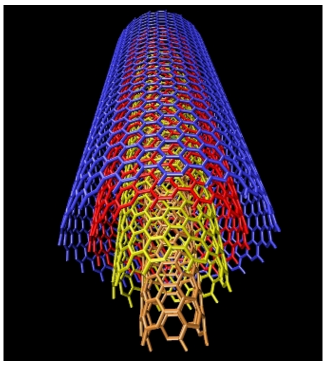
Laboratory work (2 ECTS credits)

Computational Modeling of Carbon Nanotubes

Department of Physics, University of Jyväskylä
Pekka Koskinen (pekka.koskinen@iki.fi)



1 Introduction

This laboratory work is an introduction to methods and concepts in computational materials science. You will use realistic quantum-mechanical methods methods to calculate and investigate electronic structure and the motion of atoms in carbon nanotubes. The emphasis is on learning the infrastructure of computational materials science, but you will also learn physics related to one-dimensional carbon nanotube nanostructures.

The required prerequisites are finished subject studies in physics, and preferably also first materials physics course. We do use quantum-mechanical methods, but work is dominated by classical concepts. It can be completed independently, either at the university or at home, but seeking help is encouraged before problems start to pile up.

Learning the infrastructure takes the most of the time; the steps required to complete the work itself do not take much time, once you are familiar with the programming language and the related environment. We use atomic simulation environment (ASE), which is a modern framework to perform atomistic simulations, used under python programming language.

Before the description of the actual work procedures, this guide gives a short introduction to carbon nanotubes, python programming language, python's atomic simulation environment, and the used electronic structure calculation method.

2 Carbon nanotubes

Carbon nanotubes (CNTs) are one-dimensional (or high aspect ratio) nanostructures from carbon. CNT research was boomed after their 1991 discovery by Ijima, despite preceding observations.[1] Ever since they have been serving a role of a proto-typical nanostructure. They can be found in single- or in multiple-wall form, or in bundles of several intertwined tubes. In this work we consider only single-wall nanotubes.

CNTs can be thought of made by rolling graphene (single layer of graphite) using the concept of the so-called roll-up vector $\boldsymbol{C} = n\boldsymbol{a} + m\boldsymbol{b}$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are the primitive cell unit vectors in graphene, as shown in figure 1. CNTs, and hence all of their properties, are uniquely defined by the index pair (n, m). Tubes $(n, 0)$ are called *zigzag* tubes and (n, n) *armchair* tubes, while general (n, m) tubes are called *chiral* tubes, as demonstrated in figure 2.

During the work you need to search literature to find for information related to CNTs' properties. Some sources and links include:

1. Wikipedia: http://en.wikipedia.org/wiki/Carbon_nanotube
2. Compact list of CNT properties: http://www.photon.t-u-tokyo.ac.jp/ maruyama/kataura/chirality.html
3. The nanotube site: http://www.pa.msu.edu/cmp/csc/nanotube.html
4. *Physical properties of carbon nanotubes* by R. Saito and G. Dresselhaus, limited access in Google books

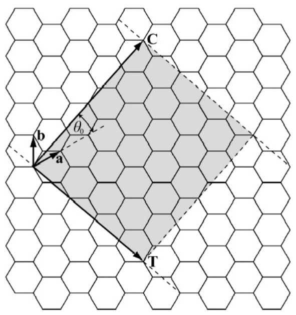


Figure 1: Rolling carbon nanotubes from graphene. The roll-up vector $\boldsymbol{C} = n\boldsymbol{a} + m\boldsymbol{b}$ defines the points the are connected when rolling up graphene. The axis of the nanotube is along $\boldsymbol{T}$ and $\boldsymbol{C}$ is perpendicular to it. The angle between $\boldsymbol{C}$ and the *zigzag* direction, θ, is called the chiral angle.

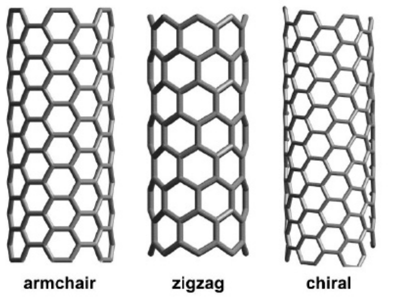


Figure 2: Armchair and zigzag tubes get their names from the appearance of profiles at tube ends; chiral tube gets its name simply because it looks like chiral.

3 Python

Python is an object-oriented *script language* (aka interpreted language). This means that code is interpreted on-fly during execution; code is neither compiled nor linked prior to execution. This makes code development and debugging fast. It can be used also interactively, giving commands one after another directly in python interpreter, which can be (mostly) invoked by the command `python`. This usage resembles the common usage of MATLAB. A typical python script is a single ASCII-file containing the commands, and the script can be run by the command `python script.py`. The script files are often named with the extension `.py`, though it is not compulsory.

For example, a simple python script plotting the function $1 + x^2$ would be:

```
from numpy import *
from pylab import *
x = linspace(0,10,20)
y = 1 + x**2
plot(x,y)
show()
```

Let us look closer what happened in this example.

1. importing module `numpy` enables easier manipulation of vectors and matrices (in MATLAB-style), for instance.
2. import module `pylab` (from matplolib library) for plotting. This has plotting commands similar to those of MATLAB.
3. create equally-spaced array from 0 to 10, using 20 points.
4. create array `y` from `x`, where `y` becomes the function $1 + x^2$ with 20 points.
5. plot `y` versus `x` using connected lines
6. pop-up the window to actually show the plot

A different way to plot the same function might be:

```
from numpy import *
from pylab import *

def function(t):
    return 1+t**2

x = linspace(0,10,20)
y = []
for a in x:
    y.append( function(a) )
scatter(x,y)
show()
```

What is different in this example:

1. define function `function` of one variable `t` that returns $1 + t^2$.
2. create an empty list `y`.
3. go through all values `a` in the array `x`, and append to the list `y` the value of `function` evaluated at `a`.
4. plot `y` versus `x` using scattered points
5. pop-up the window to actually show the plot

Python appears—and is—just as any other programming language. Some characteristic features of python, however, are worth mentioning:

1. code is case-sensitive
2. code is indentation-sensitive. This may feel strange at the beginning, but makes the code neat and easy-to read (also the code from others), since there is little freedom to fiddle with the appearance. Moreover, now you don't need to add characters like }, ;, }; that make codes often messy.
3. variables are not introduced beforehand, but on-fly. Any declaration like `x=...` will create the variable `x`, and sets the variable type (integer, float, text, etc.) to be what the code ... suggests. For instance, `x` after the line `x=1` is an integer, whereas `x` after the line `x=1.0` is a float. Using a variable before introduction will cause an error.
4. array and list indices start from 0 and stop at $N - 1$

If you are not familiar with python, you need to learn it first. This takes time, but it is time well spent. Script languages—python especially—are becoming more familiar among physicists and chemists because of their flexibility and versatility. One tool can make many tasks—calculate, organize, analyze, plot, and glue different applications together. Study the lessons $2-11$ from the tutorial www.sthurlow.com/python, until you feel confident with your skills to continue with the laboratory work. You should return to the tutorial if you find yourself fighting with too many error messages from your scripts. To practice with python, you can install it to your own computer, or use a remote server (see Sec. 5).

Bear in mind this one: Python environment is the test tubes and petri dishes of a chemist. It is the monkey wrench and oscilloscope of a physicist. It is the specimen holder and butterfly net of a biologist. In the frame of this laboratory work, it is the main infrastructure that you must master to succeed. Once you get familiar with python, and understand the phenomena, the laboratory work can be done in one hour. Spend your time by gradually learning the environment and only after start performing calculations by real systems. Play around, test and learn; do not expect your scripts to work at once.

4 Atomic Simulation Environment

Stand-alone python interpreter cannot do much—it is *modules* that makes Python scripts powerful. Atomic simulation environment (ASE) is a python module designed to make simulations and can be imported simply by

```
from ase import *
```

An example is the best illustration to get a feeling of what is to come. In the following script we optimize carbon dioxide (CO_2) molecule and calculate its vibrational spectrum—the calculations in this laboratory work do not become much more difficult:

```
from ase import *
from hotbit import *

atoms = Atoms('OCO',[(0,0,0),(1.16,0,0),(2*1.16,0,0)])
calc = Hotbit(txt='output.cal')
atoms.set_calculator(calc)

optimizer = QuasiNewton(atoms,trajectory='optimization.traj')
optimizer.run()

from ase.vibrations import Vibrations
vib = Vibrations(atoms)
vib.run()
vib.summary()
```

This looks intuitive, right? Let us look closer what happened in this example.

1. import ase module, so that it becomes *available*
2. import hotbit module. This so-called *calculator*-module calculates the forces and energies for our carbon dioxide; this is discussed more later.
3. make the CO_2 molecule. The atoms are O, C and O, and their positions are given in the list, unit of length being Ångstrom. `atoms` is the central object in the script, that carries information about the elements and their positions and velocities, but also information about the simulation box, spanned by three vectors. The simulation cell, the default one implicit in this script, is $1\,Å\times1\,Å\times1\,Å$ cube with one corner at origin. `atoms` carries also information if the system is periodic in each direction; in this example, by default, the system is non-periodic in all directions. Some atoms are outside the simulation cell, but in this example it makes no difference.

4. make the so-called calculator-object, which in this case is a `Hotbit`-calculator. This calculator uses tight-binding theory to calculate energy and forces for atoms with given positions and atomic numbers.
5. let `atoms`-object know what calculator is used to calculate forces and energies.
6. make an optimizer object, using `QuasiNewton` -method, and give it the reference to `atoms`
7. optimize the geometry of carbon dioxide, even though the bond length 1.16 Åshould already be close to optimum. Here the optimizer asks for energies and forces from atoms (which, in turn asks the calculator `calc` to calculate them) several times, and accordingly moves atoms' positions to find the minimum-energy geometry. After optimization the resulting trajectory file `optimization.traj` can be viewed using ASE's graphical user interface from the command line with the command `ag optimization.traj`.
8. import ASE's vibrations-module to enable calculation of vibrational spectra.
9. create a `Vibrations`-object, and give reference to `atoms` (which is now optimized, and we calculate vibration modes around given optimum geometry)
10. `vib.run()` causes the `vib`-object to move atoms around one by one, in order to calculate the Hessian matrix $H_{ij} = \partial^2 E/\partial x_i \partial x_j$. H_{ij} is used to calculate the vibrational eigenmodes of the system, as familiar from classical mechanics.
11. `vib`-object prints the energies of the vibrational modes

The official wiki-pages of ASE are in https://wiki.fysik.dtu.dk/ase.[2] These pages are a great source of practical information, and can be consulted anytime during this work. Note, however, that ASE provides much more methods and tools that we need—don't get lost in the volume of information. The tools in ASE can be used to move atoms, to make molecular dynamics simulations in constant energy (microcanonical MD) or constant temperature (canonical MD), to optimize geometries, or to visualize atoms. The only external help ASE tools need, is a separate calculator that calculates forces and energies. The units in ASE are Ångstrom for length, eV for energy, and atomic mass unit u for mass.

The official wiki provides a set of tutorials and you should go through at least the following ones. For practicing, you can install ASE to your own computer easily, or use the pre-installed ASE on a remote server (see Sec. 5).

1. https://wiki.fysik.dtu.dk/ase/tutorials/manipulating_atoms.html
2. https://wiki.fysik.dtu.dk/ase/tutorials/atomization.html
3. https://wiki.fysik.dtu.dk/ase/tutorials/eos.html
4. https://wiki.fysik.dtu.dk/ase/tutorials/md.html

4.1 ASE Calculator class

This work will use density-functional tight-binding (DFTB), a realistic electronic structure calculation method, to calculate energies and forces for given geometries. DFTB is faster than density-functional theory, and can be run easily on one CPU—most of your scripts should take less than $\sim$ 1 minute. Classical potentials would be faster, but they do not have the electronic structure information and build-in quantum-mechanics that tight-binding has.

The DFTB code, developed here at NanoScience Center, is called `hotbit`, and can be used only as an ASE calculator. The calculator parameters are best given right at initialization:

```
calc = Hotbit(parameter1=value1,parameter2=value2,...)
```

Much knowledge of the method and the quantum mechanics behind the scenes is not needed. Some selected features are here:

1. Hotbit solves single-particle states, the equation $\hat{H}\psi_a = \varepsilon_a \psi_a$, using mean-field potential in the density-functional spirit. Calculations are spin-paired, such that single-particle states have occupation numbers $[0, 2]$.
2. It uses $\boldsymbol{k}$-point sampling for periodic calculations, familiar from Bloch's theorem. For nonperiodic Γ-point calculations we need only one $\boldsymbol{k}$-point, and we have `kpts=(1,1,1)`, which is default. For systems periodic in z-direction we need more $\boldsymbol{k}$-points, e.g. `kpts=(1,1,10)`.

3. Our calculations do not have much charge transfer, since all atoms are carbon. Hence we always set `SCC=False`; `SCC=True` is the default. SCC is a DFTB extension to include self-consistent charge formalism.
4. The parameter `txt` is used to set the name of calculator's output file.

To summarize, the calculator object can be created as

```
calc = Hotbit(SCC=False,txt='script.cal',kpts=(1,1,nkz))
```

where `nkz>1`, since we always have periodicity in z-direction.

For more information, should you need it, read Koskinen & Mäkinen *Density-functional tight-binding for beginners Comp. Mat. Sci.* (doi:j.commatsci.2009.07.013).[3] Hotbit calculator has wiki pages under university's Trac system, https://trac.cc.jyu.fi/projects/hotbit.[4]

This work could be done, in principle, by changing each script line defining the calculator

```
calc = Hotbit(...)
```

to use any other calculator class, such as density-functional calculator GPAW,

```
calc = GPAW(...)
```

and our calculations would be state-of-the-art. It would, however, take more time, we would need to use super-computer and several CPUs—and still the results would not change much.

4.2 Examples

Here we list random code snippets—not stand-alone scripts—to demonstrate how different things can be done. Some are useful for the work.

1. Optimize `atoms`-object, until all atoms' maximum force component is < 0.01 eV/Å, and view the optimized geometry:

```
opt = QuasiNewton(atoms)
opt.run(fmax=0.01)
view(atoms)
```

2. Calculate `atoms`' cohesion energy per atom; move the first atom to origin;

```
eN = atoms.get_potential_energy()
cohesion = eN / len(atoms)
r = atoms.get_positions()
r[0,:] = (0,0,0)
atoms.set_positions(r)
```

3. Calculate same stuff using different calculator parameters:

```
for nkz in [1,2,4]:
    calc = Hotbit(...,kpts=(1,1,nkz))
    atoms.set_calculator(calc)
    qn = QuasiNewton(atoms)
    ...
```

4. Make 10 Å×10 Å×5 Å simulation box, and set the z-direction periodic. Translate atoms such that origin is at $\boldsymbol{R}_{CM}$, then center it with respect to z-coordinate. (Some atoms will be outside simulation cell, which in our examples makes no difference.)

```
atoms.set_cell( [(10,0,0),(0,10,0),(0,0,5)] )
atoms.set_pbc( False,False,True )
atoms.translate(-atoms.get_center_of_mass())
atoms.center(axis=2)
print 'pbc=',atoms.get_pbc(), 'cell=',atoms.get_cell()
```

5. Plot the density of states (DOS) after solving the electronic structure. The zero of the energy axis is shifted to the Fermi-level.

```
atoms.set_calculator(calc)
atoms.get_potential_energy()
e,dos = calc.get_DOS(broaden=True,)
plot(e,dos)
savefig('dos.png')
```

6. Read the last geometry from `optimization.traj`. (If the trajectory results from optimization, geometry is optimized.); scale the atoms' x- and y-coordinates by `c`; perform microcanonical MD with 1 fs time stepping for 100 steps, creating trajectory `md.traj`.

```
atoms = read('optimization.traj')
for atom in atoms:
    atom.x*=c
    atom.y*=c
md = VelocityVerlet(atoms,dt=1.0*fs,trajectory='md.traj')
md.run(100)
```

7. Solve and print vibrational spectrum for an optimized `atoms`-object; save a visualization of the motion of atoms in mode number 27 into file `vib.27.traj` (by default).

```
vib = Vibrations(atoms)
vib.run()
vib.summary()
vib.write_mode(27)
```

8. Make a $(10, 0)$ nanotube derived from 1.42 Å nearest neighbor bond lengths; the simulation cell is automatically periodic in z-direction with the correct height.

```
from box.systems import nanotube
(n,m) = (10,0)
atoms = nanotube('C',1.42,n,m)
```

5 Practical issues

5.1 Server

Calculations are done on a remote linux-server `linuxfs.phys.jyu.fi`. Ask server admin Vesa Apaja (room YN212, phone 4717) for a user account there.

5.2 Unix

We will not use the unix shell command-line much, but you should be familiar with tasks like copying, deleting, making and removing directories, or listing contents of an ASCII-file. If you feel the need for reminder, the link http://www.ee.surrey.ac.uk/Teaching/Unix gives a short unix tutorial.

Using light-weight remote editors on the server is the fastest way to edit scripts. Avoid editing scripts on your local machine. Some may prefer *vi* editor, or *emacs* editor, but if you are new to these, you may prefer a simple and light-weight editor like *nano*. A script is created simply by typing `nano script.py`, and then using the short-cuts shown below to save or exit the editor.

5.3 Preparing the working environment

1. Connect remotely to `linuxfs.phys.jyu.fi` with X11-forwarding.
   - Connections from the computer lab (room FL349): Start local X-server by running *Xming*. Do this once per session. Then use *PuTTY* SSH client to contact to `linuxfs.phys.jyu.fi`. Make sure the *Enable X11 forwarding* is enabled in *Connection / SSH / X11* settings.
   - Connections from elsewhere: you have to find your own solution.
   - From linux/mac: use syntax `ssh -X username@linuxfs.phys.jyu.fi`.
2. Start bash command shell (if not default) by typing `bash`.
3. Type the command `source /home/group/modeling_CNTs/hb`
   This sets up the environment and environmental variables, hence letting python know where to find ASE and hotbit.
4. Make a temporary directory where you launch the hotbit test script with the command
   `/home/group/modeling_CNTs/hotbit-test`
   Test takes bit less than a minute and needs to be done only once. All the tested items should prompt OK.

6 The laboratory work

This section describes the steps in the work. The steps are best done in the given order.

6.1 Preparations: Parameter convergence checks & cohesion energy

First make a $(5,5)$ carbon nanotube, and inspect the structure. Is the box size in z-direction right and the periodicities correct? The box size in x- and y-directions do not matter, once directions are not periodic (only in `hotbit` calculator). Calculate the potential energy with `nkz` $\boldsymbol{k}$-points in the z-direction. How large `nkz` needs to be to converge the total energy, and how do you judge the convergence? Take the limit of convergence $\sim 10^{-4}$ eV/atom.

Optimize the geometry and look at the resulting trajectory with GUI. How much did energy come down? The resulting atoms-object is almost fully optimized, but not quite. What should be done to make sure we have truly the minimum energy for our $(5,5)$-tube? Explain and speculate, but don't do it.

In addition to the $(5,5)$ tube, optimize also $(9,0)$, and $(10,0)$ tubes using the same value for `nkz`, and save the optimizations in a trajectory.

The total energy that ASE returns when using `hotbit` is the total binding energy of the N-atom system. What are the cohesion energies per atom for the $(5,5)$, $(9,0)$, and $(10,0)$ tubes? How do these energies for tubes with different diameters compare to each other and why? For reference, with the parametrization of our DFTB method, the cohesion energy of graphene is 9.63 eV. (Here DFTB makes some error—the real cohesion energy of graphene is closer to 8 eV.)

6.2 Electronic structure and cohesion energy

To inspect their densities of states we need to increase `nkz`; DOS is not converged even though energy is (energy is the integral of DOS and can converge earlier). Plot the converged DOS for $(5,5)$, $(9,0)$, and $(10,0)$ tubes using `nkz=150`. Which tubes are metallic and which are not? Are metallicities consistent with the theory prediction for nanotubes? Further, DOS consists of a series of peaks. What are these peaks called and why are they asymmetric?

6.3 Vibrational properties

Now we look at some aspects of the vibrational properties of carbon nanotubes. Radial breathing mode (RBM) is one of the most important vibration modes. Before you continue, use literature to find out how it looks like and where it is used.

Method 1: RBM energy using finite difference

Take your optimized $(5,0)$ tube and optimize it further, using smaller maximum force `fmax=0.01` in the optimization. Then displace atoms by δR in the right direction to excite RBM manually and calculate the energy difference δE to the optimized energy; this corresponds to

$$\delta E \sim \frac{1}{2}k(\delta R)^2$$

for one-dimensional harmonic oscillator. What value of δR did you choose and why? The default value `fmax=0.05` usually results in reasonably optimized structures, but why did we now use smaller value for `fmax`? Using $\omega = \sqrt{k/m}$ you can calculate the energy $\hbar\omega$ of the RBM phonon.

Hint: Note that δE has to be the energy per atom if m is the mass of one carbon atom in the equation above. Be especially careful with units and unit conversions (what is, for instance, $\hbar$ in ASE units?). Why should you avoid SI units in these calculations?

Method 2: RBM energy using microcanonical MD

Calculate next the RBM phonon energy using another approach. Take your atoms-object with manually excited RBM mode, as above. Perform microcanonical simulation for at least 100 fs time span, using 0.1 fs, 1 fs, 2 fs, 5 fs, and 10 fs time steps, while saving trajectories. What is a suitable time step, taken that in a microcanonical simulation the total energy should be conserved?

Inspecting a realiable 100 fs trajectory (using a reliable time step) with GUI should given you enough information to calculate the RBM phonon energy.

Scaling of RBM with tube diameter

Calculate the RBM phonon energy for $(5,0)$, $(7,0)$, $(10,0)$, $(12,0)$, and $(15,0)$ tubes (take the finite-difference script from above, and make a loop over the chiral index n). Plot the RBM phonon energy versus tube diameter D; it should behave approximately as

$$\hbar\omega \approx \frac{\text{constant}}{D}.$$

How well does your value for the constant compare to values in the literature?

Method 3: RBM using ASE methods

We can also use ASE's sophisticated methods to calculate the whole vibrational spectrum of a tube at once. The first three modes have different energy from all the others. What is this energy and what do these modes correspond to? How many modes are there in total and why?

Calculate the vibrations for $(5,0)$ tube and, using the RBM phonon energy obtained from finite difference calculation, find the RBM among all possible modes, and visualize the mode to check that you picked the correct one. Was the RBM energy the same as the ones obtained from finite difference or molecular dynamics methods?

7 Work report

The only requirement for the report is to be self-contained report with scientific style, that progresses in logical order, and answers in all the questions asked in the previous section. The report can be written either in Finnish or English. The report does *not* need to contain scripts; report is about physics, not about technical particulars (even though you spend much time learning those particulars). Send the report to the address given on the first page as a pdf file.

References

[1] S. Ijima. Helical microtubules of graphitic carbon. *Nature*, 354:56, 1991.

[2] ASE wiki at `https://wiki.fysik.dtu.dk/ase/`.

[3] P. Koskinen and V. Mäkinen. Density-functional tight-binding for beginners. *Comput. Mat. Sci.*, 47:237, 2009.

[4] hotbit wiki at `https://trac.cc.jyu.fi/projects/hotbit`.