

Practical sessions IV-VI, VIII, and XII

Quantum Dots I-V

LECTURERS: ALBERTO CASTRO AND MIGUEL A. L. MARQUES
Guided Construction of a 2D TDDFT code

1 Introduction

In the following, we are going to build a code that implements time-dependent density functional theory (TDDFT). For that purpose, we will build (i) a code that performs ground state DFT, i.e. a code that calculates the ground-state of a many-electron system subject to an external potential; (ii) a code that implements the time-dependent Kohn-Sham or Runge-Gross equations, i.e. that propagates in real time the Kohn-Sham orbitals subject to the time-dependent Kohn-Sham Hamiltonian; and (iii) a code that implements the linear-response formulation of TDDFT in the frequency domain.

There is obviously not enough time in this School to build these three elements from scratch. Hence, we will do a “guided construction”. The code is already written, and you get a copy of it, *except for some pieces which you will have to fill in*. This will speed up the process, but you will still have to dig into the code, and in this way you will learn the manner in which the underlying TDDFT is transformed into a working algorithm. This document is the roadmad for the construction process.

Between these two thick lines you will find side-information, suggested optional exercises, lengthier descriptions of the algorithms forming the code, comments on alternative possibilities, etc.

2 Two-dimensional problems and quantum dots.

The scope of the code is not general: it is a two-dimensional code. The reason to limit the code in this form is that in this way the computations can be done in very short time (typically seconds or minutes). In principle, however, the extension to the 3D problem is straightforward. The algorithms presented are essentially the same in the 3D world. In practice, the code that you will build is specially simple and lacks some of the features that a fully-fledged code has (e.g. non-local pseudopotentials – which are covered elsewhere in this course –, etc). This is necessary due to the time limitation. However, the final result will be certainly state-of-the-art, and able to reproduce in very short time calculations that are nowadays presented in top quality research publications. Even though you may feel disappointed by the 2D nature of the program, and think that after all it is only of academic interest, this is far from true.

Quantum dots (QDs) are artificial nano-scale devices; essentially they may be viewed as confined electron crowds. Due to their smallness, they exhibit quantum-mechanical atom-like behavior (e.g. shell structure). To some extent, we can consider quantum dots as the basic components of nanoelectronics [1]. Quantum dots are fabricated by confining metal or semiconductor conduction-band electrons in a localized region. There are several ways to achieve this localization; one of them is by making use of semiconductor interfaces. In this case, the movement of the electrons is not possible in the perpendicular direction to the interface and the thickness of the interface region is very small. The resulting structure is known as the two-dimensional electron gas (2DEG). Laterally, the electrons also have to be confined applying some kind of potential, which is typically modeled in some simple way.

Not surprisingly, DFT has been successfully applied to describe numerous examples of 2DEG QDs [2]. And, also not surprisingly, TDDFT has also played a role to describe properties related to excited-states of 2DEG QDs [4]. The program that we will work with could be a useful tool for this kind of investigations – very active nowadays –, and not only a classroom exercise.

Some important features, however, will not be incorporated: to name a few, we will assume spin-unpolarized calculations, and will not consider the possible presence of a magnetic field, or the extension to current density-functional theory (CDFT) – rather relevant in this field. Regarding numerics, you may find the design of the program, or the choice of the algorithm choice sub-optimal. We invite you to add any feature, or to improve the code in any manner, as a final exercise or “project”, if time permits.

It is common practice to use the effective mass approximation to describe the electrons in semiconductors or metals. This number in principle depends on the kinetic energy of the electron, but if it turns out to be approximately a constant, the problem is greatly simplified. We may then work with an *effective* Hamiltonian:

$$\hat{H} = \sum_{i=1}^N \frac{\mathbf{p}_i^2}{2m^*} + \sum_{i=1}^N \hat{v}^{\text{ext}}(\hat{\mathbf{r}}_i) + \sum_{i<j}^N \frac{e^2}{4\pi\epsilon |\hat{\mathbf{r}}_i - \hat{\mathbf{r}}_j|}, \quad (1)$$

For the GaAs semiconductor (very common material in QDs experiments), the effective mass m^* is 0.067 times the mass of a free electron.

To ease the numerical work, it is convenient to choose the appropriate units system. In atomic, molecular and solid-state Physics, this is usually the so-called atomic units system. If we take the effective mass approximation, it is convenient to redefine this system of units: We set the effective mass $m^* = 1$, the dielectric constant of the medium $\epsilon = 1$, Planck’s constant $\hbar = 1$ and the absolute charge of the electron $e = 1$. In the CGS-unit system, we then get the effective mass atomic units.

The unit of length is then then effective bohr $a_0^* = (\epsilon/m^*)a_0$ (a_0 is the Bohr radius); the unit of energy is the effective Hartree $\text{Ha}^* = (m^*/\epsilon^2)\text{Ha}$, and the unit of time is the effective atomic time, $u_t^* = (m_e/\hbar)\text{a.u.}$. It is assumed in the code that this effective system of units is used. (In a typical GaAs lattice, $\epsilon = 12.4\epsilon_0$).

In the following, we will assume this system of units (and will omit the symbols with asterisks, unless necessary).

3 Kohn’s theorem, and Generalized Kohn’s theorem.

The original Kohn’s theorem [5] considers an electron gas in the presence of a uniform magnetic field. It states that, regardless of the form of the electron-electron interaction, the only possible excitation frequency of the system is the cyclotron frequency, $\omega_c = eB/mc$. A very similar result may be found [6] for an electron gas in a parabolic shape quantum well: it can only absorb radiation at the bare harmonic oscillator frequency ω_0 , independently of the electron-electron interaction, and of the number of electrons in the well.

It is easy to prove the Generalized Kohn’s theorem. Assuming a two-dimensional problem, such as the one we are interested in, we depart from a Hamiltonian in the form:

$$\hat{H} = \sum_{i=1}^N \frac{\hat{p}_{i,x}^2}{2m} + \sum_{i=1}^N \frac{\hat{p}_{i,y}^2}{2m} + \sum_{i=1}^N \frac{1}{2}m\omega_0^2(\hat{x}_i^2 + \hat{y}_i^2) + \sum_{i<j}^N \hat{u}(\hat{\mathbf{r}}_i - \hat{\mathbf{r}}_j), \quad (2)$$

where the interaction \hat{u} is of arbitrary shape. By defining the operators $\hat{c}^\pm = \sum_{i=1}^N (m\omega_0 \hat{x}_i \mp i\hat{p}_{i,x})$, prove that:

- (i) for any eigenstate Φ_n , $\hat{c}^\pm \Phi_n$ is also an eigenstate, $\Phi_{n\pm 1}$, whose energy differs $\pm \hbar\omega$ from E_n ;
- (ii) the dipole operator $\sum_{i=1}^N \hat{x}_i$ only couples Φ_n to its neighbors $\Phi_{n\pm 1}$.

This result is exact, and it is not obvious that any approximation to the many-body problem, such as for example TDLDA, respects it. We will try to ascertain whether this is the case or not (answer: yes, it is). For that purpose, we will obtain the absorption spectrum of a parabolic quantum dot both assuming the normal Coulomb interaction, and also assuming a Yukawa form for the electron-electron interaction:

$$\hat{u}(\hat{\mathbf{r}}_i - \hat{\mathbf{r}}_j) = \frac{e^{-\gamma r}}{r}, \quad (r = |\hat{\mathbf{r}}_i - \hat{\mathbf{r}}_j|). \quad (3)$$

We will check that both absorption spectra are identical, and contain only one absorption peak at precisely the harmonic well frequency, as prescribed by the theorem. We will also see how this is not the case if we use a different external potential, i.e. a quartic potential well.

The Yukawa potential, Eq. (3), may be regarded as one ‘‘screened’’ Coulomb potential. It certainly does not describe the interaction between electrons in free space. In fact, it is used to describe elementary particles whose interaction is mediated by massive particles – not as the Coulomb interaction, mediated by massless photons.

However, the use of the Yukawa interaction is not limited to the elementary particles world. Screened potentials are widespread in many areas of Physics and Chemistry, since they are simple models to approximate many-body interactions [8]. For example, they may approximate the effects of the screening between charges due to the presence of a background hot plasma. In consequence, a DFT formulation for Yukawa-interacting is not an completely unrealistic exercise.

4 Brief outline of the code

In the directory QD, you will find all the files necessary for this exercise. You will have to edit some of the files to complete the code; typing `make` after each change should re-build all the programs [The needed rules for recompilation are described in the `Makefile` file; hopefully you will not need to change it, unless you want to add or remove files from the package].

The compilation produces seven programs:

1. `gs`: It performs the calculation of the ground-state electronic structure of the 2D model.
2. `td`: It propagates the time-dependent Kohn-Sham equation’s in real time.
3. `excitations`: It calculates the excitation energies in the frequency domain.
4. `coeff`: Auxiliary program that obtains the coefficients of a one-dimensional finite difference scheme.
5. `test_laplacian`: Auxiliary program that tests the calculation of the Laplacian operator on a function defined in a real space mesh.
6. `test_hartree`: Auxiliary program that tests the integration of the Hartree term.

7. sf: Program that calculates the sine Fourier transform of the dipole movement of the system, which should be placed in a `dipole` file after having run `td`.

In the following, we will partly explain the files that produce these little codes. The files have “holes”, that you will have to fill in. Also, note that the code is purposely simple-minded to increase clarity; you may think of ideas to improve on the algorithms for performance (or just elegance) reasons. The “holes” are marked by the delimiters:

```
!!!!!! MISSING CODE X
...
!!!!!! END OF MISSING CODE
```

The number `X` is an identifier number. Possible solutions for the missing parts are offered in file `missing.f90`.

5 The mesh.

- We will represent the wavefunction, densities, etc. in a mesh, rather than using a basis set. The first thing to do is to setup this real space discretization of the functions, for which purpose you have to take a look at the Fortran 90 module `mesh` in file `mesh.f90`. The comments explain the purpose of the module. You will also see your first task: to write the laplacian subroutine. For that purpose you will need an auxiliary program: `coeff`, compiled from the source in `coeff.f90`
- Visit the `coeff.f90` file. It contains the source of the `coeff` little code. This code outputs the coefficients necessary to build a real-space discretization of the second order derivative (or any other derivative).

We need the coefficients $\{c(k)\}_{k=-N}^{+N}$ to build up an expression in the form:

$$\frac{\partial^2 f}{\partial x^2}(x_0) = c_0 f(x_0) + \sum_{k=1}^N c_k f(x_k) + \sum_{k=-1}^{-N} c_k f(x_k). \quad (4)$$

This expression provides an approximation for the second derivative of a function f at a mesh point x_0 , in terms of the values of f at the neighbor points (and itself) $x_k = kh, k = -N, \dots, N$. You may then build the Laplacian simply by doing $\nabla^2 = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$. The program is already prepared for a 9-point formula ($N = 4$) of the second derivative. You just need to run the code, and use the coefficients to build up the Laplacian:

```
whatever-prompt-you-have> ./coeff
c(0)      -0.1785714E-02

c( 1: n) =   0.1600000E+01  -0.2000000E+00   0.2539683E-01  -0.1785714E-02

c(-1:-n) =   0.1600000E+01  -0.2000000E+00   0.2539683E-01  -0.1785714E-02
```

For the curious, it is maybe worthed a little explanation on what we have just done.

The most common approach to the electronic structure problem (either with DFT or with any other method) is the expansion of the wavefunctions (and related functions) in terms of a set

of basis functions. This approach has two important properties, which may be easily derived from the variational principle: (i) The approximate ground-state energy obtained with a given basis set is always an upper bound to the exact value; any supplement to the basis set will yield a lower energy; (ii) The energy displays a quadratic convergence with increasing basis set size. Despite these two nice features, basis set expansion is not the only approach to the electronic structure problem. An alternative are the “real-space” methods, which rely on the representation of functions directly on a real-space grid, either regular (as the one we are using) or adapted to the problem at hand.

In a real-space implementation, the functions are represented in a real space grid, i.e., we know their values on a selected set of sampling points, $\{x_j\}_{j=1}^M$, which typically are arranged in a regular mesh (in the following, we will assume a one-dimensional problem; the extension to two or three dimensional problems will be done later):

$$f \equiv \{f(x_j)\}_{j=1,\dots,M}. \quad (5)$$

We want to calculate its n -th derivative in a finite difference scheme.

Let us call $x_0 = 0$, and let us assume that we want to get the n -th derivative of f in $x_0 = 0$, $f^{(n)}(x = 0)$. As input information, we will use the values of f at N points to the right, and N points to the left, besides $f(x_0)$: $\{f(x_k)\}_{k=-N}^N$. The objective is to obtain a linear expression of the form:

$$f^{(n)}(x_0) = c_0 f(x_0) + \sum_{k=1}^N c_k f(x_k) + \sum_{k=-1}^{-N} c_k f(x_k). \quad (6)$$

The problem is then to obtain the set of coefficients c_k . For that purpose, we consider the set of polynomials:

$$g_l(x) = x^l, \quad l = 0, \dots, 2N. \quad (7)$$

Their n -th derivatives are:

$$g_l^{(n)}(x) = \begin{cases} l(l-1)\dots(l-n+1)x^{l-n} & , \quad n < l \\ n! & , \quad n = l \\ 0 & , \quad n > l \end{cases} \quad (8)$$

In $x = x_0 = 0$:

$$g_l^{(n)}(x_0) = \delta_{nl} n!. \quad (9)$$

We may then join Eq. 6 and 9 to obtain $2N + 1$ equations. To clarify ideas, let us begin by approximating the first derivative, $n = 1$:

$$\begin{aligned} l = 0 & : g_0^{(1)}(x_0) : 0 = c_0 + \sum_{k=1}^N c_k + \sum_{k=-1}^{-N} c_k \\ l = 1 & : g_1^{(1)}(x_0) : 1 = 0 + \sum_{k=1}^N c_k x_k + \sum_{k=-1}^{-N} c_k x_k \\ l = 2 & : g_2^{(1)}(x_0) : 0 = 0 + \sum_{k=1}^N c_k x_k^2 + \sum_{k=-1}^{-N} c_k x_k^2 \\ \dots & : \dots : \dots = \dots \\ l = 2N & : g_{2N}^{(1)}(x_0) : 0 = 0 + \sum_{k=1}^N c_k x_k^{2N} + \sum_{k=-1}^{-N} c_k x_k^{2N} \end{aligned} \quad (10)$$

It is useful to setup this linear system in matrix form. We define:

$$\mathbf{x}^T = [x_1, \dots, x_N, x_{-1}, \dots, x_{-N}], \quad (11)$$

$$\mathbf{c}^T = [c_1, \dots, c_N, c_{-1}, \dots, c_{-N}], \quad (12)$$

$$\mathbf{A}(\mathbf{x}) = \begin{bmatrix} x_1 & \dots & x_N & x_{-1} & \dots & x_{-N} \\ x_1^2 & \dots & x_N^2 & x_{-1}^2 & \dots & x_{-N}^2 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_1^{2N} & \dots & x_N^{2N} & x_{-1}^{2N} & \dots & x_{-N}^{2N} \end{bmatrix}, \quad (13)$$

and the n -th unit vector in the $2N$ dimensional space:

$$\mathbf{e}_n^T = [0, \dots, 0, 1, 0, \dots, 0]. \quad (14)$$

The coefficient c_0 will always be $c_0 = -\sum_{k=1}^N c_k - \sum_{k=-1}^{-N} c_k$. The rest of the coefficients may be derived from the resulting system, which, for $n = 1$, is:

$$\mathbf{A}(\mathbf{x})\mathbf{c} = \mathbf{e}_1 \quad (15)$$

It is very easy to generalize this expression for higher derivatives: the n -th derivative coefficients may be obtained through:

$$\mathbf{A}(\mathbf{x})\mathbf{c} = n!\mathbf{e}_n \quad (16)$$

Note that, up to now, we have not enforced a regular mesh; the positions $\{x_k\}$, measured with respect to the “problem” point $x_0 = 0$, are arbitrary. It is clear from the previous formulas, how to build finite differences schemes with irregular meshes: for each point in the mesh, one has to solve the previous linear system built with its neighboring points, and obtain the resulting coefficients (which will be different for each point). In `laplacian` subroutine, however, we have assumed a regular mesh. The neighboring points of a given point $x_0 = 0$ in a regular mesh can be easily described by:

$$x_k = kh, \quad k = -N, \dots, N. \quad (17)$$

We may now illustrate the procedure with the simplest example: approximation to the first derivative with $N = 1$, i.e. only two neighboring points. The matrix equation is:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & h & -h \\ 0 & h^2 & (-h)^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_{-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}. \quad (18)$$

Solving this linear system one obtains the well known formula:¹

$$f'(x_0) = \frac{f(x_1) - f(x_{-1})}{2h}. \quad (19)$$

The program `coeff` is setup to provide the $N = 4$ approximation to the second derivative. However, subroutine `coeff` is more general and can be used to get arbitrary derivatives, out of an arbitrary number of points, distributed non-uniformly around the problem point.

As an exercise, you may try to implement derivatives of various orders, and check how the errors behave with increasing approximation orders.

- You may test the Laplacian that you have built by running the `test_laplacian` program. It defines a Gaussian distribution in the form:

$$n(\mathbf{r}) = \frac{1}{2\pi\alpha} e^{-r^2/\alpha^2}, \quad (20)$$

(which, incidentally, is normalized: $\int d^3r n(\mathbf{r}) = 1$). The program calculates numerically the Laplacian of this function, and compares it to the exact result which may be easily obtained analytically. You may see how the accuracy depends on the ratio between the “hardness” parameter α and the grid spacing, and on the order of discretization.

¹Fornberg and Sloan [7] provided an algorithm to obtain the coefficients without solving the linear system.

6 Setting up the Hamiltonian

- Take a look at the module `states` in file `states.f90`: It holds the number of occupied and unoccupied orbitals that are to be considered. In our simple example, we will always consider spin-unpolarized calculations with doubly occupied KS orbitals. The module also contains the variables that describe the wavefunctions.
- Now you must define the external potential that confines the quantum dot. For that purpose, you must visit the `external_pot` subroutine in the `epot.f90` file. You may play with different potentials; for our first example we will need a harmonic potential in the form:

$$V_{\text{har}}(\mathbf{r}) = \frac{1}{2}\omega_0^2 r^2. \quad (21)$$

For example, to use numbers of the order of the ones in the calculations presented in Ref. [3] (maybe it is worth if you read that paper to get an idea of what we will be doing later), set ω to 0.22 Ha^* . In a following example, we will make use of a quartic potential:

$$V_{\text{quar}}(\mathbf{r}) = \alpha r^4. \quad (22)$$

A reasonable value for α in this case is 0.00008 .

- The following task is providing the code with a procedure to calculate the Hartree potential out of a given density:

$$V_{\text{H}}[n](\mathbf{r}) = \int d^3r' \frac{n(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}. \quad (23)$$

It turns out that this old problem and continues to be, one of the key computational challenges. In one and two-dimensional problems, one may actually use the obvious and slow solution: performing directly the sum on the grid. If $\{\mathbf{r}_i\}$ denote the set of grid points:

$$V_{\text{H}}[n](\mathbf{r}_i) = \sum_j \frac{n(\mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|} \delta v. \quad (24)$$

In this equation, δv denotes the volume (surface, in 2D) surrounding each grid point ($\delta v = \Delta^2$ if Δ is the grid spacing in 2D). In case of using an interaction in the form of the Yukawa potential, the equation must change accordingly:

$$V_{\text{H}}[n](\mathbf{r}_i) = \sum_j e^{-\gamma|\mathbf{r}_i - \mathbf{r}_j|} \frac{n(\mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|} \delta v. \quad (25)$$

Of course, you encounter an infinity problem when $i = j$. The way to circumvent this problem in 2D is:

$$V_{\text{H}}[n](\mathbf{r}_i) = \sum_{j \neq i} \frac{n(\mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|} \delta v + 2\Delta\sqrt{\pi}n(\mathbf{r}_i). \quad (26)$$

which is the algorithm that you may implement. We also invite you the work of thinking *why* the previous equation appropriately approximates the Hartree potential.

[Incidentally, you may also want to prove that in the Yukawa case, the $i = j$ term should be $2\pi n(\mathbf{r}_i) \frac{1 - e^{-\gamma\Delta/\sqrt{\pi}}}{\gamma}$, which reduces to the Coulomb case when $\gamma \rightarrow 0$]

Unfortunately, this easy scheme is slow, and becomes unpractical when the size of the system grows – it is easy to see that it is an $O(N^2)$, algorithm, where N is number of mesh points. In 3D one should not try to use it. An alternative is to perform the integral in Fourier space; by making use of the convolution theorem, it is easy to see that in the plane wave representation, the Coulomb (or Yukawa) interaction is diagonal. In the case of the Yukawa interaction (the Coulomb case is easily obtained by taking $\gamma \rightarrow 0$):

$$\begin{aligned}\tilde{u}_H(\mathbf{G}) &= \frac{2\pi}{\gamma\sqrt{1+\frac{G^2}{\gamma^2}}}, \\ \tilde{V}_H[n](\mathbf{G}) &= \tilde{u}_H(\mathbf{G})\tilde{n}(\mathbf{G}).\end{aligned}\tag{27}$$

However, when applying this technique to the Coulomb interaction (and also to the Yukawa interaction, depending on the magnitude of γ) for finite or aperiodic systems, one encounters one difficulty inherently linked to the plane wave representation: a plane wave representation necessarily implies periodic boundary conditions, and replication of the original charge density in an infinite array. Since the Coulomb interaction is long-ranged, the simple application of the previous equations (27) includes the interactions of the replicas with the original system. This must be avoided. One possible solution is to define a cutoff on the interaction, e.g.:

$$u_H^R(r) = \begin{cases} \frac{1}{r} & , \quad r < R \\ 0 & , \quad r > R \end{cases}\tag{28}$$

One then uses $\tilde{u}_H^R(\mathbf{G})$ in Eqs. (27).

Due to the lack of time, we have purposely shortened the discussion of the Hartree problem in the main text. Numerous authors have addressed the problem; our vanity leads us to cite our own work on the subject [20], which describes the plane wave solution, although in the 3D case (other references maybe found therein).

In the 2D case, and considering – as we have, in this program – a distribution of charge n placed in a square of side L , the procedure begins by placing it in a bigger square of side $(1+\sqrt{2})L$, padding with zeros the extra space. Then one defines an interaction in the form of Eq. (28), with $R = \sqrt{2}L$. It is easy to see that this guarantees that the interaction does not change within the original charge distribution, but at the same time avoids interaction between neighboring cells. Then one needs to get the Fourier transform of the interaction, $\tilde{u}_H^R(\mathbf{G})$ (prove this!):

$$\tilde{u}_H^R(\mathbf{G}) = R \sum_{k=1}^{\infty} J_k(RG)/(RG).\tag{29}$$

In the Yukawa case, however, one needs not to define a cutoff, since the potential is short-ranged by definition. The solution is to define the bigger cell large enough to make the interaction between cells negligible, and then apply Eqs. (27) directly.

Both options, for the Coulomb and for the Yukawa case, are implemented in the `poisson` module.

To practice some programming, you may want to code the simple solution of Eq. (26), in subroutine `poisson_sum` in the `poisson` module (file `poisson.f90`). In this module, you may see that one needs to set through the values of some variables, which interaction to use (Coulomb or Yukawa), what is the value of the Yukawa parameter in case of using it, and which method to use (the direct sum, or the plane waves approach).

To try out the accuracy of the implemented schemes, you may want to take a look at the program `test_hartree` and run it.

- Now it is time to define the exchange and correlation term, which are placed in file `vxc.f90`. The subroutine that you have to use is `vxc_lda`, which provides the exchange and correlation potential and energies in the local density approximation (LDA). The expressions are:

$$E_x[n] = \int d^3r n(\mathbf{r}) \epsilon_x^{\text{HEG}}(n(\mathbf{r})); \quad v_x[n](\mathbf{r}) = \frac{\delta E_x[n]}{\delta n(\mathbf{r})}. \quad (30)$$

$$E_c[n] = \int d^3r n(\mathbf{r}) \epsilon_c^{\text{HEG}}(n(\mathbf{r})); \quad v_c[n](\mathbf{r}) = \frac{\delta E_c[n]}{\delta n(\mathbf{r})}. \quad (31)$$

$\epsilon_x^{\text{HEG}}(n)$ and $\epsilon_c^{\text{HEG}}(n)$ are the exchange and correlation energy per particle, respectively, of the 2D HEG of density n .

The exchange term may be derived analytically (obvious exercise: derive it):

$$\epsilon_x^{\text{HEG}}(n) = -\frac{4\sqrt{2}}{3\sqrt{\pi}}\sqrt{n}. \quad (32)$$

The correlation term, however, is much more involved. One has to resort to numerical results (typically of Quantum Monte-Carlo type), which are later parameterized for easy use in DFT codes. We have chosen the expression parameterized by Attaccalite and coworkers [21], which for the spin-unpolarized case, has the form:

$$\epsilon_c^{\text{HEG}}(n) = a + (br_s + cr_s^2 + dr_s^3) \times \text{Ln} \left(1 + \frac{1}{er_s + fr_s^{3/2} + gr_s^2 + hr_s^3} \right), \quad (33)$$

where r_s is the Wigner-Seitz radius of the 2D HEG ($r_s = 1/\sqrt{\pi n}$).

You may find the generalized subroutines for the spin-polarized case in the octopus distributions.

(i) You may easily derive the exchange term for a homogeneous electron gas of arbitrary polarization $\xi = \frac{n_\uparrow - n_\downarrow}{n_\uparrow + n_\downarrow}$ by making use of the identity (*spin-scaling identity*):

$$E_x[n_\uparrow, n_\downarrow] = \frac{1}{2}E_x[2n_\uparrow] + \frac{1}{2}E_x[2n_\downarrow]. \quad (34)$$

The result is $\epsilon_x^{\text{HEG}}(n, \xi) = \frac{1}{2} [(1 + \xi)^{3/2} + (1 - \xi)^{3/2}] \epsilon_x^{\text{HEG}}(n, 0)$.

(ii) Generalize the given subroutines to allow for spin polarized cases.

- Subroutine `interaction_pot` in file `ipot.f90` has the task of building the terms of the Kohn-Sham potential that arise from the electronic interaction: the Hartree and exchange and correlation terms. It is useful if one writes it in such a way that it is easy to *disconnect* any of the terms at will, as it is done in the suggested solution in file `missing.f90`.
- In file `hpsi.f90`, you have to fill two subroutines: `hpsi` and `zhpsi`. They should apply the Kohn-Sham Hamiltonian on an input wavefunction, respectively real or complex.

7 The SCF cycle, and the ground state program.

- It is now time to build one of the mains: the `gs` program, in charge of obtaining the ground state Kohn-Sham orbitals. Note that the main program, in `gs.f90` file, consists essentially of some initializations, and a call to the `scf` subroutine, explained below and which performs the self-consistent cycle.
- An essential step in each step of the self-consistent procedure is the diagonalization of the current approximation to the Kohn-Sham Hamiltonian (the exact one at the end of the cycle). For this task we have implemented a conjugate-gradients algorithm in `conjugate_gradients` subroutine in `cg.f90` file. We have chosen to implement the simple yet succesful scheme suggested by H. Jiang, Baranger and Yang [9].

The computational research on eigensolvers starts with the works of Jacobi, long time before the existence of computers. Until the 1960s, the state of the art is dominated by the QR algorithm and related schemes, suitable for the full diagonalization of general, albeit small, matrices. The eigenproblem has thereafter proved to be ubiquitous in all disciplines of Science; In Ref. [10] you may find introductions to the topic.

In our case, we are confronted with the algebraic eigenproblem that emerges from the real-space discretization of the Kohn-Sham equations (a similar problem arises when other representations, e.g., plane waves, are used). The Kohn-Sham operator is the sum of a potential term (typically non-local, although not severely non-local) and a partial differential operator. In most DFT electronic-structure method, the solution to this eigenproblem is the most time-consuming part of the calculations. Some key features of this problem are:

- Large size. Typically, the matrix dimension is 10^5 - 10^6 (smaller, in the 2D case). Not even modern supercomputers may store the full matrix in memory; one requires solvers that need only to know how to operate the Hamiltonian on a vector.
- Sparsity. This is the reason that facilitates the solution, despite the enormous dimensions. The non-null elements of the matrix are normally a few rows around the diagonal – its number depending on the order of discretization of the Laplacian operator. Non-local pseudopotentials add more non-diagonal terms.
- Hermiticity. The Hamiltonian of a physical system should be an observable.
- One is interested only on the smallest eigenvalues, i.e. one only needs one small part of the spectrum, not the 10^5 - 10^6 eigenpairs.
- Usually, approximations to the eigenpairs are available. The reason is that the eigenproblem has to be solved at each iteration in the SCF cycle. One can use the solutions obtained in the previous step as initial guesses for the present step.
- Typically, and related to the previous point, the solution algorithms are iterative, i.e. the solutions are obtained by iterative improvement of approximate guesses.

This is a shallow enumeration of typical approaches:

- The implemented eigensolver is a conjugate gradients method, very much in the spirit of the approaches described in the papers cited in Ref. [11]. However, the preconditioning employed in these references, which is based on the fact that the kinetic term is diagonal in a plane wave approach, cannot be used in our real space case.
- Another option is the preconditioned block-Lanczos algorithm [15] implemented by Saad and collaborators, already used for DFT electronic calculations. In this case, the preconditioning is based of high-frequency filtering in real space.
- Another Lanczos-type eigensolver, namely the one implemented in the ARPACK package [12].²

²ARPACK can be found at <http://www.caam.rice.edu/software/ARPACK/>.

- And yet another free implementation of the blocked-Lanczos eigensolver is the TRLAN package.³ Regarding this approach, and the Lanczos approach to the eigenproblem, see Refs. [10, 16, 17].
- Finally, another (related) and commonly invoked algorithm suitable for this type of calculation is the Davidson algorithm [13], and, more recently, the Jacobi-Davidson scheme [14]. The precise implementation that we have tried is the JDQR package [18].

- The `scf` subroutine, in the `scf` file, takes care of closing the self-consistent cycle that solves the Kohn-Sham equations. The basic algorithm is depicted in Fig. 1; you may practice some programming by implementing it in some way in the `scf` subroutine.

Figure 1 suggests that the input density (the density that defines the Kohn-Sham Hamiltonian at each SCF cycle step $\hat{H}_{\text{KS}}[n^{(i)}]$) is the output density of the previous step (the density obtained from the wavefunctions that results of the diagonalization of the Hamiltonian of the previous step). This doesn't work properly, and one has to mix this output density with the densities of previous iterations to guarantee the convergence. The simplest method is the linear mixing:

$$n^{(i+1)} = \alpha n_{\text{output}}^{(i)} + (1 - \alpha)n^{(i)}, \quad (35)$$

for some mixing parameter α . You may implement this simple (yet very safe) scheme; more sophisticated and much more efficient options are given in Ref. [19].

- Now the `gs` program should be finished. Run it at will to test it and check that everything works fine.

If you use the confining potential defined in Eq. (21) and only one occupied orbital (i.e. one quantum dot with only two orbitals) you should get some output similar to:

```
SCF CYCLE ITER #      69
      diff =          1.0075E-07
      1      7.60044118E-01      8.57155697E-06
```

```
SCF CYCLE ENDED

      diff =          8.0597E-08
      Etot =          8.5714E-01
      1      7.60044201E-01      8.57493997E-06
```

(i) You may check how the eigenvalues depend on the strength of the interaction. It will also be interesting to see, depending on the shape of the confining potential, how the many-body excitation energies differ from the differences in eigenvalues.

(ii) Spread along the code, there are some calls to the output subroutine, in the `output.f90` file. These calls plot the functions in the 2D grid. They may be easily plotted with the `gnuplot` command `splot`. For the previous example, you will get e.g. the `rho` file with the density, whose plot is depicted in Fig. 2.

³TRLAN can be found at <http://www.nersc.gov/~kswu/>.

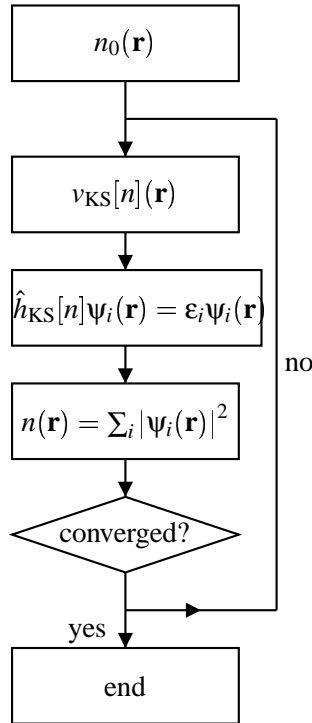


Figure 1: Flow-chart depicting a generic Kohn-Sham calculation

8 Propagators, and the time-dependent program.

- First, take a look at the main of the time-dependent program: `td`. In this case, you just need to read the file, since in fact all that the main does is calling at the end the `propagate` subroutine.
- There is nothing exciting in subroutine `propagate.f90`, as you will see. The key parameters that define the propagation (total time of simulation, and time step) are defined in the `prop_time` and `dt` variables. Note that at each time step one needs to recalculate the Hamiltonian – TDDFT is a problem that involves time-dependent Hamiltonians, no matter if there is an external perturbation or not.

The real work of propagating the wavefunctions is done by the `propagator` subroutine.

- The subroutine that implements the approximator to the quantum mechanical propagator $\hat{U}(t + \Delta t, t)$ is written in the `propagator` subroutine in then `propagator.f90` file. You will be told about propagators in other parts of the School; in this subroutine we have implemented the following approximation (you are welcome to try out other possibilities):

$$\hat{U}(t + \delta t) \approx \exp\left\{-i\frac{\delta t}{2}\hat{H}_{\text{KS}}(t + \delta t)\right\} \exp\left\{-i\frac{\delta t}{2}\hat{H}_{\text{KS}}(t)\right\}, \quad (36)$$

where the (in principle unknown) $\hat{H}_{\text{KS}}(t + \delta t)$ is approximated by considering the density that results of the wavefunctions obtained by the crude estimation:

$$\phi_i = \exp\{-i\delta t\hat{H}_{\text{KS}}(t)\}\phi_i(t). \quad (37)$$

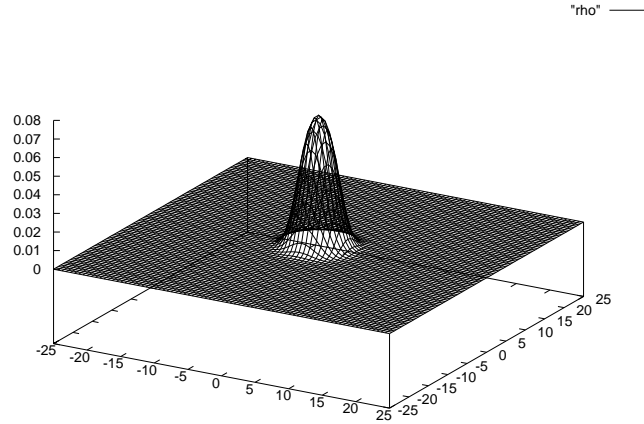


Figure 2: Electronic density of a two-electron harmonic-well quantum dot.

- The previous algorithm requires the computation of the action of the exponential of the Hamiltonian. For this purpose, it calls the `exponential` subroutine, which is the main object of the `expo` module in the `exponential.f90` file.

Elsewhere we will comment on numerical algorithms suited for this particular and important task. The most obvious you can imagine: truncating the Taylor expansion of the exponential to a certain order:

$$\exp\{-i\delta t \hat{H}_{KS}\}\phi = \sum_{i=0}^k \frac{1}{k!} \hat{H}_{KS}^k \phi. \quad (38)$$

This one you have to supply. Then you will see that there are other two options, which in fact implement the same algorithm, the so-called Lanczos-based approximation to the exponential. One of them is a simple-minded implementation of the algorithm, whereas the other makes use of the `expokit` package, a free library that implements the same idea in a more elaborate way. For the purpose of running the code, you may want to try them all and finding out which one of them is faster for each particular problem.

9 Checking the GKT.

We now have working ground-state and time-dependent codes. We can thus make our first TDDFT calculations.

- We will start with a two-electron quantum dot, modeled by a harmonic potential, such as the one defined in Eq. (21). Once that you have obtained its ground-state, you may start the `td` program to get its evolution. For that purpose, however, you have to setup a few things:
 - In `propagate` subroutine, variables `prop_time` and `dt`. Regarding the former, one needs to do a simulation long enough to “see” the frequencies that one is seeking. For the purpose of our tests, a propagation of about 2000 effective-mass atomic units should be enough (notice that we are doing calculations very similar to the ones presented in Ref. [3], where these values are also taken).

- In perturbation subroutine, the shape and magnitude of the initial perturbation. For the purpose of obtaining the optical absorption cross section, one typically uses a perturbation in the form:

$$\phi_i(\mathbf{r}, t = 0) = e^{i\mathbf{k}\cdot\mathbf{r}}\phi_i^{\text{GS}}(\mathbf{r}). \quad (39)$$

One may setup the magnitude of the perturbation k and its direction.

Then, one can run the `td` program. The program sends to standard output the total energy of the system at each time step; since the many-body Hamiltonian is time independent, this magnitude should be conserved. If it is not, you must rethink the time-step, or the characteristics of the propagation algorithm. Also, the file `dipole` is written as the program evolves. It contains three columns: the time, the dipole in direction x , and the dipole in direction y . This is the signal from which one may obtain the excitation energies.

Then you may analyze where the excitations lie by taking the sine Fourier transform of the dipole signal. For that purpose, we have put a very simple program, `sf`, whose source is in the `sf.f90` file. You may learn how it works by reading its comments.

Take a look then at the spectrum. The key questions are: how many spectral peaks do we get? At what energies?

- Now repeat the exercise, but changing to a Yukawa form for the inter-electronic interaction (setting interaction to `YUKAWA` in the `poisson` module. You also have to specify the γ parameter (`gamma` variable). A value of 2.0 a.u.^{-1} is reasonable. Notice that now you have to disconnect (or change, if you feel like doing that work) the exchange and correlation parts of the potential.

Yes, we cannot use the usual expressions for the exchange and correlation potential. The reason is that they are deduced assuming a Coulomb interaction. For the exchange-term, if you are curious, one can derive the exchange energy per particle of a homogeneous fermion gas interacting through Yukawa's potential. The result is (for 2D):

$$\epsilon_x(n) = -\frac{\gamma}{2} \left\{ {}_2F_1\left(-\frac{1}{2}, \frac{1}{2}; 2; \frac{-8\pi n}{\gamma^2}\right) - 1 \right\}, \quad (40)$$

where ${}_2F_1(a, b; c; x)$ is the so-called Gauss hypergeometric function.

(i) Derive the previous equation. (*Hint: (at least this is what I did) Follow the derivation of the Coulomb exchange energy per particle for a HEG in Ref. [22], considering 2D instead of 3D, and Yukawa interaction instead of Coulomb interaction.*)⁴

(ii) Does the previous equation reduce to the Coulomb expression for $\gamma = 0$? (*Hint: use Mathematica.*)

(iii) Implement the previous equation in the code, in order to get a local density approximation for the exchange of a Yukawa-electrons gas. (*Hint: The hypergeometric functions are defined in the GSL library. We already have interfaces to GSL functions in the file `gslwrappers.c`, so you just have to add the appropriate one.*)

I have not even tried to think about the correlation term...

- The question is now: how does the new spectrum compare with the one in which the Coulomb interaction is used?

⁴Please let me know if you obtain a different result, since I did not check previous equation with any other source...

- Now, if you still feel like working, repeat previous tests but using the quartic potential of Eq. (22). You should see that (i) there is no longer one single peak in the response (note that there may be large differences in the strengths of the different peaks) and (ii) the response obtained when using different forms for the inter-electronic interaction is no longer the same.

10 Linear-Response TDDFT

Finally, we have arranged an almost-complete code that performs TDDFT calculations within the linear response formalism [23]. This is not the place to derive the equations that are actually solved; let us just present them very quickly. Let us assume that we have obtained the set of occupied states $\{\phi_i\}$ (in the following, i and k run over occupied states) and a set of unoccupied states $\{\phi_j\}$ (in the following j and l run over unoccupied states). You may obtain these states with the `gs` program, by setting at will the `N_occ` and `N_empty` variables.

In the linear response formalism, the excitation energies may be obtained by solving the following eigenvalue equations (the excitation energies are the square roots of the eigenvalues):

$$QF_I = \Omega_I^2 F_I. \quad (41)$$

The matrix Q is m -dimensional, where m is the number of occupied-unoccupied KS orbital pairs. It is defined to be:

$$Q_{ij,kl} = \delta_{ik}\delta_{jl}\omega_{kl}^2 + 2\sqrt{\lambda_{ij}\omega_{ij}K_{ij,kl}}\sqrt{\lambda_{kl}\omega_{kl}}. \quad (42)$$

In this equation, $\omega_{ij} = \epsilon_j - \epsilon_i$, the difference of the corresponding eigenvalues. $\lambda_{ij} = f_i - f_j$ is the difference in occupation numbers of the orbitals. The key magnitude is the coupling matrix K :

$$K_{ij,kl} = 2\langle\phi_i\phi_j|\frac{1}{|\mathbf{r}-\mathbf{r}'|}|\phi_k\phi_l\rangle + 2\langle\phi_i\phi_j|\frac{\delta v_{xc}[n](\mathbf{r}')}{\delta n(\mathbf{r})}|\phi_k\phi_l\rangle. \quad (43)$$

Very importantly, in the LDA, the second term may be simplified:

$$\frac{\delta v_{xc}[n](\mathbf{r}')}{\delta n(\mathbf{r})} = \delta(\mathbf{r}-\mathbf{r}')\frac{dv_{xc}}{dn}[n(\mathbf{r})]. \quad (44)$$

Your task is now to read the `excitations` program in `excitations.f90` file, and see how the previous equations are implemented, adding whatever pieces may be missing. Then you can run any of the models of quantum-dots that you wish, and see how these results compare with the previous approaches.

Typically, the previous calculations of the excitation energies is complemented with the calculation of their strengths. By reading any of the classical references of linear response within TDDFT [23], you may locate the appropriate expressions and implement them (all the necessary quantities are already calculated previously).

References

- [1] See, for example, R. C. Ashori, *Nature* **379**, 413 (1996); M. A. Kastner, *Phys. Today* **46**, 24 (1993); P. L. McEuen, *Science* **278**, 1729 (1997).

- [2] Some (rather random) examples: E. Räsänen *et al*, Phys. Rev. B **67**, 235307 (2003); H. Jiang, H. Baranger and W. Yang, Phys. Rev. Lett. **90**, 026806 (2003); M. Koskinen, M. Manninen and S. M. Reinmann, Phys. Rev. Lett. **79**, 1389 (1997); M. Pi *et al*, Phys. Rev. B **57**, 14783 (1998).
- [3] A. Puente and Ll. Serra, Phys. Rev. Lett. **83**, 3266 (1999).
- [4] More (also random) examples: Ll. Serra *et al*, Phys. Rev. B **59**, 15290 (1999); E. Lipparini *et al*, Phys. Rev. B **60**, 8734 (1999).
- [5] W. Kohn, Phys. Rev. **15**, 1242 (1961).
- [6] L. Brey, N. F. Johnson and B. I. Halperin, Phys. Rev. B **40**, 10647 (1989).
- [7] B. Fornberg and D. M. Sloan, in *Acta Numerica 1994*, edited by A. Iserles (Cambridge University Press, 1994), pp. 203-267.
- [8] Some examples: L. Bertini and M. Mella, Phys. Rev. A **69**, 042504 (2004); J. M. Ugalde, C. Sarasola and X. López, Phys. Rev. A **56**, 1642 (1997).
- [9] H. Jiang, H. U. Baranger and W. Yang, Phys. Rev. B **68**, 165337 (2003).
- [10] Y. Saad, *Numerical methods for large eigenvalue problems*, Manchester University Press (Manchester, 1992); Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst (Eds.), SIAM (Philadelphia, 2000).
- [11] M. P. Teter, M. C. Payne and D. C. Allan, Phys. Rev. B **40**, 12255 (1989); M. C. Payne, M. P. Teter, D. C. Allan, T. A. Arias and J. D. Joannopoulos, Rev. Mod. Phys. **64** 1045 (1992).
- [12] D. S. Sorensen, SIAM J. Matrix Anal. Appl. **13**, 357, 1992.
- [13] E. R. Davidson, J. Comput. Phys. **17**, 87 (1975)
- [14] G. L. G. Sleijpen and H. A. van der Vorst, SIAM Review **42**, 267 (2000).
- [15] Y. Saad, A. Stathopoulos, J. Chelikowsky, K. Wu and S. Ögüt, BIT **36**, 1 (1996).
- [16] A. Stathopoulos, Y. Saad and K. Wu, Technical report of the Minnesota Supercomputer Institute, University of Minnesota, UMSI 96/123 (1996).
- [17] K. Wu and H. Simon, Technical report of the Lawrence Berkeley National Laboratory, 41412 (1998).
- [18] D. R. Fokkema, G. L. G. Sleijpen and H. A. van der Vorst, SIAM J. Sci. Comput. **20**, 94 (1998).
- [19] D. D. Johnson, Phys. Rev. B **38**, 12807 (1988); D. R. Bowler and M. J. Gillan, Chem. Phys. Lett. **325**, 473 (2000).
- [20] A. Castro, M. J. Stott and A. Rubio, Can. J. Phys. **81**, 1151 (2003).
- [21] C. Attaccalite, S. Moroni, P. Gori-Giorgi and G. B. Bachelet, Phys. Rev. Lett. **88**, 256601 (2002).
- [22] J. P. Perdew and S. Kurth, in: C. Fiolhais, F. Nogueira and M. A. L. Marques (Eds.) *A Primer in Density Functional Theory*, Lectures Notes in Physics **620**, (Springer Verlag, Berlin, 2003), Ch. 1.

- [23] E. K. U. Gross, J. F. Dobson, M. Petersilka, in: R. F. Nalewajski (Ed.) *Density Functional Theory* (Springer-Verlag, Berlin, 1996), p.81; M. Petersilka, U. J. Gossmann and E. K. U. Gross, *Phys. Rev. Lett.* **76**, 1212 (1996); M. E. Casida, in: D. P. Chong (Ed.) *Recent Advances in Density-Functional Methods, Part I* (World Scientific, Singapore, 1995), p. 155; M. E. Casida, in: J. M. Seminario (Ed.) *Recent Developments and Applications of Modern Density Functional Theory* (Elsevier, Amsterdam, 1996), p. 391; C. Jamorski, M. E. Casida and D. R. Salahub, *J. Chem. Phys.* **104**, 5134 (1996); G. Onida, L. Reining and A. Rubio, *Rev. Mod. Phys.* **74**, 601 (2002).