Hands-on exercise on laser-matter irradiation with TDDFT

8th School on Time-Dependent Density-Functional Theory
Benasque Center for Science (Benasque, Spain)
August 20th – 27th, 2018

1 Introduction

In these exercises, we will do some simulations on laser matter irradiation. Due to the limited time, we will work mostly with one-electron systems and models, but the level of theory we will be aiming at is that of non-adiabatic Molecular Dynamics (MD) based on time-dependent density-functional theory (TDDFT) and the Ehrenfest equations. If you don't know much about these topics: Refs [1, 2, 3] are interesting references about TDDFT; Refs [6, 7, 8, 10] are some example references for the combination of TDDFT and non-adiabatic first principles MD.

For this purpose, we will use the `octopus` code, a free code that implements these ideas. You can learn the essentials on how it works in Refs.[4, 5].

2 Basics: input file, etc.

The code can be freely downloaded at:
`http://www.octopus-code.org/`

In the web page, you can find some tutorials and documentation. Since time is limited, however, we will follow a "learn by doing" strategy.

The executable is called `octopus`. The code must find, in the directory from where it is launched, an input file that must always be called `inp`. We will be working with a number of sample input files called `inp.x`, where `x` is some number. In order to use one of them, simply copy it to the directory where you are going to do the run, and rename it to `inp`.

Let us start with the first one: `inp.1`:

```
################################################
# Ground state of a 1D model of Hydrogen
CalculationMode = gs

Dimensions = 1

BoxShape = sphere
Spacing = 0.5
Radius = 100.0

TheoryLevel = independent_particles

%Species
"Hydrogen1D" | species_user_defined | potential_formula | "-1/sqrt(1+(x)^2)" | valence | 1
%

%Coordinates
"Hydrogen1D" | 0
%
################################################
```

The lines that start after `#` are comments, not considered by the code. The rest of the lines have the basic syntax "variable = value", except for the "blocks", which start with a line with the syntax `%BlockName` and end with a single %. In between, there are a number of lines formed of various elements separated by a vertical line, `|`. Variables can go in any order, and are case insensitive.

The first input variable, `CalculationMode`, tells the code what kind of calculation is to be performed; in this case, `gs` stands for ground state calculation. The various options of any input file variable can be looked up at the code web page, under the heading "Variable reference". There you will see what other possibilities exist for `CalculationMode`. You can also show the variable documentation on you screen terminal by typing `oct-help -p VariableName`. If you want to search for a variable, but can only remember more or less what the name was, you can type `oct-help -s ApproximateName`. You will get a list of variables whose name is similar to `ApproximateName`.

The next input variable, `Dimensions`, tells the code that we will be working in one spatial direction only. For this first example we will do a simple one-dimensional model of the Hydrogen atom.

The next input variable, `BoxShape`, describes the shape of the simulation box. In 1D this is obviously irrelevant (a "sphere" in 1D is just an interval), but you may check the other options in the variable description in the web page. The variable `Spacing` defines the grid spacing: in `octopus`, the wave functions, densities, potentials, etc,, are represented in a real space "mesh" or

"grid", generally regular. `Spacing = 0.5` means that the distance between grid points is 0.5 atomic units (with few exceptions, all units in the input file are given in atomic units). It remains to specify the size of the simulation box, in this case, the `Radius` of the sphere. In this one-dimensional case, `Radius = 100` means that we will represent the system in the (-100,+100) interval.

`TheoryLevel` informs the code that we want to do an independent electrons calculation. This makes sense since we are doing one single electron. By default, the code works in density-functional mode.

Next, the block `%Species` is used to list the type of atoms (or, in general, sources of potential, hence the generic name `Species`) present in the calculation. Note, however, that "normal" atoms do not have to be defined, and therefore in some input files you will not find a `%Species` block – the code then uses default pseudopotentials. Each line of this block is the description of one atom or model potential. In this case, we only have one that we call `Hydrogen1D` (the first column of the row is the name that will identify that atom). units). The second column is the "type" of potential source that we are defining. You can see a description of the various possibilities in the web page; for this case we use the `species_user_defined` option, which informs `octopus` that we will be employing a mathematical formula to define the potential. The next columns are grouped by pairs: "`potential_formula`" tells the code that the next column contains the formula definining the potential, whereas "`valence`" tells the code that the next column contains the number of valence electrons for this atom (1).

We are therefore using a nucleus-electron potential in the form:

$$V(x) = \frac{-1}{\sqrt{1+x^2}}\,. \tag{1}$$

This type of potential, which is extensively used for 1D studies, is called *soft-Coulomb* potential.

Finally, the last block, `%Coordinates`, places the various atoms in their positions. We only want one Hydrogen atom, placed at the center of the simulation cell.

Now you can run the code, e.g.:

```
prompt> octopus
```

The code emits quite a lot of information to standard output, it is useful to store that information also in a file, e.g.:

```
prompt> octopus > out
```

Or, if you want to also see it while it is created,

```
prompt> octopus | tee out
```

The other source of information for what the code did is in the directory `static`, which in this case only contains the file `info`.

- This model calculation is in fact taken from Ref. [11]. Take a look at that paper; there you can check that the ground state energy that you obtained is indeed coincident with the correct number (provided in Table I of the paper).

- Suggestion: The two key numerical parameters in a real-space grid calculation such as the one you just did are `Spacing` and `Radius`. The correct manner to determine if they are correct is performing a convergence analysis: performing series of calculations at varying values of those parameters, and plotting the result (typically, the total energy), as a function of them.

- You will see that file `inp.2` does not differ much from `inp.1`: it corresponds to a `CalculationMode = unocc` supposed to be performed after the previous `gs` one. In this mode, the code computes "unoccupied", or "excited" states of the system. In this case, `ExtraStates = 9` excited states. The energies of these are also listed in Table I of Ref. [11]. The energies generated by octopus will be listed in a file called `static/eigenvalues`.

3 A molecule irradiated with a laser pulse.

Now we will simulate the irradiation of a molecule with a laser field. Since we cannot do heavy calculations in little time, let us use a simple one, the Na_2^+ molecule, which only has one valence electron. `octopus` uses pseudopotentials (more about this in Refs. [4, 5]), and therefore we will only be dealing with that one valence electron (all other inner electrons are supposed to be "frozen" and do not participate in the chemistry).

First, we need to get the ground state of the molecule, which can be done by making use of file `inp.3`:

```
##################################################
# Ground state of the Na2+ molecule
CalculationMode = gs
FromScratch = yes

BoxShape = sphere
Spacing = 0.7
Radius = 20.0
```

```
TheoryLevel = independent_particles

ExcessCharge = 1

%Coordinates
"Na" | -3.282843 | 0 | 0
"Na" |  3.282843 | 0 | 0
%

EigenSolver = cg
EigenSolverMaxIter = 250
EigenSolverTolerance = 1.0e-6
ConvRelDens = 1.0e-6
################################################
```

Note the presence of the `ExcessCharge` variable. The code computes the number of electron that it needs according to the type of atoms. In this case, since Na only has one valence electron, and we have two Na atoms, in principle the code would use two electrons. However, if we want to have a charged system, we need to specify it by setting this variable. The `EigenSolver...` and `ConvRelDens` variables refer to the degree of numerical convergence that should be achieved in the diagonalization of the Hamiltonian for these ground state calculations. You can learn about them in the variables description of the web page, or in the tutorials.

- One should perform a convergence analysis to be sure that the `Spacing` and `Radius` are correct. If your are too lazy to to this by hand, you can fiind a sample bash script to to this job for you on the tutorial page (section Nitrogen atom).
- By changing the bond-length given in the block `Coordinates`, you can check whether or not the system is at the equilibrium geometry. You can vary the bond length in several consecutive calculations, and plot the total energy and forces on the ions as a function of bond-length.

We will need to know which are the excitation energies of the system, and therefore we will need once again to perform a calculation in the `CalculationMode = unocc` mode with the `inp.4` file:

```
################################################
# Calculation of excited states of Na2+ molecule.
CalculationMode = unocc
ExtraStates = 4
FromScratch = yes
```

```
BoxShape = sphere
Spacing = 0.7
Radius = 20.0

TheoryLevel = independent_particles

ExcessCharge = 1

%Coordinates
"Na" | -3.282843 | 0 | 0
"Na" |  3.282843 | 0 | 0
%

EigenSolver = cg
EigenSolverMaxIter = 250
EigenSolverTolerance = 1.0e-6
ConvRelDens = 1.0e-6
##################################################
```

Now we will perform time-dependent simulations. Take a look at file `inp.5`:

```
##################################################
# Time-dependent simulation of Na2+
CalculationMode = td
FromScratch = yes

BoxShape = sphere
Spacing = 0.7
Radius = 20.0

TheoryLevel = independent_particles

ExcessCharge = 1

%Coordinates
"Na" | -3.282843 | 0 | 0
"Na" |  3.282843 | 0 | 0
%

TDEnergyUpdateIter = 1

TDPropagator = exp_mid
TDExponentialMethod = lanczos
TDExpOrder = 20

AbsorbingBoundaries = mask
AbWidth = 4.0
```

```
MoveIons = no

omega = 0.1
electric_amplitude = 0.05
tau0 = 2*(2*pi)/omega
t0 = 2*(2*pi)/omega
totaltime = t0+tau0

TDMaxSteps = 250
TDTimeStep = totaltime/TDMaxSteps

%TDExternalFields
electric_field   | i | 0 | 0 | omega | "envelope_function"
%

%TDFunctions
"envelope_function" | tdf_cosinoidal | electric_amplitude | tau0 | t0
%

TDOutput = multipoles + laser + energy
################################################
```

Here are the novelties:

- The equations are propagated by *discretizing* the time interval that is going to be simulated in smaller pieces or time steps. `TDTimeStep` determines the size of this time step. If this value is too large, the propagations will be unstable. But the larger it is, the faster we will simulate the system.

- The `TDPropagator` variable sets which propagation algorithm is used. You may read about this topic in Ref. [12], and also consult the various options in the variable description of the web page. The exponential midpoint rule, which is the option chosen in the input file, is given by:

$$\varphi(t + \Delta t) = \exp\{-i\Delta t \hat{H}(t + \Delta t/2)\}\varphi(t)\,. \tag{2}$$

- Even though the propagation scheme is now fixed, it requires of an algorithm to compute the action of the exponential of a matrix on a wave function, which is a non-trivial task when the dimension is large. The algorithm to do this is chosen by the `TDExponentialMethod` and `TDExpOrder` variables.

- When the electrons are propagated in time with an intense laser field, some ionization may occur. In order to account for this, one has to add

absorbing boundaries to the simulation box. This is done, in this case, by applying a "mask function" (`AbsorbingBoundaries = mask`) that at each time step cancels a part of the wave function that is close to the simulation box boundary.

- `MoveIons = no` means that we will perform the calculation with the nuclei fixed to their original position. We will relax this condition in the next section.

- The external field (that simulates the electric field created by a laser pulse) is described through the blocks `TDExternalFields` and `TDFunctions`. These are reasonably well described in the code web page, so we will not repeat the explanations here. Note only that the applied electric field has the form:

$$\mathbf{E}(t) = \mathrm{Re}[f(t)e^{i\omega t}\mathbf{p}]\,, \tag{3}$$

  where $f(t)$ is the envelope funciton, ω is the carrier frequency (`omega` in the `inp` file), and $\mathbf{p}$ is the polarization vector, which can be complex.

Now it is time to analyze the results. Take a look at the variable `TDOutput`: it orders the code to create some output files, with information about the multipoles of the system (monopole – i.e. the total electronic charge present in the simulation box – and the dipole), the laser field, or the energy as the system evolves in time. You will find the files in the directory `td.general`.

You should try to plot these files with some plotting program (`gnuplot` should be installed in your computers). For example, the second column of the file `multipoles` contains the time at each time step, whereas the third column contains the electronic charge in the simulation box. One of the most relevant information to learn from this simulation is the ionization yield, as a function of time.

You can also plot the laser field that you have used in this simulation: column number two of the file `laser` is the time, whereas column number three is the x component of the electric field.

- The total ionization is strongly dependent on the peak electric amplitude of the laser pulse. You can analyze this effect by performing various runs at varying values of this value.

- The polarization of the laser pulse is also relevant: how does this aspect affect the total ionization for this system? The effect of this parameter will be specially visible in the evolution of the dipole of the system. You can check this by looking at the `multipoles` file.

- Are we well converged with respect to the simulation box? This is especially difficult in cases where one studies ionization (in fact, it is impossible to obtain perfect convergence).

4 Photo-dissociation – or not.

In the previous calculation, the nuclei were frozen at their equilibrium position. In reality, they should move. To simulate this, we will utilize the Ehrenfest model. The relevant file is now `inp.6`. We will not explain this file in detail, since by now you have learnt most that it is to be known about it. The goals of this section are the following:

- The `inp.6` files contains the specification of a *non-resonant* calculation: the frequency of the laser field is not tuned to any of the excitation energies of the system (transition from the ground state to any of the excited states). What are the effects of this pulse on the system? Specifically, does it lead to the ionization of the system or not?
- Tune the laser frequency to the first resonance of the system, and check now whether or not the system dissociates. Is this dissociation accompanied of simultaneous ionization of the system?
- What happens if you keep the resonant frequency, but change the polarization direction of the laser field?

5 High harmonic generation

The goal of this exercise is to compute the high harmonic generation (HHG) spectrum emitted by a 1D model of Hydrogen when irradiated with a laser pulse. The goal is to reproduce, approximately, Fig. 1 (it is also one of the calculations present in Ref. [15]). Since you are now an expert in the use of octopus, we will not give you input files, but only some instructions.

Therefore, some hints:

- You need to build a model in 1D: use the variable “Dimensions”.
- The model is the usual soft-Coulomb Hydrogen. You need to define it such a potential with the “Species” block.
- You must use absorbing boundaries when doing the td runs.

- Be sure to find good values for the grid spacing, the simulation box, and later for the time-dependent discretization. A converged result can be obtained with a calculation of a few minutes, but only if those parameters are chosen wisely.

  Remember that the smaller the grid spacing you use, the smaller the td time step you will need [Question: why?]

- Define the laser through the "TDExternalFields" and "TDFunctions" blocks.

  The laser must have a cosinoidal envelope, and 200 oscillations of a 800 nm frequency. The amplitude is 0.03 a.u.

- Once the td run is done, you must compute the HHG spectrum; for that purpose, you should use the "oct-harmonic-spectrum" utility. This program essentially computes:

$$H(\omega) = \left| \int_0^T \mathrm{d}t \; \frac{\mathrm{d}^2}{\mathrm{d}t^2} d(t) \right|^2 , \tag{4}$$

  where $d(t)$ is the dipole moment. It can be done in two different manners:

  1. Reading the td dipole moment from the "multipoles" file, and taking the second time derivative numerically from it. This is the default mode, you just need to launch "oct-harmonic-spectrum" and it will do this. It will generate a file called "hs-mult.x" with the spectrum.
  2. If you run the td calculations with octopus adding the "dipole_acceleration" keyword to the "TDOuput" variable (for example, "TDOutput = laser + multipoles + dipole_acceleration", octopus will generate an "acceleration" file in the "td.general" directory, with the acceleration of the dipole as given en Eq. (2) in Ref. [15]. Then you can run "oct-harmonic-spectrum" with the "-m 2" option, and it will compute Eq. (1) using that, and generating a "hs-acc.x" file with the spectrum. Mathematically, this is equivalent to the former method; numerically they may differ.

- There are some variables related to the "oct-propagation-spectrum" that you may set, mainly "PropagationSpectrumMaxEnergy", and "PropagationSpectrumEnergyStep", of easy meaning: the spectrum will be plotted up to the maximum energy given by the former, and at intervals of energy given by the latter. It is good, numerically, to set the

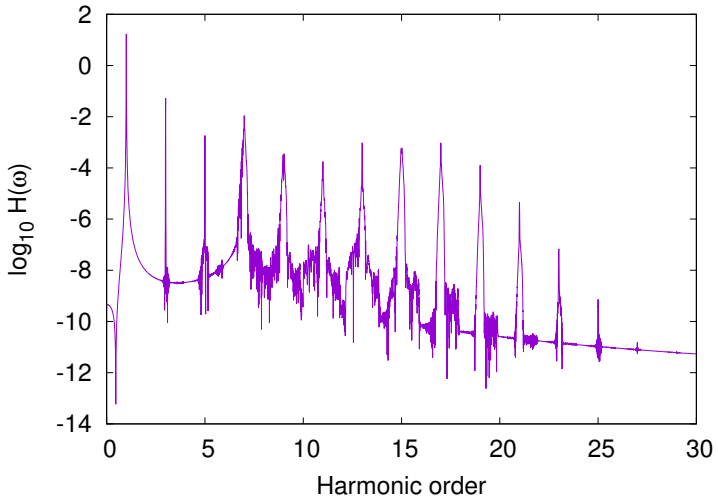

Figure 1: Harmonic spectrum emitted by the model Hydrogen molecule.

former to some multiple of the laser carrier frequency, and to set the latter at the Fourier transform base frequency $2\pi/T$, where T is the total propagation time.

"But this is not TDDFT because there is only one electron!" Yes, you are right. But once you learnt to do this, it is straightforward to add one or more electrons, and get TDDFT HHG spectra.

References

[1] M. A. L. Marques and E. K. U. Gross, Annu. Rev. Phys. Chem. **55**, 427 (2004).

[2] E. Runge and E. K. U. Gross, Phys. Rev. Lett. **52**, 997 (1984).

[3] "Time Dependent Density Functional Theory", edited by M. A. L. Marques, C. A. Ullrich, F. Nogueira, A. Rubio, K. Burke, and E. K. U. Gross (Springer Verlag, Berlin, 2006).

[4] A. Castro, H. Appel, M. Oliveira, C. A. Rozzi, X. Andrade, F. Lorenzen, E. K. U. Gross, M. A. L. Marques and A. Rubio, *Phys. Stat. Solidi B* **243**, 2465 (2006).

[5] M. A. L. Marques, A. Castro, G. F. Bertsch, and A. Rubio, Comp. Phys. Comm. **151**, 60 (2003).

[6] U. Saalmann, and R. Schmidt, Z. Phys. D **38**, 153 (1996).

[7] E. K. U. Gross, J. F. Dobson, and M. Petersilka, in "Density Functional Theory" (Topics in Current Chemistry **181**), edited by R. F. Nalewajski, p. 81 (Springer-Verlag, Berlin-Heidelberg, 1996).

[8] T. Kunert, and R. Schmidt, Phys. Rev. Lett. **86**, 5258 (2001).

[9] C. Brif, R. Chakrabarti, and H. Rabitz, New J. Phys. **12**, 075008 (2010).

[10] A. Castro, M. A. L. Marques, J. A. Alonso, G. F. Bertsch, and Angel Rubio, Eur. Phys. J. D **28**, 211 (2004).

[11] J. Javanainen, J. H. Eberly, and Q. Su, Phys. Rev. A **38**, 3430 (1988).

[12] A. Castro, M. A. L. Marques, and A. Rubio, J. Chem. Phys. **121**, 3425 (2004).

[13] Umberto De Giovannini, Daniele Varsano, M A L Marques, H Appel, E K U Gross, and A Rubio. *Ab initio* angle- and energy-resolved photoelectron spectroscopy with time-dependent density-functional theory. *Phys. Rev. A*, 85:062515, June 2012.

[14] Xavier Andrade, David A Strubbe, Umberto De Giovannini, Ask Hjorth Larsen, Micael J T Oliveira, Joseba Alberdi-Rodriguez, Alejandro Varas, Iris Theophilou, Nicole Helbig, Matthieu Verstraete, Lorenzo Stella, Fernando Nogueira, Alán Aspuru-Guzik, Alberto Castro, Miguel A L Marques, and Angel Rubio. Real-space grids and the Octopus code as tools for the development of new simulation approaches for electronic systems. January 2015.

[15] A. Castro, A. Rubio, and E. K. U. Gross, Eur. Phys. J B **88**, 191 (2015).