engn2770_computational_tutorial

September 16, 2020

1 ENGN 2770: Atomistic Reaction Engineering

1.1 Computational Tutorial

This is the pdf version. The interactive version (which includes necessary environment files) can be accessed from using the link:

https://mybinder.org/v2/gh/sharma1908shubham/engn2770_computational_tutorial/3267ff13e77706eb2fcd0e41b14989e0f60ab69f

(Note: This can take a minute or two to load and download required environment files)

Once you open the link, you can download the "engn2770_computational_tutorial.ipynb" and save it on your local computer.

Note, you will need jupyter notebook along with other necessary packages if you wish to use it on your local machine(ASE, python, matplotlib, numpy).

1.2 A) Intro to linux bash commands (https://files.fosswire.com/2007/08/fwunixref.pdf)

[3]:
```
from IPython.display import Image
```

[4]:
```
Image(filename='extras/linux_tutorial_page1.png')
```

[4]:

Job	Command
identify the directory you are in	**pwd**
see the files in the directory	**ls**
see the files in the directory with hidden files	**ls -a**
go to your home directory	**cd (or, cd ~/)**
change directory to <directory_name>	**cd** <directory_name>
move one directory up	**cd ..**
move two directories up	**cd ../../**
make directory <directory_name>	**mkdir** <directory_name>
copy a file	**cp** </path/to/file/file_name> </new/path/to/file/new_file_name>
remove a file / remove a file recursively (careful -- no way to revert if you delete a file/directory)	**rm** <file_name> / **rm -r** <file_name>
create an empty file	**touch** <file_name>

[5]:
```
Image(filename='extras/linux_tutorial_page2.png')
```

[5]:

Job	Command
move a file	**mv** </path/to/file/file_name> </new/path/to/file/new_file_name>
display the contents of the file	**cat** <file_name>
copy file from your local computer to CCV	**scp -r** </path/to/location/on/computer/file_name> <username@ssh.ccv.brown.edu:/path/to/new/location/file_name>
text editors	**vim/ vi/ nano/ gedit**

Environment Setup:

$ cd

(vi/nano/gedit) ~/.bashrc

Inside your .bashrc, type

$ source /gpfs/data/ap31/ap31/teach/2020-engn2770/settings/engn2770packages

Save your .bashrc, and type

$ source ~/.bashrc (This needs to be done only once - i.e whenever you change your ~/.bashrc)

$ engn2770_loadase (This needs to loaded every time you will be using ASE)

$ python3

$ import ase

$ ase.__file__ (The output should point to '/users/ap31/data/ap31/teach/2020-engn2770/packages/ase/ase/__init__.py'

Note: The first line in your python script should be always be #!/usr/bin/env python3

1.3 B) Python

1.3.1 Basic arithmetic

[6]:
```
# Basic arithmetic
a = 2.
b = 3.

print(a + b)
```

```
5.0
```

[7]:
```
# square of a number
sq = a**2
print(sq)
```

```
4.0
```

[8]:
```
# division & %
print(4./2)

# remainder of a division
print(15.%8)
print(8.%2) # If this is 0, then 8 is divisible by 2!
```

```
2.0
7.0
0.0
```

1.3.2 Functions

[9]:
```
# functions

# define add function
def add(a,b):
    return a+b

# define power function
def power(a,pow=2):
    return a**pow

# call add function
add_result = add(2.,3.)
print(add_result)

# call power function
power_result = power(5.,3)
print(power_result)
```

```
5.0
125.0
```

[10]:
```
sum_result = sum((2.,2.))
print(sum_result)

pow_result = pow(5.,4)
print(pow_result)
```

```
4.0
625.0
```

1.3.3 numpy, list & arrays

[11]:
```
# using numpy (Numerical Python)
import numpy as np

sum_result = np.sum((2.,-3.))
print(sum_result)

power_result = np.power(2.,8)
print(power_result)
```

```
-1.0
256.0
```

[12]:
```
# list
things = ['a',7]

for thing in things:
    print(thing)
```

```
a
7
```

[13]:
```
# Basic list commands

_list = [1,2,3,4]

#_list.append(5)
#_list.pop(2)
#_list.reverse()

print(_list)
```

```
[1, 2, 3, 4]
```

[14]:
```
import numpy as np

# return evenly spaced numbers over a specified interval

a = np.linspace(1,12,12)
print(a)
print(a.shape)

# convert a 1D array into 2D
a.reshape(3,4)
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12.]
(12,)
```

[14]:
```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.],
       [ 9., 10., 11., 12.]])
```

1.3.4 For loop & conditional statements

[15]:
```
# for loops
for element in range(5):
    print(element, element**2)
```

```
0 0
1 1
2 4
3 9
4 16
```

[16]:
```
# For loops with conditional statements

dice = [1,2,3,4,5,6]

type_list = []

for roll in dice:
    if roll % 2 == 0:
        type_list.append('even')
    else:
        type_list.append('odd')

print(type_list)
```

```
['odd', 'even', 'odd', 'even', 'odd', 'even']
```

[17]:
```
num = np.linspace(0,10,11)
print(num)
```

```
doubled_list = []

for element in num:
    if element > 5:
        doubled_list.append(element*2)

print(doubled_list)
```

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[12.0, 14.0, 16.0, 18.0, 20.0]
```

[18]:
```
# For loop with conditional statement in one line

num = np.linspace(0,10,11)
print(num)

doubled = [element * 2 for element in num if element>5]
print(doubled)
```

```
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
[12.0, 14.0, 16.0, 18.0, 20.0]
```

[19]:
```
long_words = ['platinum', 'hi', 'carbon', 'welcome', 'hydrogen', 'neptune', 'covid',␣
 ↪'python']

# checks and stores into new list based on length of a word
short_words = [word for word in long_words if len(word) < 6]

short_words
```

[19]:
```
['hi', 'covid']
```

1.4 C) Atomic Simutlation Environment

[20]:
```
# Some basic packages we will need throughout

import os
import numpy as np

import ase
from ase.parallel import paropen
from ase.io import read, write
from ase.visualize import view

# Plotting and image visualizing in Jupyter Notebook
import matplotlib
import matplotlib.pyplot as plt
from IPython.display import Image

%matplotlib inline
```

2 Module 1: Atom / Atoms

[21]:
```
# import the module (use tab tab for auto-completion)
from ase import Atom, Atoms
```

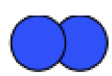
[22]:
```
# use ? to know more
#atom = Atom?
```

[23]:
```
atom = Atom('O')
atom.mass = 16
atom.charge = -2
atom.position = (0,2,3)
#atom.symbol
#atom.x
#atom.y
```

[24]:
```
# use ? to know more
#Atoms?
```

[25]:
```
d = 1.104 # N2 bondlength

# The following three are equivalent
N2 = Atoms('N2', [(0, 0, 0), (0, 0, d)])
N2 = Atoms(numbers=[7, 7], positions=[(0, 0, 0), (0, 0, d)])
N2 = Atoms([Atom('N', (0, 0, 0)), Atom('N', (0, 0, d))])

# to save and view a snapshot
write('extras/n2.png', N2, rotation='-80y')
Image(filename='extras/n2.png')

# to have a 3D view in notebook
#view(N2, viewer='x3d')
```

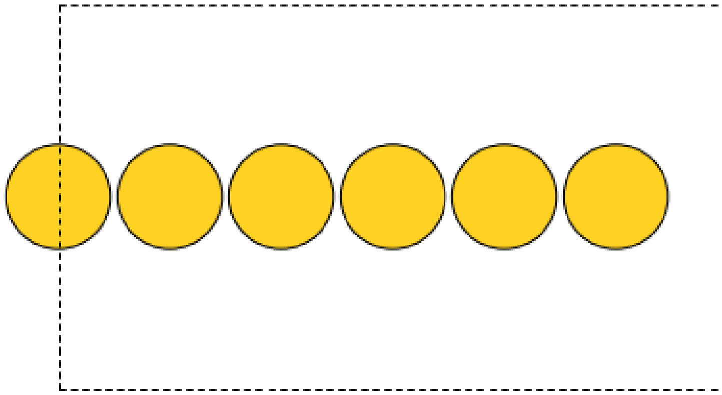
[25]:



[26]:
```
d = 2.9
L = 10.0

# construct a Au wire
wire = Atoms('Au',
             positions=[[0, L / 2., L / 2.]],
             cell=[d, L, L],
             pbc=[1, 0, 0])

wire.get_chemical_symbols()

# to save and view a snapshot
write('extras/wire.png', wire * (6,1,1))
Image(filename='extras/wire.png')
```

```
#view(wire*(6,1,1), viewer='x3d')

#view(wire*(6,1,1))
```

[26]:

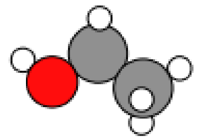


3 Module 2: Build (molecule, bulk, surfaces)

[27]:
```
from ase.build import molecule

m = molecule('CH3CH2OH')
print(m.get_chemical_symbols())
print(m.get_chemical_formula())
print(m.get_positions())

# to save and view a snapshot
write('extras/ethanol.png', m, rotation='-20x')
Image(filename='extras/ethanol.png',width=150)

#view(m, viewer='x3d')

#view(m)
```

```
['C', 'C', 'O', 'H', 'H', 'H', 'H', 'H', 'H']
C2H6O
[[ 1.168181 -0.400382  0.      ]
 [ 0.        0.559462  0.      ]
 [-1.190083 -0.227669  0.      ]
 [-1.946623  0.381525  0.      ]
 [ 0.042557  1.207508  0.886933]
 [ 0.042557  1.207508 -0.886933]
 [ 2.115891  0.1448    0.      ]
 [ 1.128599 -1.037234  0.885881]
 [ 1.128599 -1.037234 -0.885881]]
```

[27]:



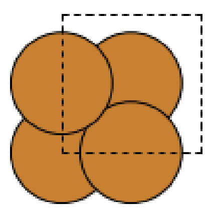
4 Optimize lattice constant

[28]:

```
from ase.build import bulk
from ase.calculators.emt import EMT
from ase.io import Trajectory

a0 = 3.6
cu = bulk('Cu', 'fcc', a=a0, cubic=True)

print(cu.get_cell())

# to save and view a snapshot
write('extras/cu.png', cu)
Image(filename='extras/cu.png',width=150)

#view(cu, viewer='x3d')

#view(cu)
```

```
Cell([3.6, 3.6, 3.6])
```

[28]:

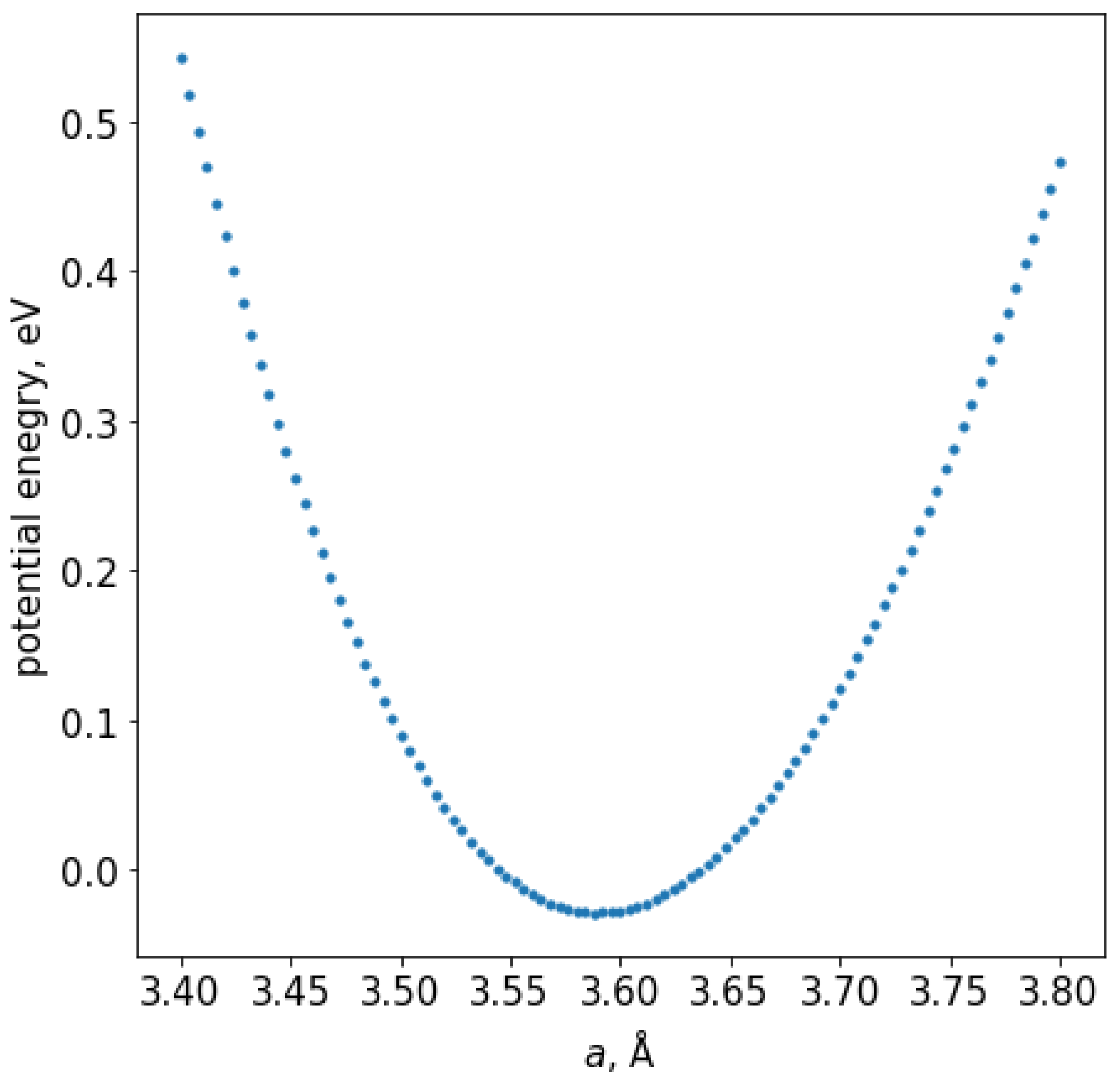


[29]:

```
# create an empty list to store energy
energy_list = []

# start with a good guess
# and create a list with fluctuations in a.

a0 = 3.6
a_list = a0 + np.linspace(-0.2, 0.2, 101)

# For loop to use each and every value of a
# and calculate & store the potential energy
```

```
for a in a_list:
    cu = bulk('Cu', 'fcc', a=a, cubic=True)
    cu.set_calculator(EMT())
    e = cu.get_potential_energy()
    energy_list.append(e)
```

[30]:
```
# plotting using matplotlib (https://matplotlib.org/)

fig, ax = plt.subplots(figsize=(7,7))
plt.scatter(a_list, energy_list, marker='o', s=10)
plt.xlabel('$a$, $\mathrm{\AA}$', fontsize=15)
plt.xticks(fontsize=15)
plt.ylabel('potential enegry, eV', fontsize=15)
plt.yticks(fontsize=15)
fig.savefig('extras/lattice_constant.png')
plt.show()
```



[31]:

```python
# index corresponding to the lowest energy

index = energy_list.index(min(energy_list))
lc = a_list[index]
print('lattice constant:%0.3f'%lc)
```

```
lattice constant:3.588
```

5 Module 3: Surface, Constraints

[32]:

```python
from ase.build import fcc111
from ase.constraints import FixAtoms
```

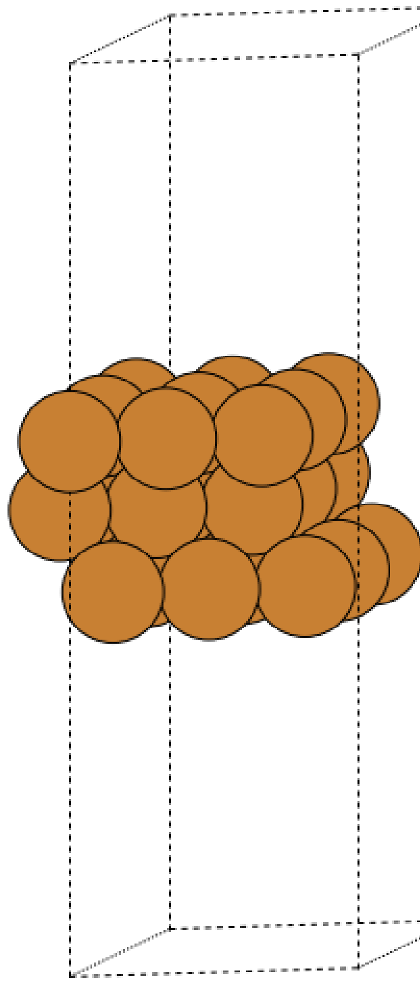
[33]:

```python
#fcc111?
```

[34]:

```python
# Create a slab using build
slab = fcc111('Cu', size=(3,3,3), a=3.588, vacuum=10)
slab.set_pbc((1,1,0))

# Use of constraints, here, Fix the bottom most layer
indices=[atom.index for atom in slab if atom.tag == 3]
constraint = FixAtoms(indices)
slab.set_constraint(constraint)

# Note that the constraint is not visible through
# the following method

write('extras/cu_slab.png', slab, rotation='10z,-80x')
Image(filename='extras/cu_slab.png')

#view(slab, viewer='x3d')
#view(slab)
```

[34]:



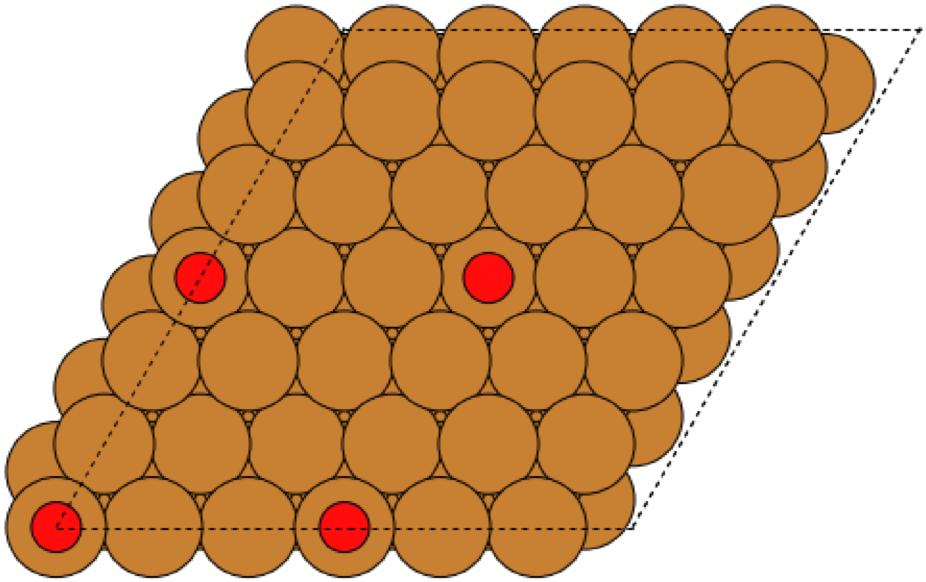
6 Module 4: Add an adsorbate

method 1

[35]:
```
from ase.build import add_adsorbate

# create a slab
slab = fcc111('Cu', size=(3,3,3), a=3.588, vacuum=10)
slab.set_pbc((1,1,0))

# create an atom and use add_adsorbate to add it to the slab
adsorbate = Atom('O')
add_adsorbate(slab, adsorbate, 1.8, 'ontop')

write('extras/slab_with_adsorbate.png', slab * (2, 2, 1))
Image(filename='extras/slab_with_adsorbate.png')
```

```
#view(slab, viewer='x3d')
#view(slab)
```

[35]:

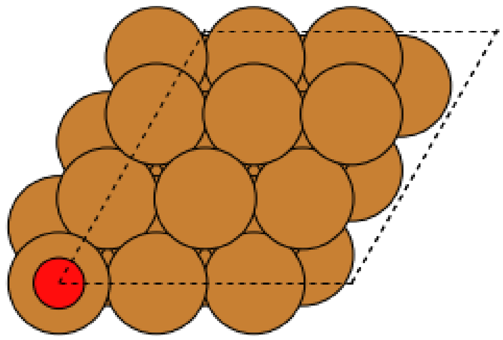


method 2

[36]:

```
a = 3.558

# create a slab
slab = fcc111('Cu', size=(3,3,3), a=3.588, vacuum=10)
slab.set_pbc((1,1,0))

# use the slab atom position to add it to the slab
adsorbate = Atom('O')
adsorbate.position = slab[18].position + (0,0,2)

slab_with_add = slab + adsorbate

write('extras/slab_with_adsorbate.png', slab_with_add)
Image(filename='extras/slab_with_adsorbate.png')

#view(slab_with_add, viewer='x3d')
#view(slab_with_add)
```

[36]:



7 Module 5: Structure optimization using EMT

[37]:
```python
from ase.constraints import FixAtoms
from ase.build import fcc111, add_adsorbate
from ase.calculators.emt import EMT
from ase.optimize import BFGS

# Construct a slab with optimized lattice constant
slab = fcc111('Cu', size=(3,3,3), a=3.588, vacuum=10)
# Periodic boundary conditions
slab.set_pbc((1,1,0))

# Construct & add an adosorbate
adsorbate = Atom('O')
add_adsorbate(slab, adsorbate, 1.8, 'ontop')

# Fix atoms
indices=[atom.index for atom in slab if atom.tag == 3]
constraint = FixAtoms(indices)
slab.set_constraint(constraint)

# Calculator
calc = EMT()
slab.set_calculator(calc)

if os.path.exists('qn.log'):
    os.remove('qn.log')

# Optimization
opt = BFGS(slab, logfile='qn.log', trajectory='qn.traj')
opt.run(fmax=0.01)
```

[37]: True

[38]:
```python
#relaxed_slab = read('qn.traj')
#view(relaxed_slab)
```

[39]:
```
myfile = open("qn.log",'r')
txt = myfile.read()
print(txt)
```

```
      Step     Time          Energy         fmax
BFGS:    0 17:33:42        6.520552        0.9096
BFGS:    1 17:33:42        6.505566        0.4225
BFGS:    2 17:33:42        6.501307        0.0559
BFGS:    3 17:33:42        6.501107        0.0437
BFGS:    4 17:33:42        6.500731        0.0845
BFGS:    5 17:33:42        6.500402        0.0943
BFGS:    6 17:33:43        6.500170        0.0693
BFGS:    7 17:33:43        6.500026        0.0385
BFGS:    8 17:33:43        6.499856        0.0668
BFGS:    9 17:33:43        6.499623        0.0791
BFGS:   10 17:33:43        6.499416        0.0562
BFGS:   11 17:33:43        6.499321        0.0183
BFGS:   12 17:33:43        6.499287        0.0128
BFGS:   13 17:33:43        6.499261        0.0155
BFGS:   14 17:33:43        6.499236        0.0165
BFGS:   15 17:33:44        6.499223        0.0082
```

[40]:
```
from ase.constraints import FixAtoms
from ase.build import fcc111, add_adsorbate
from ase.calculators.emt import EMT
from ase.optimize import BFGS

# Construct a slab with optimized lattice constant
slab = fcc111('Cu', size=(3,3,3), a=3.588, vacuum=10)
# periodic boundary condition
slab.set_pbc((1,1,0))

# Construct & add an adosorbate
adsorbate = Atom('O')
add_adsorbate(slab, adsorbate, 1.8, 'hcp')

# Fix atoms constraint to fix the last atomic layer
indices=[atom.index for atom in slab if atom.tag == 3]
constraint = FixAtoms(indices)
slab.set_constraint(constraint)

# Calculator
calc = EMT()
slab.set_calculator(calc)

if os.path.exists('qn.log'):
    os.remove('qn.log')

# Optimization
opt = BFGS(slab, logfile='qn.log', trajectory='qn.traj')
opt.run(fmax=0.01)
```

[40]: True

[41]:
```
myfile = open("qn.log",'r')
txt = myfile.read()
print(txt)
```

```
      Step     Time          Energy         fmax
BFGS:    0 17:33:44        6.570523        1.7805
BFGS:    1 17:33:44        6.508532        1.5692
BFGS:    2 17:33:44        6.388970        0.6331
BFGS:    3 17:33:44        6.368169        0.3576
BFGS:    4 17:33:44        6.363440        0.2807
BFGS:    5 17:33:44        6.346981        0.1420
BFGS:    6 17:33:45        6.345193        0.1026
BFGS:    7 17:33:45        6.343991        0.0898
BFGS:    8 17:33:45        6.342511        0.0919
BFGS:    9 17:33:45        6.340466        0.0910
BFGS:   10 17:33:45        6.339097        0.1304
BFGS:   11 17:33:45        6.338357        0.1316
BFGS:   12 17:33:45        6.337813        0.1097
BFGS:   13 17:33:46        6.337098        0.0630
BFGS:   14 17:33:46        6.336492        0.0408
BFGS:   15 17:33:46        6.336258        0.0134
BFGS:   16 17:33:46        6.336218        0.0105
BFGS:   17 17:33:46        6.336205        0.0090
```

8 Module 6: Calculate Barriers using nudged elastic band (NEB)

[42]:
```
from ase.neb import NEB
from ase.optimize import BFGS
from ase.calculators.emt import EMT

# Create a bare slab
slab = fcc111('Cu', size=(3,3,3), a=3.588, vacuum=10)
slab.set_pbc((1,1,0))

indices=[atom.index for atom in slab if atom.tag == 3]
constraint = FixAtoms(indices)
slab.set_constraint(constraint)

slab.set_calculator(EMT())

opt = BFGS(slab, logfile='bare_slab.log', trajectory='bare_slab.traj')
opt.run(fmax=0.01)

# create an adsorbate
adsorbate = Atom('O')

#######################################
```

```
###### SLAB 1: Cu with O in fcc (position 1) #######
#######################################

initial = slab + adsorbate
initial[-1].position = initial[18].position + (1.3,0.8,1)

initial.set_calculator(EMT())
opt = BFGS(initial, logfile='initial.log', trajectory='initial.traj')
opt.run(fmax=0.01)


####################################################
###### SLAB 1: Cu with O in fcc (position 2) #######
####################################################
final = slab + adsorbate
final[-1].position = final[19].position + (1.3, 0.8, 1)

final.set_calculator(EMT())
opt = BFGS(final, logfile='final.log', trajectory='final.traj')
opt.run(fmax=0.01)

n = 10

images = [initial]
for i in range(n):
    image = initial.copy()
    image.set_calculator(EMT())
    image.get_potential_energy()
    image.get_forces()
    images.append(image)

images.append(final)

neb = NEB(images)
neb.interpolate()

qn = BFGS(neb,logfile='neb.log', trajectory='neb.traj')
qn.run(fmax=0.05)
```

[42]: True