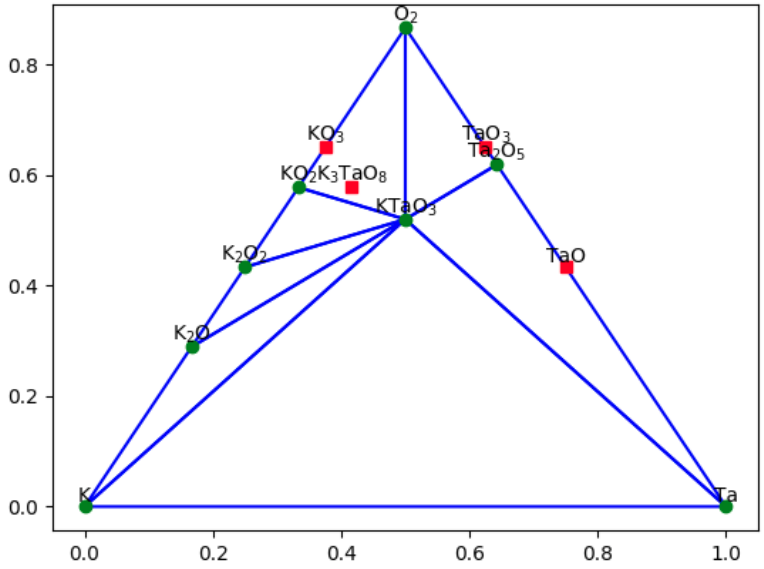
```
from ase import io
from ase.neb import NEB
from ase.optimize import MDMin
# Read initial and final states:
initial = io.read('A.traj')
final = io.read('B.traj')
# Make a band consisting of 5 images:
images = [initial]
images += [initial.copy() for i in range(3)]
images += [final]
neb = NEB(images)
# Interpolate linearly the potisions of the three middle images:
neb.interpolate()
# Set calculators:
for image in images[1:4]:
    image.set_calculator(MyCalculator(...))
# Optimize:
optimizer = MDMin(neb, trajectory='A2B.traj')
optimizer.run(fmax=0.04)

#-----------------------------------------------------------

#Parallelization over images
from ase.parallel import rank, size
from ase.calculators.emt import EMT
# Number of internal images:
n = len(images) - 2
j = rank * n // size
```

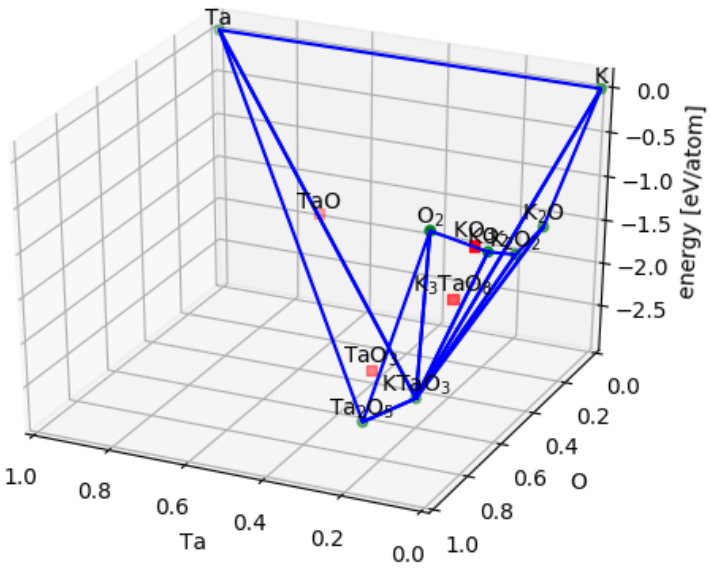
Nudged Elastic Band II

```
for i, image in enumerate(images[1:-1]):
    if i == j:
        image.set_calculator(EMT())
```

Phase Diagrams

```
# creates: ktao-2d.png, ktao-3d.png
import matplotlib.pyplot as plt
from ase.phasediagram import PhaseDiagram
references = [('K', 0), ('Ta', 0), ('O2', 0),
              ('K3TaO8', -16.167), ('KO2', -2.288),
              ('KO3', -2.239), ('Ta2O5', -19.801),
              ('TaO3', -8.556), ('TaO', -1.967),
              ('K2O', -3.076), ('K2O2', -4.257),
              ('KTaO3', -13.439)]
pd = PhaseDiagram(references)
for d in [2, 3]:
    pd.plot(dims=d, show=False)
    plt.savefig('ktao-{}d.png'.format(d))
```





计算声子： phonons.py

```
class Displacement:
    """Abstract base class for phonon and el-ph supercell calculations.

    Both phonons and the electron-phonon interaction in periodic systems can be
    calculated with the so-called finite-displacement method where the
    derivatives of the total energy and effective potential are obtained from
    finite-difference approximations, i.e. by displacing the atoms. This class
    provides the required functionality for carrying out the calculations for
    the different displacements in its ``run`` member function.

    Derived classes must overwrite the ``__call__`` member function which is
    called for each atomic displacement.

    """

    def __init__(self, atoms, calc=None, supercell=(1, 1, 1), name=None,
                 delta=0.01, refcell=None):
        """Init with an instance of class ``Atoms`` and a calculator.

        Parameters:

        atoms: Atoms object
            The atoms to work on.
        calc: Calculator
            Calculator for the supercell calculation.
        supercell: tuple
            Size of supercell given by the number of repetitions (l, m, n) of
            the small unit cell in each direction.
        name: str
            Base name to use for files.
        delta: float
            Magnitude of displacement in Ang.
        refcell: str
            Reference cell in which the atoms will be displaced. If ``None``,
            corner cell in supercell is used. If ``str``, cell in the center of
```

Calculator: abinit.py I

```
58 class Abinit(FileIOCalculator):
59     """Class for doing ABINIT calculations.
61     The default parameters are very close to those that the ABINIT
62     Fortran code would use.  These are the exceptions::
64       calc = Abinit(label='abinit', xc='LDA', ecut=400, toldfe=1e-5)
65     """
67     implemented_properties = ['energy', 'forces', 'stress', 'magmom']

78     def __init__(self, restart=None, ignore_bad_restart_file=False,
79                  label='abinit', atoms=None, scratch=None, **kwargs):

118    def write_input(self, atoms, properties=None, system_changes=None):

263    def read(self, label):
264        """Read results from ABINIT's text-output file."""

276    def read_results(self):
           ...
335        # Forces:
336        for line in lines:
337            if line.rfind('cartesian forces (ev/angstrom) at end:') > -1:
338                forces = []
339                for i in range(natoms):
340                    forces.append(np.array(
341                            [float(f) for f in next(lines).split()[1:]]))
342                self.results['forces'] = np.array(forces)
343                break
344        else:
```

Calculator: abinit.py II

```
29 345                raise RuntimeError
30 346            #
31 347            self.width = self.read_electronic_temperature()
32 348            self.nband = self.read_number_of_bands()
33 349            self.niter = self.read_number_of_iterations()
34 350            self.nelect = self.read_number_of_electrons()
35 351            self.results['magmom'] = self.read_magnetic_moment()
36
37 457        def read_number_of_iterations(self):
38
39 467        def read_electronic_temperature(self):
40
41 478        def read_number_of_electrons(self):
42
43 496        def get_kpts_info(self, kpt=0, spin=0, mode='eigenvalues'):
44
45 514        def read_magnetic_moment(self):
46
47 524        def get_fermi_level(self):
48
49 527        def get_eigenvalues(self, kpt=0, spin=0):
50
51 530        def get_occupations(self, kpt=0, spin=0):
```